SIEMENS

Über dieses Buch . . . 1

RMOS3-Funktionen 2

C-Laufzeitbibliothek 3

RMOS3

Echtzeit-Betriebssystem RMOS3
RMOS3 V3.50
Referenzhandbuch Teil III

Programmierhandbuch

07/2012
A5E03692354-01

Rechtliche Hinweise

Warnhinweiskonzept

Dieses Handbuch enthält Hinweise, die Sie zu Ihrer persönlichen Sicherheit sowie zur Vermeidung von Sachschäden beachten müssen. Die Hinweise zu Ihrer persönlichen Sicherheit sind durch ein Warndreieck hervorgehoben, Hinweise zu alleinigen Sachschäden stehen ohne Warndreieck. Je nach Gefährdungsstufe werden die Warnhinweise in abnehmender Reihenfolge wie folgt dargestellt.

⚠ GEFAHR
bedeutet, dass Tod oder schwere Körperverletzung eintreten **wird**, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

⚠ WARNUNG
bedeutet, dass Tod oder schwere Körperverletzung eintreten **kann**, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

⚠ VORSICHT
bedeutet, dass eine leichte Körperverletzung eintreten kann, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

ACHTUNG
bedeutet, dass Sachschaden eintreten kann, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

Beim Auftreten mehrerer Gefährdungsstufen wird immer der Warnhinweis zur jeweils höchsten Stufe verwendet. Wenn in einem Warnhinweis mit dem Warndreieck vor Personenschäden gewarnt wird, dann kann im selben Warnhinweis zusätzlich eine Warnung vor Sachschäden angefügt sein.

Qualifiziertes Personal

Das zu dieser Dokumentation zugehörige Produkt/System darf nur von für die jeweilige Aufgabenstellung **qualifiziertem Personal** gehandhabt werden unter Beachtung der für die jeweilige Aufgabenstellung zugehörigen Dokumentation, insbesondere der darin enthaltenen Sicherheits- und Warnhinweise. Qualifiziertes Personal ist auf Grund seiner Ausbildung und Erfahrung befähigt, im Umgang mit diesen Produkten/Systemen Risiken zu erkennen und mögliche Gefährdungen zu vermeiden.

Bestimmungsgemäßer Gebrauch von Siemens-Produkten

Beachten Sie Folgendes:

⚠ WARNUNG
Siemens-Produkte dürfen nur für die im Katalog und in der zugehörigen technischen Dokumentation vorgesehenen Einsatzfälle verwendet werden. Falls Fremdprodukte und -komponenten zum Einsatz kommen, müssen diese von Siemens empfohlen bzw. zugelassen sein. Der einwandfreie und sichere Betrieb der Produkte setzt sachgemäßen Transport, sachgemäße Lagerung, Aufstellung, Montage, Installation, Inbetriebnahme, Bedienung und Instandhaltung voraus. Die zulässigen Umgebungsbedingungen müssen eingehalten werden. Hinweise in den zugehörigen Dokumentationen müssen beachtet werden.

Marken

Alle mit dem Schutzrechtsvermerk ® gekennzeichneten Bezeichnungen sind eingetragene Marken der Siemens AG. Die übrigen Bezeichnungen in dieser Schrift können Marken sein, deren Benutzung durch Dritte für deren Zwecke die Rechte der Inhaber verletzen kann.

Haftungsausschluss

Wir haben den Inhalt der Druckschrift auf Übereinstimmung mit der beschriebenen Hard- und Software geprüft. Dennoch können Abweichungen nicht ausgeschlossen werden, so dass wir für die vollständige Übereinstimmung keine Gewähr übernehmen. Die Angaben in dieser Druckschrift werden regelmäßig überprüft, notwendige Korrekturen sind in den nachfolgenden Auflagen enthalten.

Siemens AG
Industry Sector
Postfach 48 48
90026 NÜRNBERG
DEUTSCHLAND

A5E03692354-01
℗ 08/2012 Änderungen vorbehalten

Copyright © Siemens AG 2012.
Alle Rechte vorbehalten

Inhaltsverzeichnis

1 Über dieses Buch . . . 13

1.1 Über dieses Buch . . . 13

1.2 Notationen . . . 14

2 RMOS3-Funktionen . . . 17

2.1 RmActivateTask . . . 17

2.2 RmAlloc . . . 19

2.3 RmBindTask . . . 22

2.4 RmCatalog . . . 23

2.5 RmChangeDescriptor . . . 25

2.6 RmChangeDescriptorAccess . . . 26

2.7 RmClrShIntISHandler . . . 28

2.8 RmCreateBinSemaphore . . . 29

2.9 RmCreateChildTask . . . 31

2.10 RmCreateDescriptor . . . 33

2.11 RmCreateDriver . . . 34

2.12 RmCreateFlagGrp . . . 36

2.13 RmCreateMailbox . . . 37

2.14 RmCreateMemPool . . . 38

2.15 RmCreateMessageQueue . . . 40

2.16 RmCreateOS . . . 41

2.17 RmCreateTask . . . 42

2.18 RmCreateTaskEx . . . 44

2.19 RmCreateUnit . . . 47

2.20 RmDecode . . . 49

2.21 RmDeleteBinSemaphore . . . 50

2.22 RmDeleteDescriptor . . . 51

2.23 RmDeleteFlagGrp . . . 52

2.24 RmDeleteMailbox . . . 53

2.25 RmDeleteMemPool . . . 54

2.26 RmDeleteMessageQueue . . . 55

2.27 RmDeleteTask . . . 56

2.28 RmDisableScheduler . . . 57

2.29 RmEnableScheduler . . . 58

2.30 RmEndTask ... 59
2.31 RmExclude ... 60
2.32 RmFlatToSegAddress ... 61
2.33 RmFree ... 62
2.34 RmFreeAll ... 63
2.35 RmGetAbsTime ... 64
2.36 RmGetBindTaskInfo ... 65
2.37 RmGetBinSemaphore ... 66
2.38 RmGetEntry ... 68
2.39 RmGetFlag ... 70
2.40 RmGetHWClockTime ... 72
2.41 RmGetIntHandler ... 73
2.42 RmGetLinAddress ... 75
2.43 RmGetMemPoolInfo ... 76
2.44 RmGetName ... 77
2.45 RmGetPeriodHPET0 ... 79
2.46 RmGetPhysAddress ... 80
2.47 RmGetSize ... 81
2.48 RmGetSpinLock ... 82
2.49 RmGetSpinLockIRQ ... 84
2.50 RmGetStatusMessageQueue ... 86
2.51 RmGetSysB ... 87
2.52 RmGetSystemPeriod ... 88
2.53 RmGetSystemTime ... 89
2.54 RmGetTaskID ... 90
2.55 RmGetTaskInfo ... 91
2.56 RmGetTaskPriority ... 93
2.57 RmGetTaskState ... 94
2.58 RmGetTimeHPET0 ... 95
2.59 RmHookException ... 96
2.60 RmInitFastTask ... 99
2.61 RmInitFastTick ... 101
2.62 RmInitHPET ... 103
2.63 RmInitShIntClient ... 104
2.64 RmInitSpinLock ... 105

2.65 RmInitOS....106
2.66 RmIO....108
2.67 RmKillTask....110
2.68 RmList....112
2.69 RmMapMemory....114
2.70 RmMemPoolAlloc....115
2.71 RmPauseTask....117
2.72 RmPciSearchFunction....118
2.73 RmPciSearchSubFunction....120
2.74 RmQueueStartTask....122
2.75 RmReadMessage....124
2.76 RmReAlloc....126
2.77 RmReceiveMail....128
2.78 RmReleaseBinSemaphore....130
2.79 RmReleaseSpinLock....131
2.80 RmReleaseSpinLockIRQ....133
2.81 RmRemoveFastTask....135
2.82 RmRemoveFastTick....136
2.83 RmReserveInterrupt....137
2.84 RmResetFlag....138
2.85 RmResetLocalFlag....139
2.86 RmResetOS....140
2.87 RmRestartTask....142
2.88 RmRestoreIntHandler....144
2.89 RmResumeDriver....145
2.90 RmResumeTask....146
2.91 RmRunOS....147
2.92 RmSegToFlatAddress....149
2.93 RmSendMail....150
2.94 RmSendMailCancel....152
2.95 RmSendMailDelayed....153
2.96 RmSendMessage....155
2.97 RmSetDeviceHandler....157
2.98 RmSetFlag....159
2.99 RmSetFlagDelayed....160

2.100 RmSetHWClockTime ... 162
2.101 RmSetIntDefHandler ... 164
2.102 RmSetIntDIHandler ... 166
2.103 RmSetIntISHandler ... 168
2.104 RmSetIntMailboxHandler ... 170
2.105 RmSetIntTaskHandler ... 172
2.106 RmSetLocalFlag ... 174
2.107 RmSetMailboxSize ... 175
2.108 RmSetMessageQueueSize ... 176
2.109 RmSetOS ... 177
2.110 RmSetProfilerPoint ... 179
2.111 RmSetPutcharProc ... 180
2.112 RmSetShIntISHandler1 ... 181
2.113 RmSetShIntISHandler2 ... 183
2.114 RmSetSMRCount ... 186
2.115 RmSetSVC ... 187
2.116 RmSetSystemTime ... 188
2.117 RmSetTaskPriority ... 189
2.118 RmStartTask ... 190
2.119 RmSuspendDriver ... 192
2.120 RmSuspendTask ... 193
2.121 RmUncatalog ... 194
3 C-Laufzeitbibliothek ... 195
3.1 _ctime32 ... 195
3.2 _gmtime32 ... 197
3.3 _localtime32 ... 199
3.4 _time32 ... 201
3.5 abort ... 202
3.6 abs ... 203
3.7 access ... 204
3.8 acos ... 205
3.9 asctime ... 206
3.10 asin ... 208
3.11 assert ... 209
3.12 atan ... 210

3.13 atan2 211
3.14 atexit 212
3.15 atof 213
3.16 atoi 214
3.17 atol 215
3.18 bsearch 216
3.19 calloc 217
3.20 ceil 218
3.21 changevib 219
3.22 chdir 220
3.23 checkpoint 222
3.24 chmod 223
3.25 clearerr 225
3.26 clock 226
3.27 close 227
3.28 cos 228
3.29 cosh 229
3.30 createvib 230
3.31 ctime 231
3.32 difftime 233
3.33 dismount 234
3.34 div 235
3.35 duread 236
3.36 duwrite 238
3.37 errno, errno2 240
3.38 exit 242
3.39 exp 243
3.40 fabs 244
3.41 fclose 245
3.42 fduopen 246
3.43 fdureopen 247
3.44 feof 249
3.45 ferror 250
3.46 fflush 251
3.47 fgetc 252

3.48 fgetpos ... 253
3.49 fgets ... 254
3.50 fileno ... 255
3.51 floor ... 256
3.52 fmod ... 257
3.53 fopen ... 258
3.54 fprintf ... 260
3.55 fputc ... 262
3.56 fputs ... 263
3.57 fread ... 264
3.58 free ... 265
3.59 freopen ... 266
3.60 frexp ... 269
3.61 fscanf ... 270
3.62 fseek ... 272
3.63 fsetpos ... 273
3.64 ftell ... 274
3.65 fwrite ... 275
3.66 getc ... 276
3.67 getchar ... 277
3.68 getcwd ... 278
3.69 getenv ... 279
3.70 gets ... 280
3.71 gettimestamp ... 281
3.72 getvolumestatus ... 282
3.73 getw ... 285
3.74 gmtime ... 286
3.75 inhibitabort ... 288
3.76 isalnum ... 289
3.77 isalpha ... 290
3.78 isascii ... 292
3.79 iscntrl ... 293
3.80 isdigit ... 294
3.81 isgraph ... 295
3.82 islower ... 296

3.83 isprint....297
3.84 ispunct....298
3.85 isspace....299
3.86 isupper....300
3.87 isxdigit....301
3.88 labs....302
3.89 ldexp....303
3.90 ldiv....304
3.91 localtime....305
3.92 log....307
3.93 log10....308
3.94 longjmp....309
3.95 lseek....310
3.96 malloc....312
3.97 matherr....313
3.98 memccpy....315
3.99 memchr....316
3.100 memcmp....317
3.101 memcpy....318
3.102 memmove....319
3.103 memset....320
3.104 mkdir....321
3.105 mktime....322
3.106 modf....323
3.107 mount....324
3.108 open....325
3.109 perror....327
3.110 pow....328
3.111 printf....329
3.112 putc....333
3.113 putchar....334
3.114 putenv....335
3.115 puts....336
3.116 putw....337
3.117 qsort....338

3.118 raise ... 339
3.119 rand ... 341
3.120 read ... 342
3.121 realloc ... 344
3.122 remap ... 345
3.123 remove ... 346
3.124 rename ... 347
3.125 rewind ... 348
3.126 rmdir ... 349
3.127 scanf ... 350
3.128 search ... 353
3.129 searchlong ... 356
3.130 setbuf ... 359
3.131 setjmp ... 360
3.132 setvbuf ... 361
3.133 signal ... 362
3.134 sin ... 367
3.135 sinh ... 368
3.136 sleep ... 369
3.137 snprintf ... 370
3.138 sprintf ... 372
3.139 sqrt ... 374
3.140 srand ... 375
3.141 sscanf ... 376
3.142 strcat ... 378
3.143 strchr ... 379
3.144 strcmp ... 380
3.145 strcpy ... 381
3.146 strcspn ... 382
3.147 strerror ... 383
3.148 strftime ... 384
3.149 strlen ... 386
3.150 strncat ... 387
3.151 strncmp ... 388
3.152 strncpy ... 389

3.153 strpbrk 390
3.154 strrchr 391
3.155 strspn 392
3.156 strstr 393
3.157 strtod 394
3.158 strtok 396
3.159 strtol 397
3.160 strtoul 399
3.161 sys_errlist, sys_nerr 401
3.162 tan 402
3.163 tanh 403
3.164 time 404
3.165 tmpfile 405
3.166 tmpnam 406
3.167 toascii 407
3.168 tolower, _tolower 408
3.169 toupper, _toupper 409
3.170 tzset 410
3.171 ungetc 412
3.172 unlink 413
3.173 vfprintf 414
3.174 vfscanf 416
3.175 vprintf 418
3.176 vscanf 420
3.177 vsnprintf 421
3.178 vsprintf 422
3.179 vsscanf 424
3.180 write 426
3.181 x_cr_gettaskid 428
3.182 x_cr_gettaskparam 429
3.183 x_cr_initenv 432
3.184 x_cr_killtsk 434
3.185 x_cr_setexit 435
3.186 xinitc 436
3.187 xinitt 438

Tabellen

Tabelle 1- 1 Kommandos 14
Tabelle 1- 2 Abkürzungen 14
Tabelle 1- 3 Allgemeine Datentypen 15
Tabelle 1- 4 Datentypen des RMOS3-API 15

Über dieses Buch . . . 1

1.1 Über dieses Buch . . .

Überblick

Dieses Referenzhandbuch unterstützt Sie vor allem beim Programmieren von RMOS3. Es ist das zentrale Nachschlagewerk beim Programmieren, beim Testen und beim Kontrollieren von Quellcode.

Kapitel 1

"Über dieses Buch . . ." (Seite 13) gibt Ihnen einen Leitfaden zur Dokumentation und erläutert die Notationen, die im Handbuch verwendet werden.

Kapitel 2

"RMOS3-Funktionen" (Seite 17) bietet in alphabetischer Reihenfolge eine detaillierte Beschreibung der RMOS3-Funktionen.

Kapitel 3

"C-Laufzeitbibliothek" (Seite 195) bietet in alphabetischer Reihenfolge eine detaillierte Beschreibung der CRUN-Funktionen.

1.2 Notationen

Damit die RMOS3 Dokumentation für Sie leichter lesbar wird, verwenden wir die folgende Notation:

Kommandos

Kommandos steuern Programmabläufe im Dialog- oder Batchbetrieb. Im Text werden Kommandos durch die Schriftart Courier dargestellt. Ein Kommando besteht aus mindestens einem Element. Elemente sind Konstanten oder Variablen. Sie setzen sich aus Buchstaben, Ziffern und Sonderzeichen (z.B. **#*?**.) zusammen.

Auch Programmlistings werden in der Schriftart Courier dargestellt. Sie sind zeichengetreu abgedruckt und befolgen nicht die Notation für Kommandos. So unterscheidet die Sprache C beispielsweise zwischen Groß- und Kleinbuchstaben.

Tabelle 1- 1 Kommandos

Darstellung	Eigenschaft	Erläuterung
GROSS	Konstante	Elemente in Großbuchstaben sind Konstanten. Sie werden zeichengetreu eingegeben, wobei Groß- oder Kleinbuchstaben verwendet werden können.
1847	Konstante	Zahlen sind immer Konstanten.
x	Variable	Elemente in Kleinbuchstaben sind Variablen, die durch aktuelle Elemente ersetzt werden.
()\:;>	Sonderzeichen	Sonderzeichen und Leerzeichen dienen als Begrenzer zwischen Elementen und müssen eingegeben werden.
Die Verwendung von Elementen wird durch Metazeichen beschrieben. Metazeichen werden nicht eingegeben.		
Darstellung	**Eigenschaft**	**Erläuterung**
< >	Abgrenzer	Variablen werden in spitze Klammern eingeschlossen
[ ]	Optional	Elemente in eckigen Klammern sind optional.
{a|b|c} |a|b|c|	Auswahl	Ein Element ist auszuwählen, wenn mehrere Elemente entweder untereinander in geschweiften Klammern oder nebeneinander zwischen senkrechten Strichen stehen.
...	Wiederholung	Ellipsen weisen auf die mögliche Wiederholung des vorherigen Elements hin.

Abkürzungen

Eine Reihe von Abkürzungen und Kurzbezeichnungen gilt für die gesamte RMOS3-Dokumentation:

Tabelle 1- 2 Abkürzungen

Abkürzung	Bedeutung	Textstelle
cfg	konfigurierbar	Systemaufrufe
uintmax	maximum unsigned integer (FFFF FFFFH)	Systemaufrufe
dp	data pointer (sel:off) 48-bit-Pointer	Systemaufrufe

Datentypen

Folgende Datentypen können mit den unter RMOS3 zugelassenen C-Compilern verwendet werden:

Tabelle 1- 3 Allgemeine Datentypen

Datentyp	Datenlänge
char	8 bit
short	16 bit
int	32 bit
long	32 bit
long long [1]	64 bit
void _FAR * [2]	48 bit
void _NEAR * [2]	32 bit
enum	32 bit
float	32 bit
double	64 bit

1 Nur für GNU-Programme verfügbar.

2 Zeiger sind in Flat-Programmen immer 32 Bit. Es wird nicht zwischen NEAR und FAR unterschieden.

Tabelle 1- 4 Datentypen des RMOS3-API

Name	Datentyp	Datenlänge
uchar	unsigned char	8 bit
ushort	unsigned short	16 bit
uint	unsigned int	32 bit
ulong	unsigned long	32 bit
rmfarproc [1]	Zeiger auf Funktion vom Typ void _FAR f(void)	48 bit
rmnearproc [1]	Zeiger auf Funktion vom Typ void _NEAR f(void)	32 bit
rmproc [1]	Zeiger auf Funktion vom Typ void _NEAR f(void)	32 bit

1 Zeiger sind in Flat-Programmen immer 32 Bit. Es wird nicht zwischen NEAR und FAR unterschieden.

RMOS3-Funktionen 2

2.1 RmActivateTask

Funktion

Task aktivieren

Syntax

```
#include <rmapi.h>

int    RmActivateTask(uint TaskID);
```

Parameter

Parametername	Bedeutung
TaskID	Task-ID

Beschreibung

Der Aufruf bringt eine andere Task in den Zustand BEREIT, wenn sie sich vorher im Zustand WARTEND befand.

Bei folgenden Zuständen ist der Aufruf `RmActivateTask` unzulässig und wird mit Fehlermeldung beendet:

- Beenden/Löschen durch `RmKillTask` wurde bereits angemeldet
- Debugger Breakpoint
- bei allen Laufzeitfehlern (z.B. Divisions–Fehler)

Hinweis

Die Warte-Bedingung der Zieltask wird zurückgesetzt (z.B. Zeitbedingung der `RmPauseTask` wird auf 0 gesetzt, oder Wartezeit von `RmGetFlag` wird gesetzt auf `RM_CONTINUE`). Anschliessend wechselt die Zieltask in den Zustand BEREIT.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	SVC erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Eine ungültige TaskID wurde übergeben.
22	RM_INVALID_ TASK_STATE	Aufruf bei jetzigem Zustand der Task unzulässig. (Task ist im Zustand DORMANT, RECHNEND, BEREIT oder WARTEND auf E/A-Beendigung)

Siehe auch

Debugger-Kommando QUERY

2.2 RmAlloc

Funktion

Speicherbereich aus HEAP allokieren

Syntax

```
#include <rmapi.h>

int    RmAlloc (
          ulong TimeOutValue,
          uint Mode,
          ulong Size,
          void **ppMemory);
```

Parameter

Parametername	Bedeutung		
TimeOutValue	Maximale Wartezeit bis zur Ausführung		
	0	RM_CONTINUE	Task fortsetzen, ohne auf Zuteilung des Speichers zu warten.
	-1	RM_WAIT	Auf Zuteilung des Speichers warten.
		0 ... ULONG_MAX	Zeitinterval in ms.
	Zur Zeitangabe können die Werte für Stunden. Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^{31} Millisekunden.		
	RM_HOUR(hour)		Wartet (hour) Stunden
	RM_MINUTE(min)		Wartet (min) Minuten
	RM_SECOND(sec)		Wartet (sec) Sekunden
	RM_MILLISECOND(ms)		Wartet (ms) Millisekunden
	Die Zeitangaben dürfen nur ein Vielfaches des Systemtaktes (Standard 10ms) sein.		
Mode	Allokierungsmethode für den Speicher:		
	RM_AUTOFREE (0)		Der Speicher wird automatisch mit RmFreeAll (Seite 63) freigegeben. Er ist taskspezifisch zugeordnet.
	RM_NOAUTOFREE (2)		Der Speicher wird nicht automatisch mit RmFreeAll (Seite 63) freigegeben.

Parametername	Bedeutung	
Mode (Forts.)	RM_FLATALLOC (0x0C)	Der Zeiger `ppMemory` enthält `FlatSelector:FlatOffset`. `FlatOffset` kann direkt als Pointer an Flat–Programme übergeben werden. Dieser Mode ist nur für segmentierte Programme.
	RM_STACKALLOC (0x2000)	Für Verwendung des Speichers mit `RmCreateTaskEx` als Stack. Speicher wird mit "Guarded Pages" geschützt. Ein Stacküberlauf wird erkannt. Adresse liegt oberhalb von Adresse 0xf0000000. Wenn Adresse mit RmGetPhysAddress (Seite 80) umgewandelt wird, kann auf Privilege Level 3 (PL3) nicht auf die umgerechnete Adresse zugegriffen werden.
	RM_USERALLOC (0x20)	Für Verwendung bei Speicheranforderung einer segmentierten Privilege Level 0 (PL0)-Task, die den Zeiger auf den Speicherbereich an eine PL3-Task schicken will.
	RM_SYSALLOC (0x10)	Für Verwendung bei Speicheranforderung einer segmentierten Privilege Level 3 (PL3)-Task, die den Zeiger auf den Speicherbereich an eine PL0-Task schicken will.
	Die Modes können durch | miteinander verknüpft werden z.B. RM_AUTOFREE | RM_FLATALLOC	
Size	Größe des Speicherbereichs (-1 = größter verfügbarer Block)	
ppMemory	Adresse des Zeigers auf einen Speicherbereich	

Beschreibung

Der Aufruf allokiert einen Speicherbereich der Größe `Size` aus dem HEAP.

Ein auf Privilege Level 0 (PL0) angeforderter Speicher kann bei CADUL-Programmen nur dann auf PL3 verwendet werden, wenn der Aufruf mit der Option "RM_USERALLOC" aufgerufen wurde.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
9	RM_GDT_FULL	Alle Einträge in der GDT sind belegt. Für jeden Speicherbereich wird ein GDT-Eintrag benötigt.
4	RM_GOT_TIMEOUT	In der angegebenen Zeit konnte kein entsprechender Speicherbereich allokiert werden
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.
38	RM_INVALID_SIZE	Size=0 oder Size größer als HEAP
3	RM_OUT_OF_ MEMORY	Kein Speicherbereich in der angegebenen Größe verfügbar

Siehe auch

RmCreateMemPool (Seite 38)

RmDeleteMemPool (Seite 54)

RmFree (Seite 62)

RmGetSize (Seite 81)

RmMemPoolAlloc (Seite 115)

RmReAlloc (Seite 126)

2.3 RmBindTask

Funktion

Task an einen Core binden

Syntax

```
#include <rmapi.h>

int    RmBindTask (
            uint TaskID,
            uint CoreID);
```

Parameter

Parametername	Bedeutung		
TaskID	ID der Task, die an einen Core gebunden werden soll		
CoreID	ID des Cores		
	0	RM_CORE_0	Core 0
	1	RM_CORE_1	Core 1
	2	RM_CORE_2	Core 2
	3	RM_CORE_3	Core 3
	255	RM_CORE_ALL	Task darf auf beliebigem Core ablaufen

Beschreibung

Der Aufruf bindet die Task mit der ID `TaskID` an den Core mit der ID `CoreID`.

Der Aufruf führt sofort zu einem Wechsel der Task auf einen anderen Core, falls der angegebene Core nicht mit dem momentan ausführenden Core übereinstimmt.

Eine Task sollte nur in Ausnahmefällen an einen Core gebunden werden.

Die Voreinstellung für eine Task ist, dass sie auf jedem Core ausgeführt werden kann.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt.

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Die angegebene TaskID ist ungültig.
2	RM_PARAMETER_ERROR	Die angegebene CoreID ist ungültig.

Siehe auch

RmGetBindTaskInfo (Seite 65)

2.4 RmCatalog

Funktion

Ressource in Betriebsmittelverzeichnis eintragen

Syntax

```
#include <rmapi.h>

int    RmCatalog (
          uint Type,
          uint ID,
          ulong IDEx,
          char * pName) ;
```

Parameter

Parametername	Bedeutung
Type	Betriebsmittel–Typ
ID	Betriebsmittel–ID
IDEx	Erweiterte ID
pName	Adresse des Namens, unter dem der Eintrag abgelegt wird. Der String kann nach C– oder PLM–Notation definiert sein.

Beschreibung

Der Aufruf trägt die angegebenen Parameter in den Catalog ein.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt.

Fehlercodes

ID dez	Fehlercode	Bedeutung
3	RM_OUT_OF_ MEMORY	Es ist ein interner Versuch fehlgeschlagen, Speicherplatz aus dem HEAP zu allokieren.
35	RM_INVALID_TYPE	Der angegebene `Type` ist nicht zulässig. 0<= `Type`<=11

36	RM_INVALID_ID	Die angegebene `ID` ist unzulässig. Die möglichen IDs sind abhängig von `Type`:
		`0 RM_CATALOG_TASK`
		`0<=id<=2047`
		`1 RM_CATALOG_DEVICE`
		`0<=id<=255`
		`2 RM_CATALOG_POOL`
		`0<=id<=63`
		`3 RM_CATALOG_SEMAPHORE`
		`0<=id<=4095`
		`4 RM_CATALOG_EVENTFLAG`
		`0<=id<=63`
		`6 RM_CATALOG_LOCALMAILBOX`
		`0<=id<=255`
		`7 RM_CATALOG_MISC`
		`0<=id<=65535`
		`8 ^RM_CATALOG_USER`
		`0<=id<=65535`
		`10 RM_CATALOG_UNIT`
		`0<=id<=255`
		`11 RM_CATALOG_MESSAGE`
		`0<=id<=2047`

37	RM_INVALID_ STRING	Die Länge des Strings ist unzulässig. Entweder ist sie Null oder größer als 15.
42	RM_INVALID_ POINTER	Der Zeiger auf den String ist falsch, bzw. würde eine Schutzverletzung auslösen.
47	RM_IS_ALREADY_ CATALOGED	Der angegebene String ist bereits katalogisiert. Der String muss eindeutig sein. Daher ist es nicht möglich einen String mehrmals zu katalogisieren.
100	RM_CATALOG_ EXCEEDED	Der Catalog ist voll. Maximal sind 4096 Einträge erlaubt.

Siehe auch

RmGetEntry (Seite 68)

RmGetName (Seite 77)

RmList (Seite 112)

RmUncatalog (Seite 194)

2.5 RmChangeDescriptor

Funktion

Deskriptor verändern

Syntax

```
#include <rmapi.h>

int    RmChangeDescriptor(
               selector Sel,
               ulong LinAddress,
               ulong Limit);
```

Parameter

Parametername	Bedeutung
Sel	Gültiger Selektor für Deskriptor
LinAddress	Absolute physikalische Adresse
Limit	Segmentlänge – 1

Beschreibung

`RmChangeDescriptor` trägt in einen durch einen Selektor spezifizierten Deskriptor eine neue Basisadresse und eine neue Segmentlänge ein.

Es können nur Deskriptoren aus der GDT modifiziert werden.

Hinweis

Diese Funktion ist im Speichermodell Flat nicht notwendig.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	SVC erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
21	RM_INVALID_ SELECTOR	Ein ungültiger Deskriptor wurde verwendet.

Siehe auch

RmChangeDescriptorAccess (Seite 26)

RmCreateDescriptor (Seite 33)

RmDeleteDescriptor (Seite 51)

RmFlatToSegAddress (Seite 61)

RmGetLinAddress (Seite 75)

RmGetPhysAddress (Seite 80)

RmSegToFlatAddress (Seite 149)

2.6 RmChangeDescriptorAccess

Funktion

Zugriffsrechte eines Deskriptors ändern

Syntax

```
#include <rmapi.h>

int    RmChangeDescriptorAccess(
               selector Sel,
               uint Access);
```

Parameter

Parametername	Bedeutung		
Sel	Gültiger Selektor		
Access	ACS_ER	9AH	Execute–Read–Berechtigung
	ACS_RW	92H	Read–Write–Berechtigung
	ACS_RWD	96H	Read–Write–Expand–Down–Berechtigung

Beschreibung

`RmChangeDescriptorAccess` trägt in einen durch einen Selektor spezifizierten Deskriptor ein neues Zugriffsberechtigungs–Byte ein. Dieses Byte wird im niederwertigen Byte des Access–Rights–Word abgelegt.

Wechselt durch `RmChangeDescriptorAccess` ein Datensegment der Länge n*4 Kbyte vom Typ EXPAND–UP zum Typ EXPAND–DOWN, oder umgekehrt, werden Basisadresse und Limit des Deskriptors entsprechend korrigiert. Der vorher ausgewählte physikalische Adressbereich bleibt damit unverändert.

Ist die Länge des Datensegments kein ganzzahliges Vielfaches von 4 Kbyte, ist ein Wechsel der Expanddirection nicht zulässig.

Es können nur Deskriptoren aus der GDT modifiziert werden.

Hinweis

Diese Funktion ist im Speichermodell Flat nicht notwendig, kann aber aufgerufen werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0H	RM_OK	SVC erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
21	RM_INVALID_ SELECTOR	Ein ungültiger Deskriptor wurde verwendet.
6	RM_INVALID_ SEGMENTLENGTH	Eine ungültige Segmentlänge wurde angegeben (≠ n * 4 KByte).

Siehe auch

RmChangeDescriptor (Seite 25)

RmCreateDescriptor (Seite 33)

RmDeleteDescriptor (Seite 51)

RmFlatToSegAddress (Seite 61)

RmGetLinAddress (Seite 75)

RmGetPhysAddress (Seite 80)

RmSegToFlatAddress (Seite 149)

2.7 RmClrShIntISHandler

Funktion

Shared Interrupt Handler entfernen (in RMISHCLI.LIB enthalten)

Syntax

```
#include <rmpci.h>

int    RmClrShIntISHandler (
              uint Irqx,
              uint chainID);
```

Parameter

Parametername	Bedeutung
Irqx	Interrupt Request Nummer (IRQ0-15 bzw. IRQ0-23) Sie kann mit dem Funktionsaufruf `RmPciSearchFunction` aus der Struktur PCI_CONFIG_SPACE (Element `IntLine`) ermittelt werden.
chainID	Pointer auf ID für InterruptHandlerEntry

Beschreibung

`RmClrShIntISHandler r`uft eine Funktion des Servers auf, die den entsprechenden Interrupt-Handler-Eintrag aus dem Server löscht, unabhängig vom Mode des Eintrages.

Wird der letzte Eintrag für Interrupt `Irqx` entfernt, wird der Default-Interrupt-Handler von RMOS3 installiert.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
38	RM_INVALID_SIZE	kein Handler zu löschen
41	RM_INVALID_IRQ_ NUMBER	`Irqx` ungültig
36	RM_INVALID_ID	ungültige `chainID`

Beispiel

Siehe RmSetShIntISHandler2 (Seite 183)

Siehe auch

RmInitShIntClient (Seite 104)

RmPciSearchFunction (Seite 118)

RmPciSearchSubFunction (Seite 120)

RmSetShIntISHandler1 (Seite 181)

RmSetShIntISHandler2 (Seite 183)

2.8 RmCreateBinSemaphore

Funktion

Dynamisches Semaphor definieren

Syntax

```
#include <rmapi.h>

int    RmCreateBinSemaphore(
               char *pSemaphoreName,
               uint *pSemaphoreID);
```

Parameter

Parametername	Bedeutung
pSemaphoreName	Zeiger auf einen C–String, der den Namen enthält, mit dem das Semaphor katalogisiert wird. Ist dieser Zeiger = NULL, wird das Semaphor nicht katalogisiert.
pSemaphoreID	Zeiger auf Semaphor–ID

Beschreibung

`RmCreateBinSemaphore` definiert ein dynamisches Semaphor. Die Semaphor–ID wird im angegebenen Speicherbereich zurückgegeben. Die maximale Anzahl dynamischer Semaphore beträgt 1024. Durch Editieren der Datei RMOS.INI kann die Anzahl auf 4096 erhöht werden.

Das Semaphor wird automatisch unter dem angegebenen Namen katalogisiert.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	*pSemaphoreID enthält gültige Semaphor-ID

Fehlercodes

ID dez	Fehlercode	Bedeutung
16	RM_OUT_OF_ SEMAPHORES	Die Anforderung überschreitet die maximale Anzahl von Semaphoren.
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.
100	RM_CATALOG_ EXCEEDED	Der Catalog ist voll (siehe `RmCatalog`).
37	RM_INVALID_ STRING	Die Länge des Strings ist unzulässig. Entweder ist sie Null oder größer als 15.
47	RM_IS_ALREADY_ CATALOGED	Der angegebene String ist bereits katalogisiert. Der String muss eindeutig sein. Daher ist es nicht möglich einen String mehrmals zu katalogisieren.

Siehe auch

RmDeleteBinSemaphore (Seite 50)

RmGetBinSemaphore (Seite 66)

RmReleaseBinSemaphore (Seite 130)

2.9 RmCreateChildTask

Funktion

Task beim Betriebssystem anmelden

Syntax

```
#include <rmapi.h>

int    RmCreateChildTask (
               char * pTaskName,
               ulong TaskStackSize,
               uint Priority,
               rmfarproc TaskEntry,
               uint * pTaskID);
```

Parameter

Parametername	Bedeutung
pTaskName	Zeiger auf einen C–String, der den Namen enthält, mit dem die Task katalogisiert werden soll. Ist dieser Zeiger = NULL wird die TASK nicht katalogisiert. Der C-String darf maximal 15 Zeichen + \0 umfassen.
TaskStackSize	Größe des benötigten Stack in Worten (32 Bit).
Priority	Task–Priorität (0..255)
TaskEntry	Einsprungadresse der Task
pTaskID	Zeiger auf Task–ID

Beschreibung

`RmCreateChildTask` gibt dem Betriebssystem Tasks bekannt. Die Task wird vom Zustand NICHT-EXISTENT in den Zustand DORMANT überführt. Die Task wird automatisch unter dem angegebenen Namen katalogisiert. Wird für `pTaskName` ein Nullzeiger übergeben, so erfolgt keine Katalogisierung

Beim Erzeugen der Child-Task werden von der Parent-Task die Konsole, das aktuelle Arbeitsverzeichnis und das Environment vererbt. Dabei erfolgt ein Zugriff auf das Filesystem falls die Parent-Task ein Arbeitsverzeichnis besitzt.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt, *pTaskID enthält die gültige Task-ID.

Fehlercodes

ID dez	Fehlercode	Bedeutung
3	RM_OUT_OF_ MEMORY	Nicht genügend Speicher zum Anlegen von Stacksegment oder `RmCatalog` hatte nicht genügend Speicher.
23	RM_RESOURCE_ NOT_AVAILABLE	Maximale Anzahl von Tasks ist überschritten.
37	RM_INVALID_ STRING	Die Länge des Strings ist unzulässig. Entweder ist sie Null oder größer als 15.
38	RM_INVALID_SIZE	Die Längenangabe von Stack war 0 oder ≥ 1GB
42	RM_INVALID_ POINTER	Der Zeiger auf den String ist falsch, bzw. würde eine Schutzverletzung auslösen.
47	RM_IS_ALREADY_ CATALOGED	Der angegebene String ist bereits katalogisiert. Der String muss eindeutig sein. Daher ist es nicht möglich einen String mehrmals zu katalogisieren.
60	RM_INVALID_ TASK_ENTRY	Die Einsprungaddresse in die Task ist ungültig.
100	RM_CATALOG_ EXCEEDED	Der Katalog ist voll (siehe `RmCatalog`).

Siehe auch

RmCreateTask (Seite 42)

RmDeleteTask (Seite 56)

xinitt (Seite 438)

2.10 RmCreateDescriptor

Funktion

Deskriptor erzeugen

Syntax

```
#include <rmapi.h>

int    RmCreateDescriptor(
               ulong LinAdress,
               ulong Limit,
               selector *pSel);
```

Parameter

Parametername	Bedeutung
LinAddress	Absolute physikalische Segment–Startadresse
Limit	Segmentlänge – 1
pSel	Zeiger auf einen Deskriptor

Beschreibung

`RmCreateDescriptor` erzeugt einen Deskriptor in der GDT und schreibt den Selektor auf diesen Deskriptor zurück. Der Deskriptor hat das Attribut READ/WRITE. Speicher wird unter der angegebenen Adresse nicht reserviert.

Hinweis

Es bestehen keine Einschränkungen hinsichtlich der Segmentlänge; ab einer Länge von 1 Mbyte wird die Segmentlänge allerdings auf ein Vielfaches von 4 Kbyte erweitert.

Diese Funktion ist im Speichermodell Flat nicht notwendig.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	*pSel enthält Zeiger auf Deskriptor

Fehlercodes

ID dez	Fehlercode	Bedeutung
9	RM_GDT_FULL	In der globalen Deskriptor–Tabelle ist kein Eintrag mehr frei.
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.

Siehe auch

RmChangeDescriptor (Seite 25)
RmChangeDescriptorAccess (Seite 26)
RmDeleteDescriptor (Seite 51)
RmFlatToSegAddress (Seite 61)
RmGetLinAddress (Seite 75)
RmGetPhysAddress (Seite 80)
RmSegToFlatAddress (Seite 149)

2.11 RmCreateDriver

Funktion

Treiber beim Betriebssystem anmelden

Syntax

```
#include <rmapi.h>

int    RmCreateDriver (
               uint Mode,
               char * pDeviceName,
               RmDCDStruct *pDcd,
               uint *pDriverID);
```

Parameter

Parametername	Bedeutung
Mode	Reserviert (auf 0 setzen)
pDeviceName	Zeiger auf einen C–String, der den Namen enthält, mit dem der Treiber katalogisiert werden soll. Ist dieser Zeiger = NULL, wird der Treiber nicht katalogisiert. In diesem Fall sollte sich der Treiber selber katalogisiert haben.

pDcd	Zeiger auf eine Struktur vom Typ RmDCDStruct, die wie folgt definiert ist:
	`typedef struct _RmDCDStruct`
	`{`
	`struct _RmUCDStruct _NEAR * ucd;`
	`uchar units;`
	`uchar shr;`
	`rmproc init;`
	`rmproc svc;`
	`uchar flags;`
	`uchar fmax;`
	`#if RM3`
	`uchar res[4];`
	`#else`
	`uchar res[2];`
	`#endif`
	`}RmDCDStruct;`

pDriverID	Rückgabe der Treiber–ID

Beschreibung

Mit diesem Aufruf werden Treiber dem Betriebssystem bekannt gegeben. Der Treiber wird automatisch unter dem angegebenen Namen katalogisiert.

Die Treiber sind nicht nachladbar. Der Treibercode und der DCD muss im Segment RM3_CODE32 liegen.

Hinweis

Diese Funktion ist nicht im Speichermodell Flat verwendbar.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
3	RM_OUT_OF_ MEMORY	Nicht genügend Speicher zum Anlegen von internen Datenstrukturen oder `RmCatalog` hatte nicht genügend Speicher.
100	RM_CATALOG_ EXCEEDED	Der Catalog ist voll (siehe `RmCatalog`).
37	RM_INVALID_ STRING	Die Länge des Strings ist unzulässig. Entweder ist sie Null oder größer als 15.
47	RM_IS_ALREADY_ CATALOGED	Der angegebene String ist bereits katalogisiert. Der String muss eindeutig sein. Daher ist es nicht möglich einen String mehrmals zu katalogisieren.
66	RM_INVALID_ DCD_SEGMENT	DCD liegt nicht im Nucleus Code Segment RM3_CODE32.
42	RM_INVALID_ POINTER	Der Zeiger auf den String ist falsch, bzw. würde eine Schutzverletzung auslösen.

Siehe auch

RmCreateUnit (Seite 47)

2.12 RmCreateFlagGrp

Funktion

Dynamische Flag–Gruppe definieren

Syntax

```
#include <rmapi.h>

int    RmCreateFlagGrp(
               char *pFlagGrpName,
               uint *pFlagGrpID);
```

Parameter

Parametername	Bedeutung
pFlagGrpName	Zeiger auf einen C–String, der den Namen enthält, mit dem die Flag-Gruppe katalogisiert wird. Ist dieser Zeiger = NULL, wird die Flag-Gruppe nicht katalogisiert.
pFlagGrpID	Zeiger auf ID der Flag-Gruppe

Beschreibung

`RmCreateFlagGrp` definiert eine dynamische globale Flag–Gruppe. Die ID der Flag–Gruppe wird im angegebenen Speicherbereich zurückgegeben.

Die Flag-Gruppe wird automatisch unter dem angegebenen Namen katalogisiert.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	*pFlagGrpID enthält gültige ID der Flag–ID

Fehlercodes

ID dez	Fehlercode	Bedeutung
12	RM_OUT_OF_ FLAGGROUPS	Die Anforderung überschreitet die maximale Anzahl von Ereignisflags.
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.
100	RM_CATALOG_ EXCEEDED	Der Catalog ist voll (siehe `RmCatalog`).
37	RM_INVALID_ STRING	Die Länge des Strings ist unzulässig. Entweder ist sie Null oder größer als 15.
47	RM_IS_ALREADY_ CATALOGED	Der angegebene String ist bereits katalogisiert. Der String muss eindeutig sein. Daher ist es nicht möglich einen String mehrmals zu katalogisieren.

Siehe auch

RmResetFlag (Seite 138)

RmSetFlag (Seite 159)

RmSetFlagDelayed (Seite 160)

RmDeleteFlagGrp (Seite 52)

RmGetFlag (Seite 70)

2.13 RmCreateMailbox

Funktion

Dynamische Mailbox definieren

Syntax

```
#include <rmapi.h>

int    RmCreateMailbox(
               char *pMailboxName,
               uint *pMailboxID);
```

Parameter

Parametername	Bedeutung
pMailboxName	Zeiger auf einen C–String, der den Namen enthält, mit dem die Mailbox katalogisiert wird. Ist dieser Zeiger = NULL, wird die Mailbox nicht katalogisiert.
pMailboxID	Zeiger auf ein Mailbox–ID

Beschreibung

`RmCreateMailbox` definiert eine dynamische Mailbox. Die Mailbox–ID wird im angegebenen Speicherbereich zurückgegeben.

Die Mailbox wird automatisch unter dem angegebenen Namen katalogisiert.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	`*pMailboxID` enthält gültige Mailbox–ID

Fehlercodes

ID dez	Fehlercode	Bedeutung
100	RM_CATALOG_ EXCEEDED	Der Catalog ist voll (siehe `RmCatalog`).
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.
37	RM_INVALID_ STRING	Die Länge des Strings ist unzulässig. Entweder ist sie Null oder größer als 15.
47	RM_IS_ALREADY_ CATALOGED	Der angegebene String ist bereits katalogisiert. Der String muss eindeutig sein. Daher ist es nicht möglich, einen String mehrmals zu katalogisieren.
15	RM_OUT_OF_ MAILBOXES	Die Anforderung überschreitet die maximale Anzahl von Mailboxen.
3	RM_OUT_OF_ MEMORY	Kein Speicherbereich in der angegebenen Größe verfügbar

Siehe auch

RmDeleteMailbox (Seite 53)

2.14 RmCreateMemPool

Funktion

Dynamischen Speicherpool größer 64 Kbyte anlegen

Syntax

```
#include <rmapi.h>

int    RmCreateMemPool(
               char *pPoolName,
               void *pPoolAddress,
               ulong Size,
               uint *pPoolID);
```

Parameter

Parametername	Bedeutung
pPoolName	Zeiger auf einen C–String, der den Namen enthält, mit dem der Speicherpool katalogisiert wird. Ist dieser Zeiger = NULL, wird der Speicherpool nicht katalogisiert.
pPoolAddress	Anfangsadresse des Pools
Size	Länge des Speicherbereichs in Bytes
pPoolID	Zeiger auf Pool–ID

Beschreibung

`RmCreateMemPool` definiert einen Speicherpool, der an einer Paragraphgrenze liegt. `*pPoolID` enthält gültige Speicherpool-ID. Die maximale Anzahl Speicherpools ist 8. Die Mindestgröße eines Speicherbereichs beträgt 16 Bytes.

Der Speicherplatz für einen Speicherpool kann mit `RmAlloc` aus dem Heap allokiert werden. Als Adresse für den Speicherpool wird die zurückgegebene Adresse aus `RmAlloc` verwendet.

Die Speicherpools werden beim Initialisieren auf die nächste durch 16 teilbare Basisadresse gelegt. Die Länge wird auf den nächsten durch 16 teilbaren Wert reduziert. `RmMemPoolAlloc` garantiert, dass der zurückgegebene Zeiger immer eine durch 16 teilbare Basisadresse besitzt und der Offset den Wert 0 hat.

Der Speicherpool wird automatisch unter dem angegebenen Namen katalogisiert. Wird für `pPoolName` ein Nullzeiger übergeben, so erfolgt keine Katalogisierung.

Hinweis

Es ist zu beachten, dass `pool_addr` aus Selektor und Offset besteht und die absolute Adresse nicht durch Addition von Segment (um 4 Bit nach links verschoben) und Offset, sondern durch Addition von Segmentbasisadresse und Offset gebildet wird. Bei `RmCreateMemPool` muss deshalb in `pool_addr` ein bereits existierender Selektor angegeben werden. Dieser kann entweder auf ein bereits existierendes Segment oder auf ein neu mit `RmCreateDescriptor` erzeugtes Segment zeigen. Der dynamisch erzeugte Deskriptor kann gelöscht werden, wenn der Speicherpool gelöscht wird.

Diese Funktion ist nicht im Speichermodell Flat verwendbar.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	`*pool_id_ptr` enthält gültige Speicherpool–ID

Fehlercodes

ID dez	Fehlercode	Bedeutung
39	RM_INVALID_ OFFSET	Der Offset (`pool_addr`) war außerhalb des gültigen Bereichs (<> 0000H).
38	RM_INVALID_SIZE	Eine Größenangabe war ungültig (`size` < 16).
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.
100	RM_CATALOG_ EXCEEDED	Der Catalog ist voll (siehe `RmCatalog`).
37	RM_INVALID_ STRING	Die Länge des Strings ist unzulässig. Entweder ist sie Null oder größer als 15.
47	RM_IS_ALREADY_ CATALOGED	Der angegebene String ist bereits katalogisiert. Der String muss eindeutig sein. Daher ist es nicht möglich, einen String mehrmals zu katalogisieren.
13	RM_OUT_OF_ MEMORYPOOLS	Die Anforderung überschreitet die maximale Anzahl von Speicherpools.

Siehe auch

Speicherpools

RmFree (Seite 62)

RmFreeAll (Seite 63)

RmMemPoolAlloc (Seite 115)

2.15 RmCreateMessageQueue

Funktion

Message–Queue einrichten

Syntax

```
#include <rmapi.h>

int    RmCreateMessageQueue (
               char * pMessageQueueName,
               uint TaskID);
```

Parameter

Parametername	Bedeutung
pMessageQueueName	Zeiger auf einen C–String, der den Namen enthält, mit dem die Message-Queue katalogisiert wird. Ist dieser Zeiger = NULL, wird die Message-Queue nicht katalogisiert.
TaskID	Zieltask–ID

Beschreibung

Der Aufruf richtet eine Message-Queue für die Task `TaskID` ein.

Die Message-Queue wird automatisch unter dem angegebenen Namen katalogisiert.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt.

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Task–ID ungültig
59	RM_QUEUE_EXIST	Die Message–Queue existiert bereits.
100	RM_CATALOG_ EXCEEDED	Der Catalog ist voll (siehe `RmCatalog`).
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.
37	RM_INVALID_ STRING	Die Länge des Strings ist unzulässig. Entweder ist sie Null oder größer als 15.
47	RM_IS_ALREADY_ CATALOGED	Der angegebene String ist bereits katalogisiert. Der String muss eindeutig sein. Daher ist es nicht möglich einen String mehrmals zu katalogisieren.

Siehe auch

RmDeleteMessageQueue (Seite 55)

RmReadMessage (Seite 124)

RmSendMessage (Seite 155)

2.16 RmCreateOS

Funktion

Dynamische an statische Betriebsmittel–IDs anpassen

Syntax

```
#include <rmapi.h>

int    RmCreateOS (
         uint SemaphoreNum,
         uint FlagNum,
         uint MailboxNum);
```

Parameter

Parametername	Bedeutung
SemaphoreNum	Anzahl zu erzeugender statischer Semaphore (0=unverändert)
FlagNum	Anzahl zu erzeugender statischer Flags (0=unverändert)
MailboxNum	Anzahl zu erzeugender statischer Mailboxen (0=unverändert)

Beschreibung

Dieser Aufruf erzeugt dynamische Betriebsmittel mit denselben IDs wie sie bereits in der vorhergehenden Version für statische Betriebsmittel verwendet wurden. Damit können bestehende Applikationen mit statischen Betriebsmitteln sehr einfach portiert werden. Mit diesem Aufruf kann darüber hinaus in einer neuen Applikation sehr einfach eine bestimmte Anzahl von Betriebsmitteln erzeugt werden. (Dies sollte aber im Allgemeinen vermieden werden, da der Aufruf nur zur Sicherstellung der Aufwärtskompatibilität implementiert wird). Die IDs werden von 0 bis (Anzahl der zu erzeugenden Betriebsmittel)-1 vergeben. Der Aufruf kann zu jedem Zeitpunkt aufgerufen werden. Falls bereits erzeugte Betriebsmittel durch `RmCreateOS()` nochmal erzeugt werden, bewirkt dies keine Veränderung der Betriebsmittel. Bereits erzeugte Betriebsmittel bleiben bestehen.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt.

Fehlercodes

ID dez	Fehlercode	Bedeutung
15	RM_OUT_OF_MAILBOXES	Es wurden zu viele Mailboxen angefordert.
12	RM_OUT_OF_ FLAGGROUPS	Es wurden zu viele Flags angefordert.
16	RM_OUT_OF_ SEMAPHORES	Es wurden zu viele Semaphore angefordert.

2.17 RmCreateTask

Funktion

Task beim Betriebssystem anmelden

Syntax

```
#include <rmapi.h>

int    RmCreateTask (
         char * pTaskName,
         ulong TaskStackSize,
         uint Priority,
         rmfarproc TaskEntry,
         uint * pTaskID);
```

Parameter

Parametername	Bedeutung
pTaskName	Zeiger auf einen C–String, der den Namen enthält, mit dem die Task katalogisiert werden soll. Ist dieser Zeiger = NULL wird die TASK nicht katalogisiert.
TaskStackSize	Größe des benötigten Stack in Worten (32 Bit).
Priority	Task–Priorität (0..255)
TaskEntry	Einsprungadresse der Task
pTaskID	Zeiger der Task–ID

Beschreibung

Mit dem Aufruf werden Tasks dem Betriebssystem bekannt gegeben. Aus den gegebenen Parametern wird intern automatisch ein TCD aufgebaut und mit diesem wird die Task beim Betriebsystem angemeldet. Die Task wird vom Zustand NICHT–EXISTENT in den Zustand DORMANT überführt. Die Task wird automatisch unter dem angegebenen Namen katalogisiert.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt.

Fehlercodes

ID dez	Fehlercode	Bedeutung
3	RM_OUT_OF_ MEMORY	Nicht genügend Speicher zum Anlegen von Stacksegment oder catalog hatte nicht genügend Speicher.
23	RM_RESOURCE_ NOT_AVAILABLE	Maximale Anzahl von Tasks ist überschritten.
38	RM_INVALID_SIZE	Die Längenangabe von Stack war 0 oder ≥ 1G
100	RM_CATALOG_ EXCEEDED	Der Catalog ist voll (siehe `RmCatalog`).
37	RM_INVALID_ STRING	Die Länge des Strings ist unzulässig. Entweder ist sie Null oder größer 15.
47	RM_IS_ALREADY_ CATALOGED	Der angegebene String ist bereits katalogisiert. Der String muss eindeutig sein. Daher ist es nicht möglich einen String mehrmals zu katalogisieren.
60	RM_INVALID_ TASK_ENTRY	Die Einsprungaddresse in die Task ist ungültig.
42	RM_INVALID_ POINTER	Der Zeiger auf den String ist falsch, bzw. würde Schutzverletzung auslösen.

Siehe auch

RmCreateChildTask (Seite 31)

RmCreateTaskEx (Seite 44)

RmDeleteTask (Seite 56)

xinitt (Seite 438)

2.18 RmCreateTaskEx

Funktion

Task beim Betriebssystem anmelden

Syntax

```
#include <rmapi.h>

int    RmCreateTaskEx(
               char *pTaskName,
               RmTCDStruct *pTCD,
               uint *pTaskID);
```

Parameter

Parametername	Bedeutung
pTaskName	Zeiger auf einen C–String, der den Namen enthält, mit dem die Task katalogisiert werden soll. Ist dieser Zeiger = NULL wird die TASK nicht katalogisiert.
pTCD	Zeiger auf eine Struktur vom Typ `RmTCDStruct`
pTaskID	Zeiger auf die zurückgegebene Task–ID

Beschreibung

`RmCreateTaskEx` verändert den Zustand einer dynamischen Task von NICHT EXISTENT in DORMANT. Die Struktur vom Typ `RmTCDStruct` muss vorab initialisiert sein. Die Struktur entspricht dem TCD–Block einer Task. Die Struktur wird nach dem Aufruf nicht mehr benötigt.

Die Task wird danach immer über die zurückgegebene Task–ID angesprochen. Die Task wird automatisch unter dem angegebenen Namen katalogisiert.

In den Task-Flags (`TCD.flags`) wird festgelegt, ob die Taskmerkmale mit RM_TFL_CHILD (siehe `RmCreateChildTask`) an die erzeugte Task vererbt werden sollen.

Beim Kreieren einer Task kann der Stack und das Datensegment automatisch angelegt werden. Voraussetzung dazu ist, dass Speicher für Stack und Daten aus einem Pool (HEAP) entnommen wird. Die betreffenden Segmente für Stack und Daten der Task werden beim Löschen der Task automatisch freigegeben.

Ob der Stack und/oder das Datensegment automatisch angelegt werden soll(en), wird in den Task–Flags (`TCD.FLAGS`) definiert (siehe Systemhandbuch Kapitel 2.3.8 "Bedeutung der Tasksteuerdaten (TCD)").

Ist das Flag RM_TFL_STK gesetzt, so ist die Länge des Stacks in Worten (32 Bit) in `TCD.ESP` zu hinterlegen. Ist das Flag RM_TFL_STK nicht gesetzt, ist in `TCD.SS` und `TCD.ESP` die Anfangsadresse des Stacks zu hinterlegen.

Ist das Flag RM_TFL_DS gesetzt, ist die Länge des Datensegments in `TCD.DS` zu hinterlegen. Ist RM_TFL_DS nicht gesetzt, muss `TCD.DS` bei Hochsprachentasks, die mit COMPACT–Modell übersetzt wurden, mit dem Datensegment der zu kreierenden Task vorbelegt werden. Bei Übersetzen mit LARGE–Modell braucht `TCD.DS` nicht vorbelegt zu werden.

Da `TCD.DS` ein 16–Bit–Wert ist, besteht eine Einschränkung auf 64 Kbyte beim Datensegment.

Soll beim Erzeugen einer Task mit `RmCreateTaskEx` eine automatische Stackzuweisung vorgenommen werden (vgl. Konfiguration im Systemhandbuch), muss die gewünschte Stack-Größe durch folgende Anweisung angegeben werden:

Im segmentierten Modell

```
TCD.stck =  buildptr(0,(void near *)<stackgröße>);
```

Beispiel für Stack-Größe 2048:

```
TCD.stck = buildptr(0,(void near *)2048);
```

Im Flat-Modell

```
TCD.stck = (char *) <stackgröße>;
```

Beispiel für Stack-Größe 2048:

```
TCD.stck =  (char *) 2048;
```

Hinweis

Die maximale Anzahl dynamischer Tasks ist abhängig von der Konfigurierung und gleich der Hälfte der Anzahl der konfigurierten SMR (X_TASK_NUM) in SWNUC.INC.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	`*pTaskID` enthält gültige Task-ID

Fehlercodes

ID dez	Fehlercode	Bedeutung
2	RM_PARAMETER_ERROR	Flag RM_TFL_DS darf für Flat-Aufrufe nicht gesetzt werden.
3	RM_OUT_OF_ MEMORY	Nicht genügend Speicher zum Anlegen von Stacksegment oder der Katalog ist voll.
23	RM_RESOURCE_ NOT_AVAILABLE	Maximale Anzahl von Tasks ist überschritten.
38	RM_INVALID_SIZE	Die Längenangabe von Stack war 0.
100	RM_CATALOG_ EXCEEDED	Der Katalog ist voll (siehe `RmCatalog`).
37	RM_INVALID_ STRING	Die Länge des Strings ist unzulässig. Entweder ist sie Null oder größer als 15.
47	RM_IS_ALREADY_ CATALOGED	Der angegebene String ist bereits katalogisiert. Der String muss eindeutig sein. Daher ist es nicht möglich einen String mehrmals zu katalogisieren.
60	RM_INVALID_ TASK_ENTRY	Die Einsprungaddresse in die Task ist ungültig.
42	RM_INVALID_ POINTER	Der Zeiger auf den String ist falsch, bzw. würde eine Schutzverletzung auslösen.

Siehe auch

Systemkonfigurierung

RmCreateChildTask (Seite 31)

RmCreateTask (Seite 42)

RmDeleteTask (Seite 56)

2.19 RmCreateUnit

Funktion

Unit bei dem Betriebssystem anmelden

Syntax

```
#include <rmapi.h>

int RmCreateUnit (
            char * pDeviceName,
            char * pUnitName,
            RmUCDStruct *pUcd,
            uint *pUnitID);
```

Parameter

Parametername	Bedeutung
pDeviceName	Zeiger auf einen C–String, der den Namen enthält, mit dem der Treiber katalogisiert wurde.
pUnitName	Zeiger auf einen C–String, der den Namen enthält, mit dem die Unit katalogisiert wird. Ist dieser Zeiger = NULL, wird die Unit nicht katalogisiert. In diesem Fall sollte der Treiber selbst die Unit katalogisiert haben.

pUcd	Zeiger auf eine Struktur vom Typ `RmUCDStruct`, die wie folgt definiert ist:
	
	`typedef struct _RmUCDStruct`
	`{`
	`uchar pid;`
	`uchar intno;`
	`rmfarproc intadr;`
	`ushort uns;`
	`#if RM3`
	`uchar port[256-10];`
	`#else`
	`uchar port[128-8];`
	`#endif`
	`}RmUCDStruct;`

pUnitID	Rückgabe der Unit–ID

Beschreibung

Mit diesem Aufruf werden die Units für einen Treiber dem Betriebssystem bekannt gegeben. Während der Konfigurationsphase erfolgt die Initialisierung erst mit dem Aufruf `RmRunOS()`. Im laufenden Betrieb wird die Initialisierung sofort durchgeführt.

Im UCD–Eintrag `UCD.INTADR` kann für jeden RIO–Bytetreiber eine Ausgabe Funktion für Systemmeldungen übergeben werden. Dazu muss der UCD–Eintrag `UCD.INTNO` auf 0 gesetzt sein.Sobald mit `RmSetOS()` die Konsole geändert wird, wird ebenfalls die entsprechende Putchar–Funktion installiert.

Beim Erzeugen einer Unit kann in `UCD.PID` Bit 0 bis 5 die Anzahl `N` der UCB angegeben werden, welche hintereinander liegen müssen. Alle UCBs der nachfolgenden *N* Aufrufe `RmCreateUnit` werden hintereinander abgelegt. Dies wird für die Treiber beim Baustein 8530 benötigt. Die Interrupt–Treiberhandler werden mit dem Aufruf `RmSetDeviceHandler` installiert.

Hinweis

Diese Funktion ist nicht im Speichermodell Flat verwendbar.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
3	RM_OUT_OF_ MEMORY	Nicht genügend Speicher zum Anlegen von internen Datenstrukturen oder catalog hatte nicht genügend Speicher.
100	RM_CATALOG_ EXCEEDED	Der Katalog ist voll (siehe `RmCatalog`).
37	RM_INVALID_ STRING	Die Länge des Strings ist unzulässig. Entweder ist sie Null oder größer als 12.
47	RM_IS_ALREADY_ CATALOGED	Der angegebene String ist bereits katalogisiert. Der String muss eindeutig sein. Daher ist es nicht möglich einen String mehrmals zu katalogisieren.
42	RM_INVALID_ POINTER	Der Zeiger auf den String ist falsch, bzw. würde eine Schutzverletzung auslösen.
21	RM_INVALID_ SELECTOR	Es wurde ein ungültiger Selektor verwendet.
48	RM_IS_NOT_ CATALOGED	Der Treiber ist nicht mit `pDeviceName` katalogisiert.

Siehe auch

RmCreateDriver (Seite 34)

2.20 RmDecode

Funktion

RMOS3–Codenummer decodieren

Syntax

```
#include <rmapi.h>

int    RmDecode (
         uint CodeType,
         int Code,
         char *pText) ;
```

Parameter

Parametername	Bedeutung
CodeType	Codetyp
Code	Codenummer
pText	Zeiger auf Puffer für Textrückgabe

Beschreibung

Der Aufruf decodiert die angegebene RMOS3-Codenummer und gibt im angegebenen Puffer einen C–String zurück, der die Meldung im Klartext enthält. RM_MAXDECODELEN gibt die Mindestpuffergröße an. Der Codetyp gibt die Art der Decodierung an:

RM_SVCERROR (1) SVC–Fehlernummer

RM_SVCNAME (2) SVC–Funktionsnummer zu SVC–Name

Kann der Code nicht in einen String umgesetzt werden, wird ein NULL–String zurückgeliefert.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt
-265	RM_ERROR_OUT_ OF_RANGE	Für den gegebenen Code ist kein Text vorgegeben

Fehlercodes

ID dez	Fehlercode	Bedeutung
33	RM_SVC_NOT_ CONFIGURED	SVC `RmDecode()` ist nicht vorhanden
42	RM_INVALID_ POINTER	Der übergebene Zeiger hat nicht auf einen zulässigen Speicherbereich gezeigt.
35	RM_INVALID_TYPE	`CodeType` ist ungültig.

2.21 RmDeleteBinSemaphore

Funktion

Dynamisches Semaphor löschen

Syntax

```
#include <rmapi.h>

int    RmDeleteBinSemaphore(uint SemaphoreID);
```

Parameter

Parametername	Bedeutung
SemaphoreID	Semaphor–ID

Beschreibung

`RmDeleteBinSemaphore` löscht ein mit `RmCreateBinSemaphore` definiertes dynamisches Semaphor. Der Parameter `SemaphoreID` gibt die ID des Semaphors an, das gelöscht wird.

Wurde ein Katalogeintrag erstellt, wird dieser nun wieder gelöscht.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	SVC erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Eine ungültige ID wurde übergeben.
18	RM_RESOURCE_ BUSY	Das Semaphor befindet sich noch im Besitz einer Task.

Siehe auch

RmCreateBinSemaphore (Seite 29)

RmGetBinSemaphore (Seite 66)

RmReleaseBinSemaphore (Seite 130)

2.22 RmDeleteDescriptor

Funktion

Deskriptor löschen

Syntax

```
#include <rmapi.h>

int    RmDeleteDescriptor(selector Sel);
```

Parameter

Parametername	Bedeutung
Sel	Gültiger Selektor für Deskriptor

Beschreibung

`RmDeleteDescriptor` entfernt einen durch einen Selektor spezifizierten Deskriptor aus der globalen Deskriptor–Tabelle.

Hinweis

Es erfolgt keine Überprüfung des zu löschenden Deskriptors, d.h. falls der durch Sel spezifizierte Deskriptor beispielsweise 0 ist, wird der SVC erfolgreich abgeschlossen.

Diese Funktion ist im Speichermodell Flat nicht notwendig.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Selektor aus Deskriptor–Tabelle entfernt

Fehlercodes

ID dez	Fehlercode	Bedeutung
21	RM_INVALID_ SELECTOR	Ein ungültiger Selektor wurde verwendet.

Siehe auch

RmChangeDescriptor (Seite 25)

RmChangeDescriptorAccess (Seite 26)

RmCreateDescriptor (Seite 33)

RmFlatToSegAddress (Seite 61)

RmGetLinAddress (Seite 75)

RmGetPhysAddress (Seite 80)

RmSegToFlatAddress (Seite 149)

2.23 RmDeleteFlagGrp

Funktion

Dynamische Flaggruppe löschen

Syntax

```
#include <rmapi.h>

int    RmDeleteFlagGrp(uint FlagGrpID);
```

Parameter

Parametername	Bedeutung
FlagGrpID	ID der Flag-Gruppe

Beschreibung

`RmDeleteFlagGrp` löscht eine mit `RmCreateFlagGrp` definierte dynamische globale Flag–Gruppe. Der Parameter `FlagGrpID` gibt die ID der Flag-Gruppe an, die gelöscht wird.

Wurde ein Katalogeintrag erstellt, wird dieser nun wieder gelöscht.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	SVC erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Flaggruppe = 0 oder ID ungültig
18	RM_RESOURCE_ BUSY	Es warten noch Tasks auf das Setzen von Flags dieser Flag-Gruppe (SVC `RmGetFlag`) oder ein SVC `RmSetFlagDelayed` ist noch aktiv.

Siehe auch

RmCreateFlagGrp (Seite 36)

RmGetFlag (Seite 70)

2.24 RmDeleteMailbox

Funktion

Dynamische Mailbox löschen

Syntax

```
#include <rmapi.h>

int    RmDeleteMailbox(uint MailboxID);
```

Parameter

Parametername	Bedeutung
MailboxID	Mailbox–ID

Beschreibung

`RmDeleteMailbox` löscht eine mit `RmCreateMailbox` definierte dynamische Mailbox. Der Parameter `MailboxID` gibt die ID der Mailbox an, die gelöscht wird.

Wurde ein Katalogeintrag erstellt, wird dieser nun wieder gelöscht.

Hinweis

Wenn ein Interrupthandler installiert ist, der eine Mail an eine Mailbox schickt, muss vor dem Löschen der Mailbox der Interrupthandler deinstalliert werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	SVC erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Eine ungültige ID wurde übergeben.
18	RM_RESOURCE_ BUSY	Es warten noch Tasks auf Nachrichten in dieser Mailbox, oder in der Mailbox sind noch Nachrichten enthalten oder ein SVC `RmSendMailDelayed` ist noch aktiv.

Siehe auch

RmCreateMailbox (Seite 37)

RmDeleteTask (Seite 56)

2.25 RmDeleteMemPool

Funktion

Dynamischen Speicherpool löschen

Syntax

```
#include <rmapi.h>

int    RmDeleteMemPool(uint PoolID);
```

Parameter

Parametername	Bedeutung
PoolID	Pool–ID

Beschreibung

`RmDeleteMempool` löscht einen mit `RmCreateMemPool` definierten dynamischen Speicherpool. Der Parameter `PoolID` gibt die ID des Speicherpools an, der gelöscht wird.

Wurde ein Katalogeintrag erstellt, wird dieser nun wieder gelöscht.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	SVC erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Pool–ID = 0 (entspricht Heap-ID) oder ungültige ID
18	RM_RESOURCE_ BUSY	Speicherbereiche aus diesem Pool sind noch zugeteilt.

Siehe auch

RmCreateMemPool (Seite 38)

RmMemPoolAlloc (Seite 115)

2.26 RmDeleteMessageQueue

Funktion

Message–Queue entfernen

Syntax

```
#include <rmapi.h>

int    RmDeleteMessageQueue (uint TaskID);
```

Parameter

Parametername	Bedeutung
TaskID	Task–ID

Beschreibung

Der Aufruf entfernt die Message-Queue für die Task `TaskID`.

Wurde ein Katalogeintrag erstellt, wird dieser nun wieder gelöscht.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt.

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Task–ID ungültig
58	RM_QUEUE_NOT_ EXIST	Die Message–Queue existiert nicht.
18	RM_RESOURCE_ BUSY	In der Message–Queue stehen noch Nachrichten, oder die Task mit der ID `TaskID` wartet noch auf Nachrichten.

Siehe auch

RmCreateMessageQueue (Seite 40)

RmReadMessage (Seite 124)

RmSendMessage (Seite 155)

2.27 RmDeleteTask

Funktion

Dynamische Task löschen

Syntax

```
#include <rmapi.h>

int    RmDeleteTask(uint TaskID);
```

Parameter

Parametername	Bedeutung
TaskID	Task–ID (RM_OWN_TASK = eigene Task)

Beschreibung

`RmDeleteTask` löscht die dynamische Task `TaskID`, sofern sich diese im Zustand DORMANT oder RECHNEND befindet. Bei erfolgreichem SVC verliert die Task–ID ihre Bedeutung.

Wurde ein Katalogeintrag erstellt, wird dieser nun wieder gelöscht.

Hinweis

Dieser Aufruf darf auch auf Tasks angewendet werden, die Funktionen der C-Runtime-Bibliothek (CRUN) benutzen.
Wenn ein Interrupthandler installiert ist, der eine Mail an eine Mailbox schickt, muss vor dem Löschen der Mailbox der Interrupthandler deinstalliert werden.

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	SVC erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
20	RM_TASK_NOT_ DORMANT	Es wurde versucht, eine Task zu löschen, die sich nicht im Zustand DORMANT befindet.
36	RM_INVALID_ID	Eine ungültige Task–ID wurde übergeben.

Siehe auch

x_cr_killtsk (Seite 434)

RmCreateTask (Seite 42)

RmDeleteMailbox (Seite 53)

2.28 RmDisableScheduler

Funktion

Scheduler sperren

Syntax

```
#include <rmapi.h>

int    RmDisableScheduler(void);
```

Beschreibung

`RmDisableScheduler` deaktiviert den Scheduling-Mechanismus. Bei deaktiviertem Scheduling-Mechanismus ist nur die Task aktiv, die den SVC auslöste (auch höher priorisierte Tasks erhalten keine CPU-Zuteilung). `RmDisableScheduler` sperrt alle lokalen und globalen Tasks für die Ausführung durch den Scheduler auf allen Cores. Treibertasks (wie z.B. VGA-Task) sind von dieser Sperre ausgenommen.

`RmDisableScheduler` ist nicht schachtelbar, d.h. jeder SVC führt zum Aussetzen des Scheduling.

Bei ausgesetztem Scheduling sind die Aufrufe `RmDeleteTask` und `RmRestartTask` nicht erlaubt. Ferner sind SVCs zu vermeiden, bei denen eine Task u.U. auf die Abarbeitung einer anderen Task warten muss. Darunter fallen:
`RmAlloc, RmGetEntry, RmQueueStartTask, RmReceiveMail, RmIO, RmSendMail, RmStartTask, RmGetFlag, RmGetBinSemaphore`

Ebenso kann bei ausgesetztem Scheduling ein CLI-Job nicht mit <Ctrl>+<C> abgebrochen werden.

Durch ein zu langes Aussetzen des Scheduling kann die Deterministik des Systems leiden. Dies gilt insbesondere bei Verwendung von
`RmRestartTask` und `RmPauseTask`.

Wird in einer Task `RmDisableScheduler` aufgerufen und die gleiche Task löscht sich selbst mit `RmKillTask(RM_OWN_TASK)`, wird der Scheduler wieder freigegeben.

`RmDisableScheduler` wirkt sich nicht auf den Debugger und auf die Verteiler-Task aus. Dadurch kann bei einem Anwenderfehler mit dem Debugger noch getestet werden.

Hinweis

Die Schedulersperre wird im Fall eines Laufwerkfehlers der Task aufgehoben.

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden

Siehe auch

Scheduler-Beschreibung im Benutzerhandbuch

RmEnableScheduler (Seite 58)

2.29 RmEnableScheduler

Funktion

Scheduler freigeben

Syntax

```
#include <rmapi.h>

int    RmEnable Scheduler(void);
```

Beschreibung

`RmEnableScheduler` aktiviert den Scheduling-Mechanismus, der durch `RmDisableScheduler` deaktiviert wurde.

`RmEnableScheduler` ist nicht schachtelbar, d.h. jeder SVC führt zur Wiederaufnahme des Scheduling.

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Siehe auch

Scheduler-Beschreibung im Benutzerhandbuch

RmDisableScheduler (Seite 57)

2.30 RmEndTask

Funktion

Task beenden

Syntax

```
#include <rmapi.h>

void    RmEndTask (void);
```

Beschreibung

`RmEndTask` beendet die Task-Ausführung. Die Task wird in den Zustand DORMANT überführt, wenn keine weiteren Task-Start-Anforderungen anstehen.

Hinweis

Bei Tasks, die Funktionen der C-Runtime-Bibliothek (CRUN) benutzen (z.B. `printf`), führt `RmEndTask` zum Aufruf von `exit (0)`.

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Siehe auch

Unterbrechen und Beenden von Tasks

RmDeleteTask (Seite 56)

RmQueueStartTask (Seite 122)

RmStartTask (Seite 190)

2.31 RmExclude

Funktion

Bereich eines Speicherpools reservieren

Syntax

```
#include <rmapi.h>

int    RmExclude(
         ulong PhysAddress,
         ulong Size,
         void **ppMemory);
```

Parameter

Parametername	Bedeutung
PhysAddress	Anfangsadresse als physikalische Adresse
Size	Zu reservierende Länge
ppMemory	reserviert

Beschreibung

Für den Fall, dass ein Speicherpool Bereiche enthält, die nicht vergeben werden sollten (z.B. Bildspeicher oder Speicher–E/A (MMIO)), können diese Bereiche reserviert werden. Es wird dabei eine physikalische Adresse verwendet.

Der Speicherbereich muss einem einzelnen Speicherpool angehören und darf nicht vergeben sein.

Hinweis

Ein dynamisch erzeugter Pool kann nur gelöscht werden, wenn er keine reservierten oder allokierten Bereiche enthält.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	SVC erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
17	RM_INVALID_ MEMORYBLOCK	`PhysAddress` konnte nicht zugeordnet werden oder Teile des zu reservierenden Speicherbereichs gehören nicht einem einzelnen Speicherpool an.
25	RM_MEMORY_ ALREADY_USED	Teile des zu reservierenden Speicherbereichs sind momentan belegt.
38	RM_INVALID_SIZE	Eine Größenangabe war ungültig.

Siehe auch

RmAlloc (Seite 19)
RmCreateMemPool (Seite 38)
RmDeleteMemPool (Seite 54)
RmFree (Seite 62)
RmFreeAll (Seite 63)

2.32 RmFlatToSegAddress

Funktion

Daten-Zeiger des FLAT-Modells in segmentiertes Modell umwandeln

Syntax

```
#include <rmapi.h>

int    RmFlatToSegAddress (
            ulong FlatAddress,
            ushort *pSel,
            ulong *pOffset);
```

Parameter

Parametername	Bedeutung
FlatAddress	FLAT-Adresse
pSel	Zeiger auf Selektor für ein Datensegment
pOffset	Zeiger auf Offset

Beschreibung

Der Aufruf konvertiert einen Daten-Zeiger vom FLAT-Speichermodell in das segmentierte Speichermodell.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	SVC erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.

Siehe auch

RmSegToFlatAddress (Seite 149)

2.33 RmFree

Funktion

Einen Speicherbereich task–spezifisch freigeben

Syntax

```
#include <rmapi.h>

int    RmFree(void *pMemory);
```

Parameter

Parametername	Bedeutung
pMemory	Zeiger auf den freizugebenden Speicherbereich

Beschreibung

Mit `RmFree` wird ein mit `RmAlloc` oder `RmMemPoolAlloc` von einer Task angeforderter Speicherbereich freigegeben.

Die teilweise Rückgabe eines Speicherbereichs ist nicht möglich.

Hinweis

Es wird nur die Gültigkeit des Selektors der Speicheradresse überprüft.

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	SVC erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
17	RM_INVALID_ MEMORYBLOCK	Speicherbereich war nicht zugeteilt
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.

Siehe auch

RmAlloc (Seite 19)

RmCreateMemPool (Seite 38)

RmDeleteMemPool (Seite 54)

RmFreeAll (Seite 63)

RmMemPoolAlloc (Seite 115)

RmReAlloc (Seite 126)

2.34 RmFreeAll

Funktion

Alle Speicherbereiche task–spezifisch freigeben

Syntax

```
#include <rmapi.h>

int    RmFreeAll(uint TaskID);
```

Parameter

Parametername	Bedeutung
TaskID	ID der Task, deren gesamter angeforderter Speicher freigegeben werden soll (RM_OWN_TASK = eigene Task).

Beschreibung

Mit `RmFreeAll` werden alle mit `RmAlloc` oder `RmMemPoolAlloc` von einer Task angeforderten Speicherbereiche freigegeben.

Die teilweise Rückgabe von Speicherbereichen ist nicht möglich.

Hinweis

Es erfolgt keine Fehlermeldung, wenn die Task keinen Speicher angefordert hatte. Speicher, den die Task mit RM_NOAUTOFREE angefordert hat, wird nicht freigegeben.

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	SVC erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	`TaskID` ungültig

Siehe auch

RmAlloc (Seite 19)

RmCreateMemPool (Seite 38)

RmDeleteMemPool (Seite 54)

RmFree (Seite 62)

RmMemPoolAlloc (Seite 115)

RmReAlloc (Seite 126)

2.35 RmGetAbsTime

Funktion

Datum und absolute Uhrzeit vom System übertragen

Syntax

```
#include <rmapi.h>

int    RmGetAbsTime(RmAbsTimeStruct *pAbsTime);
```

Parameter

Parametername	Bedeutung	
pAbsTime	seg:off	Zeiger auf eine Struktur vom Typ `RmAbsTimeStruct` (2 ulong, in denen die Systemvariable X_ABSTIME steht (niederwertiges Byte = niederwertige Zählerkomponente).

Beschreibung

Mit `RmGetAbsTime` wird die absolute Systemzeit (in Millisekundenauflösung), die seit Systemstart verstrichen ist, in eine Struktur vom Typ `RmAbsTimeStruct` kopiert:

```
typedef struct _RmAbsTimeStruct
        {
                 ulong lotime;
                 ulong hitime;
        } RmAbsTimeStruct;
```

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	SVC erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
42	RM_INVALID_ POINTER	pAbsTime unzulässig

Siehe auch

Beschreibung zu Timer im Benutzerhandbuch

RmSetSystemTime (Seite 188)

2.36 RmGetBindTaskInfo

Funktion

ID des Cores ermitteln, an den eine Task gebunden ist

Syntax

```
#include <rmapi.h>

int    RmGetBindTaskInfo (
         uint TaskID,
         uint *pCoreID);
```

Parameter

Parametername	Bedeutung		
TaskID	ID der Task, von der die CoreID ermittelt werden soll		
	RM_OWN_TASK	Eigene Task	
	RM_GET_ HIGHEST_ CORE_ID	höchste Core-ID des Zielsystems, hat bei nur einem Core den Wert 0, bei 2 Cores den Wert 1	
pCoreID	Zeiger auf ID des Cores		
	0	RM_CORE_0	Core 0
	1	RM_CORE_1	Core 1
	2	RM_CORE_2	Core 2
	3	RM_CORE_3	Core 3
	255	RM_CORE_ALL	Task darf auf beliebigem Core ablaufen

Beschreibung

Der Aufruf ermittelt, an welche *CoreID* die Task mit der ID *TaskID* gebunden ist.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt.

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Die angegebene `TaskID` ist ungültig.
42	RM_INVALID_POINTER	Der Zeiger `pCoreID` ist ungültig.

Siehe auch

RmBindTask (Seite 22)

2.37 RmGetBinSemaphore

Funktion

Semaphor testen und setzen

Syntax

```
#include <rmapi.h>

int    RmGetBinSemaphore(
            ulong TimeOutValue,
            uint SemaphoreID);
```

Parameter

Parametername	Bedeutung		
TimeOutValue	Maximale Wartezeit bis zur Ausführung		
	0	RM_CONTINUE	Task fortsetzen und nicht auf Semaphor warten
	-1	RM_WAIT	Auf Semaphor warten
	0 ... ULONG_MAX		Zeitinterval in ms. Die Task wartet bis sie das Semaphor erhält oder das Timeout abgelaufen ist.
	Zur Zeitangabe können die Werte für Stunden, Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^{31} Millisekunden.		
	RM_HOUR(hour)		Wartet (hour) Stunden
	RM_MINUTE(min)		Wartet (min) Minuten
	RM_SECOND(sec)		Wartet (sec) Sekunden
	RM_MILLISECOND(ms)		Wartet (ms) Millisekunden
	Die Zeitangaben dürfen nur ein Vielfaches des Systemtaktes (Standard 10ms) sein.		
SemaphoreID	Semaphor–ID		

Beschreibung

`RmGetBinSemaphore` testet und setzt ein Semaphor.

Hinweis

Die Belegung und Freigabe von Semaphoren ist nicht taskspezifisch.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	SVC erfolgreich ausgeführt
-262	RM_TASK_ WAITING	Die Task musste auf das Semaphor warten.

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Eine ungültige `SemaphoreID` wurde übergeben.
4	RM_GOT_ TIMEOUT	Der SVC wurde nach der eingestellten Timeout–Zeit abgebrochen.
23	RM_RESOURCE_ NOT_AVAILABLE	Das gewünschte Betriebsmittel ist nicht verfügbar.

Siehe auch

Automatische Prioritätsänderung durch Semaphor–Besitz

RmReleaseBinSemaphore (Seite 130)

2.38 RmGetEntry

Funktion

Eintrag in Catalog suchen

Syntax

```
#include <rmapi.h>

int    RmGetEntry (
            ulong TimeOutValue,
            char *pName,
            RmEntryStruct *pEntry);
```

Parameter

Parametername	Bedeutung		
TimeOutValue	Maximale Wartezeit bis zur Ausführung		
	0	RM_CONTINUE	Task fortsetzen, ohne Katalogisierung des Eintrags abzuwarten.
	-1	RM_WAIT	Katalogisierung des Eintrags abwarten.
	0 ... ULONG_MAX		Zeitintervall in ms. Die Task wartet entweder bis der Eintrag katalogisiert wurde oder das Timeout abgelaufen ist.
	Zur Zeitangabe können die Werte für Stunden, Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^{31} Millisekunden.		
	RM_HOUR(hour)		Wartet (hour) Stunden
	RM_MINUTE(min)		Wartet (min) Minuten
	RM_SECOND(sec)		Wartet (sec) Sekunden
	RM_MILLISECOND(ms)		Wartet (ms) Millisekunden
	Die Zeitangaben dürfen nur ein Vielfaches des Systemtaktes (Standard 10ms) sein.		
pName	Adresse des Namens, der im Catalog gesucht werden soll. Der String kann nach C– oder PLM–Notation definiert sein.		

pEntry	Adresse einer Struktur vom Typ `RmEntryStruct`, die folgendermaßen definiert ist:
	
	`typedef struct _RmEntryStruct`
	`{`
	`uchar slen;`
	`char string[16];`
	`uchar type;`
	`ulong ide;`
	`ushort id;`
	`} RmEntryStruct`		

Beschreibung

`RmGetEntry` sucht im Catalog einen Eintrag.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt
-262	RM_TASK_ WAITING	Der Aufruf musste auf die Katalogisierung des Eintrags warten.

Fehlercodes

ID dez	Fehlercode	Bedeutung
37	RM_INVALID_ STRING	Die Länge des Strings ist unzulässig. Entweder war sie Null oder größer als 15.
48	RM_IS_NOT_ CATALOGED	Der angegebene String ist nicht katalogisiert (nur bei `wait_type` == RM_CONTINUE)
4	RM_GOT_ TIMEOUT	Der angegebene Timeout ist abgelaufen, ohne dass der String katalogisiert wurde.
42	RM_INVALID_ POINTER	Der Zeiger auf den String bzw. auf die Struktur ist falsch, bzw. würde eine Schutzverletzung auslösen..

Siehe auch

RmUncatalog (Seite 194)

2.39 RmGetFlag

Funktion

Ereignisflag testen

Syntax

```
#include <rmapi.h>

int    RmGetFlag(
         ulong TimeOutValue,
         uint Type,
         uint FlagGrpID,
         uint TestMask,
         uint *pFlagMask);
```

Parameter

Parametername	Bedeutung	
TimeOutValue	Maximale Wartezeit bis zur Ausführung	
	RM_CONTINUE (0)	Task fortsetzen, ohne Setzen des Ereignisflags abzuwarten
	RM_WAIT (-1)	Setzen des Ereignisflags abwarten
	0 ... ULONG_MAX	Zeitintervall in ms. Die Task wartet entweder bis das Ereignisflag gesetzt wird oder das Timeout abgelaufen ist
	Zur Zeitangabe können die Werte für Stunden, Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^{31} Millisekunden.	
	RM_HOUR(hour)	Wartet (hour) Stunden
	RM_MINUTE(min)	Wartet (min) Minuten
	RM_SECOND(sec)	Wartet (sec) Sekunden
	RM_MILLISECOND(ms)	Wartet (ms) Millisekunden
	Die Zeitangaben dürfen nur ein Vielfaches des Systemtaktes (Standard 10ms) sein.	
Type	RM_TEST_ALL (0x0)	Testen, ob alle angegebenen Bits gesetzt sind
	RM_TEST_ONE (0x1)	Testen, ob mindestens ein Bit gesetzt ist
FlagGrpID	ID der Flag–Gruppe (0 = Test der lokalen Flaggruppe)	
TestMask	Maske legt fest, welche Bits getestet werden	
pFlagMask	Pointer auf ein uint, in dem die Werte aller Bits der Flaggruppe zurückgegeben werden.	

Beschreibung

`RmGetFlag` testet bei einer Ereignisflaggruppe, ob entweder alle (RM_TEST_ALL) oder mindestens ein angegebenes Bit (RM_TEST_ONE) gesetzt ist. Bei Angabe einer Wartezeit wird entsprechend auf das Setzen der Bits gewartet. Die Bits einer Flaggruppe werden mit `TestMask` geUNDet und in `pFlagMask` zurückgegeben.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	SVC erfolgreich ausgeführt
-262	RM_TASK_ WAITING	Der Aufruf musste auf Setzen des Flag warten.
-258	RM_FLAG_ ALREADY_SET	Das Flag war bereits gesetzt.

Fehlercodes

ID dez	Fehlercode	Bedeutung
57	RM_TEST_NOT_ OK	Testbedingungen nicht erfüllt
36	RM_INVALID_ID	Eine ungültige FlagGrpID wurde übergeben.
4	RM_GOT_ TIMEOUT	Der SVC wurde nach dem eingestellten Timeout abgebrochen.
42	RM_INVALID_ POINTER	Der Zeiger auf `pFlagMask` ist ungültig bzw. würde eine Schutzverletzung auslösen.

Siehe auch

RmResetFlag (Seite 138)

RmResetLocalFlag (Seite 139)

RmSetFlag (Seite 159)

RmSetFlagDelayed (Seite 160)

2.40 RmGetHWClockTime

Funktion

Datum und Hardware–Uhrzeit übertragen

Syntax

```
#include <rmapi.h>

int    RmGetHWClockTime(
           RmAsciiTimeStruct *pTimeInfo);
```

Parameter

Parametername	Bedeutung
pTimeInfo	Zeiger auf eine Struktur vom Typ `RmAsciiTimeStruct`

Beschreibung

Mit `RmGetHWClockTime` wird die globale Zeit (Echtzeituhr) mit Sekundenauflösung in eine Struktur vom Typ `RmAsciiTimeStruct` kopiert:

```
typedef struct _RmAsciiTimeStruct
             {
              char day[2];   /* 01 bis 31                    */
              char delim0;   /* '-'                          */
              char month[3]  /* JAN, FEB, MAR, APR, MAY, JUN,
                                JUL, AUG, SEP, OCT, NOV, DEC */
              char delim1;   /* '-'                          */
              char year[4];  /* 0 bis 9999                   */
              char blank;    /* ' '                          */
              char hour[2];  /* 0 bis 23                     */
              char colon0;   /* ':' */
              char min[2];   /* 0 bis 59                     */
              char colon1;   /* ':'                          */
              char sec[2];   /* 0 bis 59                     */
             }RmAsciiTimeStruct
```

Datum und Uhrzeit sind ASCII–Zeichen und besitzen die in den Kommentaren stehenden Wertebereiche.

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	SVC erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
42	RM_INVALID_ POINTER	`pTimeInfo` unzulässig

Siehe auch

Timer-Beschreibung im Benutzerhandbuch

RmInitOS (Seite 106)

RmSetHWClockTime (Seite 162)

2.41 RmGetIntHandler

Funktion

Interrupthandler auslesen

Syntax

```
#include <rmapi.h>

int    RmGetIntHandler (
           uint IntNum,
           rmfarproc *pHandlerEntry);
```

Parameter

Parametername	Bedeutung
IntNum	Interrupt–Nummer (0-255) IRQx (x=0 bis 63) HW-Interrupt IRQ(n) (n=0 bis 63)HW-Interrupt Auf PC-Hardware sind die HW-Interrupts von 0 bis 15 (im APIC-Modus bis 23).
pHandlerEntry	Einsprungadresse des Interrupthandlers.

Beschreibung

Der Aufruf dient dazu, den aktuellen Interrupthandler aus der IDT auszulesen.

Die Interrupt–Nummer entspricht dem zugehörigen Vektor in der Interrupt–Tabelle Selektor des betreffenden Deskriptors in der IDT.

Soll dieser Interrupthandler zu einem späteren Zeitpunkt wieder installiert werden, so muss dies mit dem Aufruf `RmSetIntDIHandler` erfolgen.

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	SVC erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
65	RM_INVALID_ INTERRUPT_ NUMBER	Ungültige Interrupt–Nummer
41	RM_INVALID_ IRQ_NUMBER	IRQx ungültig, PIC nicht definiert
42	RM_INVALID_ POINTER	Ungültiger Zeiger

Siehe auch

RmReserveInterrupt (Seite 137)
RmSetIntDefHandler (Seite 164)
RmSetIntDIHandler (Seite 166)
RmSetIntISHandler (Seite 168)
RmSetIntMailboxHandler (Seite 170)
RmSetIntTaskHandler (Seite 172)

2.42 RmGetLinAddress

Funktion

Lineare Adresse aus Pointer ermitteln

Syntax

```
#include <rmapi.h>

int    RmGetLinAddress(
           void *Pointer,
           ulong *pLinAddress);
```

Parameter

Parametername	Bedeutung
Pointer	Zeiger, dessen lineare Adresse ermittelt werden soll.
pLinAddress	Zeiger auf eine Variable, in der die lineare Adresse abgelegt wird.

Beschreibung

`RmGetLinAddress` ermittelt die lineare Adresse eines Selector:Offset–Paars (`pLinAddress`). Im Fehlerfall wird in `pLinAddress` der Wert –1 zurückgeschrieben.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	*pLinAddress enthält lineare Adresse

Fehlercodes

ID dez	Fehlercode	Bedeutung
21	RM_INVALID_ SELECTOR	Ein ungültiger Selektor wurde verwendet.
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.

Siehe auch

RmChangeDescriptor (Seite 25)

RmChangeDescriptorAccess (Seite 26)

RmCreateDescriptor (Seite 33)

RmDeleteDescriptor (Seite 51)

2.43 RmGetMemPoolInfo

Funktion

Informationen über Speicherpool abfragen.

Syntax

```
#include <rmapi.h>

int    RmGetMemPoolInfo(
               uint PoolID,
               RmMemPoolInfoStruct *pInfo);
```

Parameter

Parametername	Bedeutung
PoolID	ID des Memory Pools (RM_HEAP für Heap).
pInfo	Zeiger auf Struktur vom Typ `RmMemPoolInfoStruct`.

Beschreibung

Mit dem Aufruf `RmGetMemPoolInfo` wird die Größe des Pools, des freien Speichers und des größten verfügbaren Blocks (`RmAlloc(Size=-1))` ermittelt. Die Information über den mit `PoolID` angegebenen Pool wird in der Struktur `RmMemPoolInfoStruct` abgelegt, auf die der Zeiger `pInfo` zeigt.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Pool ID ungültig
42	RM_INVALID_ POINTER	`pInfo` ist ein ungültiger Zeiger.

Siehe auch

RmCreateMemPool (Seite 38)

RmDeleteMemPool (Seite 54)

RmMemPoolAlloc (Seite 115)

2.44 RmGetName

Funktion

Catalog nach einem Eintrag durchsuchen

Syntax

```
#include <rmapi.h>

int    RmGetName (
         uint Type,
         uint ID,
         ulong IDEx,
         char * pName);
```

Parameter

Parametername	Bedeutung
Type	Betriebsmittel–Typ
ID	Betriebsmittel–ID
IDEx	Erweiterte Betriebsmittel–ID (–1 = nicht spezifiziert)
pName	Adresse eines Puffers, in den der String abgelegt werden soll.

Beschreibung

Der Aufruf durchsucht den Catalog und gibt den zu `Type`, `ID` und `IDEx` gehörenden Namen zurück.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt.

Fehlercodes

ID dez	Fehlercode	Bedeutung

36	RM_INVALID_ID	Die angegebene id ist unzulässig. Die möglichen IDs sind abhängig von `Type`:
		`0 RM_CATALOG_TASK 0<=id<=2047`
		`1 RM_CATALOG_DEVICE 0<=id<=255`
		`2 RM_CATALOG_POOL 0<=id<=63`
		`3 RM_CATALOG_SEMAPHORE 0<=id<=4095`
		`4 RM_CATALOG_EVENTFLAG 0<=id<=63`
		`5 RM_CATALOG_CNTRL 0<=id<=255`
		`6 RM_CATALOG_LOCALMAILBOX 0<=id<=255`
		`7 RM_CATALOG_MISC 0<=id<=65535`
		`8 RM_CATALOG_USER 0<=id<=65535`
		`10 RM_CATALOG_UNIT 0<=id<=255`
		`11 RM_CATALOG_MESSAGE 0<=id<=2047`
		`255 RM_CATALOG_ALL 0<=id<=65535`

35	RM_INVALID_TYPE	Der angegebene Type ist nicht zulässig. 0<= `Type`<=11
48	RM_IS_NOT_ CATALOGED	Es wurde kein passender Eintrag gefunden.
42	RM_INVALID_ POINTER	Der Zeiger auf den String ist falsch, bzw. würde eine Schutzverletzung auslösen.

Siehe auch

RmCatalog (Seite 23)
RmGetEntry (Seite 68)
RmList (Seite 112)
RmUncatalog (Seite 194)

2.45 RmGetPeriodHPET0

Funktion

Lesen der Periode des HPET-Timer 0

Syntax

```
#include <misc86.h>

unsigned long    RmGetPeriodHPET0 (void);
```

Beschreibung

Der Wert der Periode von HPET-Timer 0 wird in Femto-Sekunden zurückgeliefert.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0		Es wurde keine Initialisierung mit dem Aufruf `RmInitHPET` durchgeführt.
>0		Wert der Periode von HPET-Timer 0 in Femto-Sekunden

Siehe auch

RmGetTimeHPET0 (Seite 95)

RmInitHPET (Seite 103)

2.46 RmGetPhysAddress

Funktion

Physikalische Adresse eines Speicherbereichs ermitteln

Syntax

```
#include <rmapi.h>

int    RmGetPhysAddress (
            void *Pointer,
            ulong *pPhysAddress);
```

Parameter

Parametername	Bedeutung
Pointer	Zu konvertierender Zeiger
pPhysAddress	Zeiger auf eine Variable, in der die physikalische Adresse abgelegt wird.

Beschreibung

Der Aufruf ermittelt die physikalische Adresse des Speicherbereichs, auf den `pPointer` zeigt.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
39	RM_INVALID_ OFFSET	Ungültiger Offset in `pPointer`
21	RM_INVALID_ SELECTOR	Ungültiger Selector in `pPointer`
42	RM_INVALID_ POINTER	Der Zeiger auf den String ist falsch, bzw. würde eine Schutzverletzung auslösen..

Siehe auch

RmChangeDescriptor (Seite 25)

RmChangeDescriptorAccess (Seite 26)

RmCreateDescriptor (Seite 33)

RmDeleteDescriptor (Seite 51)

RmGetLinAddress (Seite 75)

2.47 RmGetSize

Funktion

Länge eines vergebenen Speicherbereichs ermitteln

Syntax

```
#include <rmapi.h>

int    RmGetSize(
           void *pMemory,
           ulong *pSize);
```

Parameter

Parametername	Bedeutung
pMemory	Zeiger auf den Speicherbereich
pSize	Zeiger auf Speicherstelle, in der die Länge des Speicherbereichs zurückgeschrieben wird

Beschreibung

Mit diesem Aufruf kann die Länge eines Speicherbereichs ermittelt werden, der vorher mit `RmAlloc` angefordert wurde.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
38	RM_INVALID_SIZE	Eine Größenangabe war ungültig.
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.

Siehe auch

RmAlloc (Seite 19)

RmCreateMemPool (Seite 38)

RmDeleteMemPool (Seite 54)

RmExclude (Seite 60)

RmFree (Seite 62)

RmFreeAll (Seite 63)

RmMemPoolAlloc (Seite 115)

RmReAlloc (Seite 126)

2.48 RmGetSpinLock

Funktion

Anfordern eines SpinLocks

Syntax

```
#include <misc86.h>

int    RmGetSpinLock( RmSpinLockStruct *SpinLock );
```

Parameter

Parametername	Bedeutung

SpinLock	Adresse einer Struktur vom Typ `RmSpinLockStruct`, die folgendermaßen definiert ist:
	
	`struct _RmSpinLockStruct`
	`{`
	`  uchar Spin;`
	`  uchar CoreID;`
	`  uchar Reserved;`
	`  uchar Count;`
	`  uint  Interruptflags;`
	`} GNU_PACKED;`
	`typedef struct _RmSpinLockStruct RmSpinLockStruct;`

Beschreibung

Das SpinLock muss vorher mit dem Aufruf `RmInitSpinLock` initialisiert werden. Vor dem Aufruf `RmGetSpinLock` sollten, falls momentan nicht schon der Fall, immer die Interrupts gesperrt werden. Nach Rückkehr aus dem Aufruf `RmGetSpinLock` ist die Task im Besitz des SpinLocks. Während des Besitzes des SpinLocks sollten die Interrupts gesperrt bleiben.

Unter RMOS3 kann `RmGetSpinlock` geschachtelt werden. Dass heißt, dass die CPU, die den Spinlock besitzt, `RmGetSpinlock` öfters aufrufen kann, ohne zu blockieren. `RmReleaseSpinlock` muss dann entsprechend oft aufgerufen werden, um den Spinlock wieder freizugeben.

Deshalb sollten SpinLocks nicht auf Taskebene verwendet werden. Wenn doch, darf es in keinem Fall zu einem Wechsel der Task auf einem anderen Prozessor kommen.

SpinLocks können nur innerhalb einer geladenen Applikation benutzt werden, da der Code direkt eingebunden wird. Der Code wird also für jede geladene Applikation neu dazugebunden.

Da die SpinLock-Aufrufe keine Betriebssystemaufrufe sind, werden die entsprechenden Aufrufe auch nicht vom Profiler aufgezeichnet.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
42	RM_INVALID_ POINTER	Der Zeiger `SpinLock` ist ein NULL-Zeiger.
1	RM_SYSTEMERROR	`SpinLock` wurde nicht initialisiert

Beispiel

```
static RmSpinLockStruct lock;
   …
   // initialize spinlock
   RmInitSpinLock (&lock);
   // call spinlock
   disable();
   RmGetSpinLock (&lock);
   …
   RmReleaseSpinLock (&lock);
   enable();
```

Siehe auch

RmGetSpinLockIRQ (Seite 84)

RmInitSpinLock (Seite 105)

RmReleaseSpinLock (Seite 131)

2.49 RmGetSpinLockIRQ

Funktion

Anfordern eines SpinLocks mit Sperren der Interrupts

Syntax

```
#include <misc86.h>

int    RmGetSpinLockIRQ( RmSpinLockStruct *SpinLock );
```

Parameter

Parametername	Bedeutung

SpinLock	Adresse einer Struktur vom Typ `RmSpinLockStruct`, die folgendermaßen definiert ist:
	
	`struct _RmSpinLockStruct`
	`{`
	`  uchar Spin;`
	`  uchar CoreID;`
	`  uchar Reserved;`
	`  uchar Count;`
	`  uint  Interruptflags;`
	`} GNU_PACKED;`
	`typedef struct _RmSpinLockStruct RmSpinLockStruct;`

Beschreibung

Das SpinLock muss vorher mit dem Aufruf `RmInitSpinLock` initialisiert werden. Beim `RmGetSpinLockIRQ` werden zusätzlich im Vergleich zum Aufruf `RmGetSpinLock` in jedem Fall die Interrupts gesperrt und der Interruptzustand im SpinLock vermerkt. Nach Rückkehr aus dem Aufruf `RmGetSpinLockIRQ` ist die Task im Besitz des SpinLocks. Während des Besitzes des SpinLocks sollten die Interrupts gesperrt bleiben.

Da die SpinLock-Aufrufe keine Betriebssystemaufrufe sind, werden die entsprechenden Aufrufe auch nicht vom Profiler aufgezeichnet.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
42	RM_INVALID_ POINTER	Der Zeiger `SpinLock` ist ein NULL-Zeiger.
1	RM_SYSTEMERROR	`SpinLock` wurde nicht initialisiert

Beispiel

```
static RmSpinLockStruct lock;

…
    // initialize spinlock
    RmInitSpinLock (&lock);
    // call spinlock
    RmGetSpinLockIRQ (&lock);
    …
    RmReleaseSpinLockIRQ (&lock);
```

Siehe auch

RmGetSpinLock (Seite 82)

RmInitSpinLock (Seite 105)

RmReleaseSpinLock (Seite 131)

2.50 RmGetStatusMessageQueue

Funktion

Anzahl der vorhandenen Messages ermitteln

Syntax

```
#include <rmapi.h>

int    RmGetStatusMessageQueue(
                uint TaskID,
                uint _FAR *pNumberOfMessages);
```

Parameter

Parametername	Bedeutung
TaskID	Task-ID
pNumberOfMessages	Zeiger auf Speicherstelle, in der die Anzahl der vorhandenen Messages zurückgeschrieben wird.

Beschreibung

Mit diesem Aufruf kann die Anzahl der vorhandenen Messages in der Messagequeue ermittelt werden ohne eine Message aus der Queue zu entfernen.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Eine ungültige `TaskID` wurde übergeben.
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.

Siehe auch

RmCreateMessageQueue (Seite 40)

RmDeleteMessageQueue (Seite 55)

RmReadMessage (Seite 124)

RmSendMessage (Seite 155)

2.51 RmGetSysB

Funktion

System Control Block auslesen

Syntax

```
#include <rmapi.h>

int    RmGetSysB (RmSysB **ppSysB);
```

Parameter

Parametername	Bedeutung
ppSysB	Adresse eines Zeigers, in dem die Adresse des System Control Block abgelegt wird.

Beschreibung

Der Ausfruf dient zum Auslesen des System Control Block. In diesem sind die wichtigsten Systemparameter enthalten z.B. Systemtyp (RMOS2, RMOS3 etc.), Version, Bootlaufwerk. Außerdem werden dort die Adressen der Funktionen zur Bedienung der Hardware (vgl. `RmInitOS`) gespeichert.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
42	RM_INVALID_ POINTER	Der übergebene Pointer zeigte nicht auf einen zulässigen Speicherbereich.

Siehe auch

Struktur `RmSysB` im Referenzhandbuch Teil II, Kapitel 3

2.52 RmGetSystemPeriod

Funktion

Systemtakt abfragen

Syntax

```
#include <rmapi.h>

int    RmGetSystemPeriod(ulong *pTime);
```

Parameter

Parametername	Bedeutung
pTime	Zeiger auf Systemtakt in Mikrosekunden

Beschreibung

Mit `RmGetSystemPeriod` wird der Systemtakt ermittelt. Die Voreinstellung des Systemtakts beträgt 1 ms. Eine Änderung des Systemtakts ist mit dem SVC `RmInitOS` möglich.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
42	RM_INVALID_ POINTER	`pTime` unzulässig

Siehe auch

RmGetSysB (Seite 87)

RmInitOS (Seite 106)

2.53 RmGetSystemTime

Funktion

Datum und Uhrzeit vom System übertragen

Syntax

```
#include <rmapi.h>

int    RmGetSystemTime(RmAsciiTimeStruct *pTimeInfo);
```

Parameter

Parametername	Bedeutung
pTimeInfo	Zeiger auf eine Struktur vom Typ `RmAsciiTimeStruct`

Beschreibung

Mit `RmGetSystemTime` wird die lokale Zeit mit Sekundenauflösung in eine Struktur vom Typ `RmAsciiTimeStruct` kopiert:

```
typedef struct _RmAsciiTimeStruct
{
            char day[2];    /* Offs. 0x0: 01 bis 31                */
            char delim0;    /* Offs. 0x2:'-'                       */
            char month[3]   /* Offs. 0x3: JAN, FEB, MAR, APR,
                               MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC */
            char delim1;    /* Offs. 0x6:'-'                       */
            char year[4];   /* Offs. 0x7: 0 bis 9999               */
            char blank;     /* Offs. 0xB:' '                       */
            char hour[2];   /* Offs. 0xC: 0 bis 23                 */
            char colon0;    /* Offs. 0xE:':'                       */
            char min[2];    /* Offs. 0xF: 0 bis 59                 */
            char colon1;    /* Offs. 0x11:':'                      */
            char sec[2];    /* Offs. 0x12: 0 bis 59                */
} RmAsciiTimeStruct
```

Datum und Uhrzeit sind ASCII–Zeichen und besitzen die in den Kommentaren stehenden Wertebereiche.

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
42	RM_INVALID_ POINTER	`pTimeInfo` unzulässig

Siehe auch

Beschreibung zu Timer im Benutzerhandbuch

RmSetSystemTime (Seite 188)

2.54 RmGetTaskID

Funktion

Task–ID ermitteln

Syntax

```
#include <rmapi.h>

int    RmGetTaskID(
            uint Tcb,
            uint *pTaskID);
```

Parameter

Parametername	Bedeutung
Tcb	TCB–Offset–Adresse (RM_OWN_TASK (-1)= eigene Task)
pTaskID	Zeiger auf Task–ID

Beschreibung

`RmGetTaskID` ermittelt die Task-ID aus der mit `Tcb` angegebenen TCB–Offset–Adresse. Mit `Tcb=RM_OWN_TASK` wird die Task–ID der anfordernden Task zurückgegeben.

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.
2	RM_PARAMETER_ ERROR	`Tcb` ungültig

2.55 RmGetTaskInfo

Funktion

Taskinformation ermitteln

Syntax

```
#include <rmapi.h>

int    RmGetTaskInfo(
           uint TaskID,
           RmTaskInfoStruct *pTaskInfo);
```

Parameter

Parametername	Bedeutung
TaskID	Task–ID (RM_OWN_TASK (-1) = eigeneTask)
pTaskInfo	Zeiger auf eine Struktur vom Typ `RmTaskInfoStruct`

Beschreibung

`RmGetTaskInfo` liefert die Priorität, Task–Status, Task–Stack–Adresse, letzte Startzeit, die Task–Flags (z.B. NPX–Konfigurierung) und die Adresse des TCD–Blocks zurück. Die Information wird in einer Struktur vom Typ `RmTaskInfo` mit einer Länge von 24 Byte zurückgeschrieben:

```
typedef struct _RmTaskInfoStruct
{
char pri;             /* Offs. 0x0: task priority */
char state;           /* Offs. 0x1: task state */
uint * sp_ss;         /* Offs. 0x2: top of stack */
#if RM3
 longlong rdy_time;   /* Offs. 0x8: time, task was last ready (8
 bytes)*/
#else
 uint rdy_time[3];    /* time, task was last ready (6 bytes)*/
#endif
char cpu_id;          /* Offs. 0x10: reserved */
char flg;             /* Offs. 0x11: task flag state (only Bit 0-7)*/
RmTCDStruct *tcd_ptr; /* Offs. 0x12: pointer to TCD structure*/
}RmTaskInfoStruct
```

Das erste Integer–Element von `rdy_time` enthält die obersten, das dritte die untersten Bitstellen des 48–Bit–Zählwerts. `rdy_time[3]` bzw. longlong `rdy_time` wird immer dann aktualisiert, wenn eine Task vom Zustand DORMANT in den Zustand BEREIT wechselt.

Die niederwertigen beiden Bits von `state` können folgende Zustände annehmen:

Zustand	Wert
RM_READY	1
RM_ACTIVE	3
RM_BLOCKED	2
RM_DORMANT	0
RM_NOT_EXISTENT	255

Wenn eine Task im Zustand BLOCKED ist, wird der Grund in `state` noch genau aufgeschlüsselt. Welcher Wert von `state` welchem Grund entspricht, entnehmen Sie der Tabelle im Abschnitt "RMOS3–DEBUGGER–Bedienung" (siehe Debugger–Kommando QUERY). Der Task–Zustand NON–EXISTENT kann nicht aus `state` herausgelesen werden.

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Struktur `RmTaskInfo` wurde initialisiert

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	`TaskID` ungültig
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.

Siehe auch

Systemkonfigurierung, Taskverwaltung, TCD–Block, Debugger–Kommando QUERY

2.56 RmGetTaskPriority

Funktion

Task–Priorität ermitteln

Syntax

```
#include <rmapi.h>

int    RmGetTaskPriority(
           uint TaskID,
           uint *pPriority);
```

Parameter

Parametername	Bedeutung
TaskID	Task–ID (RM_OWN_TASK = eigene Task)
pPriority	Zeiger auf eine Speicherstelle, die die Priorität der Task enthält

Beschreibung

`RmGetTaskPriority` liefert die Task–Priorität zurück.

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	`TaskID` ungültig
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.

Siehe auch

Systemkonfigurierung, Taskverwaltung, TCD–Block, Debugger–Kommando QUERY

RmGetTaskInfo (Seite 91)

RmGetTaskState (Seite 94)

2.57 RmGetTaskState

Funktion

Task–Zustand ermitteln

Syntax

```
#include <rmapi.h>

int    RmGetTaskState(
            uint TaskID,
            uint *pTaskState);
```

Parameter

Parametername	Bedeutung
TaskID	Task–ID (RM_OWN_TASK = eigene Task)
pTaskState	Zeiger auf eine Speicherstelle, die den Zustand der Task enthält

Beschreibung

`RmGetTaskState` liefert den Task–Status zurück.

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	`TaskID` ungültig
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.

Siehe auch

Systemkonfigurierung, Taskverwaltung, TCD–Block, Debugger–Kommando QUERY

RmGetTaskPriority (Seite 93)

RmGetTaskInfo (Seite 91)

2.58 RmGetTimeHPET0

Funktion

Lesen des aktuellen Werts des HPET-Timer 0

Syntax

```
#include <misc86.h>

unsigned long long    RmGetTimeHPET0 (void);
```

Beschreibung

Der aktuelle Wert des Zählers von HPET-Timer 0 wird zurückgeliefert.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0		Es wurde keine Initialisierung mit dem Aufruf `RmInitHPET` durchgeführt.
>0		Wert des Zählers von HPET-Timer 0

Siehe auch

RmGetPeriodHPET0 (Seite 79)
RmInitHPET (Seite 103)

2.59 RmHookException

Funktion

Einhängen einer Callback-Funktion, die bei einer Exception aufgerufen wird.

Syntax

```
#include <rmapi.h>

int _FIXED _FAR    RmHookException (void _FAR * CallBack);
```

Parameter

Parametername	Bedeutung
CallBack	Callback-Funktion des Anwenders

Beschreibung

Die Ausführung der CallBack-Funktion erfolgt immer auf PL0.

Wenn die Exception im A-Zustand erfolgte, können alle RMOS3- und CRUN-Aufrufe ohne Einschränkungen benutzt werden. Es kann also z.B. auch in eine Datei geschrieben werden. Normalerweise wird eine Exception immer im A-Zustand auftreten. Wenn vom Anwender ein Interrupthandler für den I- bzw. S-Zustand installiert wurde, könnte eine Exception im I- bzw. S-Zustand auftreten.

Im I- und S-Zustand dürfen nur nicht blockierende RMOS3-Aufrufe und kein RmIO verwendet werden. Im I-Zustand ist zusätzlich zu beachten, dass das Ergebnis eines RMOS3-Aufrufes nicht zur Verfügung steht. Es dürfen nur CRUN-Aufrufe benutzt werden, die keinen Treiberaufruf beinhalten. Verwendet werden dürfen also z.B. memset, sprintf und atoi, nicht zur Verfügung stehen z.B. printf, fwrite und fread.

Wenn der übergebene Parameter den Wert 0 hat, wird eine früher eingehängte Callback-Funktion wieder entfernt.

Hinweis

Diese Funktion ist nur für GNU verfügbar. SVC-Exceptions werden damit nicht behandelt.

Bei der Meldung "No SRBs" und "No SMRs" wird der Interrupt RM_USER_INT (Vektor 245) ausgelöst.

Struktur RmExceptionDataStruct

In RMTYPES.H ist die Struktur RmExceptionDataStruct definiert:

```
struct _RmExceptionDataStruct
{
   unsigned short     state; /* I- oder S- oder A-Zustand */
   unsigned char      task_state; /* Zustand der Task, z.B. READY, ACTIVE */
   unsigned short     task_id;
   unsigned short     svc_no;
   unsigned char      intr_no; /* z.B. 0 für "Division durch Null" */
   unsigned int       err_code;
   unsigned int      eflag;
   unsigned int       eip;
   unsigned short     cs;
   unsigned int       eax;
   unsigned int       ebx;
   unsigned int       ecx;
   unsigned int      edx;
   unsigned int       esi;
```

```
   unsigned int       edi;
   unsigned int       ebp;
   unsigned int       esp;
   unsigned short     ss;
   unsigned short     ds;
   unsigned short     es;
   unsigned short     fs;
   unsigned short     gs;
} GNU_PACKED;
typedef struct _RmExceptionDataStruct RmExceptionDataStruct;
```

Parametername	Bedeutung

state	Beschreibt den Zustand, in dem die Exception aufgetreten ist und ist immer gültig. Der Zustand wird nur durch die niederwertigen 2 Bits festgelegt (Maske STATE_MASK notwendig). Bit 8 bis 15 enthalten die Id des Cores, auf dem die Exception aufgetreten ist.
	
	`#define STATE_MASK 0x0003`
	`#define A_STATE 0`
	`#define SVC_STATE 1`
	`#define I_STATE 2`
	`#define S_STATE 3`
	Das Bit 3 zeigt an, ob die Variable `err_code` einen gültigen Wert enthält.
	`#define ERROR_CODE_BIT 4`

task_state	Falls die Exception im A-Zustand aufgetreten ist, enthält diese Variable den Taskzustand (siehe `rmdef.h`).
	
	`#define RM_DORMANT 0u /* task is dormant */`
	`#define RM_READY 1u /* task in ready state */`
	`#define RM_BLOCKED 2u /* task waits for I/O */`
	`#define RM_ACTIVE 3u /* task is active */`
	`#define RM_NOTEXISTENT 255u`
	`/* task doesn't exist */`
	
	`/* Zusätzliche Information, wenn die Task im Zustand RM_BLOCKED ist */`
	
	`#define RM_STA_EF 4 /* event flag wait */`
	`#define RM_STA_SEMA (RM_STA_EF+4)`
	`/* semaphore or control wait */`

task_id	Falls die Exception im A-Zustand aufgetreten ist, enthält diese Variable die TaskID, in der die Exception aufgetreten ist.
svc_no	Nur gültig, wenn die Exception im SVC-Zustand aufgetreten ist (sollte nicht vorkommen bzw. kann nur bei einem `RmIO` mit einem eigenen Treiber auftreten). In diesem Fall gibt die Variable die SVC-Nummer an (0x15 für `RmIO`).
intr_no	Gibt die Nummer der Exception an, z.B. 0 für "DIVIDE ERROR" oder 13 für "GENERAL PROTECTION".
err_code	Nur gültig, wenn in der Variablen `state` das Bit "ERROR_CODE_BIT" gesetzt ist.
eflag … gs	Inhalt der entsprechenden Prozessorregister

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	SVC erfolgreich ausgeführt

Beispiel

```
CallBack(RmExceptionDataStruct _FAR * exception)
{
    x_nucprintf("\r\nCallBack: state: 0x%x, task_id: 0x%x, svc_no:
        0x%x, intr_no: %u\r\n",
        exception->state, exception->task_id, exception->svc_no,
        exception->intr_no);
}
…
int   Error;

Error = RmHookException (CallBack);
if (Error)
{
    printf ("*** error with RmHookException
           (error = %u)\n",Error);
}
```

2.60 RmInitFastTask

Funktion

Initialisierung einer Task zum Start in einem schnellen Zyklus.

Syntax

```
#include <fasttick.h>

int    RmInitFastTask (
             uint usecT,
             uint TaskID);
```

Parameter

Parametername	Bedeutung
usecT	Zykluszeit in Mikrosekunden, in der die Task gestartet wird.
TaskID	Task-ID derjenigen Task, die im schnellen Zyklus gestartet wird

Beschreibung

Die Funktion `RmInitFastTask` startet bis zu fünf zuvor mit `RmCreateTask` oder `RmCreateTaskEx` erzeugte Tasks im gewünschten Zeitintervall `usecT`. Dieses Zeitintervall ist taskspezifisch und ein ganzzahliges Vielfaches des Grundtaktes, welcher mit `RmInitFastTick` eingestellt ist.

Die Task mit `TaskID` wird zyklisch im Raster von `usecT` Mikrosekunden gestartet und muss sich am Ende jedes Zyklus wieder selbst mit `RmEndTask` beenden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt
-1		Folgende Fehlerursachen sind möglich: • `usecT` enthält einen ungültigen Wert. • `TaskID` hat den Wert 0 • Es sind bereits fünf Tasks im schnellen Zyklus gestartet

Fehlercodes

ID dez	Fehlercode	Bedeutung
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.
2	RM_PARAMETER_ ERROR	`Tcb` ungültig

Beispiel

```
#include<stdio.h>
#include<rmapi.h>
#include<fasttick.h>

static uint Counter;

void _FAR _FIXED DummyTask ( void )
{
    Counter++;
    RmEndTask();
}

void main (void)
{
...
    RmCreateTask ("FastTask",0x4000ul,250u,DummyTask,&TaskID);

    Error = RmInitFastTick (0u,100u);
    if (Error != RM_OK)
    {
        printf ("error with RmInitFastTick (0x%x)\n", Error);
        return;
    }

    Counter = 0;
    Error = RmInitFastTask (200u, TaskID);
    if (Error != RM_OK)
    {
        RmDeleteTask(TaskID);
        printf ("error with RmInitFastTask (0x%x)\n", Error);
        return;
    }
    RmPauseTask(1000ul);
    printf ("Counter = %d\n", Counter);
...
    Error = RmRemoveFastTask (TaskID);
    if (Error != RM_OK)
    {
        RmDeleteTask(TaskID);
        printf ("error with RmRemoveFastTask (0x%x)\n", Error);
        return;
    }
    Error = RmRemoveFastTick ();
....
}
```

Siehe auch

RmInitFastTick (Seite 101)

RmRemoveFastTick (Seite 136)

2.61 RmInitFastTick

Funktion

Initialisierung eines schnellen Zyklus.

Syntax

```
#include <fasttick.h>

int RmInitFastTick (
 uint mode,
 uint usec);
```

Parameter

Parametername	Bedeutung
mode	Der Parameter `mode` berücksichtigt den Takt des Timerbausteins und kann folgende Werte annehmen: `0`: Standard PC `1`: SMP16-CPU40 bzw. SMP16-CPU41
usec	Der Parameter `usec` kann die Werte 10, 20, 100, 200 und 500 Mikrosekunden annehmen. 10 und 20 Mikrosekunden funktionieren nur im APIC-Modus.

Beschreibung

Mit der Funktionalität des schnellen Zyklus können unter RMOS3 bis zu fünf Tasks im Raster von 100 usec gestartet werden und erlaubt die Realisierung schneller Regelalgorithmen.

`RmInitFastTick` installiert einen Timerinterupthandler der es ermöglicht, über einen SW-Interrupt eine Task zyklisch zu starten.

Die Einstellung des Grundzyklus erfolgt über den Parameter `usec` und kann die Werte 10, 20, 100, 200 oder 500 Mikrosekunden annehmen. Die Werte 10 und 20 Mikrosekunden funktionieren nur im APIC-Modus. Wird die Funktion im PIC-Modus mit diesen Werten aufgerufen, kommt die Funktion mit Rückgabewert -1 zurück.

Der Parameter `mode` erlaubt die Anpassung an spezifische Hardware-Architekturen und berücksichtigt den Standard PC Timerbaustein mit einer Taktfrequenz von 1,193180 MHz sowie den Timerbaustein der SMP16-CPU40 bzw. SMP16-CPU41 mit jeweils einer Frequenz des Timerbausteins von 1,189200 MHz.

Die Funktion `RmInitFastTask` startet bis zu fünf zuvor mit `RmCreateTask` oder `RmCreateTaskEx` erzeugte Tasks im gewünschten Zeitintervall. Dieses Zeitintervall ist taskspezifisch und ein ganzzahliges Vielfaches des Grundtaktes.

Hinweis

Nach Ablauf des Grundtaktes wird der ursprüngliche Timerinterrupthandler für den Systemtakt von RMOS3 aufgerufen.

Der RMOS3 Systemtakt muss auf 1ms eingestellt sein, um die korrekte Abarbeitung des Zyklus gewährleisten zu können.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt
-1		Folgende Fehlerursachen sind möglich: • Timertakt ist nicht 1ms. • `usec` enthält einen ungültigen Wert. • `mode` wird nicht unterstützt

Beispiel

Siehe RmInitFastTask (Seite 99)

Siehe auch

RmInitFastTask (Seite 99)
RmRemoveFastTask (Seite 135)

2.62 RmInitHPET

Funktion

Initialisierung der Funktion zum Auslesen des HPET-Timer 0

Syntax

```
#include <misc86.h>

int    RmInitHPET (void);
```

Beschreibung

Auf dem Zielsystem muss ein HPET-Timer zur Verfügung stehen und der HPET-Treiber `HPET.DRV` muss geladen sein.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt
1	RM_SYSTEMERROR	Der HPET-Treiber wurde nicht geladen bzw. es stehen keine HPET-Timer zur Verfügung.

Siehe auch

RmGetPeriodHPET0 (Seite 79)

RmGetTimeHPET0 (Seite 95)

2.63 RmInitShIntClient

Funktion

Initialisierung des Clients für shared Interrupts (in RMISHCLI.LIB enthalten)

Syntax

```
#include <Rmpci.h>

int    RmInitShIntClient( void );
```

Beschreibung

Initialisierung des Clients für shared Interrupts, indem aus einer definierten Mailbox die Zugriffsinformationen auf den Server abgeholt werden.

Nach der Initialisierung können die anderen Funktionen des Clients ausgeführt werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
48	RM_IS_NOT_ CATALOGED	Der Shared Interrupt Server ist nicht katalogisiert

Siehe auch

RmClrShIntISHandler (Seite 28)

RmPciSearchFunction (Seite 118)

RmPciSearchSubFunction (Seite 120)

RmSetShIntISHandler1 (Seite 181)

RmSetShIntISHandler2 (Seite 183)

2.64 RmInitSpinLock

Funktion

Initialisierung eines SpinLocks

Syntax

```
#include <misc86.h>

int    RmInitSpinLock( RmSpinLockStruct *SpinLock );
```

Parameter

Parametername	Bedeutung

SpinLock	Adresse einer Struktur vom Typ `RmSpinLockStruct`, die folgendermaßen definiert ist:
	
	`struct _RmSpinLockStruct`
	`{`
	` uchar Spin;`
	` uchar CoreID;`
	` uchar Reserved;`
	` uchar Count;`
	` uint Interruptflags;`
	`} GNU_PACKED;`
	`typedef struct  RmSpinLockStruct RmSpinLockStruct;`

Beschreibung

Dieser Aufruf initialisiert eine SpinLock-Struktur.

Die SpinLock-Aufrufe sind keine Betriebssystemaufrufe und werden deshalb nicht vom Profiler aufgezeichnet.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
42	RM_INVALID_ POINTER	Der Zeiger `SpinLock` ist ein NULL-Zeiger.

Siehe auch

RmGetSpinLock (Seite 82)

RmGetSpinLockIRQ (Seite 84)

RmReleaseSpinLock (Seite 131)

RmReleaseSpinLockIRQ (Seite 133)

2.65 RmInitOS

Funktion

Anfangsadressen der Unterprogramme zur Bedienung der Hardware übergeben

Syntax

```
#include <rmapi.h>

int    RmInitOS (
            ulong SystemPeriod,
            rmfarproc InitTimerProc,
            rmnearproc IssueEOIProc,
            int(_FAR *SetPICMaskProc) (uint IRQ),
            int(_FAR *SetClockProc) (void *pDate),
            int(_FAR *GetClockProc) (void *pDate));
```

Parameter

Parametername	Bedeutung
SystemPeriod	RMOS3–Systemtakt in Millisekunden (z.B. 10 für 10 ms. vgl. Timer–Funktionen) (0: bisherige Größe verwenden).
InitTimerProc	Funktion zur Initialisierung des Timers. Sie wird jedesmal aufgerufen, wenn ein neuer Systemtakt gesetzt wird (0: bisherige Größe verwenden).
IssueEOIProc	Anfangsadresse für EOI–Behandlung (vormals X_ISSUE_EOI) (0: bisherige Größe verwenden)
SetPICMaskProc	Anfangsadresse der Routine, mit der die PIC–Interrupt– Maske gesetzt wird (0: bisherige Funktion verwenden).
SetClockProc	Anfangsadresse der Routine, mit der die Echtzeituhr gesetzt wird (0: bisherige Funktion verwenden).
GetClockProc	Anfangsadresse der Routine, mit der die Echtzeituhr gelesen wird (0: bisherige Funktion verwenden).

Beschreibung

Mit diesem Aufruf werden dem System die Funktionen zur Bedienung der Hardware bekanntgegeben. So kann auch zur Laufzeit ein PIC nachträglich kaskadiert werden. Es kann auch die Auflösung des Timerinterrupts zur Laufzeit verändert werden. Es wird ebenfalls der Timerinterrupt installiert, dazu muss die Timer–Initialisierungsroutine `InitTimerProc` die Interruptnummer im System Control Block `X_SYSB.TIMER_INT` setzen. Der Aufruf kann zu jedem Zeitpunkt und unabhängig von anderen Konfigurations–SVCs aufgerufen werden. Falls die übergebene Anfangsadresse 0 ist, wird ein bereits früher übergebenes Unterprogramm weiterhin verwendet. Besitzen die Zeiger für `SetPICMaskProc`, `SetClockProc` und `GetClockProc` einen ungültigen Selektor, bzw. keinen Codeselector, werden die Dummyfunktionen `RcDummySetPICMask()`, `RcDummyGetClock()` bzw. `RcDummySetClock()` installiert. Die Funktion `SetPICMaskProc` wird von der API–Funktion `RmSetDeviceHandler()` zum Maskieren eines IRQ eingesetzt (s.d.). Beim ersten Aufruf von `RmInitOS()` müssen die Funktionen `IssueEOIProc` und `InitTimerProc` installiert werden. Als kleinster Systemtakt wird 1 ms unterstützt.

Hinweis

Diese Funktion ist nicht im Speichermodell Flat verwendbar.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
21	RM_INVALID_ SELECTOR	Es wurde ein ungültiger Selector verwendet.
35	RM_INVALID_TYPE	Der Systemtakt wurde nicht gesetzt oder ist kleiner als 1 ms.
53	RM_NO_EOI	Es wurde keine EOI-Routine installiert.
54	RM_NO_INIT_ TIMER	Es wurde keine Timerinitialisierung installiert.

Siehe auch

RmGetSysB (Seite 87)

RmGetSystemPeriod (Seite 88)

RmRunOS (Seite 147)

2.66 RmIO

Funktion

E/A–Operation anfordern

Syntax

```
#include <rmapi.h>

int    RmIO(
         uint Function,
         uint DeviceID,
         uint UnitID,
         uint FlagID,
         uint FlagMask,
         RmIOStatusStruct *pState,
         void *pParam);
```

Parameter

Parametername	Bedeutung
Function	Operationscode
DeviceID	Treiber–ID
UnitID	Geräte–ID
FlagID	Ereignisflag–Gruppe–ID
FlagMask	Maske legt fest, welche Bits nach Abschluß der Operation gesetzt werden
pState	Zeiger auf 1. Byte eines 8–Byte–Statusfelds
pParam	Zeiger auf Adresse des E/A–Parameterblocks

Beschreibung

RmIO fordert eine E/A–Operation von einem Treiber an.

Die Ereignisflag–Bits, die durch `FlagID` und `FlagMask` festgelegt sind, werden durch RMOS3 bei der Annahme des SVC zurückgesetzt und bei der Beendigung des E/A–Operation gesetzt. Es kann eine lokale oder globale Ereignisflag–Gruppe angegeben werden.

Der Beendigungsstatus der E/A–Operation wird in einem 8–Byte–Statusfeld durch den Treiber zurückgeschrieben.

Die Synchronisation der Task–Abarbeitung nach der Beendigung der E/A–Operation erfolgt entweder durch `RmGetFlag` anhand der mit `FlagMask` festgelegten Bits der Ereignisflag–Gruppe oder durch das Setzen des Wartebits im Operationscode.

Falls mehrere `RmIO` ohne Warten abgesetzt werden, muss für jeden ausstehenden Aufruf ein 8–Byte–Statusfeld und ein eigener Parameterblock als Parameter übergeben werden (durch Zeiger). Der Inhalt und die Größe des Parameterblocks hängen vom Treiber ab.

Hinweis

Diese Funktion ist nicht im Speichermodell Flat verwendbar.

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
	siehe treiberspezifische Rückgabewerte	

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Eine ungültige ID (`dev_id`, `unit_id` oder `flg_id`) wurde übergeben.
44	RM_INVALID_ FUNCTION	Eine übergebene Funktionsnummer ist ungültig oder wird nicht unterstützt (`Function`, Treiber Typ II).
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.

Siehe auch

Dokumentation der Treiber–Funktionen im Referenzhandbuch Teil II, Kapitel 5 "Funktionen und Konfiguration des Basis-Ein-/Ausgabe-Systems"

2.67 RmKillTask

Funktion

Task in Zustand DORMANT oder NICHT-EXISTENT bringen

Syntax

```
#include <rmapi.h>

int    RmKillTask(
         uint Mode,
         uint TaskID);
```

Parameter

Parametername	Bedeutung	
Mode	Gewünschter Task–Zustand:	
	RM_TASK_END (0x0)	Task in den Zustand DORMANT versetzen (entspricht `RmEndTask`)
	RM_TASK_DELETE (0x1)	Task löschen (entspricht `RmDeleteTask`)
TaskID	ID der zu löschenden Task (RM_OWN_TASK (-1) = eigene Task)	

Beschreibung

Der Aufruf bringt eine beliebige Task (auch sich selbst) in den Zustand DORMANT oder nicht–existent, unabhängig in welchem Zustand sie sich vorher befand.

Besonderheiten treten auf, wenn sich die Zieltask im Zustand WARTEND befindet. Bei folgenden Zuständen ist `RmKillTask` unzulässig und wird mit Fehlermeldung beendet:

- Debugger Breakpoint
- bei sämtlichen Laufzeitfehlern außer General Protection
- Beenden/Löschen durch `RmKillTask` bereits angemeldet (zweimaliges Anwenden des Aufrufs `RmKillTask` auf dieselbe Task)

Im folgenden Fall wird die Task nicht sofort in den Zustand DORMANT oder nicht–existent überführt, sondern lediglich angemeldet:

Warten auf Beendigung eines E/A–Auftrags:
Die betreffende Task bleibt weiterhin im Zustand WARTEND. Erst wenn der E/A–Auftrag beendet wurde, tritt die Zustandsänderung ein. Es kann also sein, dass die Task nach dem Aufruf weiterhin in einem passiven (geblockten) Zustand sichtbar bleibt. Die Task befindet sich dann in den Blockzuständen 0xCA (RM_TASK_END) bzw. 0xCE (RM_TASK_DELETE) (siehe auch Debugger–Kommando QUERY).

Option RM_TASK_DELETE

Alle Startanforderungen in der Warteschlange werden entfernt. Falls die Zieltask mit der Koordination "Warten bis bereit" oder "Warten bis Beendigung" gestartet wurde, werden alle betreffenden Tasks, die
`RmStartTask`/`RmQueueStartTask` abgesetzt haben, auf das vorzeitige Beenden/Löschen der Zieltask hingewiesen.

Option RM_TASK_END

Alle Startanforderungen in der Warteschlange bleiben erhalten. Der weitere Ablauf der aufrufenden Tasks ist der gleiche, als hätte die Zieltask `RmEndTask` abgesetzt. Falls die Zieltask mit der Koordination "Warten bis bereit" oder "Warten bis Beendigung" gestartet wurde, wird die Task, die `RmStartTask`/`RmQueueStartTask` abgesetzt hat, auf das vorzeitige Beenden/Löschen der Zieltask hingewiesen.

Hinweis

Betriebsmittel, wie z.B. Speicherpools, Mailboxen oder Semaphore, die noch im Besitz der Task sind, werden nach dem Versetzen in den Zustand
DORMANT oder dem Löschen der Task nicht automatisch freigegeben. Diese Betriebsmittel müssen - falls möglich - von einer anderen Task freigegeben werden, ansonsten sind sie für den weiteren Betrieb verloren.

`RmKillTask` darf auch auf Tasks angewandt werden, die Funktionen der CRUN benutzen.

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
35	RM_INVALID_TYPE	Ein ungültiger Parameter (`Mode`) wurde übergeben.
36	RM_INVALID_ID	Eine ungültige `TaskID` wurde übergeben.
22	RM_INVALID_ TASK_STATE	Aufruf bei jetzigem Zustand der Task unzulässig

Siehe auch

Debugger-Kommando QUERY

RmDeleteTask (Seite 56)

x_cr_killtsk (Seite 434)

2.68 RmList

Funktion

Catalog–Einträge auflisten

Syntax

```
#include <rmapi.h>

int    RmList (
         uint Type,
         uint Count,
         uint *pIndex,
         uint *pNumEntries,
         RmEntryStruct *pEntry);
```

Parameter

Parametername	Bedeutung
Type	Betriebsmittel–ID
Count	Anzahl Betriebsmittel
pIndex	Index, ab dem die Einträge gelistet werden. start = 0
pNumEntries	Anzahl der zu listenden Einträge. In `NumEntries` wird zurückgemeldet, wie viele Einträge in `pEntry` abgelegt wurden. Ist `NumEntries` u1, muss `pEntry` auf ein Feld aus Strukturen zeigen.

pEntry	Adresse einer Struktur vom Typ `RmEntryStruct`, die folgendermaßen definiert ist:
	
	`typedef struct _RmEntryStruct`
	`{`
	`  uchar  slen;  /*Offs. 0x0  */`
	`  char  string[16]; /*Offs. 0x1  */`
	`  uchar  type;  /*Offs. 0x11 */`
	`  ulong  ide;  /*Offs. 0x12 */`
	`  ushort id;  /*Offs. 0x16 */`
	`} RmEntryStruct`

Beschreibung

Der Aufruf listet eine Anzahl von Einträgen des Catalogs in dem angegebenen Puffer `pEntry` auf.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
35	RM_INVALID_TYPE	Der angegebene `Type` ist nicht zulässig. 0<= `Type` <=11
42	RM_INVALID_ POINTER	Der Zeiger auf den String ist falsch, bzw. würde eine Schutzverletzung auslösen..

Siehe auch

RmGetEntry (Seite 68)

RmGetName (Seite 77)

RmUncatalog (Seite 194)

2.69 RmMapMemory

Funktion

Physikalischen Speicher adressieren

Syntax

```
#include <rmapi.h>

int    RmMapMemory (
            ulong PhysAddress,
            ulong Length,
            void **pPointer );
```

Parameter

Parametername	Bedeutung
PhysAddress	Physikalische Anfangsadresse
Length	Länge des zu mappenden Speicherbereichs
pPointer	Adresse einer Variablen vom Typ Zeiger, in die die lineare Adresse des neu eingerichteten Speicherbereichs eingetragen wird. Über `*pPointer` können Programme direkt auf den gemappten Adressbereich zugreifen. Ist die lineare Adresse, d.h. `*pPointer` gleich NULL, konnte der Speicherbereich nicht abgebildet werden.

Beschreibung

Die Funktion `RmMapMemory` bildet einen physikalischen Speicherbereich (z.B. Dual-Port-RAM oder Memory Mapped I/O) auf einen linaren Adressraum (Anfangsadresse: `*pPointer`, Länge: `Length`) ab. Anwenderprogramme können über den zurückgelieferten Zeiger `*pPointer` auf den Speicher zugreifen (Zugriff ist READ/WRITE).

Wird `RmMapMemory` auf Privilege Level 0 (PL0) aufgerufen, gilt die Freigabe des Speichers nur auf PL0, d.h. ein Programm auf PL3 kann nicht auf den Speicher zugreifen. Wird `RmMapMemory` auf Privilege Level 3 (PL3) aufgerufen, gilt die Freigabe des Speichers auf PL0 und PL3.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.

Siehe auch

RmDeleteDescriptor (Seite 51)

2.70 RmMemPoolAlloc

Funktion

Speicherbereich aus Speicherpool allokieren

Syntax

```
#include <rmapi.h>

int    RmMemPoolAlloc (
            ulong TimeOutValue,
            uint Mode,
            uint PoolID,
            ulong Size,
            void ** ppMemory);
```

Parameter

Parametername	Bedeutung	
TimeOutValue	Maximale Wartezeit bis zur Ausführung	
	RM_CONTINUE (0)	Task fortsetzen, ohne auf Zuteilung des Speichers zu warten.
	RM_WAIT (-1)	Auf Zuteilung des Speichers warten.
	0 ... ULONG_MAX	Zeitintervall in ms. Die Task wartet entweder bis der Speicher zugeteilt wird oder das Timeout abgelaufen ist.
	Zur Zeitangabe können die Werte für Stunden, Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^{31} Millisekunden.	
	RM_HOUR(hour)	Wartet (hour) Stunden
	RM_MINUTE(min)	Wartet (min) Minuten
	RM_SECOND(sec)	Wartet (sec) Sekunden
	RM_MILLISECOND(ms)	Wartet (ms) Millisekunden
	Die Zeitangaben dürfen nur ein Vielfaches des Systemtaktes (Standard 10ms) sein.	
Mode	Allokierungsmethode für den Speicher:	
	RM_AUTOFREE (0x0)	Der Speicher wird automatisch mit `RmFreeAll()` freigegeben. Er ist taskspezifisch zugeordnet.
	RM_NOAUTOFREE (0x2)	Der Speicher wird nicht automatisch mit `RmFreeAll()` freigegeben.
	RM_FLATALLOC (0x0C)	Der Zeiger `ppMemory` enthält `FlatSelector:FlatOffset`. `FlatOffset` kann direkt als Pointer an Flat–Programme übergeben werden.
	RM_STACKALLOC (0x2000)	Für Verwendung des Speichers mit `RmCreateTaskEx` als Stack. Speicher wird mit "Guarded Pages" geschützt. Ein Stacküberlauf wird erkannt. Adresse liegt oberhalb von Adresse 0xf0000000. Wenn Adresse mit RmGetPhysAddress (Seite 80) umgewandelt wird, kann auf Privilege Level 3 (PL3) nicht auf die umgerechnete Adresse zugegriffen werden.
	RM_USERALLOC (0x20)	Für Verwendung bei Speicheranforderung einer segmentierten Privilege Level 0 (PL0)-Task, die den Zeiger auf den Speicherbereich an eine PL3-Task schicken will.

Parametername	Bedeutung	
	RM_SYSALLOC (0x10)	Für Verwendung bei Speicheranforderung einer segmentierten Privilege Level 3 (PL3)-Task, die den Zeiger auf den Speicherbereich an eine PL0-Task schicken will.
	Die Modes können durch | miteinander verknüpft werden z.B. RM_AUTOFREE | RM_FLATALLOC	
PoolID	ID des Speicherpools, aus dem der Speicherbereich angefordert wird	
Size	Größe des Speicherbereichs	
ppMemory	Adresse des Zeigers auf einen Speicherbereich.	

Beschreibung

Der Aufruf allokiert einen Speicherbereich der Größe `Size` aus dem angegebenen Speicherbereich.

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
38	RM_INVALID_SIZE	`Size`=0 oder `Size` größer als Speicherpool
36	RM_INVALID_ID	Zur angegebenen ID existiert kein Speicherpool.
3	RM_OUT_OF_ MEMORY	Kein Speicherbereich in der angegebenen Größe verfügbar
4	RM_GOT_ TIMEOUT	In der angegebenen Zeit konnte kein entsprechender Speicherbereich allokiert werden

Siehe auch

RmAlloc (Seite 19)

RmCreateMemPool (Seite 38)

RmDeleteMemPool (Seite 54)

RmFree (Seite 62)

RmFreeAll (Seite 63)

RmGetSize (Seite 81)

RmReAlloc (Seite 126)

2.71 RmPauseTask

Funktion

Zeitintervall abwarten

Syntax

```
#include <rmapi.h>

int    RmPauseTask(ulong TimeValue);
```

Parameter

Parametername	Bedeutung	
TimeValue	Zeitdauer der Pause	
	0 ... ULONG_MAX	Zeitintervall in ms.
	Zur Zeitangabe können die Werte für Stunden, Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^{31} Millisekunden.	
	RM_HOUR(hour)	Wartet (hour) Stunden
	RM_MINUTE(min)	Wartet (min) Minuten
	RM_SECOND(sec)	Wartet (sec) Sekunden
	RM_MILLISECOND (ms)	Wartet (ms) Sekunden
	Die Zeitangaben dürfen nur ein Vielfaches des Systemtaktes (Standard 10ms) sein.	

Beschreibung

Mit `RmPauseTask` wird ein vorgegebenes Zeitintervall abgewartet. Das konfigurierte Minimal–Zeitintervall wird angenommen, wenn die Anzahl der Zeiteinheiten `TimeValue=0` ist.

Eine mit `RmPauseTask` unterbrochene Task wird durch `RmResumeTask` vorzeitig vom Zustand WARTEND in BEREIT überführt.

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt
-256	RM_TASK_ RESUMED	Die Task wurde durch `RmResumeTask` fortgesetzt

Siehe auch

Minimal–Zeitintervall

RmRestartTask (Seite 142)

RmResumeTask (Seite 146)

2.72 RmPciSearchFunction

Funktion

Scant den PCI-Bus (in RMPCISCA.LIB enthalten)

Syntax

```
#include <rmpci.h>

int    RmPciSearchFunction (
            ushort VendorID,
            ushort DeviceID,
            PCI_CONFIG_SPACE ConfigSpace[],
            ushort MaxNumOfFunctions,
            ushort *NumOfFunctions);
```

Parameter

Parametername	Bedeutung
VendorID	Vendor ID der Baugruppe am PCI-Bus
DeviceID	Device ID der Baugruppe am PCI-Bus
ConfigSpace	Array vom Typ PCI_CONFIG_SPACE
MaxNumOfFunctions	Max Anzahl von Einträgen `ConfigSpace`
NumOfFunctions	Anzahl von gefundenen Funktionen (Rückgabe-Paramter)

Beschreibung

Scant den gesamten PCI-Bus über Bridges hinaus auf alle Funktionen, die der ausgewählten `VendorID` und `DeviceID` entsprechen. Um alle Devices am Bus zu suchen kann als `VendorID = 0xFFFFH` übergeben werden.

In das übergebene Array `ConfigSpace` werden die Inhalte des(der) gefundenen Device(s) eingetragen.

Die Devices werden dabei nach PCI-Konvention gefunden. Für SICOMP-IMC bedeutet das eine Zählweise von links nach rechts.

Die Funktion bricht die Suche bei vollem `ConfigSpace` Array ab.

Wird für die ausgewählte `VendorID` und `DeviceID` kein Device am Bus gefunden, dann wird `NumOfFunctions = 0` und der Rückgabewert = RM_OK gesetzt.

Der ConfigSpace `PCI_CONFIG_SPACE` ist gemäß PCI-Konvention wie folgt definiert:

```
typedef struct _PCI_CONFIG_SPACE
{
        uchar  PciBusNum;      /* Nummer des Buses /
        uchar  PciDeviceNum;   /* Nummer des Devices /
        uchar  PciFunctionNum; /* Nummer der Funktion /
        ushort VendorId;       /* Hersteller ID der Baugruppe */
        ushort DeviceId;       /* Device ID der Baugruppe */
        ushort Cmdreg;
        ushort StatReg;
        uchar  RevisionId;
        uchar  ClassCode[3];
        uchar  CacheLineSize;
        uchar  LatencyTime;
        uchar  HeaderType;
        uchar  BIST;
        uint   BaseAdr0;
        uint   BaseAdr1;
        uint   BaseAdr2;
        uint   BaseAdr3;
        uint   BaseAdr4;
        uint   BaseAdr5;
        uint   CardbusCisPtr;
        ushort SubsysVendorId; /* Sub Vendor ID der Baugruppe*/
        ushort SubSysId;       /* Subsystem ID der Baugruppe */
        uint   ExpRomBaseAdr;
        uint   reserved[2];
        uchar  IntLine;        /* Interrupt – Nummer */
        uchar  IntPin;
        uchar  Min_Gnt;
        uchar  Max_Lat;
} PCI_CONFIG_SPACE;
```

Hinweis

Die Größe des Arrays `ConfigSpace` muss größer oder gleich von `MaxNumOfFunctions` sein.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
27	RM_BOUND_ REACHED	Anzahl von MaxNumOfFunctions erreicht

Beispiel

Siehe RmSetShIntISHandler2 (Seite 183)

Siehe auch

RmClrShIntISHandler (Seite 28)

RmInitShIntClient (Seite 104)

RmPciSearchSubFunction (Seite 120)

RmSetShIntISHandler1 (Seite 181)

RmSetShIntISHandler2 (Seite 183)

2.73 RmPciSearchSubFunction

Funktion

Scant den PCI-Bus unter Berücksichtigung von PCI-Subsystemen (in RMPCISCA.LIB enthalten)

Syntax

```
#include <rmpci.h>

int    RmPciSearchSubFunction (
                  ushort VendorID,
                  ushort DeviceID,
                  ushort SubVendorID,
                  ushort SubDeviceID,
                  PCI_CONFIG_SPACE ConfigSpace[],
                  ushort MaxNumOfFunctions,
                  ushort *NumOfFunctions);
```

Parameter

Parametername	Bedeutung
VendorID	Vendor ID der Baugruppe am PCI-Bus
DeviceID	Device ID der Baugruppe am PCI-Bus
SubVendorID	Subsystem Vendor ID
SubDeviceID	Subsystem ID
ConfigSpace	Array vom Typ PCI_CONFIG_SPACE
MaxNumOfFunctions	Max Anzahl von Einträgen `ConfigSpace`
NumOfFunctions	Anzahl von gefundenen Funktionen (Rückgabe-Paramter)

Beschreibung

Scant den gesamten PCI-Bus über Bridges hinaus auf alle Funktionen, die der ausgewählten `VendorID,DeviceID,SubVendorID` und `SubDeviceID` entsprechen. Um alle Devices am Bus zu suchen kann als `VendorID = 0xFFFFH` übergeben werden.

In das übergebene Array `ConfigSpace` werden die Inhalte des(der) gefundenen Device(s) eingetragen.

Die Devices werden dabei nach PCI-Konvention gefunden. Für SICOMP-IMC bedeutet das eine Zählweise von links nach rechts.

Die Funktion bricht die Suche bei vollem `ConfigSpace` Array ab.

Wird für die ausgewählte `VendorID` und `DeviceID` kein Device am Bus gefunden, dann wird `NumOfFunctions = 0` und der Rückgabewert = RM_OK gesetzt.

Der ConfigSpace `PCI_CONFIG_SPACE` ist gemäß PCI-Konvention wie folgt definiert:

```
typedef struct _PCI_CONFIG_SPACE
{
        uchar  PciBusNum;      /* Nummer des Buses /
        uchar  PciDeviceNum;   /* Nummer des Devices /
        uchar  PciFunctionNum; /* Nummer der Funktion /
        ushort VendorId;       /* Hersteller ID der Baugruppe */
        ushort DeviceId;       /* Device ID der Baugruppe */
        ushort Cmdreg;
        ushort StatReg;
        uchar  RevisionId;
        uchar  ClassCode[3];
        uchar  CacheLineSize;
        uchar  LatencyTime;
        uchar  HeaderType;
        uchar  BIST;
        uint   BaseAdr0;
        uint   BaseAdr1;
        uint   BaseAdr2;
        uint   BaseAdr3;
        uint   BaseAdr4;
        uint   BaseAdr5;
        uint   CardbusCisPtr;
        ushort SubsysVendorId; /* Sub Vendor ID der Baugruppe*/
        ushort SubSysId;       /* Subsystem ID der Baugruppe */
        uint   ExpRomBaseAdr;
        uint   reserved[2];
        uchar  IntLine;        /* Interrupt – Nummer */
        uchar  IntPin;
        uchar  Min_Gnt;
        uchar  Max_Lat;
 } PCI_CONFIG_SPACE;
```

Hinweis

Die Größe des Arrays `ConfigSpace` muss größer oder gleich von `MaxNumOfFunctions` sein.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
27	RM_BOUND_ REACHED	Anzahl von `MaxNumOfFunctions` erreicht

Beispiel

Siehe RmSetShIntISHandler2 (Seite 183)

Siehe auch

RmClrShIntISHandler (Seite 28)

RmInitShIntClient (Seite 104)

RmPciSearchFunction (Seite 118)

RmSetShIntISHandler1 (Seite 181)

RmSetShIntISHandler2 (Seite 183)

2.74 RmQueueStartTask

Funktion

Task–Start–Anforderung in Warteschlange einreihen. Die Task wird gestartet, sobald sie sich im Zustand DORMANT befindet.

Syntax

```
#include <rmapi.h>

int    RmQueueStartTask(
              uint Wait,
              uint TaskID,
              uint Priority,
              uint RegVal1,
              uint RegVal2);
```

Parameter

Parametername	Bedeutung	
Wait	RM_NO_WAIT (0)	Zieltask starten und Task fortsetzen
	RM_WAIT_READY (1)	Warten bis Zieltask im Zustand BEREIT ist.
	RM_WAIT_END (2)	Warten bis Zieltask beendet ist
TaskID	Zieltask–ID	
Priority	0..255	Vorgegebenen Wert einstellen
	RM_TCDPRI (0x400)	Priorität aus TCD entnehmen
	RM_CURPRI (0x300)	Aktuelle Priorität der aufrufenden Task verwenden
	RM_MAXPRI (0x100)	Maximum einstellen (RM_TCDPRI, RM_CURPRI)
RegVal1	Parameter 1 (Übergabe in eax der Zieltask)	
RegVal2	Parameter 2 (Übergabe in ebx der Zieltask)	

Beschreibung

Mit `RmQueueStartTask` wird eine Task gestartet. Der SVC benötigt dieselben Parameter wie `RmStartTask`. Der Unterschied zu `RmStartTask` besteht darin, dass `RmStartTask` die Zieltask nur dann startet, wenn sie sich im Zustand DORMANT befindet. Ist bei `RmQueueStartTask` die Zieltask nicht im Zustand DORMANT, wird die Anforderung in eine Warteschlange eingereiht.

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Eine ungültige `TaskID` wurde übergeben.
49	RM_TASK_KILLED	Die Zieltask wurde vor Erreichen des Zustands BEREIT oder vor ihrer Beendigung durch `RmKillTask` in den Zustand DORMANT versetzt bzw. gelöscht
35	RM_INVALID_TYPE	Ein ungültiger Parameter (Priority) wurde übergeben.

Siehe auch

Applikationshinweise, Task–Konfiguration

RmStartTask (Seite 190)

2.75 RmReadMessage

Funktion

Nachricht aus Message–Queue abholen

Syntax

```
#include <rmapi.h>

int    RmReadMessage (
             ulong TimeOutValue,
             uint * pMessage,
             void **pMessageParam);
```

Parameter

Parametername	Bedeutung	
TimeOutValue	Gibt an, wie lange bei leerer Message–Queue auf das Eintreffen einer Nachricht gewartet werden soll.	
	RM_CONTINUE (0)	Task fortsetzen, ohne Eintreffen der Message abzuwarten
	RM_WAIT (-1)	Eintreffen der Message abwarten
	0 ... ULONG_MAX	Zeitintervall in ms. Die Task wartet entweder bis die Message eingetroffen ist oder das Timeout abgelaufen ist.
	Zur Zeitangabe können die Werte für Stunden, Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^{31} Millisekunden.	
	RM_HOUR(hour)	Wartet (hour) Stunden
	RM_MINUTE(min)	Wartet (min) Minuten
	RM_SECOND(sec)	Wartet (sec) Sekunden
	RM_MILLISECOND(ms)	Wartet (ms) Millisekunden
	Die Zeitangaben dürfen nur ein Vielfaches des Systemtaktes (Standard 10ms) sein.	
pMessage	Zeiger auf Identifikationskennung der Nachricht.	
pMessageParam	Zeiger auf den Inhalt der Nachricht	

Beschreibung

Holt die Nachricht mit der höchsten Priorität aus der Message-Queue der aufrufenden Task. Der Speicherplatz für den Zeiger auf die Message-Parameter wird von der aufrufenden Task bereitgestellt. `RmReadMessage()` trägt in `pMessageParam` den Zeiger auf die eigentlichen Message-Parameter ein. Ist keine Nachricht vorhanden wird entsprechend dem Parameter `TimeOutValue` gewartet. Wird in dieser Zeit keine Nachricht empfangen erfolgt ein Abbruch mit Timeout.

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
4	RM_GOT_ TIMEOUT	In der angegebenen Zeit konnte keine Message empfangen werden
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.
43	RM_NO_MESSAGE	Die Message-Queue enthält keine Nachricht
58	RM_QUEUE_NOT_ EXIST	Die Message–Queue existiert nicht.

Siehe auch

RmCreateMessageQueue (Seite 40)

RmDeleteMessageQueue (Seite 55)

RmSendMessage (Seite 155)

2.76 RmReAlloc

Funktion

Größe eines Speicherbereichs verändern

Syntax

```
#include <rmapi.h>

int    RmReAlloc (
           ulong TimeOutValue,
           uint Mode,
           ulong NewSize,
           void **ppMemory);
```

Parameter

Parametername	Bedeutung	
TimeOutValue	Maximale Wartezeit bis zur Ausführung	
	RM_CONTINUE (0)	Task fortsetzen, ohne auf Zuteilung des Speichers zu warten.
	RM_WAIT (-1)	Auf Zuteilung des Speichers warten.
	0 ... ULONG_MAX	Zeitintervall in ms.
	Zur Zeitangabe können die Werte für Stunden, Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^{31} Millisekunden.	
	RM_HOUR(hour)	Wartet (hour) Stunden
	RM_MINUTE(min)	Wartet (min) Minuten
	RM_SECOND(sec)	Wartet (sec) Sekunden
	RM_MILLISECOND(ms)	Wartet (ms) Millisekunden
	Die Zeitangaben dürfen nur ein Vielfaches des Systemtaktes (Standard 10ms) sein.	
Mode	Allokierungsmethode für den Speicher:	
	RM_AUTOFREE (0x0)	Der Speicher wird automatisch mit `RmFreeAll()` freigegeben. Er ist taskspezifisch zugeordnet.
	RM_NOAUTOFREE (0x2)	Der Speicher wird nicht automatisch mit `RmFreeAll()` freigegeben.
	RM_FLATALLOC (0x0C)	Der Zeiger `ppMemory` enthält `FlatSelector:FlatOffset`. `FlatOffset` kann direkt als Pointer an Flat–Programme übergeben werden.
	Die Modes können durch | miteinander verknüpft werden z.B. RM_AUTOFREE | RM_FLATALLOC	
NewSize	Neue Größe des Speicherbereichs	
ppMemory	Adresse des Zeigers auf Speicherbereich.	

Beschreibung

Der Aufruf vergrößert oder verkleinert den mit `*ppMemory` angegebenen Speicherbereich, ohne dessen Inhalt zu ändern. Der Zeiger auf den angepaßten Speicherbereich wird in `ppMemory` zurückgeliefert. Dieser Zeiger muss nicht mit dem übergebenen Zeiger übereinstimmen, da der Speicherbereich unter Umständen verschoben wurde.

Ist der Speicherbereich `*ppMemory` aus einem Pool angefordert worden, wird bei `RmReAlloc()` ebenfalls dieser Pool verwendet.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.
38	RM_INVALID_SIZE	`Size`=0 oder `Size` größer als HEAP/Speicherpool
3	RM_OUT_OF_ MEMORY	Kein Speicherbereich in der angegebenen Größe verfügbar
4	RM_GOT_ TIMEOUT	In der angegebenen Zeit konnte kein entsprechender Speicherbereich allokiert werden

Siehe auch

RmAlloc (Seite 19)

RmCreateMemPool (Seite 38)

RmDeleteMemPool (Seite 54)

RmFree (Seite 62)

RmFreeAll (Seite 63)

RmGetSize (Seite 81)

RmMemPoolAlloc (Seite 115)

2.77 RmReceiveMail

Funktion

Botschaft aus lokaler Mailbox empfangen

Syntax

```
#include <rmapi.h>

int    RmReceiveMail(
                ulong TimeOutValue,
                uint MailboxID,
                void *pMail);
```

Parameter

Parametername	Bedeutung	
TimeOutValue	Maximale Wartezeit bis zur Ausführung	
	RM_CONTINUE (0)	Task fortsetzen, ohne Eintreffen der Botschaft abzuwarten
	RM_WAIT (-1)	Eintreffen der Botschaft abwarten
	0 ... ULONG_MAX	Zeitintervall in ms. Die Task wartet entweder bis die Botschaft eingetroffen ist oder das Timeout abgelaufen ist.
	Zur Zeitangabe können die Werte für Stunden, Minuten und Sekunden durch Addition verknüpft werden.	
	RM_HOUR(hour)	Wartet (hour) Stunden
	RM_MINUTE(min)	Wartet (min) Minuten
	RM_SECOND(sec)	Wartet (sec) Sekunden
	RM_MILLISECOND(ms)	Wartet (ms) Millisekunden
MailboxID	Mailbox–ID	
pMail	Zeiger auf 12 Byte–Puffer	

Beschreibung

`RmReceiveMailbox` kopiert eine 3–Wort–Botschaft aus einer lokalen Mailbox in einen Anwender–Puffer. Die Task kann warten, bis eine Botschaft vorliegt.

Hinweis

Normalerweise wird der Nachrichtenblock für die eigentliche Botschaft aus einem Speicherpool geholt und von der adressierten Task wieder an den Speicherpool zurückgegeben. Die Wortlänge beträgt 32 Bit.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Inhalt von `*pMail` enthält Botschaft

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Mailbox–ID ungültig
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.
43	RM_NO_MESSAGE	Die Mailbox enthält keine Nachricht (RM_CONTINUE).
4	RM_GOT_ TIMEOUT	Der SVC wurde nach dem eingestelltem Timeout abgebrochen.

Siehe auch

RmCreateMailbox (Seite 37)

RmDeleteMailbox (Seite 53)

RmSendMail (Seite 150)

RmSendMailCancel (Seite 152)

RmSendMailDelayed (Seite 153)

2.78 RmReleaseBinSemaphore

Funktion

Semaphor rücksetzen

Syntax

```
#include <rmapi.h>

int    RmReleaseBinSemaphore(uint SemaphoreID);
```

Parameter

Parametername	Bedeutung
SemaphoreID	Semaphor–ID

Beschreibung

`RmReleaseBinSemaphore` setzt das Semaphor `SemaphoreID` zurück.

Hinweis

Die Belegung und Freigabe von Semaphoren ist nicht taskspezifisch.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Eine ungültige SemphoreID wurde übergeben.

Siehe auch

Automatische Prioritätsänderung durch Semaphor–Besitz im Benutzerhandbuch

RmCreateBinSemaphore (Seite 29)

RmDeleteBinSemaphore (Seite 50)

RmGetBinSemaphore (Seite 66)

2.79 RmReleaseSpinLock

Funktion

Freigeben eines SpinLocks

Syntax

```
#include <misc86.h>

int    RmReleaseSpinLock( RmSpinLockStruct *SpinLock);
```

Parameter

Parametername	Bedeutung

SpinLock	Adresse einer Struktur vom Typ `RmSpinLockStruct`, die folgendermaßen definiert ist:
	
	`struct _RmSpinLockStruct`
	`{`
	`  uchar Spin;`
	`  uchar CoreID;`
	`  uchar Reserved;`
	`  uchar Count;`
	`  uint Interruptflags;`
	`} GNU_PACKED;`
	`typedef struct _RmSpinLockStruct RmSpinLockStruct;`

Beschreibung

Mit dem Aufruf `RmReleaseSpinLock` wird das SpinLock wieder freigegeben. Nach dem Aufruf sollte der Anwender die Interrupts wieder freigegeben.

Da die SpinLock-Aufrufe keine Betriebssystemaufrufe sind, werden die entsprechenden Aufrufe auch nicht vom Profiler aufgezeichnet.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
42	RM_INVALID_ POINTER	Der Zeiger `SpinLock` ist ein NULL-Zeiger.
1	RM_SYSTEMERROR	SpinLock wurde nicht initialisiert

Beispiel

```
static RmSpinLockStruct lock;

 …
 // initialize spinlock
 RmInitSpinLock (&lock);
 // call spinlock
 disable();
 RmGetSpinLock (&lock);
 …
 RmReleaseSpinLock (&lock);
 enable();
```

Siehe auch

RmGetSpinLock (Seite 82)

RmGetSpinLockIRQ (Seite 84)

RmInitSpinLock (Seite 105)

2.80 RmReleaseSpinLockIRQ

Funktion

Freigeben eines SpinLocks mit Freigeben der Interrupts

Syntax

```
#include <misc86.h>

int    RmReleaseSpinLockIRQ( RmSpinLockStruct *SpinLock);
```

Parameter

Parametername	Bedeutung

SpinLock	Adresse einer Struktur vom Typ `RmSpinLockStruct`, die folgendermaßen definiert ist:
	
	`struct _RmSpinLockStruct`
	`{`
	`  uchar Spin;`
	`  uchar CoreID;`
	`  uchar Reserved;`
	`  uchar Count;`
	`  uint Interruptflags;`
	`} GNU_PACKED;`
	`typedef struct _RmSpinLockStruct RmSpinLockStruct;`

Beschreibung

Mit dem Aufruf `RmReleaseSpinLockIRQ` wird das SpinLock wieder freigegeben. Falls vorher die Interrupts freigegeben waren, werden sie mit dem Aufruf `RmReleaseSpinLockIRQ` auch wieder freigegeben.

Da die SpinLock-Aufrufe keine Betriebssystemaufrufe sind, werden die entsprechenden Aufrufe auch nicht vom Profiler aufgezeichnet.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
42	RM_INVALID_ POINTER	Der Zeiger `SpinLock` ist ein NULL-Zeiger.
1	RM_SYSTEMERROR	SpinLock wurde nicht initialisiert

Beispiel

```
static RmSpinLockStruct lock;
...
    // initialize spinlock
    RmInitSpinLock (&lock);
    // call spinlock
    RmGetSpinLockIRQ (&lock);
    …
    RmReleaseSpinLockIRQ (&lock);
```

Siehe auch

RmGetSpinLockIRQ (Seite 84)

RmInitSpinLock (Seite 105)

RmReleaseSpinLock (Seite 131)

2.81 RmRemoveFastTask

Funktion

Eine Task aus dem schnellen Zyklus entfernen.

Syntax

```
#include <fasttick.h>

int    RmRemoveFastTask (uint TaskID);
```

Parameter

Parametername	Bedeutung
TaskID	Task-ID derjenigen Task, die aus dem schnellen Zyklus entfernt wird

Beschreibung

Die Funktion `RmRemoveFastTask` entfernt die Task mit `TaskID` aus dem schnellen Zyklus. Die Task befindet sich dann im Zustand DORMANT.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt
-1		Ungültige TaskID

Beispiel

Siehe RmInitFastTask (Seite 99)

Siehe auch

RmInitFastTask (Seite 99)

2.82 RmRemoveFastTick

Funktion

Löschen des schnellen Zyklus.

Syntax

```
#include <fasttick.h>

int RmRemoveFastTick (void);
```

Beschreibung

Die Funktion `RmRemoveFastTick` löscht den schnellen Zyklus und stellt den ursprünglichen Timerinterrupthandler wieder her.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt
-1		Es befinden sich noch Tasks im schnellen Zyklus. Diese müssen erst beendet werden.

Beispiel

Siehe RmInitFastTask (Seite 99)

Siehe auch

RmInitFastTick (Seite 101)

2.83 RmReserveInterrupt

Funktion

Interrupt reservieren

Syntax

```
#include <rmapi.h>

int    RmReserveInterrupt (uint IntNum);
```

Parameter

Parametername	Bedeutung
IntNum	Interrupt–Nummer (0-255) IRQx (x=0 bis 63) HW-Interrupt IRQ(n) (n=0 bis 63) HW-Interrupt Auf PC-Hardware sind die HW-Interrupts von 0 bis 15 (im APIC-Modus bis 23).

Beschreibung

Mit diesem Aufruf wird ein Eintrag in der IDT (RMOS3) reserviert und später nicht mit dem Unexpected Interrupt Handler (für unerwartete Eingabe) vorbesetzt. Die Reservierung ist z.B. für die Breakpoint–Interrupts des Debuggers notwendig, damit in der Root–Task (bzw. während der Hochlaufphase) von einem bereits installierten Debugger (z.B. RMOS3–Debugger im Standalone–Modus) Breakpoints gesetzt werden können.

Im laufenden Betrieb ist dieser SVC wirkungslos.

Hinweis

Diese Funktion ist nicht im Speichermodell Flat verwendbar.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
41	RM_INVALID_ IRQ_NUMBER	IRQx ist ungültig, PIC nicht definiert.
56	RM_INVALID_ INTERRUPT_ NUMBER	Ungültige Interrupt–Nummer.

Siehe auch

RmGetIntHandler (Seite 73)
RmSetIntDefHandler (Seite 164)
RmSetIntDIHandler (Seite 166)
RmSetIntISHandler (Seite 168)
RmSetIntMailboxHandler (Seite 170)
RmSetIntTaskHandler (Seite 172)

2.84 RmResetFlag

Funktion

Ereignisflag rücksetzen

Syntax

```
#include <rmapi.h>

int    RmResetFlag(
         uint FlagGrpID,
         uint FlagMasks);
```

Parameter

Parametername	Bedeutung
FlagGrpID	Ereignisflag–Gruppe–ID
FalgMask	Maske legt fest, welche Bits rückgesetzt werden

Beschreibung

`RmResetFlag` setzt ein Ereignisflag zurück und meldet, ob es zuvor gesetzt war.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	SVC erfolgreich ausgeführt, kein Bit zurückgesetzt.
-260	RM_FLAG_RESET	Es wurde mindestens ein Bit zurückgesetzt.

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Eine ungültige `FlagGrpID` wurde übergeben.

Siehe auch

RmCreateFlagGrp (Seite 36)
RmDeleteFlagGrp (Seite 52)
RmGetFlag (Seite 70)
RmResetLocalFlag (Seite 139)
RmSetFlag (Seite 159)
RmSetFlagDelayed (Seite 160)

2.85 RmResetLocalFlag

Funktion

Lokales Ereignisflag zurücksetzen

Syntax

```
#include <rmapi.h>

int    RmResetLocalFlag(
          uint TaskID,
          uint FlagMask);
```

Parameter

Parametername	Bedeutung
TaskID	Task–ID (RM_OWN_TASK (-1) = eigene Task)
FlagMask	Maske legt fest, welche Flagbits zurückgesetzt werden sollen.

Beschreibung

`RmResetLocalFlag` setzt das lokale Ereignisflag der mit `TaskID` angegebenen Task zurück.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt
-260	RM_FLAG_RESET	Es wurde mindestens ein Bit zurückgesetzt

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	TaskID ungültig

Siehe auch

RmGetFlag (Seite 70)
RmResetFlag (Seite 138)
RmSetLocalFlag (Seite 174)

2.86 RmResetOS

Funktion

Konfigurierung des Betriebssystems starten

Syntax

```
#include <rmapi.h>

void    RmResetOS (
          uint Mode,
          ulong SystemBAddress,
          ulong RAMEndAddress,
          ulong LinPageDir,
          ulong PDFreeIndex,
          ulong ReadOnly,
          ulong LinOffset);
```

Parameter

Parametername	Bedeutung
Mode	BIT 0 = 0: NPX–Test durchführen BIT 0 = 1:kein NPX–Test, NPX–Bit wird gemäß BIT 1 gesetzt Wenn kein Test durchgeführt wird, kann mit Bit 1 weitere NPX-Angaben gemacht werden. BIT 1 = 0: NPX vorhanden BIT 1 = 1:Kein NPX vorhanden BIT 2 = 0: Unexpected Interrupthandler installieren BIT 2 = 1:Keinen Unexpected Interrupthandler installieren
SystemBAddress	Absolute 32–Bit–Adresse des System Control Block
RAMEndAddress	Absolute 32–Bit–Endadresse +1 des HEAP
LinPageDir	Beginn des Page Directories
PDFreeIndex	Oberer Page Directory Index zum Suchen von freien Stack-Einträgen
ReadOnly	Reserviert
LinOffset	Offset der physikalischen Startadresse

Beschreibung

Die Konfigurierung des Betriebssystems beginnt damit, dass der HEAP–Pool erzeugt wird. Dieser Aufruf kann nicht mehrfach aufgerufen werden und wird bereits vor `RmosStartupEntry()` in der Hochlaufphase aufgerufen. Der Parameter `RAMEndAddress` wird mit `RcRamSize()` in der Datei RMCONF.C ermittelt. Der Aufruf `RmResetOS()` teilt dem Nukleus die Länge des verfügbaren RAM mit, in dem der HEAP des RMOS3–Systems angelegt wird. Die Anfangsadresse des HEAP wird durch das Segment POOL_HEAP_DATA festgelegt, das nach dem Datensegment des Nucleus liegen muss. Der Nukleus–Stack wird installiert. Das Nukleus–Datensegment enthält zu diesem Zeitpunkt noch nicht alle zur Laufzeit notwendigen Datenbereiche. Diese werden während der Konfiguration dem freien Speicher entnommen und damit das Datensegment des Nukleus vergrößert. Der verbleibende freie Speicher steht für den HEAP zur Verfügung. Es ist darauf zu achten, dass der Bereich zwischen Start– (POOL_HEAP_DATA) und Endadresse des HEAP auf jeden Fall frei ist. (Dies muss im Builder–Lauf berücksichtigt werden). Der System Control Block enthält wichtige Betriebsystemkenngrößen, z.B. Bootlaufwerk, RMOS3– Systemtakt. Wird die Adresse eines System Control Blocks angegeben, werden folgende Einträge übernommen: Statname, Bootname, Bootmode und Bootdrive. Alle anderen Werte werden mit Defaultwerten bzw. mit den gültigen Systemwerten belegt. Danach springt die Funktion die RMOS3–Hochlaufroutine `RmosStartupEntry()` an. Die Exception–Interrupts des Prozessors werden abhängig vom Prozessortyp automatisch installiert.

Der Parameter `Mode` legt fest, ob die NPX–Checkfunktion ausgeführt werden soll (BIT 0=0) oder nicht (BIT 0=1). Wird kein Test ausgeführt, kann mit BIT 1 festgelegt werden ob eine NPX vorhanden ist (BIT 1=1) oder nicht (BIT1=0). Mit Bit 2 wird festgelegt, ob ein Unexpected Interrupthandler installiert werden soll (Bit2=0) oder nicht (Bit2=1).

Hinweis

Diese Funktion ist nicht im Speichermodell Flat verwendbar.

Siehe auch

Struktur RmSysB im Referenzhandbuch Teil II, Kapitel 3

2.87 RmRestartTask

Funktion

Task beenden und nach Zeitintervall erneut starten

Syntax

```
#include <rmapi.h>

int    RmRestartTask(
         uint Mode,
         ulong TimeValue);
```

Parameter

Parametername	Bedeutung	
Mode	RM_LAST_READY_ TIME (0)	Zeitrechnung auf letzte Überführung in den Zustand BEREIT beziehen
	RM_CURRENT_TIME (1)	Zeitrechnung auf aktuelle Zeit beziehen
TimeValue	Wartezeit bis zum erneuten Start	
	0... ULONG_MAX	Zeitintervall in ms.
	Zur Zeitangabe können die Werte für Stunden, Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^{31} Millisekunden.	
	RM_HOUR(hour)	Wartet (hour) Stunden
	RM_MINUTE(min)	Wartet (min) Minuten
	RM_SECOND(sec)	Wartet (sec) Sekunden
	RM_MILLISECOND(ms)	Wartet (ms) Sekunden
	Die Zeitangaben dürfen nur ein Vielfaches des Systemtaktes (Standard 10ms) sein.	

Beschreibung

`RmRestartTask` beendet die Task–Ausführung und startet sie erneut nach Ablauf eines Zeitintervalls.

Wenn `TimeValue=0` , wird die Task mit dem nächsten Timer–Interrupt in den Zustand BEREIT überführt.

Hinweis

`RmRestartTask` überführt eine Task in den Zustand WARTEND und nicht in den Zustand DORMANT. Im Unterschied zu `RmPauseTask` wird die Task nach Ablauf der in `RmRestartTask` festgelegten Zeit gestartet.

Eine durch `RmRestartTask` unterbrochene Task kann nur durch Ablauf des Zeitintervalls in den Zustand BEREIT überführt werden. Mit `RmQueueStartTask` wird dagegen eine weitere Task–Start–Anforderung in eine Warteschlange eingereiht.

Wurde eine Task (`main()`) vom CLI gestartet, darf sie mit `RmRestartTask` nicht erneut gestartet werden.

Hinweis

Bei einer mit `RmRestartTask` gestarteten Task ist keine Parameterübergabe in EAX, EBX möglich. Die Parameter können beim ersten Start der Task (mit einem anderen Startbefehl) übergeben und zwischengespeichert werden. Dadurch können diese Parameter nach jedem durch `RmRestartTask`erfolgten Start der Task wieder verwendet werden.

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Siehe auch

Starten, Unterbrechen, Beenden von Tasks

RmPauseTask (Seite 117)

RmResumeTask (Seite 146)

2.88 RmRestoreIntHandler

Funktion

DI–Interrupthandler restaurieren

Syntax

```
#include <rmapi.h>

int    RmRestoreIntHandler (
                uint IntNum,
                rmfarproc DIHandlerEntry);
```

Parameter

Parametername	Bedeutung
IntNum	Interrupt–Nummer (0-255) IRQx (x=0 bis 63) HW-Interrupt IRQ(n) (n=0 bis 63) HW-Interrupt Auf PC-Hardware sind die HW-Interrupts von 0 bis 15 (im APIC-Modus bis 23).
DIHandlerEntry	Einsprungadresse des Interrupthandlers.

Beschreibung

Der Aufruf restauriert einen DI–Interrupthandler, der mit `RmGetIntHandler` gelesen wurde. Unter GNU kann hierfür nicht der Aufruf `RmSetIntDIHandler` verwendet werden.

Während der neue Interrupthandler gesetzt wird, darf kein Interrupt für diesen Handler auftreten.

Die Interrupt–Nummer entspricht dem zugehörigen Vektor in der Interrupt–Tabelle Selektor des betreffenden Deskriptors in der IDT.

Mit `RmRestoreIntHandler` wird ein Interrupt–Gate in die IDT eingetragen.

Der in `DIHandlerEntry` angegebene Interrupthandler soll direkt nach einem Interrupt im DI–Zustand angesprungen werden. Der DI-Interrupthandler muss selbst ein EOI auslösen.

Bei einem HW-Interrupt muss das entsprechende EOI-Kommando zum PIC im Interrupthandler selbst erzeugt werrden.

Hinweis

Wird der SVC nicht erfolgreich ausgeführt, bleibt der bisherige Interrupthandler aktiv.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Siehe auch

RmSetIntDIHandler (Seite 166)

2.89 RmResumeDriver

Funktion

Neuen Gerätetreiber definieren

Syntax

```
#include <rmapi.h>

void    RmResumeDriver(uint DriverID);
```

Parameter

Parametername	Bedeutung
DriverID	Treiber-ID

Beschreibung

`RmResumeDriver` überführt einen Gerätetreiber vom Zustand SUSPENDIERT (WARTEND) in BEREIT. Dieser Aufruf wird z. B. benutzt, wenn eine Festplatte erst nach dem Boot-Vorgang ihre Umdrehungszahl erreicht. Der Treiber wird dann nicht beim Boot-Vorgang sondern zu einem späteren Zeitpunkt durch `RmResumeDriver` aktiviert.

Hinweis

Die Initialisierung des Geräte-Treibers wird während des Aufrufs vorgenommen, d.h. sämtliche Interrupts sind zu diesem Zeitpunkt gesperrt. Der Treiber muss in der Softwarekonfigurierung bereits konfiguriert sein.

Diese Funktion ist nicht im Speichermodell Flat verwendbar.

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Eine ungültige `DriverID` wurde übergeben.

Siehe auch

Systemkonfigurierung

RmIO (Seite 108)

RmSuspendDriver (Seite 192)

2.90 RmResumeTask

Funktion

Ablauf eines mit `RmPauseTask` oder `RmSuspendTask` eingeleiteten Intervalls beenden

Syntax

```
#include <rmapi.h>

int    RmResumeTask(uint TaskID);
```

Parameter

Parametername	Bedeutung
TaskID	Task–ID

Beschreibung

`RmResumeTask` überführt eine Task, die durch `RmPauseTask` in den Zustand WARTEND versetzt wurde, in den Zustand BEREIT.

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Eine ungültige `TaskID` wurde übergeben.
26	RM_TASK_NOT_ PAUSED	Die mittels `RmResumeTask` fortzusetzende Task wurde nicht durch `RmPauseTask` angehalten oder ist nicht mehr im Zustand WARTEND.

Siehe auch

RmPauseTask (Seite 117)

RmRestartTask (Seite 142)

RmSuspendTask (Seite 193)

2.91 RmRunOS

Funktion

Alle Units der Treiber initialisieren und Init-Task starten

Syntax

```
#include <rmapi.h>

int    RmRunOS (char *pTaskName);
```

Parameter

Parametername	Bedeutung
pTaskName	Zeiger auf einen C–String, der den Namen enthält, mit dem die Init–Task katalogisiert wurde.

Beschreibung

Der Aufruf darf nur einmal aufgerufen werden und initialisert alle Treiber–Units.Weiterhin werden alle bisher nicht benutzten Interrupts mit dem Unexpected Interrupthandler für unerwartete Eingabe vorbesetzt (abhängig von Bit2 im `Mode` Parameter, siehe `RmResetOS`) und die lokale Uhrzeit (Datum) mit der globalen Zeit vorbesetzt. Anschließend wird das Betriebssystem (Scheduler) gestartet und das Interruptsystem freigegeben. Die Konfiguration des Betriebssystems ist abgeschlossen. Ab jetzt laufen alle SVCs ohne Einschränkungen. Die Root–Task ist beendet. Mit dem übergebenem Task–Namen beim Aufruf von `RmRunOS()` wird eine bestimmte Task als Init–Task (Initialisierungstask) gestartet. Kann die Task nicht im Katalog gefunden werden, wird mit dem entsprechenden Returncode zurückgekehrt.

Das Modul, welches den Code für den System–Hochlauf enthält, wird STARTUP–Modul (RMCONF.C) genannt. Der wesentliche Unterschied ist, dass die Init–Task im Gegensatz zu `RmosStartupEntry()` eine normale RMOS3–Task ist.

Hinweis

Diese Funktion ist nicht im Speichermodell Flat verwendbar.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Im OK-Fall kehrt die Funktion nicht zurück.

Fehlercodes

ID dez	Fehlercode	Bedeutung
52	RM_INVALID_ CFG_STATE	`RmRunOS` erfolgte nicht nach `RmInitOS()`
37	RM_INVALID_ STRING	Die Länge des Strings ist unzulässig.
48	RM_IS_NOT_ CATALOGED	Der String ist nicht katalogisiert.
42	RM_INVALID_ POINTER	Der Zeiger auf den String ist falsch, bzw. würde eine Schutzverletzung auslösen.

Siehe auch

RmInitOS (Seite 106)

2.92 RmSegToFlatAddress

Funktion

Zeiger des segmentierten Modells in Flat-Modell umwandeln

Syntax

```
#include <rmapi.h>

int    RmSegToFlatAddress (
              ushort Sel,
              ulong Offset,
              ulong *pFlatAddress);
```

Parameter

Parametername	Bedeutung
Sel	Selector des zu konvertierenden Zeigers
Offset	Offset des zu konvertierenden Zeigers
pFlatAddress	Adresse des Flat-Zeigers

Beschreibung

Der Aufruf konvertiert einen Pointer vom segmentierten Speichermodell in das FLAT-Speichermodell.

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Im OK-Fall kehrt die Funktion nicht zurück.

Fehlercodes

ID dez	Fehlercode	Bedeutung
39	RM_INVALID_ OFFSET	Ungültiger Offset in `pPointer`
42	RM_INVALID_ POINTER	Der Zeiger auf den String ist falsch, bzw. würde eine Schutzverletzung auslösen.
21	RM_INVALID_ SELECTOR	Sel ungültig.

Siehe auch

RmFlatToSegAddress (Seite 61)

2.93 RmSendMail

Funktion

Botschaft an eine lokale Mailbox senden

Syntax

```
#include <rmapi.h>

int    RmSendMail(
         ulong TimeOutValue,
         uint Priority,
         uint MailboxID,
         void *pMail);
```

Parameter

Parametername	Bedeutung	
TimeOutValue	Maximale Wartezeit bis zur Ausführung	
	RM_CONTINUE (0)	Task fortsetzen, ohne Abholen der Botschaft abzuwarten
	RM_WAIT (-1)	Abholen der Botschaft abwarten
	0 ... ULONG_MAX	Zeitinterval in ms. Die Task wartet entweder bis die Botschaft abgeholt wurde oder das Timeout abgelaufen ist.
	Zur Zeitangabe können die Werte für Stunden, Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^{31} Millisekunden.	
	RM_HOUR(hour)	Wartet (hour) Stunden
	RM_MINUTE(min)	Wartet (min) Minuten
	RM_SECOND(sec)	Wartet (sec) Sekunden
	RM_MILLISECOND(ms)	Wartet (ms) Millisekunden
	Die Zeitangaben dürfen nur ein Vielfaches des Systemtaktes (Standard 10ms) sein.	
Priority	0..255	Vorgegebenen Wert einstellen
	RM_TCDPRI (0x400)	Priorität aus TCD entnehmen
	RM_CURPRI (0x300)	Aktuelle Priorität der aufrufenden Task verwenden
	RM_MAXPRI (0x100)	Maximum (RM_TCDPRI, RM_CURPRI) einstellen
MailboxID	Lokale Mailbox–ID	
pMail	Zeiger auf 3–Wort–Puffer	

Beschreibung

`RmSendMail` kopiert eine 3 Wort lange, prioritätsbehaftete Botschaft in eine lokale Mailbox. Die Task kann in den Zustand WARTEND versetzt werden, bis die Botschaft abgeholt oder der SVC nach Ablauf eines Timeout abgebrochen wird. In diesem Fall (RM_GOT_TIMEOUT) wird die entsprechende Botschaft automatisch aus der Mailbox entfernt.

Das Botschaftsformat ist frei wählbar. Es kann beispielsweise eine 3 Wort lange Botschaft oder die Adresse und Länge einer Botschaft mit folgendem Schema angegeben werden:

Botschaftswort 1 : Offsetadresse des Nachrichtenblocks
Botschaftswort 2 : Segmentadresse des Nachrichtenblocks
Botschaftswort 3 : Länge des Nachrichtenblocks in Bytes

Hinweis

Die Wortlänge beträgt 32 Bit (4 Byte).

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt.

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Eine ungültige `MailboxID` wurde übergeben.
35	RM_INVALID_TYPE	Ein ungültiger Parameter (`Priority`) wurde übergeben.
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.
4	RM_GOT_ TIMEOUT	Der SVC wurde nach dem eingestellten Timeout abgebrochen.
27	RM_BOUND_ REACHED	Die Anforderung überschreitet die eingetragene Grenze der Mailbox (siehe `RmSetMailboxSize`).

Siehe auch

RmCreateMailbox (Seite 37)

RmDeleteMailbox (Seite 53)

RmReceiveMail (Seite 128)

RmSetMailboxSize (Seite 175)

2.94 RmSendMailCancel

Funktion

Abbruch eines mit `RmSendMailDelayed` gestarteten Timers

Syntax

```
#include <rmapi.h>

int    RmSendMailCancel (
             RmMailIDStruct *pMailID,
             void *pMail);
```

Parameter

Parametername	Bedeutung
pMailID	Zeiger auf eine Struktur vom Typ `RmMailIDStruct`
pMail	Zeiger auf Puffer, in den die vorher verschickte Botschaft zurückgeschrieben wird. Die Länge der Botschaft beträgt 12 Byte.

Beschreibung

Der Aufruf bricht einen durch `RmSendMailDelayed` gestarteten Timer ab. Der Abbruch ist nur möglich, solange das Zeitintervall noch nicht abgelaufen ist oder die betreffende Botschaft noch nicht abgeholt wurde. Im letzteren Fall wird die Botschaft außerdem aus der Mailbox entfernt.

Die aufrufende Task erhält den Inhalt der Botschaft zurück, um eventuell darin enthaltene Informationen auswerten zu können.

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Ein ungültiger Timer wurde in `pMailID` übergeben. Trifft auch zu, wenn eine verschickte Botschaft bereits abgeholt wurde. Der durch `pMailID` definierte Speicher ist undefiniert.
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.

Siehe auch

RmCreateMailbox (Seite 37)

RmDeleteMailbox (Seite 53)

RmReceiveMail (Seite 128)

RmSendMail (Seite 150)

RmSendMailDelayed (Seite 153)

RmSetMailboxSize (Seite 175)

2.95 RmSendMailDelayed

Funktion

Botschaft zeitverzögert an eine Mailbox schicken

Syntax

```
#include <rmapi.h>

int    RmSendMailDelayed (
            ulong TimeValue,
            uint Priority,
            uint MailboxID,
            void *pMail,
            RmMailIDStruct *pMailID);
```

Parameter

Parametername	Bedeutung	
TimeValue	Zeit bis Botschaft abgeschickt wird	
	0... ULONG_MAX	Zeitintervall in ms.
	Zur Zeitangabe können die Werte für Stunden, Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^{31} Millisekunden.	
	RM_HOUR(hour)	Wartet (hour) Stunden
	RM_MINUTE(min)	Wartet (min) Minuten
	RM_SECOND(sec)	Wartet (sec) Sekunden
	RM_MILLISECOND(ms)	Wartet (ms) Sekunden
	Die Zeitangaben dürfen nur ein Vielfaches des Systemtaktes (Standard 10ms) sein.	
Priority	0..255	Vorgegebenen Wert einstellen
	RM_TCDPRI (0x400)	Priorität aus TCD entnehmen
	RM_CURPRI (0x300)	Aktuelle Priorität der aufrufenden Task verwenden
	RM_MAXPRI (0x100)	Maximum (RM_TCDPRI, RM_CURPRI) einstellen
MailboxID	Lokale Mailbox–ID	
pMail	Zeiger auf Botschaft. Die Länge der Botschaft beträgt 12 Byte.	
pMailID	Zeiger auf eine Struktur vom Typ `RmMailIDStruct`	

Beschreibung

Mit `RmSendMailDelayed` wird eine Botschaft zeitverzögert an eine Mailbox gesendet werden. Die aufrufende Task erhält eine Identifikation, mit der ggfs. die Aktion mittels `RmSendMailCancel` abgebrochen werden kann.

Hinweis

Ein durch `RmSetMailboxSize` festgelegter Grenzwert von Botschaften, die in einer Mailbox auf Abholung warten, wird beim Verschicken der Botschaft mit `RmSendMailDelayed` nicht berücksichtigt.

Es ist prinzipiell möglich, dass die dynamische Mailbox, an welche die Botschaft geschickt werden soll, vor Ablauf des Zeitintervalls bereits durch den Systemaufruf `RmDeleteMailbox` gelöscht wurde. In diesem Fall wird die Botschaft stillschweigend verworfen.

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
35	RM_INVALID_TYPE	Ein ungültiger Parameter (`Priority`) wurde übergeben.
36	RM_INVALID_ID	Ungültige Flag-Gruppe
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.

Siehe auch

RmCreateMailbox (Seite 37)

RmDeleteMailbox (Seite 53)

RmReceiveMail (Seite 128)

RmSendMail (Seite 150)

RmSendMailCancel (Seite 152)

RmSetMailboxSize (Seite 175)

2.96 RmSendMessage

Funktion

Nachricht in Message–Queue einhängen

Syntax

```
#include <rmapi.h>

int    RmSendMessage (
             ulong TimeOutValue,
             uint Priority,
             uint TaskID,
             uint Message,
             void *pMessageParam);
```

Parameter

Parametername	Bedeutung	
TimeOutValue	Gibt an wie lange auf das Abholen der Nachricht gewartet werden soll.	
	RM_CONTINUE (0)	Task fortsetzen, ohne Abholen der Nachricht abzuwarten
	RM_WAIT (-1)	Abholen der Nachricht abwarten
	0 ... ULONG_MAX	Zeitintervall in ms. Die Task wartet entweder bis die Nachricht abgeholt wird oder das Timeout abgelaufen ist.
	Zur Zeitangabe können die Werte für Stunden, Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^{31} Millisekunden.	
	RM_HOUR(hour)	Wartet (hour) Stunden
	RM_MINUTE(min)	Wartet (min) Minuten
	RM_SECOND(sec)	Wartet (sec) Sekunden
	RM_MILLISECOND(ms)	Wartet (ms) Millisekunden
	Die Zeitangaben dürfen nur ein Vielfaches des Systemtaktes (Standard 10ms) sein.	
Priority	Priorität der Nachricht	
TaskID	Zieltask–ID	
Message	Identifikationskennung der Nachricht.	
pMessageParam	Zeiger auf den Inhalt der Nachricht.	

Beschreibung

Der Aufruf hängt die Nachricht `Message` zusammen mit dem Zeiger auf die Message-Parameter und der vorgegebenen Priorität in die entsprechende Stelle der Message-Queue der Task `TaskID`. Der Parameter `TimeOutValue` gibt an, ob und wie lange auf das Abholen der Nachricht gewartet werden soll.

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Task–ID ungültig
42	RM_INVALID_ POINTER	Ungültiger Zeiger
35	RM_INVALID_TYPE	Ein ungültiger Parameter (`Priority`) wurde übergeben.
58	RM_QUEUE_NOT_ EXIST	Die Message–Queue existiert nicht.
27	RM_BOUND_ REACHED	Die Message–Queue ist voll.

Siehe auch

RmCreateMessageQueue (Seite 40)

RmDeleteMessageQueue (Seite 55)

RmReadMessage (Seite 124)

2.97 RmSetDeviceHandler

Funktion

Treiber–Interrupthandler erzeugen

Syntax

```
#include <rmapi.h>

int    RmSetDeviceHandler (
            uint IntNum,
            char *pDeviceName,
            char *pUnitName,
            rmfarproc HandlerEntry);
```

Parameter

Parametername	Bedeutung
IntNum	Interrupt–Nummer (0-255) IRQx (x=0 bis 63) HW-Interrupt IRQ(n) (n=0bis63) HW-Interrupt Auf PC-Hardware sind die HW-Interrupts von 0 bis 15 (im APIC-Modus bis 23).
pDeviceName	Zeiger auf einen C–String, der den Namen enthält, mit dem der Treiber katalogisiert wurde.
pUnitName	Zeiger auf einen C–String, der den Namen enthält, mit dem die Unit katalogisiert wurde.
HandlerEntry	Anfangsadresse des Interrupthandlers

Beschreibung

Mit diesem Aufruf wird ein Treiber–Interrupthandler erzeugt.

Der benötigte Speicher wird aus dem HEAP angefordert. Der Handler kann mit `RmSetIntDefHandler()` wieder freigegeben werden. Da die neuen Treiber ihre Handler selbst installieren, ist dieser Aufruf nur für alte Treiber notwendig.

Bei Übergabe einer IRQ–Nummer wird der entprechende Interrupt auf dem zugehörigen Interruptcontroller freigegeben. Dazu wird die Funktion `X_SYSB.SET_PIC_MASK(IRQxx)` aufgerufen.

Handelt es sich hierbei um einen HW–Interrupt z.B. IRQ1, wird dieser automatisch maskiert. Zum automatischen Maskieren muss jedoch mit `RmInitOS` die Funktion `SetPICMaskProc` installiert worden sein.

Alle anderen Interrupts sind vom Anwender selbst zu maskieren.

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
3	RM_OUT_OF_ MEMORY	Nicht genügend Speicher verfügbar um Interrupthandler zu installieren.
21	RM_INVALID_ SELECTOR	`HandlerEntry` hat keinen Codeselector
36	RM_INVALID_ID	Device–ID oder Unit–ID ungültig.
37	RM_INVALID_ STRING	Die Länge von `pDevicename` oder `pUnitname` ist unzulässig.
41	RM_INVALID_ IRQ_NUMBER	IRQx ist ungültig, PIC nicht definiert.
42	RM_INVALID_ POINTER	Der Zeiger auf `pDevicename` oder `pUnitname` ist falsch, bzw. würde eine Schutzverletzung auslösen oder `HandlerEntry` ist ungültig.
48	RM_IS_NOT_ CATALOGED	`pDevicename` oder `pUnitname` ist nicht katalogisiert.
56	RM_INVALID_ INTERRUPT_ NUMBER	Ungültige Interrupt–Nummer

Siehe auch

RmInitOS (Seite 106)

2.98 RmSetFlag

Funktion

Ereignisflag setzen

Syntax

```
#include <rmapi.h>

int    RmSetFlag(
         uint FlagGrpID,
         uint FlagMask);
```

Parameter

Parametername	Bedeutung
FlagGrpID	Flaggruppe–ID
FlagMask	Maske legt fest, welche Bits gesetzt werden

Beschreibung

`RmSetFlag` setzt ein Ereignisflag und meldet, ob es zuvor gesetzt war.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	SVC erfolgreich ausgeführt, kein Bit gesetzt
-259	RM_FLAG_SET	Es wurde mindestens ein Flag gesetzt
-258	RM_FLAG_ ALREADY_SET	Es wurde bereits ein Flag gesetzt

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Eine ungültige `FlagGrpID` wurde übergeben.

Siehe auch

RmCreateFlagGrp (Seite 36)

RmDeleteFlagGrp (Seite 52)

RmGetFlag (Seite 70)

RmResetFlag (Seite 138)

RmSetFlagDelayed (Seite 160)

2.99 RmSetFlagDelayed

Funktion

Ereignisflag nach Intervall setzen

Syntax

```
#include <rmapi.h>

int    RmSetFlagDelayed(
            ulong TimeValue,
            uint FlagGrpID,
            uint FlagMask);
```

Parameter

Parametername	Bedeutung	
TimeValue	Verzögerungszeit bis zum Setzen des Flags	
	0... ULONG_MAX	Zeitintervall in ms.
	Zur Zeitangabe können die Werte für Stunden, Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^{31} Millisekunden.	
	RM_HOUR(hour)	Wartet (hour) Stunden
	RM_MINUTE(min)	Wartet (min) Minuten
	RM_SECOND(sec)	Wartet (sec) Sekunden
	RM_MILLISECOND(ms)	Wartet (ms) Sekunden
	Die Zeitangaben dürfen nur ein Vielfaches des Systemtaktes (Standard 10ms) sein.	
FlagGrpID	Flaggruppe–ID	
FlagMask	Maske legt fest, welche Bits gesetzt werden	

Beschreibung

Mit `RmSetFlagDelayed` werden die mit `FlagMask` angegebenen Bits gelöscht und nach Ablauf des Zeitintervalls gesetzt. Nicht gesetzte und durch `FlagMask` angegebene Bits derselben `FlagGrpID` werden nicht verändert. Die angegebenen Bits werden bei Zeitangabe = 0 nur rückgesetzt.

Ein `RmSetFlagDelayed` wird durch ein zweites `RmSetFlagDelayed` mit identischer `FlagGrpID` und `FlagMmask` bei positiver Zeitangabe überschrieben bzw. mit Zeitangabe = 0 gelöscht.

Ein `RmResetFlag` bzw. `xref` wirkt sich nicht auf `RmSetFlagDelayed` aus.

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Eine ungültige `FlagGrpID` wurde übergeben.
2	RM_PARAMETER_ ERROR	Der SVC wurde mit falschen Parametern versorgt (`FlagMask`=0).

Siehe auch

RmCreateFlagGrp (Seite 36)

RmDeleteFlagGrp (Seite 52)

RmGetFlag (Seite 70)

RmResetFlag (Seite 138)

RmSetFlag (Seite 159)

2.100 RmSetHWClockTime

Funktion

Datum und Hardware–Uhrzeit setzen

Syntax

```
#include <rmapi.h>

int    RmSetHWClockTime(
           RmAsciiTimeStruct *pTimeInfo);
```

Parameter

Parametername	Bedeutung
pTimeInfo	Zeiger auf eine Struktur vom Typ `RmAsciiTimeStruct`

Beschreibung

Neben einer internen Software–Uhr, die als lokale Zeit bezeichnet wird, unterstützt RMOS3 optional eine globale Zeit in Form einer integrierten Echtzeituhr. Mit `RmSetHWClockTime` wird die globale Zeit (Echtzeituhr) gesetzt.

Der Aufbau der Datum– und Uhrzeit–Angaben ist identisch mit `RmAsciiTimeStruct` im Systemaufruf `RmGetHWClockTime`. Die Länge des Puffers, in dem die Zeit aufbereitet wird, beträgt 20 Bytes:

```
typedef struct _RmAsciiTimeStruct
 {
        char day[2];   /* Offs. 0x0, 01 bis 31 */
        char delim0;   /* Offs. 0x2, '-' */
        char month[3]; /* Offs. 0x3, JAN, FEB, MAR, APR, */
                       /* MAY, JUN, JUL, AUG, SEP, OCT,  */
                       /* NOV, DEC                       */
        char delim1;   /* Offs. 0x6, '-'                 */
        char year[4];  /* Offs. 0x7, 0 bis 9999          */
        char blank;    /* Offs. 0xB, ' '                 */
        char hour[2];  /* Offs. 0xC, 0 bis 23            */
        char colon0;   /* Offs. 0xE, ':'                 */
        char min[2];   /* Offs. 0xF, 0 bis 59            */
        char colon1;   /* Offs. 0x11, ':'                */
        char sec[2];   /* Offs. 0x12, 0 bis 59           */
 }RmAsciiTimeStruct
```

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
37	RM_INVALID_ STRING	Ein String ist nicht innerhalb der vorgegebenen Größe.
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.

Siehe auch

Systemkonfigurierung

RmGetHWClockTime (Seite 72)

RmInitOS (Seite 106)

2.101 RmSetIntDefHandler

Funktion

Default Interrupthandler installieren

Syntax

```
#include <rmapi.h>

int    RmSetIntDefHandler (uint IntNum);
```

Parameter

Parametername	Bedeutung
IntNum	Interrupt–Nummer (0-255) IRQx (x=0 bis 63) HW-Interrupt IRQ(n) (n=0 bis 63) HW-Interrupt Auf PC-Hardware sind die HW-Interrupts von 0 bis 15 (im APIC-Modus bis 23).

Beschreibung

Der Aufruf definiert den voreingestellten Interrupthandler für unerwartete Eingabe.

Handelt es sich hierbei um einen HW–Interrupt z.B. IRQ1, wird dieser automatisch freigegeben. Zur automatischen Freigabe muss jedoch mit `RmInitOS` die Funktion `SetPICMaskProc` installiert worden sein. Alle anderen Interrupts sind vom Anwender selbst freizugeben.

Während der neue Interrupthandler gesetzt wird, darf kein Interrupt für diesen Handler auftreten.

Die Interrupt–Nummer entspricht dem zugehörigen Vektor in der Interrupt–TabelleSelektor des betreffenden Deskriptors in der IDT.

Mit `RmSetIntDefHandler` wird ein Interrupt–Gate in die IDT eingetragen.

Hinweis

Wird der SVC nicht erfolgreich ausgeführt, bleibt der bisherige Interrupthandler aktiv.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
56	RM_INVALID_ INTERRUPT_ NUMBER	Ungültige Interrupt–Nummer
41	RM_INVALID_ IRQ_NUMBER	IRQx ungültig, PIC nicht definiert

Siehe auch

RmGetIntHandler (Seite 73)

RmReserveInterrupt (Seite 137)

RmSetIntDIHandler (Seite 166)

RmSetIntISHandler (Seite 168)

RmSetIntMailboxHandler (Seite 170)

RmSetIntTaskHandler (Seite 172)

2.102 RmSetIntDIHandler

Funktion

DI–Interrupthandler vorbesetzen

Syntax

```
#include <rmapi.h>

int    RmSetIntDIHandler (
            uint IntNum,
            rmfarproc DIHandlerEntry);
```

Parameter

Parametername	Bedeutung
IntNum	Interrupt–Nummer (0-255) IRQx (x=0 bis 63) HW-Interrupt IRQ(n) (n=0 bis 63) HW-Interrupt Auf PC-Hardware sind die HW-Interrupts von 0 bis 15 (im APIC-Modus bis 23).
DIHandlerEntry	Einsprungadresse des Interrupthandlers.

Beschreibung

Der Aufruf definiert einen DI–Interrupthandler.

Handelt es sich hierbei um einen HW–Interrupt z.B. IRQ1, wird dieser automatisch freigegeben. Zur automatischen Freigabe muss jedoch mit `RmInitOS` die Funktion `SetPICMaskProc` installiert worden sein. Alle anderen Interrupts sind vom Anwender selbst freizugeben.

Während der neue Interrupthandler gesetzt wird, darf kein Interrupt für diesen Handler auftreten.

Die Interrupt–Nummer entspricht dem zugehörigen Vektor in der Interrupt–Tabelle Selektor des betreffenden Deskriptors in der IDT.

Mit `RmSetIntDIHandler` wird ein Interrupt–Gate in die IDT eingetragen.

Der in `DIHandlerEntry` angegebene Interrupthandler soll direkt nach einem Interrupt im DI–Zustand angesprungen werden. Der DI-Interrupthandler muss selbst ein EOI auslösen.

Wird als `DIHandlerEntry` NULL angegeben, entspricht dieser Aufruf dem `RmReserveInterrupt`.

Bei einem HW-Interrupt muss das entsprechende EOI-Kommando zum PIC im Interrupthandler selbst erzeugt werrden.

Hinweis

Mit `RmSetIntDIHandler` ist es unter CADUL nicht möglich, auf Privilege Level 3 einen Interrupthandler zu installieren.

Unter GNU kann der Aufruf nicht verwendet werden, um einen Interrupthandler zu restaurieren. Es gibt dafür ein spezielles Makro:

unsigned int FarPtrInf[2];
RmGetIntHandlerSeg(0xFEu,&FarPtrInf[0]); /* get int handler 0xFE */
RmRestoreIntHandlerSeg(0xFEu, FarPtrInf[1], FarPtrInf[0]);

Wird der SVC nicht erfolgreich ausgeführt, bleibt der bisherige Interrupthandler aktiv.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
56	RM_INVALID_ INTERRUPT_ NUMBER	Ungültige Interrupt–Nummer
41	RM_INVALID_ IRQ_NUMBER	IRQx ungültig, PIC nicht definiert
42	RM_INVALID_ POINTER	Ungültiger Zeiger
72	RM_INVALID_TASK_PL	Ungültiger Privilege Level

Siehe auch

RmGetIntHandler (Seite 73)

RmReserveInterrupt (Seite 137)

RmRestoreIntHandler (Seite 144)

RmSetIntDefHandler (Seite 164)

RmSetIntISHandler (Seite 168)

RmSetIntMailboxHandler (Seite 170)

RmSetIntTaskHandler (Seite 172)

2.103 RmSetIntISHandler

Funktion

S– bzw I–Interrupthandler vorbesetzen

Syntax

```
#include <rmapi.h>

int    RmSetIntISHandler (
            uint IntNum,
            rmfarproc IHandlerEntry,
            rmfarproc SHandlerEntry);
```

Parameter

Parametername	Bedeutung
IntNum	Interrupt–Nummer (0-255) IRQx (x=0 bis 63) HW-Interrupt IRQ(n) (n=0 bis 63) HW-Interrupt Auf PC-Hardware sind die HW-Interrupts von 0 bis 15 (im APIC-Modus bis 23).
IHandlerEntry	Einsprungadresse des I–Interrupthandlers.
SHandlerEntry	Einsprungadresse des S–Interrupthandlers.

Beschreibung

Der Aufruf definiert einen I– und/oder S–Interrupthandler.

Handelt es sich hierbei um einen HW–Interrupt z.B. IRQ1, wird dieser automatisch freigegeben. Zur automatischen Freigabe muss jedoch mit `RmInitOS` die Funktion `SetPICMaskProc` installiert worden sein. Alle anderen Interrupts sind vom Anwender selbst freizugeben.

Während der neue Interrupthandler gesetzt wird, darf kein Interrupt für diesen Handler auftreten.

Der in `IHandlerEntry` bzw. `SHandlerEntry` angegebene Interrupthandler soll direkt nach einem Interrupt im I– bzw. S–Zustand angesprungen werden. Soll ein Handler nicht installiert werden, so muss NULL angegeben werden.

Der `SHandlerEntry` wird nur aufgerufen, wenn der Rückgabewert vom I-Zustand ≠ 0 ist. Ist der Rückgabewert gleich 0, erfolgt kein Übergang in den S-Zustand.

Die Interrupt–Nummer entspricht dem zugehörigen Vektor in der Interrupt–TabelleSelektor des betreffenden Deskriptors in der IDT.

Mit `RmSetIntISHandler` wird ein Interrupt–Gate in die IDT eingetragen.

Der Code für den Interrupthandler wird vom Betriebssystemkern generiert. Die Handler können dann einfache Prozeduren vom Typ FAR sein. Der Speicherbedarf für einen Interrupthandler beträgt ca. 130 Byte und wird dem HEAP entnommen.

Hinweis

Mit `RmSetIntISHandler` ist es unter CADUL nicht möglich, auf Privilege Level 3 einen Interrupthandler zu installieren.

Wird der SVC nicht erfolgreich ausgeführt, bleibt der bisherige Interrupthandler aktiv.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
3	RM_OUT_OF_ MEMORY	Nicht genügend Speicher verfügbar.
56	RM_INVALID_ INTERRUPT_ NUMBER	Ungültige Interrupt–Nummer
41	RM_INVALID_ IRQ_NUMBER	IRQx ungültig, PIC nicht definiert
42	RM_INVALID_ POINTER	Ungültiger Zeiger
72	RM_INVALID_TASK_PL	Ungültiger Privilege Level

Siehe auch

RmGetIntHandler (Seite 73)

RmReserveInterrupt (Seite 137)

RmSetIntDefHandler (Seite 164)

RmSetIntDIHandler (Seite 166)

RmSetIntMailboxHandler (Seite 170)

RmSetIntTaskHandler (Seite 172)

2.104 RmSetIntMailboxHandler

Funktion

Mailbox–Interrupt–Handler vorbesetzen

Syntax

```
#include <rmapi.h>

int    RmSetIntMailboxHandler (
               uint IntNum,
               uint MailboxID,
               uint MailPriority);
```

Parameter

Parametername	Bedeutung
IntNum	Interrupt–Nummer (0-255) IRQx (x=0 bis 63) HW-Interrupt IRQ(n) (n=0 bis 63) HW-Interrupt Auf PC-Hardware sind die HW-Interrupts von 0 bis 15 (im APIC-Modus bis 23).

MailboxID	Mailbox–ID
	Es wird eine Botschaft (der Struktur `RmIntrhandMailStruct`) an die mit `MailBoxID` angegebene Mailbox geschickt. Die Struktur wird dabei vom Betriebssystem ausgefüllt und hat unten beschriebenen Inhalt.
	Unterliegt diese Mailbox einem `RmSetMailboxSize`, d.h. es darf nur eine bestimmte Anzahl von Botschaften in der Mailbox auf Abholung warten und ist diese maximale Anzahl bereits erreicht, so wird keine Botschaft abgeschickt. Der Interrupt gilt als verloren. Die in der Datei RMTYPES.H definierte Struktur für die Botschaft hat folgendes Aussehen:
	
	`typedef struct _RmIntrhandMailStruct{`
	`/* MSG1, Offs. 0x0 */`
	`unsigned int int_no;`
	`/* MSG2, Offs. 0x4 */`
	`unsigned int int_vec:8;`
	`unsigned int int_kind:1;`
	`unsigned int lost_int_overflow:1;`
	`unsigned int dummy_2:22;`
	`/* MSG3, Offs. 0x8 */`
	`unsigned short lost_int;`
	`unsigned short dummy_3;`
	`}RmIntrhandMailStruct;`
	
	`int_no` Anzahl von Interrupts
	`int_vec` Interrupt-Vektor
	`int_kind` Art des Interrupts (HW/SW: 0/1)
	`lost_int_overflow1` = verlorengegangene Interrupts
	`dummy_2` Reserviert
	`lost_int` Anzahl verlorengegangener Interrupts
	`dummy_3` Reserviert

MailPriority	Priorität der Botschaft

Beschreibung

Der Aufruf definiert einen Handler zum Senden einer Botschaft fester Struktur.

Handelt es sich hierbei um einen HW–Interrupt z.B. IRQ1, wird dieser automatisch freigegeben. Zur automatischen Freigabe muss jedoch mit `RmInitOS` die Funktion `SetPICMaskProc` installiert worden sein. Alle anderen Interrupts sind vom Anwender selbst freizugeben.

Während der neue Interrupthandler gesetzt wird, darf kein Interrupt für diesen Handler auftreten.

Unterliegt die angegebene Mailbox einer BOUND, d.h. es darf nur eine bestimmte Anzahl von Botschaften in der Mailbox auf Abholung warten. Ist diese maximale Anzahl bereits erreicht, so wird keine Botschaft abgeschickt. Der Interrupt gilt als verloren.

Die Interrupt–Nummer entspricht dem zugehörigen Vektor in der Interrupt–TabelleSelektor des betreffenden Deskriptors in der IDT.

Mit `RmSetIntMailboxHandler` wird ein Interrupt–Gate in die IDT eingetragen. Bereits vorhandene Einträge in der IDT bleiben erhalten, können aber durch den Aufruf überschrieben werden.

Der Code für den Interrupthandler wird vom Betriebssystemkern generiert. Die Handler können dann einfache Prozeduren vom Typ FAR sein. Der Speicherbedarf für einen Interrupthandler beträgt ca. 130 Byte und wird dem HEAP entnommen.

Hinweis

Wird der SVC nicht erfolgreich ausgeführt, bleibt der bisherige Interrupthandler aktiv.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
3	RM_OUT_OF_ MEMORY	Nicht genügend Speicher verfügbar.
56	RM_INVALID_ INTERRUPT_ NUMBER	Ungültige Interrupt–Nummer
41	RM_INVALID_ IRQ_NUMBER	IRQx ungültig, PIC nicht definiert
36	RM_INVALID_ID	Ungültige Mailbox–ID

Siehe auch

RmGetIntHandler (Seite 73)
RmReserveInterrupt (Seite 137)
RmSetIntDefHandler (Seite 164)
RmSetIntDIHandler (Seite 166)
RmSetIntISHandler (Seite 168)
RmSetIntTaskHandler (Seite 172)

2.105 RmSetIntTaskHandler

Funktion

Taskstart–per–Interrupt–Handler vorbesetzen

Syntax

```
#include <rmapi.h>

int    RmSetIntTaskHandler (
               uint IntNum,
               uint TaskID);
```

Parameter

Parametername	Bedeutung
IntNum	Interrupt–Nummer (0-255) IRQx (x=0 bis 63) HW-Interrupt IRQ(n) (n=0 bis 63) HW-Interrupt Auf PC-Hardware sind die HW-Interrupts von 0 bis 15 (im APIC-Modus bis 23).
TaskID	Task–ID (RM_OWN_TASK (-1) = ID der aufrufenden Task)

Beschreibung

Der Aufruf definiert einen Handler für einen Taskstart per Interrupt.

Handelt es sich hierbei um einen HW–Interrupt z.B. IRQ1, wird dieser automatisch freigegeben. Zur automatischen Freigabe muss jedoch mit `RmInitOS` die Funktion `SetPICMaskProc` installiert worden sein. Alle anderen Interrupts sind vom Anwender selbst freizugeben.

Während der neue Interrupthandler gesetzt wird, darf kein Interrupt für diesen Handler auftreten.

Die in `TaskID` angegebene Task soll direkt nach einem Interrupt angesprungen werden. Einsprung ist der im TCD konfigurierte Startpunkt.

Die Interrupt–Nummer entspricht dem zugehörigen Vektor in der Interrupt–TabelleSelektor des betreffenden Deskriptors in der IDT.

Mit `RmSetIntTaskHandler` wird ein Interrupt–Gate in die IDT eingetragen.

Der Code für den Interrupthandler wird vom Betriebssystemkern generiert. Die Handler können dann einfache Prozeduren vom Typ FAR sein. Der Speicherbedarf für einen Interrupthandler beträgt ca. 130 Byte und wird dem HEAP entnommen.

Hinweis

Wird der SVC nicht erfolgreich ausgeführt, bleibt der bisherige Interrupthandler aktiv.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
3	RM_OUT_OF_ MEMORY	Nicht genügend Speicher verfügbar.
56	RM_INVALID_ INTERRUPT_ NUMBER	Ungültige Interrupt–Nummer
41	RM_INVALID_ IRQ_NUMBER	IRQx ungültig, PIC nicht definiert
36	RM_INVALID_ID	Ungültige Task–ID

Siehe auch

RmGetIntHandler (Seite 73)

RmReserveInterrupt (Seite 137)

RmSetIntDefHandler (Seite 164)

RmSetIntDIHandler (Seite 166)

RmSetIntISHandler (Seite 168)

RmSetIntMailboxHandler (Seite 170)

2.106 RmSetLocalFlag

Funktion

Lokales Ereignisflag setzen

Syntax

```
#include <rmapi.h>

int    RmSetLocalFlag(
            uint TaskID,
            uint FlagMask);
```

Parameter

Parametername	Bedeutung
TaskID	Task–ID (RM_OWN_TASK (-1) = eigeneTask)
FlagMask	Maske legt fest, welche Flagbits gesetzt werden sollen.

Beschreibung

`RmSetLocalFlag` setzt das lokale Ereignisflag der mit `TaskID` angegebenen Task.

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt
-259	RM_FLAG_SET	Es wurde mindestens ein Flag gesetzt
-258	RM_FLAG_ ALREADY_SET	Es wurde bereits ein Flag gesetzt

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	TaskID ungültig

Siehe auch

RmGetFlag (Seite 70)

RmResetFlag (Seite 138)

RmResetLocalFlag (Seite 139)

2.107 RmSetMailboxSize

Funktion

Grenzwerte für Mailboxes festlegen

Syntax

```
#include <rmapi.h>

int    RmSetMailboxSize (
           uint MailboxID,
           uint Limit);
```

Parameter

Parametername	Bedeutung	
MailboxID	Mailbox–ID	
Limit	1–0FFFFH	Maximale Anzahl von Botschaften in Warteschlange
	0	Bedeutet, der Grenzwert soll aufgehoben werden.

Beschreibung

Der Aufruf setzt einen Grenzwert für die Anzahl von Botschaften, die an einer Mailbox warten können. Der Grenzwert kann beliebig modifiziert und auch wieder aufgehoben werden.

Beim Überschreiten des Grenzwerts werden alle nachfolgenden Versuche, mit dem Aufruf `RmSendMail` eine Botschaft an diese Mailbox zu schicken, zurückgewiesen, bis der Grenzwert durch Abholen von Botschaften wieder unterschritten wird.

Der Grenzwert bezieht sich nur auf wartende Botschaften. Die Anzahl wartender Tasks, die auf den Empfang einer Botschaft warten, ist davon nicht betroffen.

Hinweis

Der bei Mailboxen festgelegte Grenzwert hat bei dem Systemaufruf `RmSendMailDelayed` keine Wirkung.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Mailbox–ID ungültig

Siehe auch

RmReceiveMail (Seite 128)

RmSendMail (Seite 150)

RmSendMailCancel (Seite 152)

RmSendMailDelayed (Seite 153)

2.108 RmSetMessageQueueSize

Funktion

Länge der Message–Queue festlegen

Syntax

```
#include <rmapi.h>

int    RmSetMessageQueueSize (
            uint TaskID,
            uint Limit);
```

Parameter

Parametername	Bedeutung
TaskID	Zieltask–ID
Limit	Anzahl von freien Plätzen in der Message–Queue

Beschreibung

Der Aufruf legt die Größe der Message–Queue der Task `TaskID` fest.

Der Aufruf setzt einen Grenzwert für die Anzahl von Messages, die an einer Messagequeue warten können.

Der Grenzwert kann beliebig modifiziert und auch wieder aufgehoben werden.

Beim Überschreiten des Grenzwerts werden alle nachfolgenden Versuche, mit dem Aufruf `RmSendMessage` eine Message in diese Queue zu schicken, zurückgewiesen, bis der Grenzwert durch Abholen von Messages wieder unterschritten wird.

Der Grenzwert bezieht sich nur auf wartende Botschaften.

Die Anzahl wartender Tasks, die auf den Empfang einer Botschaft warten, ist davon nicht betroffen.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Task–ID ungültig
35	RM_INVALID_TYPE	Ein ungültiger Parameter (`Limit`) wurde übergeben.
58	RM_QUEUE_NOT_ EXIST	Die Message–Queue existiert nicht.

Siehe auch

RmCreateMessageQueue (Seite 40)

RmDeleteMessageQueue (Seite 55)

RmReadMessage (Seite 124)

RmSendMessage (Seite 155)

2.109 RmSetOS

Funktion

Systemkonfiguration durchführen

Syntax

```
#include <rmapi.h>

int    RmSetOS (
         uint RoundRobinCounter,
         uint SVCExceptionType,
         char *pUnitName);
```

Parameter

Parametername	Bedeutung
RoundRobinCounter	Round–Robin–Zähler RM_RR_OFF (0) = ausgeschaltet RM_RR_TASK (-1) = taskspezifischer RR RM_RR_UNCHANGED (-2) = unverändert
SVCExceptionType	SVC–Exception–Ausgaberoutine aufrufen: RM_SVCEXC_ON (1) = ja RM_SVCEXC_OFF (0) = nein RM_SVCEXC_UNCHANGED (-1) = unverändert RM_SVCEXC_OFF_ONE (3) = eine SVC-Meldung wird abgeschaltet RM_SVCEXC_ON_ONE (2) = eine SVC-Meldung wird eingeschaltet
pUnitName	Zeiger auf einen C–String, der den Namen enthält, mit dem die Unit katalogisiert wurde, die zur Systemkonsole werden soll.

Beschreibung

Der Aufruf kann auch nach Beendigung der Konfigurationsphase aufgerufen werden. Die Systemkonsole wird für Ausgaben der Errorlogger–Task oder als Default–Konsole für die CRUN verwendet. Beim Umstellen der Konsole wird ebenfalls die Funktion `Putchar` umgestellt. Die Adresse der neuen Funktion `Putchar` wird aus dem UCD–Eintrag `UCD.INTADR` übernommen.

Mit RmSetOS können SVC-Meldungen für bestimmte SVCs dynamisch zur Laufzeit ein- und ausgeschaltet werden. Dazu muss der Parameter `SVCExceptionType` auf RM_SVCEXC_OFF_ONE oder RM_SVCEXC_ON_ONE gesetzt werden. In den höherwertigen 16 Bit des Parameters `RoundRobinCounter` gibt man die abzuschaltende SVC-Nummer an, in den niederwertigen 16 Bit den Fehlercode, der ab- oder eingeschaltet werden soll. Die SVC-Nummer kann mit Hilfe des Makros SVCNR übergeben werden.

Die in svcexc.c angegebenen SVC-Meldungen werden so wie bisher abgeschaltet, können aber dynamisch zur Laufzeit wieder eingeschaltet werden. Maximal 50 SVC-Meldungen können nachträglich abgeschaltet werden.

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
21	RM_INVALID_ SELECTOR	Die Funktion für Notausgaben der angegebenen Systemkonsole besitzt keinen Codeselector.
37	RM_INVALID_ STRING	Die Länge von `pDeviceName` oder `pUnitName` ist unzulässig. Sie war Null oder größer als 15.
48	RM_IS_NOT_ CATALOGED	`pUnitName` ist nicht katalogisiert oder die SVC-Meldung, die eingeschaltet werden soll, war nicht abgeschaltet.
35	RM_INVALID_TYPE	`SVCExceptionType` ungültig oder Unit kann nicht zur Systemkonsole werden
47	RM_IS_ALREADY_CATALO GED	SVC-Meldung ist bereits abgeschaltet.
3	RM_OUT_OF_ MEMORY	Es wurden bereits 50 SVC-Meldungen abgeschaltet (neben denen aus `svcexc.c`).

Beispiel

Die SVC-Meldung RM_PARAMETER_ERROR für den SVC `RmGetTaskID` soll abgeschaltet werden.

```
RmSetOS(SVCNR(EXC_RmGetTaskID)|RM_PARAMETER_ERROR, RM_SVCEXC_OFF_ONE, NULL);
```

Siehe auch

RmSetPutcharProc (Seite 180)

2.110 RmSetProfilerPoint

Funktion

Setzen von Orientierungspunkten im Profiler `rprof.386` für die Messung "Task Activity"

Syntax

```
#include <rmapi.h>

int    RmSetProfilerPoint (uint Point);
```

Parameter

Parametername	Bedeutung
Point	Beliebiger Wert zur Identifizierung des Aufrufs im Profiler-Messergebnis

Beschreibung

Der Aufruf `RmSetProfilerPoint` dient zum Setzen von Orientierungspunkten im Profiler für die Messung "Task Activity". Er führt keine Aktion aus. Der Parameter `Point` erscheint im Messergebnis des Profilers in der Spalte "DATA".

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Beispiel

```
Error = RmGetBinSemaphore(RM_WAIT, SemaID);
Error = RmSetProfilerPoint(1);
Error = RmGetAbsTime(&AbsTime);
Error = RmSetProfilerPoint(2);
Error = RmReleaseBinSemaphore(SemaID);
```

2.111 RmSetPutcharProc

Funktion

Adresse der Funktion `putchar()` bekanntgeben

Syntax

```
#include <rmapi.h>

int    RmSetPutcharProc (int (_FAR *PutcharProc)( char c));
```

Parameter

Parametername	Bedeutung
PutcharProc	Adresse der Funktion `putchar()`

Beschreibung

Die Funktion gibt der Funktion `x_nucprintf()` die Adresse der Funktion `putchar()` bekannt. Die Funktion `x_nucprintf()` benötigt die Adresse von `putchar()` für ihre Ausgaben, die bis zum Aufruf dieser Funktion verworfen werden. Dies entspricht der früheren Option LOG_PASSPOINT YES/NO.

Hinweis

Diese Funktion ist nicht im Speichermodell Flat verwendbar.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
42	RM_INVALID_ POINTER	`PutcharProc` ist ungültiger Zeiger.
21	RM_INVALID_ SELECTOR	`PutcharProc` enthält keinen Codeselector.

Siehe auch

RmSetOS (Seite 177)

2.112 RmSetShIntISHandler1

Funktion

Shared Interrupt Handler im Compatibility-Mode installieren
(in RMISHCLI.LIB enthalten)

Syntax

```
#include <rmpci.h>

int    RmSetShIntISHandler1 (
              uint Irqx,
              int _FIXED _FAR (*IHandlerEntry) (void),
              int _FIXED _FAR (*SHandlerEntry) (void),
              int _FIXED _FAR (*GetIntReqState) (void *),
              void *UsrContext,
              uint *chainID );
```

Parameter

Parametername	Bedeutung
Irqx	Interrupt Request Nummer (IRQ0-15 bzw. IRQ0-23) Sie kann mit dem Funktionsaufruf `RmPciSearchFunction` aus der Struktur PCI_CONFIG_SPACE (Element `IntLine`) ermittelt werden.
IHandlerEntry	Einsprungadresse des I–Interrupthandlers.
SHandlerEntry	Einsprungadresse des S–Interrupthandlers.
GetIntReqState	Pointer auf eine im Applikationstreiber zu implementierende Funktion, die überprüft, ob der Interrupthandler zuständig ist für den angenommenen Interrupt `Irqx`. Der Rückgabewert dieser Funktion ist 0, wenn der Treiber nicht zuständig ist für den momentanen Interrupt. Der Rückgabewert ist 1, wenn der Treiber zuständig ist.
UsrContext	Pointer auf eine in der Applikation abgelegte Datenstruktur, zum Datenaustausch zwischen Applikation und Treiber. Dieser Zeiger wird jedem Aufruf von `GetIntReqState` mitgegeben, um dem Treiber einen Verweis auf seinen Kontext zu ermöglichen.
chainID	Pointer auf ID für InterruptHandlerEntry (Rückgabeparameter) ID für InterruptHandlerEntry. Dies ist eine vom Shared-Interrupt-Server vergebene, eindeutige ID, die der aufrufenden Applikation mitgeteilt wird. Diese ID ist nötig, um mit `RmClrShIntISHandler` den Interrupthandler wieder entfernen zu können.

Beschreibung

Der Aufruf definiert einen I– und/oder S–Interrupthandler für shared Interrupts.

`RmSetShIntISHandler1 r`uft eine Funktion des Shared Interrupt Servers auf, die einen Interrupt-Handler im Mode 1 (Compatibility Mode) im Server installiert. Dabei wird ein I-Handler, ein S-Handler und eine zugehörige `GetIntReqState`-Funktion für den vorgegebenen `Irqx` installiert.

Die Funktion `GetIntReqState` wird beim Auftreten eines Interrupts vom Server aufgerufen und teilt dem Server über den Rückgabewert mit, ob der Interrupt von diesem Treiber zu behandeln ist oder nicht. Dies kann festgestellt werden, indem die `GetIntReqState`-Funktion das Interrupt-Register überprüft, ob diese Baugruppe den Interrupt ausgelöst hat. Anschließend wird der I-Handler und/oder der S-Handler für diesen Treiber aufgerufen.

Soll ein Handler nicht installiert werden, so muss NULL angegeben werden.

Wie bei der Funktion `RmSetIntISHandler` wird der `SHandlerEntry` nur aufgerufen, wenn der Rückgabewert vom I-Zustand ≠ 0 ist. Ist der Rückgabewert gleich 0, erfolgt kein Übergang in den S-Zustand.

ImGegensatz zur Funktion `RmSetShIntISHandler2` können bereits vorhandene Interruptroutinen weiter verwendet werden, da die separate Funktion `GetIntReqState` die Zuständigkeitsüberprüfung vornimmt.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
3	RM_OUT_OF_ MEMORY	Nicht genügend Speicher verfügbar.
41	RM_INVALID_IRQ_ NUMBER	`Irqx` ungültig
42	RM_INVALID_ POINTER	Ungültiger Zeiger
4	RM_GOT_ TIMEOUT	bei Zugriff auf verkettete Liste ist ein Timeout aufgetreten

Beispiel

Siehe RmSetShIntISHandler2 (Seite 183)

Siehe auch

RmClrShIntISHandler (Seite 28)

RmInitShIntClient (Seite 104)

RmPciSearchFunction (Seite 118)

RmPciSearchSubFunction (Seite 120)

RmSetShIntISHandler2 (Seite 183)

2.113 RmSetShIntISHandler2

Funktion

Shared Interrupt Handler im Fast-Mode installieren
(in RMISHCLI.LIB enthalten)

Syntax

```
#include <rmpci.h>

int    RmSetShIntISHandler2 (
              uint Irqx,
              int _FIXED _FAR (*IHandlerEntry) (void *),
              int _FIXED _FAR (*SHandlerEntry) (void *),
              void *UsrContext,
              uint *chainID );
```

Parameter

Parametername	Bedeutung
Irqx	Interrupt Request Nummer (IRQ0-15 bzw. IRQ0-23) Sie kann mit dem Funktionsaufruf `RmPciSearchFunction` aus der Struktur PCI_CONFIG_SPACE (Element `IntLine`) ermittelt werden.
IHandlerEntry	Einsprungadresse des I–Interrupthandlers.
SHandlerEntry	Einsprungadresse des S–Interrupthandlers.
UsrContext	Pointer auf eine in der Applikation abgelegte Datenstruktur, zum Datenaustausch zwischen Applikation und Treiber.
ChainID	Pointer auf ID für InterruptHandlerEntry (Rückgabeparameter) ID für InterruptHandlerEntry. Dies ist eine vom Shared-Interrupt-Server vergebene, eindeutige ID, die der aufrufenden Applikation mitgeteilt wird. Diese ID ist nötig, um mit `RmClrShIntISHandler` den Interrupthandler wieder entfernen zu können.

Beschreibung

Der Aufruf definiert einen I– und/oder S–Interrupthandler für shared Interrupts im Mode 2 (Fast Mode).

`RmSetShIntISHandler2 r`uft eine Funktion des Shared Interrupt Servers auf, die einen Interrupt-Handler im Mode 2 (Fast Mode) im Server installiert. Dabei wird ein I-Handler und/oder ein S-Handler für den vorgegebenen `Irqx` installiert.

Der I-Handler wird nach Auslösen des Interrupts `Irqx` aufgerufen. Er muss nun selbst prüfen, ob er für den Interrupt zuständig ist oder nicht und diesen bei Bedarf bearbeiten.
Dies kann festgestellt werden, indem in der I-Handler-Routine das Interrupt-Register überprüft wird, ob diese Baugruppe den Interrupt ausgelöst hat. Falls er nicht zuständig ist, muss der Handler sofort mit `return (2)` und ohne Interruptquittierung verlassen werden.
Falls der Handler den Interrupt zu bearbeiten hat, muss der Handler anschließend mit `return (0)` oder `return (1)` verlassen werden.
Bei `return (0)` wird der S-Handler nicht aufgerufen, bei `return (1)` wird der S-Handler aufgerufen.

Soll ein Handler nicht installiert werden, so muss NULL angegeben werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
3	RM_OUT_OF_ MEMORY	Nicht genügend Speicher verfügbar.
41	RM_INVALID_IRQ_ NUMBER	`Irqx` ungültig
42	RM_INVALID_ POINTER	Ungültiger Zeiger
4	RM_GOT_ TIMEOUT	bei Zugriff auf verkettete Liste ist ein Timeout aufgetreten

Beispiel

```
typedef struct {
        int     * pxyz;
        uint    chainID;
 } T_CALLER;

int _FIXED _FAR ISR_I_Handler(T_CALLER *Caller)
{

int IntReg;

/* Read the Interrupt-Register of board xyz*/
/* IntReg = *(Caller->pxyz+INTERRUPT_REG); */
    /* handle the interrupt, if competent for it */
    if ( IntReg != 0 )
        return (1);            /* handle the interrupt, in the S-handler */
    else
        return (2);            /* not competent of this interruptrequest */
}
int _FIXED _FAR ISR_S_Handler (T_CALLER *Caller)
{
    /* to do */
    return (RM_OK);
}

int _FAR main(void)
{
/* variables */
int       error, i;
ushort    numOfFunctions;
PCI_CONFIG_SPACE        ConfigSpace[MAX_NUM_OF_FUNCTIONS];
T_CALLER                Caller[MAX_NUM_OF_FUNCTIONS];

/* detect all functions in the whole pci system, whose vendor and */

/* device ids correspond to a xyz-PCI board                                  */
    err = RmPciSearchSubFunction(
                        xyz_VENDOR_ID,
                        xyz_DEVICE_ID,
                        xyz_SUB_VENDOR_ID,
                        xyz_SUBSYSTEM_ID,
```

```
                    ConfigSpace,
                    MAX_NUM_OF_FUNCTIONS,
                    &numOfFunctions);
    /* PCI-device found */
    if (numOfFunctions !=0 )
    {
        /* initialization of shared interrupt client */
        err = RmInitShIntClient();

        for (i=0; i<numOfFunctions;i++)
        {
            /* install interrupt handler for i and s state */
            err=RmSetShIntISHandler2(
                    IRQ(ConfigSpace[i].IntLine),
                    ISR_I_Handler,
                    ISR_S_Handler,
                    &Caller[i],
                    &Caller[i].chainID);
        }

        /* to do ...  */

        for (i=0; i<numOfFunctions;i++)
        {
            /* clear interrupt handler */
            err= RmClrShIntISHandler(
                    IRQ(ConfigSpace[i].IntLine),
                    &Caller[i].chainID);
        }
    }
    else exit(1);
}
```

Siehe auch

RmClrShIntISHandler (Seite 28)

RmInitShIntClient (Seite 104)

RmPciSearchFunction (Seite 118)

RmPciSearchSubFunction (Seite 120)

RmSetShIntISHandler1 (Seite 181)

2.114 RmSetSMRCount

Funktion

SMR–Grenze festlegen

Syntax

```
#include <rmapi.h>

int    RmSetSMRCount (uint SMRNumber);
```

Parameter

Parametername	Bedeutung
SMRNumber	Maximale Anzahl von SMRs

Beschreibung

Der Aufruf legt die Obergrenze der allokierbaren Speicherblöcke für SMRs fest.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

2.115 RmSetSVC

Funktion

SVC–Tabelle initialisieren

Syntax

```
#include <rmapi.h>

int    RmSetSVC (
         uint SVCNumber,
         rmfarproc SVCHandler);
```

Parameter

Parametername	Bedeutung
SVCNumber	SVC–Nummer
SVCHandler	Anfangsadresse der SVC–Routine

Beschreibung

Mit dem Aufruf wird die SVC–Tabelle beeinflußt. Im Ursprungszustand sind alle Einträge der Tabelle reserviert. Werden dann beim Systemhochlauf die einzelnen Funktionen initialisiert, müssen die einzelnen SVCs eingetragen werden. So trägt z.B. die Initialisierung der Eventflags die Eventflag–SVCs in die Tabelle ein. Damit wird erreicht, dass die SVC–Tabelle nicht mehr Bestandteil der Konfigurationsdatei ist. Ferner lassen sich Unstimmigkeiten in der Systemkonfigurierung vermeiden, da es nicht mehr passieren kann, dass beispielsweise das Dateiverwaltungssystem konfiguriert, der zugehörige SVC aber nicht eingetragen ist.

Hinweis

Diese Funktion ist nicht im Speichermodell Flat verwendbar.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Die angegebene SVC–Nummer liegt außerhalb der möglichen SVCs.

2.116 RmSetSystemTime

Funktion

Datum und Uhrzeit setzen

Syntax

```
#include <rmapi.h>

int    RmSetSystemTime(RmAsciiTimeStruct *pTimeInfo);
```

Parameter

Parametername	Bedeutung
pTimeInfo	Zeiger auf eine Struktur vom Typ `RmAsciiTimeStruct`

Beschreibung

Mit `RmSetSystemTime` wird die lokale Zeit gesetzt.

Der Aufbau der Datum– und Uhrzeit–Angaben ist identisch mit `RmAsciiTimeStruct` im Systemaufruf `RmGetSystemTime`. Die Länge des Puffers, in dem die Zeit aufbereitet wird, beträgt 20 Bytes:

```
typedef struct _RmAsciiTimeStruct
{
          char day[2];   /* Offs. 0x0, 01 bis 31               */
          char delim0;   /* Offs. 0x2, '-'                     */
          char month[3]; /* Offs. 0x3, JAN, FEB, MAR, APR, MAY,
                            MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC */
          char delim1;   /* Offs. 0x6, '-'                     */
          char year[4];  /* Offs. 0x7, 0 bis 9999              */
          char blank;    /* Offs. 0xB, ' '                     */
          char hour[2];  /* Offs. 0xC, 0 bis 23                */
          char colon0;   /* Offs. 0xE, ':'                     */
          char min[2];   /* Offs. 0xF, 0 bis 59                */
          char colon1;   /* Offs. 0x11, ':'                    */
          char sec[2];   /* Offs. 0x12, 0 bis 59               */
}RmAsciiTimeStruct
```

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
37	RM_INVALID_ STRING	Ein String ist nicht innerhalb der vorgegebenen Größe.
42	RM_INVALID_ POINTER	Ein Zeiger war ungültig.

Siehe auch

Systemkonfigurierung

RmGetSystemTime (Seite 89)

2.117 RmSetTaskPriority

Funktion

Task–Priorität ändern

Syntax

```
#include <rmapi.h>

int    RmSetTaskPriority(
           uint TaskID,
           uint Priority);
```

Parameter

Parametername	Bedeutung	
TaskID	Zieltask–ID (–1=eigene)	
Priority	0..255	Vorgegebenen Wert einstellen
	RM_TCDPRI (0x400)	Priorität aus TCD entnehmen
	RM_CURPRI (0x300)	Aktuelle Priorität der aufrufenden Task verwenden
	RM_MAXPRI (0x100)	Maximum einstellen (RM_TCDPRI, RM_CURPRI)
	RM_INCPRI (0x500)	Priorität der Task um 1 erhöhen
	RM_DECPRI (0x600)	Priorität der Task um 1 erniedrigen

Beschreibung

Mit `RmSetTaskPriority` wird die Priorität einer beliebigen Task verändert (auch wenn die Task im Zustand DORMANT ist).

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt
-261	RM_PRI_NOT_ CHANGED	Priorität der Task wurde nicht verändert.

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	`TaskID` ungültig
35	RM_INVALID_TYPE	Ein ungültiger Parameter (`Priority`) wurde übergeben.

Siehe auch

Beschreibung der Task–Priorität im Benutzerhandbuch

RmQueueStartTask (Seite 122)

RmStartTask (Seite 190)

2.118 RmStartTask

Funktion

Task–Start–Anforderung für Tasks, die sich im Zustand DORMANT befinden.

Syntax

```
#include <rmapi.h>

int    RmStartTask(
         uint Wait,
         uint TaskID,
         uint Priority,
         uint RegVal1,
         uint RegVal2);
```

Parameter

Parametername	Bedeutung	
Wait	RM_NO_WAIT (0)	Zieltask starten und eigene Abarbeitung fortsetzen
	RM_WAIT_READY (1)	Warten bis Zieltask im Zustand BEREIT ist
	RM_WAIT_END (2)	Warten bis Zieltask beendet ist
TaskID	Zieltask–ID	
Priority	0..255	Vorgegebenen Wert einstellen
	RM_TCDPRI (0x400)	Priorität aus TCD entnehmen
	RM_CURPRI (0x300)	Aktuelle Priorität der aufrufenden Task verwenden
	RM_MAXPRI (0x100)	Maximum (RM_TCDPRI, RM_CURPRI) einstellen
RegVal1	Parameter 1 (Übergabe in eax der Zieltask)	
RegVal2	Parameter 2 (Übergabe in ebx der Zieltask)	

Beschreibung

Mit `RmStartTask` wird eine Task gestartet. Der SVC benötigt dieselben Parameter wie `RmQueueStartTask`. Der Unterschied zu `RmQueueStartTask` besteht darin, dass `RmQueueStartTask` die Start–Anforderung in eine Warteschlange einreiht, falls sich die Task nicht im Zustand DORMANT befindet, während `RmStartTask` in diesem Fall wirkungslos ist und nur die Flags gesetzt werden.

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Zieltask in Zustand BEREIT überführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Eine ungültige `TaskID` wurde übergeben.
35	RM_INVALID_TYPE	Ein ungültiger Parameter (`Priority`) wurde übergeben.
20	RM_TASK_NOT_ DORMANT	Es wurde versucht, eine Task, die nicht im Zustand DORMANT ist, zu starten.
49	RM_TASK_KILLED	Zieltask wurde vor Erreichen des Zustands BEREIT oder vor ihrer Beendigung durch `RmKillTask` in den Zustand DORMANT versetzt bzw. gelöscht.

Siehe auch

Task–Konfiguration

RmQueueStartTask (Seite 122)

2.119 RmSuspendDriver

Funktion

Gerätetreiber suspendieren (in den Zustand WARTEND überführen)

Syntax

```
#include <rmapi.h>

int    RmSuspendDriver(uint DriverID);
```

Parameter

Parametername	Bedeutung
DriverID	Geräte-ID

Beschreibung

`RmSuspendDriver` überführt einen Gerätetreiber vom Zustand BEREIT in SUSPENDIERT (WARTEND). Vom Gerätetreiber allokierter Speicher wird nicht zurückgegeben.

Hinweis

Hat der Gerätetreiber Speicher allokiert, so muss dieser Speicher gesondert zurückgegeben werden.

Diese Funktion ist nicht im Speichermodell Flat verwendbar.

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Zieltask in Zustand BEREIT überführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	Eine ungültige `DriverID` wurde übergeben.
18	RM_RESOURCE_ BUSY	Gerätetreiber noch aktiv

Siehe auch

Dynamisches Erzeugen und Löschen von Treibern

RmResumeDriver (Seite 145)

2.120 RmSuspendTask

Funktion

Task vom Zustand BEREIT oder RECHNEND in Zustand WARTEND setzen

Syntax

```
#include <rmapi.h>

int    RmSuspendTask(uint TaskID);
```

Parameter

Parametername	Bedeutung
TaskID	Task–ID (RM_OWN_TASK (-1) = eigene Task)

Beschreibung

`RmSuspendTask` hält die mit `TaskID` angegebene Task an. Die angehaltene Task muss im Zustand BEREIT sein und befindet sich danach im Zustand WARTEND. Eine Task kann sich auch selber anhalten.

Hinweis

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

ID dez	Fehlercode	Bedeutung
36	RM_INVALID_ID	TaskID ungültig
30	RM_TASK_NOT_ READY	Task war nicht im Zustand BEREIT

Siehe auch

Systemkonfigurierung, Taskverwaltung, TCD–Block, Debugger–Kommando QUERY

RmActivateTask (Seite 17)

RmResumeTask (Seite 146)

2.121 RmUncatalog

Funktion

Betriebsmittel aus Catalog löschen

Syntax

```
#include <rmapi.h>

int    RmUncatalog (char *pName);
```

Parameter

Parametername	Bedeutung
pName	Zeiger auf ein Zeichenstring (Der String kann nach C- oder PLM-Notation definiert sein).

Beschreibung

`RmUncatalog` entfernt das durch einen Zeichenstring identifizierte Betriebsmittel aus dem Katalog.

Hinweis

Ist ein Betriebsmittel mit unterschiedlichen Strings mehrmals katalogisiert, so werden alle Einträge des Betriebsmittels aus dem Katalog gelöscht.

Diese Funktion kann nicht zur Konfiguration des RMOS3-Nukleus verwendet werden.

Rückgabewert

ID dez	Rückgabewert	Bedeutung
0	RM_OK	SVC ausgeführt, Eintrag gefunden und entfernt

Fehlercodes

ID dez	Fehlercode	Bedeutung
48	RM_IS_NOT_ CATALOGED	Eintrag nicht gefunden
42	RM_INVALID_ POINTER	Zeiger `pName` war ungültig.
37	RM_INVALID_ STRING	Stringlänge = 0 oder > 15

Siehe auch

RmGetEntry (Seite 68)

RmGetName (Seite 77)

RmList (Seite 112)

C-Laufzeitbibliothek 3

3.1 _ctime32

Funktion

Datum und Uhrzeit in einen String umwandeln

Syntax

```
#include <time.h>

char    *_ctime32 (time32_t *clock);
```

Parameter

Parametername	Bedeutung
clock	Zeiger auf ein Element vom Typ `time32_t`, das die umzuwandelnde Zeitangabe enthält

Beschreibung

`_ctime32` wandelt einen Unsigned Long–Integer (wie er von `_time32` geliefert wird) - unter Berücksichtigung der Zeitzone und gegebenenfalls Verschiebungen durch die Sommerzeit - in einen String um und liefert als Ergebnis einen Zeiger auf ein 26 Byte langes statisches Datenfeld im Format "Wochentag Monat Tag Stunden:Minuten:Sekunden Jahr\n".

Beispiel: `Mon Jan 01 12.30.45 1990\n`

Hinweis

Die Rückgabewerte zeigen auf einen Datenbereich, dessen Inhalt beim nächsten Aufruf überschrieben wird.

`_ctime32(clock)` ist äquivalent zu `asctime(_localtime32(clock))`.

Verwenden Sie `_ctime32`, wenn Sie Zeitangaben über den 19. Januar 2038 hinaus benötigen.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf einen String, der Datums– und Zeitangabe enthält

Beispiel

Das folgende Beispiel zeigt den Einsatz der verschiedenen CRUN–Zeitfunktionen:

```
#include <time.h>
main () {
 struct tm *ptm;
 char *date;
 time32_t tadd;

 putenv ("TZ=MEZ-1");

 _time32 (&tadd);
 date = _ctime32 (&tadd);
 printf ("today's date is %s", date);

 ptm = _localtime32 (&tadd);
 date = asctime (ptm);
 printf ("today's date is %s", date);

 ptm = _gmtime32 (&tadd);
 date = asctime (ptm);
 printf ("Greenwich mean time is %s", date);
}
```

3.2 _gmtime32

Funktion

Datum und Uhrzeit in Greenwich Mean Time (GMT) umwandeln

Syntax

```
#include <time.h>

struct    tm *_gmtime32 (time32_t *clock);
```

Parameter

Parametername	Bedeutung
clock	Zeiger auf ein Element vom Typ `time32_t`, das die umzuwandelnde Zeitangabe enthält

Beschreibung

`_gmtime32` wandelt Datum und Uhrzeit aus einem Unsigned Long–Integer (wie er von `_time32` geliefert wird) in GMT (Greenwich Mean Time) um. `_gmtime32` gibt einen Zeiger auf eine Struktur vom Typ `tm` zurück.

Die Struktur `tm` ist in TIME.H folgendermaßen definiert:

```
struct tm{
 int tm_sec;   /* Sekunden (0-59)              */
 int tm_min;   /* Minuten (0-59)               */
 int tm_hour;  /* Stunden (0-23)               */
 int tm_mday;  /* Monatstag (1-31)             */
 int tm_mon;   /* Monat (0-11)                 */
 int tm_year;  /* Jahr minus 1900              */
 int tm_wday;  /* Wochentag (Sonntag = 0)      */
 int tm_yday;  /* laufender Tag im Jahr (0-365) */
 int tm_isdst; /* ungleich 0 falls Sommerzeit  */
};
```

Hinweis

Die Rückgabewerte zeigen auf einen Datenbereich, dessen Inhalt beim nächsten Aufruf überschrieben wird.

Verwenden Sie `_gmtime32`, wenn Sie Zeitangaben über den 19. Januar 2038 hinaus benötigen.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf eine Struktur vom Typ `tm`

Beispiel

Das folgende Beispiel zeigt den Einsatz der verschiedenen CRUN–Zeitfunktionen:

```
#include <time.h>
main ()
{
   struct tm *ptm;
   char *date;
   time32_t tadd;
   putenv ("TZ=MEZ-1");
   _time32 (&tadd);
   date = _ctime32 (&tadd);
   printf ("today's date is %s", date);

   ptm = _localtime32 (&tadd);
   date = asctime (ptm);
   printf ("today's date is %s", date);

   ptm = _gmtime32 (&tadd);
   date = asctime (ptm);
   printf ("Greenwich mean time is %s", date);
}
```

Siehe auch

_localtime32 (Seite 199)

_time32 (Seite 201)

asctime (Seite 206)

gmtime (Seite 286)

tzset (Seite 410)

3.3 _localtime32

Funktion

Datum und Uhrzeit in Lokalzeit umwandeln

Syntax

```
#include <time.h>

struct tm    *_localtime32 (time32_t *clock);
```

Parameter

Parametername	Bedeutung
clock	Zeiger auf ein Element vom Typ `time32_t`, das die umzuwandelnde Zeitangabe enthält

Beschreibung

`_localtime32` wandelt Datum und Uhrzeit in Lokalzeit um, wobei Zeitangaben entsprechend Zeitzonen–Differenzen und gegebenenfalls Verschiebungen durch die Sommerzeit korrigiert werden. Die Sommerzeit beginnt am letzten Sonntag im März um 2.00 Uhr morgens und endet am letzten Sonntag im Oktober. `_localtime32` gibt einen Zeiger auf eine Struktur vom Typ `tm` zurück.

Die Struktur `tm` ist in TIME.H folgendermaßen definiert:

```
struct tm{
 int tm_sec;   /* Sekunden (0-59) */
 int tm_hour;  /* Stunden (0-23) */
 int tm_mday;  /* Monatstag (1-31) */
 int tm_mon;   /* Monat (0-11) */
 int tm_year;  /* Jahr minus 1900 */
 int tm_wday;  /* Wochentag (Sonntag = 0) */
 int tm_yday;  /* laufender Tag im Jahr (0-365) */
 int tm_isdst; /* ungleich 0 falls Sommerzeit */
 };
```

Hinweis

Die Rückgabewerte zeigen auf einen Datenbereich, dessen Inhalt beim nächsten Aufruf überschrieben wird.

Verwenden Sie `_localtime32`, wenn Sie Zeitangaben über den 19. Januar 2038 hinaus benötigen.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf eine Struktur vom Typ `tm`

Beispiel

Das folgende Beispiel zeigt den Einsatz der verschiedenen CRUN–Zeitfunktionen:

```
#include <time.h>
main ()
{
 struct tm *ptm;
 char *date;
 time32_t tadd;
 putenv ("TZ=MEZ-1");
 _time32 (&tadd);
 date = _ctime32 (&tadd);
 printf ("today's date is %s", date);

 ptm = _localtime32 (&tadd);
 date = asctime (ptm);
 printf ("today's date is %s", date);

 ptm = _gmtime32 (&tadd);
 date = asctime (ptm);
 printf ("Greenwich mean time is %s", date);
}
```

Siehe auch

_gmtime32 (Seite 197)

_time32 (Seite 201)

localtime (Seite 305)

3.4 _time32

Funktion

Systemzeit ermitteln, seit 01.01.1970, 00:00:00 GMT

Syntax

```
#include <time.h>

time32_t    _time32 (time32_t *tptr);
```

Parameter

Parametername	Bedeutung
tptr	Zeiger auf einen Puffer

Beschreibung

`_time32` liefert die Systemzeit als Anzahl der Sekunden, die seit dem 1. Januar 1970, GMT verstrichen sind. Wenn der Zeiger `tptr` kein NULL–Zeiger ist, wird der Rückgabewert in den mit `tptr` angegebenen Bereich geschrieben.

Rückgabewert

Rückgabewert	Bedeutung
time32_t	Anzahl der Sekunden seit 1. Januar 1970, 00:00:00 GMT

Hinweis

Verwenden Sie `_time32`, wenn Sie Zeitangaben über den 19. Januar 2038 hinaus benötigen.

Siehe auch

_gmtime32 (Seite 197)

_localtime32 (Seite 199)

asctime (Seite 206)

mktime (Seite 322)

time (Seite 404)

tzset (Seite 410)

3.5 abort

Funktion

Task beenden

Syntax

```
#include <stdlib.h>

void    abort (void);
```

Beschreibung

`abort` sendet das Signal SIGABRT zur aufrufenden Task. Beim Default–Signal–Handler wird die Task durch die CRUN–Funktion `exit` beendet.

Siehe auch

exit (Seite 242)

raise (Seite 339)

signal (Seite 362)

3.6 abs

Funktion

Absolutwert einer Integer-Zahl ermitteln

Syntax

```
#include <stdlib.h>

int    abs (int value);
```

Parameter

Parametername	Bedeutung
value	Integer–Zahl

Beschreibung

`abs` liefert den Absolutwert der in `value` angegebenen Integer–Zahl.

Rückgabewert

Rückgabewert	Bedeutung
Integer-Zahl	Absolutwert

Beispiel

```
main ()
{
 int x,y;
 x = abs (23); /* ergibt 23 */
 y = abs (-105); /* ergibt 105 */
}
```

Siehe auch

fabs (Seite 244)

labs (Seite 302)

3.7 access

Funktion

Zugriff auf eine Datei prüfen

Syntax

```
#include <io.h>

int    access (
         const char *pathname,
         int flags);
```

Parameter

Parametername	Bedeutung
pathname	Zeiger auf den Dateinamen
flags	Reserviert (muss 0 sein)

Beschreibung

`access` prüft, ob die mit `pathname` angegebene Datei im Inhaltsverzeichnis eingetragen ist. Der Parameter `flags` muss den Wert 0 besitzen.

Hinweis

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
0	Datei existiert
- 1	Datei wurde nicht gefunden oder Fehler

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt:

Fehlercode	Bedeutung
EAGAIN	Directory ist mit lock-Option geschützt
EINVAL	Syntaxfehler in `pathname`
EIO	Physikalischer Lese/Schreibfehler
EMFILE	Zu viele Dateien geöffnet
ENOENT	Datei existiert nicht
ENOMEM	Kein freier Speicherplatz verfügbar
ENOSPC	Datenträger ist voll
ENOTDIR	Pfadname enthielt ungültige Komponente, die kein Directory ist
ERANGE	Unerwarteter Fehlercode von HSFS

3.8 acos

Funktion

Arcuscosinus

Syntax

```
#include <math.h>

double    acos (double x);
```

Parameter

Parametername	Bedeutung
x	Double–Zahl

Beschreibung

`acos` berechnet den Arcuscosinus aus einer Double–Zahl. Das Ergebnis wird in Bogenmaß ausgegeben und liegt zwischen 0 und π. Das Argument der Funktion muss zwischen –1 und +1 liegen. Liegt es außerhalb, ruft `acos` die Fehlerbehandlungsfunktion `matherr` mit Exception–Typ DOMAIN auf und setzt `errno` auf EDOM. Der Rückgabewert ist in diesem Fall IND

Rückgabewert

Rückgabewert	Bedeutung
Double–Zahl	Arcuscosinus
IND	Nach Aufruf von `matherr` mit Exception-Typ DOMAIN wird `errno` auf EDOM gesetzt

Siehe auch

asin (Seite 208)

atan (Seite 210)

atan2 (Seite 211)

matherr (Seite 313)

3.9 asctime

Funktion

Datum und Uhrzeit der RMOS3-Softwareuhr in einen String umwandeln

Syntax

```
#include <time.h>

char    *asctime (struct tm *ptm);
```

Parameter

Parametername	Bedeutung
ptm	Zeiger auf Struktur vom Typ `tm`

Beschreibung

`asctime` wandelt Datum und Uhrzeit der RMOS3–Softwareuhr in einer Struktur vom Typ `tm` in einen 26 Zeichen langen String um und gibt einen Zeiger auf diesen String zurück.

Die Struktur `tm` ist in TIME.H folgendermaßen definiert:

```
struct tm{
 int tm_sec;   /* Sekunden (0-59)              */
 int tm_min;   /* Minuten (0-59)               */
 int tm_hour;  /* Stunden (0-23)               */
 int tm_mday;  /* Monatstag (1-31)             */
 int tm_mon;   /* Monat (0-11)                 */
 int tm_year;  /* Jahr minus 1900              */
 int tm_wday;  /* Wochentag (Sonntag = 0)      */
 int tm_yday;  /* laufender Tag im Jahr (0-365) */
 int tm_isdst; /* ungleich 0 falls Sommerzeit   */
};
```

Hinweis

Die Rückgabewerte zeigen auf einen Datenbereich, dessen Inhalt beim nächsten Aufruf überschrieben wird.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf String, der die Datums– und Zeitangabe enthält

Beispiel

Das folgende Beispiel zeigt den Einsatz der verschiedenen CRUN–Zeitfunktionen:

```
#include <time.h>
main ()
{
   struct tm *ptm;
   char *date;
   long tadd;

   putenv ("TZ=MEZ-1");

   time (&tadd);
   date = ctime (&tadd);
   printf ("today's date is %s", date);

   ptm = localtime (&tadd);
   date = asctime (ptm);
   printf ("today's date is %s", date);

   ptm = gmtime (&tadd);
   date = asctime (ptm);
   printf ("Greenwich mean time is %s", date);
}
```

Siehe auch

ctime (Seite 231)

gmtime (Seite 286)

localtime (Seite 305)

mktime (Seite 322)

tzset (Seite 410)

strftime (Seite 384)

3.10 asin

Funktion

Arcussinus

Syntax

```
#include <math.h>

double    asin (double x);
```

Parameter

Parametername	Bedeutung
x	Double–Zahl

Beschreibung

`asin` berechnet den Arcussinus aus einer Double–Zahl. Das Ergebnis wird in Bogenmaß ausgegeben und liegt zwischen –π/2 und +π/2. Das Argument der Funktion muss zwischen –1 und +1 liegen. Liegt es außerhalb, ruft `asin` die Fehlerbehandlungsfunktion `matherr` mit Exception–Typ DOMAIN auf und setzt `errno` auf EDOM. Der Rückgabewert ist in diesem Fall IND

Rückgabewert

Rückgabewert	Bedeutung
Double–Zahl	Arcussinus
IND	Nach Aufruf von `matherr` mit Exception–Typ DOMAIN wird `errno` auf EDOM gesetzt

Siehe auch

acos (Seite 205)

atan (Seite 210)

atan2 (Seite 211)

matherr (Seite 313)

3.11 assert

Funktion

Task, abhängig von einer Bedingung, abbrechen

Syntax

```
#include <assert.h>

void    assert (<condition>);
```

Parameter

Parametername	Bedeutung
<condition>	Boolesche Variable

Beschreibung

`assert` wertet die Bedingung `<condition>` aus. Bei FALSE (Wert = 0) wird eine Meldung auf die Standard–Fehlerausgabe ausgegeben, die den Text von `<condition>`, den Dateinamen (Wert des Makros __FILE__) und die Zeilennummer (Wert des Makros __LINE__) des aktuellen Makros `assert` enthält. Danach wird das Programm durch `abort` beendet

3.12 atan

Funktion

Arcustangens

Syntax

```
#include <math.h>

double    atan (double x);
```

Parameter

Parametername	Bedeutung
x	Double–Zahl

Beschreibung

`atan` berechnet den Arcustangens aus einer Double–Zahl. Das Ergebnis wird in Bogenmaß ausgegeben und liegt zwischen –π/2 und +π/2. Als Argumentbereich ist der gesamte Double–Zahlenbereich erlaubt

Rückgabewert

Rückgabewert	Bedeutung
Double–Zahl	Arcustangens

Siehe auch

acos (Seite 205)
asin (Seite 208)
atan2 (Seite 211)
matherr (Seite 313)

3.13 atan2

Funktion

Arcustangens von zwei Double–Zahlen

Syntax

```
#include <math.h>

double    atan2(double x, double y);
```

Parameter

Parametername	Bedeutung
x	Double–Zahl
y	Double–Zahl

Beschreibung

`atan2` berechnet den Arcustangens aus dem Quotienten zweier Double–Zahlen. Die Vorzeichen der Argumente werden zur Bestimmung des Quadranten verwendet. Das Ergebnis wird in Bogenmaß ausgegeben und liegt zwischen –π und +π. Sind beide Argumente 0, ruft `atan2` die Fehlerbehandlungsfunktion `matherr` mit Exception–Typ DOMAIN auf und setzt `errno` auf EDOM. Der Rückgabewert ist in diesem Fall IND

Rückgabewert

Rückgabewert	Bedeutung
Double–Zahl	Arcustangens des Quotienten
IND	Nach Aufruf von `matherr` mit Exception–Typ DOMAIN wird `errno` auf EDOM gesetzt

Siehe auch

acos (Seite 205)

asin (Seite 208)

atan (Seite 210)

matherr (Seite 313)

3.14 atexit

Funktion

Routinen festlegen, die bei der Beendigung einer Task aufgerufen werden sollen

Syntax

```
#include <stdlib.h>

int    atexit (void (_FIXED _FAR * func)(void));
```

Parameter

Parametername	Bedeutung
func	Zeiger auf eine Funktion mit Pascal (bzw. fixedparams) Schnittstelle. Bei Verwendung von CAD-UL oder Intel C muss die Funktion einen far Einsprung besitzen.

Beschreibung

Mit jedem Aufruf von `atexit` wird eine weitere Funktion `func` registriert. Die registrierten Funktionen (bis maximal 32) werden von der Funktion `exit` am Taskende aufgerufen. Registrierte Funktionen werden in umgekehrter Reihenfolge zu ihrer Eingabe abgearbeitet

Rückgabewert

Rückgabewert	Bedeutung
0	Funktionsregistrierung erfolgreich
≠0	Funktionsregistrierung nicht erfolgreich

Siehe auch

exit (Seite 242)

x_cr_killtsk (Seite 434)

x_cr_setexit (Seite 435)

3.15 atof

Funktion

String in eine Double-Gleitkomma-Zahl umwandeln

Syntax

```
#include <stdlib.h>

double    atof (const char *string);
```

Parameter

Parametername	Bedeutung
string	Zeiger auf einen String

Beschreibung

`atof` gibt als Ergebnis eine Double–Gleitkomma–Zahl zurück. Das Ergebnis entspricht dem Wert des Strings, auf den `string` zeigt. Der String wird abgesucht, solange gültige Werte gefunden werden. Führende Leerzeichen (definiert durch `isspace` in CTYPE.H) werden ignoriert. Die Gleitkomma–Zahl muss folgendes Format besitzen (in der angegebenen Reihenfolge):

- führende optionale Leerzeichen und/oder Tabulatoren
- ein optionales Vorzeichen
- eine Folge von Dezimalziffern, danach einen Dezimalpunkt und weitere Ziffern (zumindest einer der beiden Ziffernblöcke und der Dezimalpunkt müssen vorhanden sein).
- ein optionaler Exponent, der mit ‘e’ oder ‘E’ eingeleitet wird und aus optionalem Vorzeichen sowie einem obligatorischen Integerstring besteht

Hinweis

atof (string) ist äquivalent zu strtod (string, NULL)

Rückgabewert

Rückgabewert	Bedeutung
Double–Zahl	Double–Gleitkomma–Zahl
0.0	Keine Double–Zahl erzeugt

Fehlercodes

Im Fehlerfall wird `errno` auf folgenden Wert gesetzt:

Fehlercode	Bedeutung
ERANGE	Bei Überlauf oder Unterlauf. Bei Unterlauf gibt `atof` 0.0 zurück. Bei Überlauf HUGE_VAL mit entsprechendem Vorzeichen.

Siehe auch

isspace (Seite 299)

strtod (Seite 394)

3.16 atoi

Funktion

String in eine Integer-Zahl umwandeln

Syntax

```
#include <stdlib.h>

int    atoi (char *string);
```

Parameter

Parametername	Bedeutung
string	Zeiger auf einen String

Beschreibung

`atoi` gibt als Ergebnis eine Integer–Zahl zurück. Das Ergebnis entspricht dem Wert des Strings, auf den `string` zeigt. Der String wird abgesucht, bis ein Zeichen nicht mehr konsistent zur Basis 10 ist. Führende Leerzeichen (definiert durch `isspace` in CTYPE.H) werden ignoriert.

Hinweis

`atoi (string)` ist äquivalent zu `(int) strtol (string, (char **) 0, 10)`.

Überlauf–Bedingungen werden ignoriert.

Rückgabewert

Rückgabewert	Bedeutung
Integer–Zahl	Dezimal–Zahl

Siehe auch

atol (Seite 215)

isspace (Seite 299)

strtol (Seite 397)

strtoul (Seite 399)

3.17 atol

Funktion

String in einen Long-Integer umwandeln

Syntax

```
#include <stdlib.h>

long    atol (char *string);
```

Parameter

Parametername	Bedeutung
string	Zeiger auf einen String

Beschreibung

`atol` gibt als Ergebnis einen Long–Integer zurück. Das Ergebnis entspricht dem Wert des Strings, auf den `string` zeigt. Der String wird abgesucht, bis ein Zeichen nicht mehr konsistent zur Basis 10 ist. Führende Leerzeichen (definiert durch `isspace` in CTYPE.H) werden ignoriert.

Hinweis

`atol (string)` ist äquivalent zu `strtol (string, (char **) 0, 10)`.

Überlauf–Bedingungen werden ignoriert.

Rückgabewert

Rückgabewert	Bedeutung
Long–Integer	Dezimal–Zahl

Siehe auch

atoi (Seite 214)

strtol (Seite 397)

strtoul (Seite 399)

3.18 bsearch

Funktion

Element in einer sortierten Tabelle suchen

Syntax

```
#include <stdlib.h>

void    *bsearch(
            const void *search,
            const void *table,
            size_t count,
            size_t size,
            int (_FIXED _FAR *compar)
            (const void *p1,
            const void *p2));
```

Parameter

Parametername	Bedeutung
search	zeigt auf das Element, das gesucht werden soll
table	zeigt auf das erste Element der Tabelle
count	ist die Anzahl der Elemente in der Tabelle
size	ist die Größe eines Elements in Bytes
compar	ist der Name einer vom Anwender bereitgestellten Vergleichsfunktion. Diese wird mit zwei Argumenten aufgerufen, die auf die zu vergleichenden Elemente zeigen. `compar` muss eine Integer–Zahl zurückgeben, die kleiner, gleich oder größer 0 ist, je nachdem ob das erste Argument kleiner, gleich oder größer als das zweite ist. Die Vergleichsfunktion muss eine Pascal (bzw. fixedparams) Schnittstelle und bei Verwendung von CAD-UL oder Intel C einen far Einsprung besitzen.z.B.: `int _FIXED _FAR compar (const void*p1, const void*p2)`

Beschreibung

`bsearch` sucht ein bestimmtes Element in der mit `table` angegebenen Tabelle. Die Elemente der Tabelle müssen bereits in aufsteigender Reihenfolge sortiert sein, entsprechend einer vom Benutzer definierten Vergleichsfunktion `compar`.

Hinweis

`search` und `table` sollten vom Typ Pointer–to–Element sein und in den Typ Pointer–to–Character umgewandelt werden (casting).

Die Rückgabewerte sollten in den Typ Pointer–to–Element umgewandelt werden, auch wenn sie als Typ Pointer–to–Character deklariert sind. Die Tabellen–Elemente können beliebige Daten enthalten, da `compar` nicht unbedingt jedes Byte vergleichen muss.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf das gesuchte Element (Funktion erfolgreich)
NULL–Zeiger	Funktion nicht erfolgreich

Siehe auch

qsort (Seite 338)

3.19 calloc

Funktion

Speicher anfordern

Syntax

```
#include <malloc.h>

void    *calloc(
            size_t nmemb,
            size_t size);
```

Parameter

Parametername	Bedeutung
nmemb	Anzahl Elemente des Arrays
size	Länge eines Elements

Beschreibung

`calloc` fordert Speicher für ein Array von `nmemb` Elementen der Länge `size` an. Der Speicherplatz wird mit 0 initialisiert.

Hinweis

Der Speicherbereich wird entweder gezielt durch Aufruf der Funktion `free`, `RmFree` oder `RmFreeAll` freigegeben oder automatisch bei Beendigung der Task durch `exit`, `x_cr_killtsk`, `RmDeleteTask`, `RmEndTask` oder `RmKillTask`.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf einen Speicherplatz der angeforderten Größe
NULL–Zeiger	wenn `size` gleich 0 ist oder kein Speicherplatz verfügbar

Fehlercodes

Im Fehlerfall wird `errno` auf folgenden Wert gesetzt:

Fehlercode	Bedeutung
ENOMEM	Kein freier Speicherplatz verfügbar

Siehe auch

free (Seite 265)

malloc (Seite 312)

realloc (Seite 344)

RmFree (Seite 62)

RmFreeAll (Seite 63)

xinitc (Seite 436)

3.20 ceil

Funktion

Double-Zahl aufrunden

Syntax

```
#include <math.h>

double    ceil (double x);
```

Parameter

Parametername	Bedeutung
x	Double–Zahl

Beschreibung

`ceil` rundet das Argument auf die nächste ganze Double–Zahl auf.

Ist das Argument für `ceil` bereits ganzzahlig oder größer als die größte darstellbare ganze Zahl (2^53 – 1, wegen der 53–bit–Mantisse in Double–Zahlen), bleibt die Funktion ohne Wirkung.

Rückgabewert

Rückgabewert	Bedeutung
Double–Zahl	Aufgerundete Double–Zahl

Siehe auch

floor (Seite 256)

fmod (Seite 257)

3.21 changevib

Funktion

Beschreibungsblock eines Datenträgers ändern

Syntax

```
#include <io.h>

int    changevib (
           char *volume,
           VOL_DATA *new_vib);
```

Parameter

Parametername	Bedeutung
volume	Datenträgername (z.B. "C")
new_vib	Zeiger auf den neuen Beschreibungsblock des Datenträgers (Struktur vom Typ VOL_DATA, siehe `getvolumestatus`).

Beschreibung

Mit diesem Aufruf ist es möglich, einen statisch konfigurierten Beschreibungsblock eines Datenträgers dynamisch neu zu beschreiben. Der Aufruf darf nur auf einen Datenträger angewendet werden, der nicht angemeldet ist.

Hinweis

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich abgearbeitet
-1	Fehler

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt:

Fehlercode	Bedeutung
EACCESS	Der angegebene Datenträger ist angemeldet
ENOENT	Datenträger nicht gefunden
ERANGE	Unerwarteter Fehlercode vom HSFS

Siehe auch

createvib (Seite 230)

getvolumestatus (Seite 282)

3.22 chdir

Funktion

CWD (current working directory, aktuelles Verzeichnis) wechseln

Syntax

```
#include <direct.h>

int    chdir (char *path);
```

Parameter

Parametername	Bedeutung
path	Zeiger auf den Zugriffspfad des Verzeichnisses

Beschreibung

`chdir` stellt das mit `path` angegebene Verzeichnis als CWD (Current Working Directory, aktuelles Arbeitsverzeichnis) ein. Alle bei Funktionsaufrufen (z.B. `fopen`, `chdir`, ...) angegebenen Datei– oder Verzeichnisnamen beziehen sich auf dieses Verzeichnis, sofern nicht anders angegeben. Für `path` gelten die für Pfadnamen üblichen Regeln. Die maximale Länge des absoluten Pfadnamens des CWD ist in der Headerdatei DIRECT.H hinterlegte Wert CWD_MAX. Das CWD ist leer, bis eine Task `chdir` aufruft, d.h. es müssen absolute Pfadnamen (einschließlich Laufwerkbezeichnung) angegeben werden; dies gilt auch beim ersten Aufruf von `chdir`.

Hinweis

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich
-1	Das CWD wurde nicht verändert

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt (bei einem Fehler vom HSFS, wird dieser zusätzlich in `errno2` abgelegt):

Fehlercode	Bedeutung
E2BIG	Der absolute Pfadname des neuen CWD ist länger als CWD_MAX Zeichen
EACCESS	Auf eine Komponente von `path` darf nicht zugegriffen werden
EAGAIN	Das durch `path` angegebene Verzeichnis ist mit LOCK–Option geöffnet
EBUSY	Im (noch bestehenden) CWD ist noch eine Datei geöffnet; das CWD wurde nicht gewechselt
EINVAL	Syntaxfehler in `path`
EIO	Physikalischer Lesefehler
EMFILE	Zu viele geöffnete Dateien oder anderer Grenzwert des HSFS erreicht
ENOENT	Eine Komponente von `path` existiert nicht oder `path` ist ungültig
ENOTDIR	Eine Komponente von `path` ist kein Verzeichnis
ERANGE	Ungültiger Fehlercode des HSFS beim Öffnen des neuen CWD

Siehe auch

getcwd (Seite 278)

3.23 checkpoint

Funktion

Rückschreiben der (HSFS) Zwischenpuffer einer Datei

Syntax

```
#include <io.h>

int    checkpoint (int handle);
```

Parameter

Parametername	Bedeutung
handle	Deskriptor (Rückgabewert von `open`)

Beschreibung

Die Funktion `checkpoint` erzwingt ein Aktualisieren der Datei auf dem Datenträger, indem veränderte Bereiche der Datei, die bisher im Zwischenpuffer des HSFS gehalten wurden, auf den entsprechenden Datenträger geschrieben werden.

Hinweis

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich abgearbeitet
-1	Fehler

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt:

Fehlercode	Bedeutung
EBADF	Der Parameter `handle` ist kein gültiger Deskriptor
EINVAL	Die Anwendung der Funktion checkpoint auf Socket-Deskriptor ist nicht zulässig.
EIO	Physikalischer I/O Fehler
ERANGE	Unerwarteter Fehlercode vom HSFS

Siehe auch

fflush (Seite 251)

open (Seite 325)

3.24 chmod

Funktion

Attribute einer Datei ändern

Syntax

```
#include <io.h>

int    chmod (
         char *filename,
         int pmode);
```

Parameter

Parametername	Bedeutung
filename	Name der Datei, deren Attribute gesetzt werden sollen.
pmode	Neue Attribute für die angegebene Datei. Folgende Werte sind möglich: ATR_SYSTEM Systemdatei (ohne Bedeutung bei RMOS3) ATR_HIDDEN verborgen (ohne Bedeutung bei RMOS3) ATR_RDONLY schreibgeschützt ATR_DIREC Verzeichnis (darf nur bei Verzeichnis angegeben werden) ATR_LABEL Datenträgerlabel ATR_ARCH Archivbit, wird bei jeder Modifikation der Datei gesetzt ATR_DEVICE Datei ist eine Gerätetreiberdatei Die Angabe mehrerer Attribute erfolgt durch ODER-Verknüpfung der einzelnen Attribute.

Beschreibung

`chmod` setzt die in `pmode` angegebenen Dateiattribute für die in `filename` angegebene Datei.

Hinweis

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich abgearbeitet
-1	Fehler

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt:

Fehlercode	Bedeutung
EACCESS	Ungültige Zugriffsrechte für `filename`
EBUSY	`filename` ist bereits mit lock option geöffnet
EIO	physikalischer I/O Fehler
EMFILE	interne HSFS Limits erreicht
ENOENT	`filename` ungültig
ENOSPC	Datenträger ist voll
ERANGE	Unerwarteter Fehlercode vom HSFS

Siehe auch

search (Seite 353)

3.25 clearerr

Funktion

Fehlerstatus eines Streams zurücksetzen

Syntax

```
#include <stdio.h>

void    clearerr(FILE *stream);
```

Parameter

Parametername	Bedeutung
stream	Stream mit Struktur vom Typ FILE

Beschreibung

`clearerr` löscht den Fehler–Status, der dem angegebenen `stream` zugeordnet ist.

Siehe auch

feof (Seite 249)
ferror (Seite 250)
fileno (Seite 255)

3.26 clock

Funktion

Dummy-Funktion

Syntax

```
#include <time.h>

long    clock (void);
```

Beschreibung

`clock` ist dafür vorgesehen, die seit Taskstart vergangene Zeit zu ermitteln. In der CRUN ist `clock` eine Dummy–Funktion, die entprechend dem ANSI–Standard den Wert –1 zurückgibt.

Rückgabewert

Rückgabewert	Bedeutung
-1	Funktion ist Dummy–Funktion

3.27 close

Funktion

Offene Datei schließen

Syntax

```
#include <io.h>

int    close (int handle);
```

Parameter

Parametername	Bedeutung
handle	Deskriptor

Beschreibung

`close` schließt die mit `handle` verbundene Datei.

Hinweis

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich
-1	Funktion nicht erfolgreich

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt (wurde ein Fehler vom HSFS gemeldet, wird dieser zusätzlich in `errno2` abgelegt):

Fehlercode	Bedeutung
EAGAIN	Dieser Fehler entspricht den RMOS3–HSFS Fehlermeldungen (-16) und (-22) (tritt bei nicht erlaubten Inhaltsverzeichnisoperationen auf)
EBADF	Ungültiger Deskriptor
EIO	Physikalischer Schreib/Lesefehler
ENOMEM	Es konnte nicht ausreichend Speicher aus dem angegebenen Pool entnommen werden.
ERANGE	Interner RMOS3–Fehler

Siehe auch

fclose (Seite 245)

open (Seite 325)

write (Seite 426)

3.28 cos

Funktion

Cosinus

Syntax

```
#include <math.h>

double    cos (double x);
```

Parameter

Parametername	Bedeutung
x	Double-Zahl

Beschreibung

`cos` berechnet den Cosinus aus einer Double–Zahl.

Das Argument bei `cos` wird in Bogenmaß angegeben. Bei einem Betrag des Arguments größer als 2^52 ruft cos die Fehlerbehandlungsfunktion `matherr` mit Exception–Typ TLOSS auf, die als Ergebnis IND zurückliefert. Bei dieser Größe des Arguments sind im numerischen Format darstellbare aufeinanderfolgende Double–Zahlen um 0,5 voneinander verschieden. Durch die Genauigkeit des Arguments wird also nur noch der Quadrant festgelegt. Wäre das Argument noch größer, so ginge auch diese Information verloren.

`errno` wird nicht gesetzt, da dies laut ANSI-Norm nicht erforderlich ist.

Rückgabewert

Rückgabewert	Bedeutung
Double–Zahl	Cosinus
IND	Nach Aufruf von `matherr` mit Exception–Typ TLOSS (Betrag des Arguments ≥ 2^52)

Siehe auch

matherr (Seite 313)

sin (Seite 367)

tan (Seite 402)

3.29 cosh

Funktion

Cosinus hyperbolicus

Syntax

```
#include <math.h>

double    cosh (double x);
```

Parameter

Parametername	Bedeutung
x	Double-Zahl

Beschreibung

`cosh` berechnet den Cosinus hyperbolicus (`cosh(x) = (e^x + e^-x)/2)` einer Double–Zahl.

Bei einem Überlauf ruft `cosh` die Fehlerbehandlungsfunktion `matherr` mit Exception–Typ OVERFLOW auf und setzt `errno` auf ERANGE. Der Rückgabewert ist in diesem Fall +HUGE_VAL.

Rückgabewert

Rückgabewert	Bedeutung
Double–Zahl	Cosinus hyperbolicus
+HUGE_VAL	Nach Aufruf von `matherr` mit Exception–Typ DOMAIN wird `errno` auf ERANGE (Bereichsüberlauf) gesetzt

Siehe auch

matherr (Seite 313)

sinh (Seite 368)

tanh (Seite 403)

3.30 createvib

Funktion

Neuen Beschreibungsblock eines Datenträgers erzeugen

Syntax

```
#include <io.h>

int    createvib (
           VOL_DATA *new_vib,
           int pool_id);
```

Parameter

Parametername	Bedeutung
new_vib	Zeiger auf den Beschreibungsblock des Datenträgers (Struktur vom Typ VOL_DATA, siehe `getvolumestatus`)
pool_id	Speicherpool, aus dem die notwendigen Datenbereiche angefordert werden sollen.

Beschreibung

Mit dem Aufruf `createvib` kann dynamisch ein neuer Beschreibungsblock eines Datenträgers erzeugt werden. Der Datenträger–Name darf nicht mit dem Namen schon bestehender Datenträger übereinstimmen.

Für den Beschreibungsblock werden 160 Bytes benötigt. Wird beim Parameter `fat_buf` der Struktur VOL_DATA ein NULL–Zeiger übergeben, wird die Größe des Puffers aus den Bootsektordaten ermittelt.

Hinweis

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich abgearbeitet
-1	Fehler

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt:

Fehlercode	Bedeutung
EEXIST	Name des neuen Datenträgers ist HSFS bereits bekannt.
EINVAL	Ungültiger Parameter
ENOMEM	Es konnte nicht ausreichend Speicher aus dem angegebenen Pool entnommen werden.
ERANGE	Unerwarteter Fehlercode vom HSFS

Siehe auch

changevib (Seite 219)

getvolumestatus (Seite 282)

3.31 ctime

Funktion

Datum und Uhrzeit in einen String umwandeln

Syntax

```
#include <time.h>

char    *ctime (time_t *clock);
```

Parameter

Parametername	Bedeutung
clock	Zeiger auf ein Element vom Typ `time_t`

Beschreibung

`ctime` wandelt einen Long–Integer (wie er von `time` geliefert wird) - unter Berücksichtigung der Zeitzone und gegebenenfalls Verschiebungen durch die Sommerzeit - in einen String um und liefert als Ergebnis einen Zeiger auf ein 26 Byte langes statisches Datenfeld im Format "Wochentag Monat Tag Stunden:Minuten:Sekunden Jahr\n".

Beispiel: `Mon Jan 01 12.30.45 1990\n`

Hinweis

Die Rückgabewerte zeigen auf einen Datenbereich, dessen Inhalt beim nächsten Aufruf überschrieben wird.

`ctime(clock)` ist äquivalent zu `asctime(localtime(clock))`.

Verwenden Sie `_ctime32`, wenn Sie Zeitangaben über den 19. Januar 2038 hinaus benötigen.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf einen String, der Datums– und Zeitangabe enthält

Beispiel

Das folgende Beispiel zeigt den Einsatz der verschiedenen CRUN–Zeitfunktionen:

```
#include <time.h>
main () {
         struct tm *ptm;
         char *date;
         long tadd;

         putenv ("TZ=MEZ-1");

         time (&tadd);
         date = ctime (&tadd);
         printf ("today's date is %s", date);

         ptm = localtime (&tadd);
         date = asctime (ptm);
         printf ("today's date is %s", date);

         ptm = gmtime (&tadd);
         date = asctime (ptm);
         printf ("Greenwich mean time is %s", date);
}
```

Siehe auch

asctime (Seite 206)
gmtime (Seite 286)
localtime (Seite 305)
mktime (Seite 322)
time (Seite 404)
tzset (Seite 410)
strftime (Seite 384)

3.32 difftime

Funktion

Länge eines Zeitraums ermitteln

Syntax

```
#include <time.h>

double    difftime(
              time_t time1,
              time_t time2);
```

Parameter

Parametername	Bedeutung
time1	Anfang des Zeitraums
time2	Ende des Zeitraums

Beschreibung

`difftime` liefert die Differenz zwischen `time1` und `time2` als Double–Zahl zurück.

Rückgabewert

Rückgabewert	Bedeutung
Double–Zahl	Anzahl Sekunden von `time1` bis `time2`

Siehe auch

time (Seite 404)

3.33 dismount

Funktion

Gerät beim HSFS-Dateisystem abmelden

Syntax

```
#include <io.h>

int    dismount (const char *volume_name);
```

Parameter

Parametername	Bedeutung
volume_name	Laufwerkname

Beschreibung

`dismount` meldet ein Gerät beim HSFS–Dateisystem ab. `volume_name` muss ein gültiger Laufwerkname mit einer Länge von 1 oder 2 Zeichen sein (z.B. "A", "C", "M0").

Hinweis

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich
-1	Funktion nicht erfolgreich

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt:

Fehlercode	Bedeutung
EACCES	Es existieren noch offene Dateien oder Laufwerk nicht angemeldet.
EIO	Physikalischer Schreib/Lesefehler
ERANGE	Unerwarteter Fehlercode von HSFS

Siehe auch

mount (Seite 324)

3.34 div

Funktion

Integer–Division

Syntax

```
#include <stdlib.h>

div_t    div(
          int i1,
          int i2);
```

Parameter

Parametername	Bedeutung
i1	Integer–Zahl (Zähler)
i2	Integer–Zahl (Nenner)

Beschreibung

`div` liefert das Ergebnis der Division `i1/i2`. Dabei wird in `div_t.quot` der Quotient und in `div_t.rem` der Divisionsrest abgelegt.

`div_t` ist folgendermaßen deklariert:

```
typedef struct    {int quot;
                   int rem;
                  }div_t;
```

Rückgabewert

Rückgabewert	Bedeutung
div_t	Struktur mit Quotient und Divisionsrest.

Siehe auch

ldiv (Seite 304)

3.35 duread

Funktion

Zeichen über einen RMOS3-Treiber lesen

Syntax

```
#include <io.h>

size_t    duread(
            int device,
            int unit,
            void *buf,
            size_t nbytes);
```

Parameter

Parametername	Bedeutung
device	Device–ID des Gerätetreibers
unit	Unit–ID des Gerätetreibers
buf	Empfangspuffer
nbytes	Anzahl der zu übertragenden Bytes

Beschreibung

`duread` übergibt einen Leseauftrag an den mit `device`/`unit` angegebenen RMOS3–Treiber (z.B. BYT–Treiber). Der Treiber beendet den Leseauftrag, wenn `nbytes` Zeichen eingelesen sind oder sich ein CR in der Eingabe befindet. Der Treiber–Aufruf erfolgt mit Warten auf Beendigung.

Der angegebene Treiber muss eine zum BYT-Treiber kompatible Aufrufschnittstelle besitzen.

Hinweis

CRUN wandelt die übertragenden Daten nicht um.

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
> 0	Systemaufruf erfolgreich beendigt (der Wert gibt die Anzahl der tatsächlich gelesenen Bytes an)
- 1	Systemaufruf nicht erfolgreich

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt:

Fehlercode	Bedeutung
EBUSY	Auftrag wurde abgebrochen (Cancel)
EIO	Unerwarteter Interrupt
ENXIO	Kombination von device/unit ungültig
ERANGE	Interner RMOS3–Fehler

Siehe auch

duwrite (Seite 238)

3.36 duwrite

Funktion

Zeichen über einen RMOS3-Treiber schreiben

Syntax

```
#include <io.h>

size_t    duwrite(
             int device,
             int unit,
             void *buf,
             size_t nbytes);
```

Parameter

Parametername	Bedeutung
device	Device–ID des Gerätetreibers
unit	Unit–ID des Gerätetreibers
buf	Empfangspuffer
nbytes	Anzahl der zu übertragenden Bytes

Beschreibung

`duwrite` dient zur Ausgabe von Bytes über den mit `device`/`unit` angegebenen RMOS3–Treiber (z.B. BYT–Treiber). Die Bytes werden vom Puffer `buf` aus übertragen. Zu beachten ist, dass der Treiber nicht alle Zeichen ausgegeben kann. Das Zeichen '\0' führt zum vorzeitigen Abbruch der Ausgabe. Der Treiber–Aufruf erfolgt mit Warten auf Beendigung.

Der angegebene Treiber muss eine zum BYT-Treiber kompatible Aufrufschnittstelle besitzen.

Hinweis

CRUN wandelt die übertragenden Daten nicht um.

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
> 0	Systemaufruf erfolgreich beendigt (der Wert gibt die Anzahl der tatsächlich geschriebenen Bytes an)
- 1	Systemaufruf nicht erfolgreich

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt:

Fehlercode	Bedeutung
EBUSY	Auftrag wurde abgebrochen (Cancel)
EIO	Unerwarteter Interrupt
ENXIO	Kombination von `device/unit` ungültig
ERANGE	Interner RMOS3–Fehler

Siehe auch

duread (Seite 236)

3.37 errno, errno2

Funktion

Fehlernummern der Systemaufrufe und Bibliotheksfunktionen

Syntax

```
#include <errno.h>

extern    int errno;
extern    int errno2;
```

Beschreibung

Eine bei Systemaufrufen und Bibliotheksfunktionen auftretende Fehlerkennung, die zum symbolischen Namen einer Fehlermeldung gehört, wird der externen Variablen `errno` zugewiesen.

`errno` sollte nur dann überprüft werden, wenn der Rückgabewert eines Systemaufrufs einen Fehler anzeigt und die Definition des Aufrufs besagt, dass `errno` die Fehlernummer angibt. Der Wert 0 für `errno` ist reserviert und kann in keiner Fehlermeldung vorkommen. Ein Anwender–Programm, das den Wert von `errno` überprüft, muss die Headerdatei ERRNO.H enthalten.

Bedeutung der Fehlermeldungen (–nummern):

Fehlermeldung	Nr.	Bedeutung
E2BIG	26	Die Argument–Liste ist zu lang.
EACCES	27	Keine Berechtigung! Es wurde ein Versuch unternommen, auf eine Datei zuzugreifen, wobei die Schutz–Mechanismen des Betriebssystems diese Art des Zugriffs verbieten.
EAGAIN	1	Betriebsmittel vorübergehend nicht verfügbar, später noch einmal versuchen.
EBADF	28	Falscher Deskriptor! Entweder bezieht sich ein Deskriptor auf eine Datei bzw. Unit, die nicht geöffnet ist, oder eine Datei bzw. Unit soll gelesen (beschrieben) werden, die nur zum Schreiben (Lesen) geöffnet ist.
EBUSY	2	Gerät oder Betriebsmittel belegt! Es wurde versucht, ein Gerät in ein Dateisystem einzuhängen, das bereits eingehängt ist, oder ein Gerät auszuhängen, auf dem eine Datei in Bearbeitung ist (z.B. eine geöffnete Datei oder das aktuelle Verzeichnis).
EDOM	31	Mathematisches Argument! Das Argument einer Funktion aus der Bibliothek der mathematischen Funktionen liegt außerhalb des zulässigen Wertebereichs.
EEXIST	32	Das Objekt existiert bereits.
EFBIG	33	Datei zu groß! Eine Datei überschreitet die maximale Dateigröße.
EINVAL	5	Ungültiger Parameterwert!
EIO	36	Ein–/Ausgabefehler! Ein physikalischer Ein–/Ausgabefehler ist aufgetreten. Dieser Fehler kann nicht sofort gemeldet werden, sondern erst nach dem nächsten Aufruf, der auf den verursachenden Aufruf folgt.
EISDIR	38	Verzeichnis! Es wurde versucht, in ein Verzeichnis zu schreiben.

Fehlermeldung	Nr.	Bedeutung
EMFILE	37	Zu viele geöffnete Dateien!
ENODEV	38	Kein solches Gerät vorhanden! Versuch einer unzulässigen Operation auf einem Gerät (z.B. Lesen von einem Gerät, auf das nur geschrieben werden kann).
ENOENT	39	Keine Datei oder kein Verzeichnis dieses Namens vorhanden! Ein nicht existenter Dateiname wurde angegeben oder eines der Verzeichnisse des Pfadnamens existiert nicht.
ENOMEM	42	kein freier Speicherplatz verfügbar!
ENOSPC	44	Kein Platz mehr auf dem Gerät! Zum Schreiben einer Datei bzw. zum Anlegen eines Verzeichnisses ist kein freier Platz mehr auf dem Gerät vorhanden.
ENOTFOUND	51	Objekt nicht gefunden.
ENOTDIR	45	Kein Verzeichnis! Es wurde kein Verzeichnis angegeben, obwohl dies erforderlich ist.
ERANGE	49	Nach arithmetischen Funktionen: Ergebnis zu lang oder undefiniert! Der Wert einer Funktion aus der Bibliothek der mathematischen Funktionen kann nicht innerhalb der Maschinen–Genauigkeit dargestellt werden. Nach Ein–/Ausgabe–Operationen: Vom HSFS wurde ein unbekannter Fehlercode zurückgeliefert.

Fehlerbehandlung mit `errno2`:
Falls bei einem Funktionsaufruf eine Fehlernummer (`errno`) ungleich Null zurückgegeben wurde, kann eventuell eine detaillierte Fehlerüberprüfung erfolgen. Dies betrifft alle Funktionen, die Ein–/Ausgabe–Operationen über das RMOS3–Dateisystem oder den BYT–Treiber ausführen. In diesem Fall wird zusätzlich eine Variable `errno2` initialisiert. Diese Variable enthält dann den vom RMOS3-Dateisystem bzw. von der RMOS3 Funktion `RmIO` übergebenen Statuswert.

Hinweis

`errno` ist kompatibel zu ANSI C.
`errno2` ist eine Erweiterung des ANSI C Sprachumfanges.

Beispiel

```
#include <errno.h>

errno = 0;
errno2 = 0;
fclose(fp);
if (errno ! = 0)
{
    printf("errno: %d, errno2: %d\n", errno, errno2);
}
```

Siehe auch

perror (Seite 327)

strerror (Seite 383)

sys_errlist, sys_nerr (Seite 401)

3.38 exit

Funktion

Task bereinigen und mit festgesetztem Status beenden

Syntax

```
#include <stdlib.h>

void    exit (int status);
```

Parameter

Parametername	Bedeutung
status	Integer–Zahl

Beschreibung

`exit` beendet die aufrufende Task (Zustand DORMANT). Tasks, die vom CLI gestartet wurden, werden gelöscht (Zustand NICHT EXISTENT).

Vor dem Beenden der Task werden noch Bereinigungsaktionen durchgeführt:

Abarbeiten der Routinen, die mit `atexit` installiert wurden.

Alle durch `fopen` und `fduopen` geöffneten Streams werden geschlossen. Mit `open` eröffnete Dateien werden nicht geschlossen, außer sie wurden mit dem Flag O_CLOSEONEXIT geöffnet.

Zuvor durch `malloc`, `calloc` und `realloc` angeforderter Speicher wird zurückgegeben. Außerdem wird der Speicher freigegeben, der durch `RmAlloc` und `RmReAlloc` angefordert wurde, außer es wurde der Mode RM_NOAUTOFREE angegeben.

Unter RMOS3 wird `status` nur bei Programmen ausgewertet, die vom CLI gestartet wurden.

Siehe auch

atexit (Seite 212)

x_cr_killtsk (Seite 434)

x_cr_setexit (Seite 435)

3.39 exp

Funktion

Exponentialfunktion

Syntax

```
#include <math.h>

double    exp (double x);
```

Parameter

Parametername	Bedeutung
x	Double-Zahl

Beschreibung

`exp` berechnet e^x. e ist die Eulersche Zahl 2,71... und die Basis des natürlichen Logarithmus. Das Argument und der Rückgabewert sind Double–Zahlen. Liegt das Ergebnis der Funktion außerhalb des darstellbaren Zahlenbereichs, ruft `exp` die Fehlerbehandlungsfunktion `matherr` mit Exception–Typ OVERFLOW auf und setzt `errno` auf ERANGE. Bei einem zu kleinem Ergebnis wird `matherr` mit UNDERFLOW aufgerufen und `errno` ebenfalls auf ERANGE gesetzt.

Rückgabewert

Rückgabewert	Bedeutung
Double–Zahl	e^x
IND	Nach Aufruf von `matherr` mit Exception–Typ UNDERFLOW wird `errno` auf ERANGE gesetzt (e^x zu klein)
+HUGE_VAL	Nach Aufruf von `matherr` mit Exception–Typ OVERFLOW wird `errno` auf ERANGE gesetzt (e^x außerhalb des darstellbaren Zahlenbereichs)

Siehe auch

pow (Seite 328)

3.40 fabs

Funktion

Absolutwert einer Double-Zahl ermitteln

Syntax

```
#include <math.h>

double    fabs (double value);
```

Parameter

Parametername	Bedeutung
value	Double-Zahl

Beschreibung

`fabs` liefert den Absolutwert der in `value` angegebenen Double–Zahl.

Rückgabewert

Rückgabewert	Bedeutung
Double–Zahl	Absolutwert

Beispiel

```
main ()
{
   double z;
   z = fabs(-7.28); /* ergibt 7.28 */
}
```

Siehe auch

abs (Seite 203)

labs (Seite 302)

3.41 fclose

Funktion

Stream schließen

Syntax

```
#include <stdio.h>

int    fclose (FILE *stream);
```

Parameter

Parametername	Bedeutung
stream	Stream mit Struktur vom Typ FILE

Beschreibung

`fclose` schließt die Datei, die dem mit `stream` angegebenen Stream zugeordnet ist. Die Funktion wird bei `exit` automatisch auf alle noch geöffneten Dateien angewendet.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich
EOF	Funktion nicht erfolgreich

Siehe auch

close (Seite 227)

fopen (Seite 258)

3.42 fduopen

Funktion

Stream auf RMOS3-Treiber öffnen

Syntax

```
#include <stdio.h>

FILE    *fduopen(
           int device,
           int unit,
           const char *type);
```

Parameter

Parametername	Bedeutung
device	Device–ID des Gerätetreibers
unit	Unit–ID des Gerätetreibers
type	Lese–/Schreibmodus; gültige Werte sind: `"r"` Öffnen zum Lesen im Textmodus `"w"` öffnen zum Schreiben im Textmodus `"rb"` öffnen zum Lesen im Binärmodus `"wb"` öffnen zum Schreiben im Binärmodus

Beschreibung

`fduopen` öffnet einen Stream. Alle Lese– oder Schreibzugriffe (abhängig von `*type`) werden auf den mit `device`/`unit` angegebenen RMOS3–Treiber umgeleitet. Beachten Sie, dass der RMOS3–Treiber eventuell nicht alle Zeichen (z.B. '\0') ausgeben kann.

Der angegebene Treiber muss eine zum BYT-Treiber kompatible Aufrufschnittstelle besitzen.

Hinweis

Im Textmodus wird beim Schreiben '\n' in '\r\n' umgewandelt. Beim Lesen wird '\r\n' in '\n' umgewandelt. Im Binärmodus erfolgt keine Umwandlung.

Der Stream ist nur für die aufrufende Task gültig und darf nur von ihr benutzt werden. Stream-Pointer und Deskriptoren sind taskspezifisch und können nicht zwischen Tasks ausgetauscht werden.

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf einen Stream (Funktion erfolgreich)
NULL–Zeiger	Funktion nicht erfolgreich

Siehe auch

fopen (Seite 258)

3.43 fdureopen

Funktion

Stream auf einen Treiber umlenken

Syntax

```
#include <stdio.h>

FILE    *fdureopen(
            int device,
            int unit,
            char *mode,
            FILE *stream);
```

Parameter

Parametername	Bedeutung
device	Device–ID des Gerätetreibers, auf den der Stream umgelenkt werden soll
unit	Unit–ID des Gerätetreibers, auf den der Stream umgelenkt werden soll
mode	Lese–/Schreibmodus; gültige Werte sind: `"r"` Öffnen zum Lesen im Textmodus `"w"` Öffnen zum Schreiben im Textmodus `"rb"` Öffnen zum Lesen im Binärmodus `"wb"` Öffnen zum Schreiben im Binärmod.
stream	Stream, der zu dem mit `device/unit` angegebenen Treiber umgelenkt werden soll

Beschreibung

`fdureopen` schließt die Datei, die `stream` zugeordnet ist, öffnet eine Verbindung zu dem mit `device/unit` angegebenen RMOS3–Treiber und ordnet diese Verbindung `stream` zu.

Der angegebene Treiber muss eine zum BYT-Treiber kompatible Aufrufschnittstelle besitzen.

Hinweis

Stream-Pointer und Deskriptoren sind taskspezifisch und können nicht zwischen Tasks ausgetauscht werden.

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
stream	Funktion erfolgreich
NULL–Zeiger	Funktion nicht erfolgreich

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt:

Fehlercode	Bedeutung
EBADF	`stream` gibt keinen gültigen Stream an
EINVAL	Ungültiger Parameter (`mode` nicht zulässig)

Siehe auch

fduopen (Seite 246)

freopen (Seite 266)

3.44 feof

Funktion

Stream-Status nach EOF abfragen

Syntax

```
#include <stdio.h>

int    feof (FILE *stream);
```

Parameter

Parametername	Bedeutung
stream	Stream mit Struktur vom Typ FILE

Beschreibung

`feof` überprüft, ob beim Lesen des Stream das Dateiende (EOF) erreicht wurde.

Rückgabewert

Rückgabewert	Bedeutung
0	Dateiende wurde nicht erreicht
≠0	Dateiende wurde erreicht

Siehe auch

clearerr (Seite 225)

ferror (Seite 250)

fileno (Seite 255)

3.45 ferror

Funktion

Stream-Status nach E/A-Fehler abfragen

Syntax

```
#include <stdio.h>

int    ferror (FILE *stream);
```

Parameter

Parametername	Bedeutung
stream	Stream mit Struktur vom Typ FILE

Beschreibung

`ferror` überprüft, ob Ein– oder Ausgabefehler bei einem Lese– oder Schreibzugriff auf den mit `stream` angegebenen Stream aufgetreten sind.

Rückgabewert

Rückgabewert	Bedeutung
0	Kein Fehler
≠0	Fehler

Siehe auch

clearerr (Seite 225)

feof (Seite 249)

fileno (Seite 255)

3.46 fflush

Funktion

Stream-Puffer leeren

Syntax

```
#include <stdio.h>

int    fflush (FILE *stream);
```

Parameter

Parametername	Bedeutung
stream	Stream mit Struktur vom Typ FILE

Beschreibung

`fflush` leert den dem `stream` zugeordneten Puffer. Ist das Argument von `fflush` ein Ausgabe–Stream, werden die dazugehörigen Puffer ausgegeben. Handelt es sich um einen Eingabe–Stream, wird der Puffer geleert. Wird für `stream` NULL übergeben, werden die Puffer aller geöffneten Streams der aufrufenden Task ausgegeben bzw. geleert. `fflush` schließt die Datei nicht.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich
EOF	Funktion nicht erfolgreich

Siehe auch

close (Seite 227)

fclose (Seite 245)

fopen (Seite 258)

setbuf (Seite 359)

3.47 fgetc

Funktion

Zeichen aus einem Stream lesen

Syntax

```
#include <stdio.h>

int    fgetc (FILE *stream);
```

Parameter

Parametername	Bedeutung
stream	Stream mit Struktur vom Typ FILE

Beschreibung

`fgetc` liefert das nächste Zeichen im Puffer des mit `stream` bezeichneten Streams und setzt den Dateizeiger in diesem Stream auf das darauffolgende Zeichen.

Hinweis

Wird ein von `fgetc` zurückgegebener Integerwert als Character–Variable gespeichert und mit der Integer–Konstanten EOF verglichen, kann die Vergleichsoperation mit einem Fehler enden, denn das Hinzufügen eines Vorzeichens bei Umwandlung von Character auf Integer ist maschinenabhängig.

Im Textmodus wird ‘\r\n’ in ‘\n’ umgewandelt.

Rückgabewert

Rückgabewert	Bedeutung
Zeichen	Funktion erfolgreich
EOF	Dateiende oder Fehler

Siehe auch

fopen (Seite 258)

fread (Seite 264)

gets (Seite 280)

putc (Seite 333)

scanf (Seite 350)

3.48 fgetpos

Funktion

Momentane Position in einer Datei zurückliefern

Syntax

```
#include <stdio.h>

int    fgetpos(
          FILE *stream,
          fpos_t *pos);
```

Parameter

Parametername	Bedeutung
stream	Stream mit Struktur vom Typ FILE
pos	Dateizeiger

Beschreibung

`fgetpos` speichert die momentane Position in der durch `stream` angegebenen Datei in der mit `pos` übergebenen Variablen.

Hinweis

Bei im Textmodus geöffneten Streams ist diese Funktion nur eingeschränkt nutzbar (wegen der Umwandlung von '\n' bzw. '\r\n'). Diese Funktion darf nur dazu verwendet werden, um die Position in einem Stream per `fgetpos` zu ermitteln und später durch `fsetpos` wieder auf diese Stelle zu positionieren.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich
≠0	Funktion nicht erfolgreich

Siehe auch

fseek (Seite 272)

fsetpos (Seite 273)

3.49 fgets

Funktion

String aus einem Stream lesen

Syntax

```
#include <stdio.h>

char    *fgets(
          char *s,
          int n,
          FILE *stream);
```

Parameter

Parametername	Bedeutung
s	Zeiger auf Ziel–String
n	Anzahl Zeichen, nach der der Leseauftrag abgebrochen wird
stream	Stream mit Struktur vom Typ FILE

Beschreibung

`fgets` liest den nächsten String bis CR , `n` oder EOF vom Standard–Eingabe–Stream in den mit `s` angegebenen String. CR wird durch ein Null–Byte ersetzt.

Hinweis

Im Textmodus wird ‘\r\n’ in ‘\n’ umgewandelt.

Rückgabewert

Rückgabewert	Bedeutung
s	Zeiger auf Ziel–String (Funktion erfolgreich)
NULL–Zeiger	Keine Zeichen übertragen (z.B. wenn EOF vor dem Lesen von Zeichen erreicht wurde) oder Lesefehler bei Zugriff auf eine Datei, die zum Lesen geöffnet ist

Siehe auch

fopen (Seite 258)

fread (Seite 264)

getc (Seite 276)

gets (Seite 280)

scanf (Seite 350)

3.50 fileno

Funktion

Deskriptor ermitteln

Syntax

```
#include <stdio.h>

int    fileno (FILE *stream);
```

Parameter

Parametername	Bedeutung
stream	Stream mit Struktur vom Typ FILE

Beschreibung

`fileno` liefert den Deskriptor, der dem angegebenen `stream` zugeordnet ist.

Hinweis

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
Deskriptor	der `stream` zugeordneten Datei

Siehe auch

clearerr (Seite 225)

feof (Seite 249)

ferror (Seite 250)

3.51 floor

Funktion

Double-Zahl abrunden

Syntax

```
#include <math.h>

double    floor (double x);
```

Parameter

Parametername	Bedeutung
x	Double–Zahl

Beschreibung

`floor` rundet das Argument auf die nächste ganze Double–Zahl ab.

Ist das Argument für floor bereits ganzzahlig oder größer als die größte darstellbare ganze Zahl (2^53 – 1, wegen der 53–bit–Mantisse in Double–Zahlen), bleibt die Funktion ohne Wirkung.

Rückgabewert

Rückgabewert	Bedeutung
Double–Zahl	Abgerundete Double–Zahl

Siehe auch

ceil (Seite 218)

fmod (Seite 257)

3.52 fmod

Funktion

Divisionsrest aus zwei Double-Zahlen ermitteln

Syntax

```
#include <math.h>

double    fmod(
           double x,
           double y);
```

Parameter

Parametername	Bedeutung
x	Double–Zahl (Zähler)
y	Double–Zahl (Nenner)

Beschreibung

`fmod` berechnet den Divisionsrest aus der Division von x/y. Bei `y` gleich 0.0 wird 0.0 als Ergebnis zurückgegeben.

Rückgabewert

Rückgabewert	Bedeutung
Double–Zahl	Divisionsrest

Siehe auch

ceil (Seite 218)

floor (Seite 256)

3.53 fopen

Funktion

Stream öffnen

Syntax

```
#include <stdio.h>

FILE    *fopen(
           char *fname,
           char *mode);
```

Parameter

Parametername	Bedeutung
fname	Zeiger auf Dateispezifikation
mode	Zeiger auf Zugriffsmodus

Beschreibung

`fopen` öffnet die mit `fname` angegebene Datei mit Zugriffsmodus `mode` und ordnet ihr einen Stream mit Struktur vom Typ FILE zu.

Der Zugriffsmodus kann folgende Werte annehmen:

"r" Öffnen zum Lesen im Textmodus

"w" Erzeugen zum Schreiben (auf 0 kürzen, wenn die Datei bereits existiert) im Textmodus

"a" Öffnen zum Schreiben (Anhängen) oder erzeugen, falls nicht existent, im Textmodus

"r+" Öffnen zum Ändern (Lesen und Schreiben) im Textmodus

"w+" Erzeugen zum Ändern (auf 0 kürzen, wenn die Datei bereits existiert) im Textmodus.

"a+" Öffnen zum Ändern (Anhängen) oder Erzeugen, falls nicht vorhanden, im Textmodus.

"rb" Öffnen zum Lesen in Binärmodus.

"wb" Erzeugen zum Schreiben (auf 0 kürzen, wenn schon vorhanden) im Binärmodus

"ab" Öffnen zum Anhängen (oder Erzeugen, wenn noch nicht vorhanden) im Binärmodus

"rb+" Öffnen zum Ändern (Lesen und Schreiben) im Binärmodus

"wb+" Erzeugen zum Ändern (auf 0 kürzen, wenn schon vorhanden) im Binärmodus

"ab+" Öffnen zum Ändern (Anhängen) oder Erzeugen, falls nicht vorhanden, im Binärmodus

Hinweis

Im Textmodus wird beim Schreiben ‘\n’ in ‘\r\n’ umgewandelt. Beim Lesen wird ‘\r\n’ in ‘\n’ umgewandelt. Im Binärmodus erfolgt keine Umwandlung. Der Stream ist nur für die aufrufende Task gültig und darf nur von ihr benutzt werden.

Stream-Pointer und Deskriptoren sind taskspezifisch und können nicht zwischen Tasks ausgetauscht werden.

Rückgabewert

Rückgabewert	Bedeutung
NULL–Zeiger	Datei kann nicht geöffnet werden (Datei `fname` erlaubt keinen Zugriff, `mode` ist unzulässig, oder keine freien Streams vorhanden)
Zeiger	Konnte die Datei geöffnet werden, wird ein Zeiger auf die stream–Struktur zurückgegeben.

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt:

Fehlercode	Bedeutung
EINVAL	Ungültiger Parameter
EIO	Physikalischer Schreib–/Lesefehler
ENOSPC	Kein Speicherplatz auf Laufwerk vorhanden oder Verzeichnis voll
EBUSY	Datei ist von einer anderen Task geöffnet
EMFILE	Interner Pufferüberlauf
ENOTDIR	Eine Komponente des Pfadnamen–Präfixes ist kein Verzeichnis
ENOENT	Die angegebene Datei oder eine Komponente von `fname` existiert nicht (Laufwerk wurde eventuell nicht angemeldet (`mount`)).
EACCES	Für eine Komponente des Pfadnamen–Präfixes besteht keine Sucherlaubnis oder die mit `mode` angegebene Zugriffsart ist für die Datei `fname` nicht erlaubt.

Siehe auch

fclose (Seite 245)

fduopen (Seite 246)

3.54 fprintf

Funktion

Formatierte Ausgabe auf einen Stream

Syntax

```
#include <stdio.h>

int    fprintf(
         FILE *stream,
         char *format [,
         arg ] ...);
```

Parameter

Parametername	Bedeutung
stream	Stream mit Struktur vom Typ FILE
format	Zeiger auf einen String
arg	Argument–Liste

Beschreibung

`fprintf` verarbeitet den Eingabe–String `format`, interpretiert verschiedene Format–Kommandos und gibt das Ergebnis auf dem mit `stream` angegebenen Stream aus.

`fprintf` gibt die Anzahl der übertragenen Zeichen zurück. Bei einem Ausgabe–Fehler wird ein Wert kleiner 0 zurückgegeben.

`format` ist ein String mit eingefügten Format–Anweisungen und dazugehörigen Argumenten: `fprintf (f,"...",arg1, arg2...)`.

Siehe `printf` für Einzelheiten zum Eingabe–String, Escape–Sequenzen und Formatanweisungen.

Hinweis

Die Ausgabe auf einen Stream erfolgt normalerweise gepuffert. Bei Pufferung wird die Ausgabe erst dann ausgeführt, wenn der Puffer voll bzw. die Puffergröße gleich 1 ist, bzw. bei Zeilen–Pufferung auch wenn ein Zeilenende '\n' ausgegeben wird. Soll die Ausgabe schon früher erfolgen, muss der Funktionsaufruf `fflush` eingesetzt werden.

Im Textmodus wird '\n' in '\r\n' umgewandelt.

Rückgabewert

Rückgabewert	Bedeutung
Anzahl	der ausgegebenen Zeichen (Funktion erfolgreich)
–1	Fehler (Funktion nicht erfolgreich)

Siehe auch

fscanf (Seite 270)
printf (Seite 329)
scanf (Seite 350)
snprintf (Seite 370)
sprintf (Seite 372)
sscanf (Seite 376)
vfprintf (Seite 414)
vprintf (Seite 418)
vsnprintf (Seite 421)
vsprintf (Seite 422)

3.55 fputc

Funktion

Zeichen auf einen Stream schreiben

Syntax

```
#include <stdio.h>

int    fputc(
        int c,
        FILE *stream);
```

Parameter

Parametername	Bedeutung
c	Zeichen
stream	Stream mit Struktur vom Typ FILE

Beschreibung

`fputc` schreibt das Zeichen `c` auf den mit `stream` angegebenen Stream.

Hinweis

`fputc` ist die Funktions–Version von `putc` und funktioniert wie diese. `fputc` ist langsamer als `putc`, benötigt aber weniger Speicherplatz. Außerdem kann `fputc` einer anderen Funktion als Argument übergeben werden. Bei Streams im Textmodus wird ‘\n’ in ‘\r\n’ umgewandelt.

Rückgabewert

Rückgabewert	Bedeutung
Integer–Zahl	Übertragenes Zeichen
EOF	Funktion nicht erfolgreich, z.B. wenn die Ausgabedatei `stream` nicht zum Schreiben geöffnet oder kein Platz mehr in dieser vorhanden ist

Siehe auch

fclose (Seite 245)
ferror (Seite 250)
fopen (Seite 258)
fread (Seite 264)
printf (Seite 329)
putchar (Seite 334)
puts (Seite 336)
putw (Seite 337)
setbuf (Seite 359)

3.56 fputs

Funktion

String auf einen Stream schreiben

Syntax

```
#include <stdio.h>

int    fputs(
        char *s,
        FILE *stream);
```

Parameter

Parametername	Bedeutung
s	Zeiger auf String
stream	Stream mit Struktur vom Typ FILE

Beschreibung

`fputs` schreibt den mit Zeiger `s` angegebenen String auf den mit `stream` angegebenen Stream. Der String endet mit einem Null–Byte, das nicht mit übertragen wird.

Hinweis

Bei Streams im Textmodus wird ‘\n’ in ‘\r\n’ umgewandelt.

Rückgabewert

Rückgabewert	Bedeutung
Anzahl	der übertragenen Zeichen
EOF	Funktion nicht erfolgreich (z.B. wenn die Ausgabedatei nicht zum Schreiben geöffnet worden ist)

Siehe auch

fopen (Seite 258)

fread (Seite 264)

printf (Seite 329)

putc (Seite 333)

3.57 fread

Funktion

Daten aus einem Stream lesen

Syntax

```
#include <stdio.h>

size_t    fread(
          char *ptr,
          size_t size,
          size_t n,
          FILE *stream);
```

Parameter

Parametername	Bedeutung
n	Anzahl der zu übertragenden Datensätze
size	Größe der Datensätze
stream	Stream mit Struktur vom Typ FILE
ptr	Zeiger auf einen Puffer

Beschreibung

`fread` liest `n` Datensätze der Größe `size` vom angegebenen `stream` in den Puffer, auf den `ptr` zeigt. Der Aufruf liefert als Rückgabewert die Anzahl der gelesenen Datensätze.

Hinweis

Im Textmodus wird '\r\n' in '\n' umgewandelt.

Rückgabewert

Rückgabewert	Bedeutung
Anzahl	der übertragenen Datensätze
0	Funktion nicht erfolgreich (z.B `size` oder `n` <= 0)

Siehe auch

fopen (Seite 258)

fscanf (Seite 270)

fwrite (Seite 275)

3.58 free

Funktion

Speicher freigeben

Syntax

```
#include <malloc.h>

void    free (void *ptr);
```

Parameter

Parametername	Bedeutung
ptr	Zeiger auf mit `malloc` oder `calloc` angeforderten Speicher

Beschreibung

`free` gibt einen vorher mit `malloc` oder `calloc` angeforderten Speicher frei.

Hinweis

Das Ergebnis des Funktionsaufrufs ist undefiniert, wenn mit `ptr` ein unzulässiger Parameter übergeben wird.

Siehe auch

calloc (Seite 217)

malloc (Seite 312)

realloc (Seite 344)

3.59 freopen

Funktion

Einem Stream zugeordenete Datei austauschen

Syntax

```
#include <stdio.h>

FILE    *freopen(
            char *fname,
            char *mode,
            FILE *stream);
```

Parameter

Parametername	Bedeutung
fname	Dateispezifikation
mode	Zugriffsmodus
stream	Stream mit Struktur vom Typ FILE

Beschreibung

`freopen` schließt die `stream` zugeordnete Datei, öffnet die mit `fname` angegebene Datei mit Zugriffsmodus `mode` und ordnet sie `stream` zu, ohne Rücksicht darauf, ob dieses Öffnen erfolgreich ist. `freopen` tauscht also die `stream` zugeordnete Datei aus.

Der Zugriffsmodus kann folgende Werte annehmen:

"r" Öffnen zum Lesen im Textmodus

"w" Erzeugen zum Schreiben (auf 0 kürzen, wenn die Datei bereits existiert) im Textmodus.

"a" Öffnen zum Schreiben (Anhängen) oder erzeugen, falls nicht existent im Textmodus.

"r+" Öffnen zum Ändern (Lesen / Schreiben) im Textmodus

"w+" Erzeugen zum Ändern (auf 0 kürzen, wenn die Datei bereits existiert) im Textmodus.

"a+" Öffnen zum Ändern (Anhängen) oder erzeugen, falls nicht vorhanden im Textmodus.

"rb" Öffnen zum Lesen in Binärmodus.

"wb" Erzeugen zum Schreiben (auf 0 kürzen, wenn schon vorhanden) im Binärmodus

"ab" Öffnen zum Anhängen (oder erzeugen, wenn noch nicht vorhanden) in Binärmodus

"rb+" Öffnen zum Ändern (lesen und schreiben) in Binärmodus

"wb+" Erzeugen zum Ändern (auf 0 kürzen, wenn schon vorhanden) in Binärmodus

"ab+" Öffnen zum Ändern (Anhängen) oder erzeugen, falls nicht vorhanden, in Binärmodus

Hinweis

Im Textmodus wird beim Schreiben ‘\n’ in ‘\r\n’ umgewandelt. Beim Lesen wird ‘\r\n’ in ‘\n’ umgewandelt. Im Binärmodus erfolgt keine Umwandlung.

Stream-Pointer und Deskriptoren sind taskspezifisch und können nicht zwischen Tasks ausgetauscht werden.

Rückgabewert

Rückgabewert	Bedeutung
NULL–Zeiger	Datei `fname` erlaubt keinen Zugriff, `mode` ist unzulässig, Datei kann nicht geöffnet werden, oder keine freien Streams vorhanden
Zeiger	Konnte die Datei geöffnet werden, wird ein Zeiger auf die Stream–Struktur zuückgegeben

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt:

Fehlercode	Bedeutung
EINVAL	Ungültiger Parameter
EIO	Physikalischer Schreib–/Lesefehler
ENOSPC	Kein Speicherplatz auf Laufwerk vorhanden oder Verzeichnis voll
EBUSY	Datei ist von einer anderen Task geöffnet
EMFILE	Interner Pufferüberlauf
ENOTDIR	Eine Komponente des Pfadnamen–Präfixes ist kein Verzeichnis
ENOENT	Die angegebene Datei oder eine Komponente von `fname` existiert nicht (Laufwerk wurde eventuell nicht angemeldet (`mount`)).
EACCES	Für eine Komponente des Pfadnamen–Präfixes besteht keine Sucherlaubnis oder die mit `mode` angegebene Zugriffsart ist für die Datei `fname` nicht erlaubt.

Beispiel

Mit `freopen` können die Funktionen `stdin`, `stdout` oder `stderr` einer Task auf eine Datei umgelenkt werden. Das folgende Beispiel lenkt `stdout` auf eine Datei um (ohne Fehlerabfrage nach `freopen`):

```
#include <stdio.h>

void main (void)
{

char tmpname[128];
int  tmpdev, tmpunit;

/* Save current stdout */
tmpname[0] = '\0';
/* device==-1 -> the stream is connected to a file   */
/* device!=-1 -> the stream is connected to a driver */
if (stdout->device==-1)
             strcpy(tmpname,stdout->_fid.fname); /* get file name */
else
             {
             tmpdev = stdout->device; /* get device ID */
             tmpunit = stdout->unit; /* get unit ID */
             }

/* Redirect stdout to a file */
fp = freopen("STDOUT.LOG", "w", stdout);
...
printf("Output to stdout\n");
...
/* Restore original stdout */
if (tmpname[0]!=0)
           freopen(tmpname, "w", stdout);
else
           fdureopen(tmpdev, tmpunit, "w", stdout);
...
}
```

Siehe auch

fclose (Seite 245)

fduopen (Seite 246)

fopen (Seite 258)

3.60 frexp

Funktion

Double-Zahl zerlegen

Syntax

```
#include <math.h>

double    frexp(
            double x,
            int *nptr);
```

Parameter

Parametername	Bedeutung
x	Double–Zahl
nptr	Zeiger auf Integer (Exponent)

Beschreibung

`frexp` zerlegt eine Double–Zahl in eine Double–Zahl im Bereich zwischen 0,5 und 1 und einen Exponenten n.
Dabei gilt die Beziehung `x = frexp(x,&n) * 2^n`. `frexp` gibt eine Double–Zahl zurück und überschreibt die Integer–Zahl, auf die `nptr` zeigt. Ist das Argument 0, wird sowohl als Ergebnis, wie auch als Exponent 0 zurückgegeben. Ist das Argument INF oder ein Unnormal außerhalb des zulässigen Bereichs, ruft `frexp` die Fehlerbehandlungsfunktion `matherr` mit Exception–Typ DOMAIN auf, die als Ergebnis IND liefert. In diesem Fall wird 0 als Exponent zurückgeschrieben.

Rückgabewert

Rückgabewert	Bedeutung
Double–Zahl	im Bereich 0,5 bis 1 (Exponent ist über den Zeiger `nptr` zu erreichen)
IND	Nach Aufruf von `matherr` mit Exception–Typ DOMAIN

3.61 fscanf

Funktion

Formatierte Eingabe von einem Stream

Syntax

```
#include <stdio.h>

int    fscanf(
         FILE *stream,
         char *format,
         arg_ptr ...);
```

Parameter

Parametername	Bedeutung
stream	Stream mit Struktur vom Typ FILE
format	Zeiger auf einen String
arg_ptr	Zeiger auf einen Zielbereich

Beschreibung

`fscanf` liest Datenelemente von dem mit `stream` angegebenen Stream und interpretiert sie gemäß den mit `format` angegebenen Format-Anweisungen. Anschließend hinterlegt die Funktion die Elemente an der mit `arg_ptr` angegebenen Stelle. Siehe `scanf` für Einzelheiten zu `format` und Konvertierungszeichen.

Hinweis

Nachfolgende Leerzeichen (einschließlich Zeilenschaltung) werden nicht gelesen, außer sie sind auch im Control–String enthalten.

Bei Streams im Textmodus wird ‘\r\n’ in ‘\n’ umgewandelt.

Das Einlesen eines Datenelements wird beim ersten ungültigen Zeichen beendet. Beispielsweise wird beim Einlesen einer Dezimalzahl durch %d bei Eingabe von "123XY" der Wert 123 eingelesen, der nächste Aufruf arbeitet bei "X" weiter.

Tritt beim Einlesen einer Zahl ein Überlauf auf, d.h. der angegebene Wert ist größer als der für den zugehörigen Typ in LIMITS.H angegebene Grenzwert, wird der Grenzwert aus LIMITS.H an den Anwender zurückgeliefert und `errno` auf ERANGE gesetzt. Wird beispielsweise beim Einlesen einer Zahl vom Typ Unsigned–Short (%hd) ein Wert größer als USHRT_MAX (65535) angegeben, liefert `fscanf` den Wert USHRT_MAX zurück.

Beim Einlesen von Oktal– oder Hexadezimal–Zahlen sind Vorzeichen nicht zulässig.

Rückgabewert

Rückgabewert	Bedeutung
Anzahl	der erfolgreich verglichenen und zugewiesenen Formatelemente
0	Vergleich beginnt mit einem Konflikt zwischen Eingabe–Zeichen und Control–String
EOF	Eingabe vor dem ersten Konflikt oder Konvertierung beendet

Siehe auch

fprintf (Seite 260)
printf (Seite 329)
scanf (Seite 350)
sprintf (Seite 372)
sscanf (Seite 376)
vfprintf (Seite 414)
vfscanf (Seite 416)
vprintf (Seite 418)
vscanf (Seite 420)
vsscanf (Seite 424)
vsprintf (Seite 422)

3.62 fseek

Funktion

Datenzeiger in einem Stream positionieren

Syntax

```
#include <stdio.h>

int    fseek(
         FILE *stream,
         long offset,
         int fromwhere);
```

Parameter

Parametername	Bedeutung
stream	Stream mit Struktur vom Typ FILE
offset	Offset in Bytes
fromwhere	Startposition von offset

Beschreibung

`fseek` positioniert den Dateizeiger in dem mit `stream` angegebenen Stream für den nächsten Lese– oder Schreibzugriff. Die Position wird aus dem mit `offset` angegebenen Abstandswert und dem mit `fromwhere` angegebenen Basiswert berechnet. Das Argument `fromwhere` kann drei Werte annnehmen:

SEEK_SET (=0) Dateianfang

SEEK_CUR (=1) Aktuelle Position

SEEK_END (=2) Dateiende

Hinweis

Bei im Textmodus geöffneten Streams ist `fseek` aufgrund der Umwandlungen ‘\n’ bzw. ‘\r\n’ nur bedingt einsetzbar. Die einzigen zulässigen Operationen mit sind:

`fseek` mit Offset 0

`fseek` von Dateianfang mit einem zuvor von `ftell` gelieferten Offset

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich
–1	Fehler oder Funktion nicht erfolgreich (z.B. wenn der Stream nicht geöffnet wurde)

Siehe auch

ftell (Seite 274)

lseek (Seite 310)

rewind (Seite 348)

3.63 fsetpos

Funktion

Aktuelle Position in einer Datei setzen

Syntax

```
#include <stdio.h>

int    fsetpos(
          FILE *stream,
          const fpos_t *pos);
```

Parameter

Parametername	Bedeutung
stream	Stream mit Struktur vom Typ FILE
pos	Zeiger auf Dateiposition

Beschreibung

`fsetpos` setzt die Position `pos` in der mit `stream` angegebenen Datei und löscht das dazugehörige Dateiende–Flag. Wie bei `fseek` sind zuvor mit `ungetc` "zurückgestellte" Zeichen nach dieser Operation nicht mehr verfügbar.

Die auf `fsetpos` folgende Operation kann sowohl eine Lese– als auch eine Schreiboperation sein, wenn die Datei im entsprechenden Modus geöffnet wurde.

Hinweis

Bei im Textmodus geöffneten Streams ist diese Funktion nur eingeschränkt nutzbar (wegen der Umwandlung von ‘\n’ bzw. ‘\r\n’). Diese Funktion darf nur dazu verwendet werden, um die Position in einem Stream mit `fgetpos` zu ermitteln und später durch `fsetpos` wieder auf diese Stelle zu positionieren.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich
≠0	Fehler oder Funktion nicht erfolgreich

Siehe auch

fgetpos (Seite 253)

fseek (Seite 272)

3.64 ftell

Funktion

Dateizeiger in einem Stream ermitteln

Syntax

```
#include <stdio.h>

long    ftell (FILE *stream);
```

Parameter

Parametername	Bedeutung
stream	Stream mit Struktur vom Typ FILE

Beschreibung

`ftell` ermittelt den aktuellen Offset des Dateizeigers in Bytes vom Anfang des Streams.

Rückgabewert

Rückgabewert	Bedeutung
Long–Integer	Offset von Dateianfang in Bytes
–1	Funktion nicht erfolgreich (Datei nicht zum Lesen oder Schreiben geöffnet)

Siehe auch

fseek (Seite 272)
lseek (Seite 310)
rewind (Seite 348)

3.65 fwrite

Funktion

Daten in einen Stream schreiben

Syntax

```
#include <stdio.h>

size_t    fwrite(
            char *ptr,
            size_t size,
            size_t n,
            FILE *stream);
```

Parameter

Parametername	Bedeutung
ptr	Zeiger auf ersten Datensatz
size	Größe des Datensatzes
n	Anzahl der Datensätze
stream	Stream mit Struktur vom Typ FILE

Beschreibung

`fwrite` schreibt `n` Datensätze der angegebenen Größe `size` auf den mit `stream` angegebenen Stream, beginnend mit dem Datensatz, auf den `ptr` zeigt. Der Inhalt des Puffers bleibt unverändert.

Hinweis

Im Textmodus wird beim Schreiben ‘\n’ in ‘\r\n’ umgewandelt.

Rückgabewert

Rückgabewert	Bedeutung
size_t	Anzahl der übertragenen Datensätze
0	Funktion nicht erfolgreich (z.B `size` oder `n` ≤ 0)

Siehe auch

fopen (Seite 258)

fprintf (Seite 260)

fread (Seite 264)

3.66 getc

Funktion

Zeichen aus einem Stream lesen

Syntax

```
#include <stdio.h>

int    getc (FILE *stream);
```

Parameter

Parametername	Bedeutung
stream	Stream mit Struktur vom Typ FILE

Beschreibung

`getc` liefert das nächste Zeichen im Puffer des mit `stream` angegebenen Streams und setzt den Dateizeiger auf das darauffolgende Zeichen.

Hinweis

Wird ein mit `getc` zurückgegebener Integerwert in einer Character–Variable gespeichert und mit der Integer–Konstanten EOF verglichen, kann die Vergleichsoperation mit einem Fehler enden, denn das Hinzufügen eines Vorzeichens bei Umwandlung von Character auf Integer ist maschinenabhängig.

Da `getc` als Makro implementiert ist, behandelt es ein Stream–Argument nicht korrekt mit unvorhersehbaren Nebenwirkungen. Insbesondere liefert der Aufruf `getc (*f++)` nicht das erwartete Ergebnis. Verwenden Sie in diesem Fall `fgetc`.

Im Textmodus wird '\r\n' in '\n' umgewandelt.

Rückgabewert

Rückgabewert	Bedeutung
Integer–Zahl	Übertragenes Zeichen
EOF	Fehler oder Dateiende erreicht

Siehe auch

fgetc (Seite 252)

fopen (Seite 258)

fread (Seite 264)

getchar (Seite 277)

gets (Seite 280)

getw (Seite 285)

putc (Seite 333)

scanf (Seite 350)

3.67 getchar

Funktion

Zeichen von der Standard-Eingabe lesen

Syntax

```
#include <stdio.h>

int    getchar(void);
```

Beschreibung

`getchar` liefert das nächste Zeichen aus der Standard–Eingabe. Die Funktion ist äquivalent zu `getc(stdin)`.

Hinweis

Werden Integerwerte, die die Funktion `getchar` zurückgibt, als Character–Variable gespeichert und mit der Integer–Konstanten EOF verglichen, kann die Vergleichsoperation mit einem Fehler enden, denn das Hinzufügen eines Vorzeichens bei Umwandlung von Character auf Integer ist maschinenabhängig.

Rückgabewert

Rückgabewert	Bedeutung
Integer–Zahl	Übertragenes Zeichen
EOF	Fehler oder Dateiende erreicht

Siehe auch

fgetc (Seite 252)
fopen (Seite 258)
fread (Seite 264)
getc (Seite 276)
gets (Seite 280)
getw (Seite 285)
putc (Seite 333)
scanf (Seite 350)

3.68 getcwd

Funktion

CWD (current working directory, aktuelles Vezeichnis) liefern

Syntax

```
#include <direct.h>

char    *getcwd(
            char *buf,
            int size);
```

Parameter

Parametername	Bedeutung
buf	Speicherbereich, in dem der absolute Pfadname des CWD abgelegt wird
size	Größe des Speicherbereichs in Bytes

Beschreibung

`getcwd` schreibt den absoluten Pfadnamen des CWD (Current Working Directory, aktuelles Arbeitsverzeichnis) in den mit `buf` angegebenen Speicherbereich der Größe `size`. Ist `size` zu klein, wird `getcwd` beendet, der mit `buf` angegebene Speicherbereich bleibt unverändert. Ist `buf` gleich NULL, fordert CRUN den erforderlichen Speicherbereich mit `malloc` an, mindestens jedoch `size` Bytes, und kopiert den Pfadnamen dort hin. In diesem Fall kann der von `getcwd` zurückgegebene Zeiger vom Anwender benutzt werden, um den Speicherbereich später mit `free` freizugeben. Wurde das CWD der aufrufenden Task noch nicht mit `chdir` gesetzt, ist der Pfadname leer, d.h. das erste Byte ist '\0'.

Hinweis

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf einen String der den absoluten Pfad des CWD enthält
NULL–Zeiger	Funktion nicht erfolgreich

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt:

Fehlercode	Bedeutung
EINVAL	`size` <= 0
ERANGE	`size` < Länge des Pfadnamens + 1
ENOMEM	Kein freier Speicherplatz verfügbar

Siehe auch

chdir (Seite 220)

3.69 getenv

Funktion

Umgebungs-Variable ermitteln

Syntax

```
#include <stdlib.h>

char    *getenv (char *name);
```

Parameter

Parametername	Bedeutung
name	Zeiger auf den Namen der Umgebungs–Variablen

Beschreibung

`getenv` sucht in der Umgebung nach einem String mit Format "name=value".

Für jede Task existiert eine eigene Umgebung. Der Aufbau der Umgebung ist unter `x_cr_initenv` beschrieben.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	Zeiger auf Umgebungs–Variable
NULL–Zeiger	Funktion nicht erfolgreich

Siehe auch

putenv (Seite 335)

x_cr_initenv (Seite 432)

3.70 gets

Funktion

String aus einem Stream lesen

Syntax

```
#include <stdio.h>

char    *gets (char *s);
```

Parameter

Parametername	Bedeutung
s	Zeiger auf Speicherbereich

Beschreibung

`gets` liest den nächsten String bis CR oder EOF vom Standard–Eingabe–Stream und kopiert ihn in den mit `s` angegebenen Speicherbereich. CR wird durch ein Null–Byte ersetzt.

Hinweis

`gets` meldet keinen Fehler, wenn der zu lesende String zu lang ist. Um dieses Problem zu vermeiden sollte `fgets` verwendet werden.

Im Textmodus wird ‘\r\n’ in ‘\n’ umgewandelt.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf Ziel–Speicherbereich (Funktion erfolgreich)
NULL–Zeiger	Keine Zeichen übertragen (z.B. wenn EOF erreicht ist oder bei einem Lesefehler).

Siehe auch

fgets (Seite 254)

fopen (Seite 258)

fread (Seite 264)

getc (Seite 276)

scanf (Seite 350)

3.71 gettimestamp

Funktion

Anzahl der Clock-Zyklen seit Systemstart ermitteln

Syntax

```
#include <misc86.h>

unsigned long long    gettimestamp (void);
```

Beschreibung

gettimestamp liefert die Anzahl der Clock-Zyklen, die seit dem Systemstart vergangen sind. Anhand der Prozessorfrequenz kann damit die benötigte Zeit für eine Aktion berechnet werden.

Hinweis

Diese Funktion ist nur für GNU verfügbar.
Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
unsigned long long	Anzahl der Clock-Zyklen seit dem Systemstart

Beispiel

```
#include <misc86.h>
static unsigned int jiffies (void) {
   ulong starttime[2];
   RmGetAbsTime ( (RmAbsTimeStruct*) starttime );
   return starttime[0];
}
int _FAR _FIXED main( int argc, char* argv[] ) {
   unsigned long long starttime, endtime;
   unsigned int help, usec;

   printf("timestamp test\n");
   // schauen wie viele Zyklen eine ms hat
   help = jiffies();
   while (help == jiffies()) {;} /* warten bis ms sich aendern */
   starttime = gettimestamp();
   help = jiffies();
   while (help == jiffies()) {;} /* warten bis ms sich aendern */
   endtime = gettimestamp();
   usec = (endtime - starttime)/1000; /*Voraussetzung: timer tick 1 ms */

   starttime = gettimestamp();
   // Code dessen Zeit gemessen werden soll
   RmPauseTask(100ul); // 100 ms
   endtime = gettimestamp();
   printf("elapsed time: = %18llu cycles, %d usec\n",
   (endtime - starttime),
   (unsigned int)((endtime - starttime)/usec));
   exit(0);
}
```

Siehe auch

RmGetAbsTime (Seite 64)

3.72 getvolumestatus

Funktion

Statusinformationen eines Datenträgers ermitteln

Syntax

```
#include <io.h>

int    getvolumestatus (
                char * volume ,
                VOL_DATA * vol_vib,
                unsigned long *free_blocks,
                unsigned *open_files,
                unsigned *write_errors);
```

Parameter

Parametername	Bedeutung
volume	Datenträgername (z.B. "C")
free_blocks	Zeiger auf ein unsigned long, in das die Anzahl der freien Blöcke geschrieben wird.
open_files	Zeiger auf ein unsigned int, in das die Anzahl der offenen Files geschrieben wird.
write_errors	Zeiger auf ein unsigned int, in das die Anzahl der seit dem letzten Anmelden (`mount`) aufgetretenen Schreibfehler geschrieben wird.

vol_vib	Zeiger auf eine Struktur von Typ VOL_DATA, in der die Statusinformationen abgelegt werden.
	
	`typedef struct _VOL_DATA`
	`{`
	`char vol_name[2];`
	`unsigned long no_of_blocks;`
	`unsigned int no_of_root_blocks;`
	`unsigned char device_id;`
	`unsigned char unit_id;`
	`unsigned int flags;`
	`char *fat_buf;`
	`unsigned short fbuf_fill;`
	`unsigned long start_block;`
	`void *boot_sec;`
	`unsigned short boot_fill;`
	`} VOL_DATA;`

	vol_name Datenträgername, der bei der HSFS–Konfigurierung vergeben wurde, oder bei `createvib` oder `changevib` angegeben wurde. Besteht der Name nur aus einem Zeichen, muss das zweite Zeichen auf 0 oder Space gesetzt werden.
	no_of_blocks Anzahl der 512 Byte großen Datenblöcke, die zu diesem Datenträger gehören.
	no_of_root_blocks Anzahl der Blöcke, die für das Hauptverzeichnis reserviert sind. Es können maximal no_of_root_blocks/ 16 Dateien, bzw. Verzeichnisse im Hauptverzeichnis abgelegt werden. Nicht gültig bei FAT32.

Parametername	Bedeutung
	device_id Die Device–Nummer des Treibers für diesen Datenträger.
	unit_id Die Unit des in `device_id` angegebenen Treibers.
	Flags Datenträgerflags: VL_NO_WRITE_BUF Auf jeden logischen Schreibvorgang folgt ein physikalisches Schreiben. VL_NODATE Das Datum und die Uhrzeit wird bei diesem Datenträger nicht geschrieben. VL_WRITE_FLAG Es fand seit dem Anmelden eine Schreiboperation statt. VL_AUTO_DISMOUNT Beim Schließen einer modifizierten Datei werden die Daten der Datei und die Dateizuordnungstabelle (FAT) sofort auf das Medium geschrieben. VL_NO_IO_BUF Der Datenträger arbeitet ohne Zwischenpufferung. VL_NO_REMAP Der Datenträger ist formatiergeschützt. VL_NO_MOUNTING Der Datenträger wird bei einem OPEN-Aufruf nicht automatisch angemeldet. VL_NET Der Datenträger wird über die Netzschnittstelle angesprochen. VL_REMOVABLE Der Datenträger wird automatisch abgemeldet, wenn seine letzte Datei geschlossen wird. VL_512 Die Clustergröße beträgt 512 Byte. VL_MOUNTED Datenträger ist angemeldet.
	fat_buf Hier steht ein Zeiger auf den Puffer für die Dateizuordnungstabelle (FAT).
	fbuf_fill Nur im Speichermodell FLAT verwendetes Füllwort. Ohne Bedeutung.
	start_block Dieser Wert ist der erste Block, der diesem Datenträger logisch zugeordnet ist.
	boot_sec Unter der Adresse befindet sich der Bootsektor, der zum Formatieren und beim Anmelden des Datenträgers verwendet wird. Die Struktur hat folgendes Aussehen:

Parametername	Bedeutung

	`typedef struct _MS_BOOT_SEC`
	`{`
	`char boot_jmp[3];`
	`char disk_name[8];`
	`unsigned short bytes_per_sec;`
	`char sec_per_clust;`
	`unsigned short res_sec;`
	`char no_of_fats;`
	`unsigned short root_entries;`
	`unsigned short partition_sec1;`
	`char id_byte;`
	`unsigned short sec_per_fat;`
	`unsigned short sec_per_track;`
	`unsigned short sides;`
	`long hidden_sec;`
	`long partition_sec2;`
	`unsigned int fat32_length;`
	
	`unsigned short flags;`
	`char version[2];`
	`unsigned int root_cluster;`
	`unsigned short info_sector;`
	`unsigned short backup_boot;`
	`unsigned short reserved2[6];`
	`char fill_block[448];`
	`} MS_BOOT_SEC;`

	boot_fill Nur im Speichermodell FLAT verwendetes Füllwort. Ohne Bedeutung.

Beschreibung

Mit dem Aufruf `getvolumestatus` kann der aktuelle Zustand eines Datenträgers ermittelt werden.

Hinweis

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich abgearbeitet
-1	Fehler

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt:

Fehlercode	Bedeutung
ENOENT	Datenträger `volume` nicht gefunden
ERANGE	Unerwarteter Fehlercode vom HSFS

3.73 getw

Funktion

Wort aus einem Stream lesen

Syntax

```
#include <stdio.h>

int    getw (FILE *stream);
```

Parameter

Parametername	Bedeutung
stream	Stream mit Struktur vom Typ FILE

Beschreibung

`getw` liefert das nächste Wort im Puffer des mit `stream` angegebenen Streams und setzt einen eventuell definierten Dateizeiger auf das darauffolgende Wort. Die Wortlänge entspricht der Größe einer 4 Byte langen Integer–Zahl. `getw` setzt keine bestimmte Ausrichtung des Dateizeigers auf Wortadressen in der betreffenden Datei voraus.

Hinweis

Im Textmodus wird ‘\r\n’ in ‘\n’ umgewandelt.

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
Integer–Zahl	Übertragenes Wort
EOF	Fehler oder Dateiende erreicht

Siehe auch

fgetc (Seite 252)
fopen (Seite 258)
fread (Seite 264)
getc (Seite 276)
getchar (Seite 277)
gets (Seite 280)
putc (Seite 333)
scanf (Seite 350)

3.74 gmtime

Funktion

Datum und Uhrzeit in Greenwich Mean Time (GMT) umwandeln

Syntax

```
#include <time.h>

struct    tm *gmtime (long *clock);
```

Parameter

Parametername	Bedeutung
clock	Zeiger auf einen Long–Integer

Beschreibung

`gmtime` wandelt Datum und Uhrzeit aus einem Long–Integer (wie er von `time` geliefert wird) in GMT (Greenwich Mean Time) um. `gmtime` gibt einen Zeiger auf eine Struktur vom Typ `tm` zurück.

Die Struktur tm ist in TIME.H folgendermaßen definiert:

```
struct tm{
 int tm_sec;   /* Sekunden (0-59)               */
 int tm_min;   /* Minuten (0-59)                */
 int tm_hour;  /* Stunden (0-23)                */
 int tm_mday;  /* Monatstag (1-31)              */
 int tm_mon;   /* Monat (0-11)                  */
 int tm_year;  /* Jahr minus 1900               */
 int tm_wday;  /* Wochentag (Sonntag = 0)       */
 int tm_yday;  /* laufender Tag im Jahr (0-365) */
 int tm_isdst; /* ungleich 0 falls Sommerzeit   */
};
```

Hinweis

Die Rückgabewerte zeigen auf einen Datenbereich, dessen Inhalt beim nächsten Aufruf überschrieben wird.

Verwenden Sie `_gmtime32`, wenn Sie Zeitangaben über den 19. Januar 2038 hinaus benötigen.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf eine Struktur vom Typ `tm`

Beispiel

Das folgende Beispiel zeigt den Einsatz der verschiedenen CRUN–Zeitfunktionen:

```
#include <time.h>
main ()
{
   struct tm *ptm;
   char *date;
   long tadd;
   putenv ("TZ=MEZ-1");
   time (&tadd);
   date = ctime (&tadd);
   printf ("today's date is %s", date);

   ptm = localtime (&tadd);
   date = asctime (ptm);
   printf ("today's date is %s", date);

   ptm = gmtime (&tadd);
   date = asctime (ptm);
   printf ("Greenwich mean time is %s", date);
}
```

Siehe auch

_gmtime32 (Seite 197)

asctime (Seite 206)

ctime (Seite 231)

localtime (Seite 305)

mktime (Seite 322)

time (Seite 404)

tzset (Seite 410)

strftime (Seite 384)

3.75 inhibitabort

Funktion

Task–Abbruch verhindern oder wieder zulassen

Syntax

```
#include <io.h

int    inhibitabort (int request);
```

Parameter

Parametername	Bedeutung
request	= 1 um einen Abbruch zu verhindern, = 0 um einen Abbruch zuzulassen.

Beschreibung

Mit dieser Funktion können Sie einen eventuellen Abbruch einer von CLI kontrollierten Task durch <Ctrl>+C oder durch den CLI Befehl `cancel` verhindern bzw. wieder zulassen. Dies ist zu empfehlen, falls Sie z.B. ein Semaphor anfordern möchten, das ja bei einem eventuellen Abbruch nicht wieder freigegeben wird. Falls Sie einen Abbruch in einer allgemein verwendbaren Funktion vorbeugen möchten, sollten Sie den bisherigen Status nach dem folgenden Muster retten und wiederherstellen.

```
{
  int abort_status ;

  abort_status = inhibitabort (1) ;
 ...
  inhibitabort (abort_status) ;
}
```

Wird versucht, eine Task abzubrechen, während dies durch `inhibitabort(1)` verhindert ist, erfolgt der Abbruch der Task sofort, wenn der "Abbruchschutz" wieder aufgehoben ist (`inhibitabort(0)`).

Rückgabewert

Rückgabewert	Bedeutung
1	Abbruch wurde bisher verhindert.
0	Abbruch war bisher erlaubt.

3.76 isalnum

Funktion

Bestimmen, ob ein Zeichen alphanumerisch ist

Syntax

```
#include <ctype.h>

int    isalnum (int c);
```

Parameter

Parametername	Bedeutung
c	Zeichen

Beschreibung

`isalnum` ist ein Makro, das als Eingabe–Argument ein einzelnes Zeichen verarbeitet und als Ergebnis eine Integer–Zahl liefert, die angibt ob `c` ein alphanumerisches Zeichen ist. Ein Wert ungleich 0 bedeutet TRUE, der Wert 0 FALSE. Das Makro ist in der Headerdatei CTYPE.H definiert.

Hinweis

Das Makro setzt voraus, dass `c` im Bereich des ASCII–Zeichensatzes liegt (Code 0 bis einschließlich 127 (dezimal)).

Rückgabewert

Rückgabewert	Bedeutung
TRUE	`c` ein alphanumerisches Zeichen
FALSE	`c` kein alphanumerisches Zeichen

Siehe auch

isalpha (Seite 290)
isascii (Seite 292)
iscntrl (Seite 293)
isdigit (Seite 294)
isgraph (Seite 295)
islower (Seite 296)
isprint (Seite 297)
ispunct (Seite 298)
isspace (Seite 299)
isupper (Seite 300)
isxdigit (Seite 301)
tolower, _tolower (Seite 408)
toupper, _toupper (Seite 409)

3.77 isalpha

Funktion

Bestimmen, ob ein Zeichen alphabetisch ist

Syntax

```
#include <ctype.h>

int    isalpha (int c);
```

Parameter

Parametername	Bedeutung
c	Zeichen

Beschreibung

`isalpha` ist ein Makro, das als Eingabe–Argument ein einzelnes Zeichen verarbeitet und als Ergebnis eine Integer–Zahl liefert, die angibt ob `c` ein alphabetisches Zeichen ist. Ein Wert ungleich 0 bedeutet TRUE, der Wert 0 FALSE. Das Makro ist in der Headerdatei CTYPE.H definiert.

Hinweis

Das Makro setzt voraus, dass `c` im Bereich des ASCII–Zeichensatzes liegt (Code 0 bis einschließlich 127 (dezimal)).

Rückgabewert

Rückgabewert	Bedeutung
TRUE	`c` ein alphabetisches Zeichen
FALSE	`c` kein alphabetisches Zeichen

Siehe auch

isalnum (Seite 289)
isascii (Seite 292)
iscntrl (Seite 293)
isdigit (Seite 294)
isgraph (Seite 295)
islower (Seite 296)
isprint (Seite 297)
ispunct (Seite 298)
isspace (Seite 299)
isupper (Seite 300)
isxdigit (Seite 301)
tolower, _tolower (Seite 408)
toupper, _toupper (Seite 409)

3.78 isascii

Funktion

Bestimmen, ob ein Zeichen im Bereich des ASCII–Zeichensatzes liegt

Syntax

```
#include <ctype.h>

int    isascii (int c);
```

Parameter

Parametername	Bedeutung
c	Zeichen

Beschreibung

`isascii` ist ein Makro, das als Eingabe–Argument ein einzelnes Zeichen verarbeitet und als Ergebnis eine Integer–Zahl liefert, die angibt ob `c` im Bereich des ASCII–Zeichensatzes (Code 0 bis einschließlich 127) liegt. Ein Wert ungleich 0 bedeutet TRUE, der Wert 0 FALSE. Das Makro ist in der Headerdatei CTYPE.H definiert.

Rückgabewert

Rückgabewert	Bedeutung
TRUE	`c` im Bereich des ASCII–Zeichensatzes
FALSE	`c` nicht im Bereich des ASCII–Zeichensatzes

Siehe auch

isalnum (Seite 289)
isalpha (Seite 290)
iscntrl (Seite 293)
isdigit (Seite 294)
isgraph (Seite 295)
islower (Seite 296)
isprint (Seite 297)
ispunct (Seite 298)
isspace (Seite 299)
isupper (Seite 300)
isxdigit (Seite 301)
tolower, _tolower (Seite 408)
toupper, _toupper (Seite 409)
toascii (Seite 407)

3.79 iscntrl

Funktion

Bestimmen, ob ein Zeichen ein ASCII–Steuerzeichen ist

Syntax

```
#include <ctype.h>

int    iscntrl (int c);
```

Parameter

Parametername	Bedeutung
c	Zeichen

Beschreibung

`iscntrl` ist ein Makro, das als Eingabe–Argument ein einzelnes Zeichen verarbeitet und als Ergebnis eine Integer–Zahl liefert, die angibt ob `c` ein ASCII–Steuerzeichen (Code 127 oder < 32) ist. Ein Wert ungleich 0 bedeutet TRUE, der Wert 0 FALSE. Das Makro ist in der Headerdatei CTYPE.H definiert.

Hinweis

Das Makro setzt voraus, dass `c` im Bereich des ASCII–Zeichensatzes liegt (Code 0 bis einschließlich 127 (dezimal)).

Rückgabewert

Rückgabewert	Bedeutung
TRUE	`c` ein ASCII–Steuerzeichen
FALSE	`c` kein ASCII–Steuerzeichen

Siehe auch

isalnum (Seite 289)
isalpha (Seite 290)
isascii (Seite 292)
isdigit (Seite 294)
isgraph (Seite 295)
islower (Seite 296)
isprint (Seite 297)
ispunct (Seite 298)
isspace (Seite 299)
isupper (Seite 300)
isxdigit (Seite 301)
tolower, _tolower (Seite 408)
toupper, _toupper (Seite 409)

3.80 isdigit

Funktion

Bestimmen, ob ein Zeichen eine Dezimalzahl ist

Syntax

```
#include <ctype.h>

int    isdigit (int c);
```

Parameter

Parametername	Bedeutung
c	Zeichen

Beschreibung

`isdigit` ist ein Makro, das als Eingabe–Argument ein einzelnes Zeichen verarbeitet und als Ergebnis eine Integer–Zahl liefert, die angibt ob `c` eine Dezimalzahl ([0–9]) ist. Ein Wert ungleich 0 bedeutet TRUE, der Wert 0 FALSE. Das Makro ist in der Headerdatei CTYPE.H definiert.

Hinweis

Das Makro setzt voraus, dass `c` im Bereich des ASCII–Zeichensatzes liegt (Code 0 bis einschließlich 127 (dezimal)).

Rückgabewert

Rückgabewert	Bedeutung
TRUE	`c` eine Dezimalzahl
FALSE	`c` keine Dezimalzahl

Siehe auch

isalnum (Seite 289)

isalpha (Seite 290)

isascii (Seite 292)

iscntrl (Seite 293)

isgraph (Seite 295)

islower (Seite 296)

isprint (Seite 297)

ispunct (Seite 298)

isspace (Seite 299)

isupper (Seite 300)

isxdigit (Seite 301)

tolower, _tolower (Seite 408)

toupper, _toupper (Seite 409)

3.81 isgraph

Funktion

Bestimmen, ob ein Zeichen ein druckbares Zeichen ist

Syntax

```
#include <ctype.h>

int    isgraph (int c);
```

Parameter

Parametername	Bedeutung
c	Zeichen

Beschreibung

`isgraph` ist ein Makro, das als Eingabe–Argument ein einzelnes Zeichen verarbeitet und als Ergebnis eine Integer–Zahl liefert, die angibt ob `c` ein druckbares Zeichen ungleich Leerzeichen ist. Ein Wert ungleich 0 bedeutet TRUE, der Wert 0 FALSE. Das Makro ist in der Headerdatei CTYPE.H definiert.

Hinweis

Das Makro setzt voraus, dass `c` im Bereich des ASCII–Zeichensatzes liegt (Code 0 bis einschließlich 127 (dezimal)).

Rückgabewert

Rückgabewert	Bedeutung
TRUE	`c` ein druckbares Zeichen ungleich Leerzeichen
FALSE	`c` kein druckbares Zeichen ungleich Leerzeichen

Siehe auch

isalnum (Seite 289)
isalpha (Seite 290)
isascii (Seite 292)
iscntrl (Seite 293)
isdigit (Seite 294)
islower (Seite 296)
isprint (Seite 297)
ispunct (Seite 298)
isspace (Seite 299)
isupper (Seite 300)
isxdigit (Seite 301)
tolower, _tolower (Seite 408)
toupper, _toupper (Seite 409)

3.82 islower

Funktion

Bestimmen, ob ein Zeichen ein Kleinbuchstabe ist

Syntax

```
#include <ctype.h>

int    islower (int c);
```

Parameter

Parametername	Bedeutung
c	Zeichen

Beschreibung

`islower` ist ein Makro, das als Eingabe–Argument ein einzelnes Zeichen verarbeitet und als Ergebnis eine Integer–Zahl liefert, die angibt ob `c` ein Kleinbuchstabe ist. Ein Wert ungleich 0 bedeutet TRUE, der Wert 0 FALSE. Das Makro ist in der Headerdatei CTYPE.H definiert.

Hinweis

Das Makro setzt voraus, dass `c` im Bereich des ASCII–Zeichensatzes liegt (Code 0 bis einschließlich 127 (dezimal)).

Rückgabewert

Rückgabewert	Bedeutung
TRUE	`c` ein Kleinbuchstabe
FALSE	`c` kein Kleinbuchstabe

Siehe auch

isalnum (Seite 289)
isalpha (Seite 290)
isascii (Seite 292)
iscntrl (Seite 293)
isdigit (Seite 294)
isgraph (Seite 295)
isprint (Seite 297)
ispunct (Seite 298)
isspace (Seite 299)
isupper (Seite 300)
isxdigit (Seite 301)
tolower, _tolower (Seite 408)
toupper, _toupper (Seite 409)

3.83 isprint

Funktion

Bestimmen, ob ein Zeichen ein druckbares Zeichen ist

Syntax

```
#include <ctype.h>

int    isprint (int c);
```

Parameter

Parametername	Bedeutung
c	Zeichen

Beschreibung

`isprint` ist ein Makro, das als Eingabe–Argument ein einzelnes Zeichen verarbeitet und als Ergebnis eine Integer–Zahl liefert, die angibt ob `c` ein druckbares Zeichen (Code 32 bis 127) ist. Ein Wert ungleich 0 bedeutet TRUE, der Wert 0 FALSE. Das Makro ist in der Headerdatei CTYPE.H definiert.

Hinweis

Das Makro setzt voraus, dass `c` im Bereich des ASCII–Zeichensatzes liegt (Code 0 bis einschließlich 127 (dezimal)).

Rückgabewert

Rückgabewert	Bedeutung
TRUE	`c` ein druckbares Zeichen
FALSE	`c` kein druckbares Zeichen

Siehe auch

isalnum (Seite 289)
isalpha (Seite 290)
isascii (Seite 292)
iscntrl (Seite 293)
isdigit (Seite 294)
isgraph (Seite 295)
islower (Seite 296)
ispunct (Seite 298)
isspace (Seite 299)
isupper (Seite 300)
isxdigit (Seite 301)
tolower, _tolower (Seite 408)
toupper, _toupper (Seite 409)

3.84 ispunct

Funktion

Bestimmen, ob ein Zeichen ein Satzzeichen ist

Syntax

```
#include <ctype.h>

int    ispunct (int c);
```

Parameter

Parametername	Bedeutung
c	Zeichen

Beschreibung

`ispunct` ist ein Makro, das als Eingabe–Argument ein einzelnes Zeichen verarbeitet und als Ergebnis eine Integer–Zahl liefert, die angibt ob `c` ein Satzzeichen (kein Steuerzeichen, Leerzeichen, alphanumerisches Zeichen) ist. Ein Wert ungleich 0 bedeutet TRUE, der Wert 0 FALSE. Das Makro ist in der Headerdatei CTYPE.H definiert.

Hinweis

Das Makro setzt voraus, dass `c` im Bereich des ASCII–Zeichensatzes liegt (Code 0 bis einschließlich 127 (dezimal)).

Rückgabewert

Rückgabewert	Bedeutung
TRUE	`c` ein Satzzeichen
FALSE	`c` kein Satzzeichen

Siehe auch

isalnum (Seite 289)
isalpha (Seite 290)
isascii (Seite 292)
iscntrl (Seite 293)
isdigit (Seite 294)
isgraph (Seite 295)
islower (Seite 296)
isprint (Seite 297)
isspace (Seite 299)
isupper (Seite 300)
isxdigit (Seite 301)
tolower, _tolower (Seite 408)
toupper, _toupper (Seite 409)

3.85 isspace

Funktion

Bestimmen, ob ein Zeichen ein Leerzeichen ist

Syntax

```
#include <ctype.h>

int    isspace (int c);
```

Parameter

Parametername	Bedeutung
c	Zeichen

Beschreibung

`isspace` ist ein Makro, das als Eingabe–Argument ein einzelnes Zeichen verarbeitet und als Ergebnis eine Integer–Zahl liefert, die angibt ob `c` ein Leerzeichen (Blank, TAB, CR, LF, VT oder FF) ist. Ein Wert ungleich 0 bedeutet TRUE, der Wert 0 FALSE. Das Makro ist in der Headerdatei CTYPE.H definiert.

Hinweis

Das Makro setzt voraus, dass `c` im Bereich des ASCII–Zeichensatzes liegt (Code 0 bis einschließlich 127 (dezimal)).

Rückgabewert

Rückgabewert	Bedeutung
TRUE	`c` ist ein Leerzeichen
FALSE	`c` ist kein Leerzeichen

Siehe auch

isalnum (Seite 289)
isalpha (Seite 290)
isascii (Seite 292)
iscntrl (Seite 293)
isdigit (Seite 294)
isgraph (Seite 295)
islower (Seite 296)
isprint (Seite 297)
ispunct (Seite 298)
isupper (Seite 300)
isxdigit (Seite 301)
tolower, _tolower (Seite 408)
toupper, _toupper (Seite 409)

3.86 isupper

Funktion

Bestimmen, ob ein Zeichen ein Großbuchstabe ist

Syntax

```
#include <ctype.h>

int    isupper (int c);
```

Parameter

Parametername	Bedeutung
c	Zeichen

Beschreibung

`isupper` ist ein Makro, das als Eingabe–Argument ein einzelnes Zeichen verarbeitet und als Ergebnis eine Integer–Zahl liefert, die angibt ob `c` ein Großbuchstabe ist. Ein Wert ungleich 0 bedeutet TRUE, der Wert 0 FALSE. Das Makro ist in der Headerdatei CTYPE.H definiert.

Hinweis

Das Makro setzt voraus, dass `c` im Bereich des ASCII–Zeichensatzes liegt (Code 0 bis einschließlich 127 (dezimal)).

Rückgabewert

Rückgabewert	Bedeutung
TRUE	`c` ein Großbuchstabe
FALSE	`c` kein Großbuchstabe

Siehe auch

isalnum (Seite 289)
isalpha (Seite 290)
isascii (Seite 292)
iscntrl (Seite 293)
isdigit (Seite 294)
isgraph (Seite 295)
islower (Seite 296)
isprint (Seite 297)
ispunct (Seite 298)
isspace (Seite 299)
isxdigit (Seite 301)
tolower, _tolower (Seite 408)
toupper, _toupper (Seite 409)

3.87 isxdigit

Funktion

Bestimmen, ob ein Zeichen eine Hexadezimalzahl ist

Syntax

```
#include <ctype.h>

int    isxdigit (int c);
```

Parameter

Parametername	Bedeutung
c	Zeichen

Beschreibung

`isxdigit` ist ein Makro, das als Eingabe–Argument ein einzelnes Zeichen verarbeitet und als Ergebnis eine Integer–Zahl liefert, die angibt ob `c` eine Hexadezimalzahl ([0–9], [A–F] oder [a–f]) ist. Ein Wert ungleich 0 bedeutet TRUE, der Wert 0 FALSE. Das Makro ist in der Headerdatei CTYPE.H definiert.

Hinweis

Das Makro setzt voraus, dass `c` im Bereich des ASCII–Zeichensatzes liegt (Code 0 bis einschließlich 127 (dezimal)).

Rückgabewert

Rückgabewert	Bedeutung
TRUE	`c` eine Hexadezimalzahl
FALSE	`c` keine Hexadezimalzahl

Siehe auch

isalnum (Seite 289)
isalpha (Seite 290)
isascii (Seite 292)
iscntrl (Seite 293)
isdigit (Seite 294)
isgraph (Seite 295)
islower (Seite 296)
isprint (Seite 297)
ispunct (Seite 298)
isspace (Seite 299)
isupper (Seite 300)
tolower, _tolower (Seite 408)
toupper, _toupper (Seite 409)

3.88 labs

Funktion

Absolutwert eines Long-Integer ermitteln

Syntax

```
#include <stdlib.h>

long    labs (long value);
```

Parameter

Parametername	Bedeutung
value	Long–Integer

Beschreibung

`labs` liefert den Absolutwert des in `value` angegebenen Long–Integers.

Rückgabewert

Rückgabewert	Bedeutung
long–Integer	Absolutwert

Beispiel

```
main ()
{
long x,y;
x = labs (23);   /* ergibt 23  */
y = labs (-105); /* ergibt 105 */
}
```

Siehe auch

abs (Seite 203)

fabs (Seite 244)

3.89 ldexp

Funktion

Double-Zahl generieren

Syntax

```
#include <math.h>

double    ldexp(
           double x,
           int n);
```

Parameter

Parametername	Bedeutung
x	Double–Zahl
n	Integer-Zahl (Exponent)

Beschreibung

`ldexp` berechnet `x * 2^n` und ist damit die Umkehrfunktion von `frexp`. Liegt das Ergebnis außerhalb des darstellbaren Zahlenbereichs, ruft `ldexp` die Fehlerbehandlungsfunktion `matherr` mit Exception–Typ OVERFLOW auf und setzt `errno` auf ERANGE. Abhängig vom Vorzeichen von `x` wird +/–HUGE_VAL als Ergebnis zurückgeliefert. Tritt ein Unterlauf auf, wird `matherr` mit DOMAIN aufgerufen und `errno` auf ERANGE gesetzt. Als Ergebnis wird in diesem Fall 0.0 zurückgegeben.

Rückgabewert

Rückgabewert	Bedeutung
Double–Zahl	x * 2^n
+/–HUGE_VAL	Nach Aufruf von `matherr` mit Exception–Typ OVERFLOW wird `errno` auf ERANGE gesetzt (Bereichsüberlauf)
0.0	Nach Aufruf von `matherr` mit Exception–Typ UNDERFLOW wird `errno` auf ERANGE gesetzt (Bereichsunterlauf)

Siehe auch

frexp (Seite 269)

3.90 ldiv

Funktion

Long–Integer–Division

Syntax

```
#include <stdlib.h>

ldiv_t    ldiv(
            long l1,
            long l2);
```

Parameter

Parametername	Bedeutung
l1	Long–Integer (Zähler)
l2	Long–Integer (Nenner)

Beschreibung

`ldiv` liefert das Ergebnis der Division `l1/l2`. Dabei wird in `ldiv_t.quot` der Quotient und in `ldiv_t.rem` der Divisionsrest abgelegt.

`ldiv_t` ist folgendermaßen deklariert:

```
typedef struct {
    long quot;
    long rem;
} ldiv_t;
```

Rückgabewert

Rückgabewert	Bedeutung
ldiv_t	Struktur mit Quotient und Divisionsrest

Siehe auch

div (Seite 235)

3.91 localtime

Funktion

Datum und Uhrzeit in Lokalzeit umwandeln

Syntax

```
#include <time.h>

struct    tm *localtime (long *clock);
```

Parameter

Parametername	Bedeutung
clock	Zeiger auf einen Long–Integer

Beschreibung

`localtime` wandelt Datum und Uhrzeit in Lokalzeit um, wobei Zeitangaben entsprechend Zeitzonen–Differenzen und gegebenenfalls Verschiebungen durch die Sommerzeit korrigiert werden. Die Sommerzeit beginnt am letzten Sonntag im März um 2.00 Uhr morgens und endet am letzten Sonntag im Oktober. `localtime` gibt einen Zeiger auf eine Struktur vom Typ `tm` zurück.

Die Struktur `tm` ist in TIME.H folgendermaßen definiert:

```
struct tm{
 int tm_sec;   /* Sekunden (0-59)              */
 int tm_min;   /* Minuten (0-59)               */
 int tm_hour;  /* Stunden (0-23)               */
 int tm_mday;  /* Monatstag (1-31)             */
 int tm_mon;   /* Monat (0-11)                 */
 int tm_year;  /* Jahr minus 1900              */
 int tm_wday;  /* Wochentag (Sonntag = 0)      */
 int tm_yday;  /* laufender Tag im Jahr (0-365) */
 int tm_isdst; /* ungleich 0 falls Sommerzeit  */
 };
```

Hinweis

Die Rückgabewerte zeigen auf einen Datenbereich, dessen Inhalt beim nächsten Aufruf überschrieben wird.

Verwenden Sie `_localtime32`, wenn Sie Zeitangaben über den 19. Januar 2038 hinaus benötigen.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf eine Struktur vom Typ `tm`

Beispiel

Das folgende Beispiel zeigt den Einsatz der verschiedenen CRUN–Zeitfunktionen:

```
#include <time.h>
main ()
{
 struct tm *ptm;
 char *date;
 long tadd;
 putenv ("TZ=MEZ-1");
 time (&tadd);
 date = ctime (&tadd);
 printf ("today's date is %s", date);

 ptm = localtime (&tadd);
 date = asctime (ptm);
 printf ("today's date is %s", date);

 ptm = gmtime (&tadd);
 date = asctime (ptm);
 printf ("Greenwich mean time is %s", date);
}
```

Siehe auch

_localtime32 (Seite 199)

ctime (Seite 231)

gmtime (Seite 286)

time (Seite 404)

strftime (Seite 384)

3.92 log

Funktion

Natürlicher Logarithmus

Syntax

```
#include <math.h>

double    log (double x);
```

Parameter

Parametername	Bedeutung
x	Double–Zahl

Beschreibung

`log` berechnet den natürlichen Logarithmus einer Double–Zahl. Ist das Argument negativ, ruft `log` die Fehlerbehandlungsfunktion `matherr` mit DOMAIN auf und setzt `errno` auf EDOM. Als Ergebnis wird in diesem Fall IND zurückgegeben. Ist das Argument 0, wird `matherr` mit SING aufgerufen und `errno` auf ERANGE gesetzt. Als Ergebnis wird in diesem Fall –HUGE_VAL zurückgegeben.

Rückgabewert

Rückgabewert	Bedeutung
Double–Zahl	Natürlicher Logarithmus
–HUGE_VAL	Nach Aufruf von `matherr` mit Exception–Typ SING wird `errno` auf ERANGE gesetzt (x = 0)
IND	Nach Aufruf von `matherr` mit Exception–Typ DOMAIN wird `errno` auf EDOM gesetzt (x < 0)

Siehe auch

exp (Seite 243)

log10 (Seite 308)

3.93 log10

Funktion

Dekadischer Logarithmus

Syntax

```
#include <math.h>

double    log10 (double x);
```

Parameter

Parametername	Bedeutung
x	Double–Zahl

Beschreibung

`log10` berechnet den Logarithmus zur Basis 10 aus einer Double–Zahl. Fehlerbehandlung wie bei der Funktion `log`.

Rückgabewert

Rückgabewert	Bedeutung
Double–Zahl	Dekadischer Logarithmus
–HUGE_VAL	Nach Aufruf von `matherr` mit Exception–Typ SING wird `errno` auf ERANGE gesetzt (x = 0)
IND	Nach Aufruf von `matherr` mit Exception–Typ DOMAIN wird `errno` auf EDOM gesetzt (x < 0)

Siehe auch

exp (Seite 243)

log (Seite 307)

3.94 longjmp

Funktion

Nichtlokalen Sprung ausführen

Syntax

```
#include <setjmp.h>

void    longjmp(
            jmp_buf env,
            int retvalue);
```

Parameter

Parametername	Bedeutung
env	Sprungziel–Information
retvalue	Rückgabewert von `setjmp`. Sonderfall: Wird hier als Parameter 0 angegeben, so gibt `setjmp` den Wert 1 zurück, denn der Rückgabewert 0 ist für die Funktion `setjmp` selbst reserviert.

Beschreibung

`longjmp` führt einen nichtlokalen Sprung aus, wobei die notwendigen Angaben über das Sprungziel aus dem Parameter `env` entnommen werden. `retvalue` enthält im Normalfall den Rückgabewert der Funktion. Die Initialisierung von `env` erfolgt durch die Funktion `setjmp`. Nach Aufruf von `longjmp` wird das Programm beim zugehörigen `setjmp` fortgesetzt (so als ob `setjmp` aufgerufen wurde). Der Rückgabewert des zugehörigen `setjmp` ist dann der Parameter `retvalue` von `longjmp`.

Siehe auch

setjmp (Seite 360)

3.95 lseek

Funktion

Dateizeiger in einer Datei positionieren

Syntax

```
#include <io.h>

long    lseek(
           int handle,
           long offset,
           int fromwhere);
```

Parameter

Parametername	Bedeutung
handle	Deskriptor
offset	Offset in Bytes
fromwhere	Startposition von offset

Beschreibung

`lseek` positioniert den Dateizeiger für die mit `handle` verbundene Datei für den nächsten Lese– oder Schreibzugriff. Die Position wird aus dem mit `offset` angegebenen Abstandswert und dem mit `fromwhere` angegebenen Basiswert berechnet. Das Argument `fromwhere` kann drei Werte annnehmen:

SEEK_SET (=0)	Dateianfang
SEEK_CUR (=1)	Aktuelle Position
SEEK_END (=2)	Dateiende

Hinweis

Im Textmodus kann `lseek` wegen der Umwandlung ‘\n’ bzw. ’\r\n’ nur eingeschränkt verwendet werden.

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
Long–Integer	Dateizeiger
–1	Funktion nicht erfolgreich

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt:

Fehlercode	Bedeutung
EBADF	Deskriptor ungültig
EINVAL	`fromwhere` ungültig

Siehe auch

open (Seite 325)
read (Seite 342)
write (Seite 426)

3.96 malloc

Funktion

Speicher anfordern

Syntax

```
#include <malloc.h>

void    *malloc (size_t size);
```

Parameter

Parametername	Bedeutung
size	Größe des Speichers in Byte

Beschreibung

`malloc` fordert Speicher der Größe `size` in Byte an.

Hinweis

Der Speicherbereich wird entweder gezielt durch Aufruf der Funktion `free`, `RmFree` oder `RmFreeAll` freigegeben oder automatisch bei Beendigung der Task durch `exit`, `x_cr_killtsk`, `RmDeleteTask`, `RmEndTask` oder `RmKillTask`.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf einen Speicher der angeforderten Größe
NULL–Zeiger	wenn `size` gleich 0 ist oder nicht genug Speicher verfügbar ist

Fehlercodes

Im Fehlerfall wird `errno` auf folgenden Wert gesetzt:

Fehlercode	Bedeutung
ENOMEM	Kein freier Speicherplatz verfügbar

Siehe auch

calloc (Seite 217)
free (Seite 265)
realloc (Seite 344)
RmFree (Seite 62)
RmFreeAll (Seite 63)
xinitc (Seite 436)

3.97 matherr

Funktion

Anwendungsspezifische Fehlerbehandlung

Syntax

```
#include <math.h>

int    matherr (struct exception *e);
```

Parameter

Parametername	Bedeutung
e	Zeiger auf Struktur vom Typ `exception`

Beschreibung

`matherr` ist eine globale, anwendungsbezogene (nicht taskspezifische) Funktion zur Behandlung mathematischer Fehler, die dem Anwender zur Verfügung gestellt wird. Bei der System–Generierung muss `matherr` vor die Numerik–Bibliothek gestellt werden. `matherr` erhält als Argument einen Zeiger *e*, der auf eine Struktur vom Typ `exception` zeigt:

```
struct exception {
    int type;
    char *name;
    double arg1, arg2;
    double retval;
 } *e;
```

`type` bezeichnet den Exception–Typ:

DOMAIN	Bereichsfehler
SING	Singularität
OVERFLOW	Bereichsüberschreitung
PLOSS	Teilweiser Verlust der Genauigkeit
TLOSS	Totaler Verlust der Genauigkeit
UNDERFLOW	Bereichsunterschreitung

`name` ist ein Zeiger auf einen mit ‘\0’ abgeschlossenen String, der auf den Namen der Funktion zeigt, für die ein ungültiger Eingabewert vorliegt. Der Funktionsname wird in Kleinschreibung angegeben.

`arg1` und `arg2` ist/sind das/die Argument(e) in dem Aufruf der Funktion.

`retval` ist der Wert, den `matherr` zurückliefert, wenn ein spezieller Wert zurückgegeben werden soll. In diesem Fall muss `matherr` als Ergebnis einen Wert ungleich 0 zurückgeben (entspricht dem Ergebnis der mathematischen Operation).

Hinweis

CRUN enthält für `matherr` eine Dummy–Funktion, die als Ergebnis 0 zurückliefert, d.h. defaultmäßig wird keine Korrektur durchgeführt.

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
0	Keine Korrektur der Fehlerursache
≠0	Korrektur der Fehlerursache

Beispiel

Nachstehend wird eine Fehlerbehandlung beim Wurzelziehen aus einer negativen Zahl gezeigt. Im Beispiel wird angenommen, dass z.B. der Betrag des Arguments nicht interessiert.

```
#include <stdio.h>
#include <string.h>
#include <math.h>

int matherr(struct exception *e)
{
  /* Bereichsfehler */
  if(e->type==DOMAIN)
  {
  /* Prüfen ob sqrt-Funktion */
  if(!strcmp(e->name,"sqrt"))
  {
            /* Fehler behandeln */
            a->retval=-sqrt(-(e->arg1));
            return 1;
            }
  }
 / * Andere Fehler ignorieren */
  return 0;
}
void main(void)
{
  double result;
  result=sqrt(-1.0);
  printf("Quadratwurzel -1.0=%g\n",result);
  /* es wird +1.0 ausgegeben */
}
```

3.98 memccpy

Funktion

Zeichen von einem Quell- in einen Zielbereich kopieren

Syntax

```
#include <string.h>

void    *memccpy(
            void *s1,
            const void *s2,
            int c,
            size_t n);
```

Parameter

Parametername	Bedeutung
s1	Zeiger auf Zielbereich
s2	Zeiger auf Quellbereich
c	Vergleichszeichen
n	Anzahl zu kopierender Zeichen

Beschreibung

`memccpy` kopiert maximal `n` Zeichen vom Quellbereich `s2` in den Zielbereich `s1`. Der Vorgang wird beendet, wenn erstmals das als `c` angegebene Vergleichszeichen kopiert worden ist, oder wenn insgesamt `n` Zeichen kopiert worden sind. Die Funktion gibt den NULL–Zeiger zurück, wenn `c` nicht erkannt wird, ansonsten einen Zeiger auf das nach `c` folgende Zeichen in `s1`.

Hinweis

Diese Funktion bearbeitet den Inhalt von Speicherbereichen, die, im Gegensatz zur entsprechenden String–Funktion, nicht mit einem Null–Byte, sondern durch die Anzahl der Zeichen begrenzt sind.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf das nach `c` folgende Zeichen im Zielbereich
NULL–Zeiger	`n` Zeichen wurden kopiert

Siehe auch

memcpy (Seite 318)

memmove (Seite 319)

memset (Seite 320)

strcmp (Seite 380)

strcpy (Seite 381)

3.99 memchr

Funktion

Zeichen suchen

Syntax

```
#include <string.h>

void    *memchr(
            const void *s,
            int c,
            size_t n);
```

Parameter

Parametername	Bedeutung
s	Zeiger auf Bereich
c	Vergleichszeichen
n	Anzahl zu vergleichender Zeichen

Beschreibung

`memchr` sucht in dem durch `s` angegebenen Bereich nach dem ersten Auftreten von Zeichen `c`. Wird das Vergleichszeichen in den ersten `n` Zeichen nicht gefunden, wird ein NULL–Zeiger zurückgegeben, ansonsten ein Zeiger auf das gefundene Zeichen `c`.

Hinweis

Diese Funktion bearbeitet den Inhalt eines Speicherbereichs, der, im Gegensatz zur entsprechenden String–Funktion, nicht mit einem Null–Byte, sondern durch die Anzahl der Zeichen begrenzt ist.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf das Zeichen `c`
NULL–Zeiger	Zeichen `c` nicht gefunden

Siehe auch

memcmp (Seite 317)

strchr (Seite 379)

strcmp (Seite 380)

strncmp (Seite 388)

3.100 memcmp

Funktion

Inhalt von zwei Bereichen vergleichen

Syntax

```
#include <string.h>

int    memcmp(
          const void *s1,
          const void *s2,
          size_t n);
```

Parameter

Parametername	Bedeutung
s1	Zeiger auf ersten Bereich
s2	Zeiger auf zweiten Bereich
n	Anzahl zu vergleichender Zeichen

Beschreibung

`memcmp` vergleicht den Inhalt von zwei Bereichen `s1` und `s2` auf einer Länge von `n` Zeichen. Sind beide Bereiche identisch, wird 0 zurückgegeben. Wenn das erste sich unterscheidende Zeichen in `s2` größer als das entsprechende Zeichen in `s1` ist, wird eine positive Zahl zurückgegeben, anderenfalls eine negative Zahl.

Hinweis

Diese Funktion bearbeitet den Inhalt von Speicherbereichen, die, im Gegensatz zur entsprechenden String–Funktion, nicht mit einem Null–Byte, sondern durch die Anzahl der Zeichen begrenzt sind.

Rückgabewert

Rückgabewert	Bedeutung
Integer–Zahl	größer/gleich/kleiner 0, je nach Ergebnis der Vergleichsoperation

Siehe auch

memchr (Seite 316)

strchr (Seite 379)

strcmp (Seite 380)

strncmp (Seite 388)

3.101 memcpy

Funktion

Zeichen von einem Quell- in einen Zielbereich kopieren

Syntax

```
#include <string.h>

void    *memcpy(
            void *s1,
            const void *s2,
            size_t n);
```

Parameter

Parametername	Bedeutung
s1	Zeiger auf Zielbereich
s2	Zeiger auf Quellbereich
n	Anzahl zu kopierender Zeichen

Beschreibung

`memcpy` kopiert `n` Zeichen von Bereich `s2` nach Bereich `s1`. Die Funktion gibt einen Zeiger auf `s1` zurück.

Hinweis

Diese Funktion bearbeitet den Inhalt von Speicherbereichen, die, im Gegensatz zur entsprechenden String–Funktion, nicht mit einem Null–Byte, sondern durch die Anzahl der Zeichen begrenzt sind.

Diese Funktion darf bei überlappenden Speicherbereichen nicht benutzt werden (siehe `memmove`).

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf den Bereich `s1`

Siehe auch

memccpy (Seite 315)

memmove (Seite 319)

memset (Seite 320)

strcpy (Seite 381)

strncpy (Seite 389)

3.102 memmove

Funktion

Zeichen von einem Quell- in einen Zielbereich verschieben

Syntax

```
#include <string.h>

void    *memmove(
              void *s1,
              const void *s2,
              size_t n);
```

Parameter

Parametername	Bedeutung
s1	Zeiger auf Zielbereich
s2	Zeiger auf Quellbereich
n	Anzahl zu kopierender Zeichen

Beschreibung

`memmove` verschiebt `n` Zeichen von Bereich `s2` nach Bereich `s1`. Die Funktion gibt einen Zeiger auf `s1` zurück.

Hinweis

Diese Funktion bearbeitet den Inhalt von Speicherbereichen, die, im Gegensatz zur entsprechenden String–Funktion, nicht mit einem Null–Byte, sondern durch die Anzahl der Zeichen begrenzt sind.

Diese Funktion kann auch bei überlappenden Speicherbereichen benutzt werden (siehe `memcpy`).

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf den Bereich `s1`

Siehe auch

memccpy (Seite 315)

memcpy (Seite 318)

memset (Seite 320)

strcpy (Seite 381)

strncpy (Seite 389)

3.103 memset

Funktion

Zeichen in einen Bereich kopieren

Syntax

```
#include <string.h>

void    *memset(
            void *s,
            int c,
            size_t n);
```

Parameter

Parametername	Bedeutung
s	Zeiger auf Bereich
c	Zeichen
n	Anzahl zu kopierender Zeichen

Beschreibung

`memset` kopiert `n` mal das Zeichen `c` ab Adresse `s`. Die Funktion gibt einen Zeiger auf `s` zurück.

Hinweis

Diese Funktion bearbeitet den Inhalt von Speicherbereichen, die, im Gegensatz zur entsprechenden String–Funktion, nicht mit einem Null–Byte, sondern durch die Anzahl der Zeichen begrenzt sind.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf den Bereich `s`

Siehe auch

memccpy (Seite 315)

memcpy (Seite 318)

memmove (Seite 319)

strcpy (Seite 381)

strncpy (Seite 389)

3.104 mkdir

Funktion

Verzeichnis erzeugen

Syntax

```
#include <direct.h>

int    mkdir (char *dirname);
```

Parameter

Parametername	Bedeutung
dirname	Name des zu erzeugenden Verzeichnisses

Beschreibung

`mkdir` erzeugt das in `dirname` angegebene Verzeichnis.

Hinweis

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich abgearbeitet
-1	Fehler

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt:

Fehlercode	Bedeutung
EACCES	Ungültige Zugriffsrechte für `dirname`
EBUSY	`dirname` ist bereits mit lock option geöffnet
EEXIST	`dirname` existiert bereits
EIO	physikalischer I/O Fehler
EMFILE	interne HSFS Limits erreicht
ENOENT	`dirname` ungültig
ENOMEM	Kein freier Speicherplatz verfügbar
ENOSPC	Datenträger ist voll
ERANGE	Unerwarteter Fehlercode vom HSFS

Siehe auch

rmdir (Seite 349)

3.105 mktime

Funktion

Lokale Uhrzeit umwandeln

Syntax

```
#include <time.h>

time_t    mktime (struct tm *timeptr);
```

Parameter

Parametername	Bedeutung
timeptr	Zeiger auf Struktur vom Typ `tm`

Beschreibung

`mktime` wandelt die in der Struktur `tm` abgelegte lokale Uhrzeit in das von der Funktion `time` benötigte Uhrzeitformat um. Die ursprünglichen Komponenten `tm_wday` und `tm_yday` der Struktur `tm` werden nicht benötigt und vor dem Rücksprung der Funktion aktualisiert.

Die Struktur `tm` ist in TIME.H folgendermaßen definiert:

```
struct tm{
 int tm_sec;   /* Sekunden (0-59)               */
 int tm_min;   /* Minuten (0-59)                */
 int tm_hour;  /* Stunden (0-23)                */
 int tm_mday;  /* Monatstag (1-31)              */
 int tm_mon;   /* Monat (0-11)                  */
 int tm_year;  /* Jahr minus 1900               */
 int tm_wday;  /* Wochentag (Sonntag = 0)       */
 int tm_yday;  /* laufender Tag im Jahr (0-365) */
 int tm_isdst; /* ungleich 0 falls Sommerzeit   */
};
```

Hinweis

Die Rückgabewerte zeigen auf einen Datenbereich, dessen Inhalt beim nächsten Aufruf überschrieben wird.

Rückgabewert

Rückgabewert	Bedeutung
time_t	Uhrzeit Negativ bei Zeitangaben über den 19. Januar 2038 hinaus
–1	Funktion nicht erfolgreich

Siehe auch

asctime (Seite 206)

ctime (Seite 231)

gmtime (Seite 286)

localtime (Seite 305)

time (Seite 404)

tzset (Seite 410)

strftime (Seite 384)

3.106 modf

Funktion

Double-Zahl in eine Ganzzahl und einen Bruchteil zerlegen

Syntax

```
#include <math.h>

double    modf(
            double x,
            double *nptr);
```

Parameter

Parametername	Bedeutung
x	Double–Zahl
nptr	Zeiger auf Double–Zahl

Beschreibung

`modf` zerlegt eine Double–Zahl in einen ganzzahligen Teil und einen Teil, der kleiner als 1 ist. Die Funktion überschreibt die Double–Zahl, auf die `nptr` zeigt, mit dem ganzzahligen Anteil von `x`. Als Rückgabewert liefert die Funktion den Bruchteil von `x`.

Rückgabewert

Rückgabewert	Bedeutung
Double–Zahl	Bruchteil (der ganzzahlige Anteil ist über den Zeiger `nptr` zu erreichen)

3.107 mount

Funktion

Gerät beim HSFS-Dateisystem anmelden

Syntax

```
#include <io.h>

int    mount(
         const char *volume_name,
         int format);
```

Parameter

Parametername	Bedeutung
volume_name	Laufwerkname
format	Aufzeichnungsformat des Laufwerks

Beschreibung

`mount` meldet ein Gerät beim HSFS–Dateisystem an. `volume_name` muss ein gültiger Laufwerkname mit einer Länge von 1 oder 2 Zeichen sein (z.B. "A", "C", "M0"). `format` gibt das Aufzeichnungsformat des Geräts an und muss auf 0 gesetzt sein. Aus Kompatibilitätsgründen ist auch noch der Wert 2 zugelassen.

Hinweis

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich
–1	Funktion nicht erfolgreich

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt:

Fehlercode	Bedeutung
EACCES	Ungültiger Laufwerksname
EINVAL	Ungültiger Wert für `format`
EIO	Physikalischer Schreib/Lesefehler
EMFILE	Interner Pufferüberlauf
ERANGE	Unerwarteter Fehlercode von HSFS

Siehe auch

dismount (Seite 234)

3.108 open

Funktion

Datei für Lese- oder Schreibzugriffe öffnen

Syntax

```
#include <io.h>
#include <fcntl.h>

int    open(
        char *pathname,
        int access [,
        int permission]);
```

Parameter

Parametername	Bedeutung
pathname	Zeiger auf Dateispezifikation
access	Zugriffsparameter
permission	Zugriffsattribut

Beschreibung

`open` öffnet die mit Pfadnamen `pathname` angegebene Datei (z.B. "\\tmp\\test.log") für Lese– oder Schreibzugriffe.

Der Zugriffsparameter `access` entsteht durch eine Oder–Verknüpfung der Flags aus den nachfolgenden zwei Listen: Es ist genau ein Parameter aus der Liste 1 anzugeben, zusätzlich können Parameter aus der Liste 2 logisch oder–verknüpft werden.

Liste 1: Lese/Schreibflags

O_RDONLY	Öffnen zum Lesezugriff
O_WRONLY	Öffnen zum Schreibzugriff
O_RDWR	Öffnen zum Lese– und Schreibzugriff

Liste 2:Sonstige Flags

O_APPEND	Dateizeiger wird nach dem Öffnen auf das Ende der Datei gesetzt.
O_CREAT	Falls die Datei existiert, ist das Flag wirkungslos. Falls die Datei nicht existiert, wird eine neue Datei erzeugt und das mit permission angegebene Attribut verwendet.
O_TRUNC	Existiert die Datei, wird sie beim Öffnen auf Länge Null reduziert.
O_TEXT	Öffnen im Textmodus (nicht gleichzeitig mit O_BINARY)
O_BINARY	Öffnen im Binärmodus (Default–Einstellung) (nicht gleichzeitig mit O_TEXT anzugeben)
O_CLOSEONEXIT	Die Datei wird beim Taskende mit exit automatisch geschlossen

Das Zugriffsattribut wird mit `permission` gesteuert:

S_IREAD	Nach dem Schließen besitzt die Datei das Attribut ‘Nur–Lesen’.
S_IREAD | S_IWRITE	Lese– und Schreibzugriff auf die Datei möglich.

Hinweis

Im Textmodus wird beim Schreiben '\n' in '\r\n' umgewandelt. Beim Lesen wird '\r\n' in '\n' umgewandelt.

Deskriptoren sind taskspezifisch und können nicht zwischen Tasks ausgetauscht werden.

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
≥0	Deskriptor (Funktion erfolgreich)
–1	Funktion nicht erfolgreich

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt:

Fehlercode	Bedeutung
EACCES	Angabe von O_TEXT nicht zulässig
EBUSY	Datei ist von einer anderen Task geöffnet
EINVAL	Ungültiger Parameter
EIO	Physikalischer Schreib/Lesefehler
EMFILE	Interner Pufferüberlauf
ENOENT	Angegebene Datei existiert nicht und O_CREAT ist nicht gesetzt
ENOMEM	Es konnte nicht ausreichend Speicher aus dem angegebenen Pool entnommen werden.
ENOSPC	Kein Speicherplatz auf Laufwerk vorhanden oder Verzeichnis voll
ENOTDIR	Eine Komponente des Pfadnamen–Präfixes ist kein Verzeichnis (Laufwerk wurde eventuell nicht angemeldet (`mount`))

Siehe auch

close (Seite 227)

read (Seite 342)

write (Seite 426)

3.109 perror

Funktion

CRUN-Fehlermeldung ausgeben

Syntax

```
#include <stdio.h>

void    perror (char *str);
```

Parameter

Parametername	Bedeutung
str	Zeiger auf einen String

Beschreibung

`perror` gibt eine Meldung auf `stderr` (Standard–Fehler–Kanal) aus, die den Fehler beschreibt, der zuletzt bei einem Systemaufruf aufgetreten ist.

Zuerst wird der String, auf den `str` zeigt, ausgegeben. Danach folgen ein Punkt, ein Leerzeichen, die Meldung und ein Zeilenwechsel. Der String sollte stets den Namen des Programms enthalten, in dem der Fehler aufgetreten ist.

Die Fehlernummer wird der externen Variablen `errno` entnommen, die bei jedem Fehler neu gesetzt wird (bei einem erfolgreichen Aufruf wird errno nicht automatisch rückgesetzt).

Falls `str` auf einen NULL–String zeigt, gibt `perror` nur die Meldung und anschließend einen Zeilenwechsel aus.

Siehe auch

errno, errno2 (Seite 240)

strerror (Seite 383)

sys_errlist, sys_nerr (Seite 401)

3.110 pow

Funktion

Potenzfunktion

Syntax

```
#include <math.h>

double    pow(
           double x,
           double y);
```

Parameter

Parametername	Bedeutung
x	Double–Zahl
y	Double–Zahl

Beschreibung

`pow` berechnet die Potenzfunktion x^y (Powerfunction):

- Für x ≠ 0 und y = 0 wird der Wert 1,0 zurückgeliefert.
- Wenn x < 0, muss y ganzzahlig sein. In diesem Fall wird auch das korrekte Vorzeichen ermittelt.
- Ist x < 0 und y nicht ganzzahlig, ruft `pow` die Fehlerbehandlungsfunktion `matherr` mit Exception–Typ DOMAIN auf und setzt `errno` auf EDOM. Als Ergebnis wird in diesem Fall IND zurückgegeben. Ebenso wird in folgenden zwei Fällen verfahren:
  x < 0 und y größer als die größte darstellbare Ganzzahl
  x = 0 und y < 0

Bei einem Bereichsüberlauf bzw. –unterlauf ruft `pow` die Fehlerbehandlungsfunktion `matherr` mit Exception–Typ OVERFLOW bzw. UNDERFLOW auf und setzt `errno` auf ERANGE. In diesem Fall wird +HUGE_VAL bzw. 0 zurückgegeben.

Rückgabewert

Rückgabewert	Bedeutung
Double–Zahl	x^y
IND	Nach Aufruf von `matherr` mit Exception–Typ DOMAIN wird `errno` auf EDOM gesetzt (Bereichsfehler)
+HUGE_VAL	Nach Aufruf von `matherr` mit Exception–Typ OVERFLOW wird `errno` auf ERANGE (Bereichsüberlauf)
0	Nach Aufruf von `matherr` mit Exception–Typ UNDERFLOW wird `errno` auf ERANGE gesetzt (Bereichsunterlauf)

Siehe auch

exp (Seite 243)

3.111 printf

Funktion

Formatierte Ausgabe auf den Standard-Ausgabe-Kanal (stdout)

Syntax

```
#include <stdio.h>

int    printf(
         char *format [,arg] ...);
```

Parameter

Parametername	Bedeutung
format	Zeiger auf einen String
arg	auszugebendes Argument

Beschreibung

`printf` verarbeitet den Eingabe–String `format`, interpretiert verschiedene Format–Kommandos und gibt das Ergebnis auf dem Standard–Ausgabe–Kanal aus.

`printf` gibt die Anzahl der übertragenen Zeichen zurück. Bei einem Ausgabe–Fehler wird ein Wert kleiner 0 zurückgegeben.

`format` ist ein String mit eingefügten Format–Anweisungen und dazugehörigen Argumenten: `printf ("...",arg1, arg2...)`. Der Eingabe–String enthält drei verschiedene Elemente:

1. Zeichen, die auf dem Standard–Ausgabe–Kanal ausgegeben werden.
2. Escape–Sequenzen, die Zeichen darstellen (keine Grafik–Zeichen).
3. Format–Anweisungen für Strings, Zeichen und Zahlen.

Zugelassene Escape–Sequenzen und ihre Wirkung auf geeigneten Ausgabegeräten:

\b

Backspace: Das nächste Zeichen wird ein Zeichen vor der aktuellen Position gedruckt, es sei denn, dass diese mit dem Zeilenanfang zusammenfällt.

\f

Seitenvorschub: Das nächste Zeichen wird am Anfang der nächsten logischen Seite gedruckt.

\n

Zeilenwechsel: Das nächste Zeichen wird am Anfang der nächsten Zeile gedruckt.

\r

Carriage Return: Das nächste Zeichen wird am Anfang der aktuellen Zeile gedruckt.

\t

Horizontal–Tab: Das nächste Zeichen wird an der nächsten Horizontal–Tab–Position der aktuellen Zeile gedruckt. Die Abstände der Horizontal–Tabs sind implementierungsabhängig.

\v

Vertikal–Tab: Das nächste Zeichen wird an der nächsten Vertikal–Tab–Position gedruckt. Die Abstände der Vertikal–Tabs sind implementierungsabhängig.

Vor allen Format–Anweisungen steht das Prozentzeichen %. Am Ende der Anweisung steht eines der Konvertierungs–Zeichen. Zwischen dem Prozent– und dem Konvertierungs–Zeichen können wahlweise Zeichen stehen, die das Standard–Format abändern:

–

Das Minuszeichen kann wahlweise nach dem Prozentzeichen stehen und bewirkt linksbündig ausgerichtete Ausgabe.

#

Das Doppelkreuz # kann wahlweise danach folgen. Es bewirkt, dass vor einer oktalen oder hexadezimalen Zahl die Zeichen "0" , "0x" oder "0X" ausgedruckt werden.

+

Das Pluszeichen bewirkt, dass eine konvertierte Zahl immer mit ihrem Vorzeichen (+ oder –) ausgegeben wird.

0

Als nächstes Zeichen kann wahlweise eine Null stehen. In der Ausgabe erscheinen Nullen anstelle von Leerzeichen.

(Leerzeichen)

Ein Leerzeichen bewirkt, dass vor das Ergebnis einer Vorzeichen–Konvertierung ein Leerzeichen geschrieben wird, wenn kein Vorzeichen davor steht. Falls jedoch im Ergebnis ein Leer– und ein Vorzeichen erscheinen, wird das Leerzeichen ignoriert.

*

Als nächstes Zeichen kann wahlweise der Stern * folgen. Er bewirkt, dass das nächste Argument (das vor dem auszugebenden Element steht) als Feldlänge interpretiert wird.

(Dezimal)

Ansonsten kann eine Dezimalzahl angegeben werden, die die Feldlänge definiert. Ist die Ausgabe kleiner, so wird sie mit Leerzeichen oder Nullen aufgefüllt.

Soll ein String, eine Integer–Zahl oder eine Gleitkomma–Zahl ausgegeben werden, so kann man nach "." (Dezimalpunkt) die entsprechende Genauigkeit bestimmen, d.h.: die Stellen nach einem Komma, die maximale Länge eines String oder die Mindestanzahl von Ziffern in einer Integer–Zahl (das Feld für die Integer–Zahl wird mit führenden Nullen aufgefüllt).

l

Mit dieser Option wird angegeben, dass das Argument keine Integer–Zahl, sondern ein Long–Integer ist (falls nach l eine der Optionen d, i, o, u oder x folgt); analog wird ein Argument mit doppelter Gleitkomma–Genauigkeit anstelle von einfacher Genauigkeit gekennzeichnet (wenn nach l eine der Optionen f oder e folgt). l beeinflußt das Ausgabe–Format nicht.

ll

Mit dieser Option wird angegeben, dass das Argument keine Integer-Zahl, sondern ein long long-Integer ist (falls nach ll eine der Optionen d oder u folgt). ll beeinflußt das Ausgabe-Format nicht. Diese Option wird nur für GNU unterstützt.

h

Mit dieser Option wird angegeben, dass das Argument keine Integer–Zahl, sondern ein Short–Integer ist (falls nach h eine der Optionen d, i, o, u oder x folgt). h beeinflußt das Ausgabe–Format nicht.

Das letzte Format–Element in der Format–Anweisung gibt die Art der Ausgabe eines Datenelements an.

c

Das auszugebende Datenelement ist ein 7–bit–ASCII–Zeichen.

s

Das Argument ist ein String oder Character Pointer. Alle Zeichen bis zum abschließenden Null–Byte '\0' werden ausgegeben, außer es ist ein Ausgabefeld mit beschränkter Länge definiert.

d

Das Datenelement soll als Dezimalzahl dargestellt werden.

i

Das Datenelement soll als Dezimalzahl dargestellt werden.

o

Das Datenelement soll als Oktalzahl dargestellt werden.

u

Das Datenelement soll als Dezimalzahl ohne Vorzeichen dargestellt werden.

x, X

Das Datenelement soll als Hexadezimalzahl dargestellt werden. Bei der Option x wird in die Buchstaben "abcdef", bei der Option X wird in die Buchstaben "ABCDEF" konvertiert.

f

Das Datenelement soll als Gleitkomma–Zahl mit doppelter Genauigkeit dargestellt werden. Es wird dargestellt mit optionalem Minuszeichen, Ziffern, einem Dezimalkomma und weiteren Ziffern (– ddd.ddd). Ist keine Genauigkeit angegeben, stehen sechs Ziffern nach dem Komma.

e, E

Das Datenelement ist eine Gleitkomma–Zahl mit einfacher oder doppelter Genauigkeit und soll in exponentieller Schreibweise dargestellt werden. Man erhält die gleiche Darstellung wie bei Option f, außer dass vor dem Komma nur eine Ziffer steht und nach der Zahl (– d.ddde+–ddd) ein dreistelliger Exponent folgt. Bei Option E wird der Exponent mit großem anstatt mit kleinem e dargestellt.

g, G

Das Datenelement soll wie bei f oder e (bzw. E, wenn G angegeben ist) dargestellt werden, wobei die angegebene Anzahl signifikanter Ziffern berücksichtigt wird. Die Darstellungsweise hängt von der Größe des konvertierten Wertes ab: Die Darstellung richtet sich nur dann nach der Option e, wenn der Exponent, der sich nach der Konvertierung ergibt, kleiner als –4 oder größer als die angegebene Anzahl von Ziffern ist.

p

Das Datenelement wird als Zeiger in der Darstellung mit 8 hexadezimalen Ziffern ausgegeben.

%

Das Prozent–Zeichen selbst wird ausgegeben. Steht nach dem Prozent–Zeichen kein gültiges Konvertierungs–Zeichen, so ist das Ergebnis der Konvertierung nicht definiert.

Hinweis

Die Ausgabe auf stdout (Standard–Ausgabe) erfolgt normalerweise gepuffert. Bei Pufferung wird die Ausgabe erst dann ausgeführt, wenn der Puffer voll bzw. die Puffergröße gleich 1 ist, bzw. bei Zeilen–Pufferung auch wenn ein Zeilenende '\n' ausgegeben wird. Soll die Ausgabe schon früher erfolgen, muss der Funktionsaufruf `fflush` eingesetzt werden.

Bei Ausgabe mit printf wird der Pufferinhalt am Ende automatisch ausgegeben.

`printf` und `scanf` von 2 verschiedenen Tasks können auf einer Konsole nicht gleichzeitig abgearbeitet werden. Erst wenn der `printf`-Aufruf bzw. der `scanf`-Aufruf einer Task abgearbeitet ist, kann der `printf`- bzw. `scanf`-Aufruf der anderen Task abgearbeitet werden. Während eine Task im `scanf`-Aufruf auf Eingabe wartet, kann kein `printf`-Aufruf von einer anderen Task auf die gleiche Konsole ausgegeben werden.

Rückgabewert

Rückgabewert	Bedeutung
Anzahl	der ausgegebenen Zeichen (Funktion erfolgreich)
–1	Fehler (Funktion nicht erfolgreich)

Beispiel

Ausgabe von "100% participation" auf dem Standard–Ausgabe–Kanal:

```
main () {printf("%d%% participation",100);}
```

Siehe auch

fprintf (Seite 260)
scanf (Seite 350)
snprintf (Seite 370)
sprintf (Seite 372)
vfprintf (Seite 414)
vprintf (Seite 418)
vsnprintf (Seite 421)
vsprintf (Seite 422)
vfscanf (Seite 416)
vscanf (Seite 420)
vsscanf (Seite 424)

3.112 putc

Funktion

Zeichen auf einen Stream schreiben

Syntax

```
#include <stdio.h>

int    putc(
         int c,
         FILE *stream);
```

Parameter

Parametername	Bedeutung
c	Zeichen
stream	Stream mit Struktur vom Typ FILE

Beschreibung

`putc` schreibt das Zeichen `c` in die mit `stream` angegebene Datei. Ein eventuell definierter Dateizeiger zeigt auf die Position, ab der geschrieben wird.

Hinweis

Da `putc` als Makro implementiert ist, behandelt es ein Stream–Argument nicht korrekt mit unvorhersehbaren Nebenwirkungen. Insbesondere liefert der Aufruf `putc (*f++)` nicht das erwartete Ergebnis. In diesem Fall sollte `fgetc` verwendet werden.

Bei Streams im Textmodus wird ‘\n’ in ‘\r\n’ umgewandelt.

Rückgabewert

Rückgabewert	Bedeutung
Integer–Zahl	Übertragenes Zeichen
EOF	Funktion nicht erfolgreich, z.B. wenn die Ausgabedatei `stream` nicht zum Schreiben geöffnet oder kein Platz mehr in dieser vorhanden ist

Siehe auch

fopen (Seite 258)

fputc (Seite 262)

printf (Seite 329)

putchar (Seite 334)

puts (Seite 336)

putw (Seite 337)

3.113 putchar

Funktion

Zeichen auf die Standard-Ausgabe schreiben

Syntax

```
#include <stdio.h>

int    putchar (int c);
```

Parameter

Parametername	Bedeutung
c	Zeichen

Beschreibung

`putchar` schreibt das nächste Zeichen auf die Standard-Ausgabe. Die Funktion ist äquivalent zu `putc (c, stdout)`.

Hinweis

Bei `putchar` erfolgt die Ausgabe eines Zeichens wegen der internen Pufferung nicht sofort nach dem Aufruf von `putchar`, sondern erst wenn der Puffer voll ist. Ist dieser Effekt nicht erwünscht, so kann direkt nach `putchar` die Funktion `fflush` aufgerufen werden um den Puffer sofort zu leeren. Syntax: `fflush(stdin);`

Bei Streams im Textmodus wird '\n' in '\r\n' umgewandelt.

Rückgabewert

Rückgabewert	Bedeutung
Integer–Zahl	Übertragenes Zeichen
EOF	Fehler oder Dateiende erreicht

Siehe auch

fopen (Seite 258)

fputc (Seite 262)

putc (Seite 333)

puts (Seite 336)

putw (Seite 337)

3.114 putenv

Funktion

Umgebungs–Variable ändern oder hinzufügen

Syntax

```
#include <stdlib.h>

int    putenv (char *string);
```

Parameter

Parametername	Bedeutung
string	Zeiger auf einen String mit dem Format "`name=value`"

Beschreibung

`putenv` setzt die Umgebungs–Variable `name` auf den Wert `value`. Das Argument string zeigt auf einen String mit dem Format "`name=value`". Ist die Variable bereits vorhanden, wird sie geändert, sonst wird sie erzeugt. In beiden Fällen wird der mit `string` angegebene String ein Element der Umgebung. Ein Ändern des Strings ändert somit auch die Umgebung. Der von `string` belegte Platz wird freigegeben, wenn `putenv` ein neuer String übergeben wird, der die Variable `name` neu definiert.

Für jede Task existiert eine eigene Umgebung. Der Aufbau der Umgebung ist unter `x_cr_initenv` beschrieben.

Hinweis

`putenv` ändert die Adresse der Umgebung, d.h. wurde mit `x_cr_gettaskparam` die Adresse der Umgebung ermittelt, ist diese Adresse nach `putenv` nicht mehr gültig.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich
≠0	Funktion nicht erfolgreich (z.B. wenn `putenv` nicht genügend Platz für die erweiterte Umgebung anlegen kann)

Siehe auch

getenv (Seite 279)

x_cr_gettaskparam (Seite 429)

x_cr_initenv (Seite 432)

3.115 puts

Funktion

String auf einen Stream schreiben

Syntax

```
#include <stdio.h>

int    puts (char *s);
```

Parameter

Parametername	Bedeutung
s	Zeiger auf einen String

Beschreibung

`puts` schreibt den mit `s` angegebenen String, der mit dem Null–Byte abschließt, auf `stdout` (Standard–Ausgabe–Stream), gefolgt von einem Zeilenwechsel. Das abschließende Null–Byte wird nicht übertragen.

Hinweis

Bei Streams im Textmodus wird ‘\n’ in ‘\r\n’ umgewandelt.

Rückgabewert

Rückgabewert	Bedeutung
Integer–Zahl	Anzahl der übertragenen Zeichen
EOF	Fehler oder Funktion nicht erfolgreich (z.B. wenn die Funktion auf eine Datei schreiben will, die nicht zum Schreiben geöffnet worden ist)

Siehe auch

fputs (Seite 263)

putc (Seite 333)

putw (Seite 337)

3.116 putw

Funktion

Wort auf einen Stream schreiben

Syntax

```
#include <stdio.h>

int    putw(
         int w,
         FILE *stream);
```

Parameter

Parametername	Bedeutung
w	Integer–Zahl
stream	Stream mit Struktur vom Typ `FILE`

Beschreibung

`putw` schreibt das mit `w` angegebene Wort (d.h. Integer–Zahl) auf den Ausgabe–Stream `stream`. Ein eventuell definierter Dateizeiger zeigt auf die Position, ab der geschrieben wird. Ein Wort hat die Größe einer 4 Byte langen Integer–Zahl. `putw` berücksichtigt keine Ausrichtung nach Wortadressen in der betreffenden Datei.

Hinweis

Bei Streams im Textmodus wird '\n' in '\r\n' umgewandelt.

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich
≠0	Fehler oder Funktion nicht erfolgreich

Siehe auch

fopen (Seite 258)
fputc (Seite 262)
printf (Seite 329)
putc (Seite 333)
putchar (Seite 334)
puts (Seite 336)

3.117 qsort

Funktion

Sortierfunktion

Syntax

```
#include <stdlib.h>

void    qsort(
          char *base,
          unsigned n,
          unsigned width,
          int (_FIXED _FAR *compar)
                        (const void *p0,
                         const void *p1));
```

Parameter

Parametername	Bedeutung
base	Zeiger auf die Basis der Tabelle
n	Anzahl der Elemente
width	Größe der Elemente in Bytes

compar	Name einer benutzerdefinierten Funktion, die zwei Datenelemente miteinander vergleicht. Die Funktion muss eine Integer–Zahl kleiner/ gleich/größer 0 zurückgeben, je nachdem ob das erste Argument (p0) kleiner, gleich oder größer als das zweite (p1) ist.
	Die Vergleichsfunktion muss eine Pascal (bzw. fixedparams) Schnittstelle und bei Verwendung von CAD-UL oder Intel C einen far Einsprung besitzen, z.B.:
	
	`int _FIXED _FAR compar (`
	`const void *p0,`
	`const void*p1)`

Beschreibung

`qsort` sortiert Datenelemente nach der mit `compar` vorgegebenen Reihenfolge.

Hinweis

`base` sollte vom Datentyp Pointer–to–Element sein und in den Typ Pointer–to–Character umgewandelt werden.

Die Elemente können neben den zu vergleichenden Werten auch andere Werte enthalten, da `compar` nicht jedes Byte vergleichen muss.

Siehe auch

bsearch (Seite 216)

memcmp (Seite 317)

strcmp (Seite 380)

strncmp (Seite 388)

3.118 raise

Funktion

Signal erzeugen

Syntax

```
#include <signal.h>

int    raise (int sig);
```

Parameter

Parametername	Bedeutung
sig	Signal

Beschreibung

`raise` erzeugt ein Signal, das gemäß der mit `signal` aufgesetzten Exception–Behandlung verarbeitet wird.

Mit `raise` lassen sich folgende Signale erzeugen:

SIGFPE Dieses Signal tritt bei folgenden Ausnahmen auf:

- Aufruf von `raise(SIGFPE)`
- Exception–Interrupt 0: Division durch 0
- Exception–Interrupt 4: Overflow
- Exception–Interrupt 7: Gerät nicht verfügbar
- Exception–Interrupt 16: durch NDP 80x87

SIGILL Dieses Signal tritt bei folgenden Ausnahmen auf:

- Aufruf von `raise (SIGILL)`
- Exception–Interrupt 6: Illegal Opcode

SIGSEGV Dieses Signal tritt bei folgenden Ausnahmen auf:

- Aufruf von `raise (SIGEGV)`
- Exception–Interrupt 5: Bound Violation
- Exception–Interrupt 9: Coprozessor
- Segmentation Violation (nur 80287)
- Exception–Interrupt 10: TSS–Fehler
- Exception–Interrupt 11: Segment nicht vorhanden
- Exception–Interrupt 13: 80286/80386 Pseudo Protection: General Protection Fault
- Exception–Interrupt 14: Page–Fehler
- Exception–Interrupt 17: Alignment–Check–Fehler

Die Behandlung von Exception–Interrupt 9 (Segmentation Violation durch den 80287) wird nicht durch CRUN unterstützt, wenn die Applikation auf einem PC/AT realisiert wird.

SIGABRT Dieses Signal tritt bei folgender Ausnahme auf:

- Aufruf von `raise (SIGABRT)`

SIGTERM Dieses Signal tritt bei folgender Ausnahme auf:

- Aufruf von `raise (SIGTERM)`

SIGUSR1 Dieses Signal tritt bei folgender Ausnahme auf:

- Aufruf von `raise (SIGUSR1)`

SIGUSR2 Dieses Signal tritt bei folgender Ausnahme auf:

- Aufruf von `raise(SIGUSR2)`

Hinweis

Vor der Ausführung eines Signals wird der entsprechende Tabelleneintrag für den Handler wieder auf SIG_DFL gesetzt.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich und Ausnahmebehandlung über `ret` beendet
≠0	Funktion nicht erfolgreich (z.B. bei einem undefinierten Wert von `sig`)

Siehe auch

signal (Seite 362)

3.119 rand

Funktion

Integer–Zufallszahl erzeugen

Syntax

```
#include <stdlib.h>

int    rand (void);
```

Beschreibung

`rand` liefert als Ergebnis eine Integer–Zufallszahl aus dem Definitionsbereich 0 bis 32767. Wird `rand` vor `srand` aufgerufen, wird die Folge durch `srand` automatisch neu initialisiert, wobei für das Argument `seed` der Wert 1 eingesetzt wird.

Rückgabewert

Rückgabewert	Bedeutung
Integer–Zahl	Zufallszahl

Siehe auch

srand (Seite 375)

3.120 read

Funktion

Daten aus einer Datei lesen

Syntax

```
#include <io.h>

size_t    read(
            int handle,
            void *buf,
            size_t nbytes);
```

Parameter

Parametername	Bedeutung
handle	Deskriptor
buf	Zeiger auf Puffer
nbytes	Anzahl zu übertragender Bytes

Beschreibung

`read` versucht, `nbytes` Bytes aus der mit `handle` verbundenen Datei in den Puffer `buf` zu lesen.`handle` ist ein Deskriptor, den `open` zurückgibt. Der Dateizeiger wird bei jedem Lesevorgang aktualisiert.

Hinweis

Bei einer im Textmodus geöffneten Datei bzw. Unit wird '\r\n' in '\n' umgewandelt.

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
Integer–Zahl	Anzahl der übertragenen Bytes (Funktion erfolgreich)
0	Dateiende erreicht
-1	Funktion nicht erfolgreich

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt:

Fehlercode	Bedeutung
EACCES	Ungültiges Zugriffsrecht
EAGAIN	Datei bereits geöffnet
EINVAL	Ungültiger Parameter
EIO	Physikalischer Schreib/Lesefehler
ENOENT	Die angegebene Datei oder eine Komponente von `fname` existiert nicht (Laufwerk wurde eventuell nicht angemeldet (`mount`)).
ENOMEM	Es konnte nicht ausreichend Speicher dem angegebenen Pool entnommen werden.
ERANGE	Interner RMOS3–Fehler

Siehe auch

close (Seite 227)

open (Seite 325)

write (Seite 426)

3.121 realloc

Funktion

Größe eines Speicherbereichs ändern

Syntax

```
#include <malloc.h>

void    *realloc(
             void *ptr,
             size_t size);
```

Parameter

Parametername	Bedeutung
ptr	Zeiger auf Speicher
size	neue Speichergröße

Beschreibung

`realloc` ändert die Größe des mit `ptr` angegebenen Speicherbereichs (der durch `malloc` oder `calloc` angefordert wurde) auf `size` Bytes und liefert einen Zeiger auf den (eventuell verschobenen) Speicherbereich. Der Speicherinhalt bleibt bis zu der Länge unverändert, die dem kleineren Wert der beiden Speichergrößen entspricht.

Hinweis

Der Speicherbereich wird entweder gezielt durch Aufruf der Funktion `free`, `RmFree` oder `RmFreeAll` freigegeben oder automatisch bei Beendigung der Task durch `exit`, `x_cr_killtsk`, `RmDeleteTask`, `RmEndTask` oder `RmKillTask`.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf Speicher der angeforderten Größe
NULL–Zeiger	Funktion nicht erfolgreich (z.B. wenn `size` gleich 0 ist). Der mit `ptr` angegebene Speicherbereich wurde durch den Funktionsaufruf nicht verändert

Fehlercodes

Im Fehlerfall wird `errno` auf folgenden Wert gesetzt:

Fehlercode	Bedeutung
ENOMEM	Kein freier Speicherplatz verfügbar

Siehe auch

RmFree (Seite 62)
RmFreeAll (Seite 63)
calloc (Seite 217)
free (Seite 265)
malloc (Seite 312)
xinitc (Seite 436)

3.122 remap

Funktion

Datenträger formatieren

Syntax

```
#include <io.h>

int    remap(
         const char *volume_name,
         int format);
```

Parameter

Parametername	Bedeutung
volume_name	Zeiger auf Laufwerkname
format	Aufzeichnungsformat des Laufwerks

Beschreibung

`remap` formatiert einen Datenträger neu. Der Datenträger muss beim HSFS abgemeldet sein (`dismount`).

`volume_name` muss ein gültiger Laufwerkname mit einer Länge von 1 oder 2 Zeichen sein (z.B. "A", "C", "P0"). `format` gibt das Aufzeichnungsformat des Geräts an und muss auf 0 gesetzt sein. Aus Kompatibilitätsgründen ist auch noch der Wert 2 zugelassen.

Hinweis

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich
–1	Funktion nicht erfolgreich

Fehlercodes

Im Fehlerfall wird `errno` auf folgenden Wert gesetzt:

Fehlercode	Bedeutung
EACCES	Laufwerk ist angemeldet oder formatiergeschützt
EINVAL	Unzulässiger Wert für `format`
EIO	Physikalischer Schreibfehler oder ungültiger Laufwerkname
ERANGE	Unerwarteter Fehlercode vom HSFS

Siehe auch

dismount (Seite 234)

mount (Seite 324)

3.123 remove

Funktion

Datei löschen

Syntax

```
#include <stdio.h>

int    remove (const char *pathname);
```

Parameter

Parametername	Bedeutung
pathname	Zeiger auf Dateispezifikation

Beschreibung

`remove` löscht die mit `pathname` angegebene Datei.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich
–1	Funktion nicht erfolgreich

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt:

Fehlercode	Bedeutung
EACCES	Das Löschen der NUL-Datei ist nicht erlaubt
EAGAIN	Datei ist noch geöffnet
EBADF	`pathname` ist ungültig bzw. die Dateiattribute lassen das Löschen der Datei nicht zu
EBUSY	Datei mit Länge 0 kann nicht gelöscht werden
ENOTDIR	Pfadname enthielt eine ungültige Komponente
ENOENT	Datei existiert nicht
EIO	Physikalischer Lese/Schreibfehler

Siehe auch

rename (Seite 347)

unlink (Seite 413)

3.124 rename

Funktion

Namen einer Datei ändern

Syntax

```
#include <stdio.h>

int    rename(
          const char *pathname,
          const char *newname);
```

Parameter

Parametername	Bedeutung
pathname	Zeiger auf alte Dateispezifikation
newname	Zeiger auf neue Dateispezifikation

Beschreibung

`rename` ändert den Namen einer Datei. `pathname` ist dabei der alte Dateiname und `newname` der neue Dateiname.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich
–1	Funktion nicht erfolgreich

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt:

Fehlercode	Bedeutung
EACCES	Angabe der NUL-Datei ist nicht erlaubt
EBADF	Umbenennen der Datei nicht möglich (z.B. `newname` existiert bereits)
EINVAL	Ungültiger Parameter
EIO	Physikalischer Schreib/Lesefehler
ENOSPC	Kein Speicherplatz auf Laufwerk oder Verzeichnis voll
EBUSY	Datei ist von einer anderen Task geöffnet
ENOENT	Angegebene Datei existiert nicht
ENOTDIR	Eine Komponente des Pfadnamen–Präfixes ist kein Verzeichnis (Laufwerk eventuell nicht angemeldet (`mount`))
EMFILE	Interner Pufferüberlauf

Siehe auch

remove (Seite 346)

unlink (Seite 413)

3.125 rewind

Funktion

Datenzeiger in einem Stream positionieren

Syntax

```
#include <stdio.h>

void    rewind (FILE *stream);
```

Parameter

Parametername	Bedeutung
stream	Stream mit Struktur vom Typ `FILE`

Beschreibung

`rewind` positioniert den Dateizeiger auf den Anfang des mit `stream` angegebenen Streams. `rewind(stream)` entspricht der Funktion `fseek(stream,(long)0,0)`, mit dem Unterschied, dass `rewind` keinen Rückgabewert liefert.

Siehe auch

fseek (Seite 272)

lseek (Seite 310)

3.126 rmdir

Funktion

Verzeichnis löschen

Syntax

```
#include <direct.h>

int    rmdir (char *dirname);
```

Parameter

Parametername	Bedeutung
dirname	Name des zu löschenden Verzeichnisses

Beschreibung

`rmdir` löscht das in `dirname` angegebene Verzeichnis.

Hinweis

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich abgearbeitet
-1	Fehler

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt:

Fehlercode	Bedeutung
EACCES	Ungültige Zugriffsrechte für `dirname`
EBUSY	`dirname` ist bereits mit lock option geöffnet
EINVAL	`dirname` ist kein Verzeichnis
EIO	physikalischer I/O Fehler
ENOENT	`dirname` ungültig
ENOSPC	Datenträger ist voll
ERANGE	Unerwarteter Fehlercode vom HSFS

Siehe auch

chdir (Seite 220)
getcwd (Seite 278)
mkdir (Seite 321)

3.127 scanf

Funktion

Formatierte Eingabe von `stdin`

Syntax

```
#include <stdio .h>

int    scanf (char *format, [arg_ptr] ...);
```

Parameter

Parametername	Bedeutung
format	Zeiger auf einen String
arg_ptr	Zeiger auf einen Zielbereich

Beschreibung

`scanf` liest Datenelemente von `stdin` (Standard–Eingabe–Kanal) und interpretiert sie gemäß den mit `format` angegebenen Format–Anweisungen. Anschließend hinterlegt die Funktion die Elemente an der mit `arg_ptr` angegebenen Stelle.

Der Format–String kann folgende Elemente enthalten:

1. Leerzeichen (Blank, TAB, LF, FF), bei denen die Eingabe bis zum nächsten Nicht–Leerzeichen gelesen wird, außer es folgt "[" oder "c".
2. Ein normales Zeichen (nicht %), das mit dem nächsten Zeichen der Eingabe–Datei übereinstimmen muss.
3. Konvertierungs–Angaben, bestehend aus dem Prozentzeichen "%" und einem darauffolgenden Konvertierungszeichen.

Wahlweise können folgen: die Angabe der maximalen Feldlänge oder das Zeichen "*", das angibt, dass das betreffende Element ignoriert werden soll. Damit kann ein Eingabe–Feld, das übersprungen werden soll, markiert werden. Ein Eingabe–Feld ist als String von Nicht–Leerzeichen definiert; es reicht bis zum nächsten ungültigen Zeichen oder bis zur maximalen Feldlänge, falls definiert.

Als Konvertierungszeichen sind die Zeichen c, d, e, f, i, u, o, s, g, p und x definiert.

c

Das zu konvertierende Datenelement ist ein einzelnes Zeichen. Der Zeiger `arg_ptr` zeigt auf ein Zeichen. Gültige Zeichen sind auch die Leerzeichen Blank, Tab und Carriage–Return. Das erste Zeichen, das ungleich einem dieser Leerzeichen ist, erhält man mit "%1s". Wenn eine Feldlänge angegeben wird, sollte das zugehörige Argument für ein Zeichen–Array gelten: die angegebene Anzahl von Zeichen wird gelesen.

s

Das zu konvertierende Datenelement ist ein String. Der Zeiger `arg_ptr` zeigt auf ein Zeichen–Array, das den erwarteten Eingabe–String und ein automatisch angehängtes Null–Byte aufnehmen soll.

d

Das zu konvertierende Datenelement ist eine Dezimalzahl. Der Zeiger `arg_ptr` zeigt auf eine Integer–Zahl.

o

Das zu konvertierende Datenelement ist eine Oktalzahl. Der Zeiger `arg_ptr` zeigt auf eine Integer–Zahl.

x

Das zu konvertierende Datenelement ist eine Hexadezimal–Zahl. Der Zeiger `arg_ptr` zeigt auf eine Integer–Zahl.

u

Das zu konvertierende Datenelement ist eine Dezimalzahl ohne Vorzeichen. Der Zeiger `arg_ptr` zeigt auf eine Integer–Zahl.

i

Das zu konvertierende Datenelement ist eine Integerzahl. Beginnt diese Zahl mit 0, so handelt es sich um eine Oktalzahl; beginnt sie mit 0x, so handelt es sich um eine Hexadezimal–Zahl; ansonsten wird eine Dezimalzahl eingelesen. `arg_ptr`zeigt auf eine Integer–Zahl.

p

Das zu konvertierende Datenelement ist ein Zeiger, der aus bis zu 8 hexadezimalen Ziffern besteht.

f

Das zu konvertierende Datenelement ist eine Gleitkomma–Zahl mit einfacher Genauigkeit und wird wie folgt dargestellt: ein optionales Minus–Zeichen, eine Folge von Ziffern, die einen Dezimalpunkt enthalten kann, und ein optionales Feld für den Exponenten.
[+|–]ddd [.[ddd]][E|e[+|–]ddd]
Der Zeiger *arg_ptr* zeigt auf die Gleitkomma–Zahl.

e

Wie f.

g

Wie f.

%

In der Eingabe wird an der betreffenden Stelle ein einzelnes Prozentzeichen "%" erwartet. Eine weitere Konvertierung findet nicht statt.

[

zeigt String–Elemente an. Das Überspringen der Leerzeichen wird unterdrückt. Nach der linken eckigen Klammer folgen ein Set von Zeichen, genannt Scan–Set und eine rechte eckige Klammer. Das Eingabe–Feld ist so lang, dass es die maximale Anzahl der Zeichen im Scan–Set aufnehmen kann. Das Zeichen ^, falls als erstes Zeichen im Scan–Set angegeben, dient als Zusatz–Operator. Es definiert den Scan–Set als Zeichensatz aller Zeichen, die nicht im Rest des Scan–Set–String enthalten sind.

Vor den Format–Zeichen d, o, x, u, f, g und e kann wahlweise auch l angegeben werden. Das jeweilige Argument wird dann nicht als Integer, sondern als Long–Decimal geschrieben, bzw. nicht als Gleitpunktzahl mit einfacher, sondern doppelter Genauigkeit. Vor den Format–Zeichen d, u, o und x kann wahlweise auch h angegeben werden. Damit ist definiert, dass in

der Argument–Liste kein Zeiger auf eine Integer–Variable, sondern auf eine Short–Variable steht.

Es sind einige Konventionen zur Struktur des Scan–Set zu beachten. Eine Folge von Zeichen kann durch die Schreibweise first–last angegeben werden, d.h. [0–9] kann für [0123456789] stehen. Dieser Konvention folgend, muss "first" kleiner/gleich "last" sein. Das Zeichen "–" bedeutet immer sich selbst, auch dann, wenn es das erste oder letzte Zeichen im Scan–Set ist. Um die rechte eckige Klammer als Element des Scan–Set zu kennzeichnen, muss sie (eventuell nach einem vorhergehenden Zeichen ^) am Anfang des Scan–Set stehen. In diesem Fall wird sie nicht als abschließende Klammer interpretiert. Das entsprechende Argument muss auf ein Zeichen–Array zeigen, das groß genug sein muss, um das Datenfeld und eine automatisch angehängte 0 aufzunehmen. Es muss zumindest ein passendes Zeichen angegeben sein, damit die Konvertierung erfolgreich ist.

Hinweis

Nachfolgende Leerzeichen (einschließlich Zeilenschaltung) werden nicht gelesen, außer sie sind auch im Control–String enthalten.

Bei Streams im Textmodus wird ‘\r\n’ in ‘\n’ umgewandelt.

Das Einlesen eines Datenelements wird beim ersten ungültigen Zeichen beendet. Beispielsweise wird beim Einlesen einer Dezimalzahl durch %d bei Eingabe von "123XY" der Wert 123 eingelesen, der nächste Aufruf arbeitet bei "X" weiter.

Tritt beim Einlesen einer Zahl ein Überlauf auf, d.h. der angegebene Wert ist größer als der für den zugehörigen Typ in LIMITS.H angegebene Grenzwert, wird der Grenzwert aus LIMITS.H an den Anwender zurückgeliefert und `errno` auf ERANGE gesetzt. Wird beispielsweise beim Einlesen einer Zahl vom Typ Unsigned–Short (%hd) ein Wert größer als USHRT_MAX (65535) angegeben, liefert scanf den Wert USHRT_MAX zurück.

Beim Einlesen von Oktal– oder Hexadezimal–Zahlen sind Vorzeichen nicht zulässig.

`printf` und `scanf` von 2 verschiedenen Tasks können auf einer Konsole nicht gleichzeitig abgearbeitet werden. Erst wenn der `printf`-Aufruf bzw. der `scanf`-Aufruf einer Task abgearbeitet ist, kann der `printf`- bzw. `scanf`-Aufruf der anderen Task abgearbeitet werden. Während eine Task im `scanf`-Aufruf auf Eingabe wartet, kann kein `printf`-Aufruf von einer anderen Task auf die gleiche Konsole ausgegeben werden.

Rückgabewert

Rückgabewert	Bedeutung
Anzahl	der erfolgreich verglichenen und zugewiesenen Formatelemente
0	Vergleich beginnt mit einem Konflikt zwischen Eingabe–Zeichen und Control–String
EOF	Eingabe vor dem ersten Konflikt oder Konvertierung beendet

Siehe auch

fscanf (Seite 270)
printf (Seite 329)
sscanf (Seite 376)
vfscanf (Seite 416)
vscanf (Seite 420)
vsscanf (Seite 424)
vsnprintf (Seite 421)

3.128 search

Funktion

Suche nach Dateien

Syntax

```
#include <io.h>

int    search (
          int *record,
          char *search_string,
          DIR_ENTRY *dir_entry);
```

Parameter

Parametername	Bedeutung
record	Adresse eines int, das den Directory-Record angibt, bei dem der Suchvorgang startet. Um den Suchvorgang vom Anfang des Verzeichnisses zu starten, muss `record` = 0 gesetzt werden. Nach erfolgreicher Suche wird dorthin die Record-Nummer des gefundenen Directory-Eintrages geschrieben. Um bei einer Suche mit Wildcards (*, ?) weitere Übereinstimmungen zu finden, muss record vor erneutem Aufruf von `search` um 1 erhöht werden.
search_string	relativer oder kompletter Pfad, dessen letzter Teil das Suchkriterium enthält. (z.B.: "C:/TEST/TEST.*")

dir_entry	Zeiger auf eine Struktur vom Typ DIR_ENTRY, die den gefundenen Direktory-Eintrag aufnimmt.
	
	`typedef struct _DIR_ENTRY`
	`{`
	`char file_name[8];`
	`char delim;`
	`char file_ext[3];`
	`unsigned short blocking;`
	`unsigned long file_size;`
	`unsigned short reserved1[3];`
	`char attrib;`
	`unsigned char reserved2;`
	`unsigned short time;`
	`unsigned short date;`
	`unsigned short reserved3;`
	`} DIR_ENTRY;`
	Die Einträge haben folgende Bedeutung:
	`file_name` Dateiname, linksbündig, mit Blanks aufgefüllt.
	`Delim` Trennzeichen zwischen Dateinamen und Dateierweiterung
	`file_ext` Dateierweiterung, linksbündig, mit Blanks aufgefüllt
	`blocking` Satzlänge (immer 1)
	`file_size` Dateilänge
	`attrib` Dateiattribute.

Parametername	Bedeutung
[dir_entry Fortsetzung]	Folgende Werte für `attrib` sind möglich: ATR_SYSTEM Systemdatei (ohne Bedeutung bei RMOS3) ATR_HIDDEN verborgen (ohne Bedeutung bei RMOS3) ATR_RDONLY schreibgeschützt ATR_DIREC Verzeichnis (darf nur bei Verzeichnissen angegeben werden) ATR_LABEL Datenträgerlabel ATR_ARCH Archivbit, wird bei jeder Modifikation der Datei gesetzt ATR_DEVICE Datei ist eine Gerätetreiberdatei Die Angabe mehrerer Attribute erfolgt durch ODER-Verknüpfung der einzelnen Attribute. `time` Uhrzeit der letzten Änderung an der Datei.[1)] `date` Datum der letzten Änderung an der Datei [2)] `reserved1`, `reserved2`, `reserved3` Die Werte dieser Einträge sind nicht definiert.

1) Uhrzeitwerte sind nach folgendem Schema abgelegt:

Bit 15															Bit 0
H	H	H	H	H	M	M	M	M	M	M	S	S	S	S	S

dabei ist:
H binäre Anzahl der Stunden (0..23)
M binäre Anzahl der Minuten (0..59) -
S binäre Anzahl der Sekunden geteilt durch 2 (0..29)

2) Datumswerte sind nach folgendem Schema abgelegt:

Bit 15															Bit 0
Y	Y	Y	Y	Y	Y	Y	M	M	M	M	D	D	D	D	D

dabei ist:
Y die binäre Anzahl der Jahre – 1980 (0..119,1980..2099)
M die binäre Anzahl der Monate (1..12)
D die binäre Anzahl der Tage (1..31)

Beschreibung

Mit dem Aufruf `search` ist es möglich, ein Verzeichnis nach einer bestimmten Datei oder einer Datei, deren Name einem bestimmten Suchmuster genügt, zu durchsuchen, ohne dass die Datei erst geöffnet werden muss. Da die Suche mit einer beliebigen Blocknummer des Verzeichnisses beginnen kann, lassen sich mit `search` auf einfache Weise Listen von Dateien aufbauen, deren Namen einem bestimmten Muster genügen. `search` wird somit häufig eingesetzt, um den Inhalt eines Verzeichnisses aufzulisten. Es werden auch Namen von Dateien zurückgemeldet, die als unsichtbar gekennzeichnet sind.

Hinweis

Beim HSFS wird der Funktion `search` eine Eintragsnummer übergeben, ab der sie mit der Suche beginnen soll. Ist `search` erfolgreich, gibt sie eine Eintragsnummer zurück, die dem tatsächlichen Index der gefundenen Datei im Verzeichnis entspricht. Diese Nummer kann wieder als Eingangsparameter für den nächsten `search`–Funktionsaufruf verwendet werden.

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich abgearbeitet
-1	Fehler

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt:

Fehlercode	Bedeutung
EAGAIN	Öffnen des Directories, in dem der Suchvorgang erfolgen soll, ist nicht möglich (Directory ist mit Lock-Option von anderer Task geöffnet).
EINVAL	Der Parameter `search_string` ist ungültig oder der Datenträger ist nicht angemeldet (`mount`).
EIO	Physikalischer I/O Fehler.
ENOMEM	Nicht ausreichend freier Speicherplatz im Heap.
ENOSPC	Öffnen des Directories, in dem der Suchvorgang erfolgen soll, ist nicht möglich (Directory ist voll).
ENOTDIR	Eine Komponente von `search_string` ist kein Directory.
ENOTFOUND	Es wurde keine Datei gefunden, die mit `search_string` übereinstimmt.
ERANGE	Unerwarteter Fehlercode vom HSFS

Siehe auch

searchlong (Seite 356)

3.129 searchlong

Funktion

Suche nach Dateien und Verzeichnissen mit langen Namen

Syntax

```
#include <io.h>

int    searchlong (
             int *record,
             char *search_string,
             DIR_ENTRY *dir_entry,
             char *longname);
```

Parameter

Parametername	Bedeutung
record	Adresse eines int, das den Directory-Record angibt, bei dem der Suchvorgang startet. Um den Suchvorgang vom Anfang des Verzeichnisses zu starten, muss `record` = 0 gesetzt werden. Nach erfolgreicher Suche wird dorthin die Record-Nummer des gefundenen Directory-Eintrages geschrieben. Um bei einer Suche mit Wildcards (*, ?) weitere Übereinstimmungen zu finden, muss `record` vor erneutem Aufruf von `searchlong` um 1 erhöht werden.
search_string	relativer oder kompletter Pfad, dessen letzter Teil das Suchkriterium enthält. (z.B.: "C:/TEST/TEST.*")

dir_entry	Zeiger auf eine Struktur vom Typ DIR_ENTRY, die den gefundenen Direktory-Eintrag aufnimmt.
	
	`typedef struct _DIR_ENTRY`
	`{`
	`char file_name[8];`
	`char delim;`
	`char file_ext[3];`
	`unsigned short blocking;`
	`unsigned long file_size;`
	`unsigned short reserved1[3];`
	`char attrib;`
	`unsigned char reserved2;`
	`unsigned short time;`
	`unsigned short date;`
	`unsigned short reserved3;`
	`} DIR_ENTRY;`
	`Die Einträge haben folgende` Bedeutung:
	`file_name` Dateiname, linksbündig, mit Blanks aufgefüllt.
	`delim` Trennzeichen zwischen Dateinamen und Dateierweiterung
	`file_ext` Dateierweiterung, linksbündig, mit Blanks aufgefüllt
	`blocking` Satzlänge (immer 1)
	`file_size` Dateilänge
	`attrib` Dateiattribute

Parametername	Bedeutung
	Folgende Werte für `attrib` sind möglich: ATR_SYSTEM Systemdatei (ohne Bedeutung bei RMOS3) ATR_HIDDEN verborgen (ohne Bedeutung bei RMOS3) ATR_RDONLY schreibgeschützt ATR_DIREC Verzeichnis (darf nur bei Verzeichnissen angegeben werden) ATR_LABEL Datenträgerlabel ATR_ARCH Archivbit, wird bei jeder Modifikation der Datei gesetzt ATR_DEVICE Datei ist eine Gerätetreiberdatei Die Angabe mehrerer Attribute erfolgt durch ODER-Verknüpfung der einzelnen Attribute. `time` Uhrzeit der letzten Änderung an der Datei.[1)] `date` Datum der letzten Änderung an der Datei [2)] `reserved1`, `reserved2`, `reserved3` Die Werte dieser Einträge sind nicht definiert.
longname	Zeiger auf einen Speicherbereich der als Ergebnis den langen Namen des gefundenen Eintrags enthält. Wenn der Dateiname kurz ist, wird der kurze Dateiname hier eingetragen. Der Anwender muss dafür sorgen, dass der Speicherbereich genügend gross ist (max. Länge ist 255).

1) Uhrzeitwerte sind nach folgendem Schema abgelegt:

Bit 15															Bit 0
H	H	H	H	H	M	M	M	M	M	M	S	S	S	S	S
dabei ist: H binäre Anzahl der Stunden (0..23) M binäre Anzahl der Minuten (0..59) S binäre Anzahl der Sekunden geteilt durch 2 (0..29)															

2) Datumswerte sind nach folgendem Schema abgelegt:

Bit 15															Bit 0
Y	Y	Y	Y	Y	Y	Y	M	M	M	M	D	D	D	D	D
dabei ist: Y die binäre Anzahl der Jahre – 1980 (0..119,1980..2099) M die binäre Anzahl der Monate (1..12) D die binäre Anzahl der Tage (1..31)															

Beschreibung

Mit dem Aufruf `searchlong` ist es möglich, ein Verzeichnis nach einer bestimmten Datei oder einer Datei, deren Name einem bestimmten Suchmuster genügt, zu durchsuchen, ohne dass die Datei erst geöffnet werden muss. Da die Suche mit einer beliebigen Blocknummer des Verzeichnisses beginnen kann, lassen sich mit `searchlong` auf einfache Weise Listen von Dateien aufbauen, deren Namen einem bestimmten Muster genügen. `searchlong` wird somit häufig eingesetzt, um den Inhalt eines Verzeichnisses aufzulisten. Es werden auch Namen von Dateien zurückgemeldet, die als unsichtbar gekennzeichnet sind.

Hinweis

Beim HSFS wird der Funktion `searchlong` eine Eintragsnummer übergeben, ab der sie mit der Suche beginnen soll. Ist `searchlong` erfolgreich, gibt sie eine Eintragsnummer zurück, die dem tatsächlichen Index der gefundenen Datei im Verzeichnis entspricht. Diese Nummer kann wieder als Eingangsparameter für den nächsten `searchlong`–Funktionsaufruf verwendet werden.

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich abgearbeitet
-1	Fehler

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt:

Fehlercode	Bedeutung
EAGAIN	Öffnen des Directories, in dem der Suchvorgang erfolgen soll, ist nicht möglich (Directory ist mit Lock-Option von anderer Task geöffnet).
EINVAL	Der Parameter `search_string` ist ungültig oder der Datenträger ist nicht angemeldet (`mount`).
EIO	Physikalischer I/O Fehler.
ENOMEM	Nicht ausreichend freier Speicherplatz im Heap.
ENOSPC	Öffnen des Directories, in dem der Suchvorgang erfolgen soll, ist nicht möglich (Directory ist voll).
ENOTDIR	Eine Komponente von `search_string` ist kein Directory.
ENOTFOUND	Es wurde keine Datei gefunden, die mit `search_string` übereinstimmt.
ERANGE	Unerwarteter Fehlercode vom HSFS

Siehe auch

search (Seite 353)

3.130 setbuf

Funktion

Puffer für einen Ein–/Ausgabe–Stream zuweisen

Syntax

```
#include <stdio.h>

void    setbuf(
           FILE *stream,
           char *buf);
```

Parameter

Parametername	Bedeutung
stream	Stream mit Struktur vom Typ `FILE`
buf	Zeiger auf einen Puffer

Beschreibung

`setbuf` weist einen Puffer für den mit `stream` angegebenen Ein–/Ausgabe–Stream zu. `setbuf` kann nach Öffnen eines Streams verwendet werden, aber nur bevor von dem Stream gelesen oder auf dem Stream geschrieben wird. Wenn `buf` kein NULL–Zeiger ist, wird der mit `buf` angegebene Bereich anstelle eines automatisch zugewiesenen Puffers verwendet. Ist `buf` ein NULL–Zeiger, ist die Ein–/Ausgabe auf dem angegebenen Stream ungepuffert.

Die Konstante BUFSIZ, die in der Headerdatei STDIO.H (bzw. STDIODEP.H) definiert ist, gibt für `setbuf` die Größe des benötigten Bereichs an: `char buf [BUFSIZ];`

Puffer werden beim ersten lesenden oder schreibenden Zugriff auf einen Stream zugewiesen. Ausgaben auf stdout (Standard–Ausgabe–Stream) sind standardmäßig zeilengepuffert, Eingaben von stdin (Standard–Eingabe–Stream) vollständig gepuffert und Ausgaben nach stderr (Standard–Fehler–Stream) ungepuffert. Alle anderen Ein– und Ausgabe–Streams sind vollständig gepuffert.

Siehe auch

fopen (Seite 258)

setvbuf (Seite 361)

3.131 setjmp

Funktion

Marke für nichtlokalen Sprung setzen

Syntax

```
#include <setjmp.h>

int    setjmp (jmp_buf env);
```

Parameter

Parametername	Bedeutung
env	Puffer für Sprungziel–Information

Beschreibung

`setjmp` setzt eine Marke für einen nichtlokalen Sprung. Ein nichtlokaler Sprung kann mit der Funktion `longjmp` ausgeführt werden. `env` enthält nach dem Aufruf von `setjmp` die notwendigen Angaben für `longjmp`. Nach einem Aufruf von `setjmp` enthält `env` beispielsweise die aktuellen Registerwerte, nicht aber eventuelle Variablen der Speicherklasse `auto`.

Rückgabewert

Rückgabewert	Bedeutung
0	`setjmp` wurde ausgeführt
≠ 0	Es wurde ein `longjmp` ausgeführt. Der Rückgabewert von `setjmp` ist der Wert des Parameters `retvalue` bei `longjmp`.

Siehe auch

longjmp (Seite 309)

3.132 setvbuf

Funktion

Puffer für einen Ein–/Ausgabe–Stream zuweisen

Syntax

```
#include <stdio.h>

int    setvbuf(
           FILE *stream,
           char *buf,
           int type,
           size_t size);
```

Parameter

Parametername	Bedeutung
stream	Stream mit Struktur vom Typ `FILE`
buf	Zeiger auf einen Puffer
type	Puffermethode
size	Größe des angeforderten Puffers

Beschreibung

`setvbuf` weist einen Puffer für den mit `stream` angegebenen Ein–/Ausgabe–Stream zu. `setvbuf` kann nach Öffnen eines Streams verwendet werden, aber nur bevor von dem Stream gelesen oder auf dem Stream geschrieben wird. Wenn `buf` kein NULL–Zeiger ist, wird der mit `buf` angegebene Bereich anstelle eines automatisch zugewiesenen Puffers verwendet. Ist `buf` ein NULL–Zeiger, ist die Ein–/Ausgabe auf dem angegebenen Stream ungepuffert.

Das Argument `type` legt fest, wie der Stream gepuffert werden soll. Die zulässigen Werte sind in der Headerdatei STDIO.H definiert:

_IOFBF vollständig gepufferte Ein–/Ausgabe

_IOLBF zeilengepufferte Ausgabe; der Puffer wird geleert, wenn eine neue Zeile geschrieben wird, der Puffer voll ist oder Input angefordert wird

_IONBF vollständig ungepufferte Ein–/Ausgabe

`size` gibt die Größe des benötigten Puffers an. Die Konstante BUFSIZ in der Headerdatei STDIO.H gilt als geeignete Puffergröße. Bei ungepufferter Ein–/Ausgabe werden `buf` und `size` nicht interpretiert.

Puffer werden beim ersten lesenden oder schreibenden Zugriff auf einen Stream zugewiesen. Ausgaben auf stdout (Standard–Ausgabe–Stream) sind standardmäßig zeilengepuffert, Eingaben von stdin (Standard–Eingabe–Stream) vollständig gepuffert und Ausgaben nach stderr (Standard–Fehler–Stream) ungepuffert. Alle anderen Ein– und Ausgabe–Streams sind vollständig gepuffert.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich
≠ 0	Unzulässiger Wert für `type` oder `size`

Siehe auch

fopen (Seite 258)
setbuf (Seite 359)

3.133 signal

Funktion

Funktion zur Exception-Behandlung installieren

Syntax

```
#include <signal.h>

void(_VAR _FAR
*_VAR _FAR signal(
             int sig,
             void (_VAR _FAR *func)(int,...)))(int,...);
```

Parameter

Parametername	Bedeutung
sig	Signal
func	Zeiger auf eine Funktion mit C-Schnittstelle (`cdecl` bzw. `varparams`). Bei Verwendung von CAD-UL oder Intel C muss die Funktion einen far Einsprung besitzen.

Beschreibung

Ausnahmezustände wie Division durch Null, Adressierungsfehler oder fehlerhafte Fließkommaoperationen lösen ein Signal aus. Signale werden entweder durch Exception–Interrupts oder durch entsprechende Programmierung (Funktion `raise`) erzeugt.

Wenn ein Signal auftritt, prüft CRUN, ob eine von ihr verwaltete Task aktiv ist und führt in diesem Fall die zugehörige Funktion aus, ansonsten wird der RMOS3–Exception–Handler angesprungen. Mit `signal` lassen sich die Kombination "Signal / auszuführende Funktion" verändern und anwendungsspezifische Funktionen zur Behandlung von Ausnahmen definieren. Dies geschieht durch einen entsprechenden taskspezifischen Tabelleneintrag nachdem `signal` aufgerufen wurde. CRUN besitzt drei vordefinierte Funktionen zur Behandlung von Signalen:

SIG_DFL Bei Auslösen der Ausnahme durch einen Exception–Interrupt wird der RMOS3–Exception–Handler angesprungen, ansonsten wird die Task beendet (`exit(1)`). Diese Funktion ist nach der Ausführung von `xinitt` allen Signalen dieser Task zugeordnet.

SIG_IGN Die Funktion ignoriert ein Signal. Das Programm wird an der unterbrochenen Stelle fortgesetzt. SIG_IGN ist bei Signalen, die durch Exception–Interrupts ausgelöst werden, nicht erlaubt.

SIG_ERR Die Funktion wird als Rückgabeparameter von signal benutzt, um einen unzulässigen Aufruf von signal anzuzeigen.

Folgende Signale werden in CRUN unterstützt:

SIGFPE Dieses Signal tritt bei folgenden Ausnahmen auf:

- Aufruf von `raise(SIGFPE)`
- Exception–Interrupt 0: Division durch 0
- Exception–Interrupt 4: Overflow
- Exception–Interrupt 7: Gerät nicht verfügbar
- Exception–Interrupt 16: durch NDP 80x87

SIGILL Dieses Signal tritt bei folgenden Ausnahmen auf:

- Aufruf von `raise (SIGILL)`
- Exception–Interrupt 6: Illegal Opcode

SIGSEGV Dieses Signal tritt bei folgenden Ausnahmen auf:

- Aufruf von `raise (SIGEGV)`
- Exception–Interrupt 5: Bound Violation
- Exception–Interrupt 9: Coprozessor Segmentation Violation (nur 80287)
- Exception–Interrupt 10: TSS–Fehler
- Exception–Interrupt 11: Segment nicht vorhanden
- Exception–Interrupt 13: 80286/80386 Pseudo Protection: General Protection Fault
- Exception–Interrupt 14: Page–Fehler
- Exception–Interrupt 17: Alignment–Check–Fehler

Die Behandlung von Exception–Interrupt 9 (Segmentation Violation durch den 80287) wird nicht durch CRUN unterstützt, wenn die Applikation auf einem PC/AT realisiert wird.

SIGABRT Dieses Signal tritt bei folgender Ausnahme auf:

- Aufruf von `raise (SIGABRT)`

SIGTERM Dieses Signal tritt bei folgender Ausnahme auf:

- Aufruf von `raise (SIGTERM)`

SIGUSR1 Dieses Signal tritt bei folgender Ausnahme auf:

- Aufruf von `raise (SIGUSR1)`

SIGUSR2 Dieses Signal tritt bei folgender Ausnahme auf:

- Aufruf von `raise(SIGUSR2)`

Mit `raise` können alle Signale erzeugt werden.

Hinweise zur Programmierung von Anwenderroutinen

Die Funktion zur eigenen Behandlung von Ausnahmen sollte folgende Schnittstelle besitzen:

```
void _VAR _FAR handler( int sig, int type)
```

Die Übergabe der Parameter erfolgt nach C-Notation: "varparams" (CAD-UL, Intel) bzw. "cdecl" (Borland). Es ist somit jederzeit möglich, den Parameter type zu ignorieren. Die Datei SIGNAL.H verlangt folgende Schnittstelle:

```
void _VAR _FAR handler( int sig, ...)
```

Falls die Funktion für die Ausnahmebehandlung in C programmiert wird, muss deshalb beim Aufruf von signal mit dieser Funktion entweder

- die erzeugte Warnung ignoriert werden
- ein entsprechender Cast gemacht werden

CRUN übergibt folgende Werte auf dem Stack vor Aufruf der Funktion zur Ausnahmebehandlung:

```
int sig
```

Dieser Parameter kann die Werte SIGFPE, SIGILL, SIGSEGV, SIGABRT, SIGTERM, SIGUSR1 und SIGUSR2 annehmen.

```
int type
```

Dieser Parameter gibt an, durch welche Ursache das Signal erzeugt wurde:

FPE_INTDIV0:	Signal durch Interrupt 0 ausgelöst
FPE_INTOVFLOW:	Signal durch Interrupt 4 ausgelöst
SEGV_BOUND:	Signal durch Interrupt 5 ausgelöst
ILL_EXECUTION:	Signal durch Interrupt 6 ausgelöst
FPE_DEVICENAVAIL:	Signal durch Interrupt 7 ausgelöst
SEGV_80287:	Signal durch Interrupt 9 ausgelöst
SEGV_SEGNPRESENT:	Signal durch Interrupt 11 ausgelöst
SEGV_PROTECTION:	Signal durch Interrupt 13 ausgelöst
SEGV_PAGE:	Signal durch Interrupt 14 ausgelöst
FPE_EXCEPTION:	Signal durch einen Interrupt des NDP 80x87 ausgelöst
SEGV_ALIGN:	Signal durch Interrupt 17 ausgelöst
SIG_EXPLICITGEN:	Signal durch Aufruf von raise ausgelöst.

Durch die optionale Auswertung des Parameters type ist eine korrekte Behandlung des Signals möglich. Folgende Stacklayouts sind nach Eintritt in die Handlerfunktion möglich (nur für Signale die durch Interrupts ausgelöst wurden):

Signale, die **nicht** durch einen NDP 80x87 ausgelöst wurden:

PSW	Vom Prozessor gesicherte Flags
CS EIP	An dieser Adresse wurde der Interrupt ausgelöst.
Error-Code	Error-Code der CPU (nur bei manchen Interrupts)
FS GS ES DS EBP EDI ESI EAX EBX ECX EDX	Gesicherte CPU–Register
9 x int	interne Daten, bei flat zusätzlich 9 x int
type	Integer–Parameter für Signal–Handler des Anwenders
sig	Integer–Parameter für Signal–Handler des Anwenders
CS EIP	Rückkehr–Adresse der internen CRUN–Routine

Signal, das vom NDP 80x87 ausgelöst wurde:

PSW	Vom Prozessor gesicherte Flags
CS EIP	An dieser Adresse wurde der Interrupt ausgelöst.
Error-Code	Error-Code der CPU (nur bei manchen Interrupts)
FS GS ES DS EBP EDI ESI EAX EBX ECX EDX	Gesicherte CPU–Register
14 x int	Mit dem Befehl FNSTENV gesicherter 80x87 Zustand
9 x int	interne Daten, bei flat zusätzlich 9 x int
type	Integer–Parameter für Signal–Handler des Anwenders
sig	Integer–Parameter für Signal–Handler des Anwenders
CS EIP	Rückkehr–Adresse zur internen CRUN–Routine

Bei der Programmierung einer eigenen Ausnahmefunktion sind die Schnittstellenkonventionen des Compilers einzuhalten (`cdecl` bzw. `varparams`, `far` bei CAD-UL oder Intel C). Die Ausführung der Ausnahmefunktion findet im A–Zustand von RMOS3 statt.

Rückgabewert

Rückgabewert	Bedeutung
SIG_ERR	Im Fehlerfall
Zeiger	Adresse des alten (jetzt ersetzten) Signal–Handlers

Hinweis

Vor der Ausführung eines Signals wird der entsprechende Tabelleneintrag für den Handler wieder auf SIG_DFL gesetzt.

Wird ein Signal durch einen Exception–Interrupt ausgelöst, darf der angesprungene Signal–Handler nicht mit `ret` beendet werden, d.h. es darf nicht an die Stelle zurückgesprungen werden, an der der Interrupt auftrat. Außerdem darf für Signale, die durch Exceptions ausgelöst werden, nicht SIG_IGN eingetragen werden. Es ist jedoch erlaubt, die Task, in der die Exception auftrat, durch `exit` zu beenden oder durch den Befehl `longjmp` an eine andere Stelle der Task zu springen.

Solange `signal` von keiner Task aufgerufen wurde, bleiben für Exception–Interrupts der CPU die von RMOS3 installierten Handler eingetragen.

Beispiel

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>

void _VAR _FAR handler( int sig, int type)
{ printf("Signal: %d, Reason: %d\n", sig, type);
  exit(1);
}
int gen_interrupt_0( int number)
{ return( 5 / number); /* Number is 0: A signal is created */
}
void taskx()
{ signal( SIGFPE, handler);
  /* handler handles now all SIGFPE's */
  gen_interrupt_0(0);
  /* Point of no return: because handler causes an exit(1) */
}
```

Siehe auch

raise (Seite 339)

3.134 sin

Funktion

Sinus

Syntax

```
#include <math.h>

double    sin (double x);
```

Parameter

Parametername	Bedeutung
x	Double–Zahl

Beschreibung

`sin` berechnet den Sinus aus einer Double–Zahl.

Bei `sin` wird das Argument in Bogenmaß angegeben. Bei einem Betrag des Arguments größer als 2^52 ruft sin die Fehlerbehandlungsfunktion `matherr` mit Exception–Typ TLOSS auf, die als Ergebnis IND zurückliefert. Bei dieser Größe des Arguments sind im numerischen Format darstellbare aufeinanderfolgende Double–Zahlen um 0,5 voneinander verschieden. Durch die Genauigkeit des Arguments wird also nur noch der Quadrant festgelegt. Wäre das Argument noch größer, so ginge auch diese Information verloren.

`errno` wird nicht gesetzt, da dies laut ANSI-Norm nicht erforderlich ist.

Rückgabewert

Rückgabewert	Bedeutung
Double–Zahl	Sinus
IND	Nach Aufruf von `matherr` mit Exception–Typ TLOSS (Betrag des Arguments ≥ 2^52)

Siehe auch

cos (Seite 228)

tan (Seite 402)

3.135 sinh

Funktion

Sinus hyperbolicus

Syntax

```
#include <math.h>

double    sinh (double x);
```

Parameter

Parametername	Bedeutung
x	Double–Zahl

Beschreibung

`sinh` berechnet den Sinus hyperbolicus (sinh(x) = (e^x – e^–x)/2) einer Double–Zahl.

Bei einem Überlauf ruft sinh die Fehlerbehandlungsfunktion `matherr` mit Exception–Typ OVERFLOW und setzt `errno` auf ERANGE. Der Rückgabewert ist in diesem Fall +/– HUGE_VAL (das Vorzeichen hängt vom Vorzeichen des Arguments ab).

Rückgabewert

Rückgabewert	Bedeutung
Double–Zahl	Sinus hyperbolicus
+/–HUGE_VAL	Nach Aufruf von `matherr` mit Exception–Typ DOMAIN wird `errno` auf ERANGE (Bereichsüberschreitung) gesetzt

Siehe auch

cosh (Seite 229)

tanh (Seite 403)

3.136 sleep

Funktion

Task für bestimmte Zeit anhalten

Syntax

```
#include <time.h>

unsigned    sleep (unsigned seconds);
```

Parameter

Parametername	Bedeutung
seconds	Anzahl Sekunden

Beschreibung

`sleep` hält die aufrufende Task für die mit `seconds` angegebene Anzahl von Sekunden an.

Ein vorzeitiges Wecken der Task kann durch die RMOS3-Funktion `RmResumeTask` erfolgen.

Hinweis

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
0	Task wurde nicht vorzeitig geweckt
Unsigned	Differenz zwischen angeforderter Zeit und tatsächlicher Wartezeit, wenn die aufrufende Task vorzeitig wieder geweckt wurde. Differenz = `seconds` – tatsächlich pausierte Zeit

Siehe auch

RmResumeTask (Seite 146)

3.137 snprintf

Funktion

Formatierte Ausgabe auf einen String

Syntax

```
#include <stdio.h>

int    snprintf(
         char *s,
         size_t size,
         const char *format [, arg]…);
```

Parameter

Parametername	Bedeutung
s	Zeiger auf einen String
size	Maximale Anzahl der Zeichen inklusive '\0' die geschrieben werden sollen, nicht größer als Länge von `s`
format	Zeiger auf einen String
arg	Argument–Liste

Beschreibung

`snprintf` verarbeitet den Eingabe–String `format`, interpretiert verschiedene Format–Kommandos und kopiert den mit einem Null–Byte abschließenden Ergebnis–String mit einer maximalen Länge von `size` Bytes in den mit `s` angegebenen Bereich. Der Anwender ist dafür verantwortlich, dass `s` mindestens die Größe `size` hat. `snprintf` gibt die Anzahl der übertragenen Zeichen zurück (Null–Byte '\0' nicht eingerechnet). Bei einem Ausgabe–Fehler wird ein Wert kleiner 0 zurückgegeben.

`format` ist ein String mit eingefügten Format–Anweisungen und dazugehörigen Argumenten: `snprintf (s, buflen, "...",arg1, arg2...).`

Siehe `printf` für Einzelheiten zum Eingabe–String, Escape–Sequenzen und Formatanweisungen.

Rückgabewert

Rückgabewert	Bedeutung
Anzahl	der ausgegebenen Zeichen (Funktion erfolgreich)
–1	Fehler (Funktion nicht erfolgreich)

Beispiel

Ausgabe von "hello world!" auf den String `array`:

```
main ()

{
      char array [20];
      snprintf (array, sizeof(array), "hello world!" );
}
```

Siehe auch

fprintf (Seite 260)
printf (Seite 329)
scanf (Seite 350)
sprintf (Seite 372)
vprintf (Seite 418)
vsnprintf (Seite 421)
vsprintf (Seite 422)

3.138 sprintf

Funktion

Formatierte Ausgabe auf einem String

Syntax

```
#include <stdio.h>

int    sprintf(
        char *s,
        char *format [,
        arg] ...);
```

Parameter

Parametername	Bedeutung
s	Zeiger auf einen String
format	Zeiger auf einen String
arg	Argument–Liste

Beschreibung

`sprintf` verarbeitet den Eingabe–String `format`, interpretiert verschiedene Format–Kommandos und kopiert den mit einem Null–Byte abschließenden Ergebnis–String in den mit `s` angegebenen Bereich. Der Anwender ist dafür verantwortlich, dass genügend Speicherplatz zur Aufnahme des Strings zur Verfügung steht. `sprintf` gibt die Anzahl der übertragenen Zeichen zurück (Null–Byte '\0' nicht eingerechnet). Bei einem Ausgabe–Fehler wird ein Wert kleiner 0 zurückgegeben.

`format` ist ein String mit eingefügten Format–Anweisungen und dazugehörigen Argumenten: `sprintf (s, "...",arg1, arg2...).`

Siehe `printf` für Einzelheiten zum Eingabe–String, Escape–Sequenzen und Formatanweisungen.

Rückgabewert

Rückgabewert	Bedeutung
Anzahl	der ausgegebenen Zeichen (Funktion erfolgreich)
–1	Fehler (Funktion nicht erfolgreich)

Beispiel

Ausgabe von "hello world!" auf den String `array`:

```
main ()

{
      char array [20];
      sprintf (array, "hello world!" );
}
```

Siehe auch

fprintf (Seite 260)
fscanf (Seite 270)
printf (Seite 329)
scanf (Seite 350)
snprintf (Seite 370)
sscanf (Seite 376)
vfprintf (Seite 414)
vprintf (Seite 418)
vsnprintf (Seite 421)
vsprintf (Seite 422)

3.139 sqrt

Funktion

Quadratwurzelfunktion

Syntax

```
#include <math.h>

double    sqrt (double x);
```

Parameter

Parametername	Bedeutung
x	Double–Zahl

Beschreibung

`sqrt` berechnet die Quadratwurzel aus einer Double–Zahl. Ist das Argument negativ, ruft `sqrt` die Fehlerbehandlungsfunktion `matherr` mit Exception–Typ DOMAIN auf und setzt `errno` auf EDOM. Als Rückgabewert wird in diesem Fall IND zurückgeliefert. Wird die Quadratwurzel aus –0 gezogen, wird der Wert –0 zurückgeliefert. –0 ist ein spezielles Zahlenformat im Coprozessor. Die Rückgabe von –0 für `sqrt(-0)` ist konform zur IEEE–754–Norm.

Rückgabewert

Rückgabewert	Bedeutung
Double–Zahl	Quadratwurzel
–0	wenn x = –0
IND	Nach Aufruf von `matherr` mit Exception–Typ DOMAIN wird `errno` auf EDOM gesetzt.

3.140 srand

Funktion

Initialisierungswert für `rand` setzen

Syntax

```
#include <stdlib.h>

void    srand (unsigned int seed);
```

Parameter

Parametername	Bedeutung
seed	Initialisierungswert

Beschreibung

`srand` verwendet das Argument `seed` als Initialisierungswert für die neue Folge von Pseudozufallszahlen, die bei aufeinanderfolgenden Funktionsaufrufen von `rand` zurückgegeben werden sollen. Ruft man später die Funktion `srand` mit demselben Initialisierungswert auf, wird dieselbe Folge von Pseudozufallszahlen erzeugt.

Siehe auch

rand (Seite 341)

3.141 sscanf

Funktion

Formatierte Eingabe von einem String

Syntax

```
#include <stdio.h>

int    sscanf(
        char *s,
        char *format,
        [arg_ptr] ...);
```

Parameter

Parametername	Bedeutung
s	Zeiger auf einen String
format	Zeiger auf einen String
arg_ptr	Zeiger auf einen Zielbereich

Beschreibung

`sscanf` liest Datenelemente vom dem mit `s` angegebenen String und interpretiert sie gemäß den mit format angegebenen Format–Anweisungen. Anschließend hinterlegt die Funktion die Elemente an der mit `arg_ptr` angegebenen Stelle. Siehe `scanf` für Einzelheiten zum Format–String und Konvertierungszeichen.

Hinweis

Nachfolgende Leerzeichen (einschließlich Zeilenschaltung) werden nicht gelesen, außer sie sind auch im Control–String enthalten.

Das Einlesen eines Datenelements wird beim ersten ungültigen Zeichen beendet. Beispielsweise wird beim Einlesen einer Dezimalzahl durch %d bei Eingabe von "123XY" der Wert 123 eingelesen, der nächste Aufruf arbeitet bei "X" weiter.

Tritt beim Einlesen einer Zahl ein Überlauf auf, d.h. der angegebene Wert ist größer als der für den zugehörigen Typ in LIMITS.H angegebene Grenzwert, wird der Grenzwert aus LIMITS.H an den Anwender zurückgeliefert und `errno` auf ERANGE gesetzt. Wird beispielsweise beim Einlesen einer Zahl vom Typ Unsigned–Short (%hd) ein Wert größer als USHRT_MAX (65535) angegeben, liefert `sscanf` den Wert USHRT_MAX zurück.

Beim Einlesen von Oktal– oder Hexadezimal–Zahlen sind Vorzeichen nicht zulässig.

Rückgabewert

Rückgabewert	Bedeutung
Anzahl	der erfolgreich verglichenen und zugewiesenen Formatelemente
0	Vergleich beginnt mit einem Konflikt zwischen Eingabe–Zeichen und Control–String
EOF	Eingabe vor dem ersten Konflikt oder Konvertierung beendet

Siehe auch

fprintf (Seite 260)
fscanf (Seite 270)
printf (Seite 329)
scanf (Seite 350)
sprintf (Seite 372)
vfprintf (Seite 414)
vfscanf (Seite 416)
vprintf (Seite 418)
vscanf (Seite 420)
vsprintf (Seite 422)
vsscanf (Seite 424)

3.142 strcat

Funktion

String an einen zweiten String anhängen

Syntax

```
#include <string.h>

char    *strcat(
            char *s1,
            const char *s2);
```

Parameter

Parametername	Bedeutung
s1	Zeiger auf ersten String
s2	Zeiger auf zweiten String

Beschreibung

`strcat` hängt String `s2` an String `s1` an. Die Funktion verändert den String `s1`, ohne jedoch auf Überlauf des für `s1` vorgesehenen Speichers zu prüfen. Daher muss der Speicher für `s1` groß genug definiert sein, da der neue String ansonsten andere Daten überschreibt.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf Ergebnis–String (`s1`)

Siehe auch

strchr (Seite 379)
strcmp (Seite 380)
strcpy (Seite 381)
strcspn (Seite 382)
strlen (Seite 386)
strncat (Seite 387)
strncmp (Seite 388)
strpbrk (Seite 390)
strrchr (Seite 391)
strspn (Seite 392)
strstr (Seite 393)
strtok (Seite 396)

3.143 strchr

Funktion

Zeichen in einem String suchen

Syntax

```
#include <string.h>

char    *strchr(
            const char *s,
            int c);
```

Parameter

Parametername	Bedeutung
s	Zeiger auf String
c	Vergleichszeichen

Beschreibung

`strchr` sucht das erste auftretende Zeichen `c` im String `s`. Die Funktion gibt einen Zeiger auf `c` zurück, oder einen NULL–Zeiger, falls `c` in `s` nicht enthalten ist.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf Zeichen
NULL–Zeiger	Zeichen nicht gefunden

Siehe auch

strcat (Seite 378)
strcmp (Seite 380)
strcpy (Seite 381)
strcspn (Seite 382)
strlen (Seite 386)
strncat (Seite 387)
strncmp (Seite 388)
strpbrk (Seite 390)
strrchr (Seite 391)
strspn (Seite 392)
strstr (Seite 393)
strtok (Seite 396)

3.144 strcmp

Funktion

Zwei Strings vergleichen

Syntax

```
#include <string.h>

int    strcmp(
          const char *s1,
          const char *s2);
```

Parameter

Parametername	Bedeutung
s1	Zeiger auf ersten String
s2	Zeiger auf zweiten String

Beschreibung

`strcmp` vergleicht zwei Strings. Der Rückgabewert ist 0, wenn die Strings identisch sind, positiv wenn `s1` lexikograpisch größer als `s2` ist, und negativ, wenn `s1` lexikograpisch kleiner als `s2` ist.

Rückgabewert

Rückgabewert	Bedeutung
Integer–Zahl	größer/gleich/kleiner 0, je nach Ergebnis

Siehe auch

strchr (Seite 379)

strcat (Seite 378)

strcpy (Seite 381)

strcspn (Seite 382)

strlen (Seite 386)

strncat (Seite 387)

strncmp (Seite 388)

strpbrk (Seite 390)

strrchr (Seite 391)

strspn (Seite 392)

strstr (Seite 393)

strtok (Seite 396)

3.145 strcpy

Funktion

String in einen zweiten String kopieren

Syntax

```
#include <string.h>

char    *strcpy(
            char *s1,
            const char *s2);
```

Parameter

Parametername	Bedeutung
s1	Zeiger auf ersten String
s2	Zeiger auf zweiten String

Beschreibung

`strcpy` kopiert String `s2` in String `s1` bis einschließlich Null–Byte.

`strcpy` verändert den String `s1`, ohne jedoch auf Überlauf des für `s1` vorgesehenen Speichers zu prüfen. Daher muss der Speicher für `s1` groß genug definiert sein, da der neue String ansonsten andere Daten überschreibt.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf Ergebnis–String (`s1`)

Siehe auch

strchr (Seite 379)

strcmp (Seite 380)

strcat (Seite 378)

strcspn (Seite 382)

strlen (Seite 386)

strncat (Seite 387)

strncmp (Seite 388)

strpbrk (Seite 390)

strrchr (Seite 391)

strspn (Seite 392)

strstr (Seite 393)

strtok (Seite 396)

3.146 strcspn

Funktion

Zwei Strings vergleichen

Syntax

```
#include <string.h>

size_t    strcspn(
             const char *s1,
             const char *s2);
```

Parameter

Parametername	Bedeutung
s1	Zeiger auf ersten String
s2	Zeiger auf zweiten String

Beschreibung

`strcspn` gibt die Anzahl Zeichen ab Anfang von String `s1` zurück, die nicht in String `s2` vorkommen.

Rückgabewert

Rückgabewert	Bedeutung
size_t	Anzahl der aufeinanderfolgenden Zeichen in `s1`, die nicht mit den Zeichen in `s2` übereinstimmen

Siehe auch

strchr (Seite 379)
strcmp (Seite 380)
strcpy (Seite 381)
strcat (Seite 378)
strlen (Seite 386)
strncat (Seite 387)
strncmp (Seite 388)
strpbrk (Seite 390)
strrchr (Seite 391)
strspn (Seite 392)
strstr (Seite 393)
strtok (Seite 396)

3.147 strerror

Funktion

Fehlertext ermitteln

Syntax

```
#include <string.h>

char    *strerror (int errnum);
```

Parameter

Parametername	Bedeutung
errnum	Fehlernummer

Beschreibung

`strerror` liefert einen Zeiger auf einen Fehlertext, passend zu `errnum`. Für `errnum` sind die Werte erlaubt, die von `errno` gemeldet werden können.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf Fehlertext

Siehe auch

errno, errno2 (Seite 240)

perror (Seite 327)

sys_errlist, sys_nerr (Seite 401)

3.148 strftime

Funktion

Datum und Uhrzeit formatiert ausgeben

Syntax

```
#include <time.h>

size_t    strftime(
             char *s,
             size_t maxsize,
             char *format,
             struct tm *timeptr);
```

Parameter

Parametername	Bedeutung
s	Array
maxsize	Anzahl Formate
format	Zeit– und Datumsformat
timeptr	Zeiger auf Struktur `tm`

Beschreibung

`strftime` schreibt Zeichen nach Array `s` nach den im String `format` angegeben Formaten. Der Formatstring besteht aus keinem, einem oder mehreren Konvertierungsangaben und anderen Zeichen. Eine Formatangabe beginnt mit dem Zeichen %, gefolgt von einem Zeichen mit der Formatierungsangabe. Alle anderen Zeichen werden unverändert nach Array `s` kopiert. Bis maximal `maxsize` Zeichen, einschließlich des abschließenden Zeichens '\0', werden kopiert. Die zulässigen Formate sind:

%a Abgekürzter Wochentag
%A Wochentag
%b Abgekürzter Monatsname
%B Monatsname
%c Datums– und Zeitausgabe
%d Tag des Monats (01–31)
%H Dezimalangabe der Stunde (00–23)
%I Dezimalangabe der Stunde (01–12)
%j Tag des Jahres (001–366)
%m Dezimalangabe des Monats (01–12)
%M Dezimalangabe der Minute (00–59)
%p Angabe von AM oder PM (Vor– oder Nachmittag)
%S Dezimalangabe der Sekunden (00–59)
%U Dezimalangabe der Kalenderwoche (00–53). Erster Wochentag ist der Sonntag.
%w Dezimalangabe des Wochentags (0 (Sonntag) – 6)
%W Dezimalangabe der Kalenderwoche (00–53). Erster Wochentag ist der Montag.
%x Ausgabe des Datums
%X Ausgabe der Uhrzeit
%y Dezimalausgabe des Jahres ohne die Jahrhunderte (00–99)
%Y Dezimalausgabe des Jahres mit Jahrhunderten
%Z Ausgabe der Zeitzone
%% Ausgabe des Zeichens '%

Die Zeit– und Datumsangaben werden in englischer Sprache ausgegeben.

Rückgabewert

Rückgabewert	Bedeutung
Integer–Zahl	Länge des erzeugten Strings (ohne '\0')
0	Inhalt von String `s` undefiniert

Siehe auch

asctime (Seite 206)

ctime (Seite 231)

gmtime (Seite 286)

localtime (Seite 305)

mktime (Seite 322)

time (Seite 404)

tzset (Seite 410)

3.149 strlen

Funktion

Länge eines Strings ermitteln

Syntax

```
#include <string.h>

size_t    strlen (const char *s);
```

Parameter

Parametername	Bedeutung
s	Zeiger auf String

Beschreibung

`strlen` gibt die Anzahl der Zeichen im String `s` zurück, das abschließende Null–Byte nicht eingerechnet.

Rückgabewert

Rückgabewert	Bedeutung
Integer–Zahl	String–Länge

Siehe auch

strchr (Seite 379)
strcmp (Seite 380)
strcpy (Seite 381)
strcspn (Seite 382)
strcat (Seite 378)
strncat (Seite 387)
strncmp (Seite 388)
strpbrk (Seite 390)
strrchr (Seite 391)
strspn (Seite 392)
strstr (Seite 393)
strtok (Seite 396)

3.150 strncat

Funktion

Teil–String an einen zweiten String hängen

Syntax

```
#include <string.h>

char    *strncat(
             char *s1,
             const char *s2,
             size_t n);
```

Parameter

Parametername	Bedeutung
s1	Zeiger auf ersten String
s2	Zeiger auf zweiten String
n	Anzahl Zeichen

Beschreibung

`strncat` hängt maximal `n` Zeichen von String `s2` an String `s1` an. Die Funktion verändert den String `s1`, ohne jedoch auf Überlauf des für `s1` vorgesehenen Speichers zu prüfen. Daher muss der Speicher für `s1` groß genug definiert sein, da der neue String ansonsten andere Daten überschreibt.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf Ergebnis–String (`s1`)

Siehe auch

strchr (Seite 379)

strcmp (Seite 380)

strcpy (Seite 381)

strcspn (Seite 382)

strlen (Seite 386)

strcat (Seite 378)

strncmp (Seite 388)

strpbrk (Seite 390)

strrchr (Seite 391)

strspn (Seite 392)

strstr (Seite 393)

strtok (Seite 396)

3.151 strncmp

Funktion

Zwei Teil–Strings vergleichen

Syntax

```
#include <string.h>

int    strncmp(
           const char *s1,
           const char *s2,
           size_t n);
```

Parameter

Parametername	Bedeutung
s1	Zeiger auf ersten String
s2	Zeiger auf zweiten String
n	Anzahl Zeichen

Beschreibung

`strncmp` vergleicht zwei Strings bis maximal `n` Zeichen. Der Rückgabewert ist 0, wenn die Strings identisch sind, positiv wenn `s1` lexikograpisch größer als `s2` ist, und negativ, wenn `s1` lexikograpisch kleiner als `s2` ist.

Rückgabewert

Rückgabewert	Bedeutung
Integer–Zahl	größer/gleich/kleiner 0, je nach Ergebnis

Siehe auch

strchr (Seite 379)

strcmp (Seite 380)

strcpy (Seite 381)

strcspn (Seite 382)

strlen (Seite 386)

strncat (Seite 387)

strcat (Seite 378)

strpbrk (Seite 390)

strrchr (Seite 391)

strspn (Seite 392)

strstr (Seite 393)

strtok (Seite 396)

3.152 strncpy

Funktion

Teil–String kopieren

Syntax

```
#include <string.h>

int    strncpy(
           char *s1,
           const char *s2,
           size_t n);
```

Parameter

Parametername	Bedeutung
s1	Zeiger auf Ziel–String
s2	Zeiger auf Quell–String
n	Anzahl Zeichen

Beschreibung

`strncpy` kopiert maximal `n` Zeichen von String `s2` in String `s1`. Ist `s2` kürzer als `n` Zeichen, so wird `s1` mit Null–Bytes aufgefüllt.

strncpy verändert den String `s1`, ohne jedoch auf Überlauf des für `s1` vorgesehenen Speichers zu prüfen. Daher muss der Speicher für `s1` groß genug definiert sein, da der neue String ansonsten andere Daten überschreibt.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf den Anfang des Ziel–Strings

Siehe auch

strcat (Seite 378)
strchr (Seite 379)
strcmp (Seite 380)
strcpy (Seite 381)
strcspn (Seite 382)
strlen (Seite 386)
strncat (Seite 387)
strpbrk (Seite 390)
strrchr (Seite 391)
strspn (Seite 392)
strstr (Seite 393)
strtok (Seite 396)

3.153 strpbrk

Funktion

Zeichen in zwei Strings vergleichen

Syntax

```
#include <string.h>

char    *strpbrk(
             const char *s1,
             const char *s2);
```

Parameter

Parametername	Bedeutung
s1	Zeiger auf ersten String
s2	Zeiger auf zweiten String

Beschreibung

`strpbrk` ermittelt im String `s1` das erste Auftreten eines Zeichens aus String `s2`. Die Funktion gibt einen Zeiger auf das gefundene Zeichen (in `s1`) zurück, oder einen NULL–Zeiger, falls kein Zeichen aus `s2` in `s1` enthalten ist.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf Zeichen (in `s1`)
NULL–Zeiger	Zeichen nicht gefunden

Siehe auch

strchr (Seite 379)

strcmp (Seite 380)

strcpy (Seite 381)

strcspn (Seite 382)

strlen (Seite 386)

strncat (Seite 387)

strncmp (Seite 388)

strcat (Seite 378)

strrchr (Seite 391)

strspn (Seite 392)

strstr (Seite 393)

strtok (Seite 396)

3.154 strrchr

Funktion

Zeichen in einem String suchen

Syntax

```
#include <string.h>

char    *strrchr(
            const char *s,
            int c);
```

Parameter

Parametername	Bedeutung
s	Zeiger auf String
c	Vergleichszeichen

Beschreibung

`strrchr` sucht das letzte auftretende Zeichen `c` im String `s`. Die Funktion gibt einen Zeiger auf `c` zurück, oder einen NULL–Zeiger, falls `c` in `s` nicht enthalten ist. Das Begrenzungszeichen (Null–Byte) wird als Teil des Strings interpretiert.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf Zeichen
NULL–Zeiger	Zeichen nicht gefunden

Siehe auch

strchr (Seite 379)
strcmp (Seite 380)
strcpy (Seite 381)
strcspn (Seite 382)
strlen (Seite 386)
strncat (Seite 387)
strpbrk (Seite 390)
strcat (Seite 378)
strspn (Seite 392)
strstr (Seite 393)
strtok (Seite 396)

3.155 strspn

Funktion

Zwei Strings vergleichen

Syntax

```
#include <string.h>

size_t    strspn(
             const char *s1,
             const char *s2);
```

Parameter

Parametername	Bedeutung
s1	Zeiger auf ersten String
s2	Zeiger auf zweiten String

Beschreibung

`strspn` gibt die Anzahl Zeichen ab Anfang von String `s1` zurück, die in String `s2` vorkommen.

Beispiel: Wenn `s1` gleich "abcabcdefgdef" und `s2` gleich "cabbage" ist, gibt die Funktion das Ergebnis 6 zurück, da im zweiten String kein "d" enthalten ist.

Rückgabewert

Rückgabewert	Bedeutung
Integer–Zahl	Anzahl der aufeinanderfolgenden Zeichen in `s1`, die mit den Zeichen in `s2` übereinstimmen

Siehe auch

strchr (Seite 379)
strcmp (Seite 380)
strcpy (Seite 381)
strcspn (Seite 382)
strlen (Seite 386)
strncat (Seite 387)
strncmp (Seite 388)
strpbrk (Seite 390)
strrchr (Seite 391)
strcat (Seite 378)
strstr (Seite 393)
strtok (Seite 396)

3.156 strstr

Funktion

Zwei Strings vergleichen

Syntax

```
#include <string.h>

char    *strstr(
            const char *s1,
            const char *s2);
```

Parameter

Parametername	Bedeutung
s1	Zeiger auf ersten String
s2	Zeiger auf zweiten String

Beschreibung

`strstr` gibt die erste Übereinstimmung von String `s2` mit String `s1` an. Die Funktion gibt einen Zeiger auf die erste Übereinstimmung (in `s1`) zurück, ansonsten einen NULL–Zeiger.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf Zeichen (in `s1`)
NULL–Zeiger	Zeichen nicht gefunden

Siehe auch

strchr (Seite 379)
strcmp (Seite 380)
strcpy (Seite 381)
strcspn (Seite 382)
strlen (Seite 386)
strncat (Seite 387)
strncmp (Seite 388)
strpbrk (Seite 390)
strrchr (Seite 391)
strspn (Seite 392)
strcat (Seite 378)
strtok (Seite 396)

3.157 strtod

Funktion

String in eine Double–Gleitkomma–Zahl umwandeln

Syntax

```
#include <stdlib.h>

double    strtod(
             const char *string,
             char **ptr);
```

Parameter

Parametername	Bedeutung
string	Zeiger auf ein String
ptr	Zeiger

Beschreibung

`strtod` gibt als Ergebnis eine Double–Gleitkomma–Zahl zurück. Das Ergebnis entspricht dem Wert des Strings, auf den `string` zeigt. Der String wird abgesucht, solange gültige Werte gefunden werden. Führende Leerzeichen (definiert durch `isspace` in CTYPE.H) werden ignoriert. Die Gleitkomma–Zahl muss folgendes Format besitzen (in der angegebenen Reihenfolge):

- führende optionale Leerzeichen und / oder Tabulatoren
- ein optionales Vorzeichen
- eine Folge von Dezimalziffern, danach einen Dezimalpunkt und weitere Ziffern (zumindest einer der beiden Ziffernblöcke und der Dezimalpunkt müssen vorhanden sein).
- ein optionaler Exponent, der mit ‘e’ oder ‘E’ eingeleitet wird und aus einem optionalen Vorzeichen sowie einem obligatorischen Integerstring besteht.

Wenn der Wert von `ptr` ungleich `((char**)0)` ist, enthält `*ptr` einen Zeiger auf das Zeichen, bei dem der Scan–Vorgang abgebrochen wurde. Falls keine Double–Gleitkomma–Zahl erzeugt wird, wird `*ptr` mit `string` initialisiert und 0.0 zurückgegeben.

Rückgabewert

Rückgabewert	Bedeutung
Double–Zahl	Double–Gleitkomma–Zahl; bei `ptr` ungleich `((char**)0)` wird mit `ptr` ein Zeiger auf das Zeichen zurückgegeben, das zum Abbruch des Scan–Vorgangs führte
0.0	Keine Double–Zahl erzeugt; `*ptr` wird mit `string` initialisiert (falls `ptr` ungleich NULL ist)

Fehlercodes

Im Fehlerfall wird `errno` auf folgenden Wert gesetzt:

Fehlercode	Bedeutung
ERANGE	Bei Überlauf oder Unterlauf. Bei einem Unterlauf gibt `strtod` den Wert 0.0 zurück. Bei einem Überlauf gibt `strtod` den Wert HUGE_VAL mit entspr. Vorzeichen zurück.

Siehe auch

scanf (Seite 350)

3.158 strtok

Funktion

Zeichen ungleich Begrenzungszeichen in einem String suchen

Syntax

```
#include <string.h>

char    *strtok(
            char *s1,
            const char *s2);
```

Parameter

Parametername	Bedeutung
s1	Zeiger auf einen String
s2	Zeiger auf einen String mit Begrenzungszeichen

Beschreibung

`strtok` gibt einen Zeiger auf das erste Zeichen in String `s1` zurück, das kein Begrenzungszeichen ist. Die Funktion benötigt als Eingabe den aus Zeichen bestehenden String `s1` und den aus Begrenzungszeichen bestehenden String `s2`. Das nachfolgende Begrenzungszeichen wird in das Null–Byte '\0' umgewandelt. Die Funktion verwendet einen statischen Zeiger, der auf die jeweils aktuelle Position im String zeigt. Wird bei einem neuen Funktionsaufruf nicht `s1`, sondern NULL übergeben, so setzt `strtok` die Analyse des Ursprungs–Strings fort. In `s2` können bei jedem Aufruf andere Begrenzungszeichen stehen. Wenn kein Zeichen mehr in `s1` steht, wird ein NULL–Zeiger zurückgegeben.

Hinweis

Da gefundene Begrenzerzeichen in das Null–Byte '\0' umgewandelt werden, darf sich der String `s1` nicht im CODE–Segment befinden.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf das erste Zeichen des spezifizierten Elements (in `s1`)
0	Keine Elemente mehr vorhanden

Siehe auch

strchr (Seite 379)
strcmp (Seite 380)
strcpy (Seite 381)
strcspn (Seite 382)
strlen (Seite 386)
strncat (Seite 387)
strncmp (Seite 388)
strpbrk (Seite 390)
strrchr (Seite 391)
strspn (Seite 392)
strstr (Seite 393)
strcat (Seite 378)

3.159 strtol

Funktion

String in eine Integer–Zahl umwandeln

Syntax

```
#include <stdlib.h>

long    strtol(
           char *string,
           char **ptr,
           int base);
```

Parameter

Parametername	Bedeutung
string	Zeiger auf einen String
ptr	Zeiger
base	Zahlenbasis

Beschreibung

`strtol` gibt als Ergebnis einen Long–Integer zurück. Das Ergebnis entspricht dem Wert des Strings, auf den `string` zeigt. Der String wird abgesucht, bis ein Zeichen nicht mehr konsistent zur Basis `base` ist. Führende Leerzeichen (definiert durch `isspace` in CTYPE.H) werden ignoriert.

Ist der Wert von `ptr` ungleich `((char**)0)`, enthält `*ptr` einen Zeiger auf das Zeichen, bei dem der Scan–Vorgang abgebrochen wurde. Falls keine Integer–Zahl erzeugt wird, wird `*ptr` mit `string` initialisiert und 0 zurückgegeben.

Ist `base` positiv und nicht größer als 36, dient es als Basis der Umwandlung. Das Ergebnis hat wahlweise ein Vorzeichen; führende Nullen werden unterdrückt. "0x" bzw. "0X" wird ignoriert, falls `base` gleich 16 ist.

Wenn `base` gleich 0 ist, bestimmt der String selbst die Basis: nach einem optionalen Vorzeichen bedeutet eine führende Null oktale Umwandlung; führende Zeichen "0x" bzw. "0X" bedeuten hexadezimale Umwandlung. Sonst wird eine dezimale Umwandlung vorgenommen.

Hinweis

Überlauf–Bedingungen werden ignoriert.

Rückgabewert

Rückgabewert	Bedeutung
Long–Integer	Integer–Zahl; bei `ptr` ungleich `((char**)0)` wird mit `ptr` ein Zeiger auf das Zeichen zurückgegeben, das zum Abbruch des Scan–Vorgangs führte
0	Keine Integer–Zahl erzeugt; `*ptr` wird mit `string` initialisiert (falls `ptr` ungleich NULL ist)

Siehe auch

atoi (Seite 214)
atol (Seite 215)
isspace (Seite 299)
strtoul (Seite 399)

3.160 strtoul

Funktion

String in Unsigned–Long umwandeln

Syntax

```
#include <stdlib.h>

unsigned long    strtoul(
                    const char *str,
                    char **endptr,
                    int radix);
```

Parameter

Parametername	Bedeutung
str	Zeiger auf einen String
endptr	Zeiger
radix	Zahlenbasis

Beschreibung

`strtoul` gibt als Ergebnis einen vorzeichenlosen Long–Integer zurück. Das Ergebnis entspricht dem Wert des Strings, auf den `str` zeigt. Der String wird abgesucht, bis ein Zeichen nicht mehr konsistent zur Basis `radix` ist. Führende Leerzeichen (definiert durch `isspace` in CTYPE.H) werden ignoriert.

Wenn der Wert von `endptr` ungleich `((char**)0)` ist, enthält `*endptr` einen Zeiger auf das Zeichen, bei dem der Scan–Vorgang abgebrochen wurde. Falls keine Integer–Zahl erzeugt wird, wird `*endptr` mit `str` initialisiert und 0 zurückgegeben.

Ist `radix` positiv und nicht größer als 36, dient es als Basis der Umwandlung. Das Ergebnis hat wahlweise ein Vorzeichen; führende Nullen werden unterdrückt. "0x" bzw. "0X" wird ignoriert, falls `radix` gleich 16 ist.

Wenn `radix` gleich 0 ist, bestimmt der String selbst die Basis: nach einem optionalen Vorzeichen bedeutet eine führende Null oktale Umwandlung; führende Zeichen "0x" bzw. "0X" bedeuten hexadezimale Umwandlung. Sonst wird eine dezimale Umwandlung vorgenommen.

Hinweis

Überlauf–Bedingungen werden ignoriert.

Rückgabewert

Rückgabewert	Bedeutung
Unsigned–Long	Unsigned–Long–Integer; bei `endptr` ungleich `((char**)0)` wird mit `endptr` ein Zeiger auf das Zeichen zurückgegeben, das zum Abbruch des Scan–Vorgangs führte
0	Keine Integer–Zahl erzeugt; `*endptr` wird mit `str` initialisiert (falls `endptr` ungleich NULL ist)

Siehe auch

atoi (Seite 214)
atol (Seite 215)
isspace (Seite 299)
strtol (Seite 397)

3.161 sys_errlist, sys_nerr

Funktion

String–Array mit Fehlermeldungen

Syntax

```
#include <errno.h>

extern char    *sys_errlist[ ];
extern int     sys_nerr;
```

Beschreibung

`sys_errlist` ist ein String–Array mit Fehlermeldungen. Es dient dazu, konsistente Meldungen auszugeben. In dem Array kann die Variable `errno` als Index verwendet werden, damit die Meldung ohne Zeilenschaltung erscheint. Die maximale Anzahl von Fehlermeldungen im Array ist in der externen Variablen `sys_nerr` definiert.

Siehe auch

strerror (Seite 383)

errno, errno2 (Seite 240)

perror (Seite 327)

3.162 tan

Funktion

Tangens

Syntax

```
#include <math.h>

double    tan (double x);
```

Parameter

Parametername	Bedeutung
x	Double–Zahl

Beschreibung

`tan` berechnet den Tangens aus einer Double–Zahl.

Bei `tan` wird das Argument in Bogenmaß angegeben. Bei einem Betrag des Arguments größer als 2^52 ruft `tan` die Fehlerbehandlungsfunktion `matherr` mit Exception–Typ TLOSS auf, die als Ergebnis IND zurückliefert. Bei dieser Größe des Arguments sind im numerischen Format darstellbare aufeinanderfolgende Double–Zahlen um 0,5 voneinander verschieden. Durch die Genauigkeit des Arguments wird also nur noch der Quadrant festgelegt. Wäre das Argument noch größer, so ginge auch diese Information verloren.

`errno` wird nicht gesetzt, da dies laut ANSI-Norm nicht erforderlich ist.

Rückgabewert

Rückgabewert	Bedeutung
Double–Zahl	Tangens
IND	Nach Aufruf von matherr mit Exception–Typ TLOSS (Betrag des Arguments ≥ 2^52)

Siehe auch

cos (Seite 228)

sin (Seite 367)

3.163 tanh

Funktion

Tangens hyperbolicus

Syntax

```
#include <math.h>

double    tanh (double x);
```

Parameter

Parametername	Bedeutung
x	Double–Zahl

Beschreibung

`tanh` berechnet den Tangens hyperbolicus (tanh(x) = (e^x–e^–x)/(e^x+e^–x)) einer Double–Zahl. Der Ergebnisbereich liegt zwischen –1 und +1.

Rückgabewert

Rückgabewert	Bedeutung
Double–Zahl	Tangens hyperbolicus

Siehe auch

cosh (Seite 229)

sinh (Seite 368)

3.164 time

Funktion

Systemzeit ermitteln, seit 01.01.1970, 00:00:00 GMT

Syntax

```
#include <time.h>

long    time (long *tptr);
```

Parameter

Parametername	Bedeutung
tptr	Zeiger auf einen Puffer

Beschreibung

`time` liefert die Systemzeit als Anzahl der Sekunden, die seit dem 1. Januar 1970, GMT verstrichen sind. Wenn der Zeiger `tptr` kein NULL–Zeiger ist, wird der Rückgabewert in den mit `tptr` angegebenen Bereich geschrieben.

Rückgabewert

Rückgabewert	Bedeutung
Long–Integer	Anzahl der Sekunden seit 1. Januar 1970, 00:00:00 GMT

Hinweis

Verwenden Sie `_time32`, wenn Sie Zeitangaben über den 19. Januar 2038 hinaus benötigen.

Siehe auch

strftime (Seite 384)

_time32 (Seite 201)

asctime (Seite 206)

ctime (Seite 231)

gmtime (Seite 286)

localtime (Seite 305)

mktime (Seite 322)

tzset (Seite 410)

3.165 tmpfile

Funktion

Temporäre Datei anlegen

Syntax

```
#include <stdio.h>

FILE    *tmpfile (void);
```

Beschreibung

`tmpfile` legt eine temporäre Datei an. Die Funktion gibt einen Zeiger auf die Struktur vom Typ `FILE` für die Datei zurück. Eine temporäre Datei wird gelöscht, wenn sich die Task beendet, die sie geöffnet hat, oder wenn die Datei geschlossen wird. Die temporäre Datei wird zur Aktualisierung mit ('w+') geöffnet (siehe `fopen`).

`tmpfile` erzeugt die Datei in dem Verzeichnis, das beim Aufruf von `xinitt` bzw. `xinitc` durch den Parameter `tmppath` angegeben wurde.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf Stream–Struktur
NULL–Zeiger	Fehler oder Funktion nicht erfolgreich

Siehe auch

fopen (Seite 258)

tmpnam (Seite 406)

unlink (Seite 413)

xinitc (Seite 436)

xinitt (Seite 438)

3.166 tmpnam

Funktion

Namen für eine temporäre Datei erzeugen

Syntax

```
#include <stdio.h>

char    *tmpnam(char *s);
```

Parameter

Parametername	Bedeutung
s	= 0 `tmpnam` legt sein Ergebnis in einem internen statischen Datenbereich ab und gibt einen Zeiger auf diesen Bereich zurück. Dessen Inhalt wird von einem erneuten Funktionsaufruf überschrieben. ≠ 0 Hier wird angenommen, dass `s` einen Speicherbereich angibt. Dessen Größe in Bytes legt die in STDIO.H definierte Konstante L_tmpnam fest. `tmpnam` legt sein Ergebnis in diesem Speicherbereich ab und gibt `s` zurück.

Beschreibung

`tmpnam` erzeugt einen Dateinamen für eine temporäre Datei. Der Dateiname wird immer mit dem Pfad–Präfix erzeugt, der beim Aufruf von `xinitt` bzw. `xinitc` durch den Parameter `tmppath` angegeben wurde.

Ruft eine einzige Task mehr als TMP_MAX–mal (siehe Datei STDIO.H) die Funktionen `tmpfile` oder `tmpnam` auf, liefern diese Namen, die schon einmal erzeugt worden sind.

Während der Zeit zwischen dem Erzeugen des Dateinamens und dem Öffnen der dazugehörigen Datei, kann eine zweite Task eine Datei mit dem gleichen Namen erzeugen. Dies wird vermieden, wenn die zweite Task die beschriebenen Funktionen aufruft und geeignete Dateinamen vereinbart werden.

Hinweis

Mit jedem Funktionsaufruf von `tmpnam` wird ein anderer Dateiname erzeugt. Dateien, die mit `tmpnam` und `fopen` angelegt wurden, sind nur in dem Sinn temporär, dass sie in einem Verzeichnis für zeitweisen Zugriff stehen und ihre Namen eindeutig sind. Nicht mehr benötigte Dateien müssen mit `unlink` oder `remove` gelöscht werden.

Rückgabewert

Rückgabewert	Bedeutung
Zeiger	auf internen statischen Bereich (bei `s` = 0)
Zeiger	auf `s` (bei `s` ≠ 0)

Siehe auch

fopen (Seite 258)
tmpfile (Seite 405)
unlink (Seite 413)
xinitc (Seite 436)
xinitt (Seite 438)

3.167 toascii

Funktion

Zeichen in eine Integer–Zahl umwandeln

Syntax

```
#include <ctype.h>

int    toascii (int c);
```

Parameter

Parametername	Bedeutung
c	Zeichen

Beschreibung

`toascii` verarbeitet als Argument das Zeichen `c` und liefert als Ergebnis eine Integer–Zahl (ASCII–Code). Nicht–ASCII–Bits werden ausgeblendet. Im Fehlerfall ist der Rückgabewert nicht definiert.

Rückgabewert

Rückgabewert	Bedeutung
Integer–Zahl	ASCII–Code (Funktion erfolgreich)

Siehe auch

isascii (Seite 292)

tolower, _tolower (Seite 408)

toupper, _toupper (Seite 409)

3.168 tolower, _tolower

Funktion

Zeichen in einen Kleinbuchstaben umwandeln

Syntax

```
#include <ctype.h>

int    tolower (int c);
int    _tolower (int c);
```

Parameter

Parameter	Bedeutung
c	Zeichen

Beschreibung

`_tolower` und `tolower` verarbeiten als Argument das Zeichen `c` und liefern als Ergebnis einen Kleinbuchstaben (ASCII–Code). Im Fehlerfall ist der Rückgabewert nicht definiert.

Das Makro `_tolower` erfordert als Argument einen Buchstaben in Großschreibung. Wird diese Beschränkungen mißachtet, ist das Ergebnis nicht definiert.

Das Makro `tolower` ist nicht so schnell wie `_tolower`, unterliegt aber keinen Beschränkungen.

Rückgabewert

Rückgabewert	Bedeutung
Integer–Zahl	ASCII–Code (Funktion erfolgreich)

Siehe auch

toascii (Seite 407)

islower (Seite 296)

isupper (Seite 300)

toupper, _toupper (Seite 409)

3.169 toupper, _toupper

Funktion

Zeichen in einen Großbuchstaben umwandeln

Syntax

```
#include <ctype.h>

int    toupper (int c);
int    _toupper (int c);
```

Parameter

Parametername	Bedeutung
c	Zeichen

Beschreibung

`toupper` und `_toupper` verarbeiten als Argument das Zeichen `c` und liefern als Ergebnis einen Großbuchstaben (ASCII–Code). Im Fehlerfall ist der Rückgabewert nicht definiert.

Das Makro `_toupper` erfordert als Argument einen Buchstaben in Kleinschreibung. Wird diese Beschränkungen mißachtet, ist das Ergebnis nicht definiert.

Das Makro `toupper` ist nicht so schnell wie `_toupper`, unterliegt aber keinen Beschränkungen.

Rückgabewert

Rückgabewert	Bedeutung
Integer–Zahl	ASCII–Code (Funktion erfolgreich)

Siehe auch

toascii (Seite 407)

tolower, _tolower (Seite 408)

islower (Seite 296)

isupper (Seite 300)

3.170 tzset

Funktion

Zeitzonenanpassung durchführen

Syntax

```
#include <time.h>

void    tzset (void);
```

Beschreibung

`tzset` berechnet anhand der Umgebungs–Variablen TZ die Zeitzonen–Anpassung und legt die Ergebnisse in den globalen Variablen `timezone`, `daylight` und `tzname` ab.

Name	Bedeutung	
timezone	enthält die Differenz zwischen Greenwicher Zeit und der lokalen Zeitzone in Sekunden.	
daylight	ist ungleich 0, falls eine Umrechnung auf Sommerzeit erforderlich ist.	
TZ	TZ hat das Format nnn[+/-]d[d][mmm]	
	nnn	ist der Name der Zeitzone; er ist ohne Bedeutung, muss aber angegeben werden
	[+/-]d[d]	ist die Differenz der lokalen Zeit zu GMT. Wahlweise mit oder ohne Vorzeichen, einstellig oder zweistellig.
	mmm	ist der Name der Sommerzeit. Der Name ist ohne Bedeutung. Die Angabe des Namens bewirkt, dass die Sommerzeitanpassung aktiv ist. (Sommerzeit reicht vom letzten Sonntag im März bis zum letzten Sonntag im Oktober) Sofern gesetzt, ändert TZ somit die externen Variablen `timezone` und `daylight`. Außerdem setzt `tzset` von `TZ` die Zeitzonen `tzname`.
tzname	Array, in dem die Namen der Zeitzonen enthalten sind `char *tzname[2];`	

`TZ` wird durch `xinitt` auf `TZ=MEZ-1MSZ` (Mitteleuropäische Uhrzeit mit Sommerzeit) initialisiert.

`tzset` wird von `asctime` aufgerufen, kann aber auch explizit vom Benutzer aufgerufen werden.

Hinweis

Die Rückgabewerte zeigen auf einen Datenbereich, dessen Inhalt beim nächsten Aufruf überschrieben wird.

Beispiel

Das folgende Beispiel zeigt den Einsatz der verschiedenen CRUN–Zeitfunktionen:

```
#include <time.h>

main ()
{
struct tm *ptm;
char *date;
long tadd;

putenv ("TZ=MEZ-1");

time (&tadd);
date = ctime (&tadd);
printf ("today’s date is %s", date)□

ptm = localtime (&tadd);
date = asctime (ptm);
printf ("today’s date is %s", date);

ptm = gmtime (&tadd);
date = asctime (ptm);
printf ("Greenwich mean time is %s", date);
}
```

Siehe auch

strftime (Seite 384)

asctime (Seite 206)

ctime (Seite 231)

gmtime (Seite 286)

localtime (Seite 305)

mktime (Seite 322)

time (Seite 404)

xinitt (Seite 438)

3.171 ungetc

Funktion

Zeichen in einen Stream zurückschreiben

Syntax

```
#include <stdio.h>

int    ungetc(
         char c,
         FILE *stream);
```

Parameter

Parametername	Bedeutung
c	Zeichen
stream	Stream mit Struktur vom Typ `FILE`

Beschreibung

`ungetc` schreibt das mit `c` angegebene Zeichen in den Puffer zum mit `stream` angegebenen Stream zurück. Das Zeichen wird mit dem nächsten Funktionsaufruf von `getc` vom Puffer auf diesen Stream zurückgegeben. Der Rückgabewert ist `c`, der Status von `stream` bleibt unverändert.

Wenn `c` gleich EOF ist, bleibt der Puffer unverändert und der Rückgabewert ist gleich 0.

Es wird immer mindestens ein Zeichen zurückgeschrieben, vorausgesetzt der Stream ist gepuffert und es wurde bereits von ihm gelesen.

Rückgabewert

Rückgabewert	Bedeutung
Integer–Zahl	Übertragenes Zeichen
0	Übertragenes Zeichen ist EOF
EOF	Fehler

Siehe auch

fseek (Seite 272)

getc (Seite 276)

setbuf (Seite 359)

3.172 unlink

Funktion

Datei löschen

Syntax

```
#include <io.h>

int    unlink( const char *pathname);
```

Parameter

Parametername	Bedeutung
pathname	Zeiger auf Dateispezifikation

Beschreibung

`unlink` löscht die mit `pathname` angegebene Datei.

Hinweis

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich
–1	Funktion nicht erfolgreich

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt:

Fehlercode	Bedeutung
EACCES	Das Löschen der NUL-Datei ist nicht erlaubt
EAGAIN	Datei ist noch geöffnet
EBADF	`pathname` ist ungültig bzw. die Dateiattribute lassen das Löschen der Datei nicht zu
EIO	Physikalischer Lese/Schreibfehler
ENOENT	Datei existiert nicht
ENOTDIR	Pfadname enthielt eine ungültige Komponente

Siehe auch

remove (Seite 346)

rename (Seite 347)

3.173 vfprintf

Funktion

Formatierte Ausgabe auf einen Stream

Syntax

```
#include <stdio.h>
#include <stdarg.h>

int    vfprintf(
         FILE *stream,
         char *format,
         va_list ap);
```

Parameter

Parametername	Bedeutung
siehe fprintf (Seite 260)	

Beschreibung

`vfprintf` entspricht der Funktion `fprintf`. Im Gegensatz zu dieser wird sie aber nicht mit einer variablen Anzahl von Argumenten aufgerufen, sondern mit Hilfe einer Argumente–Liste, die in der Header–Datei STDARG.H definiert ist.

STDARG.H definiert den Typ `va_list` und einen Satz von Makros zum Absuchen einer Argument–Liste, die aus verschiedenen Datentypen bestehen kann.

Das Argument `ap` aus der Familie `vprint` der Bibliotheks–Routinen hat ebenfalls den Datentyp `va_list`, es wird zusammen mit den Makros `va_start`, `va_arg` und `va_end` aus der Header–Datei verwendet.

`va_start(ap,paramn)`

Das Makro `va_start(ap, paramn)` - `ap` ist vom Typ `va_list` - muss aufgerufen werden, bevor auf die Argument–Liste zugegriffen und sie durchsucht werden kann. `paramn` ist der letzte konstante Parameter, der übergeben wird.

`va_arg(ap,atype)`

Das Makro `va_arg(ap, atype)` ermittelt in der Liste `ap` das nächste Argument. Das Argument `atype` gibt den Datentyp an, der für das zurückgegebene Argument erwartet wird.

`va_end(ap)`

Das Makro `va_end(ap)` muss ausgeführt werden, wenn alle gewünschten Argumente gefunden wurden. Die Argument–Liste in `ap` kann erneut abgewandert werden, wenn nach `va_end` wieder `va_start` aufgerufen wird.

Rückgabewert

Rückgabewert	Bedeutung
siehe fprintf (Seite 260)	

Hinweis

Bei Streams im Textmodus wird ‘\n’ in ‘\r\n’ umgewandelt.

Beispiel

Das Beispiel zeigt, wie sich mit `vfprintf` ein Fehlerbehandlungs–Programm erstellen läßt. `va_start` wird ausgeführt, um die Parameterliste zu initialisieren. Die restlichen Argumente zu `error`, d.h. `arg1` usw., werden der Funktion `vfprintf` im Argument `ap` übergeben.

```
#include <stdio.h>
#include <stdarg.h>

/*
* error sollte aufgerufen werden mit
* error (function_name, format, arg1, arg2,...);
*
*/
void error (char *function_name, char *format, ...)
{
     va_list ap;

     va_start (ap,format);

/* Namen der Funktion ausgeben, die den Fehler verursachte */
    fprintf (stderr,"ERROR in %s:", function_name);

/* Rest der Meldung ausgeben */
     vfprintf (stderr, format, ap);
     va_end (ap);
     abort ();
}
```

Siehe auch

printf (Seite 329)

sprintf (Seite 372)

vprintf (Seite 418)

vsnprintf (Seite 421)

vsprintf (Seite 422)

vsscanf (Seite 424)

3.174 vfscanf

Funktion

Formatierte Eingabe von einem String

Syntax

```
#include <stdio.h>

int    vfscanf(
         FILE *stream,
         char *format,
         va_list ap);
```

Parameter

Parametername	Bedeutung
stream	Stream mit Struktur vom Typ `FILE`
fmtstr	Zeiger auf einen String
ap	Argumentenliste

Beschreibung

`vfscanf` liest Datenelemente vom dem mit `stream` angegebenen Stream und interpretiert sie gemäß den mit `format` angegebenen Format-Anweisungen und hinterlegt sie in `ap`.

Siehe `scanf` für Einzelheiten zum Format–String und Konvertierungszeichen.

Hinweis

Nachfolgende Leerzeichen (einschließlich Zeilenschaltung) werden nicht gelesen, außer sie sind auch im Control–String enthalten.

Das Einlesen eines Datenelements wird beim ersten ungültigen Zeichen beendet. Beispielsweise wird beim Einlesen einer Dezimalzahl durch %d bei Eingabe von "123XY" der Wert 123 eingelesen, der nächste Aufruf arbeitet bei "X" weiter.

Tritt beim Einlesen einer Zahl ein Überlauf auf, d.h. der angegebene Wert ist größer als der für den zugehörigen Typ in LIMITS.H angegebene Grenzwert, wird der Grenzwert aus LIMITS.H an den Anwender zurückgeliefert und `errno` auf ERANGE gesetzt. Wird beispielsweise beim Einlesen einer Zahl vom Typ Unsigned–Short (%hd) ein Wert größer als USHRT_MAX (65535) angegeben, liefert `vfscanf` den Wert USHRT_MAX zurück.

Beim Einlesen von Oktal– oder Hexadezimal–Zahlen sind Vorzeichen nicht zulässig.

Rückgabewert

Rückgabewert	Bedeutung
Anzahl	der erfolgreich verglichenen und zugewiesenen Formatelemente
0	Vergleich beginnt mit einem Konflikt zwischen Eingabe–Zeichen und Control–String
EOF	Eingabe vor dem ersten Konflikt oder Konvertierung beendet

Siehe auch

fscanf (Seite 270)

printf (Seite 329)

scanf (Seite 350)

sscanf (Seite 376)

vscanf (Seite 420)

vsscanf (Seite 424)

3.175 vprintf

Funktion

Formatierte Ausgabe auf den Standard-Ausgabe-Kanal

Syntax

```
#include <stdio.h>
#include <stdarg.h>

int    vprintf(
          char *format,
          va_list ap);
```

Parameter

Parametername	Bedeutung
siehe printf (Seite 329)	

Beschreibung

`vprintf` entspricht der Funktion `printf`. Im Gegensatz zu dieser wird sie aber nicht mit einer variablen Anzahl von Argumenten aufgerufen, sondern mit Hilfe einer Argumente–Liste, die in der Header–Datei STDARG.H definiert ist.

STDARG.H definiert den Typ `va_list` und einen Satz von Makros zum Absuchen einer Argument–Liste, die aus verschiedenen Datentypen bestehen kann.

Das Argument `ap` aus der Familie `vprint` der Bibliotheks–Routinen hat ebenfalls den Datentyp `va_list`, es wird zusammen mit den Makros `va_start`, `va_arg` und `va_end` aus der Header–Datei verwendet.

`va_start(ap,paramn)`

Das Makro `va_start(ap, paramn)` - `ap` ist vom Typ `va_list` - muss aufgerufen werden, bevor auf die Argument–Liste zugegriffen und sie durchsucht werden kann. `paramn` ist der letzte konstante Parameter, der übergeben wird.

`va_arg(ap,atype)`

Das Makro `va_arg(ap, atype)` ermittelt in der Liste `ap` das nächste Argument. Das Argument `atype` gibt den Datentyp an, der für das zurückgegebene Argument erwartet wird.

`va_end(ap)`

Das Makro `va_end(ap)` muss ausgeführt werden, wenn alle gewünschten Argumente gefunden wurden. Die Argument–Liste in `ap` kann erneut abgewandert werden, wenn nach `va_end` wieder `va_start` aufgerufen wird.

Hinweis

Bei Streams im Textmodus wird ‘\n’ in ‘\r\n’ umgewandelt.

Rückgabewert

Rückgabewert	Bedeutung
siehe printf (Seite 329)	

Beispiel

Siehe vfprintf (Seite 414)

Siehe auch

vfprintf (Seite 414)

fprintf (Seite 260)

snprintf (Seite 370)

sprintf (Seite 372)

vsnprintf (Seite 421)

vsprintf (Seite 422)

3.176 vscanf

Funktion

Formatierte Eingabe von einem String

Syntax

```
#include <stdio.h>

int    vscanf(
         const char *format,
         va_list ap);
```

Parameter

Parametername	Bedeutung
format	Zeiger auf einen String
ap	Argumentenliste

Beschreibung

`vscanf` liest Datenelemente von `stdin` (Standard–Eingabe–Kanal) und interpretiert sie gemäß den mit `format` angegebenen Format–Anweisungen und hinterlegt sie in `ap`.

Siehe `scanf` für Einzelheiten zum Format–String und Konvertierungszeichen.

Hinweis

Nachfolgende Leerzeichen (einschließlich Zeilenschaltung) werden nicht gelesen, außer sie sind auch im Control–String enthalten.

Das Einlesen eines Datenelements wird beim ersten ungültigen Zeichen beendet. Beispielsweise wird beim Einlesen einer Dezimalzahl durch %d bei Eingabe von "123XY" der Wert 123 eingelesen, der nächste Aufruf arbeitet bei "X" weiter.

Tritt beim Einlesen einer Zahl ein Überlauf auf, d.h. der angegebene Wert ist größer als der für den zugehörigen Typ in LIMITS.H angegebene Grenzwert, wird der Grenzwert aus LIMITS.H an den Anwender zurückgeliefert und `errno` auf ERANGE gesetzt. Wird beispielsweise beim Einlesen einer Zahl vom Typ Unsigned–Short (%hd) ein Wert größer als USHRT_MAX (65535) angegeben, liefert `vscanf` den Wert USHRT_MAX zurück.

Beim Einlesen von Oktal– oder Hexadezimal–Zahlen sind Vorzeichen nicht zulässig.

Rückgabewert

Rückgabewert	Bedeutung
Anzahl	der erfolgreich verglichenen und zugewiesenen Formatelemente
0	Vergleich beginnt mit einem Konflikt zwischen Eingabe–Zeichen und Control–String
EOF	Eingabe vor dem ersten Konflikt oder Konvertierung beendet

Siehe auch

vfscanf (Seite 416)
fscanf (Seite 270)
printf (Seite 329)
scanf (Seite 350)
sscanf (Seite 376)
vsscanf (Seite 424)

3.177 vsnprintf

Funktion

Formatierte Ausgabe auf einem String

Syntax

```
#include <stdio.h>

int    vsnprintf(
          char *s,
          size_t size,
          const char *format,
          va_list ap);
```

Parameter

Parametername	Bedeutung
s	Zeiger auf einen String
size	Maximale Anzahl der Zeichen inklusive '\0' die geschrieben werden sollen, nicht größer als Länge von `s`
format	Zeiger auf einen String
ap	Argument–Liste

Beschreibung

`vsnprintf` entspricht der Funktion `snprintf`. Im Gegensatz zu dieser wird sie aber nicht mit einer variablen Anzahl von Argumenten aufgerufen, sondern mit Hilfe einer Argumente–Liste, die in der Header–Datei STDARG.H definiert ist.

STDARG.H definiert den Typ `va_list` und einen Satz von Makros zum Absuchen einer Argument–Liste, die aus verschiedenen Datentypen bestehen kann.

Das Argument `ap` aus der Familie `vprint` der Bibliotheks–Routinen hat ebenfalls den Datentyp `va_list`, es wird zusammen mit den Makros `va_start`, `va_arg` und `va_end` aus der Header–Datei verwendet.

Rückgabewert

Rückgabewert	Bedeutung
Anzahl	der ausgegebenen Zeichen (Funktion erfolgreich)
–1	Fehler (Funktion nicht erfolgreich)

Siehe auch

vfprintf (Seite 414)

vprintf (Seite 418)

fprintf (Seite 260)

printf (Seite 329)

scanf (Seite 350)

snprintf (Seite 370)

sprintf (Seite 372)

vsprintf (Seite 422)

3.178 vsprintf

Funktion

Formatierte Ausgabe auf einen String

Syntax

```
#include <stdio.h>
#include <stdarg.h>

int    vsprintf(
         char *s,
         char *format,
         va_list ap);
```

Parameter

Parametername	Bedeutung
siehe sprintf (Seite 372)	

Beschreibung

`vsprintf` entspricht der Funktion `sprintf`. Im Gegensatz zu dieser wird sie aber nicht mit einer variablen Anzahl von Argumenten aufgerufen, sondern mit Hilfe einer Argumente–Liste, die in der Header–Datei STDARG.H definiert ist.

STDARG.H definiert den Typ `va_list` und einen Satz von Makros zum Absuchen einer Argument–Liste, die aus verschiedenen Datentypen bestehen kann.

Das Argument `ap` aus der Familie `vprint` der Bibliotheks–Routinen hat ebenfalls den Datentyp `va_list`, es wird zusammen mit den Makros `va_start`, `va_arg` und `va_end` aus der Header–Datei verwendet.

`va_start(ap,paramn)`

Das Makro `va_start(ap, paramn)` - `ap` ist vom Typ `va_list` - muss aufgerufen werden, bevor auf die Argument–Liste zugegriffen und sie durchsucht werden kann. `paramn` ist der letzte konstante Parameter, der übergeben wird.

`va_arg(ap,atype)`

Das Makro `va_arg(ap, atype)` ermittelt in der Liste `ap` das nächste Argument. Das Argument `atype` gibt den Datentyp an, der für das zurückgegebene Argument erwartet wird.

`va_end(ap)`

Das Makro `va_end(ap)` muss ausgeführt werden, wenn alle gewünschten Argumente gefunden wurden. Die Argument–Liste in `ap` kann erneut abgewandert werden, wenn nach `va_end` wieder `va_start` aufgerufen wird.

Rückgabewert

Rückgabewert	Bedeutung
siehe sprintf (Seite 372)	

Beispiel

Siehe vfprintf (Seite 414)

Siehe auch

vfprintf (Seite 414)

vprintf (Seite 418)

vsnprintf (Seite 421)

fprintf (Seite 260)

printf (Seite 329)

snprintf (Seite 370)

3.179 vsscanf

Funktion

Formatierte Eingabe von einem String

Syntax

```
#include <stdio.h>

int    vsscanf(
         const char *s,
         char *format,
         va_list ap);
```

Parameter

Parametername	Bedeutung
s	Zeiger auf einen String
format	Zeiger auf einen String
ap	Argumentenliste

Beschreibung

`vsscanf` liest Datenelemente vom dem mit `s` angegebenen String und interpretiert sie gemäß den mit `format` angegebenen Format–Anweisungen und hinterlegt sie in `ap`.

Siehe `scanf` für Einzelheiten zum Format–String und Konvertierungszeichen.

Hinweis

Nachfolgende Leerzeichen (einschließlich Zeilenschaltung) werden nicht gelesen, außer sie sind auch im Control–String enthalten.

Das Einlesen eines Datenelements wird beim ersten ungültigen Zeichen beendet. Beispielsweise wird beim Einlesen einer Dezimalzahl durch %d bei Eingabe von "123XY" der Wert 123 eingelesen, der nächste Aufruf arbeitet bei "X" weiter.

Tritt beim Einlesen einer Zahl ein Überlauf auf, d.h. der angegebene Wert ist größer als der für den zugehörigen Typ in LIMITS.H angegebene Grenzwert, wird der Grenzwert aus LIMITS.H an den Anwender zurückgeliefert und `errno` auf ERANGE gesetzt. Wird beispielsweise beim Einlesen einer Zahl vom Typ Unsigned–Short (%hd) ein Wert größer als USHRT_MAX (65535) angegeben, liefert `vscanf` den Wert USHRT_MAX zurück.

Beim Einlesen von Oktal– oder Hexadezimal–Zahlen sind Vorzeichen nicht zulässig.

Rückgabewert

Rückgabewert	Bedeutung
Anzahl	der erfolgreich verglichenen und zugewiesenen Formatelemente
0	Vergleich beginnt mit einem Konflikt zwischen Eingabe–Zeichen und Control–String
EOF	Eingabe vor dem ersten Konflikt oder Konvertierung beendet

Siehe auch

vfprintf (Seite 414)
vfscanf (Seite 416)
vscanf (Seite 420)
fscanf (Seite 270)
printf (Seite 329)
scanf (Seite 350)
sscanf (Seite 376)

3.180 write

Funktion

Daten in eine Datei schreiben

Syntax

```
#include <io.h>

size_t    write(
            int handle,
            void *buf,
            size_t nbytes);
```

Parameter

Parametername	Bedeutung
handle	Deskriptor
buf	Zeiger auf Puffer
nbytes	Anzahl zu übertragender Bytes

Beschreibung

`write` schreibt `nbytes` Bytes aus dem Puffer `buf` in die mit `handle` verbundenen Datei. `handle` ist ein Deskriptor, der mit `open` zurückgegeben wird. Der Dateizeiger wird bei jedem Schreibvorgang aktualisiert.

Hinweis

Bei einer im Textmodus geöffneten Datei bzw. Unit wird '\n' in '\r\n' umgewandelt.

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
Integer–Zahl	Anzahl der übertragenen Bytes (Funktion erfolgreich)
–1	Funktion nicht erfolgreich

Fehlercodes

Im Fehlerfall wird `errno` auf einen der folgenden Werte gesetzt:

Fehlercode	Bedeutung
EACCES	Ungültiges Zugriffsrecht
EAGAIN	Dateisatz gesperrt
EBADF	Deskriptor nicht korrekt
EIO	Physikalischer Schreib/Lesefehler
EINVAL	Ungültiger Parameter
ENOENT	Die angegebene Datei oder eine Komponente von `fname` existiert nicht (Laufwerk wurde eventuell nicht angemeldet (`mount`)).
ENOMEM	Es konnte nicht ausreichend Speicher dem angegebenen Pool entnommen werden.
ENOSPC	Kein Speicherplatz vorhanden
ERANGE	Interner RMOS3–Fehler

Siehe auch

close (Seite 227)
open (Seite 325)
read (Seite 342)

3.181 x_cr_gettaskid

Funktion

ID der aufrufenden Task ermitteln

Syntax

```
#include <task.h>

unsigned int    x_cr_gettaskid (void);
```

Beschreibung

Mit dieser Funktion kann ein Programm die eigene Task–ID feststellen.

Hinweis

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
int	ID der aufrufenden Task.

Siehe auch

x_cr_gettaskparam (Seite 429)

3.182 x_cr_gettaskparam

Funktion

`stdin`, `stdout`, `stderr` und Taskumgebung ermitteln

Syntax

```
#include <task.h>

int    x_cr_gettaskparam(
           unsigned taskid,
           STDSTRUCT *stddata,
           char **env);
```

Parameter

Parametername	Bedeutung
taskid	ID der Task, deren Parameter ausgelesen werden sollen. Wird hier –1 angegeben, werden die Parameter der aufrufenden Task ausgelesen
stddata	Zeiger auf eine Struktur vom Typ `STDSTRUCT` (siehe unten), in die (falls kein NULL–Zeiger übergeben wird) die Parameter der durch `taskid` angegebenen Task eingetragen werden.
env	Adresse eines Zeigers, der (falls ungleich NULL) nach dem Funktionsaufruf auf die Umgebung der in `taskid` angegebenen Task zeigt. Der Zeiger wird auf NULL gesetzt, wenn die Umgebung der angegebenen Task leer ist.

Beschreibung

`x_cr_gettaskparam` ermittelt `stdin`, `stdout`, `stderr` sowie die Umgebung einer von CRUN verwalteten Task `taskid` und gibt diese Parameter über `stddata` und `env` zurück.

Definition von `STDSTRUCT`:

```
struct std_struct
{
int     stdin_dev;
int     stdin_unit;
int     stdout_dev;
int     stdout_unit;
int     stderr_dev;
int     stderr_unit;
char    *stdin_fname;
char    *stdout_fname;
char    *stderr_fname;
char    *tmp_path;
};
typedef struct std_struct STDSTRUCT;
```

`stdin_dev`

Device–ID für stdin. Sind `stdin_dev` und `stdin_unit` gleich –1, so ist `stdin` auf eine Datei umgelenkt. `stdin_fname` zeigt dann auf den Namen dieser Datei.

`stdin_unit`

Unit–ID für `stdin`, wenn `stdin_dev` ungleich –1. Ansonsten, falls selbst ungleich –1, Deskriptor eines Socket–Streams.

stdout_dev

Device–ID für stdout. Sind stdout_dev und stdin_dev gleich –1, so ist stdout auf eine Datei umgelenkt. stdout_fname zeigt dann auf den Namen dieser Datei.

stdout_unit

Unit–ID für stdout, wenn stdout_dev ungleich –1. Ansonsten, falls selbst ungleich –1, Deskriptor eines Socket–Streams.

stderr_dev

Device–ID für stderr. Sind stderr_dev und stderr_unit gleich –1, so ist stderr auf eine Datei umgelenkt. stderr_fname zeigt dann auf den Namen dieser Datei.

stderr_unit

Unit–ID für stderr, wenn stderr_dev ungleich –1. Ansonsten, falls selbst ungleich –1, Deskriptor eines Socket–Streams.

stdin_fname

Name der Datei, auf die stdin umgelenkt ist; nur gültig, wenn stdin_dev und stdin_unit gleich –1 sind.

stdout_fname

Name der Datei, auf die stdout umgelenkt ist; nur gültig, wenn stdout_dev und stdout_unit gleich –1 sind.

stderr_fname

Name der Datei, auf die stderr umgelenkt ist; nur gültig, wenn stderr_dev und stderr_unit gleich –1 sind.

tmp_path

Name eines Verzeichnisses für temporäre Dateien. Wurde sowohl beim Aufruf von xinitt als auch beim Aufruf von xinitc ein NULL–Zeiger angegeben, wird tmp_path gleich NULL gesetzt (siehe auch xinitc und xinitt).

Mit Hilfe von x_cr_gettaskparam (in Kombination mit xinitt und x_cr_initenv) kann eine Task die Initialisierungsdaten einer anderen, unter CRUN ablaufenden, Task übernehmen. Siehe x_cr_initenv für den Aufbau der Umgebung. Falls der Aufruf xinitt bereits durchgeführt wurde, muss für die Übernahme der Initialisierungsdaten ein freopen oder fdureopen Aufruf abgesetzt werden.

Hinweis

Die Pointer stdin_fname, stdout_fname und stderr_fname sind nach einem Aufruf von fclose, freopen bzw. fdureopen auf den zugehörigen Stream (z.B. stdin) ungültig. Die Werte für Device-ID und Unit-ID eines Streams können nach einem freopen oder fdureopen Aufruf auf diesen Stream verändert sein.

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich
–1	Funktion nicht erfolgreich (die mit `taskid` angegebene Task wird nicht von CRUN verwaltet)

Beispiel

```
unsigned TaskID;
char *Env;

STDSTRUCT StdData;

            ...
x_cr_gettaskparam (TaskID, &StdData, &Env);

xinitt (StdData.stdin_dev, StdData.stdin_unit,
        StdData.stdout_dev, StdData.stdout_unit,
        StdData.stderr_dev, StdData.stderr_unit,
        StdData.tmp_path);

if (StdData.stdin_dev==-1)
            freopen (StdData.stdin_fname, "r", stdin);
if (StdData.stdout_dev==-1)
            freopen (StdData.stdout_fname, "w", stdout);
if (StdData.stderr_dev==-1)
            freopen (StdData.stderr_fname, "w", stderr);
x_cr_initenv (Env);
```

Siehe auch

x_cr_gettaskid (Seite 428)

x_cr_initenv (Seite 432)

xinitc (Seite 436)

xinitt (Seite 438)

3.183 x_cr_initenv

Funktion

Taskumgebung initialisieren

Syntax

```
#include <task.h>

int    x_cr_initenv (char *env);
```

Parameter

Parametername	Bedeutung
env	Zeiger auf Umgebungs–Variable

Beschreibung

`x_cr_initenv` initialisiert die Umgebung der aufrufenden Task mit der durch `env` übergebenen Umgebung. Steht für die zu übernehmende Umgebung nicht ausreichend Speicher zur Verfügung, so wird `x_cr_initenv` mit Rückgabewert –1 beendet, ohne dass eine Initialisierung vorgenommen wird. Ruft eine Task mit bereits initialisierter Umgebung `x_cr_initenv` auf, wird die bestehende Umgebung gelöscht und die neue Umgebung übernommen. Steht in diesem Fall nicht ausreichend Speicher zur Verfügung, wird die alte Umgebung nicht gelöscht und der Aufruf mit Rückgabewert –1 abgebrochen. Bei erfolgreicher Initialisierung wird der Funktionsaufruf mit Rückgabewert 0 beendet. Bei `x_cr_initenv` kann z.B. eine durch `x_cr_gettaskparam` ermittelte Umgebung einer anderen Task angegeben werden. Wurde die durch `env` übergebene Umgebung nicht durch `x_cr_gettaskparam` ermittelt, sondern vom Anwender selbst erzeugt, so ist zu beachten, dass die Namen der Umgebungs–Variablen in Großbuchstaben geschrieben werden müssen.

Speicherbedarf:

Ist `env` ungleich NULL, fordert x`_cr_initenv` mit `malloc` Speicher für die zu setzende Umgebung an. Dabei wird soviel Speicher benötigt, wie sich aus der Länge der einzelnen Umgebungs–Variablen in `env` ergibt, sowie ein zusätzliches Byte für jede Variable (Ende–Marke für die Variable) und ein weiteres Byte (Ende–Marke für die Umgebung).

Aufbau der Umgebung

Die Umgebung ist ein Feld vom Typ `char`, in dem die einzelnen Umgebungs–Variablen hintereinander, jeweils durch ein 0–Byte getrennt, abgespeichert werden. Das Ende der Umgebung wird durch zwei aufeinander folgende 0–Bytes gekennzeichnet.

Hinweis

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich
–1	Funktion nicht erfolgreich (nicht genügend Speicher verfügbar, um die angegebene Umgebung für die aufrufende Task zu übernehmen, keine Initialisierung möglich)

Beispiel

Es existieren die zwei Umgebungs–Variablen:

```
PROMPT=$P$G
PATH=A:BIN;A:DEMO\BIN
```

Die Umgebung selbst sieht dann wie folgt aus (‘\0’ bedeutet ein 0–Byte):

```
PROMPT=$P$G’\0’PATH=A:BIN;A:DEMO\BIN’\0’’\0’
```

Siehe auch

x_cr_gettaskparam (Seite 429)

getenv (Seite 279)

putenv (Seite 335)

3.184 x_cr_killtsk

Funktion

Task löschen

Syntax

```
#include <task.h>

int    x_cr_killtsk (unsigned TaskID);
```

Parameter

Parametername	Bedeutung
TaskID	ID der Task, die gelöscht werden soll

Beschreibung

`x_cr_killtsk` beendet und löscht die durch `TaskID` angegebene Task (Zustand NICHT EXISTENT).

Vor dem Beenden der Task werden noch Bereinigungsaktionen durchgeführt:

- Abarbeiten der Routinen, die mit `atexit` installiert wurden.
- Alle durch `fopen` und `fduopen` geöffneten Streams werden geschlossen. Mit `open` eröffnete Dateien werden nicht geschlossen.
- Zuvor durch `malloc`, `calloc` und `realloc` angeforderter Speicher wird zurückgegeben. Außerdem wird der Speicher freigegeben, der durch `RmAlloc` und `RmReAlloc` angefordert wurde, außer es wurde der Mode RM_NOAUTOFREE angegeben.

Hinweis

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich
–1	Der SVC LOCKS ist nicht konfiguriert
–2	Die TCB–Adresse konnte nicht ermittelt werden.
–3	Task-Status konnte nicht ermittelt werden.
–4	Task konnte nicht gelöscht werden.

Siehe auch

atexit (Seite 212)

exit (Seite 242)

3.185 x_cr_setexit

Funktion

Taskspezifischen Exit–Handler setzen

Syntax

```
#include <task.h>

int    x_cr_setexit (void (_FIXED _FAR*exithandler)(int));
```

Parameter

Parametername	Bedeutung
exithandler	Adresse einer Funktion, die bei Beendigung der Task aufgerufen werden soll. Diese Adresse wird von CRUN ungeprüft übernommen. Die Funktion muss eine Pascal (bzw. `fixedparams`) Schnittstelle und bei Verwendung von CAD-UL oder Intel C einen `far` Einsprung besitzen

Beschreibung

`x_cr_setexit` übergibt CRUN mit `exithandler` die Adresse einer Funktion, die von `exit` unmittelbar vor Beendigung der Task aufgerufen wird. Dabei übergibt CRUN beim Aufruf dieses Exit–Handlers den bei `exit` angegebenen Status. Der Exit–Handler sollte die Task beenden; es muss nicht mehr in das aufrufende `exit` zurückgesprungen werden. Nach dem Anspringen des Exit–Handlers dürfen unter keinen Umständen CRUN–Funktionen aufgerufen werden. Der Aufruf `x_cr_setexit(NULL)` entfernt einen zuvor installierten Exit–Handler. Nach `xinitt` ist in CRUN die Adresse des Exit–Handlers mit NULL belegt, d.h. es ist kein Exit–Handler installiert

Hinweis

`x_cr_setexit` darf bei Tasks, die durch den CLI gestartet werden ("main"), nicht benutzt werden, da der CLI die Funktion `x_cr_setexit` zum Ausführen von Bereinigungsaktionen benutzt.

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich
–1	Funktion nicht erfolgreich (die aufrufende Task wurde bisher nicht von CRUN verwaltet; eine automatische Initialisierung der Task (siehe `xinitt`) war nicht möglich)

Siehe auch

atexit (Seite 212)

exit (Seite 242)

xinitt (Seite 438)

3.186 xinitc

Funktion

CRUN initialisieren

Syntax

```
#include <task.h>

int    xinitc(
          unsigned int poolid,
          unsigned int reserved1,
          int reserved2,
          int reserved3,
          char *tmppath);
```

Parameter

Parametername	Bedeutung
poolid	ID eines Speicher–Pools, aus dem CRUN bei den Aufrufen `malloc`, `realloc` und `calloc` Speicher entnimmt. `poolid` muss zu einem in der Konfigurierung festgelegten Speicher–Pool gehören, der groß genug definiert ist.
reserved1	Dieser Parameter ist reserviert und wird ignoriert.
reserved2	Dieser Parameter ist reserviert und wird ignoriert
reserved3	Dieser Parameter ist reserviert und wird ignoriert.
tmppath	Verzeichnis, das die Funktionen `tmpnam` und `tmpfile` durchsuchen, um festzustellen, ob ein Dateiname schon vergeben ist. Falls `tmpnam` und `tmpfile` nicht verwendet werden, kann hier optional auch ein NULL–Zeiger übergeben werden. Ein gültiger Verzeichnisname ist beispielsweise "C:\\TEMP".

Beschreibung

`xinitc` initialisiert die Datenstrukturen für die CRUN-interne Taskverwatung.

`xinitc` muss einmal aufgerufen werden, bevor andere Aufrufe an CRUN abgesetzt werden dürfen. Wird `xinitc` mehrmals aufgerufen, wird nur der erste Aufruf berücksichtigt, alle weiteren werden ignoriert. Wurde `xinitc` zu einem Zeitpunkt, an dem die Initialisierung erforderlich ist, noch nicht aufgerufen, so wird eine automatische Initialisierung vorgenommen, mit `poolid = -1 (Heap), tmppath = NULL`.

Hinweis

Der Aufruf `xinitc` wird beim Systemhochlauf automatisch durchgeführt. Er wird nicht mehr benötigt und existiert nur noch aus Gründen der Kompatibilität zu früheren Versionen.

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion nicht ausgeführt, da CRUN bereits initialisiert
1	Funktion ausgeführt
–1	Der mit `poolid` angegebene Speicherpool ist ungültig oder kein Speicher mehr vorhanden.

Siehe auch

x_cr_gettaskparam (Seite 429)

calloc (Seite 217)

malloc (Seite 312)

realloc (Seite 344)

tmpfile (Seite 405)

tmpnam (Seite 406)

xinitt (Seite 438)

3.187 xinitt

Funktion

Taskspezifische Initialisierung der Runtime-Umgebung vornehmen

Syntax

```
#include <task.h>

int    xinitt(
        int stdin_dev,
        int stdin_unit,
        int stdout_dev,
        int stdout_unit,
        int stderr_dev,
        int stderr_unit,
        char *tmppath);
```

Parameter

Parametername	Bedeutung
stdin_dev	ID des Treibers von dessen Gerät `stdin` eine Eingabe erwartet
stdin_unit	ID des Geräts von dem `stdin` eine Eingabe erwartet.
stdout_dev	ID des Treibers über dessen Gerät `stdout` eine Ausgabe macht.
stdout_unit	ID des Geräts über das `stdout` eine Ausgabe macht.
stderr_dev	ID des Treibers über dessen Gerät `stderr` eine Ausgabe macht.
stderr_unit	ID des Geräts über das `stderr` eine Ausgabe macht.
tmppath	Verzeichnis (z.B. "C:\\TEMP"), das von den Funktionen `tmpnam` und `tmpfile` durchsucht wird, um festzustellen, ob ein Dateiname schon vergeben ist. Optional kann auch ein NULL–Zeiger übergeben werden. Es wird dann der in `xinitc` übergebene Name verwendet.

Beschreibung

`xinitt` initialisiert die taskspezifischen Streams für `stdin`, `stdout` und `stderr`, sowie den taskspezifischen Pfad für temporäre Dateien. Die Environmentvariable TZ wird auf den Default-Wert "TZ=MEZ-1MSZ" gesetzt. Das Current Working Directory wird nicht gesetzt.

`stdin` erwartet Eingaben vom Wertepaar `stdin_dev`/`stdin_unit`, `stdout` macht Ausgaben über das Wertepaar `stdout_dev`/`stdout_unit` und `stderr` macht Ausgaben über das Wertepaar `stderr_dev`/`stderr_unit`. Im Regelfall stehen die Wertepaare für ein vom BYT–Treiber verwaltetes Gerät (Device–ID/Unit–ID). `xinitt` kehrt mit 0 zurück, wenn eine Initialisierung möglich war, mit –1, wenn sie nicht möglich (z.B. wenn keine Datenstrukturen für die Streams allokiert werden konnten), oder mit 1, wenn die aufgerufene Task bereits initialisiert ist.

`tmppath` gibt ein Verzeichnis an, das die Funktionen `tmpnam` und `tmpfile` durchsuchen, ob ein temporärer Dateiname schon vorhanden ist. Ein gültiger `tmppath` ist beispielsweise "C:\TEMP".

xinitt muss von jeder Task zu Beginn einmal aufgerufen werden. Erst nach dem erfolgreichen Aufruf dürfen die eigentlichen C–Bibliotheksfunktionen benutzt werden (Bei Tasks, die vom CLI gestartet werden, führt der CLI den Aufruf von xinitt aus). Die initialisierten Streams sind für jede Task individuell. stdout von Task 1 kann die Ausgaben über ein ganz anderes Gerät machen als Task 2 (abhängig von den Parametern für xinitt). Bei Verwendung von stdin, stdout, stderr, errno, errno2, timezone und daylight wird ebenfalls eine taskspezifische Initialisierung vorausgesetzt.

Falls xinitt nicht zu Beginn einer Task aufgerufen wird, erfolgt beim ersten Bibliotheks–Aufruf, der eine taskspezifische Runtime–Umgebung voraussetzt, eine automatische Initialisierung.

Wurde die Task durch RmCreateTask erzeugt, wird bei der automatischen Initialisierung die System-Konsole für stdin, stdout und stderr gültig. Die Environmentvariable TZ wird auf den Default-Wert "TZ=MEZ-1MSZ" gesetzt. Das Current Working Directory wird nicht gesetzt.

Wurde die Task durch RmCreateChildTask erzeugt, werden bei der automatischen Initialisierung die Konsole für stdin, stdout und stderr, das Environment und das Current Working Directory von der Parent-Task übernommen. Ist stdin, stdout und stderr auf eine Datei umgelenkt, wird statt dessen die System-Konsole benutzt.

Hinweis

Zeiger auf Streams (Typ FILE) sind nur innerhalb einer Task gültig. Es ist nicht zulässig, dass ein Zeiger auf einen Stream beispielsweise über eine Mailbox an eine andere Task weitergegeben und von dieser für Ausgaben verwendet wird.

Diese Funktion ist eine Erweiterung des ANSI C Sprachumfanges.

Rückgabewert

Rückgabewert	Bedeutung
0	Funktion erfolgreich
–1	Funktion nicht erfolgreich
1	Die aufrufende Task ist bereits initialisiert

Siehe auch

x_cr_gettaskparam (Seite 429)

x_cr_setexit (Seite 435)

xinitc (Seite 436)

RmCreateChildTask (Seite 31)

RmCreateTask (Seite 42)

tmpfile (Seite 405)

tmpnam (Seite 406)