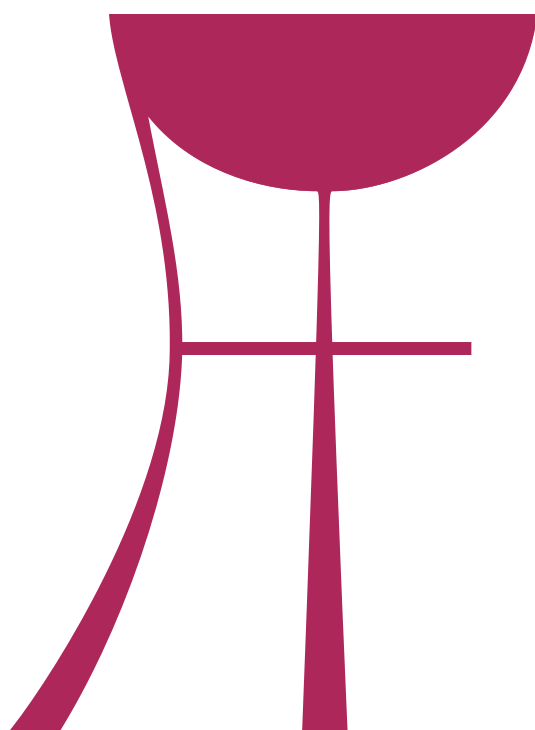
μITRON4.0仕様

Ver. 4.03.00



ITRON

監修　坂村　健

編集／発行　社団法人 トロン協会

μITRON4.0仕様（Ver. 4.03.00）

本仕様書の著作権は，(社) トロン協会に属しています.

(社) トロン協会は，本仕様書の全文または一部分を改変することなく複写し，無料または実費程度の費用で再配布することを許諾します．ただし，本仕様書の一部分を再配布する場合には，μITRON4.0仕様書からの抜粋である旨，抜粋した箇所，および本仕様書の全文を入手する方法を明示することを条件とします．その他，本仕様ならびに本仕様書の利用条件の詳細については，6.1 節を参照してください.

本仕様ならびに本仕様書に関する問い合わせ先は，下記の通りです.

(社) トロン協会 ITRON仕様検討グループ

〒108-0073 東京都港区三田1丁目3番39号 勝田ビル5階

TEL: 03-3454-3191　FAX: 03-3454-3224

---

§ TRONは“The Real-time Operating system Nucleus”の略称です。

§ ITRONは“Industrial TRON”の略称です。

§ μITRONは“Micro Industrial TRON”の略称です。

§ BTRONは“Business TRON”の略称です。

§ CTRONは“Central and Communication TRON”の略称です。

§ T-Kernelは，T-Engineフォーラムが推進しているオープン開発プラットフォーム仕様の名称です。

§ TRON，ITRON，μITRON，BTRON，CTRONおよびT-Kernelは，特定の商品ないしは商品群を指す名称ではありません。

監修の言葉

ITRONは，機器組込み制御用リアルタイム・オペレーティングシステムの仕様としてトロンプロジェクトの発足とともに歩みはじめ，22年が経過した．この間，2001年にT-Engineプロジェクトを開始して以降，トロンプロジェクトにおける最先端リアルタイムOSの標準化活動の場は，次世代ITRONとしてのT-Kernelシリーズに移ったが，従来のμITRON仕様OSが依然として日本で最も多く使われている組込みシステム用OSであることは紛れもない事実である．

これは，ITRONの技術的特長であるリアルタイム性，リソースを浪費しないコンパクトさ，柔軟な仕様適合性，オープンアーキテクチャポリシーが強く支持されている結果であり，ITRONの提唱者としては誠に喜ばしいことだが，一方で，組込み機器の機能の高度化や複雑化，大規模化にも対応して行く必要があり，そのためには，T-Kernelへの移行は不可欠と考えている．

今後は，ITRON/T-Kernelの開発者とユーザからなる「トロンコミュニティ」全体の拡大を図りながら，T-Kernelへの移行を進めて行きたい．

今回のVer.4.03への改訂は，そのような目的で行った．ひとつには，仕様自体は大きな変更をせず，わかりにくかった部分をよりわかり易くしたことと，もうひとつはμITRONからT-Kernelへの移行をし易くするための仕組みを作ったことである．前者には，仕様書中に散在している実装定義についての記述を一覧表にまとめ，利用者の便宜を図ったことも含まれる．また，後者は具体的には，ベーシックプロファイルを新たに設けたことである．

ベーシックプロファイルは，μITRON3.0，μITRON4.0，T-Kernelにおいて同等の機能を持つサービスコールを規定したもので，このプロファイル内で動作するミドルウェアやアプリケーションであれば，μITRONからT-Kernelへの移植をより容易に行うことができる．

このように，μITRON4.0仕様Ver.4.03には，相矛盾するようだが，従来のμITRONユーザへの一層のサービス向上の担い手となると同時に，T-Kernelへの良い橋渡し役になってもらいたいと考えている．

2006年12月
坂村 健
トロンプロジェクト
プロジェクトリーダー

仕様書の構成

本仕様書は，μITRON4.0仕様（ないしは，μITRON4.0リアルタイムカーネル仕様）のC言語APIを規定するものである．仕様のバージョン番号は，仕様書の表紙ならびに各ページの右上に表示されている．

本仕様書の構成は次の通りである．

第1章では，TRONプロジェクトならびにITRONサブプロジェクトの概要とITRON仕様の設計方針について紹介した後，μITRON4.0仕様の位置付けについて解説する．この章の内容は，μITRON4.0仕様策定の背景について説明するもので，μITRON4.0仕様の本体ではない．

第2章は，μITRON4.0仕様ならびにそれと整合するように標準化されるソフトウェア部品仕様に共通する規定を定めるものである．第3章では，μITRON4.0仕様における各種の概念と，複数の機能単位に共通する定義を示す．第4章では，μITRON4.0仕様の各機能を，機能単位毎に説明する．第5章では，付属的な規定について述べる．

第6章には，仕様の保守体制や参考文献など，仕様に関連する参考情報を掲載する．第7章には，仕様を読む上で参考となる一覧表などを掲載する．ここに掲載する一覧表は，第2章〜第5章の内容を別の観点から整理したものである．第6章および第7章の内容は，μITRON4.0仕様の本体ではない．

仕様書の記述形式

本仕様書では，次の記述形式を採用している．

【スタンダードプロファイル】

この項では，μITRON4.0 仕様のスタンダードプロファイルにおける規定を示す．具体的には，μITRON4.0 仕様に規定される機能の中でスタンダードプロファイルがサポートすべき機能の範囲，スタンダードプロファイルがサポートすべきサービスコールおよび静的APIの機能説明中でスタンダードプロファイルには適用されない規定，μITRON4.0仕様では規定されていないがスタンダードプロファイルには適用される規定などについて述べる．

【補足説明】

この項は，μITRON4.0仕様の本体だけではわかりにくい点や誤解するおそれのある点について説明を補足するもので，μITRON4.0仕様の本体ではない．

【μITRON3.0仕様との相違】

この項では，μITRON3.0 仕様との主な相違と，変更の理由について説明する．μITRON3.0仕様と異なる規定がされている点を中心に説明し，μITRON4.0仕様で追加された点や規定が明確になった点については網羅していない．この項の内容は，μITRON4.0仕様の本体ではない．

【仕様決定の理由】

この項では，仕様の決定理由として特に説明を必要とする点について説明する．この項の内容は，μITRON4.0仕様の本体ではない．

また，第4章のサービスコールと静的APIの機能説明では，上記に加えて次の記述形式を用いる．

各サービスコールまたは静的APIの機能説明は, 次の形式のヘッダで開始する．

APIの名称　　APIの説明　　　　　　　　　　　　　プロファイル

「APIの名称」には，サービスコールまたは静的APIの名称を記述する．「APIの説明」では，サービスコールまたは静的APIの機能を簡潔に記述する．「プロファイル」欄に【S】と記されたサービスコールおよび静的APIは，スタンダードプロファイルでサポートしなければならないものである．【B】はベーシックプロファイルを示す．

【静的API】

この項では，静的APIのシステムコンフィギュレーションファイル中での記述形式を示す．

【C言語API】

この項では，サービスコールのC言語からの呼出し形式を示す．

【パラメータ】

この項では，サービスコールおよび静的APIに渡すパラメータをリストアップし，それぞれのパラメータのデータ型と名称，簡単な説明を示す．

【リターンパラメータ】

この項では，サービスコールが返すリターンパラメータをリストアップし，それぞれのリターンパラメータのデータ型と名称，簡単な説明を示す．

【エラーコード】

この項では，サービスコールが返すメインエラーコードをリストアップし，それぞれのメインエラーコードを返す理由を簡単に説明する．ただし，多くのサービスコールが共通の理由で返すメインエラーコードについては，サービスコール毎には記述しない（2.1.6節参照）．

【機能】

この項では，サービスコールおよび静的APIの機能について説明する．

サービスコールや定数などの名称で，イタリック体で記述した文字は，他の文字に置き換えられるものであることを示す．例えば，cre_*yyy*（*yyy*がイタリック体）は，cre_tsk，cre_sem，cre_flgなどをあらわす．

オブジェクト属性やサービスコールの動作モードなどのパラメータで，いくつかの値を選択して設定する場合には，次のような記述方法を用いる．

［*x*］	*x*を指定しても指定しなくてもよいことを示す．
x ｜ *y*	*x*と*y*をビット毎に論理和をとって指定することを示す．
x ‖ *y*	*x*または*y*のいずれか片方を指定できることを示す．

例えば，((TA_HLNG ‖ TA_ASM) ｜ ［TA_ACT］）は，以下の4種類のいずれかの指定ができることを示す．

TA_HLNG
TA_ASM
（TA_HLNG ｜ TA_ACT）
（TA_ASM ｜ TA_ACT）

目次

監修の言葉 i
仕様書の構成 ii
仕様書の記述形式 iii
目次 v
サービスコール索引 ix

第1章　μITRON4.0仕様策定の背景 1

1.1　トロンプロジェクト 1
1.1.1　トロンプロジェクトとは？ 1
1.1.2　プロジェクトの基本コンセプト 1
1.1.3　これまでの成果 2
1.1.4　今後の展望 2
1.2　ITRON仕様の歴史と現状 4
1.2.1　組込みシステムの現状と特徴 4
1.2.2　組込みシステム用のRTOSに対する要求 4
1.2.3　ITRON仕様の現状 6
1.3　ITRON仕様の設計方針 8
1.4　μITRON4.0仕様の位置付け 10
1.4.1　ITRONサブプロジェクトの第2フェーズの標準化活動 10
1.4.2　μITRON4.0仕様の策定の必要性 12
1.4.3　スタンダードプロファイルの導入 12
1.4.4　より広いスケーラビリティの実現 14
1.4.5　μITRON4.0仕様における新機能 15
1.4.6　μITRON4.0仕様公開後の標準化活動 19

第2章　ITRON仕様共通規定 21

2.1　ITRON仕様共通概念 21
2.1.1　用語の意味 21
2.1.2　APIの構成要素 22
2.1.3　オブジェクトのID番号とオブジェクト番号 24
2.1.4　優先度 25
2.1.5　機能コード 26
2.1.6　サービスコールの返値とエラーコード 26
2.1.7　オブジェクト属性と拡張情報 28
2.1.8　タイムアウトとノンブロッキング 28
2.1.9　相対時間とシステム時刻 30
2.1.10　システムコンフィギュレーションファイル 30
2.1.11　静的APIの文法とパラメータ 34
2.2　APIの名称に関する原則 36
2.2.1　ソフトウェア部品識別名 36

2.2.2 サービスコール 37
2.2.3 コールバック 37
2.2.4 静的API 37
2.2.5 パラメータとリターンパラメータ 37
2.2.6 データ型 39
2.2.7 定数 39
2.2.8 マクロ 40
2.2.9 ヘッダファイル 40
2.2.10 カーネルとソフトウェア部品の内部識別子 41
2.3 ITRON仕様共通定義 41
2.3.1 ITRON仕様共通データ型 41
2.3.2 ITRON仕様共通定数 43
2.3.3 ITRON仕様共通マクロ 47
2.3.4 ITRON仕様共通静的API 48
第3章 μITRON4.0仕様の概念と共通定義 49
3.1 基本的な用語の意味 49
3.2 タスク状態とスケジューリング規則 50
3.2.1 タスク状態 50
3.2.2 タスクのスケジューリング規則 53
3.3 割込み処理モデル 55
3.3.1 割込みハンドラと割込みサービスルーチン 55
3.3.2 割込みの指定方法と割込みサービスルーチンの起動 57
3.4 例外処理モデル 58
3.4.1 例外処理の枠組み 58
3.4.2 CPU例外ハンドラで行える操作 59
3.5 コンテキストとシステム状態 59
3.5.1 処理単位とコンテキスト 59
3.5.2 タスクコンテキストと非タスクコンテキスト 60
3.5.3 処理の優先順位とサービスコールの不可分性 61
3.5.4 CPUロック状態 63
3.5.5 ディスパッチ禁止状態 64
3.5.6 ディスパッチ保留状態の間のタスク状態 66
3.6 非タスクコンテキストからのサービスコール呼出し 67
3.6.1 非タスクコンテキストから呼び出せるサービスコール 67
3.6.2 サービスコールの遅延実行 69
3.6.3 呼び出せるサービスコールの追加 70
3.7 システム初期化手順 71
3.8 オブジェクトの登録とその解除 72
3.9 処理単位の記述形式 73
3.10 カーネル構成定数／マクロ 74
3.11 カーネル共通定義 75

3.11.1 カーネル共通定数 75
3.11.2 カーネル共通構成定数 76

第4章　µITRON4.0仕様の機能 79

4.1　タスク管理機能 79
4.2　タスク付属同期機能 101
4.3　タスク例外処理機能 113
4.4　同期・通信機能 126
4.4.1　セマフォ 126
4.4.2　イベントフラグ 135
4.4.3　データキュー 147
4.4.4　メールボックス 159
4.5　拡張同期・通信機能 170
4.5.1　ミューテックス 170
4.5.2　メッセージバッファ 181
4.5.3　ランデブ 194
4.6　メモリプール管理機能 214
4.6.1　固定長メモリプール 214
4.6.2　可変長メモリプール 224
4.7　時間管理機能 235
4.7.1　システム時刻管理 235
4.7.2　周期ハンドラ 240
4.7.3　アラームハンドラ 250
4.7.4　オーバランハンドラ 258
4.8　システム状態管理機能 266
4.9　割込み管理機能 278
4.10 サービスコール管理機能 291
4.11 システム構成管理機能 296

第5章　付属規定 305

5.1　µITRON4.0仕様準拠の条件 305
5.1.1　基本的な考え方 305
5.1.2　µITRON4.0仕様準拠の最低機能 306
5.1.3　µITRON4.0仕様に対する拡張 307
5.2　ベーシックプロファイル 308
5.3　自動車制御用プロファイル 309
5.3.1　制約タスク 310
5.3.2　自動車制御用プロファイルに含まれる機能 310
5.4　仕様のバージョン番号 313
5.5　メーカコード 313

第6章　付録 317

6.1　仕様と仕様書の利用条件 317

6.2 仕様の保守体制と参考情報 ... 318
6.3 仕様策定の経緯とバージョン履歴 ... 319

第7章 リファレンス 321

7.1 サービスコール一覧 ... 321
7.2 静的API一覧 ... 326
7.3 スタンダードプロファイルの静的APIとサービスコール ... 327
7.4 データ型 ... 329
7.5 パケット形式 ... 332
7.6 定数とマクロ ... 339
7.7 構成定数とマクロ ... 341
7.8 エラーコード一覧 ... 342
7.9 機能コード一覧 ... 344
7.10 実装毎に規定すべき事項（実装定義）一覧表 ... 346
索引 ... 353
静的API索引 ... 360

サービスコール索引

この索引は，μITRON4.0仕様のサービスコールのアルファベット順の索引である．

acp_por ランデブの受付 204
acre_alm アラームハンドラの生成（ID番号自動割付け） 252
acre_cyc 周期ハンドラの生成（ID番号自動割付け） 243
acre_dtq データキューの生成（ID番号自動割付け） 150
acre_flg イベントフラグの生成（ID番号自動割付け） 137
acre_isr 割込みサービスルーチンの生成（ID番号自動割付け） 283
acre_mbf メッセージバッファの生成（ID番号自動割付け） 184
acre_mbx メールボックスの生成（ID番号自動割付け） 162
acre_mpf 固定長メモリプールの生成（ID番号自動割付け） 216
acre_mpl 可変長メモリプールの生成（ID番号自動割付け） 226
acre_mtx ミューテックスの生成（ID番号自動割付け） 174
acre_por ランデブポートの生成（ID番号自動割付け） 198
acre_sem セマフォの生成（ID番号自動割付け） 128
acre_tsk タスクの生成（ID番号自動割付け） 83
act_tsk タスクの起動 87
cal_por ランデブの呼出し 201
cal_svc サービスコールの呼出し 295
can_act タスク起動要求のキャンセル 88
can_wup タスク起床要求のキャンセル 106
chg_i*xx* 割込みマスクの変更 289
chg_pri タスク優先度の変更 94
clr_flg イベントフラグのクリア 142
cre_alm アラームハンドラの生成 252
cre_cyc 周期ハンドラの生成 243
cre_dtq データキューの生成 150
cre_flg イベントフラグの生成 137
cre_isr 割込みサービスルーチンの生成 283
cre_mbf メッセージバッファの生成 184
cre_mbx メールボックスの生成 162
cre_mpf 固定長メモリプールの生成 216
cre_mpl 可変長メモリプールの生成 226
cre_mtx ミューテックスの生成 174
cre_por ランデブポートの生成 198
cre_sem セマフォの生成 128
cre_tsk タスクの生成 83
def_exc CPU例外ハンドラの定義 298

def_inh 割込みハンドラの定義......281
def_ovr オーバランハンドラの定義......260
def_svc 拡張サービスコールの定義......293
def_tex タスク例外処理ルーチンの定義......118
del_alm アラームハンドラの削除......254
del_cyc 周期ハンドラの削除......246
del_dtq データキューの削除......152
del_flg イベントフラグの削除......139
del_isr 割込みサービスルーチンの削除......285
del_mbf メッセージバッファの削除......187
del_mbx メールボックスの削除......165
del_mpf 固定長メモリプールの削除......218
del_mpl 可変長メモリプールの削除......228
del_mtx ミューテックスの削除......176
del_por ランデブポートの削除......200
del_sem セマフォの削除......130
del_tsk タスクの削除......86
dis_dsp ディスパッチの禁止......271
dis_int 割込みの禁止......287
dis_tex タスク例外処理の禁止......122
dly_tsk 自タスクの遅延......112
ena_dsp ディスパッチの許可......272
ena_int 割込みの許可......288
ena_tex タスク例外処理の許可......123
exd_tsk 自タスクの終了と削除......91
ext_tsk 自タスクの終了......90
frsm_tsk 強制待ち状態からの強制再開......111
fsnd_dtq データキューへの強制送信......155
fwd_por ランデブの回送......206
get_i*xx* 割込みマスクの参照......290
get_mpf 固定長メモリブロックの獲得......219
get_mpl 可変長メモリブロックの獲得......229
get_pri タスク優先度の参照......96
get_tid 実行状態のタスクIDの参照......268
get_tim システム時刻の参照......238
iact_tsk タスクの起動......87
ifsnd_dtq データキューへの強制送信......155
iget_tid 実行状態のタスクIDの参照......268
iloc_cpu CPUロック状態への移行......269
ipsnd_dtq データキューへの送信（ポーリング）......153

iras_tex タスク例外処理の要求 120
irel_wai 待ち状態の強制解除 107
irot_rdq タスクの優先順位の回転 267
iset_flg イベントフラグのセット 140
isig_sem セマフォ資源の返却 131
isig_tim タイムティックの供給 239
iunl_cpu CPUロック状態の解除 270
iwup_tsk タスクの起床 104
loc_cpu CPUロック状態への移行 269
loc_mtx ミューテックスのロック 177
pacp_por ランデブの受付（ポーリング） 204
pget_mpf 固定長メモリブロックの獲得（ポーリング） 219
pget_mpl 可変長メモリブロックの獲得（ポーリング） 229
ploc_mtx ミューテックスのロック（ポーリング） 177
pol_flg イベントフラグ待ち（ポーリング） 143
pol_sem セマフォ資源の獲得（ポーリング） 132
prcv_dtq データキューからの受信（ポーリング） 156
prcv_mbf メッセージバッファからの受信（ポーリング） 190
prcv_mbx メールボックスからの受信（ポーリング） 167
psnd_dtq データキューへの送信（ポーリング） 153
psnd_mbf メッセージバッファへの送信（ポーリング） 188
ras_tex タスク例外処理の要求 120
rcv_dtq データキューからの受信 156
rcv_mbf メッセージバッファからの受信 190
rcv_mbx メールボックスからの受信 167
ref_alm アラームハンドラの状態参照 257
ref_cfg コンフィギュレーション情報の参照 300
ref_cyc 周期ハンドラの状態参照 249
ref_dtq データキューの状態参照 158
ref_flg イベントフラグの状態参照 146
ref_isr 割込みサービスルーチンの状態参照 286
ref_mbf メッセージバッファの状態参照 192
ref_mbx メールボックスの状態参照 169
ref_mpf 固定長メモリプールの状態参照 222
ref_mpl 可変長メモリプールの状態参照 233
ref_mtx ミューテックスの状態参照 180
ref_ovr オーバランハンドラの状態参照 264
ref_por ランデブポートの状態参照 212
ref_rdv ランデブの状態参照 213
ref_sem セマフォの状態参照 134

ref_sys システムの状態参照 277
ref_tex タスク例外処理の状態参照 125
ref_tsk タスクの状態参照 97
ref_tst タスクの状態参照（簡易版） 100
ref_ver バージョン情報の参照 301
rel_mpf 固定長メモリブロックの返却 221
rel_mpl 可変長メモリブロックの返却 231
rel_wai 待ち状態の強制解除 107
rot_rdq タスクの優先順位の回転 267
rpl_rdv ランデブの終了 210
rsm_tsk 強制待ち状態からの再開 111
set_flg イベントフラグのセット 140
set_tim システム時刻の設定 237
sig_sem セマフォ資源の返却 131
slp_tsk 起床待ち 103
snd_dtq データキューへの送信 153
snd_mbf メッセージバッファへの送信 188
snd_mbx メールボックスへの送信 166
sns_ctx コンテキストの参照 273
sns_dpn ディスパッチ保留状態の参照 276
sns_dsp ディスパッチ禁止状態の参照 275
sns_loc CPUロック状態の参照 274
sns_tex タスク例外処理禁止状態の参照 124
sta_alm アラームハンドラの動作開始 255
sta_cyc 周期ハンドラの動作開始 247
sta_ovr オーバランハンドラの動作開始 262
sta_tsk タスクの起動（起動コード指定） 89
stp_alm アラームハンドラの動作停止 256
stp_cyc 周期ハンドラの動作停止 248
stp_ovr オーバランハンドラの動作停止 263
sus_tsk 強制待ち状態への移行 109
tacp_por ランデブの受付（タイムアウトあり） 204
tcal_por ランデブの呼出し（タイムアウトあり） 201
ter_tsk タスクの強制終了 92
tget_mpf 固定長メモリブロックの獲得（タイムアウトあり） 219
tget_mpl 可変長メモリブロックの獲得（タイムアウトあり） 229
tloc_mtx ミューテックスのロック（タイムアウトあり） 177
trcv_dtq データキューからの受信（タイムアウトあり） 156
trcv_mbf メッセージバッファからの受信（タイムアウトあり） 190
trcv_mbx メールボックスからの受信（タイムアウトあり） 167

tslp_tsk 起床待ち（タイムアウトあり）..103
tsnd_dtq データキューへの送信（タイムアウトあり）............................153
tsnd_mbf メッセージバッファへの送信（タイムアウトあり）................188
twai_flg イベントフラグ待ち（タイムアウトあり）................................143
twai_sem セマフォ資源の獲得（タイムアウトあり）................................132
unl_cpu CPUロック状態の解除..270
unl_mtx ミューテックスのロック解除..179
wai_flg イベントフラグ待ち..143
wai_sem セマフォ資源の獲得..132
wup_tsk タスクの起床..104

第1章 μITRON4.0仕様策定の背景

1.1 トロンプロジェクト

1.1.1 トロンプロジェクトとは？

トロン（TRON : The Real-time Operating system Nucleus）は，理想的なコンピュータアーキテクチャの構築を目的として，1984年に東京大学の坂村健博士によって提案された新しいコンピュータOS仕様であり，産業界と大学の協力のもと，新しいコンピュータ体系の実現を目指している．トロンプロジェクトの究極のゴールは「どこでもコンピュータ環境，ユビキタスネットワーク社会」の実現である．

1.1.2 プロジェクトの基本コンセプト

どこでもコンピュータ

トロンプロジェクトは，身の回りの環境にコンピュータ組込みの「カシコイ」機器を遍在させ，それらをネットワークで結ぶことにより人々の生活を助ける「どこでもコンピュータ環境，ユビキタスネットワーク社会」の構築を目的として始められたプロジェクトである．

モバイルを含めた幅広い機器に搭載するためにサイズはコンパクトであり，実環境で利用できるリアルタイム性を重視している．

また，どこでもコンピュータ環境とはコンピュータ組込み機器が人間と環境間のすべての面でのインタフェースとなるという環境である．そこで情報を持てる者と持たざる者の格差，デジタルデバイドが大きな問題となる．コンピュータはだれにでも使えるものでなければならない．そのためトロンでは，プロジェクト開始直後から「イネーブルウェア (Enableware)」というコンセプトで障害者対応も考えてきた．

また，どこでもコンピュータ環境でネットワークに対する不正アクセスを許せば，深刻なプライバシー問題や，さらには機器の不正遠隔操作による実害も考えられる．そのため，環境を構成する個々のコンピュータでのセキュリティ保証が必要であり，そのための標準セキュリティ基盤構築をeTRONで行っている．

オープンアーキテクチャ

トロンプロジェクトの成果は公開された仕様という形で一般に入手できる．この仕様をもとに誰でも自由に製品を開発し市場に参入できる．

1.1.3 これまでの成果

以下の仕様を決定し仕様書を公開した

ITRON

組込みシステム用リアルタイムOS 仕様（ITRON，ITRON2，μITRON2，μITRON3.0，μITRON4.0）

JTRON

JavaとITRONのハイブリッドOS仕様（JTRON1.0，JTRON2.0，JTRON2.1）

BTRON

GUI(Graphical User Interface)のOS仕様とその関連仕様（BTRON/286，BTRON1，BTRON2，BTRON3）

CTRON

通信制御や情報処理を目的としたOSインタフェース仕様

トロンヒューマンインタフェース

各種の電子機器のヒューマンインタフェースの標準ガイドライン

1.1.4 今後の展望

T-Engineプロジェクトの推進

次世代リアルタイムシステムのプラットフォームT-Engineにより組込みシステム分野における世界的なイニシアチブの獲得にむけた活動を行う. ITRON仕様は広く世の中に普及することを目的に，OS本体ではなくOSインタフェースを規定する「弱い標準化」の方針を採ったが，結果としてソフトウェアの移植性が必ずしも良くないという問題が生じた．この反省から開始されたT-Engineプロジェクトでは，ハードウェア，カーネル，オブジェクトフォーマットを規定し「より強い標準化」を行っている.

トロン先進技術の研究開発

セキュアなアーキテクチャ基盤(eTRON)や，次世代ユビキタスコンピューティング環境としての超機能分散システム(HFDS : Highly Functionally Distributed System)といった，日本型IT技術やインフラの構築に向けた，先進技術の研究開発，およびそれと関連する動向調査などを行う.

ITRON仕様検討

T-Engineフォーラムと連携を密して，T-Kernelへの移行がスムーズできるようにするための仕様検討を行なう．また，仕様書をわかりやすくするための検討を行なう．

多文字OSの応用

電子政府や地域情報システム，電子ブックシステムといった，BTRON仕様OSの多文字応用を促進する活動を行う．

教育・普及

組込み機器向けリアルタイムシステムに関する技術者の育成，普及活動を行う

プロモーション活動

トロンプロジェクトの活動成果のマーケティング，プロモーション活動を行う．

1.2 ITRON仕様の歴史と現状

1.2.1 組込みシステムの現状と特徴

マイクロプロセッサ技術の発展により，機器組込み制御システム（組込みシステム）の応用分野は拡大の一途をたどっている．当初は，工場の生産ラインの制御など産業用途が中心であったものが，通信機器やオフィス機器などの業務機器分野，さらに近年は，自動車やオーディオ／ビデオ機器，テレビ，携帯電話，電子楽器やゲームマシン，洗濯機，エアコン，照明器具などの民生機器分野に急激に拡大し，今では身の回りのほとんどの電気／電子機器に組込みシステムが応用されるようになっている．

それと並行して，制御対象となる機器の高機能化や複合化に伴って，組込みシステムの大規模化・複雑化も著しい．それに加えて，最近顕著になってきた機器のデジタル化の流れも，マイクロプロセッサの高性能化によりソフトウェアで実現可能な処理が増えていることとあいまって，組込みシステムの重要性を増す結果となっている．

一般に，民生機器に代表される小規模な組込みシステムには，産業機器に代表される規模の大きい組込みシステムと比較して，機器の製造個数が極めて多く，単価が安いという特徴がある．そのため，大規模な組込みシステムでは開発コストを下げることが重視されるのに対して，小規模な組込みシステムでは最終製品の製造コストを下げることが重視される傾向にある．また，特に民生機器分野では，厳しい機器開発競争からシステム開発期間の短縮に対する要求が著しく，また一度販売した機器のソフトウェアを改修することはほとんどないことから，システム開発のライフサイクルが極めて短いことも特徴の1つとなっている．

小規模な組込みシステムの分野では，プロセッサコアに加えて，ROM，RAM，汎用I/Oデバイス，用途に応じたデバイスなどを1チップ化したMCU（Micro Controller Unit ; 1チップマイコンと呼ばれる場合もある）が広く使われている．MCU上のソフトウェアを開発する際には，最終製品のコストダウンの要請から，ハードウェア資源の制約，とりわけメモリ容量の制約が問題になる．また，高いコストパフォーマンスが求められるMCUは，しばしばアプリケーションに最適化して設計されるため，プロセッサコアの種類が極めて多いことも特筆すべき点である．

このような小規模な組込みシステムの分野においても，ソフトウェアの大規模化・複雑化や開発期間短縮に対する要求から，ソフトウェアの生産性の向上は重要な課題となっており，C言語などの高級言語を使うケースや，µITRON仕様などのRTOSを用いるケースが一般的になりつつある．

1.2.2 組込みシステム用のRTOSに対する要求

マイクロプロセッサの高性能化が進む一方で，民生機器など大量生産される機

器への応用が広がっていることから，組込みシステムのコストパフォーマンス向上に対する要請は以前と同様極めて強いものがある．また，組込みシステムの応用分野の拡大に伴って，RTOSを扱うべきソフトウェア技術者も増加しており，システム設計者やプログラマの教育の重要性も高い．

このことを裏付けるデータとして，トロン協会が1996年より毎年実施しているアンケート調査においても，組込みシステム用のRTOSの問題点として，「使いこなせる技術者が不足またはいない」「OSにより仕様の違いが大きく切替えの負担が大きい」という教育上や標準化の問題を挙げた回答者が最も多く，「OSのサイズや使用リソースが大きすぎる」「性能・機能が要求条件に適合しない」という適合性に関する問題を挙げた回答者がそれに次いで多くを占めるという結果が例年得られている．

このような背景からITRONサブプロジェクトでは，概念や用語の統一といった教育面を特に重視して，広い範囲の組込みシステムに共通に適用できるRTOS仕様の標準化が必要であると考え，プロジェクトに取り組んできた．

組込みシステム用のRTOS仕様の標準化にあたって最も困難な問題として，ハードウェアの持つ性能を最大限に発揮させるという要求と，ソフトウェアの生産性を向上させるという要求のトレードオフの解決が挙げられる．ハードウェア資源の制約が厳しいMCUベースのシステムにおいては，与えられたハードウェアの性能を最大限に発揮できることが，RTOSを採用する前提条件となる．一方で，ソフトウェア生産性の向上はRTOSを用いる最大の動機であるが，ソフトウェアの生産性を向上させるためにOSが提供するサービスの抽象度を上げたり，用いるハードウェアによらずにソースコードレベルの完全な移植性を確保しようとすると，OSが提供するサービスとハードウェアアーキテクチャとのギャップが実行時のオーバヘッドにつながり，ハードウェアの持つ性能を最大限に発揮させることが難しくなる．

この2つの要求の最適なトレードオフは，組込み機器の性質に大きく依存する．具体的には，小規模なシステムにおいては，最終製品のコストダウンの要求から，実行時性能を低下させてまでソフトウェアの移植性を向上させる意義は少ない．それに対して，既存のソフトウェア部品を用いる場合や，ソフトウェアの再利用が不可欠な規模の大きいシステムにおいては，ソフトウェアの移植性は極めて重要な要求となる．さらに，2つの要求の最適なトレードオフは，マイクロプロセッサ技術の発展によって常に変化している．

また，小規模なシステムと大規模なシステムでは，RTOSに求める機能にも大きな違いがある．小規模な組込みシステムに，必要性の低い高度な機能を持ったOSを用いると，プログラムサイズが大きくなり，実行時性能も低下する結果となる．逆に大規模なシステムでは，高度な機能を持ったOSを用いてソフトウェアの生産性向上を図るべきである．

以上より，組込みシステムの規模や性質に応じて，RTOSに対する要求は大きく異なることがわかる．システムの規模や性質毎に別々のRTOS仕様を定義することも可能ではあるが，ソフトウェア技術者の教育面やソフトウェア部品の

流通性，開発支援ツールのサポート面を考えると，多種多様な組込みシステムに共通に適用できるスケーラビリティを持ったRTOS仕様を定義することが望ましい．

組込みシステム用のRTOS仕様に対する以上の要求を簡単に整理すると，次のようになる．

- ハードウェアの持つ性能を最大限に発揮できること
- ソフトウェアの生産性向上に役立つこと
- 各規模のシステムに適用できるスケーラビリティを持つこと

以上で述べた技術的な要求事項に加えて，仕様が真の意味でオープンであることも重要な要件となる．組込みシステムが身の回りのあらゆる電気／電子機器に適用されることを考えると，仕様書が誰にでも入手可能な形で一般に公開されるだけでなく，それに基づいた製品を誰もが自由にロイヤリティなしで実装・販売できることも満たさなくてはならない要件である．

1.2.3 ITRON仕様の現状

ITRONサブプロジェクトは，1984年に開始して以来，組込みシステム用の標準RTOS仕様について検討を行い，その結果として，一連のITRONリアルタイムカーネル仕様を策定・公開してきた．カーネル仕様に重点を置いて標準化を行ってきたのは，小規模な組込みシステムでは，カーネルの機能のみが利用されるケースが多いためである．

最初のITRON仕様は，1987年にITRON1仕様という形でまとめられた．ITRON1仕様に従っていくつかのリアルタイムカーネルが開発・応用され，仕様の適用性の検証に重要な役割を果たした．その後，小規模な8〜16ビットのMCUに適用するために機能を絞り込んだµITRON仕様（Ver. 2.0），逆に大規模な32ビットのプロセッサに適用するためのITRON2仕様の検討を進め，共に1989年に仕様を公開した．この内µITRON仕様は，極めて限られた計算能力とメモリ容量しか持たないMCU上でも実用的な性能を発揮することができたために，多くの種類のMCU用に実装され，極めて多くの組込みシステムに応用された．実際，組込みシステム向けの主要なMCUのほとんどすべてに，µITRON仕様のカーネルが開発されているといっても過言ではない．

このように，µITRON仕様が広範な分野に応用されるにしたがって，それぞれの機能の必要性や性能に対する要求がより正確にわかってきた．また，MCUの適用分野が広がるに従って，µITRON仕様カーネルを32ビットMCU用に実装するという仕様設計時に想定していなかった適用例も出てきた．そこで，それまでのITRON仕様を再度見直し，8ビットから32ビットまでの各規模のMCUに適用できるスケーラビリティを持った仕様を策定する作業を行った結果，1993年にµITRON3.0仕様を公開した．µITRON3.0仕様には，1つの組込みシステムをネットワークで接続された複数のプロセッサで実現する場合に適用するための，接続機能も含まれている．µITRON3.0の仕様書の英語版は，

"μITRON3.0: An Open and Portable Real-Time Operating System for Embedded Systems" という書名で，IEEE CS Pressから出版されている．

ITRON仕様に準拠したリアルタイムカーネル製品としては, 現在トロン協会に登録されているものだけでも，約60種類のプロセッサ用に36種類の製品がある．最近では，米国のソフトウェアベンダがITRON仕様カーネルを開発する例も出てきている．また，μITRON仕様カーネルは，規模が小さく比較的容易に実装することができるために, ユーザが自社内専用に開発しているケースも多く，製品化されているもの以外にも多くの実装例がある．また，フリーソフトウェアとして配付されているμITRON仕様カーネルも，複数種類ある．

言うまでもなく, このように多くのITRON仕様カーネルが実装されるのは，広い応用分野と極めて多くの応用事例があるためである. ITRON仕様カーネルが使用されている機器の例を表1-1に挙げる．また，先に紹介したトロン協会によるアンケート調査でも, ITRON仕様が特に民生機器の分野において広く使われており，事実上の業界標準仕様となっていることがわかる．また，ITRON仕様カーネルを使っているケースの中で，自社製のITRON仕様カーネルを使用しているケースが多くあり, ITRON仕様が真にオープンな標準仕様となっていることがわかる.

表1-1. ITRON仕様カーネルの主な適用分野

分野	機器の例
AV機器，家電	テレビ，ビデオ，DVDレコーダ，デジタルカメラ，セットトップボックス，オーディオ機器，電子レンジ，炊飯器，エアコン，洗濯機
個人用情報機器，娯楽／教育機器	PDA，電子手帳，カーナビ，ゲーム機，電子楽器
パソコン周辺機器，OA機器	プリンタ，スキャナ，ディスクドライブ，DVDドライブ，コピー，FAX，ワープロ
通信機器	留守番電話機，ISDN電話機，携帯電話，PHS，ATMスイッチ，放送機器／設備，無線設備，人工衛星
運輸機器，工業制御／FA機器，その他	自動車，プラント制御，工業用ロボット，エレベータ，自動販売機，医療用機器，業務用データ端末

リアルタイムカーネル仕様以外では，BTRON仕様のファイルシステムと互換性のあるファイル管理機能を持つITRON/FILE仕様を公開している.

このように，ITRONリアルタイムカーネル仕様は，多くのメーカが規模の異なる様々なプロセッサ用に実装を行い，その多くが製品化され，また広く応用さ

れている．特にμITRON仕様カーネルは，今までメモリ容量や実行速度の制約によってRTOSが使用できなかったシングルチップのMCUへの適用が進んでおり，μITRON仕様がこの分野における世界最初の標準リアルタイムカーネル仕様の地位を築きつつあるということができる．

ITRONサブプロジェクトでは，このような実績をベースとして，標準化の対象をカーネル仕様からソフトウェア部品や開発環境などの周辺仕様へと広げると同時に，応用分野毎の調査研究・標準化活動を進めている（1.4.1節参照）．さらに将来的な方向性としては，トロンプロジェクトのゴールであるHFDSの実現へ向けての検討を進めていく計画である．

1.3 ITRON仕様の設計方針

1.2.2節で述べた要求事項を満たすために，ITRON仕様を設計するにあたって，以下の設計方針を設定している．

- ハードウェアの過度の仮想化を避け，ハードウェアに対する適応化を考慮する．

  ハードウェアの持つ性能を最大限に発揮させ，高いリアルタイム性能を得るためには，仕様作成にあたって，ハードウェアを過度に仮想化することは避けなければならない．ハードウェアに対する適応化とは，ハードウェアの持つ性能や性質に応じてRTOSの仕様や内部の実装方法を変え，システム全体としての性能向上をはかることをいう．

  具体的には，ITRON仕様において，ハードウェアによらず標準化すべき事項と，ハードウェアの持つ性能や性質に応じて最適になるように決定してよい事項を明確に分離した．標準化した項目には，タスクのスケジューリング規則，システムコールの名称と機能，パラメータの名前・順序・意味，エラーコードの名前と意味などが含まれる．一方，標準化すると実行時性能の低下につながるような部分については無理に標準化せず，標準化・仮想化による性能の低下を避けるように配慮した．具体的には，パラメータのビット数や割込みハンドラの起動方法については，実装毎に決める方針としている．

- アプリケーションに対する適応化を考慮する．

  アプリケーションに対する適応化とは，アプリケーションに必要となるカーネルの機能や要求される性能に応じて，カーネルの仕様や内部の実装方法を変更し，システム全体として性能向上をはかるアプローチをいう．組込みシステムの場合，OSのオブジェクトコードはアプリケーション毎に生成するため，アプリケーションに対する適応化が特に有効に働く．

  具体的には，カーネルが提供する各種の機能をできる限り独立させ，アプリケーション毎に必要な機能だけを用いることができるように考慮して，仕様の設計を行っている．また，システムコールの単機能化により，必要な機能のみを組み込むことを容易にしている．実際，多くのμITRON仕様カーネル

は，カーネル自身がライブラリの形で提供されており，アプリケーションプログラムとリンクするだけでカーネルの必要なモジュールだけが組み込まれる仕組みになっている．

- ソフトウェア技術者の教育を重視する．

  小規模な組込みシステムにおいては，あるシステム用に開発されたソフトウェアがそのまま次に開発するシステムに使えることはまれであり，ソフトウェアの互換性や移植性は，それほど重視されない傾向にある．それよりも，カーネル仕様の標準化によって，ソフトウェア技術者の教育が容易になったり，用語や概念の統一を通じて技術者間の意志疎通がスムーズになることが重要と考えられる．

  ITRON仕様では，ソフトウェア技術者の教育を重視し，標準化を通じて一度覚えた事が広く活用できるよう配慮している．具体的には，仕様における用語の使い方や，システムコールなどの名称の決め方などは，できる限り一貫性を持つよう配慮した．また，教育用のテキストの作成にも力を入れている．

- 仕様のシリーズ化やレベル分けを行う．

  多種多様なハードウェアへの適用を可能にするため，仕様のシリーズ化やレベル分けを行う．これまでに作成したリアルタイムカーネル仕様の内，μITRON仕様（Ver. 2.0）は主に8～16ビットMCUを用いた小規模なシステム向けに作成したもので，ITRON2仕様は32ビットプロセッサ向けの仕様である．またこれらの仕様の中でも，機能毎の必要度に応じたレベル分けを行い，カーネルを実装する際には必要度の高い機能のみを実装すればよいものとしている．μITRON3.0仕様では，システムコールのレベル分けにより，1つの仕様で小規模なプロセッサから大規模なプロセッサまでをカバーしている．

  また，ネットワークで接続された分散システムのための仕様や，マルチプロセッサシステムのための仕様も，一連のITRON仕様シリーズの中で標準化を検討している．

- 豊富な機能を提供する．

  カーネルが提供するプリミティブを少数に限定するのではなく，性質の異なる豊富なプリミティブを提供するというアプローチを取る．アプリケーションやハードウェアの性質に適合したプリミティブを利用することで，実行時性能やプログラムの書きやすさの向上が期待できる．

これらの設計方針のいくつかに共通するコンセプトとして，「弱い標準化」がある．弱い標準化とは，共通化すると実行時性能の低下につながるような部分については無理に標準化を行わず，ハードウェアやアプリケーションに依存して決めるべき部分として残すアプローチのことをいう．弱い標準化の考え方により，多種多様なハードウェアの上で，その性能を最大限に発揮させることが可能になる（図1-1）．

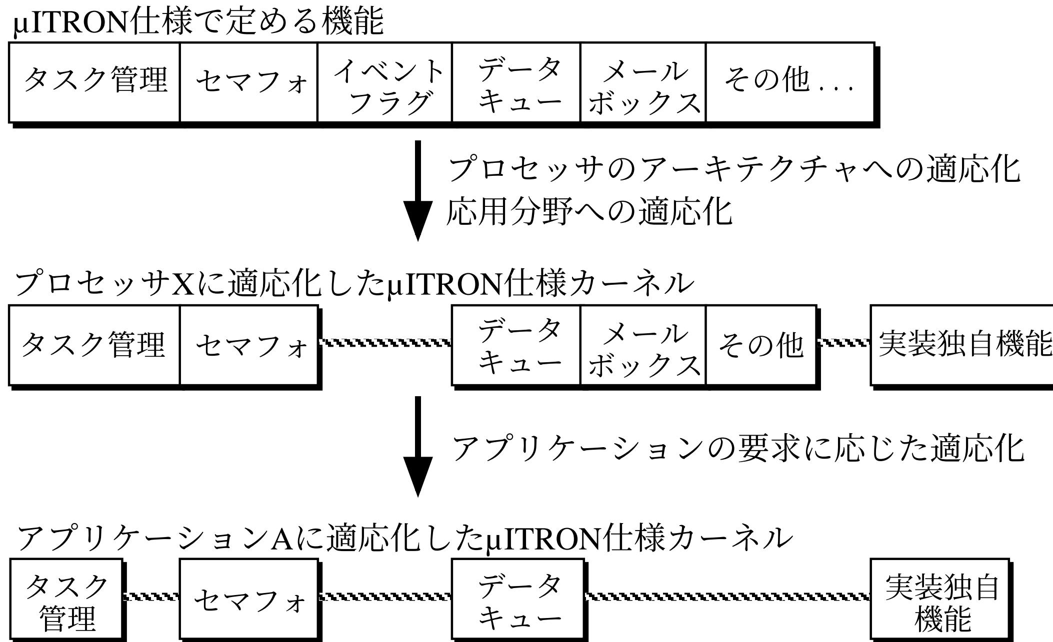


図1-1. μITRON仕様における適応化

1.4 μITRON4.0仕様の位置付け

1.4.1 ITRONサブプロジェクトの第2フェーズの標準化活動

先に述べたように，ITRONサブプロジェクトでは，これまでリアルタイムカーネル仕様を中心に標準化活動を行ってきたが，組込みシステムの大規模化・複雑化が進行するに伴い，リアルタイムカーネルを取り巻く環境を意識した標準化活動の必要性が高まっている．そこで，1996年頃から開始したITRONサブプロジェクトの第2フェーズにおいては，標準化の範囲をカーネル仕様の標準化から周辺仕様を含めた標準化，とりわけ組込みシステム向けのソフトウェア部品のための標準化へと広げて活動を進めている．

ソフトウェア部品のための標準化として，具体的には，ソフトウェア部品の開発・流通を促す条件を整えることに加えて，分野毎のソフトウェア部品インタフェースの標準化を進めている．

一つめのソフトウェア部品の開発・流通を促す条件を整えるための検討として，さらに次の2つの課題について検討を行っている．最初の課題は，現在実装されているITRON仕様カーネルは実装毎の仕様の違いが大きいために，ソフトウェア部品の流通性が十分に確保できないという問題を解決することである．そのためには，弱い標準化の利点を残しつつ，カーネル仕様の標準化レベルを上げることが必要になる．2つめの課題は，リアルタイム性を持ったソフトウェア部品をサポートすることである．ソフトウェア部品の中にはリアルタイム性を求められるものが多くあり，ソフトウェア部品のリアルタイム制約を満たしつつアプリケーションと共存させたり，複数のソフトウェア部品の併

用を可能にする枠組みが求められる．

これらの2つの課題に関する検討結果は，μITRON4.0仕様に反映されている．また，リアルタイムカーネルを用いた組込みシステム設計の標準的な手法を提示するとともに，ハードリアルタイム性を持ったソフトウェア部品をサポートするためのアプリケーション設計ガイドラインの作成を行っている．

分野毎のソフトウェア部品インタフェースの標準化としては，TCP/IPプロトコルスタックのAPI（Application Program Interface）と，Java実行環境とのインタフェースの標準化に取り組んだ．

組込みシステムの分野においても，TCP/IPプロトコルスタックの重要性が増している．TCP/IPのAPIとして広く使われているソケットインタフェースは，組込みシステム（特に小規模なもの）に用いるには，オーバヘッドが大きい，プロトコルスタック内で動的メモリ管理が必要になるといった問題が指摘されている．組込みシステムのための標準的なTCP/IP APIであるITRON TCP/IP API仕様は，ソケットインタフェースの持つこれらの問題を解決し，コンパクトで効率の良いTCP/IPプロトコルスタックを実現することを可能にするために設計された．ITRON TCP/IP API仕様Ver.1.00は，1998年5月に公開した．また，μITRON4.0仕様，T-Kernel，IPv6に対応したVer.2.00は，2006年7月に公開した．

Javaの技術を組込みシステムに適用する現実的な方法として，Java実行環境をITRON仕様カーネル上に実現し，アプリケーションシステムのJavaに適した部分はJavaプログラムの形で，ITRON仕様カーネルの利点を活かせる部分はITRONのタスクとして実装する方法がある．ここで，JavaプログラムとITRONタスクの間の通信インタフェースの標準化が重要な課題となる．JTRON2.0仕様は，この標準インタフェースを定めることを目的に設計されたもので，1998年10月に公開した．

ソフトウェア部品関連以外では，応用分野を絞り込んでの調査研究・標準化活動として，自動車制御分野におけるITRON仕様カーネルに対する要求事項を整理し，標準仕様に対する提案をまとめる活動を行った．そこでの検討結果も，μITRON4.0仕様に反映されている．

デバッグツールに関する標準化については，デバッグツールがITRON仕様カーネルをサポートするためのインタフェースを規定したITRONデバッギングインタフェース仕様Ver.1.00.00を2001年5月に公開した．

これらの他にも，デバイスドライバ設計ガイドラインの作成などの活動を進めている．

1.4.2 μITRON4.0仕様の策定の必要性

前節で紹介した第2フェーズの活動の中で，リアルタイムカーネル仕様を今一度見直す必要性が指摘され，ITRONリアルタイムカーネル仕様としては第4世代に位置付けられるμITRON4.0仕様の策定を行うこととなった．μITRON4.0仕様の策定が必要な理由は，次の4点に集約できる．

(a) ソフトウェアの移植性の向上

近年の組込みソフトウェアの大規模化・複雑化により，アプリケーションソフトウェアを異なるITRON仕様カーネルへ容易に移植できることが，従来よりも強く求めらるようになってきた．また，ソフトウェア部品の流通を促すためにも，ITRON仕様カーネル上に構築されるソフトウェアの移植性の向上は，極めて重要な課題となっている．

(b) ソフトウェア部品向けの機能の追加

従来のITRON仕様では，外販することを前提にソフトウェア部品を構築するには，機能的に不足している部分があった．例えば，ソフトウェア部品のあるサービスルーチンが，どのようなコンテキストから呼び出されたか調べる方法は，拡張レベルでしか用意されていなかった．

(c) 新しい要求・検討成果の反映

ITRONサブプロジェクトでは，ハードリアルタイムサポート研究会（1996年11月～1998年3月）においてハードリアルタイムシステムの構築を容易にするためにリアルタイムカーネルが持つべき機能の検討を，RTOS自動車応用技術委員会（1997年6月～1998年3月）において自動車制御応用におけるリアルタイムカーネルに対する要求事項の整理を行ってきた．これらの新しい要求や検討成果を，リアルタイムカーネル仕様に反映する必要がある．

(d) 半導体技術の進歩への対応

μITRON3.0仕様を策定してから約13年が経過しており，その間に半導体技術は大きく進歩し，組込みシステムに用いられるプロセッサの性能向上やメモリ容量の増加も著しい．そのため，有用な機能ではあるが，μITRON3.0仕様の策定時点ではオーバヘッドが大きいために標準化を見送ったが，現在の技術では許容できるオーバヘッドで実現できると考えられるものもある．

1.4.3 スタンダードプロファイルの導入

ソフトウェアの移植性を向上させるためには，実装が備えるべき機能セットや，それぞれのサービスコールの機能仕様を，厳格に定めることが必要である．言い換えると，仕様の標準化の度合いを強くする必要がある．

一方，μITRON仕様はこれまで，ソフトウェアの移植性よりも，ハードウェアやプロセッサへの適応化を可能にし，実行時のオーバヘッドや使用メモリの削

減を重視する「弱い標準化」の方針に基づいて標準化を行ってきた．「弱い標準化」の方針によりμITRON仕様は，8ビットから64ビットまでの広い範囲のプロセッサに適用可能なスケーラビリティを実現しており，μITRON仕様が広く受け入れられている重要な理由の一つとなっている．ところが，ソフトウェア移植性の向上とスケーラビリティの実現は本質的に矛盾する面が多く，一つの仕様で両者を同時に実現することは困難である．

そこでμITRON4.0仕様では，仕様全体としては「弱い標準化」の方針を維持しつつ，ソフトウェアの移植性を向上させるために，標準的な機能セットとその仕様を厳格に定めるというアプローチを採用している．この標準的な機能セットを「スタンダードプロファイル」と呼ぶ．標準的な機能セットを定めるにあたっては，μITRON仕様カーネルの応用分野としては大きめのシステムを想定する．これは一般に，システムの規模が大きいほど，ソフトウェアの移植性が重視されるためである．

スタンダードプロファイルを定義することにより，ソフトウェア部品などの移植性を重視するソフトウェアはスタンダードプロファイルの機能のみを用いて構築することを推奨し，逆にソフトウェアの移植性を重視する分野向けのカーネルはスタンダードプロファイルに準拠して実装することを推奨することになる．

さらに，スタンダードプロファイル内でも，考えられる限り，ソフトウェアの移植性を向上させるとともに，スケーラビリティも実現できるような仕様としている．具体的には，小さいオーバヘッドで実現できる範囲で，割込みハンドラの移植性が向上するような仕組みを導入している．例えば従来のμITRON仕様では，割込みハンドラの中でより優先度の高い割込みハンドラが多重起動されるのを禁止するための，移植性を確保できる方法が用意されていなかったが，μITRON4.0仕様ではこれを可能にしている．

スケーラビリティの実現に関しては，従来のμITRON仕様と同様，サービスコールの単機能化などの方針により，豊富な機能を用意した上でライブラリリンクの機構で必要のない機能がリンクされないような工夫を行っている．さらに，ライブラリリンクの機構だけでは必要な機能だけをリンクすることが難しい場面では，カーネルではより複雑な機能を実現するのに必要なプリミティブのみを提供するという方針を採っている．これにより，カーネルに改造を加えず複雑な機能を実現することを可能にする一方，複雑な機能を必要としないアプリケーションでのオーバヘッドを最小限にすることができる．

スタンダードプロファイルが想定するシステムイメージは，具体的には次のようなものである．

- ハイエンドの16ビットないしは32ビットプロセッサを使用．
- カーネルのプログラムサイズが 10 〜 20KB 程度（すべての機能を使った場合）．
- システム全体が一つのモジュールにリンクされる．
- カーネルオブジェクトは静的に生成される．

システム全体が一つのモジュールにリンクされることから，サービスコールはサブルーチンコールで呼び出すことになる．また，プロテクションの機構は持たない．

スタンダードプロファイルでサポートすべき機能には，μITRON3.0仕様のレベルSのすべての機能（一部の機能は仕様を変更または拡張した），一部のレベルEの機能（タイムアウト付きのサービスコール，固定長メモリプール，周期ハンドラなど；周期ハンドラは仕様を整理した），新たに導入したいくつかの機能（タスク例外処理機能，データキュー，システム状態参照機能など）が含まれている．また，後述するオブジェクトの生成情報を記述するための静的APIもサポートされる．

1.4.4 より広いスケーラビリティの実現

前節で述べた通り，μITRON4.0仕様全体としては「弱い標準化」の方針を維持し，従来の仕様以上に広範なスケーラビリティの実現を狙っている．

まず，最小の機能セットを，従来のμITRON仕様よりもさらにコンパクトに実現可能なものとし，従来にも増して小規模なシステムへ適用できるものとしている．具体的には，従来の仕様では必須としていた待ち状態のサポートを必須とせず，それに代えて休止状態のサポートを必須としている．待ち状態を持たないカーネルでは，すべてのタスクを同一のスタック空間を用いて動作させることが可能で，スタックのためのメモリ領域の削減や，タスク切替えオーバヘッドの削減を図ることができる．

逆にスタンダードプロファイルを越える要求に対応するために，豊富な拡張機能を定義しており，μITRON4.0仕様のフルセットは従来のμITRON仕様のフルセットよりも豊富な機能を持っている．具体的には，μITRON3.0仕様の機能の内，接続機能を除くほとんどすべての機能を持つことに加えて，μITRON4.0仕様に新たに導入した機能として，スタンダードプロファイルに含まれる新機能（タスク例外処理機能，データキュー，システム状態参照機能など），ID番号の自動割付けを行うオブジェクト生成機能，移植性を確保して割込み処理を記述するための割込みサービスルーチンの機能，優先度継承／上限プロトコルをサポートするミューテックス，タスクが与えられたプロセッサ時間を使い切ったことを検出するオーバランハンドラなどが含まれている．μITRON4.0仕様のフルセットは，ITRON2仕様のフルセットとの対比においても，それほど遜色のない機能を持っているということができる．

また，スタンダードプロファイルに加えて，自動車制御用プロファイルを定義している．自動車制御用プロファイルは，文字どおり自動車制御応用への適用を狙ったものであるが，スタンダードプロファイルが対象としているよりも小規模なシステムに対して，ソフトウェアの移植性を向上させるための機能セットという位置付けを持っている．具体的には，スタンダードプロファイルと比較して，タイムアウト付きのサービスコール，強制待ち状態，タスク例外処理機能，メールボックス，固定長メモリプールなどがサポートの必要のない機能

となっている．逆に，自動車制御用プロファイル独自の機能として，待ち状態に入ることができないタスク（これを制約タスクと呼ぶ）がある．待ち状態に入らないことから，同じ優先度を持つ制約タスクでスタック領域を共有することが可能となり，メモリ使用量を削減することができる．待ち状態に入るサービスコールを呼んだ時にエラーとなることに依存しない限りは，制約タスクを通常のタスクに置き換えてもシステムの振舞いは変化せず，その意味では，制約タスクという独自の機能を持つにもかかわらず，自動車制御用プロファイルはスタンダードプロファイルに対する下位互換性を有している．

図1-2は，μITRON4.0仕様の機能レベルを，μITRON3.0仕様との関係で図示したものである．μITRON4.0仕様は，従来のμITRON仕様に比べて，より小規模のシステムへも，より大規模なシステムへも適用できるような仕様となっているということができる．

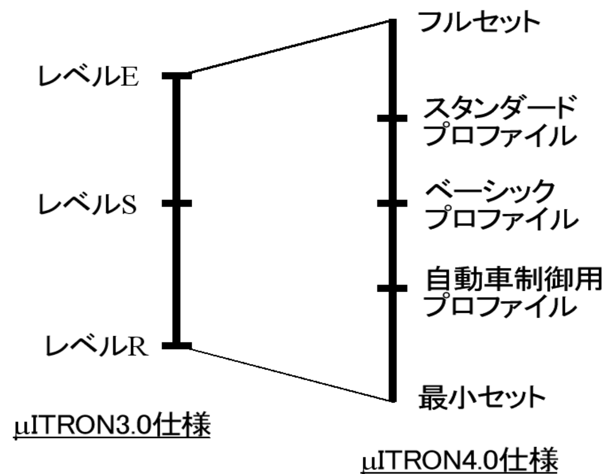


図1-2. μITRON3.0仕様とμITRON4.0仕様の機能レベル

Ver.4.03において，新たにベーシックプロファイルを規定した．これは，μITRON3.0/μITRON4.0/T-Kernelにおいて同等の機能を持つサービスコールを規定したもので，上記の複数のカーネル間でアプリケーションの移植がより容易になることを目的としている．

1.4.5 μITRON4.0仕様における新機能

前述したように，μITRON4.0仕様には従来のμITRON仕様が持っていなかった新機能が数多く導入されている．以下では，μITRON4.0仕様における新機能を簡単に紹介する．

例外処理のための機能

μITRON4.0仕様では，従来のμITRON仕様では実装依存として定義していなかった例外処理の枠組みを定義している．

プロセッサが何らかの例外条件を検出すると，プロセッサはCPU例外ハンドラ

を起動する．CPU例外ハンドラは，例外の種類毎にアプリケーションで定義することができる．CPU例外ハンドラはシステム全体で共通であるため，CPU例外ハンドラ内で例外が発生したコンテキストや状況を調べ，必要であれば例外が発生したタスクに処理を任せることができる．タスクに処理を任せるために用意したのが，タスク例外処理機能である．

タスク例外処理機能は，UNIXのシグナル機能を簡略化したような機能で，ITRON2仕様の強制例外に類似の機能である．タスク例外処理機能の典型的な用途として，次のようなものが挙げられる．

- ゼロ除算などのCPU例外をタスクに伝える．
- 他のタスクに終了要求を出す．
- タスクにデッドラインが来たことを通知する．

µITRON4.0仕様が例外処理のために定義している機能は，より複雑な例外処理機能を実現する場合にそのプリミティブとして使うことができるよう配慮して設計されている．

データキュー

データキューは，1ワードのデータをメッセージとして通信するための機構である．µITRON3.0仕様では，メールボックスを実装するために，リンクリストを使う方法とリングバッファを使う方法のいずれを用いてもよいものとしていた．それに対してµITRON4.0仕様では，ソフトウェアの移植性を向上させるために，メールボックスの実装をリンクリストを使う方法に限定すると同時に，リングバッファを使って実装したメールボックスと同等のデータキューを，別のオブジェクトとして規定することとした．

データキューは，自動車制御応用において強い要求があったことから，当初自動車制御用プロファイル独自の機能として導入する方向で検討したが，データキューが他の応用分野においても有用であること，データキューを使わないアプリケーションにおいてはそのためのプログラムがリンクされないように実装できることから，スタンダードプロファイルに含めることとなった．

システム状態参照機能

他で開発されるアプリケーションから呼び出されることを前提としてソフトウェア部品を構築する場合には，ソフトウェア部品の各サービスルーチンが，どのようなコンテキストから呼ばれても対応できることが求められる．ところがµITRON3.0仕様では，現在のシステム状態を調べるには，レベルEの機能であるref_sysを使うしか方法がなかった．ref_sysをサポートしていないカーネル製品も多く，サポートしている場合でも，他の情報も一緒に参照できるためにオーバヘッドが大きくなるという問題があった．

そこでµITRON4.0仕様には，現在のシステム状態を小さいオーバヘッドで参照する機能として，5つのサービスコール（sns_*yyy*）を導入した．これらのサービスコールはいかなるコンテキストからも呼び出すことができ，いずれもブー

ル値を返す（エラーを返すことはない）．これらのサービスコールを用いると，例えば，待ち状態に入るサービスコールを呼べる状態にあるかどうかを，小さいオーバヘッドで調べることができる．

また，これらのサービスコールを用いると，排他制御が必要な処理を行うために，一時的にCPUロック状態（またはディスパッチ禁止状態）にし，処理が終わった後で元の状態に戻すことが容易にできる．これに関連して，μITRON3.0仕様では，ディスパッチ禁止状態でloc_cpuを呼び出してCPUロック状態にすると元の状態に戻す方法がなかったが，μITRON4.0仕様ではディスパッチ禁止状態とCPUロック状態を独立な状態とすることで，この問題も解決している．

ID番号の自動割付けを行うオブジェクト生成機能

μITRON3.0仕様では，オブジェクトを動的に生成する場合には，生成するオブジェクトのID番号をアプリケーションで指定しなければならなかったが，大規模なシステムにおいては，使っていないID番号を管理するのが面倒になるという問題があった．そこでμITRON4.0仕様では，生成するオブジェクトのID番号をアプリケーションで指定するのではなく，カーネルが割り付けたID番号を用いてオブジェクトを生成するサービスコールを新たに導入した．割り付けられたID番号は，サービスコールの返値として，アプリケーションに返される．

割込みサービスルーチン

割込み処理のアーキテクチャは，プロセッサやシステムにより違いが大きく，標準化の難しい部分である．従来のμITRON仕様では，割込みハンドラの記述方法はプロセッサやシステムに最適に定めるべき事項と考えて標準化していなかったが，デバイスドライバの移植性を向上させるためには，移植性の高い割込み処理の記述方法が求められている．

そこでμITRON4.0仕様では，従来の仕様と同様の割込みハンドラに加えて，移植性を確保して割込み処理を記述するための割込みサービスルーチンの機能を新たに導入した．割込みサービスルーチンは，割込みの発生源となるデバイスのみに依存して記述できることを目指して，仕様が設計されている．

ミューテックス

厳しい時間制約を持つリアルタイムシステムを構築する場合には，優先度逆転現象を防ぐための仕組みとして，優先度継承プロトコルや優先度上限プロトコルが必要になる．ミューテックスは，優先度継承プロトコルと優先度上限プロトコルをサポートする排他制御のための機構として，μITRON4.0仕様で新たに導入した機能である．ミューテックスの導入にあたっては，POSIXのリアルタイム拡張におけるミューテックスの機能を参考にした．

オーバランハンドラ

厳しい時間制約を持つリアルタイムシステムを構築する場合に必要となるもう一つの機能として，µITRON4.0仕様では，タスクが与えられたプロセッサ時間を使い切ったことを検出するためのオーバランハンドラの機能を導入した．

システムが時間制約を満たしていないことを検出する最も単純な方法は，処理がデッドラインまでに終了しないことを検出する方法である（処理がデッドラインまでに終了しないことの検出は，アラームハンドラなどを使うことで実現できる）．ところがこの方法には，優先度の高いタスクがデッドラインまで実行を続けた場合，優先度の低いタスクは連鎖的にデッドラインを守れなくなるという問題がある．この問題を解決するには，タスクが与えられたプロセッサ時間を使い切ったことを検出するための機構が必要になる．

コンフィギュレーション方法の標準化

スタンダードプロファイルでは，タスクやセマフォなどのカーネルオブジェクトは静的に生成されることを想定している．そのため，スタンダードプロファイルに準拠したカーネル上に実現されたアプリケーションソフトウェアを，他のスタンダードプロファイル準拠のカーネル上に移植するためには，アプリケーションプログラムそのものを移植することに加えて，カーネルオブジェクトの生成情報を新しいカーネルに移行することも必要になる．

従来のµITRON仕様では，カーネルオブジェクトの生成情報を記述する方法は標準化していなかったため，生成情報の記述方法はカーネル製品毎にかなり異なったものとなっていた．例えば，オブジェクトの生成情報を直接C言語の構造体の初期化データの形で記述させる製品もあれば，GUIを持ったコンフィギュレータを備えている製品もある．このような状況では，大規模なアプリケーションを他のカーネルに移植する際に，生成情報の移植工数が無視できなくなる．ここで，書き直し作業そのものの工数はそれほど大きくなくても，製品毎に異なる記述方法を学習するのにかかる時間も工数に含めて考えるべきであることに注意して欲しい．

そこでµITRON4.0仕様では，オブジェクトの生成情報の記述方法と，それを基にカーネルやソフトウェア部品をコンフィギュレーションする方法を標準化している．システムコンフィギュレーションファイル中での，オブジェクトの生成情報の記述方法を，静的APIと呼ぶ．静的APIの名称は，同じ機能を持つサービスコールの名称を大文字で記述したものとしており，静的APIとサービスコールでパラメータを一致させている（ただし，パケットへのポインタの代わりに，パケットの各要素の値を“{”と“}”の中に列挙する）．これは，片方を覚えればもう片方も自然に覚えられるという教育的効果を狙ったものである．

また，静的APIを処理するコンフィギュレータは，ID番号が与えられないオブジェクトにID番号を自動的に割り付ける機能を持たなければならない．これにより，別々に開発されたモジュールを組み上げてアプリケーションを構築する場合にも，オブジェクトのID番号割付けの管理を省略することができ，大

規模なアプリケーション開発においては特に有効な機能である．

μITRON4.0仕様のコンフィギュレーション方法のもう一つの特徴は，一つのシステムコンフィギュレーションファイル中に，カーネルの静的APIに加えて，ソフトウェア部品のための静的APIを混在して記述することを可能にしている点である．システムコンフィギュレーションファイルを，ソフトウェア部品とカーネルのコンフィギュレータに順に処理されることによって，ソフトウェア部品のコンフィギュレーションによりソフトウェア部品が必要とするカーネルオブジェクトが変わるような複雑な状況にも対応することができる．

以上で紹介した新機能に加えて，μITRON4.0仕様では，ソフトウェアの移植性を向上させるために，個々のサービスコールの機能の中でμITRON3.0仕様では実装依存としていた事項や曖昧になっていた事項について，新たに規定するなどの方法で実装依存性を削減している．さらに，用語や概念の再整理，パラメータのデータ型の再整理，エラーコードの再整理，サービスコールの機能コードの再割付け，カーネルの構成を取り出すための定数やマクロの標準化，システムの初期化手順の標準化など，μITRON3.0仕様に対して数々の改善を加えている．

1.4.6 μITRON4.0仕様公開後の標準化活動

ITRONデバッギングインタフェース仕様

デバッグツールがITRON4.0仕様カーネルをサポートするためのインタフェースを規定したITRONデバッギングインタフェース仕様Ver.1.00.00を2001年5月に公開した．

μITRON4.0仕様保護機能拡張（μITRON4.0/PX仕様）

μITRON4.0仕様に保護機能を追加するための検討を行ない，2002年7月にμITRON4.0仕様保護機能拡張（μITRON4.0/PX仕様）Ver.1.00.00を公開した．

ITRON TCP/IP API仕様Ver.2.00

ITRON TCP/IP API仕様をμITRON4.0仕様，T-Kernel，IPv6に対応させるための検討を行い，ITRON TCP/IP API仕様Ver.2.00として2006年7月に公開した．

第2章 ITRON仕様共通規定

ITRON仕様共通規定は，µITRON4.0仕様ならびにそれと整合するように標準化されるソフトウェア部品仕様（この規定の中では，これらをITRON仕様と総称する）に共通する規定を定めるものである．この規定の中で，「カーネル仕様」とはµITRON4.0仕様を，「スタンダードプロファイル」とはµITRON4.0仕様のスタンダードプロファイルを指す．

【補足説明】

この章は，ソフトウェア部品仕様にも共通に適用できるように記述しているが，µITRON4.0仕様の理解を容易にするために，必要に応じてµITRON4.0仕様とそのスタンダードプロファイルに固有の事項にも言及する．

2.1 ITRON仕様共通概念

2.1.1 用語の意味

ITRON仕様において用いる用語の意味を次の通りに定める．

- 「実装定義」とは，ITRON仕様で定める機能仕様の中で，ITRON仕様では標準化せず，実装毎に規定すべき事項であることを示す．実装について説明する製品マニュアルなどで，実装における規定を示さなければならない．アプリケーションプログラムの中で，実装定義の事項に依存している部分は，移植性が確保されない．
- 「実装依存」とは，ITRON仕様で定める機能仕様の中で，実装ないしはシステムの動作条件によって振舞いが変わる事項であることを示す．アプリケーションが実装依存の事項に依存している場合の振舞いは，ITRON仕様上は保証されない．
- 「未定義」とは，そのような状況における振舞いに関して何も保証されないことを示す（すなわち，システムダウンを引き起こすかもしれない）．仕様に定めのない事項は，一般的には未定義である．アプリケーションが未定義の状況を作り出した場合の振舞いは，ITRON仕様上は保証されない．
- 「実装独自」とは，ITRON仕様で定める範囲外の機能仕様を，実装において規定したものであることを示す．

【補足説明】

実装における規定は，規定の内容が実装毎に一通りに固定されている必要はなく，カーネルやソフトウェア部品の構成により規定の内容が変わってもよい．構成により規定の内容が変わる場合には，実装について説明する製品マニュアルなどで，カーネルやソフトウェア部品の構成方法と，それぞれの構成における規定の内容を示さなければならない．

2.1.2 APIの構成要素

API（Application Program Interface）とは，アプリケーションプログラムがカーネルやソフトウェア部品を使う場合のインタフェースのことをいう．APIは次の要素で構成される．

(A) サービスコール

アプリケーションプログラムからカーネルまたはソフトウェア部品を呼び出すインタフェースをサービスコールと呼ぶ．ITRON仕様では，サービスコールの名称，機能，パラメータとリターンパラメータの種類・順序・名称・データ型を標準化する．

C言語APIでは，サービスコールは関数呼出しの形で定義されるが，同じ機能を持つならプリプロセッサマクロなどの形で実現してもよい．

【μITRON3.0仕様との相違】

μITRON3.0仕様ではシステムコールと呼んでいたが，ソフトウェア部品への対応を考慮して，サービスコールと呼ぶことにした．カーネルのサービスコールをシステムコールと呼んでも良い．

(B) コールバック

ソフトウェア部品からアプリケーションプログラムが登録したルーチンを呼び出すインタフェースをコールバック，呼び出されるルーチンをコールバックルーチンと呼ぶ．ITRON仕様では，コールバックルーチンの名称，機能，パラメータとリターンパラメータの種類・順序・名称・データ型を標準化する．

コールバックルーチンが実行されるコンテキストは，それぞれのソフトウェア部品仕様で規定する．

【補足説明】

カーネル仕様ではコールバックを用いていない．

(C) 静的API

システムコンフィギュレーションファイル中に記述し，カーネルやソフトウェア部品の構成を決定したり，オブジェクトの初期状態を定義するためのインタフェースを，静的APIと呼ぶ．ITRON仕様では，静的APIの名称，機能，パラメータの種類・順序・名称・データ型を標準化する．

オブジェクトを登録するサービスコールなどに対して，それに対応する静的APIを規定する．サービスコールに対応する静的APIは，システムコンフィギュレーションファイル中に記述された順序で，それぞれに対応するサービスコールをシステム初期化時に実行するのと等価の機能を持つ．また，サービスコールに対応しない静的API，カーネルやソフトウェア部品で共通に利用するITRON仕様共通静的APIもある．

(D) パラメータとリターンパラメータ

サービスコール・コールバックルーチン・静的APIに渡すデータをパラメータ，サービスコールやコールバックルーチンが返すデータをリターンパラメータと呼ぶ．ITRON仕様では，パラメータとリターンパラメータの名称とデータ型を標準化する．

C言語APIでは，関数の返値以外のリターンパラメータは，リターンパラメータを入れる領域へのポインタをC言語の関数の引数（以下，単に引数と呼ぶ）として渡すか，複数のパラメータまたはリターンパラメータを含む構造体（これをパケットと呼ぶ）に入れて返すことで実現される．この場合，リターンパラメータを入れる領域へのポインタはパラメータとは考えないこととし，パラメータとしてはリストアップしない．それに対して，パケットへのポインタは，パラメータとしてリストアップする．

C言語APIで，リターンパラメータの名称の前に"p_"を付加した名称（ただし，リターンパラメータの名称の先頭が"pk_"の場合は，それを"ppk_"に置き換えた名称）の引数があれば，その引数はリターンパラメータを入れる領域へのポインタをあらわす．また，パラメータのサイズが大きいなどの理由で，パラメータを入れた領域へのポインタを引数として渡す場合も同様である．

サービスコールに対してリターンパラメータ（またはパラメータ）を入れる領域やパケットへのポインタを渡した場合，そのサービスコールの処理が完了した後は，アプリケーションはそれらの領域を別の目的に使うことができるのが原則である．また，コールバックに対してリターンパラメータ（またはパラメータ）を入れる領域やパケットへのポインタを渡した場合，そのコールバックの処理が完了した後は，ソフトウェア部品はそれらの領域を別の目的に使うことができるのが原則である．これらの原則の例外となるケースについては，サービスコールやコールバックの機能説明でその旨を明示する．

【仕様決定の理由】

関数の引数と返値の名称は，カーネルやソフトウェア部品のAPIを直接的に定めるものではないが，仕様書や製品マニュアルなどの中で頻繁に使われることから，ITRON仕様で標準を定める．

(E) データ型

ITRON仕様では，パラメータやリターンパラメータなどのデータ型の名称と意味を標準化する．データ型によっては，その型定義をITRON仕様で標準化する場合もある．

ITRON仕様でビット数が規定されていないデータ型に対して，C言語の型のビット数よりも少ない有効ビット数ないしは型で表現できる範囲よりも狭い有効範囲を，実装定義に定めることができる．

(F) 定数

ITRON仕様では，パラメータやリターンパラメータなどに用いる定数，サービ

スコールの機能コードとして用いる定数の名称・意味・値を標準化する．C言語APIでは，定数はプリプロセッサマクロとして定義する．

(G) マクロ

カーネルやソフトウェア部品を呼び出さず，システムの状態に依存せずに値の変換などを行うためのインタフェースをマクロと呼ぶ．ITRON仕様では，マクロの名称と意味を標準化する．C言語APIでは，マクロはプリプロセッサマクロとして定義する．

(H) ヘッダファイル

カーネルやソフトウェア部品毎に，サービスコールの宣言と，データ型・定数・マクロの定義などを含むヘッダファイルを一つまたは複数用意する．ITRON仕様では，ヘッダファイルの名称を標準化する．また，複数のヘッダファイルを用意する場合には，どのヘッダファイルにどの宣言および定義が含まれるかを標準化する．

また，ITRON仕様共通定義で規定されるデータ型，定数，マクロの定義などを含むヘッダファイルを用意し，カーネルならびにソフトウェア部品毎に用意するヘッダファイルからインクルードする．

オブジェクトのID番号などの自動割付けを行うコンフィギュレータは，自動割付けを行った結果を，自動割付け結果ヘッダファイルに生成する．ITRON仕様では，自動割付け結果ヘッダファイルの名称を標準化する．

ITRON仕様で標準化されたヘッダファイルを，複数のファイルに分割して実装してもよい．また，同じヘッダファイルを複数回インクルードしてもエラーとならないようにしなければならない．

【補足説明】

同じヘッダファイルを複数回インクルードしてもエラーとならないようにするためには，ヘッダファイルの先頭で特定の識別子（ここでは“_KERNEL_H_”とする）をプリプロセッサマクロとして定義（“#define _KERNEL_H_”）し，ヘッダファイル全体を“#ifndef _KERNEL_H_”と“#endif”で囲めばよい．

2.1.3 オブジェクトのID番号とオブジェクト番号

カーネルやソフトウェア部品が操作対象とする資源を，オブジェクトと総称する．オブジェクトは，その種類毎に，番号によって識別する．オブジェクトを識別する番号がカーネルやソフトウェア部品のAPIに閉じており，アプリケーションが自由に番号を割り付けることができる場合に，その識別のための番号をID番号と呼ぶ．それに対して，カーネルやソフトウェア部品の内部や外部からの条件によってオブジェクトを識別する番号が定まる場合に，それをオブジェクト番号と呼ぶ．

ID番号で識別されるオブジェクトは，アプリケーションによって生成することで，カーネルやソフトウェア部品に登録される．それに対して，オブジェクト

番号で識別されるオブジェクトは，カーネルやソフトウェア部品の内部や外部からの条件で意味が与えられるため，生成の対象とはならない．オブジェクト番号で識別されるオブジェクトをカーネルやソフトウェア部品に登録することを，オブジェクトの定義と呼ぶ．

オブジェクトを種類分けしない場合には，オブジェクトのID番号には1から連続した正の値を用いる．保護機能を入れるなどの目的で，オブジェクトをユーザオブジェクトとシステムオブジェクトに分類する場合には，ユーザオブジェクトには1から連続した正のID番号，システムオブジェクトには(–5)から小さい方へ連続した負のID番号を用いる．この場合，ID番号の自動割付けの対象となるのは，正のID番号を持ったユーザオブジェクトのみである．(–4)～0は，特別な意味に用いるために予約されている．

【スタンダードプロファイル】

スタンダードプロファイルでは，オブジェクトを種類分けする必要はなく，負の値のID番号をサポートする必要はない．少なくとも1～255の範囲の正の値のID番号をサポートしなければならない．

【補足説明】

オブジェクト番号で識別されるオブジェクトの例として，割込みハンドラやランデブがある．割込みハンドラ番号はハードウェア条件から，ランデブ番号はカーネル内部の条件から番号が定まり，アプリケーションが自由に番号を割り付けることはできない．

2.1.4 優先度

優先度は，タスクやメッセージなどの処理順序を制御するために，アプリケーションによって与えられるパラメータである．優先度には，1から連続した正の値を用い，値が小さいほど優先して処理される（優先度が高い）．

【スタンダードプロファイル】

スタンダードプロファイルでは，少なくとも1～16の16段階のタスク優先度をサポートしなければならない．また，少なくともタスク優先度以上の段階数のメッセージ優先度をサポートしなければならない．

【µITRON3.0仕様との相違】

µITRON3.0仕様では，システムのための優先度として負の値が使えることになっていたが，使い途が少ないため正の値に限定した．実装独自に負の値の優先度を使えるようにすることは許される．また，少なくとも1～8の8段階以上のタスク優先度をサポートしなければならないものとしていたが，µITRON4.0仕様全体では最低限の段階数を規定せず，スタンダードプロファイルにおいて1～16の16段階以上とした．

2.1.5 機能コード

機能コードは，サービスコールを識別するために，各サービスコールに割り付けられる番号である．機能コードは，サービスコールをソフトウェア割込みで呼び出す場合などに利用するもので，サービスコールをサブルーチンコールで呼び出す場合には用いる必要はない．

ITRON仕様のカーネルおよびソフトウェア部品のサービスコールには，それぞれ異なる負の値の機能コードを割り付ける．ただし，(–4) ～ 0 は，特別な意味に用いるために予約されている．正の値の機能コードは，拡張サービスコールをあらわす．

2.1.6 サービスコールの返値とエラーコード

サービスコールの返値は原則として符号付きの整数で，エラーが発生した場合には負の値のエラーコード，処理を正常に終了した場合はE_OK（＝0）または正の値とする．正常終了した場合の返値の意味はサービスコール毎に規定する．この原則の例外として，真偽値（BOOL型）を返すサービスコールと，呼び出されるとリターンすることのないサービスコールがある．リターンすることのないサービスコールは，C言語APIでは返値を持たないもの（すなわちvoid型の関数）として宣言する．

エラーコードは，下位8ビットのメインエラーコードと，それを除いた上位ビットのサブエラーコードで構成される．メインエラーコード，サブエラーコード共に負の値とする（サブエラーコードの値とは，エラーコードの値を符号拡張して8ビット右シフトしたものである）．したがって，それらを組み合わせたエラーコードも負の値となる．メインエラーコードの名称，意味，値は，カーネルおよびソフトウェア部品で共通とし，ITRON仕様共通定義で規定する．メインエラーコードは，検出の必要性や発生状況などにより，エラークラスに分類される．

ITRON仕様の各サービスコールの機能説明においては，サービスコールが返すメインエラーコードのみを記述するのを原則とし，サブエラーコードは実装定義とする．ただし，ソフトウェア部品仕様において，サブエラーコードについても規定する場合もある．サービスコールの機能説明などに「E_*XXXXX*エラーを返す」ないしは「E_*XXXXX*エラーとなる」という記述があった場合，メインエラーコードをE_*XXXXX*とするエラーコードを返すことを意味する．

警告クラスのメインエラーコードである場合を除いては，サービスコールがエラーコードを返した場合には，サービスコールを呼び出したことによる副作用はない（言い換えると，サービスコールを呼び出したことで，システムの状態は変化していない）のが原則である．ただし，サービスコールの機能上，サービスコールを呼び出したことによる副作用を防げない場合は例外とし，サービスコールの機能説明でその旨を明示する．

ITRON 仕様を実装する場合には，オーバヘッドを削減するために，一部のエ

ラーの検出を省略することができる．検出を省略することができるエラーは，メインエラーコードが属するエラークラスによって定めるのを原則とし，エラークラス毎に検出を省略することができる旨を明示する．この原則の例外となるケースについては，サービスコールの機能説明でその旨を明示する．エラーの検出を省略したことにより，本来検出すべきエラーを検出できなかった場合の振舞いは未定義である．

次のメインエラーコードは，多く（または，ほとんどすべて）のサービスコールで発生する可能性があるため，サービスコールが返すメインエラーコードとしてサービスコール毎には記述しないのを原則とする．

E_SYS	システムエラー
E_NOSPT	未サポート機能
E_RSFN	予約機能コード
E_CTX	コンテキストエラー
E_MACV	メモリアクセス違反
E_OACV	オブジェクトアクセス違反
E_NOMEM	メモリ不足

ただし，これらのエラーが発生する理由がサービスコール独特のものである場合などには，この原則にかかわらず，サービスコール毎に記述する．

サービスコールが複数のエラーを検出すべき状況で，どのエラーを示すエラーコードを返すかは，実装依存とする．

【補足説明】

E_OK（＝0）は正常終了を示す返値であり，エラーコードではない．ただし，便宜上，サービスコールが返すエラーコードとして記述する場合がある．

サービスコールでエラーが発生したかを，返値の下位8ビットが負の値であるかで判断する方法は正しくない．これは，サービスコールの処理が正常に終了し，返値が正の値であった場合でも，返値の下位8ビットが負の値である可能性があるためである．

【μITRON3.0仕様との相違】

エラーコードをメインエラーコードとサブエラーコードから構成されるものとし，カーネルとソフトウェア部品でメインエラーコードを共通化することとした．サブエラーコードは，エラーが発生した原因をより細かく報告することを目的としている．例えば，メインエラーコードがE_PAR（パラメータエラー）の場合に，どのパラメータの値が不正であったかを示すためにサブエラーコードを使うことができる．また，E_OKはエラーコードではないと規定した．

一部のエラーの検出を省略することができることを明示し，どのエラーの検出を省略できるかをエラークラス毎に規定することとした．また，サービスコール毎に記述しないメインエラーコードを見直した．

μITRON3.0仕様では，サービスコールの返値に関する原則として，正の値を返す場合を規定していたが，実際に正の値を返すカーネルのサービスコールはな

かった．μITRON4.0仕様では，カーネルのサービスコールに，正の値を返すものがある．また，真偽値を返すサービスコールを新たに導入した．

2.1.7 オブジェクト属性と拡張情報

ID番号で識別されるオブジェクトは，原則としてオブジェクト属性を持つ．オブジェクト番号で識別されるオブジェクトが，オブジェクト属性を持つ場合もある．オブジェクト属性は，オブジェクト登録時に指定し，オブジェクトの細かな動作の違いやオブジェクトの初期状態を定める．オブジェクト属性にTA_*XXXXX*が指定されている場合，そのオブジェクトを「TA_*XXXXX*属性のオブジェクト」と呼ぶ．オブジェクト登録後にオブジェクト属性を読み出すインタフェースは，一般には用意されない．

各オブジェクトのオブジェクト属性に指定できる値とその意味は，オブジェクトを登録するサービスコールまたは静的APIの機能説明で規定する．特に指定すべきオブジェクト属性がない場合には，TA_NULL（＝0）を指定する．

実行主体となるオブジェクトは，拡張情報を持つ場合がある．拡張情報はオブジェクト登録時に指定し，オブジェクトが実行を始める時にパラメータとして渡される情報で，カーネルやソフトウェア部品自身の動作には影響を与えない．オブジェクトを指定して拡張情報を読み出すインタフェースは，一般には用意されない．

【補足説明】

実行主体となるオブジェクトで拡張情報を持つものの例として，タスク，周期ハンドラなどのタイムイベントハンドラ，割込みサービスルーチンがある．

【μITRON3.0仕様との相違】

μITRON3.0仕様では, ID番号で識別されるオブジェクトは原則として拡張情報を持っていたが，必要なものだけに限定することとした．関連して，拡張情報はオブジェクトが実行を始める時にパラメータとして渡されるものとし，オブジェクトの状態参照サービスコールでは読み出せないことを原則とした．

2.1.8 タイムアウトとノンブロッキング

待ち状態に入る可能性のあるサービスコールには，必要に応じて，タイムアウトとノンブロッキングの機能を持たせる．

タイムアウトは，指定された時間が経過しても処理が完了しない場合に，処理をキャンセルしてサービスコールからリターンするものである（この時，サービスコールはE_TMOUTエラーを返す）．そのため，「サービスコールがエラーコードを返した場合には，サービスコールを呼び出したことによる副作用はない」という原則より，タイムアウトした場合には，サービスコールを呼び出したことで，システムの状態は変化していないのが原則である．ただし，サービスコールの機能上，処理のキャンセル時に元の状態に戻せない場合は例外と

し，サービスコールの機能説明でその旨を明示する．

タイムアウト時間を0に設定すると，サービスコールの中で待ち状態に入るべき状況になっても，待ち状態には入らない．そのため，タイムアウト時間を0としたサービスコール呼出しでは，待ち状態に入る可能性がない．タイムアウト時間を0としたサービスコール呼出しを，ポーリングと呼ぶ．すなわち，ポーリングを行うサービスコールでは，待ち状態に入る可能性がない．次に説明するノンブロッキングとは，処理がキャンセルされるか継続されるかの違いがある．

ノンブロッキングは，サービスコールの中で待ち状態に入るべき状況になった場合に，処理を継続したままサービスコールからリターンするものである（この時，サービスコールはE_WBLKエラーを返す）．そのため，サービスコールからリターンしても処理は継続しており，処理が完了した時点（または，処理がキャンセルされた時点）で，何らかの方法でアプリケーションプログラムに処理完了が通知される．サービスコールからリターンした後も処理が継続しているため，リターンパラメータ（またはパラメータ）を入れる領域やパケットは，サービスコールからリターンした後も，処理が完了するまでの間は別の目的に使ってはならないことを原則とする．

サービスコールの中で待ち状態になっている場合，またはノンブロッキング指定による処理が継続している場合，サービスコールによる処理がペンディングされているという．

ITRON仕様の各サービスコールの機能説明では，タイムアウトがない（言い換えると，永久待ちの）場合の振舞いを説明するのを原則とする．サービスコールの機能説明などで「待ち状態に入る」ないしは「待ち状態に移行させる」と記述されている場合でも，タイムアウト時間を指定をした場合には，指定時間経過後に待ち状態が解除され，メインエラーコードをE_TMOUTとしてサービスコールからリターンする．また，ポーリングの場合には，待ち状態に入らずにメインエラーコードをE_TMOUTとしてサービスコールからリターンする．ノンブロッキング指定をした場合には，待ち状態に入らずにメインエラーコードをE_WBLKとしてサービスコールからリターンする．

タイムアウト指定（TMO型）は，正の値でタイムアウト時間，TMO_POL（＝0）でポーリング，TMO_FEVR（＝–1）で永久待ちを指定する．また，サービスコールによっては，TMO_NBLK（＝–2）でノンブロッキングを指定することができる．タイムアウト時間が指定された場合，タイムアウトの処理は，サービスコールが呼び出されてから，指定された以上の時間が経過した後に行うことを保証しなければならない．

【補足説明】

カーネルのサービスコールは，ノンブロッキングの機能を持っていない．ポーリングを行うサービスコールでは待ち状態に入らないため，それを呼び出したタスクの優先順位は変化しない．

一般的な実装においては，タイムアウト時間に1が指定されると，サービスコー

ルが呼び出されてから2回めのタイムティックでタイムアウト処理を行う．タイムアウト時間に0を指定することはできないため（0はTMO_POLに割り付けられている），このような実装では，サービスコールが呼び出された後の最初のタイムティックでタイムアウトすることはない．

2.1.9 相対時間とシステム時刻

イベントの発生する時刻を，サービスコールを呼び出した時刻などからの相対値で指定する場合には，相対時間（RELTIM型）を用いる．相対時間を用いてイベントの発生時刻が指定された場合，イベントの処理は，基準となる時刻から指定された以上の時間が経過した後に行うことを保証しなければならない．

イベントの発生間隔など，イベントの発生する時刻以外を指定する場合にも，相対時間（RELTIM型）を用いる．その場合，指定された相対時間の解釈方法は，それぞれの場合毎に定める．

時刻を絶対値で指定する場合には，システム時刻（SYSTIM型）を用いる．カーネル仕様には現在のシステム時刻を設定する機能が用意されているが，この機能を用いてシステム時刻を変更した場合にも，相対時間を用いて指定されたイベントが発生する実世界の時刻（これを実時刻と呼ぶ）は変化しない．言い換えると，相対時間を用いて指定されたイベントが発生するシステム時刻は変化することになる．

【補足説明】

一般的な実装においては，相対時間に1が指定されると，サービスコールが呼び出されてから2回めのタイムティックでイベント処理を行う．また，相対時間に0が指定されると，サービスコールが呼び出された後の最初のタイムティックでイベント処理を行う．

2.1.10 システムコンフィギュレーションファイル

カーネルやソフトウェア部品の構成やオブジェクトの初期状態を定義するためのファイルを，システムコンフィギュレーションファイルと呼ぶ．システムコンフィギュレーションファイルには，カーネルやソフトウェア部品の静的APIとITRON仕様共通静的API（以下，単に共通静的APIと呼ぶ）に加えて，C言語処理系のプリプロセッサディレクティブを記述することができる．システムコンフィギュレーションファイル中の静的APIを解釈して，カーネルやソフトウェア部品を構成するためのツールを，コンフィギュレータと呼ぶ．

システムコンフィギュレーションファイルの処理手順は次の通りである（図2-1）．システムコンフィギュレーションファイルは，まず，C言語のプリプロセッサに通される．次にソフトウェア部品のコンフィギュレータによって順に処理され，最後にカーネルのコンフィギュレータによって処理される．

ソフトウェア部品のコンフィギュレータは，渡されたファイル中に含まれる自分自身に対する静的APIと共通静的APIを解釈し，自分自身の構成や初期化に

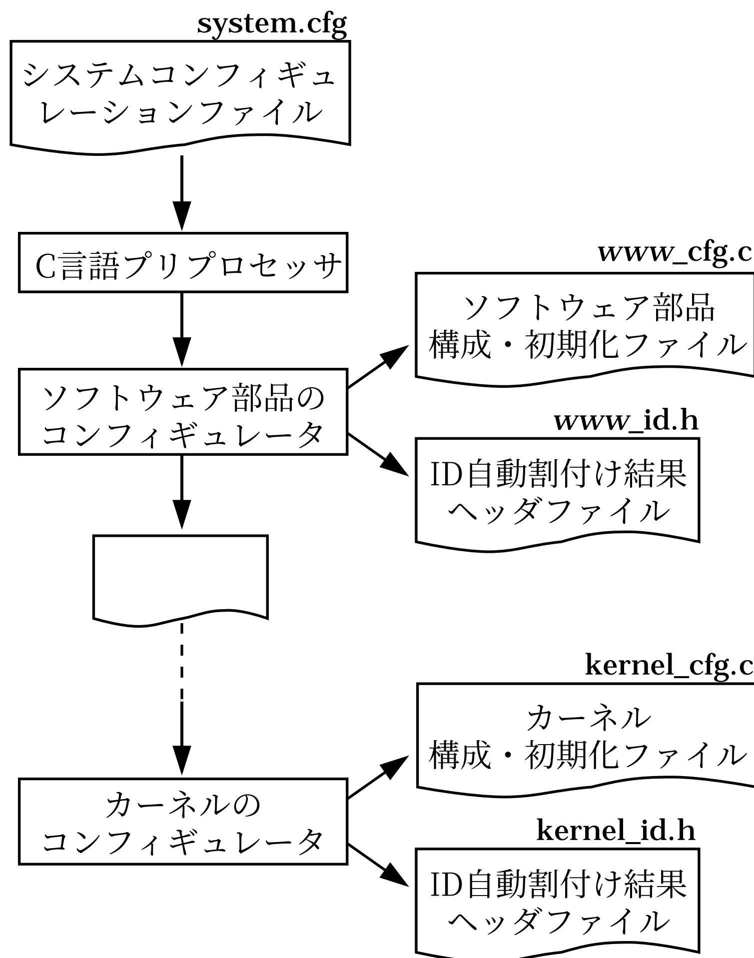


図2-1. システムコンフィギュレーションファイルの処理手順

必要なファイルをC言語のソースファイルの形で，ID自動割付け結果ヘッダファイルをC言語のヘッダファイルの形で生成する．また，渡されたファイルから自分自身に対する静的APIを取り除き，以降のコンフィギュレータに対する静的APIを追加し（必要な場合のみ），次のコンフィギュレータに渡す.

カーネルのコンフィギュレータは，渡されたファイル中のすべての静的APIを解釈し，自分自身の構成や初期化に必要なファイルをC言語のソースファイルの形で，ID自動割付け結果ヘッダファイルをC言語のヘッダファイルの形で生成する．自分自身に対する静的APIまたは共通静的APIとして解釈できない記述が含まれている場合には，エラーを報告する.

カーネルおよびソフトウェア部品のコンフィギュレータは，“#” で始まる行を無視する．ソフトウェア部品のコンフィギュレータは，“#” で始まる行を，そのまま次のコンフィギュレータに渡す.

【補足説明】

ソフトウェア部品のコンフィギュレータが，以降のコンフィギュレータに対する静的APIを追加する場合には，追加する静的APIのパラメータ中に，システテ

ムコンフィギュレーションファイルまたはそこからプリプロセッサディレクティブ（“#include”）を用いてインクルードされるファイル中で定義されたプリプロセッサマクロを用いてはならない．これは，それらのプリプロセッサマクロは，最初にC言語プリプロセッサに通された時点で展開されるためである．

システムコンフィギュレーションファイルの処理手順を，図2-2の例を用いて説明する．なお，ID番号の自動割付けについては2.1.11節を，共通静的APIについては2.3.4節を，それぞれ参照すること．

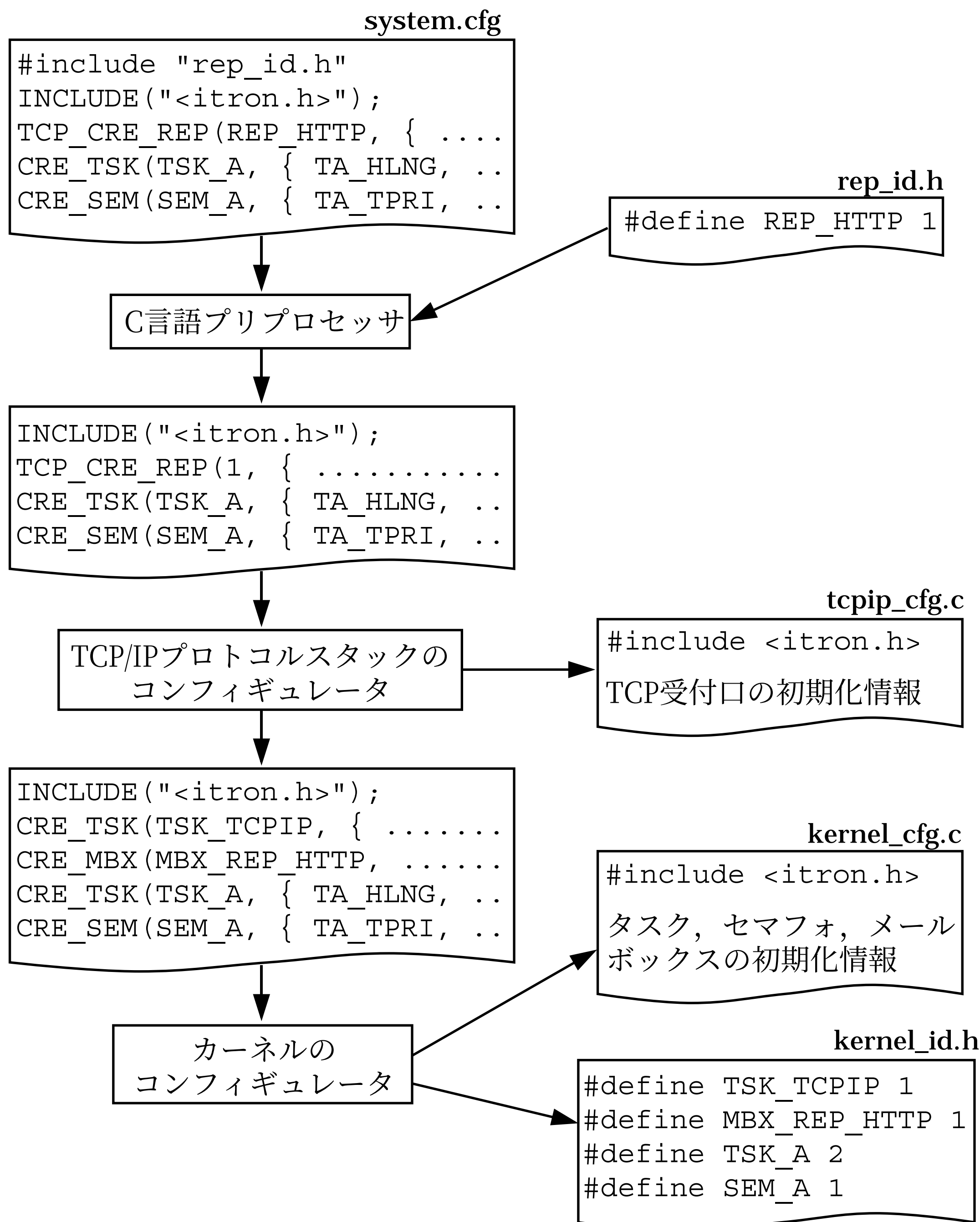


図2-2. システムコンフィギュレーションファイルの処理例

最初に，システムコンフィギュレーションファイルがC言語のプリプロセッサに通されると，プリプロセッサディレクティブ（“#include”）によるインクルード処理が行われ，プリプロセッサマクロ（この例では，REP_HTTP）が展開される．

次に，ソフトウェア部品の一つであるTCP/IPプロトコルスタックのコンフィギュレータは，渡されたファイル中に含まれる自分自身に対する静的API（この例では，TCP_CRE_REP）と共通静的API（INCLUDE）を解釈し，TCP/IPプロトコルスタックの構成や初期化に必要なファイルtcpip_cfg.cを生成する．ここで，tcpip_cfg.cの中の#includeは，共通静的APIのINCLUDEから生成したものである．この例では，解釈された静的APIにID番号自動割付けの対象となる識別子が含まれていないため，ID番号自動割付け結果ファイルは生成しない（空のID番号自動割付け結果ファイルを生成してもよい）．また，TCP/IPプロトコルスタックのコンフィギュレータは，自分自身の構成に必要なカーネルの静的API（この例では，TSK_TCPIPに対するCRE_TSKとMBX_REP_HTTPに対するCRE_MBX）を追加し，カーネルのコンフィギュレータに渡す．

最後にカーネルのコンフィギュレータは，渡されたファイル中に含まれるすべての静的APIを解釈し，カーネルの構成や初期化に必要なファイルkernel_cfg.cを生成する．ここで，kernel_cfg.cの中の#includeは，共通静的APIのINCLUDEから生成したものである．また，静的APIに含まれるID番号自動割付けの対象となる識別子（この例では，TSK_TCPIP，MBX_REP_HTTP，TSK_A，SEM_A）に整数値を割り付け，その結果をID番号自動割付け結果ファイルkernel_id.hとして生成する．

【仕様決定の理由】

システムコンフィギュレーションファイルの処理手順を標準化したのは，カーネルとソフトウェア部品が独立に開発された場合に対応するためである．

システムコンフィギュレーションファイルを最初にC言語プリプロセッサに通すのは，次のようなことが可能になるためである．

- プリプロセッサのインクルードディレクティブを用いて，システムコンフィギュレーションファイルを複数のファイルに分割することができる．例えば，ソフトウェア部品を組み込む場合に，それに必要な静的APIを独立したファイルに記述しておき，そのファイルをシステムコンフィギュレーションファイルからインクルードするといった使い方が考えられる．
- オブジェクトのID番号やオブジェクト番号を，直接整数値で記述する代わりに，整数値に展開されるプリプロセッサマクロを用いて記述することができる．
- システムコンフィギュレーションファイル中にプリプロセッサの条件ディレクティブ（“#ifdef”など）を記述して，カーネルやソフトウェア部品の構成やオブジェクトの初期状態を条件によって変えることができる．

コンフィギュレータに“#”で始まる行を無視させるのは，プリプロセッサがソースファイルなどに関する情報（そのような情報は，“#”で始まる行として

生成されるのが一般的である）を生成する場合に対応するためである．“#” で始まる行を読み込み，エラーメッセージなどの生成に利用することは許される．

2.1.11 静的APIの文法とパラメータ

静的APIの文法は，C言語の関数呼出し式の文法に準ずる．また，サービスコールに対応する静的APIのパラメータは，対応するサービスコールのC言語APIに準ずる．ただし，パラメータにパケットへのポインタがある場合には，パケット中の各要素を “,” で区切り，“{” と “}” で囲んだ形で記述する．

静的APIのパラメータは，記述に使える式によって次の4種類に分類される．

(a) 自動割付け対応整数値パラメータ

整数値（負の数を含む），単一の識別子，またはそれらに展開されるプリプロセッサマクロ（後に述べる制限あり）のみを記述できるパラメータ．ID番号自動割付けの対象となるオブジェクトのID番号などがこれに該当する．

自動割付け対応整数値パラメータに単一の識別子が記述された場合，その静的APIを処理するコンフィギュレータが，識別子に整数値を割り付ける．これを，コンフィギュレータによるID番号の自動割付けと呼ぶ．コンフィギュレータは，各識別子を割り付けた整数値にマクロ定義するプリプロセッサディレクティブ（“#define”）を含むファイルを，自動割付け結果ヘッダファイルとして生成する．整数値を割り付けられた識別子は，そのコンフィギュレータおよびそれ以降のコンフィギュレータが処理する静的APIのパラメータ中で，割り付けられた整数値に展開されるプリプロセッサマクロと同等なものとして用いることができる．

(b) 自動割付け非対応整数値パラメータ

整数値（負の数を含む）またはそれに展開されるプリプロセッサマクロ（後に述べる制限あり）のみを記述できるパラメータ．ID番号自動割付けの対象とならないオブジェクトのID番号やオブジェクト番号などがこれに該当する．

(c) プリプロセッサ定数式パラメータ

プリプロセッサによって解釈できる定数式のみを記述できるパラメータ．定数およびマクロと，プリプロセッサが解釈できる演算子のみを用いることができる．オブジェクト属性などがこれに該当する．

(d) 一般定数式パラメータ

C言語の任意の定数式を記述できるパラメータ．多くのパラメータはこれに該当する．

静的APIの各パラメータがどれに分類されるかは，静的API毎に定める．自動割付け対応整数値パラメータ，自動割付け非対応整数値パラメータ，プリプロセッサ定数式パラメータに該当するものは，静的APIの機能説明においてその

旨を明示し，明示のないものは一般定数式パラメータである．

ITRON仕様共通静的APIとして，ファイルをインクルードする機能を規定している．そのため，システムコンフィギュレーションファイル中でファイルをインクルードするために，プリプロセッサディレクティブ（“#include”）を用いる方法と，共通静的APIを用いる方法がある．この2つの方法には次の違いがある．

- 自動割付け対応整数値パラメータと自動割付け非対応整数値パラメータ（以下，これらをあわせて整数値パラメータと呼ぶ）の記述にプリプロセッサマクロを用いる場合には，システムコンフィギュレーションファイルまたはそこからプリプロセッサディレクティブを用いてインクルードされるファイル中で定義されたプリプロセッサマクロのみを用いることができる．
- プリプロセッサディレクティブを用いてインクルードされるファイルには，静的APIとプリプロセッサディレクティブのみを記述することができる．それに対して，静的APIを用いてインクルードされるファイルには，C言語の宣言や定義とプリプロセッサディレクティブのみを記述することができる．

メモリ領域をカーネルに確保させるための指定などに用いるNULLは，静的APIのパラメータ中では識別子として認識される．計算結果が0になる定数式はNULLと解釈されるとは限らず，そのような定数式を記述した場合の振舞いは実装依存とする．また，コンフィギュレータが処理する前に，NULLがプリプロセッサによってマクロ展開されてはならない．すなわち，システムコンフィギュレーションファイルおよびそこからプリプロセッサディレクティブを用いてインクルードされるファイル中で，NULLをプリプロセッサマクロとして定義してはならない．

静的APIに文法エラーやパラメータ数の過不足があった場合には，それを検出したコンフィギュレータがエラーを報告する．また，コンフィギュレータは，静的APIで生成されたオブジェクトのID番号が連続にならない場合にも，エラーを報告することができる．静的APIの処理において検出すべきエラーと，エラーを検出した場合の扱いについては，実装定義とする．

【スタンダードプロファイル】

ほとんどの静的APIで，実装独自にパラメータを追加することが許されているが，そのような実装をスタンダードプロファイルに準拠させるためには，追加したパラメータが記述されていない場合でも，デフォルト値を補うなどの方法で正しく処理できることが必要である．

【補足説明】

静的APIは，システムコンフィギュレーションファイル中でフリーフォーマットで記述できる．すなわち，ワードの間に空白文字や改行，コメントが含まれていてもよい．また，静的APIの最後には“;”が必要である．

C言語の列挙型の定数，sizeofはプリプロセッサで解釈できないため，プリプロセッサ定数式パラメータに用いることはできない．

ファイル構成上，システムコンフィギュレーションファイルからプリプロセッサディレクティブを用いてインクルードされるファイル中から，NULLのプリプロセッサマクロ定義を取り除くことが難しい場合には，次の方法で対応することができる．まず，システムコンフィギュレーションファイルの先頭で特定の識別子（ここでは"CONFIGURATOR"とする）をプリプロセッサマクロとして定義（"#define CONFIGURATOR"）し，問題となるNULLのプリプロセッサマクロ定義を"#ifndef CONFIGURATOR"と"#endif"で囲む．

【仕様決定の理由】

静的APIのパラメータを4種類に分類したのは，コンフィギュレータの実装を単純にするためである．オブジェクトのID番号やオブジェクト番号はコンフィギュレータで解釈することが必須であるため，自動割付けが可能なものを除いては，プリプロセッサを通すと整数値になるものに限定した（整数値パラメータ）．また，オブジェクト属性など，その値によって登録するオブジェクトの構造が変わる可能性のあるパラメータについては，コンフィギュレータが生成するC言語のソースコード中に，そのパラメータに関する条件ディレクティブが記述できるように，プリプロセッサが解釈できる定数式に限定した（プリプロセッサ定数式パラメータ）．その他のパラメータについては，C言語の任意の定数式を記述できることとした（一般定数式パラメータ）．ただし，この方法でコンフィギュレータを実現した場合，コンフィギュレータがすべてのパラメータの定数値を知ることができず，コンフィギュレータによるパラメータエラーのチェックには限界がある．すべてのパラメータの定数値を知りたい場合には，コンフィギュレータからコンパイラを呼び出し，定数式を定数値に変換する方法が考えられるが，この方法ではコンパイラ毎にコンフィギュレータの修正が必要になる可能性があるため，標準とはしないこととした．

2.2 APIの名称に関する原則

2.2.1 ソフトウェア部品識別名

標準化されるソフトウェア部品のAPIの名称に用いるために，その種類を2～4文字程度であらわすソフトウェア部品識別名を定める．複数の機能単位から構成されるソフトウェア部品の場合には，機能単位毎にソフトウェア部品識別名を与える場合もある．ソフトウェア部品識別名は，ソフトウェア部品仕様で定める．

独自にAPIを定義したソフトウェア部品については，この規定の適用範囲外ではあるが，標準化されたソフトウェア部品との名称の衝突を避けるために，ソフトウェア部品識別名を5文字以上とするか，ソフトウェア部品識別名の前に"v"の文字を付加することを推奨する．

以下，ソフトウェア部品識別名を小文字で表記したものを*www*，大文字で表記したものを*WWW*と記述する．

2.2.2 サービスコール

カーネルのサービスコールの名称は，*xxx*で操作の方法，*yyy*で操作の対象をあらわし，*xxx_yyy*の形を基本とする．*xxx_yyy*から派生したサービスコールは，派生したことをあらわす文字*z*を付加し，*zxxx_yyy*の形とする．二重に派生したサービスコールの場合には，派生をあらわす文字を2つ付加し，*zzxxx_yyy*の形となる．

ソフトウェア部品のサービスコールの名称は，*www_xxx_yyy* ないしは *www_zxxx_yyy*の形とする．

実装独自に用意するサービスコール名称は，*xxx*または*zxxx*の前に“v”の文字を付加し，v*xxx_yyy*，v*zxxx_yyy*，*www_*v*xxx_yyy*，*www_*v*zxxx_yyy*の形とするのを原則とする．ただし，カーネル仕様において，非タスクコンテキストから呼び出せるサービスコールに付加する“i”の文字が“v”と重なる場合には,“i”を前にしてiv*xxx_yyy*の形とする．

【補足説明】

μITRON4.0仕様において，*xxx*，*yyy*，*z*に用いている省略名とその元になった英語を表2-1に示す.

2.2.3 コールバック

コールバックの名称は，パラメータの形でのみ仕様にあらわれるため，パラメータの名称の原則に従う.

2.2.4 静的API

静的APIの名称は，原則として，対応するサービスコールの名称を大文字で表記したものとする．サービスコールに対応しない静的APIの名称は，サービスコールの名称に関する原則に準拠して定めた名称を大文字で表記したものとする.

カーネルやソフトウェア部品で共通に利用すべきITRON仕様共通静的APIの名称と意味は，ITRON仕様共通定義で規定する.

2.2.5 パラメータとリターンパラメータ

パラメータとリターンパラメータの名称は，すべて小文字で4～7文字程度とする. パラメータとリターンパラメータの名称に関して, 以下の原則を設ける.

～id	～ID（オブジェクトのID番号，ID型）
～no	～番号（オブジェクト番号）
～atr	～属性（オブジェクト属性，ATR型）
～stat	～状態（オブジェクト状態，STAT型）
～mode	～モード（サービスコールの動作モード，MODE型）

表2-1. μITRON4.0仕様における省略名と元になった英語

xxx	元になった英語
acp	accept
act*	activate
att	attach
cal	call
can	cancel
chg	change
clr	clear
cre	create
def	define
del	delete
dis	disable
dly	delay
ena	enable
exd	exit and delete
ext	exit
fwd	forward
get	get
loc*	lock
pol	poll
ras	raise
rcv	receive
ref	reference
rel	release
rot	rotate
rpl	reply
rsm	resume
set	set
sig	signal
slp	sleep
snd	send
sns	sense
sta	start
stp	stop
sus	suspend
ter	terminate
unl	unlock
wai*	wait
wup*	wake up

yyy	元になった英語
alm	alarm handler
cfg	configuration
cpu	CPU
ctx	context
cyc	cyclic handler
dpn	dispatch pending
dsp	dispatch
dtq	data queue
exc	exception
flg	eventflag
inh	interrupt handler
ini	initilization
int	interrupt
isr	interrupt service routine
mbf	message buffer
mbx	mailbox
mpf	fixed-sized memory pool
mpl	memory pool
mtx	mutex
ovr	overrun handler
por	port
pri	priority
rdq	ready queue
rdv	rendezvous
sem	semaphore
sys	system
svc	service call
tex	task exception
tid	task ID
tim	time
tsk	task
tst	task status
ver	version

z	元になった英語
a	automatic ID assignment
f	force
i	interrupt
p	poll
t	timeout

※*を付した省略名は*yyy*としても用いている.

〜pri	〜優先度（優先度，PRI型）
〜sz	〜サイズ（単位はバイト数，SIZE型またはUINT型）
〜cnt	〜の個数（単位は個数，UINT型）
〜ptn	〜パターン
〜tim	〜時刻，〜時間
〜cd	〜コード
i〜	〜の初期値
max〜	〜の最大値
min〜	〜の最小値
left〜	残り〜
p_〜	リターンパラメータ（またはパラメータ）を入れる領域へのポインタ
pk_〜	パケットへのポインタ
pk_c*yyy*	cre_*yyy*に渡すパケットへのポインタ
pk_d*yyy*	def_*yyy*に渡すパケットへのポインタ
pk_r*yyy*	ref_*yyy*に渡すパケットへのポインタ
pk_*www*_c*yyy*	*www*_cre_*yyy*に渡すパケットへのポインタ
pk_*www*_d*yyy*	*www*_def_*yyy*に渡すパケットへのポインタ
pk_*www*_r*yyy*	*www*_ref_*yyy*に渡すパケットへのポインタ
ppk_〜	パケットへのポインタを入れる領域へのポインタ

パラメータやリターンパラメータの名称が同じであれば，原則として同じデータ型である．

2.2.6 データ型

データ型の名称は，すべて大文字で2〜10文字程度とする．データ型の名称に関して，以下の原則を設ける．

〜P	ポインタのデータ型
T_〜	パケットのデータ型（構造体）
T_C*YYY*	cre_*yyy*に渡すパケットのデータ型
T_R*YYY*	ref_*yyy*に渡すパケットのデータ型
T_*WWW*_〜	ソフトウェア部品で用いる構造体
T_*WWW*_C*YYY*	*www*_cre_*yyy*に渡すパケットのデータ型
T_*WWW*_R*YYY*	*www*_ref_*yyy*に渡すパケットのデータ型

カーネルやソフトウェア部品で共通に利用すべきITRON仕様共通データ型の名称と意味は，ITRON仕様共通定義で規定する．

2.2.7 定数

定数の名称はすべて大文字で記述し，以下の原則に従って定める．

(A) ITRON仕様共通定数

カーネルやソフトウェア部品で共通に利用すべきITRON仕様共通定数の名称については特に原則を定めず，その意味および値と共に，ITRON仕様共通定義で規定する．

(B) エラーコード

ITRON仕様で定めるメインエラーコードの名称はE_*XXXXX*の形（*XXXXX*は，2～5文字程度），実装独自に定義するメインエラーコードの名称はEV_*XXXXX*の形とする．

サブエラーコードの名称に関しては，特に原則を定めない．

エラークラスの名称は，EC_*XXXXX*の形（*XXXXX*は，2～5文字程度）とする．

(C) その他の定数

その他の定数の名称は，T*UU*_*XXXXX*またはT*UU*_*WWW*_*XXXXX*の形とする（*UU*は1～3文字程度，*XXXXX*は2～7文字程度）．同じパラメータに用いられる定数の名称については，*UU*の部分を共通にする．リターンパラメータについても同様とする．また，ソフトウェア部品のAPIで，多くのサービスコールやコールバックに共通に用いる定数については, *WWW*_*XXXXX*の形とする場合もある．

上記に加えて，その他の定数の名称に関して，以下の原則を設ける．

TA_～	オブジェクトの属性値
TFN_～	サービスコールの機能コード
TFN_*XXX*_*YYY*	*xxx*_*yyy*の機能コード
TFN_*WWW*_*XXX*_*YYY*	*www*_*xxx*_*yyy*の機能コード
TSZ_～	～のサイズ
TBIT_～	～のビット数
TMAX_～	～の最大値
TMIN_～	～の最小値

2.2.8 マクロ

マクロの名称はすべて大文字で記述し，定数と同様の原則に従って定める．特に，カーネルやソフトウェア部品で共通に利用すべきITRON仕様共通マクロの名称と意味は，ITRON仕様共通定義で規定する．

2.2.9 ヘッダファイル

ITRON仕様共通定義で規定されるデータ型，定数，マクロの定義などを含むヘッダファイルの名称を“itron.h”，カーネル仕様で定められるサービスコールの宣言と，データ型，定数，マクロの定義などを含むヘッダファイルの名称を“kernel.h”，カーネルのコンフィギュレータが生成する自動割付け結果ヘッダ

ファイルの名称を“kernel_id.h”とする．

ソフトウェア部品仕様で定められるサービスコールの宣言などを含むヘッダファイルの名称と，ソフトウェア部品のコンフィギュレータが生成する自動割付け結果ヘッダファイルの名称は，ソフトウェア部品識別名で始まる名称とすることを原則として，ソフトウェア部品毎に定める．

2.2.10 カーネルとソフトウェア部品の内部識別子

アプリケーションプログラムとリンクして用いるカーネルおよびソフトウェア部品においては，アプリケーションプログラムとの名称の衝突を避けるために，カーネルやソフトウェア部品の内部に閉じて使われるルーチンやメモリ領域などの識別子の内，オブジェクトファイルのシンボル表に登録され外部から参照できるもの（これを内部識別子と呼ぶ）の名称に関する原則を次のように定める．

カーネルの内部識別子は，C言語レベルで，_kernel_または_KERNEL_で始まる名称とすることを原則とする．ソフトウェア部品の内部識別子は，C言語レベルで，_*www*_または_*WWW*_で始まる名称とすることを原則とする．

2.3 ITRON仕様共通定義

2.3.1 ITRON仕様共通データ型

ヘッダファイルitron.hは，64ビット整数をあらわすデータ型（D，UD，VD）を除いて，以下のITRON仕様共通データ型の定義を含まなければならない．

B	符号付き8ビット整数
H	符号付き16ビット整数
W	符号付き32ビット整数
D	符号付き64ビット整数
UB	符号無し8ビット整数
UH	符号無し16ビット整数
UW	符号無し32ビット整数
UD	符号無し64ビット整数
VB	データタイプが定まらない8ビットの値
VH	データタイプが定まらない16ビットの値
VW	データタイプが定まらない32ビットの値
VD	データタイプが定まらない64ビットの値
VP	データタイプが定まらないものへのポインタ
FP	プログラムの起動番地（ポインタ）
INT	プロセッサに自然なサイズの符号付き整数

UINT	プロセッサに自然なサイズの符号無し整数
BOOL	真偽値（TRUEまたはFALSE）
FN	機能コード（符号付き整数）
ER	エラーコード（符号付き整数）
ID	オブジェクトのID番号（符号付き整数）
ATR	オブジェクト属性（符号無し整数）
STAT	オブジェクトの状態（符号無し整数）
MODE	サービスコールの動作モード（符号無し整数）
PRI	優先度（符号付き整数）
SIZE	メモリ領域のサイズ（符号無し整数）
TMO	タイムアウト指定（符号付き整数, 時間単位は実装定義）
RELTIM	相対時間（符号無し整数，時間単位は実装定義）
SYSTIM	システム時刻（符号無し整数，時間単位は実装定義）
VP_INT	データタイプが定まらないものへのポインタまたはプロセッサに自然なサイズの符号付き整数
ER_BOOL	エラーコードまたは真偽値（符号付き整数）
ER_ID	エラーコードまたはID番号（符号付き整数，負のID番号は表現できない）
ER_UINT	エラーコードまたは符号無し整数（符号付き整数，符号無し整数を表現する場合の有効ビット数はUINTより1ビット短い）

これらのデータ型の中で，VB，VH，VW，VD，VP_INTをどのような型に定義するかは実装依存である．これらのデータ型の変数に値を代入する場合や，変数の値を参照する場合には，明示的に型変換（キャスト）しなければならない．

またSYSTIMは，システム時刻の表現に必要なビット数が整数型として扱えるビット数よりも多い場合には，構造体として定義してもよい．構造体として定義する場合の構造体の内容については，実装定義とする．

【スタンダードプロファイル】

スタンダードプロファイルでは, ITRON仕様共通データ型の有効ビット数と時間単位を次の通りに定める．

INT	16ビット以上
UINT	16ビット以上
FN	16ビット以上
ER	8ビット以上
ID	16ビット以上
ATR	8ビット以上
STAT	16ビット以上

MODE	8ビット以上
PRI	16ビット以上
SIZE	ポインタと同じビット数
TMO	16ビット以上，時間単位は1ミリ秒
RELTIM	16ビット以上，時間単位は1ミリ秒
SYSTIM	16ビット以上，時間単位は1ミリ秒

【補足説明】

SIZEは，タスクのスタックサイズや可変長メモリプールの全体のサイズなど，大きいメモリ領域のサイズを指定する場合に用いる．メッセージの長さなど，小さいメモリ領域のサイズにはUINTを用いる．

SYSTIMを構造体として定義した場合，通常の演算子（例えば，“+”や“–”）を使ってSYSTIM型の値に対する演算を行うことができなくなる．このような場合にもアプリケーションプログラムの移植性を保つためには，SYSTIM型の値に対する演算は関数呼出しの形で記述し，各実装におけるSYSTIMの定義に合致した演算モジュールを用意するといった方法を採る必要がある．

ER_BOOLは負のエラーコードまたは真偽値（TRUEまたはFALSE），ER_IDは負のエラーコードまたは正のID番号，ER_UINTは負のエラーコードまたは非負の整数値（有効ビット数はUINTより1ビット短い）を表すことのできるデータ型であり，いずれも符号付き整数に定義しなければならない．

【µITRON3.0仕様との相違】

CYCTIME，ALMTIME，DLYTIMEをRELTIMに統合し，SYSTIMEをSYSTIMに改名した．また，STAT，MODE，SIZEを新設した．複合データ型としては，BOOL_IDを廃止し，VP_INT，ER_BOOL，ER_ID，ER_UINTを新設した．メモリ領域のサイズは，符号無し整数で扱うことを原則とした．

2.3.2 ITRON仕様共通定数

ヘッダファイルitron.hは，以下のすべてのITRON仕様共通定数の定義を含まなければならない．

(1) 一般

NULL	0	無効ポインタ
TRUE	1	真
FALSE	0	偽
E_OK	0	正常終了

【µITRON3.0仕様との相違】

無効ポインタを，C言語との整合性を重視して，NADR（＝–1）からNULL（＝0）に変更した．

(2) メインエラーコード

(A) 内部エラークラス（EC_SYS，–5～–8）

カーネルやソフトウェア部品の内部エラーのクラス．このエラークラスに属するエラーの検出は，実装依存で省略することができる．

E_SYS –5 システムエラー

カーネルやソフトウェア部品の内部で発生した原因不明のエラー．

(B) 未サポートエラークラス（EC_NOSPT，–9～–16）

ITRON仕様で規定されていないか, 実装でサポートされていない機能であることを示すエラーのクラス．このエラークラスに属するエラーの検出は，実装定義で省略することができる．

E_NOSPT –9 未サポート機能

ITRON仕様では規定されているが，実装ではサポートされていない機能であることを示すエラー．サービスコールの一部ないしは全部の機能をサポートしていない場合に，このエラーを返す．E_RSFN，E_RSATRに該当する場合を除く．

E_RSFN –10 予約機能コード

ITRON仕様で規定されていないか，実装でサポートしていない機能コードが指定されたことを示すエラー．ソフトウェア割込みによりサービスコールを呼び出す場合に発生する．

E_RSATR –11 予約属性

ITRON仕様で規定されていないか，実装でサポートしていない属性値であることを示すエラー．

(C) パラメータエラークラス（EC_PAR，–17～–24）

パラメータの値の不正を示すエラーで，ほぼ静的に検出できるもののクラス．このエラークラスに属するエラーの検出は，実装定義で省略することができる．

E_PAR –17 パラメータエラー

パラメータの値の不正を示すエラーで，ほぼ静的に検出できるもの．E_IDに該当する場合を除く．

E_ID –18 不正ID番号

オブジェクトのID番号が使用できる範囲外であることを示すエラー. ID番号で識別されるオブジェクトに対してのみ発生する．

(D) 呼出しコンテキストエラークラス（EC_CTX，–25～–32）

サービスコールを呼び出したコンテキストに依存して発生するエラーのクラス．このエラークラスに属するエラーの検出は，実装定義で省略することがで

きる．

E_CTX –25 コンテキストエラー

サービスコールを呼び出したコンテキストが，そのサービスコールを呼び出せる状態にないことを示すエラー．E_MACV，E_OACV，E_ILUSEに該当する場合を除く．

E_MACV –26 メモリアクセス違反

サービスコール内でアクセスするメモリ領域として，サービスコールを呼び出したコンテキストからはアクセスできない領域が指定されたことを示すエラー．メモリのない領域が指定された場合にも，このエラーを返す．

E_OACV –27 オブジェクトアクセス違反

サービスコールでアクセスするオブジェクトとして，サービスコールを呼び出したコンテキストからはアクセスできないオブジェクトが指定されたことを示すエラー．オブジェクトをユーザオブジェクトとシステムオブジェクトに分類している場合に，システムオブジェクトにアクセスできないコンテキストからシステムオブジェクトをアクセスしようとした時，このエラーを返す．

E_ILUSE –28 サービスコール不正使用

サービスコールの使用方法が正しくないことを示すエラーで，サービスコールを呼び出したコンテキストや操作対象のオブジェクトの状態に依存して発生するもの．

(E) 資源不足エラークラス（EC_NOMEM，–33～–40）

サービスコールの実行に必要な資源が不足していることを示すエラーのクラス．このエラークラスに属するエラーの検出は，省略することができない．

E_NOMEM –33 メモリ不足

内部で動的にメモリ領域の確保を行うサービスコールで，必要なメモリ領域を確保できなかったことを示すエラー．

E_NOID –34 ID番号不足

ID番号の自動割付けを行うオブジェクト生成のサービスコールで，割付け可能なID番号がないことを示すエラー．

(F) オブジェクト状態エラークラス（EC_OBJ，–41～–48）

操作対象のオブジェクトの状態に依存して，サービスコールを実行できないことを示すエラーのクラス．操作対象のオブジェクトの状態によっては，同じサービスコール呼び出しでもエラーとなるとは限らないため，動的にチェックすることが必要である．このエラークラスに属するエラーの検出は，省略することができない．

E_OBJ –41 オブジェクト状態エラー

操作対象のオブジェクトの状態に依存するエラーで，E_NOEXS と E_QOVRに該当しないもの.

E_NOEXS –42 オブジェクト未生成

操作対象のオブジェクトが生成されていないために，アクセスできないことを示すエラー．このエラーが返るのは，オブジェクトのID番号が使用できる範囲内である場合に限られるため，このエラーが返ったID番号を指定して，オブジェクトの生成を行うことができる.

E_QOVR –43 キューイングオーバフロー

キューイングまたはネスト可能なサービスコールで，その上限回数を超えたことを示すエラー.

(G) 待ち解除エラークラス（EC_RLWAI，–49〜–56）

サービスコールの中で一旦待ち状態に入った後，待ち状態が解除されたことを示すエラーのクラス．このエラークラスに属するエラーの検出は，省略することができない.

E_RLWAI –49 待ち状態の強制解除

待ち状態が強制解除された，または待ち状態にある処理がキャンセルされたことを示すエラー.

E_TMOUT –50 ポーリング失敗またはタイムアウト

タイムアウトが発生した，またはポーリングに失敗したことを示すエラー.

E_DLT –51 待ちオブジェクトの削除

待つ対象となっているオブジェクトが削除されたことを示すエラー.

E_CLS –52 待ちオブジェクトの状態変化

待つ対象となっているオブジェクトの状態変化により，そのサービスコールを実行できない状態となったことを示すエラー．同様のオブジェクトの状態変化がサービスコール呼出し前に起こっていた場合，サービスコール中で待ち状態に入らずに，このエラーを返すことがある.

【補足説明】

E_CLSは，通信回線からのデータ受信を行うサービスコールで，受信待ちの間に回線が異常切断されたことを知らせる場合などに用いる．サービスコールを呼び出す前に回線が異常切断されていた場合にも，同じメインエラーコードを用いることができる.

(H) 警告クラス（EC_WARN，–57〜–64）

処理を実行したが，注意すべき状況（警告を発すべき状況）が発生したことを示すエラーのクラス．このエラークラスに属するエラーは，「サービスコール

でエラーが発生した場合には，サービスコールを呼び出したことによる副作用はない」という原則の例外となっている．このエラークラスに属するエラーの検出は，省略することができない．

E_WBLK –57 ノンブロッキング受付け

ノンブロッキング指定で呼び出したサービスコールの処理が継続していることを示すメインエラーコード．

E_BOVR –58 バッファオーバフロー

データをバッファに受け取ったが，バッファ領域に入らない部分を捨てたことを示すメインエラーコード．

(I) 予約エラーコード（–5～–96で上に定義のないもの）

ITRON仕様の将来の拡張のために予約されているメインエラーコード．

(J) 実装独自エラーコード（–97～–128）

実装独自に定義するためのメインエラーコード．メインエラーコードの名称をEV_*XXXXX*の形としなければならない．

【µITRON3.0仕様との相違】

カーネル仕様の新機能のためのメインエラーコード（E_ILUSE，E_NOID）とソフトウェア部品仕様のためのメインエラーコード（E_CLS，E_WBLK，E_BOVR）を追加し，接続機能用のエラーコード（EN_*XXXXX*）とITRON/FILE仕様専用のエラーコード（E_INOSPT）を削除した．メインエラーコードのエラークラスへの分類を一部見直し，値を割り付けなおした．この際に，メインエラーコードを下位8ビットとしたことから，8ビットの符号付き整数として見た場合も負の値となるようにしている．また，エラー番号（errno）は廃止した．

(3) オブジェクト属性

TA_NULL 0 オブジェクト属性を指定しない

(4) タイムアウト指定

TMO_POL 0 ポーリング
TMO_FEVR –1 永久待ち
TMO_NBLK –2 ノンブロッキング

2.3.3 ITRON仕様共通マクロ

ヘッダファイルitron.hは，以下のすべてのITRON仕様共通マクロの定義を含まなければならない．

(1) エラーコード生成・分解マクロ

ER ercd = ERCD (ER mercd, ER sercd)

メインエラーコードとサブエラーコードからエラーコードを生成する.

ER mercd = MERCD (ER ercd)

エラーコードからメインエラーコードを取り出す.

ER sercd = SERCD (ER ercd)

エラーコードからサブエラーコードを取り出す.

2.3.4 ITRON仕様共通静的API

(1) ファイルのインクルード

INCLUDE (文字列) ;

システムコンフィギュレーションファイルから，プリプロセッサ定数式パラメータおよび一般定数式パラメータの解釈に必要なC言語の宣言・定義とプリプロセッサマクロの定義などを含んだファイルをインクルードするための指定．パラメータの文字列を，プリプロセッサの “#include” の後ろにそのまま置いて解釈できるように記述する.

【補足説明】

ITRON仕様共通静的APIの記述例を示す.

INCLUDE ("<itron.h>") ;
INCLUDE ("\"memory.h\"") ;

【仕様決定の理由】

パラメータを文字列としたのは，システムコンフィギュレーションファイルを最初にプリプロセッサを通す時に展開されないようにするためである.

第3章 μITRON4.0仕様の概念と共通定義

3.1 基本的な用語の意味

(1) タスクと自タスク

プログラムの並行実行の単位を「タスク」と呼ぶ．すなわち，一つのタスク中のプログラムは逐次的に実行されるのに対して，異なるタスクのプログラムは並行して実行が行われる．ただし，並行して実行が行われるというのは，アプリケーションから見た概念的な動作であり，実装上はカーネルの制御のもとで，それぞれのタスクが時分割で実行される．

また，サービスコールを呼び出したタスクを「自タスク」と呼ぶ．

(2) ディスパッチとディスパッチャ

プロセッサが実行するタスクを切り替えることを「ディスパッチ」（または「タスクディスパッチ」）と呼ぶ．また，ディスパッチを実現するカーネル内の機構を「ディスパッチャ」（または「タスクディスパッチャ」）と呼ぶ．

(3) スケジューリングとスケジューラ

次に実行すべきタスクを決定する処理を「スケジューリング」（または「タスクスケジューリング」）と呼ぶ．また，スケジューリングを実現するカーネル内の機構を「スケジューラ」（または「タスクスケジューラ」）と呼ぶ．スケジューラは，一般的な実装では，サービスコール処理の中やディスパッチャの中で実現される．

(4) コンテキスト

一般に，プログラムの実行される環境を「コンテキスト」と呼ぶ．コンテキストが同じというためには，少なくとも，プロセッサの動作モードが同一で，用いているスタック空間が同一（スタック領域が一連）でなければならない．ただし，コンテキストはアプリケーションから見た概念であり，独立したコンテキストで実行すべき処理であっても，実装上は同一のプロセッサ動作モードで同一のスタック空間で実行されることもある．

(5) 優先順位

処理が実行される順序を決める順序関係を「優先順位」と呼ぶ．優先順位の低い処理を実行中に，優先順位の高い処理が実行できる状態になった場合，優先順位の高い処理を先に実行するのが原則である．

【補足説明】

優先度は，タスクやメッセージの処理順序を制御するために，アプリケーションによって与えられるパラメータである．それに対して優先順位は，仕様中で処理の実行順序を明確にするために用いる概念である．タスク間の優先順位は，タスクの優先度に基づいて定められる．

(6) 割込みの禁止

割込み要求があっても，それに対応する処理（割込みハンドラの起動など）を保留することを，「割込みの禁止」と呼ぶ．次にその割込みが許可された時点で，割込み要求がなくなっていなければ，割込み要求に対応する処理が行われる．

3.2 タスク状態とスケジューリング規則

3.2.1 タスク状態

タスク状態は，大きく次の5つに分類される．この内，広義の待ち状態は，さらに3つの状態に分類される．また，実行状態と実行可能状態を総称して，実行できる状態と呼ぶ．

(a) 実行状態（RUNNING）

現在そのタスクを実行中であるという状態．ただし，非タスクコンテキストを実行している間は，非タスクコンテキストの実行を開始する前に実行していたタスクが実行状態であるものとする．

(b) 実行可能状態（READY）

そのタスクを実行する準備は整っているが，そのタスクよりも優先順位の高いタスクが実行中であるために，そのタスクを実行できない状態．言い換えると，実行できる状態のタスクの中で最高の優先順位になればいつでも実行できる状態．

(c) 広義の待ち状態

そのタスクを実行できる条件が整わないために，実行ができない状態．言い換えると，何らかの条件が満たされるのを待っている状態．タスクが広義の待ち状態にある間，プログラムカウンタやレジスタなどのプログラムの実行状態を表現する情報は保存されている．タスクを広義の待ち状態から実行再開する時には，プログラムカウンタやレジスタなどを広義の待ち状態になる直前の値に戻す．広義の待ち状態は，さらに次の3つの状態に分類される．

(c.1) 待ち状態（WAITING）

何らかの条件が整うまで自タスクの実行を中断するサービスコールを呼び出したことにより，実行が中断された状態．

(c.2) 強制待ち状態（SUSPENDED）

他のタスクによって，強制的に実行を中断させられた状態．ただしμITRON4.0仕様では，自タスクを強制待ち状態にすることもできる．

(c.3) 二重待ち状態（WAITING-SUSPENDED）

待ち状態と強制待ち状態が重なった状態．待ち状態にあるタスクに対して，強制待ち状態への移行が要求されると，二重待ち状態に移行させる．

(d) 休止状態（DORMANT）

タスクがまだ起動されていないか，実行を終了した後の状態．タスクが休止状態にある間は，実行状態を表現する情報は保存されていない．タスクを休止状態から起動する時には，タスクの起動番地から実行を開始する．また，レジスタの内容は保証されない．

(e) 未登録状態（NON-EXISTENT）

タスクがまだ生成されていないか，削除された後の，システムに登録されていない仮想的な状態．

実装によっては，以上のいずれにも分類されない過渡的な状態が存在する場合がある（3.5.6節参照）．

実行可能状態に移行したタスクが，現在実行中のタスクよりも高い優先順位を持つ場合には，実行可能状態への移行と同時にディスパッチが起こり，即座に実行状態へ移行する場合がある．この場合，それまで実行状態であったタスクは，新たに実行状態へ移行したタスクにプリエンプトされたという．また，サービスコールの機能説明などで，「実行可能状態に移行させる」と記述されている場合でも，タスクの優先順位によっては，即座に実行状態に移行させる場合もある．

タスクの起動とは，休止状態のタスクを実行可能状態に移行させることをいう．このことから，休止状態と未登録状態以外の状態を総称して，起動された状態と呼ぶことがある．タスクの終了とは，起動された状態のタスクを休止状態に移行させることをいう．

タスクの待ち解除とは，タスクが待ち状態の時は実行可能状態に，二重待ち状態の時は強制待ち状態に移行させることをいう．また，タスクの強制待ちからの再開とは，タスクが強制待ち状態の時は実行可能状態に，二重待ち状態の時は待ち状態に移行させることをいう．

一般的な実装におけるタスクの状態遷移を図3-1に示す．実装によっては，この図にない状態遷移を行う場合がある．

【補足説明】

待ち状態と強制待ち状態は直交関係にあり，強制待ち状態への移行の要求は，タスクの待ち解除条件には影響を与えない．言い換えると，タスクが待ち状態にあるか二重待ち状態にあるかで，タスクの待ち解除条件は変化しない．そのため，資源獲得のための待ち状態（セマフォ資源の獲得待ち状態やメモリブロックの獲得待ち状態など）にあるタスクに強制待ち状態への移行が要求さ

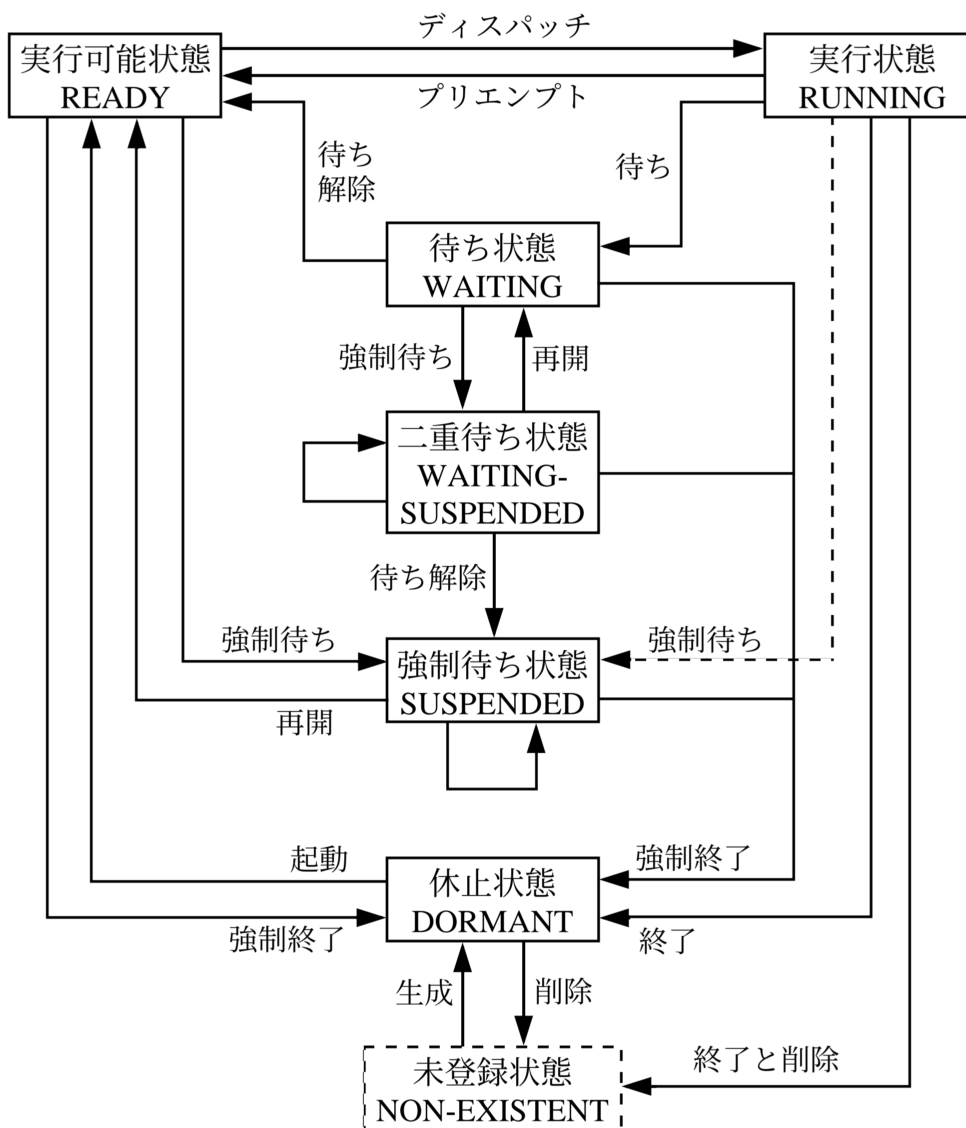


図3-1. タスクの状態遷移

れ，二重待ち状態になった場合にも，強制待ち状態への移行が要求されなかった場合と同じ条件で資源の割付け（セマフォ資源やメモリブロックの割付けなど）が行われる．

【µITRON3.0仕様との相違】

タスク状態の英語名を，形容詞形に統一した．具体的には，実行状態をRUNからRUNNINGに，待ち状態をWAITからWAITINGに，強制待ち状態をSUSPENDからSUSPENDEDに，二重待ち状態をWAIT-SUSPENDからWAITING-SUSPENDEDに変更した．

自タスクを強制待ち状態にすることができることとした．これは，自タスクによる待ちと他のタスクによる待ちを区別しないAPI（POSIXスレッドやJavaスレッドなどのAPI）のインタフェーサを，µITRON4.0仕様のカーネル上に実装するのを容易にするためである．

【仕様決定の理由】

ITRON仕様において待ち状態（自タスクによる待ち）と強制待ち状態（他のタスクによる待ち）を区別しているのは，それらが重なる場合があるためである．それらが重なった状態を二重待ち状態として区別することで，タスクの状態遷移が明確になり，サービスコールの理解が容易になる．それに対して，待ち状態のタスクはサービスコールを呼び出せないため，複数の種類の待ち状態（例えば，起床待ち状態とセマフォ資源の獲得待ち状態）が重なることはない．ITRON仕様では，他のタスクによる待ちには一つの種類（強制待ち状態）しかないため，強制待ち状態が重なった状況を強制待ち要求のネストと扱うことで，タスクの状態遷移を明確にしている．

3.2.2 タスクのスケジューリング規則

ITRON仕様においては，タスクに与えられた優先度に基づくプリエンプティブな優先度ベーススケジューリング方式を採用している．同じ優先度を持つタスク間では，FCFS（First Come First Served）方式によりスケジューリングを行う．具体的には，タスクのスケジューリング規則はタスク間の優先順位を用いて，タスク間の優先順位はタスクの優先度によって，それぞれ次のように規定される．

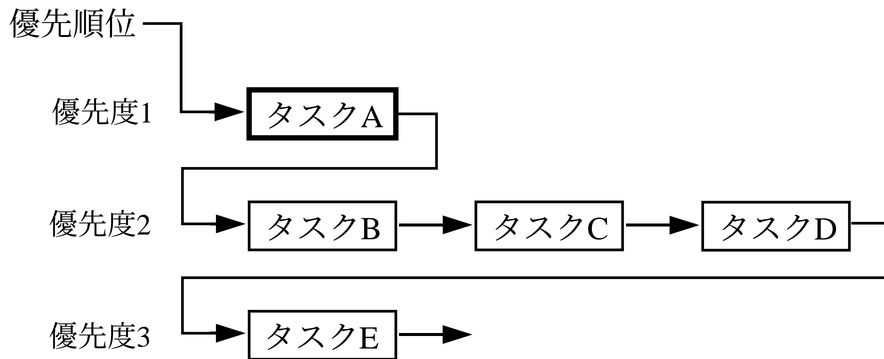
実行できるタスクが複数ある場合には，その中で最も優先順位の高いタスクが実行状態となり，他は実行可能状態となる．タスク間の優先順位は，異なる優先度を持つタスク間では，高い優先度を持つタスクの方が高い優先順位を持つ．同じ優先度を持つタスク間では，先に実行できる状態（実行状態または実行可能状態）になったタスクの方が高い優先順位を持つ．ただし，サービスコールの呼出しにより，同じ優先度を持つタスク間の優先順位が変更される場合がある．

最も高い優先順位を持つタスクが替わった場合には，ただちにディスパッチが起こり，実行状態のタスクが切り替わる．ただし，ディスパッチが起こらない状態になっている場合には，実行状態のタスクの切替えは，ディスパッチが起こる状態となるまで保留される．

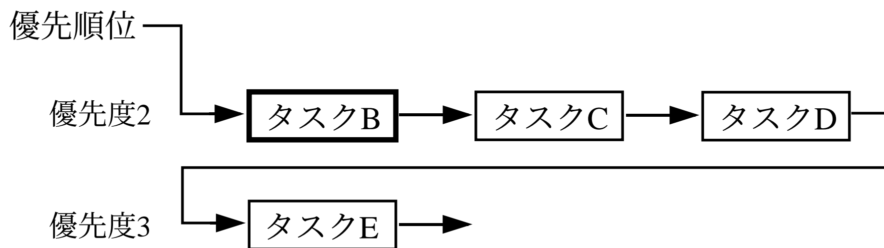
【補足説明】

ITRON仕様のスケジューリング規則では，優先順位の高いタスクが実行できる状態にある限り，それより優先順位の低いタスクは全く実行されない．すなわち，最も高い優先順位を持つタスクが待ち状態に入るなどの理由で実行できない状態とならない限り，他のタスクは全く実行されない．この点で，複数のタスクを公平に実行しようというTSS（Time Sharing System）のスケジューリング方式とは根本的に異なっている．ただし，同じ優先度を持つタスク間の優先順位は，サービスコールを用いて変更することが可能である．アプリケーションがそのようなサービスコールを用いて，TSSにおける代表的なスケジューリング方式であるラウンドロビン方式を実現することができる．

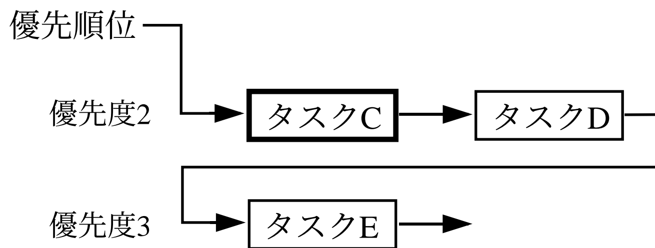
同じ優先度を持つタスク間では，先に実行できる状態（実行状態または実行可能状態）になったタスクの方が高い優先順位を持つことを，図3-2の例を用いて説明する．図3-2 (a)は，優先度1のタスクA，優先度3のタスクE，優先度2



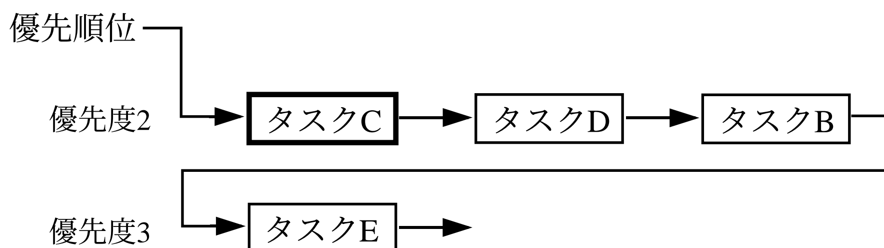
(a) 最初の状態の優先順位



(b) タスクBが実行状態になった後の優先順位



(c) タスクBが待ち状態になった後の優先順位



(d) タスクBが待ち解除された後の優先順位

図3-2. タスク間の優先順位

のタスクB，タスクC，タスクDがこの順序で起動された後のタスク間の優先順位を示す．この状態では，最も優先順位の高いタスクAが実行状態となって

いる．

ここでタスクAが終了すると，次に優先順位の高いタスクBを実行状態に遷移させる（図3-2 (b)）．その後タスクAが再び起動されると，タスクBはプリエンプトされて実行可能状態に戻るが，この時タスクBは，タスクCとタスクDのいずれよりも先に実行できる状態になっていたことから，同じ優先度を持つタスクの中で最高の優先順位を持つことになる．すなわち，タスク間の優先順位は図3-2 (a)の状態に戻る．

次に，図3-2 (b)の状態でタスクBが待ち状態になった場合を考える．タスクの優先順位は実行できるタスクの間で定義されるため，タスク間の優先順位は図3-2 (c)の状態となる．その後タスクBが待ち解除されると，タスクBはタスクCとタスクDのいずれよりも後に実行できる状態になったことから，同じ優先度を持つタスクの中で最低の優先順位となる（図3-2 (d)）．

以上を整理すると，実行可能状態のタスクが実行状態になった後に実行可能状態に戻った直後には，同じ優先度を持つタスクの中で最高の優先順位を持っているのに対して，実行状態のタスクが待ち状態になった後に待ち解除されて実行できる状態になった直後には，同じ優先度を持つタスクの中で最低の優先順位となる．

【μITRON3.0仕様との相違】

レディキューは実装上の概念であるため，仕様の説明には，レディキューに代えて優先順位という用語を用いることにした．

タスクが強制待ち状態から実行可能状態に移行した時に，同じ優先度内で最低の優先順位になるものとした．これは，実装依存性を削減するためである．

3.3 割込み処理モデル

3.3.1 割込みハンドラと割込みサービスルーチン

μITRON4.0仕様では，外部割込み（以下，単に割込みと呼ぶ）によって起動される処理として，割込みハンドラと割込みサービスルーチンがある．

割込みハンドラは，プロセッサの機能のみに依存して起動することを基本とする．したがって，割込みコントローラ（IRC；Interrupt Request Controller）の操作は，カーネルではなく，割込みハンドラで行う．割込みハンドラの記述方法は，一般にはプロセッサの割込みアーキテクチャや用いるIRCなどに依存するため，実装定義とする．割込みハンドラをそのままの形で異なるシステムに移植することはできない．

それに対して割込みサービスルーチンは，割込みハンドラから起動するルーチンで，プロセッサの割込みアーキテクチャや用いるIRCなどに依存せずに記述することができる．すなわち，IRCの操作は割込みサービスルーチンを起動する割込みハンドラで行われ，割込みサービスルーチンではIRCを操作する必要はない．

μITRON4.0仕様では，アプリケーションが用意した割込みハンドラを登録するためのAPI（DEF_INHなど）と，割込みサービスルーチンを登録するためのAPI（ATT_ISRなど）の両方を規定しており，実装ではそのいずれかのAPIを提供すればよい．割込みハンドラを登録するためのAPIを提供する場合，割込みハンドラの起動の前後に行うべき処理を含むルーチン（これを，割込みハンドラの出入口処理と呼ぶ）をカーネルで用意し，割込みハンドラ属性によっては，用意した出入口処理を経由して割込みハンドラを起動することができる．割込みサービスルーチンを登録するためのAPIのみを提供する場合には，割込みサービスルーチンを起動する割込みハンドラは，カーネルが用意する．また，両方のAPIを提供することも可能であるが，その場合，両方のAPIを併用した場合の振舞いは実装定義とする．

カーネルは，ある優先度よりも高い優先度を持つ割込み（禁止できない割込みを含む）を，管理しないものとすることができる．このような割込みを，カーネルの管理外の割込みと呼ぶ．どの優先度より高い優先度を持つものをカーネルの管理外の割込みとするかは，実装定義である．カーネルの管理外の割込みによって起動される割込みハンドラからは，カーネルのサービスコールを呼び出すことができない．この仕様書で，割込み（ないしは，割込みハンドラ）という場合には，カーネルの管理外の割込み（ないしは，それによって起動される割込みハンドラ）は含まないものとする．

図3-3にμITRON4.0仕様における割込み処理の概念モデルを示す．ただし，この図はあくまでも概念モデルを示すものであり，実際の実現方法は実装やアプリケーション毎に自由に決定することができる．

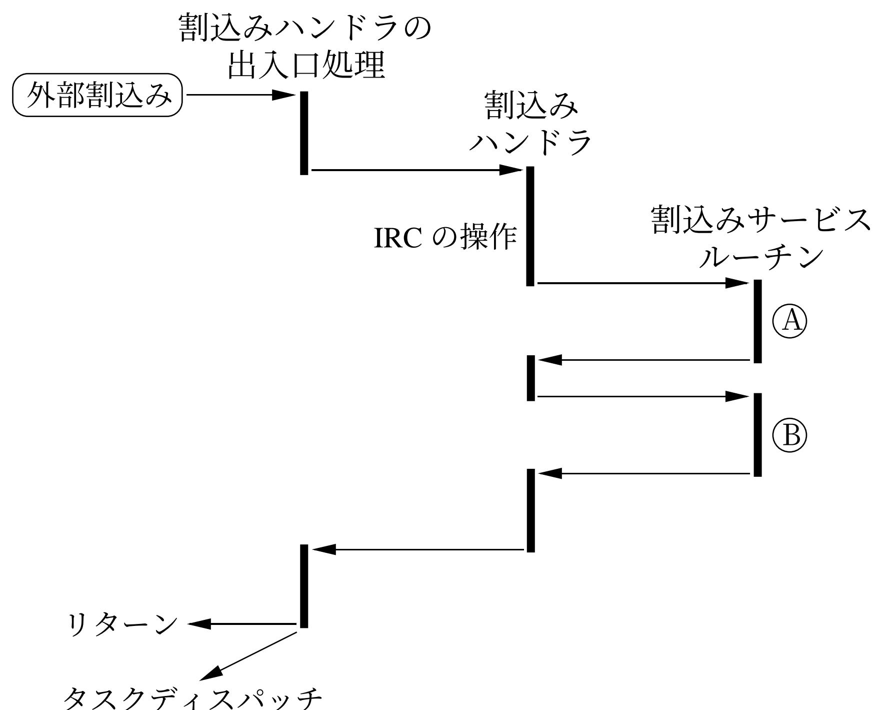


図3-3. 割込み処理モデル

【補足説明】

割込みハンドラの出入口処理で必要な処理として，割込みハンドラ内で使用するレジスタの保存と復帰，スタックの切替え，タスクディスパッチ処理，プロセッサレベルでの割込みハンドラからの復帰が挙げられる．この中で実際にどの処理が必要かは，場合によって異なる．また，これらの処理の中で，どれをカーネルが用意する出入口処理で行い，どれをアプリケーションが登録する割込みハンドラで行わなければならないとするかは，実装定義である（割込みハンドラ属性によっても異なる）．

割込みサービスルーチンを起動する際に，割込みハンドラで行わなければならないIRCの操作として，IRCからの割込み要因の取り出し，取り出した割込み要因による分岐，IRCのエッジトリガのクリア，IRCの割込みサービス中フラグのクリアが挙げられる．また，割込みサービスルーチンを起動する前に，CPUロック解除状態にすることが必要である．

割込みサービスルーチンを起動する際のオーバヘッドを削減するために，割込みハンドラの出入口処理と割込みハンドラを一体で用意する方法や，割込みハンドラ内に割込みサービスルーチンをインライン展開する方法を採ることができる．

【スタンダードプロファイル】

スタンダードプロファイルでは，割込みハンドラを登録するためのAPIと割込みサービスルーチンを登録するためのAPIのいずれかをサポートしなければならない．

【仕様決定の理由】

アプリケーションの割込み処理部分の移植性を向上させるために，割込みハンドラよりも高い移植性を持つ割込みサービスルーチンを導入した．一方，移植性の低い割込みハンドラを残したのは，用いるIRCに依存しない形でカーネルを提供できるようにするためである．

3.3.2 割込みの指定方法と割込みサービスルーチンの起動

μITRON4.0仕様では，割込みを指定するための方法として，割込み番号と割込みハンドラ番号の2つがある．また，割込みサービスルーチンは，ID番号で識別される．

割込みハンドラ番号（INHNO型）は，割込みハンドラを登録する際に，割込みハンドラを登録する対象となる割込みを指定するために用いる番号である．指定された割込みは，IRCを操作しない範囲で判別できるのが基本であることから，一般的な実装では，プロセッサの割込みベクトル番号に対応する．割込みベクトルを持たないプロセッサにおいては，指定することができる割込みハンドラ番号は一つのみとなる場合もある．

割込み番号（INTNO型）は，割込みサービスルーチンを登録する際に，割込みサービスルーチンを登録する対象となる割込みを指定するために用いる番号

である．また，割込み番号は，割込みを個別に禁止／許可するためのサービスコール（dis_int，ena_int）で，禁止／許可の対象となる割込みを指定するためにも用いられる．割込みサービスルーチンの起動や，割込みの個別の禁止／許可は，IRCを操作して実現することを想定していることから，一般的な実装では，割込み番号はIRCへの割込み要求入力ラインに対応する．

割込みサービスルーチンは，デバイスからの割込み要求ラインに対応して用意するのを基本とする．IRCへの一つの割込み要求入力ラインには，複数のデバイスからの割込み要求が接続される場合があるため，一つの割込み番号に対して複数の割込みサービスルーチンを登録することができる．割込み番号で指定された割込みが発生した場合には，その割込み番号に対して登録されたすべての割込みサービスルーチンを順に起動する．割込みサービスルーチンを起動する順序は実装依存である．一つの割込み番号に対して登録された複数の割込みサービスルーチンを区別するために，割込みサービスルーチンはID番号を用いて識別する．

【補足説明】

IRCの一つの割込み要求入力ラインに複数のデバイスからの割込み要求が接続されていても，割込みを要求したデバイスがプロセッサに割込みベクトル番号を供給する場合には，プロセッサレベルで割込み要因を判別することができる．このような場合，異なる割込みベクトル番号を供給する割込みは，異なる割込み番号を持つものと解釈することができる．

3.4 例外処理モデル

3.4.1 例外処理の枠組み

μITRON4.0仕様では，例外処理のための機能として，CPU例外ハンドラとタスク例外処理の機能を規定する．

CPU例外ハンドラは，プロセッサが検出するCPU例外によって起動される．CPU例外ハンドラは，CPU例外の種類毎に，アプリケーションで登録することができる．CPU例外ハンドラの起動の前後に行うべき処理を含むルーチン（これを，CPU例外ハンドラの出入口処理と呼ぶ）をカーネルで用意し，CPU例外ハンドラ属性によっては，用意した出入口処理を経由してCPU例外ハンドラを起動することができる．

同一のCPU例外に対するCPU例外ハンドラはシステム全体で共通であるため，CPU例外ハンドラ内でCPU例外が発生したコンテキストや状態を調べ，必要であればタスク例外処理機能を用いてCPU例外が発生したタスクに例外処理を任せることができる．

タスク例外処理機能は，タスクを指定してタスク例外処理を要求するサービスコールを呼び出すことで，指定したタスクに実行中の処理を中断させ，タスク例外処理ルーチンを実行させるための機能である．タスク例外処理ルーチンの

実行は，タスクと同じコンテキストで行われる．タスク例外処理ルーチンからリターンすると，中断された処理の実行が継続される．タスク毎に一つのタスク例外処理ルーチンを，アプリケーションで登録することができる．タスク例外処理機能については，4.3節で述べる．

【スタンダードプロファイル】

スタンダードプロファイルでは，CPU例外ハンドラとタスク例外処理ルーチンをサポートしなければならない．

3.4.2 CPU例外ハンドラで行える操作

CPU例外ハンドラの記述方法は，一般にはプロセッサのCPU例外アーキテクチャやカーネルの実装などに依存するため，実装定義とする．CPU例外ハンドラをそのままの形で異なるシステムに移植することはできない．

CPU例外ハンドラ内で呼出し可能なサービスコールは実装定義である．ただし，CPU例外ハンドラ内で次の操作を行う方法を用意しなければならない．これらの操作を行う方法も，実装定義である．

(a) CPU例外が発生したコンテキストや状態の読出し．具体的には，CPU例外が発生した処理でsns_*yyy*を呼び出した場合の結果を，CPU例外ハンドラ内で取り出せること．

(b) CPU例外が発生したタスクのID番号の読出し（例外を発生させたのがタスクである場合のみ）．

(c) タスク例外処理の要求．具体的には，CPU例外ハンドラ内で，ras_texと同等の操作ができること．

ここで，CPUロック状態でCPU例外が発生した場合には，(b)および(c)の操作ができることは要求されない．

3.5 コンテキストとシステム状態

3.5.1 処理単位とコンテキスト

μITRON4.0仕様のカーネルは，次の処理単位の実行制御を行う．

- (a) 割込みハンドラ
  - (a.1) 割込みサービスルーチン
- (b) タイムイベントハンドラ
- (c) CPU例外ハンドラ
- (d) 拡張サービスコールルーチン
- (e) タスク
  - (e.1) タスク例外処理ルーチン

割込みハンドラおよび割込みサービスルーチンは，それぞれ独立したコンテキストで実行される．以下この節では，割込みサービスルーチンに関する記述が

別にない限りは，割込みハンドラに関する記述は割込みサービスルーチンにも適用される．

タイムイベントハンドラは，時間をきっかけとして起動される処理で，周期ハンドラ，アラームハンドラ，オーバランハンドラの3つの種類がある．タイムイベントハンドラは，それぞれ独立したコンテキストで実行される．周期ハンドラについては4.7.2節，アラームハンドラについては4.7.3節，オーバランハンドラについては4.7.4節で述べる．

CPU例外ハンドラは，CPU例外毎およびCPU例外が発生したコンテキスト毎に，それぞれ独立したコンテキストで実行される．

拡張サービスコールルーチンは，拡張サービスコールの呼出しにより起動される処理で，アプリケーションによって登録される．拡張サービスコールルーチンは，拡張サービスコール毎および拡張サービスコールが呼び出されたコンテキスト毎に，それぞれ独立したコンテキストで実行される．拡張サービスコールルーチンについては，4.10節で述べる．

タスクは，それぞれ独立したコンテキストで実行される．タスク例外処理ルーチンは，そのタスクのコンテキスト内で実行される．以下この節では，タスク例外処理ルーチンに関する記述が別にない限りは，タスクに関する記述はタスク例外処理ルーチンにも適用される．

以上の処理単位に分類されない処理として，カーネル自身の処理がある．カーネルの処理は，サービスコール処理，ディスパッチャ，割込みハンドラ（または割込みサービスルーチン）とCPU例外ハンドラの出入口処理などから構成される．カーネルが実行されるコンテキストは，アプリケーションの振舞いには影響しないため，特に規定しない．

【μITRON3.0仕様との相違】

タイマハンドラに代えて，タイムイベントハンドラという用語を用いることにした．また，拡張SVCハンドラに代えて，拡張サービスコールルーチンという用語を用いることにした．

3.5.2 タスクコンテキストと非タスクコンテキスト

タスクの処理の一部とみなすことのできるコンテキストを総称してタスクコンテキスト，そうでないコンテキストを総称して非タスクコンテキストと呼ぶ．

タスクの実行されるコンテキストは，タスクコンテキストに分類される．割込みハンドラおよびタイムイベントハンドラが実行されるコンテキストは，非タスクコンテキストに分類される．CPU例外ハンドラおよび拡張サービスコールルーチンが実行されるコンテキストがいずれに分類されるかは，それぞれ，CPU例外が発生したコンテキストおよび拡張サービスコールが呼び出されたコンテキストに依存して，次のように定められる．

タスクコンテキストを実行中にCPU例外が発生した場合には，CPU例外ハンド

ラが実行されるコンテキストがタスクコンテキストと非タスクコンテキストのいずれに分類されるかは実装定義とする．非タスクコンテキストを実行中にCPU例外が発生した場合には，CPU例外ハンドラが実行されるコンテキストは非タスクコンテキストに分類される．

タスクコンテキストから拡張サービスコールを呼び出した場合には，拡張サービスコールルーチンが実行されるコンテキストはタスクコンテキストに分類される．非タスクコンテキストから拡張サービスコールを呼び出した場合には，拡張サービスコールルーチンが実行されるコンテキストは非タスクコンテキストに分類される．

μITRON4.0仕様では，タスクコンテキストから呼び出すサービスコールと，非タスクコンテキストから呼び出すサービスコールを区別して扱う．非タスクコンテキストからのサービスコール呼出しについては，3.6節で述べる．

非タスクコンテキストから，暗黙で自タスクを指定するサービスコールや，自タスクを広義の待ち状態にする可能性のあるサービスコールが呼び出された場合には，E_CTXエラーを返す．また，サービスコールに渡すパラメータとして，自タスクを指定するTSK_SELF（＝0）を用いることもできない．渡された場合には，E_IDエラーとなる．

【補足説明】

3.5.3節で述べるように，CPU例外ハンドラの優先順位はディスパッチャよりも高いため，CPU例外ハンドラ実行中はディスパッチが起こらない．そのため，CPU例外ハンドラがタスクコンテキストで実行される実装で，CPU例外ハンドラ内で自タスクを広義の待ち状態にする可能性のあるサービスコールを呼び出した場合の振る舞いは未定義である．ただし，エラーを報告する場合はE_CTXを返すものとする．

【μITRON3.0仕様との相違】

タスク部とタスク独立部に代えて，それぞれタスクコンテキストと非タスクコンテキストという用語を用いることとした．カーネルが実行されるコンテキストは規定していないため，過渡的な状態という用語は用いていない．また，μITRON4.0仕様ではプロセッサの動作モードについて規定していないため，準タスク部の概念は定義せず，タスクコンテキストに含まれるものとした．

3.5.3 処理の優先順位とサービスコールの不可分性

μITRON4.0仕様のカーネルが，各処理単位とディスパッチャを実行する優先順位は次のように規定される．

各処理単位とディスパッチャの間では，次の順序で優先順位が高い．

(1) 割込みハンドラ，タイムイベントハンドラ，CPU例外ハンドラ
(2) ディスパッチャ（カーネルの処理の一部）
(3) タスク

割込みハンドラの優先順位は，ディスパッチャの優先順位よりも高い．割込みハンドラおよび割込みサービスルーチン相互間の優先順位は，それらを起動する外部割込みの優先度に対応して定めることを基本に，実装定義で定める．

タイムイベントハンドラ（オーバランハンドラを除く）の優先順位は，isig_tim を呼び出した割込みハンドラの優先順位以下で，ディスパッチャの優先順位よりも高いという範囲内で，実装定義で定める．オーバランハンドラの優先順位は，ディスパッチャの優先順位よりも高いという範囲内で，実装定義で定める．

CPU例外ハンドラの優先順位は，CPU例外が発生した処理の優先順位と，ディスパッチャの優先順位のいずれよりも高い．割込みハンドラやタイムイベントハンドラとの間の優先順位は，実装定義である．

拡張サービスコールルーチンの優先順位は，拡張サービスコールを呼び出した処理の優先順位よりも高く，拡張サービスコールを呼び出した処理よりも高い優先順位を持つ他のいずれの処理の優先順位よりも低い．

タスクの優先順位は，ディスパッチャの優先順位よりも低い．タスク相互間の優先順位は，タスクのスケジューリング規則によって定められる．

カーネルのサービスコール処理は不可分に実行され，サービスコール処理の途中の状態がアプリケーションから見えないのが基本であるが，システムの応答性を向上させるために，サービスコール処理の途中でアプリケーションプログラムが実行される実装も認める．この場合にも，アプリケーションがサービスコールを用いて観測できる範囲で，サービスコールが不可分に実行された場合と同様に振る舞わなければならない．このように振る舞わせることを，サービスコールの不可分性を保証するという．ただし，実装独自の機能を追加する場合には，実装方式によっては高い応答性を保ちつつサービスコールの不可分性を保証することが難しい場合が考えられる．そのような場合には，サービスコールの不可分性の原則を緩めることを認める．

カーネルのサービスコール処理が不可分に実行される場合には，サービスコール処理は最も高い優先順位を持つことになるが，上述の理由から，サービスコール処理の優先順位は，サービスコールを呼び出した処理の優先順位よりも高いという条件を満たす範囲で，実装依存とする．

サービスコール処理以外のカーネルの処理（ディスパッチャ，割込みハンドラとCPU例外ハンドラの出入口処理など）も，サービスコール処理と同様とする．

【スタンダードプロファイル】

スタンダードプロファイルに準拠する実装では，スタンダードプロファイル機能の範囲内で，サービスコールの不可分性を保証しなければならない．

【補足説明】

ディスパッチャの優先順位は割込みハンドラの優先順位よりも低いため，すべての割込みハンドラの処理を終えるまで，ディスパッチは起こらない．これを，従来の仕様では遅延ディスパッチの原則と呼んでいた．タイムイベントハンドラ，CPU例外ハンドラについても同様である．

3.5.4 CPUロック状態

システムは，CPUロック状態かCPUロック解除状態かのいずれかの状態をとる．CPUロック状態では，カーネルの管理外の割込みを除くすべての割込みが禁止され，ディスパッチも起こらない．また，割込みが禁止されていることから，タイムイベントハンドラの起動も保留される．CPUロック状態は，実行中の処理の優先順位が，他のいずれの処理よりも高くなった状態とみなすこともできる．実装によっては，CPUロック状態とCPUロック解除状態のどちらにも該当しない過渡的な状態が存在する場合がある．

CPUロック状態に移行することを「CPUロックする」，CPUロック解除状態に移行することを「CPUロックを解除する」ともいう．

CPUロック状態では，以下のサービスコールを呼び出すことができる．

loc_cpu／iloc_cpu	CPUロック状態への移行
unl_cpu／iunl_cpu	CPUロック状態の解除
sns_ctx	コンテキストの参照
sns_loc	CPUロック状態の参照
sns_dsp	ディスパッチ禁止状態の参照
sns_dpn	ディスパッチ保留状態の参照
sns_tex	タスク例外処理禁止状態の参照

ここで，loc_cpu／iloc_cpuは，タスクコンテキストからはloc_cpu，非タスクコンテキストからはiloc_cpuを呼び出せることを示す（unl_cpu／iunl_cpuについても同様）．CPUロック状態で，これら以外のサービスコールが呼び出された場合には，E_CTXエラーを返す．

割込みハンドラの実行開始直後に，CPUロック状態・CPUロック解除状態・どちらにも該当しない過渡的な状態のいずれになっているかは実装依存である．ただし，CPUロック状態または過渡的な状態で，割込みハンドラを実行開始する場合，割込みハンドラ内でCPUロック解除状態にするための方法を，実装定義で用意することが必要である．また，CPUロック解除状態にした後に，割込みハンドラから正しくリターンするための方法も実装定義とする．実装で規定されている方法に従わずにリターンした場合の振舞いは未定義である．割込みハンドラからのリターン直後は，CPUロック解除状態になる．

割込みサービスルーチンとタイムイベントハンドラの実行開始直後は，CPUロック解除状態になっている．アプリケーションは，これらのルーチン／ハンドラからリターンする前に，CPUロック解除状態にしなければならない．CPUロック状態でこれらのルーチン／ハンドラからリターンした場合の振舞いは未定義である．これらのルーチン／ハンドラからのリターン直後は，CPUロック解除状態になる．

CPU例外ハンドラの起動とそこからのリターンによって，CPUロック／ロック解除状態は変化しない．アプリケーションは，CPU例外ハンドラ内でCPUロック／ロック解除状態を変化させた場合，CPU例外ハンドラからリターンする前

に元の状態に戻さなければならない. 元の状態に戻さずにCPU例外ハンドラからリターンした場合の振舞いは未定義である.

拡張サービスコールルーチンの起動とそこからのリターンによって, CPUロック／ロック解除状態は変化しない.

タスクの実行開始直後は，CPUロック解除状態になっている．アプリケーションは，自タスクを終了させる前に，CPUロック解除状態にしなければならない．CPUロック状態で自タスクを終了させた場合の振舞いは未定義である．

タスク例外処理ルーチンの起動とそこからのリターンによって, CPUロック／ロック解除状態は変化しない．ただし，CPUロック状態でタスク例外処理ルーチンが起動されるかどうかは，この仕様上は規定されない．アプリケーションは, タスク例外処理ルーチン内でCPUロック／ロック解除状態を変化させた場合, タスク例外処理ルーチンからリターンする前に元の状態に戻さなければならない. 元の状態に戻さずにタスク例外処理ルーチンからリターンした場合の振舞いは未定義である.

【補足説明】

CPUロック解除状態であっても，割込みが許可されているとは限らない．

「CPU例外ハンドラの起動とそこからのリターンによって，CPUロック／ロック解除状態は変化しない」とは，次のことを意味する．すなわち，CPU例外ハンドラの実行開始直後は, CPU例外がCPUロック状態で発生した場合にはCPUロック状態，CPUロック解除状態で発生した場合にはCPUロック解除状態になっている．また，CPU例外ハンドラからのリターン直後は，CPU例外ハンドラからCPUロック状態でリターンした場合にはCPUロック状態，CPUロック解除状態でリターンした場合にはCPUロック解除状態になる．拡張サービスコールルーチンとタスク例外処理ルーチンについても，これと同様である．

【μITRON3.0仕様との相違】

CPUロック／ロック解除状態の意味を変更した．具体的には，CPUロック状態を，割込みとタスクディスパッチが禁止された状態と考えていたのを，それらとは概念的には独立した状態と扱うことにした．また，CPUロック状態では，一部の例外を除いてサービスコールを呼び出せないのを基本とした．

3.5.5 ディスパッチ禁止状態

システムは, ディスパッチ禁止状態かディスパッチ許可状態かのいずれかの状態をとる．ディスパッチ禁止状態では，ディスパッチは起こらない．ディスパッチ禁止状態は，実行中の処理の優先順位が，ディスパッチャよりも高くなった状態とみなすこともできる．実装によっては，ディスパッチ禁止状態とディスパッチ許可状態のどちらにも該当しない過渡的な状態が存在する場合がある．

ディスパッチ禁止状態に移行することを「ディスパッチを禁止する」，ディスパッチ許可状態に移行することを「ディスパッチを許可する」ともいう．

ディスパッチ禁止状態では，タスクコンテキストから呼び出せるサービスコールに次の制限がある．すなわち，ディスパッチ禁止状態で自タスクを広義の待ち状態にする可能性のあるサービスコールが呼び出された場合には，E_CTXエラーを返す．それに対して，非タスクコンテキストから呼び出せるサービスコールは，ディスパッチ禁止状態でも制限を受けない．

割込みハンドラ，割込みサービスルーチン，タイムイベントハンドラ，CPU例外ハンドラの起動と，それらからのリターンによって，ディスパッチ禁止／許可状態は変化しない．アプリケーションは，これらのハンドラ／ルーチン内でディスパッチ禁止／許可状態を変化させた場合，ハンドラ／ルーチンからリターンする前に元の状態に戻さなければならない．元の状態に戻さずにハンドラ／ルーチンからリターンした場合の振舞いは未定義である．

拡張サービスコールルーチンの起動とそこからのリターンによって，ディスパッチ禁止／許可状態は変化しない．

タスクの実行開始直後は，ディスパッチ許可状態になっている．アプリケーションは，自タスクを終了させる前に，ディスパッチ許可状態にしなければならない．ディスパッチ禁止状態で自タスクを終了させた場合の振舞いは未定義である．

タスク例外処理ルーチンの起動と，そこからのリターンによって，ディスパッチ禁止／許可状態は変化しない．

ディスパッチ禁止／許可状態は，CPUロック／ロック解除状態とは独立に管理される．

【補足説明】

ディスパッチ禁止状態から自タスクを広義の待ち状態にする可能性のあるサービスコールが呼び出された場合に実装定義でE_CTXエラーとする仕様は，サービスコール単位に適用される．

具体的には，タイムアウト付きのサービスコール（例えばtwai_sem）において，タイムアウト時間にTMO_POL（ポーリング）を指定して呼び出した場合の振る舞いを，E_CTXエラーとするか，ポーリングを行うサービスコール（例えばpol_sem）と同じ動作とするかは実装定義とする．

「割込みハンドラの起動とそこからのリターンによって，ディスパッチ禁止／許可状態は変化しない」とは，次のことを意味する．すなわち，割込みハンドラの実行開始直後は，割込みがディスパッチ禁止状態で発生した場合にはディスパッチ禁止状態，ディスパッチ許可状態で発生した場合にはディスパッチ許可状態になっている．また，割込みハンドラからのリターン直後は，割込みハンドラからディスパッチ禁止状態でリターンした場合にはディスパッチ禁止状態，ディスパッチ許可状態でリターンした場合にはディスパッチ許可状態になる．割込みサービスルーチン，タイムイベントハンドラ，CPU例外ハンドラ，拡張サービスコールルーチン，タスク例外処理ルーチンについても，これと同様である．

非タスクコンテキストからディスパッチ禁止／許可状態を変更するためのサービスコールは，μITRON4.0仕様では用意されていない．そのため，実装独自の拡張を行わない限りは，割込みハンドラやタイムイベントハンドラ内でディスパッチ禁止／許可状態を変化させることはできない．CPU例外ハンドラが非タスクコンテキストで実行される場合には，CPU例外ハンドラについても同様である．

ディスパッチ禁止／許可状態はCPUロック状態と独立に管理されるため，例えば，ディスパッチ禁止状態でCPUロック状態に移行し，その後CPUロックを解除しても，ディスパッチ禁止状態は保持されている．また，CPUロック状態でも，ディスパッチ禁止状態かディスパッチ許可状態かを参照することができる．

【μITRON3.0仕様との相違】

ディスパッチ禁止状態の意味を変更した．具体的には，ディスパッチ禁止状態を，CPUロック状態とは独立に管理される状態とした．

3.5.6 ディスパッチ保留状態の間のタスク状態

ディスパッチャよりも優先順位の高い処理が実行されている間，CPUロック状態の間，およびディスパッチ禁止状態の間は，ディスパッチが起こらない．この状態をディスパッチ保留状態と呼び，その間のタスク状態については次のように定める．

ディスパッチ保留状態では，実行中のタスクがプリエンプトされるべき状況となっても，新たに実行すべき状態となったタスクにはディスパッチされない．実行すべきタスクへのディスパッチは，ディスパッチが起こる状態となるまで保留される．ディスパッチが保留されている間は，それまで実行中であったタスクが実行状態であり，ディスパッチが起こる状態となった後に実行すべきタスクは実行可能状態である．

また，実装独自に，非タスクコンテキストから実行中のタスクを強制待ち状態や休止状態に移行させるサービスコールを追加した場合や，ディスパッチ禁止状態で自タスクを強制待ち状態に移行させるサービスコールを呼出し可能とした場合には，ディスパッチ保留状態の間のタスク状態について次のように定める．

ディスパッチ保留状態で，実行中のタスクを強制待ち状態や休止状態へ移行させようとした場合，タスクの状態遷移はディスパッチが起こる状態となるまで保留される．状態遷移が保留されている間は，それまで実行中であったタスクは過渡的な状態にあると考え，その具体的な扱いは実装依存とする．一方，ディスパッチが起こる状態となった後に実行すべきタスクは，実行可能状態である．

【補足説明】

ディスパッチ保留状態の間のタスク状態を，図3-4の例を用いて説明する．タ

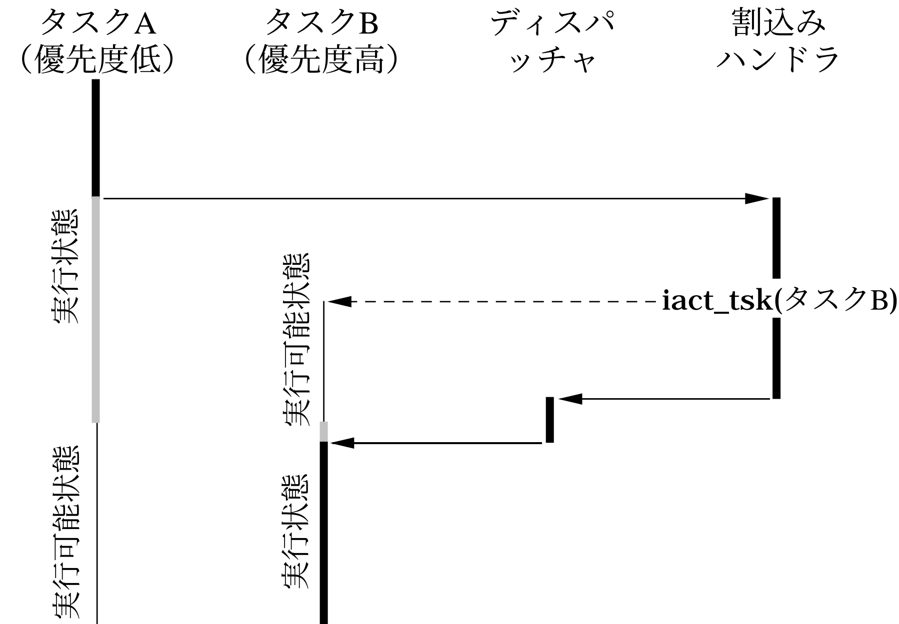


図3-4. ディスパッチ保留状態とタスク状態

スクAの実行中に発生した割込みによって起動された割込みハンドラから，タスクAよりも高い優先度を持つタスクBが起動された場合を考える．割込みハンドラの優先順位はディスパッチャの優先順位より高いため，割込みハンドラが実行されている間はディスパッチ保留状態であり，ディスパッチは起こらない．割込みハンドラの実行が終了すると，ディスパッチャを実行し，実行するタスクをタスクAからタスクBに切り替える．

割込みハンドラ内でタスクBが起動された後も，ディスパッチャが実行されるまでの間はタスクAが実行状態であり，タスクBは実行可能状態である．ディスパッチャの実行後は，タスクBが実行状態，タスクAが実行可能状態となる．なお，ディスパッチャの実行は不可分に行われるのが原則であり，その間のタスク状態は，この仕様上は規定されない．

3.6 非タスクコンテキストからのサービスコール呼出し

3.6.1 非タスクコンテキストから呼び出せるサービスコール

どのコンテキストからでも呼び出せる一部のサービスコールを除いては，非タスクコンテキストから呼び出せるサービスコールはサービスコール名に“i”の文字を付加することとし，タスクコンテキストから呼び出せるサービスコールと区別する．言い換えると，サービスコールは次の3つに分類することができる．

(a) 非タスクコンテキスト専用のサービスコール

サービスコール名が "i" で始まるサービスコールを，非タスクコンテキスト専用のサービスコールと呼び，名前の通り，非タスクコンテキストから呼び出すことができる．

非タスクコンテキスト専用のサービスコールがタスクコンテキストから呼び出された場合には，E_CTXエラーを返す．

【補足説明】

具体的には，次のサービスコールがこれに該当する．

iact_tsk	タスクの起動
iwup_tsk	タスクの起床
irel_wai	待ち状態の強制解除
iras_tex	タスク例外処理の要求
isig_sem	セマフォ資源の返却
iset_flg	イベントフラグのセット
ipsnd_dtq	データキューへの送信（ポーリング）
ifsnd_dtq	データキューへの強制送信
isig_tim	タイムティックの供給
irot_rdq	タスクの実行順位の回転
iget_tid	実行状態のタスクIDの参照
iloc_cpu	CPUロック状態への移行
iunl_cpu	CPUロック状態の解除

(b) どのコンテキストからでも呼び出せるサービスコール

サービスコール名が sns_*yyy* の形のサービスコールは，どのコンテキストからでも呼び出すことができる．すなわち，タスクコンテキスト，非タスクコンテキストのいずれからも呼び出すことができる．

【補足説明】

具体的には，次のサービスコールがこれに該当する．

sns_ctx	コンテキストの参照
sns_loc	CPUロック状態の参照
sns_dsp	ディスパッチ禁止状態の参照
sns_dpn	ディスパッチ保留状態の参照
sns_tex	タスク例外処理禁止状態の参照

(c) タスクコンテキスト専用のサービスコール

その他のサービスコールを，タスクコンテキスト専用のサービスコールと呼び，名前の通り，タスクコンテキストから呼び出すことができる．

タスクコンテキスト専用のサービスコールが非タスクコンテキストから呼び出された場合には，E_CTXエラーを返す．

【補足説明】

実装独自の拡張として，（同等の機能を持つ非タスクコンテキスト専用のサー

ビスコールが用意されていないサービスコールに対して）非タスクコンテキスト専用のサービスコールを追加すること，非タスクコンテキスト専用のサービスコールをタスクコンテキストからも呼び出せるようにすること，タスクコンテキスト専用のサービスコールを非タスクコンテキストからも呼び出せるようにすることは，いずれも許される．詳しくは，3.6.3節を参照すること．

【µITRON3.0仕様との相違】

非タスクコンテキスト専用のサービスコールを，“i” で始まる名称にすることを規定した．また，どのサービスコールに対してそれと同等の機能を持つ非タスクコンテキスト専用のサービスコールを用意するかを明確に規定し，機能コードは規定したサービスコールに対してのみ割り付けた．そのため，µITRON3.0 仕様で機能コードを割り付けた非タスクコンテキスト専用のサービスコールのいくつかに対して，µITRON4.0仕様では機能コードを割り付けていない．

3.6.2 サービスコールの遅延実行

長い割込み禁止区間を設けずにサービスコールの不可分性を保証することを可能にするために，非タスクコンテキストから呼び出されたサービスコールの実行を，最大限，ディスパッチャよりも高い優先順位を持つ処理の実行が終わるまで遅延することができる．これを，サービスコールの遅延実行と呼ぶ．

ただし，次のサービスコールを遅延実行することは許されない．

iget_tid	実行状態のタスクIDの参照
iloc_cpu	CPUロック状態への移行
iunl_cpu	CPUロック状態の解除
sns_ctx	コンテキストの参照
sns_loc	CPUロック状態の参照
sns_dsp	ディスパッチ禁止状態の参照
sns_dpn	ディスパッチ保留状態の参照
sns_tex	タスク例外処理禁止状態の参照

サービスコールを遅延実行する場合にも，遅延実行することを許されないものを除いて，サービスコールの実行順序がサービスコールの呼び出された順序に一致しなければならない．

遅延実行を認めた結果，非タスクコンテキストから呼び出したサービスコールに対して，操作対象のオブジェクトの状態に依存して発生するエラーをサービスコール呼出し時に検出できなくなり，遅延実行しない場合と同じエラーコードを返せなくなる場合がある．このような場合には，遅延実行しない場合に返るエラーコードに代えて，E_OKを返すことができる．実際にどのようなエラー時にエラーコードを返すことを省略できるかは，サービスコール毎に定める．

サービスコールを遅延実行するために，遅延実行すべきサービスコールを記憶しておく必要がある．そのためのメモリ領域が不足した場合には，サービス

コールはE_NOMEMエラーを返さなければならない．

【補足説明】

サービスコールの実行を実際にどの時点まで遅延させるかは，仕様において規定された振舞いと同じ振舞いをする限りは自由である．具体的には，ディスパッチ保留状態で呼び出されたサービスコールの実行を，ディスパッチが起こる状態となるまで遅延できる場合がある．ただしiras_texについては，ディスパッチ禁止状態でも実行しなければならない場合があるので，注意が必要である（詳しくは，iras_texの補足説明を参照）．

遅延実行されるサービスコールがE_OKを返した場合，後でそのサービスコールの処理を呼び出すことを保証しなければならない．

【μITRON3.0仕様との相違】

サービスコールの遅延実行に関して明確に規定することとした．

3.6.3 呼び出せるサービスコールの追加

μITRON4.0仕様に*xxx_yyy*（または*zxxx_yyy*）という名称のタスクコンテキスト専用のサービスコールが用意されている場合で，非タスクコンテキストから呼び出せるそれと同等の機能のサービスコールを実装独自に追加する場合には，実装独自に追加するサービスコール名称の原則（サービスコール名称の前に“v”を付加する）にかかわらず，i*xxx_yyy*（またはiz*xxx_yyy*）の名称とする．非タスクコンテキストから呼び出したサービスコールの遅延実行により，一部のエラーコードを返すことを省略する場合にも，同等の機能であるとみなす．

また，実装独自の拡張として，タスクコンテキスト専用のサービスコールをそのままの名称で非タスクコンテキストからも呼び出せるようにした場合，呼び出せるようにしたサービスコール名の前に“i”の文字を付加した名称のサービスコールを，非タスクコンテキストから呼び出せるようにしなければならない．逆に，非タスクコンテキスト専用のサービスコールをそのままの名称でタスクコンテキストからも呼び出せるようにした場合，呼び出せるようにしたサービスコール名の先頭の“i”を外した名称のサービスコールを，タスクコンテキストから呼び出せるようにしなければならない．

これらの規則は，実装独自のサービスコールにも同様に適用される．v*xxx_yyy*という名称のサービスコールを実装独自に用意した場合で，非タスクコンテキストからもそれと同等の機能を持つサービスコールを呼び出せるようにする場合には，少なくともiv*xxx_yyy*という名称で呼び出せるようにしなければならない．

【補足説明】

3.6.1節で，タスクコンテキスト専用のサービスコールが非タスクコンテキストから呼び出された場合には，E_CTXエラーを返すと規定している．それにもかかわらず，実装独自の拡張として，タスクコンテキスト専用のサービスコールを非タスクコンテキストからも呼び出せるようにすることが許されるのは，次

の理由による．

E_CTX エラーの検出は，実装定義で省略することができる（2.3.2 節）．一方，エラーの検出を省略したことにより，本来検出すべきエラーを検出できなかった場合の振舞いは，仕様上は未定義である（2.1.6 節）．仕様上は未定義の事項を実装独自に規定することは許されるため，タスクコンテキスト専用のサービスコールが非タスクコンテキストから呼び出された場合の振舞いを，実装独自に定めることは許される．そのため，実装独自の拡張として，タスクコンテキスト専用のサービスコールを非タスクコンテキストから呼び出せるようにすることは許される．

非タスクコンテキスト専用のサービスコールをタスクコンテキストから呼び出せるようにすることが許されるのも，同じ理由による．

3.7 システム初期化手順

システム初期化手順は，次のようにモデル化される（図3-5）．これはあくまでも概念モデルを示すものであり，実際のシステム初期化手順は，この概念モデルと同じ振舞いをする限りは，実装依存に最適化することができる．

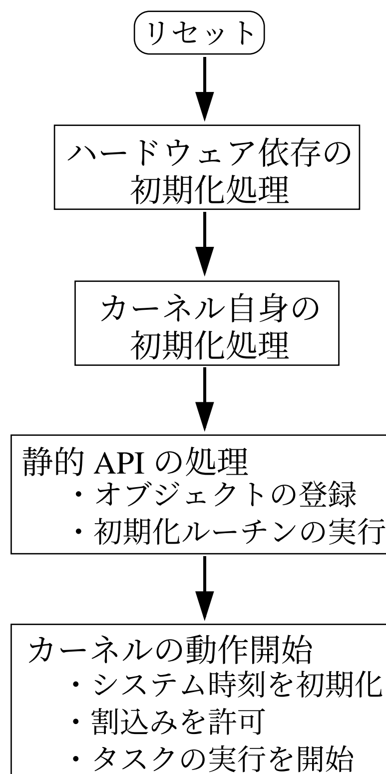


図3-5. システム初期化手順

システムがリセットされると，最初に，ハードウェア依存の初期化処理を行う．ハードウェア依存の初期化処理はカーネルの管理外であり，アプリケー

ションで用意する．ハードウェア依存の初期化処理の最後で，カーネルの初期化処理を呼び出す. カーネルの初期化処理を呼び出す方法は, 実装定義である.

カーネルの初期化処理が呼び出されると，まずカーネル自身の初期化処理（カーネル内のデータ構造の初期化など）を行い，次に静的APIの処理（オブジェクトの登録など）を行う．ATT_INI以外の静的APIの処理は，システムコンフィギュレーションファイル中に記述された順序で行う．静的APIの処理中にエラーを検出した場合の扱いについては，実装定義とする．

静的APIの処理には，初期化ルーチンの実行も含まれる．初期化ルーチンはアプリケーションで用意され，ATT_INIを使ってカーネルに登録される．初期化ルーチンは，カーネルの管理外の割込みを除くすべての割込みを禁止した状態で実行する．カーネルの管理外の割込みを禁止するかどうかは，実装定義である．また，初期化ルーチン内でサービスコールを呼び出せるか否か，呼び出せる場合にはどのようなサービスコールが呼び出せるかも，実装定義である．初期化ルーチンは，システムコンフィギュレーションファイル中にATT_INIの記述のある順序で実行する．初期化ルーチンの実行と，他の静的APIの処理との順序関係は，実装依存である．

静的APIの処理が終了すると，カーネルの動作を開始する．すなわち，タスクの実行を開始する．また，この時点で初めて割込みを許可する．システム時刻は，この時点を時刻0とする．

3.8 オブジェクトの登録とその解除

ID番号で識別されるオブジェクトは，オブジェクトを生成する静的API（CRE_*YYY*）またはサービスコール（cre_*yyy*）によってカーネルに登録され，オブジェクトを削除するサービスコール（del_*yyy*）によってその登録が解除される．オブジェクトが削除された後は，同じID番号で新しくオブジェクトを生成することができる．オブジェクト生成時には，生成するオブジェクトのID番号と生成に必要な情報を指定する．オブジェクト削除時には，削除するオブジェクトのID番号を指定する．

カーネルに登録できるオブジェクトの最大数やID番号の範囲は，実装定義である．また，サービスコールを用いて生成できるオブジェクトの最大数やID番号の範囲の指定方法も，実装定義である．

オブジェクトを追加する静的API（ATT_*YYY*）は，オブジェクトのID番号を指定せずにオブジェクトを生成し，カーネルに追加登録する．生成したオブジェクトはID番号を持たないため，この方法で登録したオブジェクトをID番号を指定して操作することはできない．そのため，この方法で生成したオブジェクトは削除することができない．

ID番号の自動割付けを行うオブジェクト生成のサービスコール（acre_*yyy*）は，生成するオブジェクトのID番号を，オブジェクトが登録されていないID番号の中から割り付ける．生成したオブジェクトに割り付けたID番号は，サービ

スコールの返値としてアプリケーションに返される．サービスコールの返値が負の値の場合はエラーの発生を意味するため，この方法で割り付けるID番号は正の値に限定される．割り付けることができるID番号がない場合には，サービスコールはE_NOIDエラーを返す．

自動割付けされるID番号の範囲の指定方法は，実装定義である．また，オブジェクトが登録されていないID番号の中から，生成するオブジェクトにID番号を割り付ける方法は，実装依存とする．

同期・通信オブジェクトの削除は，そのオブジェクトにおいて条件成立を待っているタスクがある場合にも行うことができる．この場合，削除されるオブジェクトにおいて条件成立を待っているタスクは待ち解除する．また，待ち解除されたタスクに対しては，待ち状態に入ったサービスコールの返値としてE_DLTエラーを返す．条件成立を待っているタスクが複数ある場合には，同期・通信オブジェクトの待ち行列につながれていた順序で待ち解除する．そのため，実行可能状態に移行したタスクで同じ優先度を持つものの間では，待ち行列の中で前につながれていたタスクの方が高い優先順位を持つことになる．同期・通信オブジェクトが複数の待ち行列を持つ場合には，異なる待ち行列で条件成立を待っていたタスクの間では，待ち解除する順序は実装依存である．

【スタンダードプロファイル】

スタンダードプロファイルでは，少なくとも1～255の範囲のID番号をサポートしなければならない．また，スタンダードプロファイルでサポートしなければならないID番号で識別されるオブジェクトについては，少なくとも255個のオブジェクトを登録できることが必要である．

3.9 処理単位の記述形式

μITRON4.0仕様では，割込みサービスルーチン，タイムイベントハンドラ（周期ハンドラ，アラームハンドラ，オーバランハンドラ），拡張サービスコールルーチン，タスク，タスク例外処理ルーチンの各処理単位をC言語で記述する場合の形式を規定する．処理単位をカーネルに登録する際のオブジェクト属性にTA_HLNG（高級言語用のインタフェースで処理単位を起動）が指定された場合には，規定された形式で記述されているものと仮定して処理単位を起動する．

それに対して，これらの処理単位をアセンブリ言語で記述する方法については，実装定義とする．処理単位をカーネルに登録する際のオブジェクト属性にTA_ASM（アセンブリ言語用のインタフェースで処理単位を起動）が指定された場合には，実装毎に規定された形式で記述されているものと仮定して処理単位を起動する．

割込みハンドラおよびCPU例外ハンドラを記述する方法と，これらをカーネルに登録する場合のオブジェクト属性の指定方法については実装定義とし，μITRON4.0仕様では規定しない．

【補足説明】

μITRON4.0仕様では，割込みハンドラからリターンするためのサービスコール（従来の仕様では，ret_int）について規定していない．これは，割込みハンドラの記述方法を実装定義としたためで，従来の仕様でret_intが行っていた処理が不要になったわけではない．実装によっては，割込みハンドラを記述するために，ret_intに相当するサービスコールが用意される場合もある．CPU例外ハンドラからのリターンについても，これと同様である．

また μITRON4.0 仕様では，タイムイベントハンドラからリターンするためのサービスコール（従来の仕様では，ret_tmr）についても規定していない．これは，C言語でタイムイベントハンドラを記述する場合には，関数からの単なるリターンによりタイムイベントハンドラからリターンできるものと定めたためで，従来の仕様でret_tmrが行っていた処理が不要になったわけではない．実装によっては，アセンブリ言語でタイムイベントハンドラを記述するために，ret_tmr に相当するサービスコールが用意される場合もある．割込みサービスルーチン，拡張サービスコールルーチン，タスク例外処理ルーチンからのリターンについても，これと同様である．

【μITRON3.0仕様との相違】

μITRON4.0仕様では，各処理単位をC言語で記述する場合の形式を規定し，処理単位からリターンするためのサービスコール（ret_*yyy*）については，処理単位をアセンブリ言語で記述する場合にのみ必要なものであることから，規定しないこととした．

3.10 カーネル構成定数／マクロ

カーネル構成定数／マクロは，アプリケーションプログラムの移植性を向上させることを目的に，アプリケーションプログラムからカーネルの構成を参照するために用いるものである．カーネル構成定数／マクロは，アプリケーションプログラムから参照できるようになっていれば，どのように定義するかは実装依存である．

カーネル構成定数／マクロは，それに関連する機能をサポートしていない場合には定義しない．

【補足説明】

カーネル構成定数／マクロを定義する方法として，カーネルのヘッダファイル中で固定的に定義する方法や，コンフィギュレータによって定義を生成する方法が考えられる．また，アプリケーションが用意するヘッダファイルでカーネル構成定数／マクロを定義し，その情報を使ってカーネルを生成する方法もある．

【μITRON3.0仕様との相違】

カーネル構成定数／マクロは，μITRON4.0仕様において新たに導入した機能で

ある．

3.11 カーネル共通定義

3.11.1 カーネル共通定数

(1) オブジェクト属性

TA_HLNG	0x00	高級言語用のインタフェースで処理単位を起動
TA_ASM	0x01	アセンブリ言語用のインタフェースで処理単位を起動
TA_TFIFO	0x00	タスクの待ち行列をFIFO順に
TA_TPRI	0x01	タスクの待ち行列をタスクの優先度順に
TA_MFIFO	0x00	メッセージのキューをFIFO順に
TA_MPRI	0x02	メッセージのキューをメッセージの優先度順に

【μITRON3.0仕様との相違】

TA_HLNGとTA_ASMの値を逆にした．

(2) カーネルで用いるメインエラーコード

カーネルでは，2.3.2節に規定されているメインエラーコードの中で，E_CLS，E_WBLK，E_BOVRの3つを除いたメインエラーコードを用いる．

【スタンダードプロファイル】

スタンダードプロファイルで発生し，検出しなければならないメインエラーコードは次の通りである．

E_OBJ	オブジェクト状態エラー
E_QOVR	キューイングオーバフロー
E_RLWAI	待ち状態の強制解除
E_TMOUT	ポーリング失敗またはタイムアウト

スタンダードプロファイルに準拠したカーネル上でポータビリティを確保したいアプリケーションは，上に挙げた以外のメインエラーコードが検出されることに依存してはならない．

【補足説明】

スタンダードプロファイルで発生しないか，検出を省略することができるメインエラーコードは，次の通りである．

(a) カーネルでは使用しないもの

E_CLS，E_WBLK，E_BOVR

(b) スタンダードプロファイルの機能では発生しないもの

E_OACV，E_NOID，E_NOEXS，E_DLT

(c) 発生するかどうかは実装に依存するもの

E_SYS，E_RSFN，E_NOMEM

(d) 検出を省略することができるもの

E_NOSPT，E_RSATR，E_PAR，E_ID，E_CTX，E_MACV，E_ILUSE

(3) サービスコールの機能コード

カーネルのサービスコールには，(–0xe0) ～ (–0x05) の範囲の機能コードを割り付ける．ただし，cal_svcには機能コードを割り付けない．機能コードの具体的な割付けは，第4章において機能単位毎に規定する．

(–0xe0) ～ (–0x05) の範囲の機能コードで，この仕様で割り付けられていないものは，将来のカーネルの機能拡張のために予約されている．(–0x100) ～ (–0xe1) の範囲の機能コードは，実装独自のサービスコールに用いることができる．(–0x200) ～ (–0x101) の範囲の機能コードは，将来のカーネルの機能拡張のために予約されているが，必要であれば実装独自のサービスコールに用いてもよいものとする．

【μITRON3.0仕様との相違】

機能コードの値を割り付けなおした．

【仕様決定の理由】

スタンダードプロファイルに含まれるサービスコールの機能コードを (–0x80) ～ (–0x05) の範囲としているが，これは，8ビットで表現できるようにするためである．

(4) その他のカーネル共通定数

TSK_SELF	0	自タスク指定
TSK_NONE	0	該当するタスクが無い

【μITRON3.0仕様との相違】

TSK_NONEを新たに導入した．μITRON3.0仕様では．該当するタスクが無い場合にはFALSE（＝0）を用いている．

3.11.2 カーネル共通構成定数

(1) 優先度の範囲

TMIN_TPRI	タスク優先度の最小値（＝1）
TMAX_TPRI	タスク優先度の最大値
TMIN_MPRI	メッセージ優先度の最小値（＝1）
TMAX_MPRI	メッセージ優先度の最大値

【スタンダードプロファイル】

スタンダードプロファイルでは，これらのカーネル構成定数を定義しなければ

ならない．また，TMAX_TPRIは16以上，TMAX_MPRIはTMAX_TPRI以上でなければならない．

【補足説明】

TMIN_TPRIとTMIN_MPRIは，この仕様上は1に固定されているが，実装独自の拡張として，1以外の値でカーネルを構成できるようにすることは許される．

(2) バージョン情報

TKERNEL_MAKER	カーネルのメーカコード
TKERNEL_PRID	カーネルの識別番号
TKERNEL_SPVER	ITRON仕様のバージョン番号
TKERNEL_PRVER	カーネルのバージョン番号

【スタンダードプロファイル】

スタンダードプロファイルでは，これらのカーネル構成定数を定義しなければならない．

【補足説明】

バージョン情報をあらわす定数値の意味については，ref_verの機能説明を参照すること．

第4章 μITRON4.0仕様の機能

4.1 タスク管理機能

タスク管理機能は，タスクの状態を直接的に操作／参照するための機能である．タスクを生成／削除する機能，タスクを起動／終了する機能，タスクに対する起動要求をキャンセルする機能，タスクの優先度を変更する機能，タスクの状態を参照する機能が含まれる．タスクはID番号で識別されるオブジェクトである．タスクのID番号をタスクIDと呼ぶ．タスク状態とスケジューリング規則については，3.2節を参照すること．

タスクは，実行順序を制御するために，ベース優先度と現在優先度を持つ．単にタスクの優先度といった場合には，タスクの現在優先度を指す．タスクのベース優先度は，タスクの起動時にタスクの起動時優先度に初期化する．ミューテックス機能を使わない場合には，タスクの現在優先度は常にベース優先度に一致している．そのため，タスク起動直後の現在優先度は，タスクの起動時優先度になっている．ミューテックス機能を使う場合に現在優先度がどのように設定されるかについては，4.5.1節で述べる．

タスクに対する起動要求は，キューイングされる．すなわち，すでに起動されているタスクを再度起動しようとすると，そのタスクを起動しようとしたという記録が残り，後でそのタスクが終了した時に，タスクを自動的に再起動する．ただし，起動コードを指定してタスクを起動するサービスコール（sta_tsk）では，起動要求はキューイングされない．タスクに対する起動要求のキューイングを実現するために，タスクは起動要求キューイング数を持つ．タスクの起動要求キューイング数は，タスクの生成時に0にクリアする．

タスクを起動する際には，そのタスクの拡張情報（exinf）をパラメータとして渡す．ただし，起動コードを指定してタスクを起動するサービスコール（sta_tsk）によってタスクが起動された場合には，拡張情報に代えて，指定された起動コードを渡す.

カーネルは，タスクの終了時に，ミューテックスのロック解除を除いては，タスクが獲得した資源（セマフォ資源，メモリブロックなど）を解放する処理を行わない．タスク終了時に資源を解放するのは，アプリケーションの責任である．

タスクの生成時，起動時，終了時，削除時には，それぞれ次のような処理を行う必要がある．タスクの生成時に行うべき処理には，タスクの起動要求キューイング数のクリア，タスク例外処理ルーチンの定義されていない状態への初期化（4.3節参照），上限プロセッサ時間の設定されていない状態への初期化（4.7.4節参照）が含まれる．タスクの起動時に行うべき処理には，タスクのベース優先度（および現在優先度）の初期化，起床要求キューイング数のクリア（4.2節参照），強制待ち要求ネスト数のクリア（4.2節参照），保留例外要因のクリ

ア（4.3節参照），タスク例外処理禁止状態への設定（4.3節参照）が含まれる．タスクの終了時に行うべき処理には，タスクがロックしているミューテックスのロック解除（4.5.1節参照），タスクの上限プロセッサ時間の設定の解除（4.7.4節参照）が含まれる．タスクの削除時に行うべき処理には，タスクのスタック領域の解放（スタック領域をカーネルで確保した場合のみ）が含まれる．

タスクのC言語による記述形式は次の通りとする．

```
void task ( VP_INT exinf )
{
        タスク本体
        ext_tsk ( ) ;
}
```

タスクのメインルーチンからリターンした場合には，ext_tskを呼び出した場合と同じ振舞いをする（すなわちタスクを終了する）．また，exd_tskを呼び出して，タスクの終了と同時にタスクを削除することもできる．

タスク管理機能に関連して，次のカーネル構成定数を定義する．

TMAX_ACTCNT　　タスクの起動要求キューイング数の最大値

タスク生成情報およびタスク状態のパケット形式として，次のデータ型を定義する．

```
typedef struct t_ctsk {
        ATR         tskatr ;        /* タスク属性 */
        VP_INT      exinf ;         /* タスクの拡張情報 */
        FP          task ;          /* タスクの起動番地 */
        PRI         itskpri ;       /* タスクの起動時優先度 */
        SIZE        stksz ;         /* タスクのスタックサイズ（バイト
                                       数）*/
        VP          stk ;           /* タスクのスタック領域の先頭番地 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_CTSK ;

typedef struct t_rtsk {
        STAT        tskstat ;       /* タスク状態 */
        PRI         tskpri ;        /* タスクの現在優先度 */
        PRI         tskbpri ;       /* タスクのベース優先度 */
        STAT        tskwait ;       /* 待ち要因 */
        ID          wobjid ;        /* 待ち対象のオブジェクトのID番号 */
        TMO         lefttmo ;       /* タイムアウトするまでの時間 */
        UINT        actcnt ;        /* 起動要求キューイング数 */
        UINT        wupcnt ;        /* 起床要求キューイング数 */
        UINT        suscnt ;        /* 強制待ち要求ネスト数 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RTSK ;

typedef struct t_rtst {
        STAT        tskstat ;       /* タスク状態 */
        STAT        tskwait ;       /* 待ち要因 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RTST ;
```

タスク管理機能の各サービスコールの機能コードは次の通りである．

TFN_CRE_TSK	–0x05	cre_tskの機能コード
TFN_ACRE_TSK	–0xc1	acre_tskの機能コード
TFN_DEL_TSK	–0x06	del_tskの機能コード
TFN_ACT_TSK	–0x07	act_tskの機能コード
TFN_IACT_TSK	–0x71	iact_tskの機能コード
TFN_CAN_ACT	–0x08	can_actの機能コード
TFN_STA_TSK	–0x09	sta_tskの機能コード
TFN_EXT_TSK	–0x0a	ext_tskの機能コード
TFN_EXD_TSK	–0x0b	exd_tskの機能コード
TFN_TER_TSK	–0x0c	ter_tskの機能コード
TFN_CHG_PRI	–0x0d	chg_priの機能コード
TFN_GET_PRI	–0x0e	get_priの機能コード
TFN_REF_TSK	–0x0f	ref_tskの機能コード
TFN_REF_TST	–0x10	ref_tstの機能コード

【スタンダードプロファイル】

スタンダードプロファイルでは，タスクを動的に生成／削除する機能（cre_tsk，acre_tsk，del_tsk），起動コードを指定してタスクを起動する機能（sta_tsk），タスクを終了と同時に削除する機能（exd_tsk），タスクの状態を参照する機能（ref_tsk，ref_tst）を除いて，タスク管理機能をサポートしなければならない．

スタンダードプロファイルでは，少なくとも1回のタスクの起動要求キューイングをサポートしなければならない．したがって，TMAX_ACTCNTは1以上でなければならない．

【補足説明】

タスクを実行するコンテキストと状態については，次のように整理できる．

- タスクは，それぞれ独立したコンテキストで実行する（3.5.1節参照）．タスクを実行するコンテキストは，タスクコンテキストに分類される（3.5.2節参照）．
- タスクの優先順位は，ディスパッチャの優先順位よりも低い（3.5.3節参照）．
- タスクの実行開始直後は，CPUロック解除状態かつディスパッチ許可状態になっている．自タスクを終了する際には，CPUロック解除状態かつディスパッチ許可状態にしなければならない（3.5.4節と3.5.5節を参照）．

タスクの起動要求キューイングをサポートしない場合には，TMAX_ACTCNTは0とする．

【μITRON3.0仕様との相違】

タスクに対して直接操作を行う機能で，広義の待ち状態とは関係しないもののみをタスク管理機能に分類することにした．具体的には，タスクの優先順位を回転する機能（rot_rdq），実行状態のタスクIDを参照する機能（get_tid），タスクディスパッチを禁止／許可する機能（dis_dsp，ena_dsp）をシステム状態管

理機能に分類した．また，タスクの待ち状態を強制解除する機能（rel_wai）をタスク付属同期機能に分類した．

タスクに対する起動要求をキューイングする機能を追加し，それをサポートするためのサービスコール（act_tsk，can_act）を新設した．起動コードを指定してタスクを起動するサービスコール（sta_tsk）は，μITRON3.0仕様との互換性のために残した（スタンダードプロファイル外）．

ミューテックスの導入に伴って，タスクのベース優先度の概念を新たに導入した．ただし，ミューテックス機能を使わない場合には，タスクの現在優先度は常にベース優先度に一致しており，実質的な変化はない．

タスクのメインルーチンからのリターンによっても，タスクを終了させることができることとした．

CRE_TSK	タスクの生成（静的API）	【S】
cre_tsk	タスクの生成	
acre_tsk	タスクの生成（ID番号自動割付け）	

【静的API】

CRE_TSK (ID tskid, { ATR tskatr, VP_INT exinf, FP task, PRI itskpri,
SIZE stksz, VP stk }) ;

【C言語API】

ER ercd = cre_tsk (ID tskid, T_CTSK *pk_ctsk) ;
ER_ID tskid = acre_tsk (T_CTSK *pk_ctsk) ;

【パラメータ】

ID	tskid	生成対象のタスクのID番号（acre_tsk以外）
T_CTSK *	pk_ctsk	タスク生成情報を入れたパケットへのポインタ（CRE_TSKではパケットの内容を直接記述する）

pk_ctskの内容（T_CTSK型）

ATR	tskatr	タスク属性
VP_INT	exinf	タスクの拡張情報
FP	task	タスクの起動番地
PRI	itskpri	タスクの起動時優先度
SIZE	stksz	タスクのスタック領域のサイズ（バイト数）
VP	stk	タスクのスタック領域の先頭番地

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cre_tskの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acre_tskの場合

ER_ID	tskid	生成したタスクの ID 番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（tskidが不正あるいは使用できない；cre_tskのみ）
E_NOID	ID番号不足（割付け可能なタスクIDがない；acre_tskのみ）
E_NOMEM	メモリ不足（スタック領域などが確保できない）
E_RSATR	予約属性（tskatrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_ctsk, task, itskpri, stksz, stkが不正）
E_OBJ	オブジェクト状態エラー（対象タスクが登録済み；

cre_tskのみ）

【機能】

tskidで指定されるID番号を持つタスクを, pk_ctskで指定されるタスク生成情報に基づいて生成する．具体的には，対象タスクを未登録状態から休止状態または実行可能状態に移行させ，それに伴って必要な処理を行う．tskatrはタスクの属性，exinfはタスクを起動する際にパラメータとして渡す拡張情報，taskはタスクの起動番地，itskpriはタスク起動時のベース優先度の初期値を指定する起動時優先度，stkszはタスクのスタック領域のサイズ（バイト数），stkはタスクのスタック領域の先頭番地である．

CRE_TSKにおいては，tskidは自動割付け対応整数値パラメータ，tskatrはプリプロセッサ定数式パラメータである．

acre_tskは，生成するタスクのID番号をタスクが登録されていないID番号の中から割り付け，割り付けたID番号を返値として返す．

tskatrには,（(TA_HLNG ‖ TA_ASM）｜[TA_ACT]）の指定ができる. TA_HLNG（＝0x00）が指定された場合には高級言語用のインタフェースで，TA_ASM（＝0x01）が指定された場合にはアセンブリ言語用のインタフェースでタスクを起動する．TA_ACT（＝0x02）が指定されない場合には，対象タスクを休止状態に移行させ，タスクの生成時に行うべき処理を行う．TA_ACTが指定された場合には，対象タスクを実行可能状態に移行させ，タスクの生成時に行うべき処理に加えて，タスクの起動時に行うべき処理を行う．タスクを起動する際のパラメータとしては，タスクの拡張情報を渡す．

stkで指定された番地からstkszバイトのメモリ領域を, タスクを実行するためのスタック領域として使用する．stkにNULL（＝0）が指定された場合には，stkszで指定されたサイズのメモリ領域を，カーネルが確保する．

stkszに実装定義の最大値よりも大きい値が指定された場合には, E_PARエラーを返す．

【スタンダードプロファイル】

スタンダードプロファイルでは，tskatrにTA_ASMが指定された場合の機能はサポートする必要がない．また，stkにNULL以外の値が指定された場合の機能もサポートする必要がない．

【補足説明】

タスクから呼び出されたサービスコールやタスクの実行中に起動された割込みハンドラ（ないしは，それらの出入口処理）がタスクと一連のスタック領域を使う場合，それらの使うスタック領域もタスクのスタック領域に含まれる．タスクのスタック領域のサイズを求める方法は，実装について説明する製品マニュアルなどで説明すべきである．

タスクのスタック領域の先頭番地には，スタックとして使用するメモリ領域の中での最小番地を指定する．そのため，タスクの実行開始直後のスタックポインタの値とは，一般には一致しない．

cre_tskの対象タスクに自タスクを指定することはできない．自タスクが指定された場合には，未登録状態でないためにE_OBJエラーを返す．

stkにNULLが指定された場合にカーネルが確保するスタック領域のサイズは，stkszに指定されたサイズ以上であれば，それよりも大きくてもよい．

【μITRON3.0仕様との相違】

タスク生成情報に，タスクのスタック領域の先頭番地（stk）を追加した．μITRON3.0仕様との互換性を保つためには，stkにNULLを指定すればよい．

タスク生成情報を入れたパケット中でのtskatrとexinfの順序を交換した．また，exinfのデータ型をVPからVP_INTに，stkszのデータ型をINTからSIZEにそれぞれ変更した．

タスク属性（TA_ACT）により，タスク生成後にタスクを実行可能状態とする機能を追加した．これは，タスクを静的に生成する場合を考慮したためである．タスクがコプロセッサを使用することを示すタスク属性は規定しないことにした（必要なら実装独自に対応できる）．

acre_tskは新設のサービスコールである．

del_tsk　タスクの削除

【C言語API】

ER ercd = del_tsk (ID tskid) ;

【パラメータ】

ID	tskid	削除対象のタスクのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（tskidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象タスクが未登録）
E_OBJ	オブジェクト状態エラー（対象タスクが休止状態でない）

【機能】

tskidで指定されるタスクを削除する．具体的には，対象タスクを休止状態から未登録状態に移行させ，タスクの削除時に行うべき処理を行う．

対象タスクが休止状態でない場合には，E_OBJエラーを返す（ただし，対象タスクが未登録状態の時はE_NOEXSエラーとなる）．

【補足説明】

対象タスクに自タスクを指定することはできない．自タスクが指定された場合には，休止状態でないためにE_OBJエラーを返す．自タスクを削除する場合には，exd_tskを用いる．

act_tsk タスクの起動 【S】
iact_tsk 【S】

【C言語API】

```
ER ercd = act_tsk ( ID tskid ) ;
ER ercd = iact_tsk ( ID tskid ) ;
```

【パラメータ】

ID	tskid	起動対象のタスクのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（tskidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象タスクが未登録）
E_QOVR	キューイングオーバフロー（起動要求キューイング数のオーバフロー）

【機能】

tskidで指定されるタスクを起動する．具体的には，対象タスクを休止状態から実行可能状態に移行させ，タスクの起動時に行うべき処理を行う．タスクを起動する際のパラメータとして，タスクの拡張情報を渡す．

対象タスクが休止状態でない場合には，タスクに対する起動要求をキューイングする（ただし，対象タスクが未登録状態の時はE_NOEXSエラーとなる）．具体的には，タスクの起動要求キューイング数に1を加える．タスクの起動要求キューイング数に1を加えると起動要求キューイング数の最大値を越える場合には，E_QOVRエラーを返す．

非タスクコンテキストから呼び出された場合で，サービスコールを遅延実行する場合には，E_QOVRエラーを返すことを，実装定義で省略することができる．

tskidにTSK_SELF（＝0）が指定されると，自タスクを対象タスクとする．ただし，非タスクコンテキストからの呼出しでこの指定が行われた場合には，E_IDエラーを返す．

【補足説明】

スタンダードプロファイルでは，タスクの起動要求キューイング数の最大値は，1以上であればいくつであってもよい．したがって，スタンダードプロファイルに準拠したカーネルで，起動要求がキューイングされているタスクを指定してこのサービスコールを呼び出しても，E_QOVRエラーが返るとは限らない．

【μITRON3.0仕様との相違】

新設のサービスコールである．

can_act　　タスク起動要求のキャンセル　　【S】

【C言語API】

ER_UINT actcnt = can_act (ID tskid) ;

【パラメータ】

ID	tskid	起動要求のキャンセル対象のタスクのID番号

【リターンパラメータ】

ER_UINT	actcnt	キューイングされていた起動要求の回数（正の値または0）またはエラーコード

【エラーコード】

E_ID	不正ID番号（tskidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象タスクが未登録）

【機能】

tskidで指定されるタスクに対してキューイングされている起動要求をキャンセルし，キューイングされていた起動要求の回数を返す．具体的には，タスクの起動要求キューイング数を0にクリアし，クリアする前の起動要求キューイング数を返す．

tskidにTSK_SELF（＝0）が指定されると，自タスクを対象タスクとする．

【補足説明】

休止状態のタスクを対象タスクとして呼び出すこともできる．この場合，休止状態のタスクには起動要求がキューイングされていないために，サービスコールの返値は0となる．

このサービスコールは，タスクを周期的に起動して処理を行う場合に，周期内に処理が終わっているかどうかを判定するために用いることができる．具体的には，前の起動要求に対する処理が終了した時点でcan_actを呼び出し，その返値が1以上の値であった場合，前の起動要求に対する処理が周期内に終了せず，次の起動要求が行われたことがわかる．したがって，処理の遅れに対する処置をとることができる．

【μITRON3.0仕様との相違】

新設のサービスコールである．

sta_tsk タスクの起動（起動コード指定） 【B】

【C言語API】

```
ER ercd = sta_tsk ( ID tskid, VP_INT stacd ) ;
```

【パラメータ】

ID	tskid	起動対象のタスクのID番号
VP_INT	stacd	タスクの起動コード

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（tskidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象タスクが未登録）
E_OBJ	オブジェクト状態エラー（対象タスクが休止状態でない）

【機能】

tskidで指定されるタスクを起動する．具体的には，対象タスクを休止状態から実行可能状態に移行させ，タスクの起動時に行うべき処理を行う．タスクを起動する際のパラメータとして，stacdの値を渡す．

対象タスクが休止状態でない場合には，タスクに対する起動要求をキューイングせず，E_OBJ エラーを返す（ただし，対象タスクが未登録状態の時はE_NOEXSエラーとなる）．

【補足説明】

対象タスクに自タスクを指定することはできない．自タスクが指定された場合には，休止状態でないためにE_OBJエラーを返す．

【μITRON3.0仕様との相違】

stacdのデータ型をINTからVP_INTに変更した．

ext_tsk 自タスクの終了 【S】【B】

【C言語API】

```
void ext_tsk ( ) ;
```

【パラメータ】

なし

【リターンパラメータ】

このサービスコールからはリターンしない

【機能】

自タスクを終了させる．具体的には，自タスクを実行状態から休止状態に移行させ，タスクの終了時に行うべき処理を行う．

自タスクに対する起動要求がキューイングされている場合，具体的には，自タスクの起動要求キューイング数が1以上の場合には，起動要求キューイング数から1を減じ，自タスクを実行可能状態に移行させる．この時，タスク起動時に行うべき処理を行う．タスクを起動する際のパラメータとしては，タスクの拡張情報を渡す．

このサービスコールは，リターンすることがないサービスコールである．したがって，サービスコール内で何らかのエラーを検出した場合でも，エラーコードを返すことはできない．サービスコール内のエラーを検出するか否かと，もし検出した場合の振る舞いは実装定義とする．

【補足説明】

このサービスコールは，自タスクに対する起動要求がキューイングされている場合，自タスクをいったん終了させた後に再起動する．そのため，自タスクがロックしているミューテックスをロック解除し，上限プロセッサ時間の設定を解除する．また，タスクのベース優先度（および現在優先度），起床要求キューイング数，強制待ち要求ネスト数，保留例外要因，タスク例外処理禁止／許可状態を，タスクの起動直後の状態に初期化する．再起動されたタスクの優先順位は，同じ優先度を持つタスクの中で最低の優先順位となる．

サービスコール内でエラーを検出した場合，検出したエラーに関する情報を，エラーログに残すなどの方法が考えられる．

タスクのメインルーチンからリターンした場合も，ext_tskを呼び出した場合と同じ振舞いをする．

【μITRON3.0仕様との相違】

起動要求をキューイングする機能を追加したため，このサービスコールは，自タスクを終了させた後に再起動する場合がある．

exd_tsk　自タスクの終了と削除

【C言語API】

void exd_tsk ( ) ;

【パラメータ】

なし

【リターンパラメータ】

このサービスコールからはリターンしない

【機能】

自タスクを終了させ，さらに自タスクを削除する．具体的には，自タスクを実行状態から未登録状態に移行させ，タスクの終了時に行うべき処理と削除時に行うべき処理を行う．

このサービスコールは，リターンすることがないサービスコールである．したがって，サービスコール内で何らかのエラーを検出した場合でも，エラーコードを返すことはできない．サービスコール内のエラーを検出するか否かと，もし検出した場合の振る舞いは実装定義とする．

【補足説明】

このサービスコールが呼び出されると，起動要求がキューイングされているかどうかにかかわらず，自タスクを終了させ，削除する（未登録状態では，起動要求キューイング数は意味を持たない）．

サービスコール内でエラーを検出した場合，検出したエラーに関する情報を，エラーログに残すなどの方法が考えられる．

ter_tsk　　タスクの強制終了　　【S】【B】

【C言語API】

ER ercd = ter_tsk (ID tskid) ;

【パラメータ】

ID	tskid	強制終了対象のタスクのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（tskidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象タスクが未登録）
E_ILUSE	サービスコール不正使用（対象タスクが自タスク）
E_OBJ	オブジェクト状態エラー（対象タスクが休止状態）

【機能】

tskidで指定されるタスクを強制的に終了させる．具体的には，対象タスクを休止状態に移行させ，タスクの終了時に行うべき処理を行う．

対象タスクに対する起動要求がキューイングされている場合，具体的には，対象タスクの起動要求キューイング数が1以上の場合には，起動要求キューイング数から1を減じ，対象タスクを実行可能状態に移行させる．この時，タスクの起動時に行うべき処理を行う．タスクを起動する際のパラメータとしては，タスクの拡張情報を渡す

対象タスクが休止状態の時はE_OBJエラーを返す．また，このサービスコールで自タスクを終了させることはできない．対象タスクが自タスクの場合には，E_ILUSEエラーを返す．

【補足説明】

このサービスコールは，対象タスクが広義の待ち状態にある場合にも，対象タスクを強制的に終了させる．対象タスクが何らかの待ち行列につながれていた場合には，対象タスクを待ち行列から外す．その際に，その待ち行列で待っている他のタスクの待ち解除が必要になる場合がある（snd_mbfの機能説明とget_mplの機能説明を参照）．

このサービスコールは，対象タスクに対する起動要求がキューイングされている場合，対象タスクをいったん終了させた後に再起動する．そのため，対象タスクがロックしているミューテックスをロック解除し，上限プロセッサ時間の設定を解除する．また，タスクのベース優先度（および現在優先度），起床要求キューイング数，強制待ち要求ネスト数，保留例外要因，タスク例外処理禁止／許可状態を，タスクの起動直後の状態に初期化する．再起動されたタスクの優先順位は，同じ優先度を持つタスクの中で最低の優先順位となる．

【μITRON3.0仕様との相違】

対象タスクに自タスクが指定された場合のエラーを, E_OBJからE_ILUSEに変更した.

起動要求をキューイングする機能を追加したため，このサービスコールは，対象タスクを終了させた後に再起動する場合がある.

chg_pri タスク優先度の変更 【S】【B】

【C言語API】

ER ercd = chg_pri (ID tskid, PRI tskpri) ;

【パラメータ】

ID	tskid	変更対象のタスクのID番号
PRI	tskpri	変更後のベース優先度

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（tskidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象タスクが未登録）
E_PAR	パラメータエラー（tskpriが不正）
E_ILUSE	サービスコール不正使用（上限優先度の違反）
E_OBJ	オブジェクト状態エラー（対象タスクが休止状態）

【機能】

tskidで指定されるタスクのベース優先度を，tskpriで指定される値に変更する．それに伴って，タスクの現在優先度も変更する．

tskidにTSK_SELF（＝0）が指定されると，自タスクを対象タスクとする．また，tskpriにTPRI_INI（＝0）が指定されると，対象タスクのベース優先度を，タスクの起動時優先度に変更する．

このサービスコールを実行した結果，対象タスクの現在優先度が変化した場合および現在優先度がベース優先度に一致している場合（ミューテックス機能を使わない場合には，この条件は常に成り立つ）には，次の処理を行う．対象タスクが実行できる状態である場合，タスクの優先順位を，変更後の優先度にしたがって変化させる．変更後の優先度と同じ優先度を持つタスクの間では，対象タスクの優先順位を最低とする．対象タスクが何らかのタスク優先度順の待ち行列につながれている場合にも，その待ち行列の中での順序を，変更後の優先度にしたがって変化させる．変更後の優先度と同じ優先度を持つタスクの間では，対象タスクを最後につなぐ．

対象タスクがTA_CEILING属性のミューテックスをロックしているか，ロックを待っている場合で，tskpriで指定されたベース優先度が，それらのミューテックスのいずれかの上限優先度よりも高い場合には，E_ILUSEエラーを返す．

【補足説明】

このサービスコールを呼び出した結果，対象タスクのタスク優先度順の待ち行列の中での順序が変化した場合，対象タスクないしはその待ち行列で待っている他のタスクの待ち解除が必要になる場合がある（snd_mbfの機能説明と

get_mplの機能説明を参照）.

対象タスクが，TA_INHERIT属性のミューテックスのロック待ち状態である場合，このサービスコールでベース優先度を変更したことにより，推移的な優先度継承の処理が必要になる場合がある．

ミューテックス機能を使わない場合には，対象タスクに自タスク，変更後の優先度に自タスクのベース優先度を指定してこのサービスコールが呼び出されると，自タスクの実行順位は同じ優先度を持つタスクの中で最低となる．そのため，このサービスコールを用いて，実行権の放棄を行うことができる．

【μITRON3.0仕様との相違】

ミューテックスの導入に伴って, chg_priはタスクのベース優先度を変更するものとした．tskpriにTPRI_INIを指定できる機能を，標準的な機能とした．

get_pri タスク優先度の参照 【S】

【C言語API】

ER ercd = get_pri (ID tskid, PRI *p_tskpri) ;

【パラメータ】

ID	tskid	参照対象のタスクのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード
PRI	tskpri	対象タスクの現在優先度

【エラーコード】

E_ID	不正ID番号（tskidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象タスクが未登録）
E_PAR	パラメータエラー（p_tskpriが不正）
E_OBJ	オブジェクト状態エラー（対象タスクが休止状態）

【機能】

tskidで指定されるタスクの現在優先度を参照し，tskpriに返す.

tskidにTSK_SELF（＝0）が指定されると，自タスクを対象タスクとする.

【補足説明】

chg_priがタスクのベース優先度を変更するのに対して，get_priはタスクの現在優先度を参照する.

【μITRON3.0仕様との相違】

新設のサービスコールである．このサービスコールを新設したのは，送信するメッセージの優先度に自タスクの優先度を設定する場合などに，自タスクの優先度を小さいオーバヘッドで取り出す方法が必要となるためである.

【仕様決定の理由】

tskpri をサービスコールの返値としなかったのは，他の類似のサービスコール（get_*yyy*）との整合性と，実装独自に優先度に負の値を使う場合を考慮したためである.

ref_tsk タスクの状態参照

【C言語API】

ER ercd = ref_tsk (ID tskid, T_RTSK *pk_rtsk) ;

【パラメータ】

ID	tskid	参照対象のタスクのID番号
T_RTSK *	pk_rtsk	タスク状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rtskの内容（T_RTSK型）

STAT	tskstat	タスク状態
PRI	tskpri	タスクの現在優先度
PRI	tskbpri	タスクのベース優先度
STAT	tskwait	待ち要因
ID	wobjid	待ち対象のオブジェクトのID番号
TMO	lefttmo	タイムアウトするまでの時間
UINT	actcnt	起動要求キューイング数
UINT	wupcnt	起床要求キューイング数
UINT	suscnt	強制待ち要求ネスト数

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（tskidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象タスクが未登録）
E_PAR	パラメータエラー（pk_rtskが不正）

【機能】

tskidで指定されるタスクに関する状態を参照し，pk_rtskで指定されるパケットに返す．対象タスクが未登録状態の場合には，E_NOEXSエラーを返す．

tskstatには，対象タスクの状態によって，次のいずれかの値を返す．

TTS_RUN	0x01	実行状態
TTS_RDY	0x02	実行可能状態
TTS_WAI	0x04	待ち状態
TTS_SUS	0x08	強制待ち状態
TTS_WAS	0x0c	二重待ち状態
TTS_DMT	0x10	休止状態

対象タスクが休止状態でない場合に，tskpriには対象タスクの現在優先度，tskbpriにはベース優先度を返す．対象タスクが休止状態の場合にこれらに返す値は実装依存である．

対象タスクが待ち状態（二重待ち状態を含む）の場合に，tskwaitには，対象タスクが待ち状態になっている要因によって，次のいずれかの値を返す．対象タスクが待ち状態でない場合にtskwaitに返す値は実装依存である．

TTW_SLP	0x0001	起床待ち状態
TTW_DLY	0x0002	時間経過待ち状態
TTW_SEM	0x0004	セマフォ資源の獲得待ち状態
TTW_FLG	0x0008	イベントフラグ待ち状態
TTW_SDTQ	0x0010	データキューへの送信待ち状態
TTW_RDTQ	0x0020	データキューからの受信待ち状態
TTW_MBX	0x0040	メールボックスからの受信待ち状態
TTW_MTX	0x0080	ミューテックスのロック待ち状態
TTW_SMBF	0x0100	メッセージバッファへの送信待ち状態
TTW_RMBF	0x0200	メッセージバッファからの受信待ち状態
TTW_CAL	0x0400	ランデブの呼出し待ち状態
TTW_ACP	0x0800	ランデブの受付待ち状態
TTW_RDV	0x1000	ランデブの終了待ち状態
TTW_MPF	0x2000	固定長メモリブロックの獲得待ち状態
TTW_MPL	0x4000	可変長メモリブロックの獲得待ち状態

対象タスクが待ち状態（二重待ち状態を含む）で，起床待ち状態，時間経過待ち状態，ランデブの終了待ち状態のいずれでもない場合に，wobjidには，待ち対象のオブジェクトのID番号を返す．それ以外の場合にwobjidに返す値は実装依存である．

対象タスクが待ち状態（二重待ち状態を含む）で，時間経過待ち状態でない場合に，lefttmoには，対象タスクがタイムアウトするまでの時間を返す．具体的には, タイムアウトとなる時刻から現在時刻を減じた値を返す. ただし, lefttmoに返す値は，タイムアウトするまでの時間以下でなければならない．そのため，次のタイムティックでタイムアウトする場合には，lefttmoに0を返す．対象タスクが永久待ち（タイムアウトなし）で待ち状態になっている場合には，lefttmoにTMO_FEVRを返す．対象タスクが待ち状態（二重待ち状態を含む）でないか, 時間経過待ち状態である場合には, lefttmoに返す値は実装依存である．

actcntには，対象タスクの起動要求キューイング数を返す．

対象タスクが休止状態でない場合に, wupcntには対象タスクの起床要求キューイング数，suscntには強制待ち要求ネスト数を返す．対象タスクが休止状態の場合にこれらに返す値は実装依存である．

tskidにTSK_SELF（＝0）が指定されると，自タスクを対象タスクとする．

【µITRON3.0仕様との相違】

実装依存で参照できるとしていた情報の多くを, 標準的に参照できるものとした．その際に，リターンパラメータの名称をwidからwobjidに変更した．また，参照できる情報に，タスクのベース優先度（tskbpri），タイムアウトするまでの時間（lefttmo），起動要求キューイング数（actcnt）を追加し，拡張情報を削

除した．

タスク状態を返すパケット中での tskstat と tskpri の順序を交換した．tskstat のデータ型をUINTからSTATに変更した．また，パラメータとリターンパラメータの順序を変更した．

特定のタスク状態の場合にのみ意味を持つリターンパラメータに，その他のタスク状態の場合に返す値を実装依存とした．例えば，タスクが休止状態の場合にtskpriに返す値は実装依存である．

tskwaitに返す値を割り付けなおした．

【仕様決定の理由】

対象タスクが時間経過待ち状態の場合に lefttmo に返す値を実装依存としたのは，dly_tskによる遅延時間がRELTIM型（符号無し整数）で指定されるために，TMO型（符号付き整数）のlefttmoに返せるとは限らないためである．

ref_tst　　タスクの状態参照（簡易版）

【C言語API】

ER ercd = ref_tst (ID tskid, T_RTST *pk_rtst) ;

【パラメータ】

ID	tskid	参照対象のタスクのID番号
T_RTST *	pk_rtst	タスク状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rtstの内容（T_RTST型）

STAT	tskstat	タスク状態
STAT	tskwait	待ち要因

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（tskidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象タスクが未登録）
E_PAR	パラメータエラー（pk_rtstが不正）

【機能】

tskidで指定されるタスクに関する最低限の状態を参照し，pk_rtstで指定されるパケットに返す.

このサービスコールは，ref_tskの簡易版である．tskstatとtskwaitには，ref_tskで返すのと同じ値を返す.

tskidにTSK_SELF（＝0）が指定されると，自タスクを対象タスクとする.

【仕様決定の理由】

ref_tskを用いると，タスクに関する各種の状態を参照できるが，タスク情報など最低限の状態のみを参照したい場合にはオーバヘッドが大きい．小さいオーバヘッドでタスクに関する最低限の状態を参照するためのサービスコールとして，ref_tstを新設した.

【μITRON3.0仕様との相違】

新設のサービスコールである.

4.2 タスク付属同期機能

タスク付属同期機能は，タスクの状態を直接的に操作することによって同期を行うための機能である．タスクを起床待ちにする機能とそこから起床する機能，タスクの起床要求をキャンセルする機能，タスクの待ち状態を強制解除する機能，タスクを強制待ち状態へ移行する機能とそこから再開する機能，自タスクの実行を遅延する機能が含まれる．

タスクに対する起床要求は，キューイングされる．すなわち，起床待ち状態でないタスクを起床しようとすると，そのタスクを起床しようとしたという記録が残り，後でそのタスクが起床待ちに移行しようとした時に，タスクを起床待ち状態にしない．タスクに対する起床要求のキューイングを実現するために，タスクは起床要求キューイング数を持つ．タスクの起床要求キューイング数は，タスクの起動時に0にクリアする．

タスクに対する強制待ち要求は，ネストされる．すなわち，すでに強制待ち状態（二重待ち状態を含む）になっているタスクを再度強制待ち状態に移行させようとすると，そのタスクを強制待ち状態に移行させようとしたという記録が残り，後でそのタスクを強制待ち状態（二重待ち状態を含む）から再開させようとした時に，強制待ちからの再開を行わない．タスクに対する強制待ち要求のネストを実現するために，タスクは強制待ち要求ネスト数を持つ．タスクの強制待ち要求ネスト数は，タスクの起動時に0にクリアする．

タスク付属同期機能に関連して，次のカーネル構成定数を定義する．

TMAX_WUPCNT	タスクの起床要求キューイング数の最大値
TMAX_SUSCNT	タスクの強制待ち要求ネスト数の最大値

タスク付属同期機能の各サービスコールの機能コードは次の通りである．

TFN_SLP_TSK	–0x11	slp_tskの機能コード
TFN_TSLP_TSK	–0x12	tslp_tskの機能コード
TFN_WUP_TSK	–0x13	wup_tskの機能コード
TFN_IWUP_TSK	–0x72	iwup_tskの機能コード
TFN_CAN_WUP	–0x14	can_wupの機能コード
TFN_REL_WAI	–0x15	rel_waiの機能コード
TFN_IREL_WAI	–0x73	irel_waiの機能コード
TFN_SUS_TSK	–0x16	sus_tskの機能コード
TFN_RSM_TSK	–0x17	rsm_tskの機能コード
TFN_FRSM_TSK	–0x18	frsm_tskの機能コード
TFN_DLY_TSK	–0x19	dly_tskの機能コード

【スタンダードプロファイル】

スタンダードプロファイルでは，タスク付属同期機能をサポートしなければならない．

スタンダードプロファイルでは，少なくとも1回のタスクの起床要求キューイ

ングをサポートしなければならない．また，タスクの強制待ち状態をサポートしなければならない．したがって，TMAX_WUPCNTとTMAX_SUSCNTは，ともに1以上でなければならない．

【補足説明】

TMAX_WUPCNTは, タスクの起床待ちをサポートしない場合には定義しない．タスクの起床要求キューイングをサポートしない場合には0とする．また，タスクの強制待ち状態をサポートしない場合には，TMAX_SUSCNTは定義しない．したがって，TMAX_SUSCNTが0に定義されることはない．

【μITRON3.0仕様との相違】

タスクの待ち状態を強制解除する機能（rel_wai），自タスクの実行を遅延する機能（dly_tsk）をタスク付属同期機能に分類することにした．

slp_tsk	起床待ち	【S】【B】
tslp_tsk	起床待ち（タイムアウトあり）	【S】

【C言語API】

ER ercd = slp_tsk ( ) ;
ER ercd = tslp_tsk (TMO tmout) ;

【パラメータ】

TMO	tmout	タイムアウト指定（tslp_tskのみ）

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	パラメータエラー（tmoutが不正；tslp_tskのみ）
E_RLWAI	待ち状態の強制解除（待ち状態の間にrel_waiを受付）
E_TMOUT	ポーリング失敗またはタイムアウト（tslp_tskのみ）

【機能】

自タスクを起床待ち状態に移行させる．ただし，自タスクに対する起床要求がキューイングされている場合，具体的には，自タスクの起床要求キューイング数が1以上の場合には，起床要求キューイング数から1を減じ，自タスクを待ち状態に移行させず，そのまま実行を継続する．

tslp_tskは，slp_tskにタイムアウトの機能を付け加えたサービスコールである．tmoutには, 正の値のタイムアウト時間に加えて, TMO_POL（＝0）とTMO_FEVR（＝–1）を指定することができる．

【補足説明】

このサービスコールは，自タスクに対する起床要求がキューイングされている場合に，自タスクをいったん待ち状態とはしない．そのため，自タスクの優先順位は変化しない．

slp_tskの処理をポーリングで行う専用のサービスコールは用意されてない．必要があれば，can_wupにより類似の機能を実現することができる．

wup_tsk　タスクの起床　【S】【B】
iwup_tsk　【S】【B】

【C言語API】

```
ER ercd = wup_tsk ( ID tskid ) ;
ER ercd = iwup_tsk ( ID tskid ) ;
```

【パラメータ】

ID	tskid	起床対象のタスクのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（tskidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象タスクが未登録）
E_OBJ	オブジェクト状態エラー（対象タスクが休止状態）
E_QOVR	キューイングオーバフロー（起床要求キューイング数のオーバフロー）

【機能】

tskidで指定されるタスクを，起床待ち状態から待ち解除する．待ち解除されたタスクに対しては，待ち状態に入ったサービスコールの返値としてE_OKを返す．

対象タスクが起床待ち状態でない場合には，タスクに対する起床要求をキューイングする（ただし，対象タスクが未登録状態の時はE_NOEXSエラー，休止状態の時はE_OBJエラーとなる）．具体的には，タスクの起床要求キューイング数に1を加える．タスクの起床要求キューイング数に1を加えると起床要求キューイング数の最大値を越える場合には，E_QOVRエラーを返す．

非タスクコンテキストから呼び出された場合で，サービスコールを遅延実行する場合には，E_OBJエラーとE_QOVRエラーを返すことを，実装定義で省略することができる．

tskidにTSK_SELF（＝0）が指定されると，自タスクを対象タスクとする．ただし，非タスクコンテキストからの呼出しでこの指定が行われた場合には，E_IDエラーを返す．

【補足説明】

スタンダードプロファイルでは，タスクの起床要求キューイング数の最大値は，1以上であればいくつであってもよい．したがって，スタンダードプロファイルに準拠したカーネルで，起床要求がキューイングされているタスクを指定してこのサービスコールを呼び出しても，E_QOVRエラーが返るとは限らない．

【μITRON3.0仕様との相違】

対象タスクに自タスクを指定できることとした．これは，act_tskとの整合性を考慮したためである．

can_wup　　タスク起床要求のキャンセル　　【S】【B】

【C言語API】

ER_UINT wupcnt = can_wup (ID tskid) ;

【パラメータ】

ID	tskid	起床要求のキャンセル対象のタスクのID番号

【リターンパラメータ】

ER_UINT	wupcnt	キューイングされていた起床要求の回数（正の値または0）またはエラーコード

【エラーコード】

E_ID	不正ID番号（tskidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象タスクが未登録）
E_OBJ	オブジェクト状態エラー（対象タスクが休止状態）

【機能】

tskidで指定されるタスクに対してキューイングされている起床要求をキャンセルし，キューイングされていた起床要求の回数を返す．具体的には，タスクの起床要求キューイング数を0にクリアし，クリアする前の起床要求キューイング数を返す．

tskidにTSK_SELF（＝0）が指定されると，自タスクを対象タスクとする．

【補足説明】

このサービスコールは，タスクを周期的に起床して処理を行う場合に，周期内に処理が終わっているかどうかを判定するために用いることができる．具体的には，前の起床要求に対する処理が終了した時点でcan_wupを呼び出し，その返値が1以上の値であった場合，前の起床要求に対する処理が周期内に終了せず，次の起床要求が行われたことがわかる．したがって，処理の遅れに対する処置をとることができる．

【μITRON3.0仕様との相違】

キューイングされていた起床要求の回数（wupcnt）を，サービスコールの返値として返すこととした．

rel_wai 待ち状態の強制解除 【S】【B】
irel_wai 【S】【B】

【C言語API】

ER ercd = rel_wai (ID tskid) ;
ER ercd = irel_wai (ID tskid) ;

【パラメータ】

ID	tskid	待ち状態の強制解除対象のタスクのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（tskidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象タスクが未登録）
E_OBJ	オブジェクト状態エラー（対象タスクが待ち状態でない）

【機能】

tskidで指定されるタスクが待ち状態にある場合に，強制的に待ち解除を行う．すなわち，対象タスクが待ち状態の時は実行可能状態に，二重待ち状態の時は強制待ち状態に移行させる．このサービスコールにより待ち解除されたタスクに対しては，待ち状態に入ったサービスコールの返値としてE_RLWAIエラーを返す．

対象タスクが待ち状態（二重待ち状態を含む）でない場合には，E_OBJエラーを返す（ただし，対象タスクが未登録状態の時はE_NOEXSエラーとなる）．非タスクコンテキストから呼び出された場合で，サービスコールを遅延実行する場合には，E_OBJエラーを返すことを，実装定義で省略することができる．

【補足説明】

対象タスクに自タスクを指定することはできない．自タスクが指定された場合には，待ち状態でないためにE_OBJエラーを返す．

このサービスコールでは，強制待ちからの再開は行わない．強制待ちからの再開を行う必要がある場合には，frsm_tsk（またはrsm_tsk）を用いる．

対象タスクが何らかの待ち行列につながれていた場合には，対象タスクを待ち行列から外す．その際に，その待ち行列で待っている他のタスクの待ち解除が必要になる場合がある（snd_mbfの機能説明とget_mplの機能説明を参照）．

rel_waiとwup_tskには次のような違いがある．

- wup_tskは起床待ち状態からのみ待ち解除するが，rel_waiは任意の要因による待ち状態から待ち解除する．
- 起床待ち状態になっていたタスクから見ると，wup_tskによる待ち解除は正

常終了（E_OK）であるのに対して，rel_waiによる強制的な待ち解除はエラー（E_RLWAI）である．

• wup_tskの場合は，対象タスクが起床待ち状態でない場合には，要求がキューイングされる．それに対してrel_waiの場合は，対象タスクが待ち状態でない場合にはE_OBJエラーとなる．

sus_tsk 強制待ち状態への移行 【S】【B】

【C言語API】

ER ercd = sus_tsk (ID tskid) ;

【パラメータ】

ID	tskid	移行対象のタスクのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（tskidが不正あるいは使用できない）
E_CTX	コンテキストエラー（ディスパッチ禁止状態で対象タスクに自タスクを指定，その他のコンテキストエラー）
E_NOEXS	オブジェクト未生成（対象タスクが未登録）
E_OBJ	オブジェクト状態エラー（対象タスクが休止状態）
E_QOVR	キューイングオーバフロー（強制待ち要求ネスト数のオーバフロー）

【機能】

tskid で指定されるタスクを強制待ち状態にして，タスクの実行を中断させる．具体的には，対象タスクが実行できる状態の時は強制待ち状態に，待ち状態の時は二重待ち状態に移行させる．また，対象タスクの強制待ち要求ネスト数に1を加える．タスクの強制待ち要求ネスト数に1を加えると強制待ち要求ネスト数の最大値を越える場合には，E_QOVRエラーを返す．

このサービスコールは，ディスパッチ禁止状態でも呼び出すことができるが，ディスパッチ禁止状態で自タスクを対象タスクとして呼び出された場合には，E_CTXエラーを返す．

tskidにTSK_SELF（＝0）が指定されると，自タスクを対象タスクとする．

【補足説明】

このサービスコールは，パラメータによっては自タスクを強制待ち状態にする可能性があるにもかかわらず，ディスパッチ禁止状態でも呼び出すことができる．したがって，「ディスパッチ禁止状態から自タスクを広義の待ち状態にする可能性のあるサービスコールが呼び出された場合に実装定義でE_CTXエラーとする仕様は，サービスコール単位に適用される」という原則の例外となっている．

スタンダードプロファイルでは，タスクの強制待ち要求ネスト数の最大値は，1以上であればいくつであってもよい．したがって，スタンダードプロファイルに準拠したカーネルで，強制待ち状態のタスクを指定してこのサービスコールを呼び出しても，E_QOVRエラーが返るとは限らない．

【μITRON3.0仕様との相違】

対象タスクに自タスクを指定できることとした.

rsm_tsk 強制待ち状態からの再開 【S】【B】
frsm_tsk 強制待ち状態からの強制再開 【S】

【C言語API】

ER ercd = rsm_tsk (ID tskid) ;
ER ercd = frsm_tsk (ID tskid) ;

【パラメータ】

ID	tskid	再開対象のタスクのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（tskidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象タスクが未登録）
E_OBJ	オブジェクト状態エラー（対象タスクが強制待ち状態でない）

【機能】

tskidで指定されるタスクの強制待ちを解除し，タスクの実行を再開させる．具体的な処理内容は次の通りである．

rsm_tskは，対象タスクの強制待ち要求ネスト数から1を減じ，減じた後の強制待ち要求ネスト数が0の場合には，対象タスクが強制待ち状態の時は実行可能状態に，二重待ち状態の時は待ち状態に移行させる．減じた後の強制待ち要求ネスト数が1以上の場合には，対象タスクの状態を変化させない．

frsm_tskは，対象タスクの強制待ち要求ネスト数を0とし，対象タスクが強制待ち状態の時は実行可能状態に，二重待ち状態の時は待ち状態に移行させる．

対象タスクが強制待ち状態（二重待ち状態を含む）でない場合には，E_OBJエラーを返す（ただし，対象タスクが未登録状態の時はE_NOEXSエラーとなる）．

【補足説明】

対象タスクに自タスクを指定することはできない．

【μITRON3.0仕様との相違】

タスクを強制待ち状態から実行可能状態に移行させる時に，同じ優先度内で最低の優先順位とするものとした．詳しくは，3.2.1節を参照すること．

dly_tsk 自タスクの遅延 【S】【B】

【C言語API】

ER ercd = dly_tsk (RELTIM dlytim) ;

【パラメータ】

RELTIM	dlytim	自タスクの遅延時間（相対時間）

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	パラメータエラー（dlytimが不正）
E_RLWAI	待ち状態の強制解除（待ち状態の間にrel_waiを受付）

【機能】

自タスクを時間経過待ち状態に移行させ，dlytimで指定される時間の間，実行を一時的に停止する．具体的には，サービスコールが呼び出された時刻からdlytimで指定された相対時間後に待ち解除されるよう設定し，自タスクを時間経過待ち状態に移行させる．指定された相対時間後に待ち解除された場合，このサービスコールは正常終了し，E_OKを返す．

dlytimは，サービスコールが呼び出された時刻を基準に，時間経過待ち状態から待ち解除される時刻を指定する相対時間と解釈する．

【補足説明】

時間経過待ち状態からの解除は，システム時刻に依存して行われる処理である．そのため，時間経過待ち状態からの解除は，待ち解除すべき時刻以降の最初のタイムティックで行う．また，時間経過待ち状態からの解除は，このサービスコールが呼び出されてから，dlytimで指定された以上の時間が経過した後に行うことを保証しなければならない（2.1.9節参照）．このサービスコールは，dlytimに0が指定された場合にも，自タスクを待ち状態に移行させる．

時間経過待ち状態は待ち状態の一種であり, rel_waiにより強制的に待ち解除することができる．タスクの遅延時間には，タスクが二重待ち状態になっている時間も含まれる．

このサービスコールは，tslp_tskと異なり，dlytimで指定された時間が経過して待ち解除された場合に正常終了となる．時間経過待ち状態にあるタスクを指定して wup_tsk が呼び出されても，時間経過待ち状態からの解除は行わない．dlytim で指定された時間が経過する前に dly_tsk が終了するのは，ter_tsk かrel_waiが呼び出された場合に限られる．

【μITRON3.0仕様との相違】

dlytimのデータ型をDLYTIMEからRELTIMに変更した．

4.3 タスク例外処理機能

タスク例外処理機能は，タスクに発生した例外事象の処理を，タスクのコンテキストで行うための機能である．タスク例外処理ルーチンを定義する機能，タスク例外処理を要求する機能，タスク例外処理を禁止／許可する機能，タスク例外処理に関する状態を参照する機能が含まれる．

タスク例外処理を要求するサービスコールが呼び出され，タスクに対してタスク例外処理が要求されると，タスクが実行中の処理を中断し，タスク例外処理ルーチンを起動する．タスク例外処理ルーチンは，タスクと同じコンテキストで実行する．タスク例外処理ルーチンからリターンすると，中断した処理の実行を継続する．アプリケーションは，タスク毎に一つのタスク例外処理ルーチンを登録することができる．タスクの生成直後は，タスク例外処理ルーチンは登録されていない．

タスクに対してタスク例外処理を要求する場合には，要求する例外処理の種類をあらわすタスク例外要因を指定する．カーネルは，タスク毎に，要求されたがまだ処理されていないタスク例外要因を管理する．これを，保留例外要因と呼ぶ．保留例外要因は，処理されていない例外処理要求がない時には0となっている．処理されていない例外処理要求があるタスクに対して，再度タスク例外処理が要求された場合には，タスクの保留例外要因を，新たに要求された例外処理のタスク例外要因とのビット毎の論理和に更新する．保留例外要因は，タスクの起動時に0にクリアする．

タスクは，タスク例外処理禁止状態かタスク例外処理許可状態かのいずれかの状態をとる．タスク例外処理禁止状態に移行することを「タスク例外処理を禁止する」，タスク例外処理許可状態に移行することを「タスク例外処理を許可する」ともいう．タスクの実行開始直後は，タスク例外処理禁止状態とする．

実装定義で，拡張サービスコールルーチンの起動によりタスク例外処理を禁止し，そこからのリターンにより起動前の状態に戻すことができる．このような実装では，拡張サービスコールルーチン内でタスク例外処理を許可してはならないため，拡張サービスコールルーチンからena_texが呼び出された場合にはE_CTXエラーを返す．

タスク例外処理許可状態であり，保留例外要因が0でなく，タスクが実行状態であり，非タスクコンテキストまたはCPU例外ハンドラが実行されていないという4つの条件が揃うと，タスク例外処理ルーチンを起動する．起動するタスク例外処理ルーチンには，起動時の保留例外要因（texptn）とタスクの拡張情報（exinf）をパラメータとして渡す．またこの時，タスクをタスク例外処理禁止状態に移行させ，保留例外要因を0にクリアする．

タスク例外処理ルーチンからリターンすると，タスク例外処理ルーチンを起動する前に実行していた処理の実行を継続する．この時，タスクをタスク例外処理許可状態に移行させる．ここで，保留例外要因が0でない場合には，再びタスク例外処理ルーチンを起動する．

タスク例外処理機能では，次のデータ型を用いる．

TEXPTN　　タスク例外要因のビットパターン（符号無し整数）

タスク例外処理ルーチンのC言語による記述形式は次の通りとする．

```
void texrtn ( TEXPTN texptn, VP_INT exinf )
{
        タスク例外処理ルーチン本体
}
```

タスク例外処理機能に関連して，次のカーネル構成定数を定義する．

TBIT_TEXPTN　　タスク例外要因のビット数（TEXPTNの有効ビット数）

タスク例外処理ルーチン定義情報およびタスク例外処理状態のパケット形式として，次のデータ型を定義する．

```
typedef struct t_dtex {
        ATR        texatr ;     /* タスク例外処理ルーチン属性 */
        FP         texrtn ;     /* タスク例外処理ルーチンの起動番
                                    地 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_DTEX ;

typedef struct t_rtex {
        STAT       texstat ;    /* タスク例外処理の状態 */
        TEXPTN     pndptn ;     /* 保留例外要因 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RTEX ;
```

タスク例外処理機能の各サービスコールの機能コードは次の通りである．

TFN_DEF_TEX	–0x1b	def_texの機能コード
TFN_RAS_TEX	–0x1c	ras_texの機能コード
TFN_IRAS_TEX	–0x74	iras_texの機能コード
TFN_DIS_TEX	–0x1d	dis_texの機能コード
TFN_ENA_TEX	–0x1e	ena_texの機能コード
TFN_SNS_TEX	–0x1f	sns_texの機能コード
TFN_REF_TEX	–0x20	ref_texの機能コード

【スタンダードプロファイル】

スタンダードプロファイルでは，タスク例外処理ルーチンを動的に定義する機能（def_tex），タスク例外処理の状態を参照する機能（ref_tex）を除いて，タスク例外処理機能をサポートしなければならない．

スタンダードプロファイルでは，タスク例外処理機能で用いるデータ型の有効ビット数を次の通りに定める．

TEXPTN　　16ビット以上

したがって，TBIT_TEXPTNは16以上でなければならない．

【補足説明】

この仕様の範囲内では，CPUロック状態でタスク例外処理ルーチンを起動すべき条件が揃うことはない．それに対して，ディスパッチ禁止状態でタスク例外処理ルーチンを起動すべき条件が揃うことはあり，その場合には，タスク例外処理ルーチンを起動しなければならない．

タスク例外処理ルーチンを実行するコンテキストと状態については，次のように整理できる．

- タスク例外処理ルーチンは，タスクと同じコンテキストで実行する（3.5.1節参照）．タスク例外処理ルーチンを実行するコンテキストは，タスクコンテキストに分類される．
- タスク例外処理ルーチンの起動と，そこからのリターンによって，CPUロック／ロック解除状態とディスパッチ禁止／許可状態は変化しない（3.5.4節と3.5.5節を参照）．ただし，CPUロック状態でタスク例外処理ルーチンを起動するかどうかは規定されない．

タスク例外処理禁止／許可状態が変化する状況は，次のように整理できる．

- タスク起動時には，タスク例外処理禁止状態とする．
- タスク例外処理ルーチンの起動時に，タスク例外処理禁止状態とし，タスク例外処理ルーチンからのリターン時に，タスク例外処理許可状態に戻す．
- dis_tex が呼び出されると，タスク例外処理禁止状態とし，ena_tex が呼び出されると，タスク例外処理許可状態とする．
- def_texによりタスク例外処理ルーチンの定義が解除されると，タスク例外処理禁止状態とする．

タスク例外処理ルーチンはタスクと同じコンテキストで実行するため，C言語の標準ライブラリ関数のlongjmpを用いて，タスク例外処理ルーチンから大域脱出することができる．この場合カーネルは，タスク例外処理ルーチンが終了したことを検知できないため，タスク例外処理許可状態には戻さない．タスク例外処理許可状態に戻したい場合には，アプリケーションでena_texを呼び出して，タスク例外処理を許可する必要がある．また，タスク例外処理ルーチンからの大域脱出を用いる場合，グローバルなデータ構造の一貫性が失われる間は，タスク例外処理を禁止する必要がある（仕様決定の理由を参照）．

タスク例外処理ルーチンからリターンした直後に再びタスク例外処理ルーチンを起動する場合，タスク例外処理ルーチンの実行開始直後のスタックポインタの値が，最初に起動した時の値と同じでなければならない．つまり，タスク例外処理ルーチンの再起動により，スタック上に無駄な領域が残ってはならない．さもないと，タスク例外処理ルーチンの連続起動によって使用されるスタック領域の上限サイズを押さえることができなくなる．

μITRON4.0仕様では，タスク例外要因毎に例外をマスクする機能は用意していないが，仕様に規定された機能を用いて，アプリケーションで同等の機能を実現することができる．具体的には，タスク毎のタスク例外処理マスクをアプリ

ケーションで管理し，タスク例外処理ルーチンの先頭で，渡された例外要因がマスクされているかどうかを調べる．マスクされていた場合には，その例外要因でタスク例外処理ルーチンが起動された旨を記録して，すぐにリターンする（実際には，マスクされた例外要因とマスクされていない例外要因が重なった場合に対応する必要がある）．後でタスク例外処理マスクを解除する時点で，マスクを解除する例外要因でタスク例外処理ルーチンが起動された記録があるか調べ，記録がある場合にはその時点でタスク例外処理ルーチンを呼び出す．

タスク例外処理ルーチンを起動する時にタスク例外処理を禁止するため，そのままでは，タスク例外処理ルーチンは多重起動されない．タスク例外処理ルーチンが複雑になる場合（特に，内部で待ち状態になる場合）には，タスク例外処理ルーチンの実行中に発生した例外により，タスク例外処理ルーチンを多重に起動したいケースが考えられる．このような場合には，タスク例外処理ルーチン内でena_texを呼び出してタスク例外処理を許可することで，タスク例外処理ルーチンの多重起動を実現することができる．この時，タスク例外処理ルーチンが無際限に多重起動することを防ぐための仕組みが必要である．例えば，上述のタスク例外要因毎にマスクする方法を併用し，処理中の例外要因をマスクする方法などが考えられる．

また，タスク例外処理ルーチンを起動する時にタスク例外処理を禁止するため，タスク例外処理ルーチン中でCPU例外が発生した場合，CPU例外ハンドラからタスク例外処理を要求しても，CPU例外ハンドラからリターンすると元の処理が継続してしまう．この時，CPU例外ハンドラ内でCPU例外を発生させた要因が取り除かれないと，リターン直後に再度CPU例外が発生し，CPU例外の発生を無際限に繰り返すおそれがある．同様のことは，タスク例外処理禁止状態で発生したCPU例外すべてにあてはまる．

そこで，基本的には，CPU例外ハンドラからタスク例外処理を要求する場合には，タスク例外処理を禁止している間はCPU例外が発生しないようにすることが必要である．ただし，ソフトウェアのバグやハードウェアの誤動作でCPU例外の発生が避けられない場合もあり，このような場合にもCPU例外が無際限に発生するのを避けるには，CPU例外ハンドラ中でCPU例外を発生させたタスクのタスク例外処理禁止状態を参照し，禁止状態であった場合には特別な処理を行う必要がある．また，必要なら上記の方法でタスク例外処理ルーチンの多重起動を可能にするなどして，タスク例外処理を禁止する区間を短く押さえることも必要であろう．

アプリケーションとカーネルでスタックを切り替える実装では，タスク例外処理ルーチンの多重起動を可能にするために，カーネルスタックまたはタスク制御ブロック（TCB；Task Control Block）に保存した情報をアプリケーションスタックに移動させる必要が生ずる場合が多い．例えば，タスクがプリエンプトされている状態でタスク例外処理が要求され，次にそのタスクが実行状態となった時点でタスク例外処理ルーチンを起動する場合には，プリエンプト直前の状態（一般的には，カーネルスタックかTCBに保存されている）をアプリ

ケーションスタックに積み直し，タスク例外処理ルーチンからのリターン時に，アプリケーションスタックに積まれた情報を元にプリエンプト直前の状態に戻す必要がある．

【μITRON3.0仕様との相違】

タスク例外処理機能は，μITRON4.0仕様において新たに導入した機能である．

【仕様決定の理由】

μITRON4.0仕様では，カーネルでは単純なタスク例外処理機能のみを提供し，より高度な機能が必要な場合にはアプリケーションでそれを実現できるように工夫した．これにより，カーネルをコンパクトに保ったまま，アプリケーションの高度な要求にも対応できるようにすることを狙っている．

仕様の本体では，タスク例外処理ルーチンはタスクと同じコンテキストで実行されるという記述にとどめ，longjmpによる大域脱出への言及を補足説明としたのは，以下で述べる理由により，タスク例外処理ルーチンから安易にlongjmpを使うのは危険なためである．また，スタンダードプロファイルの適用範囲を考えると，タスク例外処理ルーチンから強制脱出する方法は，タスクの終了（ext_tsk）のみで十分であると考えたためである．

タスク例外処理ルーチンから安易にlongjmpを使うと，例えば，グローバルなデータ構造を操作する関数の実行途中にタスク例外処理ルーチンが起動され，起動されたタスク例外処理ルーチン中からlongjmpで大域脱出すると，操作中のデータ構造の一貫性が保たれなくなる可能性がある．このようなケースを考えると，タスク例外処理ルーチン中からlongjmpで大域脱出する方法は，かなりの注意を払わない限り危険であるということができる．具体的には，データ構造の一貫性が失われる間は，タスク例外処理を禁止しなければならない．

DEF_TEX タスク例外処理ルーチンの定義（静的API） 【S】
def_tex タスク例外処理ルーチンの定義

【静的API】

DEF_TEX (ID tskid, { ATR texatr, FP texrtn }) ;

【C言語API】

ER ercd = def_tex (ID tskid, T_DTEX *pk_dtex) ;

【パラメータ】

ID	tskid	定義対象のタスクのID番号
T_DTEX *	pk_dtex	タスク例外処理ルーチン定義情報を入れたパケットへのポインタ（DEF_TEXではパケットの内容を直接記述する）

pk_dtexの内容（T_DTEX型）

ATR	texatr	タスク例外処理ルーチン属性
FP	texrtn	タスク例外処理ルーチンの起動番地

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（tskidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象タスクが未登録）
E_RSATR	予約属性（texatrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_dtex，texrtnが不正）

【機能】

tskidで指定されるタスクに，pk_dtexで指定されるタスク例外処理ルーチン定義情報に基づいて，タスク例外処理ルーチンを定義する．texatrはタスク例外処理ルーチンの属性，texrtnはタスク例外処理ルーチンの起動番地である．

DEF_TEXにおいては，tskidは自動割付け非対応整数値パラメータ，texatrはプリプロセッサ定数式パラメータである．

pk_dtexにNULL（＝0）が指定されると，すでに定義されているタスク例外処理ルーチンの定義を解除し，タスク例外処理ルーチンが定義されていない状態にする．この時，タスクの保留例外要因を0にクリアし，タスクをタスク例外処理禁止状態に移行させる．また，すでにタスク例外処理ルーチンが定義されているタスクに対して，再度タスク例外処理ルーチンが定義された場合には，以前の定義を解除し，新しい定義に置き換える．この時には，保留例外要因のクリアとタスク例外処理の禁止は行わない．

def_texは，tskidにTSK_SELF（＝0）が指定されると，自タスクを対象タスクと

する．

texatrには，（TA_HLNG ‖ TA_ASM）の指定ができる．TA_HLNG（＝0x00）が指定された場合には高級言語用のインタフェースで，TA_ASM（＝0x01）が指定された場合にはアセンブリ言語用のインタフェースでタスク例外処理ルーチンを起動する．

【スタンダードプロファイル】

スタンダードプロファイルでは，texatrにTA_ASMが指定された場合の機能はサポートする必要がない．

【補足説明】

一度定義されたタスク例外処理ルーチンは，pk_dtexにNULLが指定されてdef_texが呼び出されるか，タスクが削除されるまで有効である．

DEF_TEXによりタスク例外処理ルーチンを定義する対象タスクは，システムコンフィギュレーションファイル中で，それより前に記述されたCRE_TSKで生成されたものでなければならない．

【仕様決定の理由】

タスク例外処理ルーチンの定義を解除する時に，保留例外要因のクリアとタスク例外処理の禁止を行うのは，タスク例外処理ルーチンが定義されていない時には，保留例外要因が0でタスク例外処理が禁止された状態を保つためである（タスク例外処理ルーチンが定義されていない時には，保留例外要因がセットされることはなく，タスク例外処理を許可することもできないため，この状態が保たれる）．

ras_tex　タスク例外処理の要求　【S】
iras_tex　【S】

【C言語API】

ER ercd = ras_tex (ID tskid, TEXPTN rasptn) ;
ER ercd = iras_tex (ID tskid, TEXPTN rasptn) ;

【パラメータ】

ID	tskid	要求対象のタスクのID番号
TEXPTN	rasptn	要求するタスク例外処理のタスク例外要因

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（tskidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象タスクが未登録）
E_PAR	パラメータエラー（rasptnが不正）
E_OBJ	オブジェクト状態エラー（対象タスクが休止状態，対象タスクにタスク例外処理ルーチンが定義されていない）

【機能】

tskidで指定されるタスクに対して，rasptnで指定されるタスク例外要因によって，タスク例外処理を要求する．すなわち，対象タスクの保留例外要因を，サービスコール呼出し前の保留例外要因とrasptnの値のビット毎の論理和に更新する．

tskidにTSK_SELF（＝0）が指定されると，自タスクを対象タスクとする．ただし，非タスクコンテキストからの呼出しでこの指定が行われた場合には，E_IDエラーを返す．

対象タスクが休止状態の時はE_OBJエラーを返す．また，対象タスクにタスク例外処理ルーチンが定義されていない場合にも，E_OBJエラーを返す．非タスクコンテキストから呼び出された場合で，サービスコールを遅延実行する場合には，E_OBJエラーを返すことを，実装定義で省略することができる．

rasptnに0が指定された場合には，E_PARエラーを返す．

【補足説明】

このサービスコールにより，タスク例外処理ルーチンを起動する条件が揃った場合には，タスク例外処理ルーチンを起動する処理を行う．

このサービスコールでは，対象タスクが広義の待ち状態である場合に，保留例外要因の更新のみを行い，待ち解除や強制待ちからの再開は行わない．待ち解除や強制待ちからの再開を行う必要がある場合には，それぞれrel_waiやfrsm_tsk（またはrsm_tsk）を用いる．

非タスクコンテキストから呼び出されたサービスコールを遅延実行する場合に，ディスパッチが起こる状態となるまで実行を遅延できるサービスコールが多いが，このサービスコールはディスパッチ禁止状態でも実行しなければならない．具体的な例として，ディスパッチ禁止状態で起動された割込みハンドラで，実行状態のタスクを対象タスクとしてタスク例外処理が要求された場合には，割込みハンドラからのリターン直後にタスク例外処理ルーチンを起動しなければならない．これを使うと，ディスパッチ禁止状態で暴走するタスクに対して，割込みハンドラからタスク例外処理を要求することによって，暴走を止めることができる．ただし，CPUロック状態で暴走するタスクや，ディスパッチに加えてタスク例外処理も禁止した状態で暴走するタスクは，この方法で止めることはできない．

dis_tex　　タスク例外処理の禁止　　【S】

【C言語API】

```
ER ercd = dis_tex ( ) ;
```

【パラメータ】

なし

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_OBJ	オブジェクト状態エラー（自タスクにタスク例外処理ルーチンが定義されていない）

【機能】

自タスクを，タスク例外処理禁止状態に移行させる．自タスクにタスク例外処理ルーチンが定義されていない場合には，E_OBJエラーを返す．

ena_tex タスク例外処理の許可 【S】

【C言語API】

ER ercd = ena_tex ( ) ;

【パラメータ】

なし

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_OBJ	オブジェクト状態エラー（自タスクにタスク例外処理ルーチンが定義されていない）
E_CTX	コンテキストエラー（タスク例外処理を許可してはならないコンテキストからの呼出し，その他のコンテキストエラー）

【機能】

自タスクを，タスク例外処理許可状態に移行させる．自タスクにタスク例外処理ルーチンが定義されていない場合には，E_OBJエラーを返す．

拡張サービスコールルーチン内でタスク例外処理を許可してはならない実装で，このサービスコールが拡張サービスコールルーチンから呼び出された場合には，E_CTXエラーを返す．

【補足説明】

このサービスコールにより，タスク例外処理ルーチンを起動する条件が揃った場合には，タスク例外処理ルーチンを起動する処理を行う．

sns_tex　タスク例外処理禁止状態の参照　【S】

【C言語API】

BOOL state = sns_tex ( ) ;

【パラメータ】

なし

【リターンパラメータ】

BOOL	state	タスク例外処理禁止状態

【機能】

実行状態のタスク（タスクコンテキストから呼び出された場合は，自タスクに一致する）が，タスク例外処理禁止状態の場合にTRUE，タスク例外処理許可状態の場合にFALSEを返す．非タスクコンテキストから呼び出された場合で，実行状態のタスクがない時には，TRUEを返す．

【補足説明】

タスク例外処理ルーチンが定義されていないタスクは，タスク例外処理禁止状態に保たれているため，実行状態のタスクにタスク例外処理ルーチンが定義されていない場合には，このサービスコールはTRUEを返す．

ref_tex　　タスク例外処理の状態参照

【C言語API】

ER ercd = ref_tex (ID tskid, T_RTEX *pk_rtex) ;

【パラメータ】

ID	tskid	状態参照対象のタスクのID番号
T_RTEX *	pk_rtex	タスク例外処理状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rtexの内容（T_RTEX型）

STAT	texstat	タスク例外処理の状態
TEXPTN	pndptn	保留例外要因

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（tskidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象タスクが未登録）
E_PAR	パラメータエラー（pk_rtexが不正）
E_OBJ	オブジェクト状態エラー（対象タスクが休止状態，対象タスクにタスク例外処理ルーチンが定義されていない）

【機能】

tskidで指定されるタスクの，タスク例外処理に関する状態を参照し，pk_rtexで指定されるパケットに返す．

texstatには，対象タスクがタスク例外処理許可状態かタスク例外処理禁止状態かによって，次のいずれかの値を返す．

TTEX_ENA	0x00	タスク例外処理許可状態
TTEX_DIS	0x01	タスク例外処理禁止状態

pndptnには，対象タスクの保留例外要因を返す．処理されていない例外処理要求がないときには，pndptnには0を返す．

tskidにTSK_SELF（＝0）が指定されると，自タスクを対象タスクとする．

対象タスクが休止状態の時は，E_OBJエラーを返す．また，対象タスクにタスク例外処理ルーチンが定義されていない場合にも，E_OBJエラーを返す．

4.4 同期・通信機能

同期・通信機能は，タスクとは独立したオブジェクトにより，タスク間の同期・通信を行うための機能である．セマフォ，イベントフラグ，データキュー，メールボックスの各機能が含まれる．

【μITRON3.0仕様との相違】

メールボックスの実現方法をリンクリストを使う方法に限定し，リングバッファで実現したメールボックスと同等のデータキューの機能を新たに導入した．

4.4.1 セマフォ

セマフォは，使用されていない資源の有無や数量を数値で表現することにより，その資源を使用する際の排他制御や同期を行うためのオブジェクトである．セマフォ機能には，セマフォを生成／削除する機能，セマフォの資源を獲得／返却する機能，セマフォの状態を参照する機能が含まれる．セマフォはID番号で識別されるオブジェクトである．セマフォのID番号をセマフォ IDと呼ぶ．

セマフォは，対応する資源の有無や数量を表現する資源数と，資源の獲得を待つタスクの待ち行列を持つ．資源を返却する側（イベントを知らせる側）では，セマフォの資源数を1つ増やす．一方，資源を獲得する側（イベントを待つ側）では，セマフォの資源数を1つ減らす．セマフォの資源数が足りなくなった場合（具体的には，資源数を減らすと資源数が負になる場合），資源を獲得しようとしたタスクは，次に資源が返却されるまでセマフォ資源の獲得待ち状態となる．セマフォ資源の獲得待ち状態になったタスクは，そのセマフォの待ち行列につながれる．

また，セマフォに対して資源が返却され過ぎるのを防ぐために，セマフォ毎に最大資源数を設定することができる．最大資源数を越える資源がセマフォに返却されようとした場合（具体的には，セマフォの資源数を増やすと最大資源数を越える場合）には，エラーを報告する．

セマフォ機能に関連して，次のカーネル構成定数を定義する．

TMAX_MAXSEM　　セマフォの最大資源数の最大値

セマフォ生成情報およびセマフォ状態のパケット形式として，次のデータ型を定義する．

```
typedef struct t_csem {
        ATR         sematr ;      /* セマフォ属性 */
        UINT        isemcnt ;     /* セマフォの資源数の初期値 */
        UINT        maxsem ;      /* セマフォの最大資源数 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_CSEM ;

typedef struct t_rsem {
```

```
        ID          wtskid ;    /* セマフォの待ち行列の先頭のタスク
                                    のID番号 */
        UINT        semcnt ;    /* セマフォの現在の資源数 */
        /* 実装独自に他のフィールドを追加してもよい */
    } T_RSEM ;
```

セマフォ機能の各サービスコールの機能コードは次の通りである．

TFN_CRE_SEM	–0x21	cre_semの機能コード
TFN_ACRE_SEM	–0xc2	acre_semの機能コード
TFN_DEL_SEM	–0x22	del_semの機能コード
TFN_SIG_SEM	–0x23	sig_semの機能コード
TFN_ISIG_SEM	–0x75	isig_semの機能コード
TFN_WAI_SEM	–0x25	wai_semの機能コード
TFN_POL_SEM	–0x26	pol_semの機能コード
TFN_TWAI_SEM	–0x27	twai_semの機能コード
TFN_REF_SEM	–0x28	ref_semの機能コード

【スタンダードプロファイル】

スタンダードプロファイルでは，セマフォを動的に生成／削除する機能（cre_sem，acre_sem，del_sem），セマフォの状態を参照する機能（ref_sem）を除いて，セマフォ機能をサポートしなければならない．

スタンダードプロファイルでは，セマフォの最大資源数として，少なくとも65535以上の値が指定できなければならない．スタンダードプロファイルではTMAX_MAXSEMを定義する必要はないが，定義する場合には65535以上の値となる．

【仕様決定の理由】

スタンダードプロファイルでTMAX_MAXSEMを定義する必要がないとしたのは，TMAX_MAXSEMを参照する必要があるのはセマフォを生成する時であるのに対して，スタンダードプロファイルではセマフォを動的に生成する機能をサポートする必要がないためである．

CRE_SEM	セマフォの生成（静的API）	【S】
cre_sem	セマフォの生成	
acre_sem	セマフォの生成（ID番号自動割付け）	

【静的API】

CRE_SEM (ID semid, { ATR sematr, UINT isemcnt, UINT maxsem }) ;

【C言語API】

ER ercd = cre_sem (ID semid, T_CSEM *pk_csem) ;

ER_ID semid = acre_sem (T_CSEM *pk_csem) ;

【パラメータ】

ID	semid	生成対象のセマフォのID番号（acre_sem以外）
T_CSEM *	pk_csem	セマフォ生成情報を入れたパケットへのポインタ（CRE_SEMではパケットの内容を直接記述する）

pk_csemの内容（T_CSEM型）

ATR	sematr	セマフォ属性
UINT	isemcnt	セマフォの資源数の初期値
UINT	maxsem	セマフォの最大資源数

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cre_semの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acre_semの場合

ER_ID	semid	生成したセマフォのID番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（semidが不正あるいは使用できない；cre_semのみ）
E_NOID	ID番号不足（割付け可能なセマフォIDがない；acre_semのみ）
E_RSATR	予約属性（sematrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_csem，isemcnt，maxsemが不正）
E_OBJ	オブジェクト状態エラー（対象セマフォが登録済み；cre_semのみ）

【機能】

semidで指定されるID番号を持つセマフォを，pk_csemで指定されるセマフォ生成情報に基づいて生成する．sematrはセマフォの属性，isemcntはセマフォ生成

後の資源数の初期値，maxsemはセマフォの最大資源数である．

CRE_SEMにおいては，semidは自動割付け対応整数値パラメータ，sematrはプリプロセッサ定数式パラメータである．

acre_semは, 生成するセマフォのID番号をセマフォが登録されていないID番号の中から割り付け，割り付けたID番号を返値として返す.

sematrには,（TA_TFIFO ‖ TA_TPRI）の指定ができる．セマフォの待ち行列は, TA_TFIFO（＝0x00）が指定された場合にはFIFO順，TA_TPRI（＝0x01）が指定された場合にはタスクの優先度順となる．

isemcntにmaxsemよりも大きい値が指定された場合には, E_PARエラーを返す．また，maxsemに0が指定された場合や，セマフォの最大資源数の最大値（TMAX_MAXSEM）よりも大きい値が指定された場合にも，E_PARエラーを返す.

【μITRON3.0仕様との相違】

セマフォ生成情報から拡張情報を削除した．また，isemcntとmaxsemのデータ型をINTからUINTに変更した.

acre_semは新設のサービスコールである．

del_sem　　セマフォの削除

【C言語API】

ER ercd = del_sem (ID semid) ;

【パラメータ】

ID	semid	削除対象のセマフォのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（semidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象セマフォが未登録）

【機能】

semidで指定されるセマフォを削除する.

【補足説明】

対象セマフォに対して資源の獲得を待っているタスクがある場合の扱いについては，3.8節を参照すること.

sig_sem　セマフォ資源の返却　【S】【B】
isig_sem　【S】【B】

【C言語API】

```
ER ercd = sig_sem ( ID semid ) ;
ER ercd = isig_sem ( ID semid ) ;
```

【パラメータ】

ID	semid	資源返却対象のセマフォのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（semidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象セマフォが未登録）
E_QOVR	キューイングオーバフロー（最大資源数を越える返却）

【機能】

semidで指定されるセマフォに対して，資源を1つ返却する．具体的には，対象セマフォに対して資源の獲得を待っているタスクがある場合には，待ち行列の先頭のタスクを待ち解除する．この時，対象セマフォの資源数は変化しない．また，待ち解除されたタスクに対しては，待ち状態に入ったサービスコールの返値としてE_OKを返す．資源の獲得を待っているタスクがない場合には，対象セマフォの資源数に1を加える．

セマフォの資源数に1を加えるとセマフォの最大資源数を越える場合には，E_QOVRエラーを返す．非タスクコンテキストから呼び出された場合で，サービスコールを遅延実行する場合には，E_QOVRエラーを返すことを，実装定義で省略することができる．

wai_sem	セマフォ資源の獲得	【S】【B】
pol_sem	セマフォ資源の獲得（ポーリング）	【S】【B】
twai_sem	セマフォ資源の獲得（タイムアウトあり）	【S】

【C言語API】

ER ercd = wai_sem (ID semid) ;
ER ercd = pol_sem (ID semid) ;
ER ercd = twai_sem (ID semid, TMO tmout) ;

【パラメータ】

ID	semid	資源獲得対象のセマフォのID番号
TMO	tmout	タイムアウト指定（twai_semのみ）

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（semidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象セマフォが未登録）
E_PAR	パラメータエラー（tmoutが不正；twai_semのみ）
E_RLWAI	待ち状態の強制解除（待ち状態の間に rel_wai を受付；pol_sem以外）
E_TMOUT	ポーリング失敗またはタイムアウト（wai_sem以外）
E_DLT	待ちオブジェクトの削除（待ち状態の間に対象セマフォが削除；pol_sem以外）

【機能】

semidで指定されるセマフォから，資源を1つ獲得する．具体的には，対象セマフォの資源数が1以上の場合には，セマフォの資源数から1を減じ，自タスクを待ち状態とせずにサービスコールの処理を終了する．対象セマフォの資源数が0の場合には，自タスクを待ち行列につなぎ，セマフォ資源の獲得待ち状態に移行させる．この時，対象セマフォの資源数は0のまま変化しない．

他のタスクがすでに待ち行列につながっている場合，自タスクを待ち行列につなぐ処理は次のように行う．セマフォ属性にTA_TFIFO（＝0x00）が指定されている場合には，自タスクを待ち行列の末尾につなぐ．TA_TPRI（＝0x01）が指定されている場合には，自タスクを優先度順で待ち行列につなぐ．同じ優先度のタスクの中では，自タスクを最後につなぐ．

pol_semは，wai_semの処理をポーリングで行うサービスコール，twai_semは，wai_semにタイムアウトの機能を付け加えたサービスコールである．tmoutには，正の値のタイムアウト時間に加えて，TMO_POL（＝0）とTMO_FEVR（＝–1）を指定することができる．

【補足説明】

twai_semは，tmoutにTMO_POLが指定された場合，E_CTXエラーにならない限りはpol_semと全く同じ動作をする．また，tmoutにTMO_FEVRが指定された場合は，wai_semと全く同じ動作をする．

【μITRON3.0仕様との相違】

サービスコールの名称を，preq_semからpol_semに変更した．

ref_sem　　セマフォの状態参照

【C言語API】

ER ercd = ref_sem (ID semid, T_RSEM *pk_rsem) ;

【パラメータ】

ID	semid	状態参照対象のセマフォのID番号
T_RSEM *	pk_rsem	セマフォ状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rsemの内容（T_RSEM型）

ID	wtskid	セマフォの待ち行列の先頭のタスクのID番号
UINT	semcnt	セマフォの現在の資源数

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（semidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象セマフォが未登録）
E_PAR	パラメータエラー（pk_rsemが不正）

【機能】

semid で指定されるセマフォに関する状態を参照し，pk_rsem で指定されるパケットに返す．

wtskidには，対象セマフォの待ち行列の先頭のタスクのID番号を返す．資源の獲得を待っているタスクがない場合には，TSK_NONE（＝0）を返す．

semcntには，対象セマフォの現在の資源数を返す．

【補足説明】

wtskid ≠ TSK_NONE と semcnt ≠ 0 が同時に成立することはない．

【μITRON3.0仕様との相違】

参照できる情報から拡張情報を削除した．待ちタスクの有無ではなく，待ち行列の先頭のタスクの ID 番号を返すこととした．これに伴って，リターンパラメータの名称とデータ型を変更した．

semcntのデータ型をINTからUINTに変更した．また，パラメータとリターンパラメータの順序を変更した．

4.4.2 イベントフラグ

イベントフラグは，イベントの有無をビット毎のフラグで表現することにより，同期を行うためのオブジェクトである．イベントフラグ機能には，イベントフラグを生成／削除する機能，イベントフラグをセット／クリアする機能，イベントフラグで待つ機能，イベントフラグの状態を参照する機能が含まれる．イベントフラグはID番号で識別されるオブジェクトである．イベントフラグのID番号をイベントフラグIDと呼ぶ．

イベントフラグは，対応するイベントの有無をビット毎に表現するビットパターンと，そのイベントフラグで待つタスクの待ち行列を持つ．イベントフラグのビットパターンを，単にイベントフラグと呼ぶ場合もある．イベントを知らせる側では，イベントフラグのビットパターンの指定したビットをセットないしはクリアすることが可能である．一方，イベントを待つ側では，イベントフラグのビットパターンの指定したビットのすべてまたはいずれかがセットされるまで，タスクをイベントフラグ待ち状態にすることができる．イベントフラグ待ち状態になったタスクは，そのイベントフラグの待ち行列につながれる．

イベントフラグ機能では，次のデータ型を用いる．

FLGPTN　　イベントフラグのビットパターン（符号無し整数）

イベントフラグ機能に関連して，次のカーネル構成定数を定義する．

TBIT_FLGPTN　　イベントフラグのビット数（FLGPTNの有効ビット数）

イベントフラグ生成情報およびイベントフラグ状態のパケット形式として，次のデータ型を定義する．

```
typedef struct t_cflg {
        ATR         flgatr ;      /* イベントフラグ属性 */
        FLGPTN      iflgptn ;     /* イベントフラグのビットパターンの
                                     初期値 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_CFLG ;

typedef struct t_rflg {
        ID          wtskid ;      /* イベントフラグの待ち行列の先頭の
                                     タスクのID番号 */
        FLGPTN      flgptn ;      /* イベントフラグの現在のビットパ
                                     ターン */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RFLG ;
```

イベントフラグ機能の各サービスコールの機能コードは次の通りである．

TFN_CRE_FLG	–0x29	cre_flgの機能コード
TFN_ACRE_FLG	–0xc3	acre_flgの機能コード
TFN_DEL_FLG	–0x2a	del_flgの機能コード
TFN_SET_FLG	–0x2b	set_flgの機能コード

TFN_ISET_FLG	–0x76	iset_flgの機能コード
TFN_CLR_FLG	–0x2c	clr_flgの機能コード
TFN_WAI_FLG	–0x2d	wai_flgの機能コード
TFN_POL_FLG	–0x2e	pol_flgの機能コード
TFN_TWAI_FLG	–0x2f	twai_flgの機能コード
TFN_REF_FLG	–0x30	ref_flgの機能コード

【スタンダードプロファイル】

スタンダードプロファイルでは，イベントフラグを動的に生成／削除する機能（cre_flg，acre_flg，del_flg），イベントフラグの状態を参照する機能（ref_flg）を除いて，イベントフラグ機能をサポートしなければならない．

スタンダードプロファイルでは，イベントフラグで複数のタスクが待ち状態になる機能（イベントフラグ属性のTA_WMUL指定）をサポートする必要はない．

スタンダードプロファイルでは，イベントフラグのビット数は16ビット以上でなければならない．したがって，TBIT_FLGPTNは16以上でなければならない．また，イベントフラグ機能で用いるデータ型の有効ビット数は次の通りとなる．

FLGPTN 16ビット以上

【補足説明】

スタンダードプロファイル以外では，イベントフラグのビット数に対する制限はない．したがって，1ビットのイベントフラグ機能を提供することも許される．C言語では任意ビット数の整数型を扱うことができないため，このような場合には，FLGPTNの有効ビット数（＝TBIT_FLGPTN）は実装で定義されるデータ型のビット数と一致しなくなる．

【μITRON3.0仕様との相違】

イベントフラグのビットパターンを入れるパラメータおよびリターンパラメータのデータ型を，UINTから専用のデータ型として新設したFLGPTNに変更した．

CRE_FLG	イベントフラグの生成（静的API）	【S】
cre_flg	イベントフラグの生成	
acre_flg	イベントフラグの生成（ID番号自動割付け）	

【静的API】

CRE_FLG (ID flgid, { ATR flgatr, FLGPTN iflgptn }) ;

【C言語API】

ER ercd = cre_flg (ID flgid, T_CFLG *pk_cflg) ;
ER_ID flgid = acre_flg (T_CFLG *pk_cflg) ;

【パラメータ】

ID	flgid	生成対象のイベントフラグのID番号（acre_flg以外）
T_CFLG *	pk_cflg	イベントフラグ生成情報を入れたパケットへのポインタ（CRE_FLGではパケットの内容を直接記述する）

pk_cflgの内容（T_CFLG型）

ATR	flgatr	イベントフラグ属性
FLGPTN	iflgptn	イベントフラグのビットパターンの初期値

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cre_flgの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acre_flgの場合

ER_ID	flgid	生成したイベントフラグのID番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（flgidが不正あるいは使用できない；cre_flgのみ）
E_NOID	ID番号不足（割付け可能なイベントフラグIDがない；acre_flgのみ）
E_RSATR	予約属性（flgatrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_cflg，iflgptnが不正）
E_OBJ	オブジェクト状態エラー（対象イベントフラグが登録済み；cre_flgのみ）

【機能】

flgidで指定されるID番号を持つイベントフラグを，pk_cflgで指定されるイベントフラグ生成情報に基づいて生成する．flgatrはイベントフラグの属性，iflgptn

はイベントフラグ生成後のビットパターンの初期値である．

CRE_FLGにおいては，flgidは自動割付け対応整数値パラメータ，flgatrはプリプロセッサ定数式パラメータである．

acre_flgは，生成するイベントフラグのID番号をイベントフラグが登録されていないID番号の中から割り付け，割り付けたID番号を返値として返す．

flgatrには，((TA_TFIFO ‖ TA_TPRI)｜(TA_WSGL ‖ TA_WMUL)｜[TA_CLR])の指定ができる．イベントフラグの待ち行列は，TA_TFIFO（＝0x00）が指定された場合にはFIFO順，TA_TPRI（＝0x01）が指定された場合にはタスクの優先度順となる．TA_WSGL（＝0x00）が指定された場合には，一つのイベントフラグで同時に複数のタスクが待ち状態となることを許さない．TA_WMUL（＝0x02）が指定された場合には，同時に複数のタスクが待ち状態になることを許す．TA_CLR（＝0x04）が指定された場合には，タスクをイベントフラグ待ち状態から待ち解除する時に，イベントフラグのビットパターンのすべてのビットをクリアする．

【スタンダードプロファイル】

スタンダードプロファイルでは，flgatrにTA_WMULが指定された場合の機能はサポートする必要がない．

【補足説明】

イベントフラグ待ち状態のタスクは，待ち解除条件を満たせば待ち行列の先頭ではなくても待ち解除されるため，待ち行列につながれた順序で待ち解除されるとは限らない．例えば，TA_TFIFO属性のイベントフラグの場合でも，待ち状態のタスクがFIFO順で待ち解除されるわけではない．

flgatrにTA_WSGLが指定された場合には，TA_TFIFOとTA_TPRIのどちらを指定しても，イベントフラグの振舞いは同じである．

TA_CLR属性のイベントフラグの場合，タスクを一つ待ち解除した時点でイベントフラグのすべてのビットをクリアするため，複数のタスクを同時に待ち解除することはない．

【µITRON3.0仕様との相違】

イベントフラグのクリア指定を，wai_flgの待ちモードで指定する方法から，イベントフラグ属性で指定する方法に変更した．これは，一つのイベントフラグで，クリア指定付きの待ちとクリア指定無しの待ちが混在することはほとんどないと考えたためである．

イベントフラグ属性（TA_TPRI）により，イベントフラグの待ち行列をタスクの優先度順とする機能を追加した．

イベントフラグ生成情報から拡張情報を削除した．iflgptnのデータ型をUINTからFLGPTNに変更した．また，TA_WMULの値を変更した．

acre_flgは新設のサービスコールである．

del_flg イベントフラグの削除

【C言語API】

```
ER ercd = del_flg ( ID flgid ) ;
```

【パラメータ】

ID	flgid	削除対象のイベントフラグのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（flgidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象イベントフラグが未登録）

【機能】

flgidで指定されるイベントフラグを削除する．

【補足説明】

対象イベントフラグで待っているタスクがある場合の扱いについては，3.8節を参照すること．

set_flg	イベントフラグのセット	【S】【B】
iset_flg		【S】【B】

【C言語API】

ER ercd = set_flg (ID flgid, FLGPTN setptn) ;
ER ercd = iset_flg (ID flgid, FLGPTN setptn) ;

【パラメータ】

ID	flgid	セット対象のイベントフラグのID番号
FLGPTN	setptn	セットするビットパターン

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（flgidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象イベントフラグが未登録）
E_PAR	パラメータエラー（setptnが不正）

【機能】

flgid で指定されるイベントフラグに対して，setptn で指定されるビットをセットする．具体的には，対象イベントフラグのビットパターンを，サービスコール呼出し前のビットパターンとsetptnの値のビット毎の論理和に更新する．

対象イベントフラグのビットパターンが更新された結果，そのイベントフラグで待っているタスクの待ち解除条件を満たした場合には，該当するタスクを待ち解除する．具体的には，イベントフラグの待ち行列の先頭のタスクから順に待ち解除条件を満たしているかを調べ，待ち解除条件を満たしているタスクが見つかれば，そのタスクを待ち解除する．また，待ち解除されたタスクに対しては，待ち状態に入ったサービスコールの返値としてE_OKを返し，待ち解除時のビットパターンとして，この時の（待ち解除条件を満たした）ビットパターンを返す．またこの時，対象イベントフラグ属性にTA_CLR（＝0x04）が指定されている場合には，イベントフラグのビットパターンのすべてのビットをクリアし，サービスコールの処理を終了する．TA_CLRが指定されていない場合には，待ち行列のさらに後ろのタスクについても待ち解除の条件を満たしているかを調べ，待ち行列の最後のタスクまで調べた時点で終了する．待ち解除条件については，wai_flgの機能説明を参照すること．

イベントフラグ属性にTA_WMUL（＝0x02）が指定されており，TA_CLRが指定されていない場合，set_flgの一回の呼出しで複数のタスクが待ち解除される可能性がある．この場合，待ち解除されるタスクが複数ある場合には，イベントフラグの待ち行列につながれていた順序で待ち解除される．そのため，実行可能状態に移行したタスクで同じ優先度を持つものの間では，待ち行列の中で前につながれていたタスクの方が高い優先順位を持つことになる．

【補足説明】

setptnの全ビットが0の場合には，何もしない．

【μITRON3.0仕様との相違】

setptnのデータ型をUINTからFLGPTNに変更した．

clr_flg　　イベントフラグのクリア　　【S】【B】

【C言語API】

ER ercd = clr_flg (ID flgid, FLGPTN clrptn) ;

【パラメータ】

ID	flgid	セット対象のイベントフラグのID番号
FLGPTN	clrptn	クリアするビットパターン（ビット毎の反転値）

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（flgidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象イベントフラグが未登録）
E_PAR	パラメータエラー（clrptnが不正）

【機能】

flgidで指定されるイベントフラグに対して, clrptnの対応するビットが0になっているビットをクリアする．具体的には，対象イベントフラグのビットパターンを，サービスコール呼出し前のビットパターンとclrptnの値のビット毎の論理積に更新する.

【補足説明】

clrptnの全ビットが1の場合には，何もしない.

【μITRON3.0仕様との相違】

clrptnのデータ型をUINTからFLGPTNに変更した.

wai_flg	イベントフラグ待ち	【S】【B】
pol_flg	イベントフラグ待ち（ポーリング）	【S】【B】
twai_flg	イベントフラグ待ち（タイムアウトあり）	【S】

【C言語API】

```
ER ercd = wai_flg ( ID flgid, FLGPTN waiptn, MODE wfmode,
                    FLGPTN *p_flgptn ) ;
ER ercd = pol_flg ( ID flgid, FLGPTN waiptn, MODE wfmode,
                    FLGPTN *p_flgptn ) ;
ER ercd = twai_flg ( ID flgid, FLGPTN waiptn, MODE wfmode,
                    FLGPTN *p_flgptn, TMO tmout ) ;
```

【パラメータ】

ID	flgid	待ち対象のイベントフラグのID番号
FLGPTN	waiptn	待ちビットパターン
MODE	wfmode	待ちモード
TMO	tmout	タイムアウト指定（twai_flgのみ）

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード
FLGPTN	flgptn	待ち解除時のビットパターン

【エラーコード】

E_ID	不正ID番号（flgidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象イベントフラグが未登録）
E_PAR	パラメータエラー（waiptn，wfmode，p_flgptn，tmoutが不正）
E_ILUSE	サービスコール不正使用（TA_WSGL属性が指定されたイベントフラグで待ちタスクあり）
E_RLWAI	待ち状態の強制解除（待ち状態の間に rel_wai を受付；pol_flg以外）
E_TMOUT	ポーリング失敗またはタイムアウト（wai_flg以外）
E_DLT	待ちオブジェクトの削除（待ち状態の間に対象イベントフラグが削除；pol_flg以外）

【機能】

flgidで指定されるイベントフラグのビットパターンが，waiptnとwfmodeで指定される待ち解除条件を満たすのを待つ．flgptnには，待ち解除される時のイベントフラグのビットパターンを返す．具体的な処理内容は次の通りである．

対象イベントフラグのビットパターンがwaiptnとwfmodeで指定される待ち解除条件を満たしている場合には，自タスクを待ち状態とせずにサービスコールの処理を終了する．flgptnには，この時の（待ち解除条件を満たした）ビット

パターンを返す．またこの時，対象イベントフラグ属性にTA_CLR（＝0x04）が指定されている場合には，イベントフラグのビットパターンのすべてのビットをクリアする．

対象イベントフラグのビットパターンがwaiptnとwfmodeで指定される待ち解除条件を満たしていない場合には，自タスクを待ち行列につなぎ，イベントフラグ待ち状態に移行させる．

対象イベントフラグ属性にTA_WSGL（＝0x00）が指定されており，イベントフラグの待ち行列に他のタスクがつながれている場合には，待ち解除条件を満たしているかどうかにかかわらず，E_ILUSEエラーとなる．

wfmodeには，（TWF_ANDW ‖ TWF_ORW）の指定ができる．waiptnとwfmodeで指定される待ち解除条件とは，wfmodeにTWF_ANDW（＝0x00）が指定された場合には，対象イベントフラグのビットパターンのwaiptnで指定されるビットのすべてがセットされるという条件である．TWF_ORW（＝0x01）が指定された場合には，対象イベントフラグのビットパターンのwaiptnで指定されるビットのいずれかがセットされるという条件である．

他のタスクがすでに待ち行列につながっている場合，自タスクを待ち行列につなぐ処理は次のように行う．イベントフラグ属性にTA_TFIFO（＝0x00）が指定されている場合には，自タスクを待ち行列の末尾につなぐ．TA_TPRI（＝0x01）が指定されている場合には，自タスクを優先度順で待ち行列につなぐ．同じ優先度のタスクの中では，自タスクを最後につなぐ．

pol_flgは，wai_flgの処理をポーリングで行うサービスコール，twai_flgは，wai_flgにタイムアウトの機能を付け加えたサービスコールである．tmoutには，正の値のタイムアウト時間に加えて，TMO_POL（＝0）とTMO_FEVR（＝–1）を指定することができる．

waiptnに0が指定された場合には，E_PARエラーを返す．

【補足説明】

twai_flgは，tmoutにTMO_POLが指定された場合，E_CTXエラーにならない限りはpol_flgと全く同じ動作をする．また，tmoutにTMO_FEVRが指定された場合は，wai_flgと全く同じ動作をする．

【μITRON3.0仕様との相違】

パラメータとリターンパラメータの順序を変更した．また，waitpnとflgptnのデータ型をUINTからFLGPTNに，wfmodeのデータ型をUINTからMODEにそれぞれ変更した．

待ちモードのクリア指定（TWF_CLR）を廃止した．同等の機能として，TA_CLR属性のイベントフラグを用意している．また，TWF_ORWの値を変更した．

【仕様決定の理由】

waiptnに0が指定された場合をE_PARエラーとしたのは，この指定をした場合に，イベントフラグのビットパターンがどのような値になっても待ち解除状態を満たさないため，イベントフラグ待ち状態から抜けることができなくなるた

めである.

ref_flg イベントフラグの状態参照

【C言語API】

ER ercd = ref_flg (ID flgid, T_RFLG *pk_rflg) ;

【パラメータ】

ID	flgid	状態参照対象のイベントフラグのID番号
T_RFLG *	pk_rflg	イベントフラグ状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rflgの内容（T_RFLG型）

ID	wtskid	イベントフラグの待ち行列の先頭のタスクのID番号
FLGPTN	flgptn	イベントフラグの現在のビットパターン

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（flgidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象イベントフラグが未登録）
E_PAR	パラメータエラー（pk_rflgが不正）

【機能】

flgidで指定されるイベントフラグに関する状態を参照し，pk_rflgで指定されるパケットに返す．

wtskidには，対象イベントフラグの待ち行列の先頭のタスクのID番号を返す．イベントを待っているタスクがない場合には，TSK_NONE（＝0）を返す．

flgptnには，対象イベントフラグの現在のビットパターンを返す．

【μITRON3.0仕様との相違】

参照できる情報から拡張情報を削除した．待ちタスクの有無ではなく，待ち行列の先頭のタスクのID番号を返すこととした．これに伴って，リターンパラメータの名称とデータ型を変更した．

flgptnのデータ型をUINTからFLGPTNに変更した．また，パラメータとリターンパラメータの順序を変更した．

4.4.3 データキュー

データキューは，1ワードのメッセージ（これをデータと呼ぶ）を受渡しすることにより，同期と通信を行うためのオブジェクトである．データキュー機能には，データキューを生成／削除する機能，データキューに対してデータを送信／強制送信／受信する機能，データキューの状態を参照する機能が含まれる．データキューはID番号で識別されるオブジェクトである．データキューのID番号をデータキューIDと呼ぶ．

データキューは，データの送信を待つタスクの待ち行列（送信待ち行列）とデータの受信を待つタスクの待ち行列（受信待ち行列）を持つ．また，送信されたデータを格納するためのデータキュー領域を持つ．データを送信する側（イベントを知らせる側）では，送信したいデータをデータキューに入れる．データキュー領域に空きがない場合は，データキュー領域に空きができるまでデータキューへの送信待ち状態になる．データキューへの送信待ち状態になったタスクは，そのデータキューの送信待ち行列につながれる．一方，データを受信する側（イベントを待つ側）では，データキューに入っているデータを一つ取り出す．データキューにデータが入っていない場合は，次にデータが送られてくるまでデータキューからの受信待ち状態になる．データキューからの受信待ち状態になったタスクは，そのデータキューの受信待ち行列につながれる．

データキュー領域に格納できるデータの数を0にすることで，同期メッセージ機能を実現することができる．すなわち，送信側のタスクと受信側のタスクが，それぞれ相手のタスクがサービスコールを呼び出すのを待ち合わせ，両者がサービスコールを呼び出した時点で，データの受渡しが行われる．

データキューで送受信されるデータ（1ワードのメッセージ）は，整数値であっても，送信側と受信側で共有しているメモリ上に置かれたメッセージの先頭番地であってもよい．送受信されるデータは，送信側から受信側にコピーされる．

データキュー機能に関連して，次のカーネル構成マクロを定義する．

SIZE dtqsz = TSZ_DTQ (UINT dtqcnt)

dtqcnt 個のデータを格納するのに必要なデータキュー領域のサイズ（バイト数）

データキュー生成情報およびデータキュー状態のパケット形式として，次のデータ型を定義する．

```
typedef struct t_cdtq {
        ATR         dtqatr ;      /* データキュー属性 */
        UINT        dtqcnt ;      /* データキュー領域の容量（データの
                                       個数）*/
        VP          dtq ;         /* データキュー領域の先頭番地 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_CDTQ ;

typedef struct t_rdtq {
```

```
        ID          stskid ;      /* データキューの送信待ち行列の先頭
                                     のタスクのID番号 */
        ID          rtskid ;      /* データキューの受信待ち行列の先頭
                                     のタスクのID番号 */
        UINT        sdtqcnt ;     /* データキューに入っているデータの
                                     数 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RDTQ ;
```

データキュー機能の各サービスコールの機能コードは次の通りである．

TFN_CRE_DTQ	–0x31	cre_dtqの機能コード
TFN_ACRE_DTQ	–0xc4	acre_dtqの機能コード
TFN_DEL_DTQ	–0x32	del_dtqの機能コード
TFN_SND_DTQ	–0x35	snd_dtqの機能コード
TFN_PSND_DTQ	–0x36	psnd_dtqの機能コード
TFN_IPSND_DTQ	–0x77	ipsnd_dtqの機能コード
TFN_TSND_DTQ	–0x37	tsnd_dtqの機能コード
TFN_FSND_DTQ	–0x38	fsnd_dtqの機能コード
TFN_IFSND_DTQ	–0x78	ifsnd_dtqの機能コード
TFN_RCV_DTQ	–0x39	rcv_dtqの機能コード
TFN_PRCV_DTQ	–0x3a	prcv_dtqの機能コード
TFN_TRCV_DTQ	–0x3b	trcv_dtqの機能コード
TFN_REF_DTQ	–0x3c	ref_dtqの機能コード

【スタンダードプロファイル】

スタンダードプロファイルでは，データキューを動的に生成／削除する機能（cre_dtq，acre_dtq，del_dtq），データキューの状態を参照する機能（ref_dtq）を除いて，データキュー機能をサポートしなければならない．

スタンダードプロファイルでは，データキュー領域の容量（格納できるデータの個数）として，少なくとも255以上の値が指定できなければならない．

スタンダードプロファイルでは，TSZ_DTQを定義する必要はない．

【補足説明】

データキュー領域に格納できるデータの数を0にした場合のデータキューの動作を，図4-1の例を用いて説明する．この図で，タスクAとタスクBは非同期に実行しているものとする．

- もしタスクAが先にsnd_dtqを呼び出した場合には，タスクBがrcv_dtqを呼び出すまでタスクAは待ち状態となる．この時タスクAは，データキューへの送信待ち状態になっている（図4-1 (a)）．
- 逆にタスクBが先にrcv_dtqを呼び出した場合には，タスクAがsnd_dtqを呼び出すまでタスクBは待ち状態となる．この時タスクBは，データキューからの受信待ち状態になっている（図4-1 (b)）．
- タスクAがsnd_dtqを呼び出し，タスクBがrcv_dtqを呼び出した時点で，タ

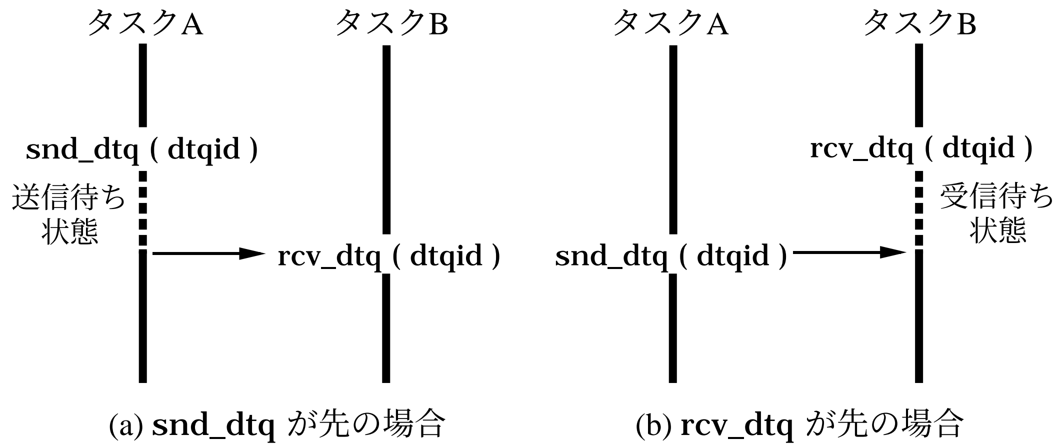


図4-1. データキューによる同期通信

スクAからタスクBへデータの受渡しが行われる．その後は，両タスクとも実行できる状態となる．

データキューは，リングバッファで実装することを想定している．

【μITRON3.0仕様との相違】

データキュー機能は，μITRON3.0仕様のリングバッファで実現したメールボックスと同等の機能であり，μITRON4.0 仕様において新たに導入した機能である．

CRE_DTQ	データキューの生成（静的API）	【S】
cre_dtq	データキューの生成	
acre_dtq	データキューの生成（ID番号自動割付け）	

【静的API】

CRE_DTQ (ID dtqid, { ATR dtqatr, UINT dtqcnt, VP dtq }) ;

【C言語API】

ER ercd = cre_dtq (ID dtqid, T_CDTQ *pk_cdtq) ;

ER_ID dtqid = acre_dtq (T_CDTQ *pk_cdtq) ;

【パラメータ】

ID	dtqid	生成対象のデータキューのID番号（acre_dtq以外）
T_CDTQ *	pk_cdtq	データキュー生成情報を入れたパケットへのポインタ（CRE_DTQではパケットの内容を直接記述する）

pk_cdtqの内容（T_CDTQ型）

ATR	dtqatr	データキュー属性
UINT	dtqcnt	データキュー領域の容量（データの個数）
VP	dtq	データキュー領域の先頭番地

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cre_dtqの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acre_dtqの場合

ER_ID	dtqid	生成したデータキューのID番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（dtqidが不正あるいは使用できない；cre_dtqのみ）
E_NOID	ID 番号不足（割付け可能なデータキューID がない；acre_dtqのみ）
E_NOMEM	メモリ不足（データキュー領域などが確保できない）
E_RSATR	予約属性（dtqatrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_cdtq，dtqcnt，dtqが不正）
E_OBJ	オブジェクト状態エラー（対象データキューが登録済み；cre_dtqのみ）

【機能】

dtqidで指定されるID番号を持つデータキューを，pk_cdtqで指定されるデータキュー生成情報に基づいて生成する．dtqatrはデータキューの属性，dtqcntはデータキュー領域に格納できるデータの個数，dtqはデータキュー領域の先頭番地である．

CRE_DTQにおいては，dtqidは自動割付け対応整数値パラメータ，dtqatrとdtqcntはプリプロセッサ定数式パラメータである．

acre_dtqは，生成するデータキューのID番号をデータキューが登録されていないID番号の中から割り付け，割り付けたID番号を返値として返す．

dtqatrには，（TA_TFIFO ‖ TA_TPRI）の指定ができる．データキューの送信待ち行列は，TA_TFIFO（＝0x00）が指定された場合にはFIFO順，TA_TPRI（＝0x01）が指定された場合にはタスクの優先度順となる．

dtqで指定された番地から，dtqcnt個のデータを格納するのに必要なサイズのメモリ領域を，データキュー領域として使用する．TSZ_DTQを用いると，アプリケーションプログラムから，dtqcnt個のデータを格納するのに必要なサイズを知ることができる．dtqにNULL（＝0）が指定された場合には，必要なサイズのメモリ領域をカーネルが確保する．

dtqcntに実装定義の最大値よりも大きい値が指定された場合には，E_PARエラーを返す．dtqcntに0を指定することは可能である．

【スタンダードプロファイル】

スタンダードプロファイルでは，dtqにNULL以外の値が指定された場合の機能はサポートする必要がない．

【補足説明】

データキューの受信待ち行列は，常にFIFO順となる．また，データキューに送信されるデータは優先度を持たず，データキュー中のデータの順序もFIFO順のみをサポートしている．言い換えると，先に送信されたデータから順に受信されるのが原則である．ただし，snd_dtqとfsnd_dtqを併用した場合には，snd_dtqで送信待ちしているタスクがあってもfsnd_dtqでデータを送信することはできるため，fsnd_dtqで送信されたデータがsnd_dtqで送信されたデータを追い越すことがある．

del_dtq　　データキューの削除

【C言語API】

ER ercd = del_dtq (ID dtqid) ;

【パラメータ】

ID	dtqid	削除対象のデータキューのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（dtqidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象データキューが未登録）

【機能】

dtqidで指定されるデータキューを削除する．データキュー領域をカーネルで確保した場合には，その領域を解放する．

【補足説明】

対象データキューに入っていたデータは破棄される．対象データキューで送信または受信を待っているタスクがある場合の扱いについては，3.8節を参照すること．

snd_dtq	データキューへの送信	【S】
psnd_dtq	データキューへの送信（ポーリング）	【S】
ipsnd_dtq		【S】
tsnd_dtq	データキューへの送信（タイムアウトあり）	【S】

【C言語API】

ER ercd = snd_dtq (ID dtqid, VP_INT data) ;
ER ercd = psnd_dtq (ID dtqid, VP_INT data) ;
ER ercd = ipsnd_dtq (ID dtqid, VP_INT data) ;
ER ercd = tsnd_dtq (ID dtqid, VP_INT data, TMO tmout) ;

【パラメータ】

ID	dtqid	送信対象のデータキューのID番号
VP_INT	data	データキューへ送信するデータ
TMO	tmout	タイムアウト指定（tsnd_dtqのみ）

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（dtqidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象データキューが未登録）
E_PAR	パラメータエラー（tmoutが不正；tsnd_dtqのみ）
E_RLWAI	待ち状態の強制解除（待ち状態の間に rel_wai を受付；snd_dtq，tsnd_dtqのみ）
E_TMOUT	ポーリング失敗またはタイムアウト（snd_dtq以外）
E_DLT	待ちオブジェクトの削除（待ち状態の間に対象データキューが削除；snd_dtq，tsnd_dtqのみ）

【機能】

dtqidで指定されるデータキューに，dataで指定されるデータを送信する．具体的な処理内容は次の通りである．

対象データキューで受信を待っているタスクがある場合には，受信待ち行列の先頭のタスクに送信するデータを渡し，そのタスクを待ち解除する．この時，待ち解除されたタスクに対しては，待ち状態に入ったサービスコールの返値としてE_OKを返し，データキューから受信したデータとしてdataの値を返す．

受信を待っているタスクがない場合は，送信するデータをデータキューの末尾に入れる．データキュー領域に空きがない場合には，自タスクを送信待ち行列につなぎ，データキューへの送信待ち状態に移行させる．

他のタスクがすでに送信待ち行列につながっている場合，自タスクを送信待ち行列につなぐ処理は次のように行う．データキュー属性にTA_TFIFO（＝0x00）が指定されている場合には，自タスクを送信待ち行列の末尾につなぐ．

TA_TPRI（＝0x01）が指定されている場合には，自タスクを優先度順で送信待ち行列につなぐ．同じ優先度のタスクの中では，自タスクを最後につなぐ．

psnd_dtq と ipsnd_dtq は，snd_dtq の処理をポーリングで行うサービスコール，tsnd_dtqは，snd_dtqにタイムアウトの機能を付け加えたサービスコールである．tmoutには，正の値のタイムアウト時間に加えて，TMO_POL（＝0）とTMO_FEVR（＝–1）を指定することができる．

psnd_dtq と ipsnd_dtq は，対象データキューで受信を待っているタスクがなく，データキュー領域に空きがない場合には，E_TMOUTエラーを返す．非タスクコンテキストから呼び出された場合で，サービスコールを遅延実行する場合には，E_TMOUTエラーを返すことを，実装定義で省略することができる．

【補足説明】

tsnd_dtqは，tmoutにTMO_POLが指定された場合，E_CTXエラーにならない限りはpsnd_dtqと全く同じ動作をする．また，tmoutにTMO_FEVRが指定された場合は，snd_dtqと全く同じ動作をする．

fsnd_dtq データキューへの強制送信 【S】
ifsnd_dtq 【S】

【C言語API】

ER ercd = fsnd_dtq (ID dtqid, VP_INT data) ;
ER ercd = ifsnd_dtq (ID dtqid, VP_INT data) ;

【パラメータ】

ID	dtqid	送信対象のデータキューのID番号
VP_INT	data	データキューへ送信するデータ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（dtqidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象データキューが未登録）
E_ILUSE	サービスコール不正使用（対象データキューのデータキュー領域の容量が0）

【機能】

dtqidで指定されるデータキューに，dataで指定されるデータを強制送信する．具体的な処理内容は次の通りである．

対象データキューで受信を待っているタスクがある場合には，受信待ち行列の先頭のタスクに送信するデータを渡し，そのタスクを待ち解除する．この時，待ち解除されたタスクに対しては，待ち状態に入ったサービスコールの返値としてE_OKを返し，データキューから受信したデータとしてdataの値を返す．

受信を待っているタスクがない場合は，送信するデータをデータキューの末尾に入れる．ここで，データキュー領域に空きがない場合には，データキューの先頭のデータを抹消し，データキュー領域に必要な領域を確保する．この場合でも，送信するデータはデータキューの末尾に入れる．

これらのサービスコールで，データキュー領域の容量が0のデータキューに対してデータを強制送信することはできない．対象データキューのデータキュー領域の容量が0の場合には，E_ILUSEエラーを返す．

【補足説明】

これらのサービスコールは，対象データキューで送信を待っているタスクがある場合でも，データを強制送信する．

対象データキューのデータキュー領域の容量が0の場合には，データキューで受信を待っているタスクがある場合でも，E_ILUSEエラーを返す．

rcv_dtq データキューからの受信 【S】
prcv_dtq データキューからの受信（ポーリング） 【S】
trcv_dtq データキューからの受信（タイムアウトあり） 【S】

【C言語API】

ER ercd = rcv_dtq (ID dtqid, VP_INT *p_data) ;
ER ercd = prcv_dtq (ID dtqid, VP_INT *p_data) ;
ER ercd = trcv_dtq (ID dtqid, VP_INT *p_data, TMO tmout) ;

【パラメータ】

ID	dtqid	受信対象のデータキューのID番号
TMO	tmout	タイムアウト指定（trcv_dtqのみ）

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード
VP_INT	data	データキューから受信したデータ

【エラーコード】

E_ID	不正ID番号（dtqidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象データキューが未登録）
E_PAR	パラメータエラー（p_data，tmoutが不正）
E_RLWAI	待ち状態の強制解除（待ち状態の間に rel_wai を受付；prcv_dtq以外）
E_TMOUT	ポーリング失敗またはタイムアウト（rcv_dtq以外）
E_DLT	待ちオブジェクトの削除（待ち状態の間に対象データキューが削除；prcv_dtq以外）

【機能】

dtqidで指定されるデータキューからデータを受信し，dataに返す．具体的な処理内容は次の通りである．

対象データキューにデータが入っている場合には，その先頭のデータを取り出し，dataに返す．データキューで送信を待っているタスクがある場合には，送信待ち行列の先頭のタスクが送信しようとしているデータをデータキューの末尾に入れ，そのタスクを待ち解除する．この時，待ち解除されたタスクに対しては，待ち状態に入ったサービスコールの返値としてE_OKを返す．

データが入っていない場合で，対象データキューで送信を待っているタスクがある場合には（このような状況が起こるのは，データキュー領域の容量が0の場合のみである），送信待ち行列の先頭のタスクから，そのタスクが送信しようとしているデータを受け取り，そのタスクを待ち解除する．この時，待ち解除されたタスクに対しては，待ち状態に入ったサービスコールの返値としてE_OKを返す．dataには，受け取ったデータを返す．

データが入っていない場合で，送信を待っているタスクもない場合には，自タ

スクを受信待ち行列につなぎ，データキューからの受信待ち状態に移行させる．他のタスクがすでに受信待ち行列につながっている場合には，自タスクを受信待ち行列の末尾につなぐ．

prcv_dtq は，rcv_dtq の処理をポーリングで行うサービスコール，trcv_dtq は，rcv_dtq にタイムアウトの機能を付け加えたサービスコールである．tmout には，正の値のタイムアウト時間に加えて，TMO_POL（＝0）と TMO_FEVR（＝–1）を指定することができる．

【補足説明】

trcv_dtq は，tmout に TMO_POL が指定された場合，E_CTX エラーにならない限りは prcv_dtq と全く同じ動作をする．また，tmout に TMO_FEVR が指定された場合は，rcv_dtq と全く同じ動作をする．

ref_dtq　　データキューの状態参照

【C言語API】

ER ercd = ref_dtq (ID dtqid, T_RDTQ *pk_rdtq) ;

【パラメータ】

ID	dtqid	状態参照対象のデータキューのID番号
T_RDTQ *	pk_rdtq	データキュー状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rdtqの内容（T_RDTQ型）

ID	stskid	データキューの送信待ち行列の先頭のタスクのID番号
ID	rtskid	データキューの受信待ち行列の先頭のタスクのID番号
UINT	sdtqcnt	データキューに入っているデータの数

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（dtqidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象データキューが未登録）
E_PAR	パラメータエラー（pk_rdtqが不正）

【機能】

dtqidで指定されるデータキューに関する状態を参照し, pk_rdtqで指定されるパケットに返す.

stskidには, 対象データキューの送信待ち行列の先頭のタスクのID番号を返す. 送信を待っているタスクがない場合には，TSK_NONE（＝0）を返す.

rtskidには, 対象データキューの受信待ち行列の先頭のタスクのID番号を返す. 受信を待っているタスクがない場合には，TSK_NONE（＝0）を返す.

sdtqcntには，対象データキューに現在入っているデータの個数を返す.

【補足説明】

rtskid ≠ TSK_NONEとsdtqcnt ≠ 0が同時に成立することはない．また，stskid ≠ TSK_NONEの時には，sdtqcntはデータキュー領域の容量に一致する.

4.4.4 メールボックス

メールボックスは，共有メモリ上に置かれたメッセージを受渡しすることにより，同期と通信を行うためのオブジェクトである．メールボックス機能には，メールボックスを生成／削除する機能，メールボックスに対してメッセージを送信／受信する機能，メールボックスの状態を参照する機能が含まれる．メールボックスはID番号で識別されるオブジェクトである．メールボックスのID番号をメールボックスIDと呼ぶ．

メールボックスは，送信されたメッセージを入れるためのメッセージキューと，メッセージの受信を待つタスクの待ち行列を持つ．メッセージを送信する側（イベントを知らせる側）では，送信したいメッセージをメッセージキューに入れる．一方，メッセージを受信する側（イベントを待つ側）では，メッセージキューに入っているメッセージを一つ取り出す．メッセージキューにメッセージが入っていない場合は，次にメッセージが送られてくるまでメールボックスからの受信待ち状態になる．メールボックスからの受信待ち状態になったタスクは，そのメールボックスの待ち行列につながれる．

メールボックスによって実際に送受信されるのは，送信側と受信側で共有しているメモリ上に置かれたメッセージの先頭番地のみである．すなわち，送受信されるメッセージの内容のコピーは行わない．

カーネルは，メッセージキューに入っているメッセージを，リンクリストにより管理する．アプリケーションプログラムは，送信するメッセージの先頭に，カーネルがリンクリストに用いるための領域を確保しなければならない．この領域をメッセージヘッダと呼ぶ．また，メッセージヘッダと，それに続くアプリケーションがメッセージを入れるための領域をあわせて，メッセージパケットと呼ぶ．メールボックスへメッセージを送信するサービスコールは，メッセージパケットの先頭番地をパラメータとする．また，メールボックスからメッセージを受信するサービスコールは，メッセージパケットの先頭番地をリターンパラメータとして返す．メッセージキューをメッセージの優先度順にする場合には，メッセージの優先度を入れるための領域も，メッセージヘッダ中に持つ必要がある．

カーネルは，メッセージキューに入っている（ないしは，入れようとしている）メッセージのメッセージヘッダ（メッセージ優先度のための領域を除く）の内容を書き換える．一方，アプリケーションは，メッセージキューに入っているメッセージのメッセージヘッダ（メッセージ優先度のための領域を含む）の内容を書き換えてはならない．アプリケーションによってメッセージヘッダの内容が書き換えられた場合の振舞いは未定義である．この規定は，アプリケーションプログラムがメッセージヘッダの内容を直接書き換えた場合に加えて，メッセージヘッダの番地をカーネルに渡し，カーネルにメッセージヘッダの内容を書き換えさせた場合にも適用される．したがって，すでにメッセージキューに入っているメッセージを再度メールボックスに送信した場合の振舞いは未定義となる．

メッセージヘッダのデータ型として，次のデータ型を定義する．

T_MSG　　　　メールボックスのメッセージヘッダ
T_MSG_PRI　　メールボックスの優先度付きメッセージヘッダ

T_MSG型の定義やサイズは，実装定義である．それに対してT_MSG_PRI型は，T_MSG型を用いて次のように定義する．

```
typedef struct t_msg_pri {
        T_MSG       msgque ;    /* メッセージヘッダ */
        PRI         msgpri ;    /* メッセージ優先度 */
} T_MSG_PRI ;
```

メールボックス機能に関連して，次のカーネル構成マクロを定義する．

SIZE mprihdsz = TSZ_MPRIHD (PRI maxmpri)

送信されるメッセージの優先度の最大値がmaxmpriのメールボックスに必要な優先度別メッセージキューヘッダ領域のサイズ（バイト数）

メールボックス生成情報およびメールボックス状態のパケット形式として，次のデータ型を定義する．

```
typedef struct t_cmbx {
        ATR         mbxatr ;    /* メールボックス属性 */
        PRI         maxmpri ;   /* 送信されるメッセージの優先度の最
                                   大値 */
        VP          mprihd ;    /* 優先度別のメッセージキューヘッダ
                                   領域の先頭番地 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_CMBX ;

typedef struct t_rmbx {
        ID          wtskid ;    /* メールボックスの待ち行列の先頭の
                                   タスクのID番号 */
        T_MSG *     pk_msg ;    /* メッセージキューの先頭のメッセー
                                   ジパケットの先頭番地 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RMBX ;
```

メールボックス機能の各サービスコールの機能コードは次の通りである．

TFN_CRE_MBX	–0x3d	cre_mbxの機能コード
TFN_ACRE_MBX	–0xc5	acre_mbxの機能コード
TFN_DEL_MBX	–0x3e	del_mbxの機能コード
TFN_SND_MBX	–0x3f	snd_mbxの機能コード
TFN_RCV_MBX	–0x41	rcv_mbxの機能コード
TFN_PRCV_MBX	–0x42	prcv_mbxの機能コード
TFN_TRCV_MBX	–0x43	trcv_mbxの機能コード
TFN_REF_MBX	–0x44	ref_mbxの機能コード

【スタンダードプロファイル】

スタンダードプロファイルでは，メールボックスを動的に生成／削除する機能

（cre_mbx，acre_mbx，del_mbx），メールボックスの状態を参照する機能（ref_mbx）を除いて，メールボックス機能をサポートしなければならない．

スタンダードプロファイルでは，TSZ_MPRIHDを定義する必要はない．

【補足説明】

メールボックス機能では，メッセージヘッダの領域をアプリケーションプログラムで確保することとしているため，メッセージキューに入れることができるメッセージの数には上限がない．また，メッセージを送信するサービスコールで待ち状態になることもない．

アプリケーションプログラムをメッセージヘッダの定義やサイズが異なるカーネルへも移植可能とするためには，アプリケーションで用いるメッセージパケットをC言語の構造体として定義し，その先頭にT_MSG型ないしはT_MSG_PRI型のフィールドを確保すればよい．また，メッセージに優先度を設定する場合には，メッセージパケットの先頭に確保したT_MSG_PRI型のフィールド中のmsgpriフィールドに代入する．メッセージヘッダのサイズを知る必要がある場合には，sizeof (T_MSG) ないしはsizeof (T_MSG_PRI) を用いることができる．

メッセージパケットとしては，固定長メモリプールまたは可変長メモリプールから動的に確保したメモリブロックを用いることも，静的に確保した領域を用いることも可能である．一般的な使い方としては，送信側のタスクがメモリプールからメモリブロックを確保し，それをメッセージパケットとして送信し，受信側のタスクはメッセージの内容を取り出した後にそのメモリブロックを直接メモリプールに返却するという手順をとることが多い．

【μITRON3.0仕様との相違】

メールボックスの実現方法を，リンクリストを使う方法に限定した．

CRE_MBX メールボックスの生成（静的API） 【S】
cre_mbx メールボックスの生成
acre_mbx メールボックスの生成（ID番号自動割付け）

【静的API】

CRE_MBX (ID mbxid, { ATR mbxatr, PRI maxmpri, VP mprihd }) ;

【C言語API】

ER ercd = cre_mbx (ID mbxid, T_CMBX *pk_cmbx) ;

ER_ID mbxid = acre_mbx (T_CMBX *pk_cmbx) ;

【パラメータ】

ID	mbxid	生成対象のメールボックスのID番号（acre_mbx以外）
T_CMBX *	pk_cmbx	メールボックス生成情報を入れたパケットへのポインタ（CRE_MBXではパケットの内容を直接記述する）

pk_cmbxの内容（T_CMBX型）

ATR	mbxatr	メールボックス属性
PRI	maxmpri	送信されるメッセージの優先度の最大値
VP	mprihd	優先度別のメッセージキューヘッダ領域の先頭番地

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cre_mbxの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acre_mbxの場合

ER_ID	mbxid	生成したメールボックスのID番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mbxidが不正あるいは使用できない；cre_mbxのみ）
E_NOID	ID番号不足（割付け可能なメールボックスIDがない；acre_mbxのみ）
E_NOMEM	メモリ不足（優先度別のメッセージキューヘッダ領域などが確保できない）
E_RSATR	予約属性（mbxatrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_cmbx，maxmpri，mprihdが不正）
E_OBJ	オブジェクト状態エラー（対象メールボックスが登録済み；cre_mbxのみ）

【機能】

mbxidで指定されるID番号を持つメールボックスを，pk_cmbxで指定されるメールボックス生成情報に基づいて生成する．mbxatrはメールボックスの属性，maxmpriはメールボックスに送信されるメッセージの優先度の最大値，mprihdはメールボックスの優先度別のメッセージキューヘッダ領域の先頭番地である．maxmpriとmprihdは，mbxatrにTA_MPRI（＝0x02）が指定された場合にのみ有効である．

CRE_MBXにおいては，mbxidは自動割付け対応整数値パラメータ，mbxatrとmaxmpriはプリプロセッサ定数式パラメータである．

acre_mbxは，生成するメールボックスのID番号をメールボックスが登録されていないID番号の中から割り付け，割り付けたID番号を返値として返す．

mbxatrには，((TA_TFIFO ‖ TA_TPRI) ｜ (TA_MFIFO ‖ TA_MPRI))の指定ができる．メールボックスの待ち行列は，TA_TFIFO（＝0x00）が指定された場合にはFIFO順，TA_TPRI（＝0x01）が指定された場合にはタスクの優先度順となる．また，メールボックスのメッセージキューは，TA_MFIFO（＝0x00）が指定された場合にはFIFO順，TA_MPRI（＝0x02）が指定された場合にはメッセージの優先度順となる．

mbxatrにTA_MPRIが指定された場合，mprihdで指定された番地から，送信されるメッセージの優先度の最大値がmaxmpriの場合に必要なサイズのメモリ領域を，優先度別のメッセージキューヘッダ領域として使用する．TSZ_MPRIHDを用いると，アプリケーションプログラムから，送信されるメッセージの優先度の最大値がmaxmpriの場合に必要なサイズを知ることができる．mprihdにNULL（＝0）が指定された場合には，必要なサイズのメモリ領域をカーネルが確保する．

maxmpriに0が指定された場合や，メッセージ優先度の最大値（TMAX_MPRI）よりも大きい値が指定された場合には，E_PARエラーを返す．

【スタンダードプロファイル】

スタンダードプロファイルでは，mprihdにNULL以外の値が指定された場合の機能はサポートする必要がない．

【補足説明】

メールボックスの優先度別のメッセージキューヘッダ領域を活用し，メッセージのキューを優先度毎に用意する場合には，次のような配慮が必要である．

メッセージのキューを優先度毎に用意する方法は，メッセージ優先度の段階数が少ない場合には有効な方法であるが，メッセージ優先度の段階数が多い場合には，メモリ使用量が大きくなるために現実的ではない．そこで，メッセージ優先度の段階数が多い場合にも対応するために，メッセージ優先度の段階数によってキューの構造を変えるといった工夫が必要になる．例えば，送信されるメッセージの優先度の最大値が一定の値以下である場合にはメッセージのキューを優先度毎に用意し，そうでない場合にはすべてのメッセージを一つの

キューで管理するといった方法が考えられる．この場合，TSZ_MPRIHD は，maxmpriが一定の値よりも大きい場合には一定値を返すように定義する．また，CRE_MBXにおいてmaxmpriをプリプロセッサ定数式パラメータとしているのは，カーネルのコンフィギュレータが生成するC言語のソースコード中にmaxmpriに関する条件ディレクティブを記述し，maxmpriが一定の値よりも大きい場合にカーネル内部のデータ構造を変化させることを可能にするためである．

一方，すべてのメッセージを一つのキューで管理し，優先度別のメッセージキューヘッダ領域を使わない実装も可能である．このような実装では，TSZ_MPRIHDはmaxmpriの値にかかわらず一定値を返すように定義する．

【μITRON3.0仕様との相違】

メールボックス生成情報に，送信されるメッセージの優先度の最大値（maxmpri）と優先度別のメッセージキューヘッダ領域の先頭番地（mprihd）を追加し，拡張情報とリングバッファの大きさ（実装依存の情報）を削除した．

acre_mbxは新設のサービスコールである．

del_mbx　　　メールボックスの削除

【C言語API】

```
ER ercd = del_mbx ( ID mbxid ) ;
```

【パラメータ】

ID	mbxid	削除対象のメールボックスのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mbxidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象メールボックスが未登録）

【機能】

mbxidで指定されるメールボックスを削除する．優先度別のメッセージキューヘッダ領域をカーネルで確保した場合には，その領域を解放する．

【補足説明】

対象メールボックスのメッセージキューに入っていたメッセージは破棄される．対象メールボックスで受信を待っているタスクがある場合の扱いについては，3.8節を参照すること．

snd_mbx メールボックスへの送信 【S】【B】

【C言語API】

ER ercd = snd_mbx (ID mbxid, T_MSG *pk_msg) ;

【パラメータ】

ID	mbxid	送信対象のメールボックスのID番号
T_MSG *	pk_msg	メールボックスへ送信するメッセージパケットの先頭番地

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mbxidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象メールボックスが未登録）
E_PAR	パラメータエラー（pk_msgが不正，メッセージパケット中のメッセージ優先度（msgpri）が不正）

【機能】

mbxidで指定されるメールボックスに，pk_msgを先頭番地とするメッセージを送信する．具体的な処理内容は次の通りである．

対象メールボックスで受信を待っているタスクがある場合には，待ち行列の先頭のタスクにpk_msgで指定されたメッセージパケットの先頭番地を渡し，そのタスクを待ち解除する．この時，待ち解除されたタスクに対しては，待ち状態に入ったサービスコールの返値としてE_OKを返し，メールボックスから受信したメッセージパケットの先頭番地としてpk_msgの値を返す.

受信を待っているタスクがない場合は，pk_msgを先頭番地とするメッセージパケットをメッセージキューに入れる．ここで，メールボックス属性にTA_MFIFO（＝0x00）が指定されている場合には，pk_msgを先頭番地とするメッセージパケットをメッセージキューの末尾に入れる．TA_MPRI（＝0x02）が指定されている場合には，メッセージパケットを優先度順でメッセージキューに入れる．同じ優先度のメッセージパケットの中では，新たに送信されたメッセージパケットを最後に入れる．

メールボックス属性にTA_MPRI（＝0x02）が指定されている場合，pk_msgを先頭番地とするメッセージパケットの先頭に，T_MSG_PRI型のメッセージヘッダがあるものと仮定し，メッセージの優先度をその中のmsgpriフィールドから取り出す.

【µITRON3.0仕様との相違】

サービスコールの名称を，snd_msgからsnd_mbxに変更した.

rcv_mbx	メールボックスからの受信	【S】【B】
prcv_mbx	メールボックスからの受信（ポーリング）	【S】【B】
trcv_mbx	メールボックスからの受信（タイムアウトあり）	【S】

【C言語API】

ER ercd = rcv_mbx (ID mbxid, T_MSG **ppk_msg) ;
ER ercd = prcv_mbx (ID mbxid, T_MSG **ppk_msg) ;
ER ercd = trcv_mbx (ID mbxid, T_MSG **ppk_msg, TMO tmout) ;

【パラメータ】

ID	mbxid	受信対象のメールボックスのID番号
TMO	tmout	タイムアウト指定（trcv_mbxのみ）

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード
T_MSG *	pk_msg	メールボックスから受信したメッセージパケットの先頭番地

【エラーコード】

E_ID	不正ID番号（mbxidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象メールボックスが未登録）
E_PAR	パラメータエラー（ppk_msg，tmoutが不正）
E_RLWAI	待ち状態の強制解除（待ち状態の間に rel_wai を受付；prcv_mbx以外）
E_TMOUT	ポーリング失敗またはタイムアウト（rcv_mbx以外）
E_DLT	待ちオブジェクトの削除（待ち状態の間に対象メールボックスが削除；prcv_mbx以外）

【機能】

mbxidで指定されるメールボックスからメッセージを受信し，その先頭番地をpk_msgに返す．具体的な処理内容は次の通りである．

対象メールボックスのメッセージキューにメッセージが入っている場合には，その先頭のメッセージパケットを取り出し，その先頭番地をpk_msgに返す．

メッセージが入っていない場合は，自タスクを待ち行列につなぎ，メールボックスからの受信待ち状態に移行させる．

他のタスクがすでに待ち行列につながっている場合，自タスクを待ち行列につなぐ処理は次のように行う．メールボックス属性にTA_TFIFO（＝0x00）が指定されている場合には，自タスクを待ち行列の末尾につなぐ．TA_TPRI（＝0x01）が指定されている場合には，自タスクを優先度順で待ち行列につなぐ．同じ優先度のタスクの中では，自タスクを最後につなぐ．

prcv_mbxは，rcv_mbxの処理をポーリングで行うサービスコール，trcv_mbxは，rcv_mbxにタイムアウトの機能を付け加えたサービスコールである．tmoutに

は，正の値のタイムアウト時間に加えて，TMO_POL（＝0）とTMO_FEVR（＝–1）を指定することができる．

【補足説明】

trcv_mbxは，tmoutにTMO_POLが指定された場合，E_CTXエラーにならない限りはprcv_mbxと全く同じ動作をする．また，tmoutにTMO_FEVRが指定された場合は，rcv_mbxと全く同じ動作をする．

【μITRON3.0仕様との相違】

サービスコールの名称を，rcv_msg, prcv_msg, trcv_msgから，それぞれrcv_mbx，prcv_mbx，trcv_mbxに変更した．また，パラメータとリターンパラメータの順序を変更した．

ref_mbx　　メールボックスの状態参照

【C言語API】

ER ercd = ref_mbx (ID mbxid, T_RMBX *pk_rmbx) ;

【パラメータ】

ID	mbxid	状態参照対象のメールボックスのID番号
T_RMBX *	pk_rmbx	メールボックス状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rmbxの内容（T_RMBX型）

ID	wtskid	メールボックスの待ち行列の先頭のタスクのID番号
T_MSG *	pk_msg	メッセージキューの先頭のメッセージパケットの先頭番地

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（mbxidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象メールボックスが未登録）
E_PAR	パラメータエラー（pk_rmbxが不正）

【機能】

mbxidで指定されるメールボックスに関する状態を参照し，pk_rmbxで指定されるパケットに返す．

wtskidには，対象メールボックスの待ち行列の先頭のタスクのID番号を返す．受信を待っているタスクがない場合には，TSK_NONE（＝0）を返す．

pk_msgには，対象メールボックスのメッセージキューの先頭のメッセージパケットの先頭番地を返す．メッセージキューにメッセージが入っていない場合には，NULL（＝0）を返す．

【補足説明】

wtskid ≠ TSK_NONEとpk_msg ≠ NULLが同時に成立することはない．

【μITRON3.0仕様との相違】

参照できる情報から拡張情報を削除した．待ちタスクの有無ではなく，待ち行列の先頭のタスクのID番号を返すこととした．これに伴って，リターンパラメータの名称とデータ型を変更した．

パラメータとリターンパラメータの順序を変更した．

4.5 拡張同期・通信機能

拡張同期・通信機能は，タスクとは独立したオブジェクトにより，タスク間の高度な同期・通信を行うための機能である．ミューテックス，メッセージバッファ，ランデブの各機能が含まれる．

【スタンダードプロファイル】

スタンダードプロファイルでは，拡張同期・通信機能はサポートする必要がない．

【μITRON3.0仕様との相違】

ミューテックスは新たに導入した機能である．

4.5.1 ミューテックス

ミューテックスは，共有資源を使用する際にタスク間で排他制御を行うためのオブジェクトである．ミューテックスは，排他制御に伴う上限のない優先度逆転を防ぐための機構として，優先度継承プロトコル（priority inheritance protocol）と優先度上限プロトコル（priority ceiling protocol）をサポートする．ミューテックス機能には，ミューテックスを生成／削除する機能，ミューテックスをロック／ロック解除する機能，ミューテックスの状態を参照する機能が含まれる．ミューテックスはID番号で識別されるオブジェクトである．ミューテックスのID番号をミューテックスIDと呼ぶ．

ミューテックスは，ロックされているかどうかの状態と，ロックを待つタスクの待ち行列を持つ．また，カーネルは，各ミューテックスに対してそれをロックしているタスクを，各タスクに対してそれがロックしているミューテックスの集合を管理する．タスクは，資源を使用する前に，ミューテックスをロックする．ミューテックスが他のタスクにロックされていた場合には，ミューテックスがロック解除されるまで，ミューテックスのロック待ち状態となる．ミューテックスのロック待ち状態になったタスクは，そのミューテックスの待ち行列につながれる．タスクは，資源の使用を終えると，ミューテックスのロックを解除する．

ミューテックスは，ミューテックス属性にTA_INHERIT（＝0x02）を指定することにより優先度継承プロトコルを，TA_CEILING（＝0x03）を指定することにより優先度上限プロトコルをサポートする．TA_CEILING属性のミューテックスに対しては，そのミューテックスをロックする可能性のあるタスクの中で最も高いベース優先度を持つタスクのベース優先度を，ミューテックス生成時に上限優先度として設定する．TA_CEILING属性のミューテックスを，その上限優先度よりも高いベース優先度を持つタスクがロックしようとした場合，E_ILUSEエラーとなる．また，TA_CEILING属性のミューテックスをロックしているかロックを待っているタスクのベース優先度を，chg_priによってそのミューテックスの上限優先度よりも高く設定しようとした場合，chg_priが

E_ILUSEエラーを返す．

これらのプロトコルを用いた場合，上限のない優先度逆転を防ぐために，ミューテックスの操作に伴ってタスクの現在優先度を変更する．優先度継承プロトコルと優先度上限プロトコルに厳密に従うなら，タスクの現在優先度を，次に挙げる優先度の最高値に常に一致するように変更する必要がある．これを，厳密な優先度制御規則と呼ぶ．

- タスクのベース優先度
- タスクが TA_INHERIT 属性のミューテックスをロックしている場合，それらのミューテックスのロックを待っているタスクの中で，最も高い現在優先度を持つタスクの現在優先度
- タスクが TA_CEILING 属性のミューテックスをロックしている場合，それらのミューテックス中で，最も高い上限優先度を持つミューテックスの上限優先度

ここで，TA_INHERIT属性のミューテックスを待っているタスクの現在優先度が，ミューテックス操作かchg_priによるベース優先度の変更に伴って変更された場合，そのミューテックスをロックしているタスクの現在優先度の変更が必要になる場合がある．これを推移的な優先度継承と呼ぶ．さらにそのタスクが，別のTA_INHERIT属性のミューテックスを待っていた場合には，そのミューテックスをロックしているタスクに対して推移的な優先度継承の処理が必要になる場合がある．

μITRON4.0仕様では，上述の厳密な優先度制御規則に加えて，現在優先度を変更する状況を限定した優先度制御規則（これを簡略化した優先度制御規則と呼ぶ）を規定し，どちらを採用するかは実装定義とする．具体的には，簡略化した優先度制御規則においては，タスクの現在優先度を高くする方向の変更はすべて行うのに対して，現在優先度を低くする方向の変更は，タスクがロックしているミューテックスがない時（または，なくなった時）にのみ行う（この場合には，タスクの現在優先度をベース優先度に一致させる）．より具体的には，次の状況でのみ現在優先度を変更する処理を行えばよい．

- タスクがロックしているTA_INHERIT属性のミューテックスを，そのタスクよりも高い現在優先度を持つタスクが待ち始めた時
- タスクがロックしている TA_INHERIT 属性のミューテックスを待っているタスクが，前者のタスクよりも高い現在優先度に変更された時
- タスクが，そのタスクの現在優先度よりも高い上限優先度を持つTA_CEILING属性のミューテックスをロックした時
- タスクがロックしているミューテックスがなくなった時
- chg_priによりタスクのベース優先度を変更した場合で，そのタスクがロックしているミューテックスがないか，変更後のベース優先度が現在優先度よりも高い時

ミューテックスの操作に伴ってタスクの現在優先度を変更した場合には，次の処理を行う．優先度を変更されたタスクが実行できる状態である場合，タスク

の優先順位を，変更後の優先度にしたがって変化させる．変更後の優先度と同じ優先度を持つタスクの間での優先順位は，実装依存である．優先度が変更されたタスクが何らかのタスク優先度順の待ち行列につながれている場合にも，その待ち行列の中での順序を，変更後の優先度にしたがって変化させる．変更後の優先度と同じ優先度を持つタスクの間での順序は，実装依存である．

タスクが終了する時に，そのタスクがロックしているミューテックスが残っている場合には，それらのミューテックスをすべてロック解除する．ロックしているミューテックスが複数ある場合には，それらをロック解除する順序は実装依存である．ロック解除の具体的な処理内容については，unl_mtx の機能説明を参照すること．

ミューテックス生成情報およびミューテックス状態のパケット形式として，次のデータ型を定義する．

```
typedef struct t_cmtx {
        ATR         mtxatr ;      /* ミューテックス属性 */
        PRI         ceilpri ;     /* ミューテックスの上限優先度 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_CMTX ;

typedef struct t_rmtx {
        ID          htskid ;      /* ミューテックスをロックしているタ
                                     スクのID番号 */
        ID          wtskid ;      /* ミューテックスの待ち行列の先頭の
                                     タスクのID番号 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RMTX ;
```

ミューテックス機能の各サービスコールの機能コードは次の通りである．

TFN_CRE_MTX	–0x81	cre_mtxの機能コード
TFN_ACRE_MTX	–0xc6	acre_mtxの機能コード
TFN_DEL_MTX	–0x82	del_mtxの機能コード
TFN_LOC_MTX	–0x85	loc_mtxの機能コード
TFN_PLOC_MTX	–0x86	ploc_mtxの機能コード
TFN_TLOC_MTX	–0x87	tloc_mtxの機能コード
TFN_UNL_MTX	–0x83	unl_mtxの機能コード
TFN_REF_MTX	–0x88	ref_mtxの機能コード

【補足説明】

TA_TFIFO属性またはTA_TPRI属性のミューテックスは，最大資源数が1のセマフォ（バイナリセマフォ）と同等の機能を持つ．ただし，ミューテックスは，ロックしたタスク以外はロック解除できない，タスク終了時に自動的にロック解除されるなどの違いがある．

ここでいう優先度上限プロトコルは，広い意味での優先度上限プロトコルで，最初に優先度上限プロトコルとして提案されたアルゴリズムではない．厳密には，highest locker protocolなどと呼ばれているアルゴリズムである．

ミューテックスの操作に伴ってタスクの現在優先度を変更した結果，優先度を変更されたタスクのタスク優先度順の待ち行列の中での順序が変化した場合，優先度を変更されたタスクないしはその待ち行列で待っている他のタスクの待ち解除が必要になる場合がある（snd_mbfの機能説明とget_mplの機能説明を参照）．

【μITRON3.0仕様との相違】

ミューテックス機能は，μITRON4.0仕様において新たに導入した機能である．ミューテックスをセマフォとは別のオブジェクトとして導入したのは，計数型セマフォで優先度継承プロトコルをサポートするのが難しいためである．

【仕様決定の理由】

ミューテックスの操作に伴ってタスクの現在優先度を変更した場合に，変更後の優先度と同じ優先度を持つタスクの間での優先順位を実装依存としたのは，次の理由による．アプリケーションによっては，ミューテックス機能による現在優先度の変更が頻繁に発生する可能性があり，それに伴ってタスク切替えが多発するのは望ましくない（同じ優先度を持つタスクの間での優先順位を最低とすると，不必要なタスク切替えが起こる）．理想的には，タスクの優先度ではなく優先順位を継承するのが望ましいが，このような仕様にすると実装上のオーバヘッドが大きくなるため，実装依存とすることにした．

CRE_MTX ミューテックスの生成（静的API）
cre_mtx ミューテックスの生成
acre_mtx ミューテックスの生成（ID番号自動割付け）

【静的API】

CRE_MTX (ID mtxid, { ATR mtxatr, PRI ceilpri }) ;

【C言語API】

ER ercd = cre_mtx (ID mtxid, T_CMTX *pk_cmtx) ;

ER_ID mtxid = acre_mtx (T_CMTX *pk_cmtx) ;

【パラメータ】

ID	mtxid	生成対象のミューテックスのID番号（acre_mtx以外）
T_CMTX *	pk_cmtx	ミューテックス生成情報を入れたパケットへのポインタ（CRE_MTXではパケットの内容を直接記述する）

pk_cmtxの内容（T_CMTX型）

ATR	mtxatr	ミューテックス属性
PRI	ceilpri	ミューテックスの上限優先度

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cre_mtxの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acre_mtxの場合

ER_ID	mtxid	生成したミューテックスのID番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mtxidが不正あるいは使用できない；cre_mtxのみ）
E_NOID	ID番号不足（割付け可能なミューテックスIDがない；acre_mtxのみ）
E_RSATR	予約属性（mtxatrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_cmtx，ceilpriが不正）
E_OBJ	オブジェクト状態エラー（対象ミューテックスが登録済み；cre_mtxのみ）

【機能】

mtxidで指定されるID番号を持つミューテックスを，pk_cmtxで指定されるミューテックス生成情報に基づいて生成する．mtxatrはミューテックスの属性，

ceilpriはミューテックスの上限優先度である．ceilpriは，mtxatrにTA_CEILING（＝0x03）が指定された場合にのみ有効である．

CRE_MTXにおいては，mtxidは自動割付け対応整数値パラメータ，mtxatrはプリプロセッサ定数式パラメータである．

acre_mtxは，生成するミューテックスのID番号をミューテックスが登録されていないID番号の中から割り付け，割り付けたID番号を返値として返す．

mtxatrには，（TA_TFIFO ‖ TA_TPRI ‖ TA_INHERIT ‖ TA_CEILING）の指定ができる．ミューテックスの待ち行列は，TA_TFIFO（＝0x00）が指定された場合にはFIFO順，その他が指定された場合にはタスクの優先度順となる．また，TA_INHERIT（＝0x02）が指定された場合には優先度継承プロトコル，TA_CEILING（＝0x03）が指定された場合には優先度上限プロトコルにしたがって，タスクの現在優先度を制御する．

del_mtx　　ミューテックスの削除

【C言語API】

ER ercd = del_mtx (ID mtxid) ;

【パラメータ】

ID	mtxid	削除対象のミューテックスのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mtxidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象ミューテックスが未登録）

【機能】

mtxidで指定されるミューテックスを削除する．

【補足説明】

対象ミューテックスがロックされていた場合，それをロックしていたタスクにとっては，ロックしているミューテックスが減ることになる．そのため，対象ミューテックスがTA_INHERIT属性かTA_CEILING属性の場合には，それをロックしていたタスクの現在優先度の変更が必要になる場合がある．簡略化した優先度制御規則を採用している場合には，ミューテックスを削除した結果，タスクがロックしているミューテックスがなくなった場合にのみ，タスクの現在優先度を変更する．

削除されたミューテックスをロックしているタスクには，ミューテックスが削除されたことはすぐには通知されず，後でミューテックスをロック解除しようとした時点でエラーが報告される．タスクがミューテックスをロックしている間にミューテックスが削除されるのが不都合な場合には，ミューテックスを削除するタスクが，ミューテックスをロックした状態で削除すればよい．

対象ミューテックスのロックを待っているタスクがある場合の扱いについては，3.8節を参照すること．

loc_mtx ミューテックスのロック
ploc_mtx ミューテックスのロック（ポーリング）
tloc_mtx ミューテックスのロック（タイムアウトあり）

【C言語API】

```
ER ercd = loc_mtx ( ID mtxid ) ;
ER ercd = ploc_mtx ( ID mtxid ) ;
ER ercd = tloc_mtx ( ID mtxid, TMO tmout ) ;
```

【パラメータ】

ID	mtxid	ロック対象のミューテックスのID番号
TMO	tmout	タイムアウト指定（tloc_mtxのみ）

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mtxidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象ミューテックスが未登録）
E_PAR	パラメータエラー（tmoutが不正；tloc_mtxのみ）
E_ILUSE	サービスコール不正使用（ミューテックスの多重ロック，上限優先度の違反）
E_RLWAI	待ち状態の強制解除（待ち状態の間に rel_wai を受付；ploc_mtx以外）
E_TMOUT	ポーリング失敗またはタイムアウト（loc_mtx以外）
E_DLT	待ちオブジェクトの削除（待ち状態の間に対象ミューテックスが削除；ploc_mtx以外）

【機能】

mtxidで指定されるミューテックスをロックする．具体的には，対象ミューテックスがロックされていない場合には，自タスクがミューテックスをロックした状態にし，自タスクを待ち状態とせずにサービスコールの処理を終了する．対象ミューテックスがロックされている場合には，自タスクを待ち行列につなぎ，ミューテックスのロック待ち状態に移行させる．

他のタスクがすでに待ち行列につながっている場合，自タスクを待ち行列につなぐ処理は次のように行う．ミューテックス属性にTA_TFIFO（＝0x00）が指定されている場合には，自タスクを待ち行列の末尾につなぐ．その他の属性が指定されている場合には，自タスクを優先度順で待ち行列につなぐ．同じ優先度のタスクの中では，自タスクを最後につなぐ．

自タスクがすでに対象ミューテックスをロックしている場合には，E_ILUSEエラーを返す．また，対象ミューテックスがTA_CEILING属性の場合で，自タスクのベース優先度が対象ミューテックスの上限優先度よりも高い場合にも，

E_ILUSEエラーを返す.

ploc_mtxは，loc_mtxの処理をポーリングで行うサービスコール，tloc_mtxは，loc_mtxにタイムアウトの機能を付け加えたサービスコールである. tmoutには，正の値のタイムアウト時間に加えて，TMO_POL（＝0）とTMO_FEVR（＝–1）を指定することができる.

【補足説明】

TA_INHERIT属性のミューテックスを対象にサービスコールが呼び出され，自タスクをミューテックスのロック待ち状態に移行させた場合で，自タスクの現在優先度が対象ミューテックスをロックしているタスクの現在優先度よりも高い場合には，後者のタスクの現在優先度を，前者のタスクの現在優先度に変更する.

また，TA_INHERIT属性のミューテックスのロックを待っているタスクを，タイムアウトまたはrel_waiを受け付けたことにより待ち解除する場合には, そのミューテックスをロックしているタスクの現在優先度の変更が必要になる場合がある．簡略化した優先度制御規則を採用している場合には，この変更は行わない.

TA_CEILING属性のミューテックスを対象にサービスコールが呼び出され，自タスクがミューテックスをロックした場合で，対象ミューテックスの上限優先度が自タスクの現在優先度よりも高い場合には，自タスクの現在優先度を，ミューテックスの上限優先度に変更する.

tloc_mtxは，tmoutにTMO_POLが指定された場合，E_CTXエラーにならない限りはploc_mtxと全く同じ動作をする．また，tmoutにTMO_FEVRが指定された場合は，loc_mtxと全く同じ動作をする.

unl_mtx　ミューテックスのロック解除

【C言語API】

ER ercd = unl_mtx (ID mtxid) ;

【パラメータ】

ID	mtxid	ロック解除対象のミューテックスのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mtxidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象ミューテックスが未登録）
E_ILUSE	サービスコール不正使用（対象ミューテックスをロックしていない）

【機能】

mtxidで指定されるミューテックスをロック解除する．具体的には，対象ミューテックスに対してロックを待っているタスクがある場合には，待ち行列の先頭のタスクを待ち解除し，待ち解除されたタスクが対象ミューテックスをロックした状態にする．この時，待ち解除されたタスクに対しては，待ち状態に入ったサービスコールの返値としてE_OKを返す．ロックを待っているタスクがない場合には，対象ミューテックスをロックされていない状態にする．

対象ミューテックスを自タスクがロックしていない場合には，E_ILUSEエラーを返す．

【補足説明】

対象ミューテックスがTA_INHERIT属性かTA_CEILING属性の場合には，このサービスコールを呼び出したことにより，自タスクの現在優先度の変更が必要になる場合がある．簡略化した優先度制御規則を採用している場合には，ミューテックスをロック解除した結果，自タスクがロックしているミューテックスがなくなった場合にのみ，自タスクの現在優先度を変更する．

ref_mtx　　ミューテックスの状態参照

【C言語API】

ER ercd = ref_mtx (ID mtxid, T_RMTX *pk_rmtx) ;

【パラメータ】

ID	mtxid	状態参照対象のミューテックスのID番号
T_RMTX *	pk_rmtx	ミューテックス状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rmtxの内容（T_RMTX型）

ID	htskid	ミューテックスをロックしているタスクの ID番号
ID	wtskid	ミューテックスの待ち行列の先頭のタスクのID番号

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（mtxidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象ミューテックスが未登録）
E_PAR	パラメータエラー（pk_rmtxが不正）

【機能】

mtxidで指定されるミューテックスに関する状態を参照し，pk_rmtxで指定されるパケットに返す．

htskidには，対象ミューテックスをロックしているタスクのID番号を返す．対象ミューテックスがロックされていない場合には，TSK_NONE（＝0）を返す．

wtskidには，対象ミューテックスの待ち行列の先頭のタスクのID番号を返す．ロックを待っているタスクがない場合には，TSK_NONE（＝0）を返す．

【補足説明】

htskid＝TSK_NONEとwtskid≠TSK_NONEが同時に成立することはない．

4.5.2 メッセージバッファ

メッセージバッファは，可変長のメッセージを受渡しすることにより，同期と通信を行うためのオブジェクトである．メッセージバッファ機能には，メッセージバッファを生成／削除する機能，メッセージバッファに対してメッセージを送信／受信する機能，メッセージバッファの状態を参照する機能が含まれる．メッセージバッファはID番号で識別されるオブジェクトである．メッセージバッファのID番号をメッセージバッファ IDと呼ぶ．

メッセージバッファは，メッセージの送信を待つタスクの待ち行列（送信待ち行列）とメッセージの受信を待つタスクの待ち行列（受信待ち行列）を持つ．また，送信されたメッセージを格納するためのメッセージバッファ領域を持つ．メッセージを送信する側（イベントを知らせる側）では，送信したいメッセージをメッセージバッファにコピーする．メッセージバッファ領域の空き領域が足りなくなった場合，メッセージバッファ領域に十分な空きができるまでメッセージバッファへの送信待ち状態になる．メッセージバッファへの送信待ち状態になったタスクは，そのメッセージバッファの送信待ち行列につながれる．一方，メッセージを受信する側（イベントを待つ側）では，メッセージバッファに入っているメッセージを一つ取り出す．メッセージバッファにメッセージが入っていない場合は，次にメッセージが送られてくるまでメッセージバッファからの受信待ち状態になる．メッセージバッファからの受信待ち状態になったタスクは，そのメッセージバッファの受信待ち行列につながれる．

メッセージバッファ領域のサイズを0にすることで，同期メッセージ機能を実現することができる．すなわち，送信側のタスクと受信側のタスクが，それぞれ相手のタスクがサービスコールを呼び出すのを待ち合わせ，両者がサービスコールを呼び出した時点で，メッセージの受渡しが行われる．

メッセージバッファ機能に関連して，次のカーネル構成マクロを定義する．

SIZE mbfsz = TSZ_MBF (UINT msgcnt, UINT msgsz)

サイズがmsgszバイトのメッセージをmsgcnt個格納するのに必要なメッセージバッファ領域のサイズ（目安のバイト数）

このマクロは，あくまでもメッセージバッファ領域のサイズを決める際の目安として使うためのものである．このマクロを使って，異なるサイズのメッセージを格納するのに必要なサイズを決定することはできない．

メッセージバッファ生成情報およびメッセージバッファ状態のパケット形式として，次のデータ型を定義する．

```
typedef struct t_cmbf {
        ATR         mbfatr ;      /* メッセージバッファ属性 */
        UINT        maxmsz ;      /* メッセージの最大サイズ（バイト
                                      数）*/
        SIZE        mbfsz ;       /* メッセージバッファ領域のサイズ
                                      （バイト数）*/
```

```
        VP          mbf ;       /* メッセージバッファ領域の先頭番
                                   地 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_CMBF ;

typedef struct t_rmbf {
        ID          stskid ;    /* メッセージバッファの送信待ち行列
                                   の先頭のタスクのID番号 */
        ID          rtskid ;    /* メッセージバッファの受信待ち行列
                                   の先頭のタスクのID番号 */
        UINT        smsgcnt ;   /* メッセージバッファに入っている
                                   メッセージの数 */
        SIZE        fmbfsz ;    /* メッセージバッファ領域の空き領域
                                   のサイズ（バイト数，最低限の管理
                                   領域を除く）*/
        /* 実装独自に他のフィールドを追加してもよい */
} T_RMBF ;
```

メッセージバッファ機能の各サービスコールの機能コードは次の通りである．

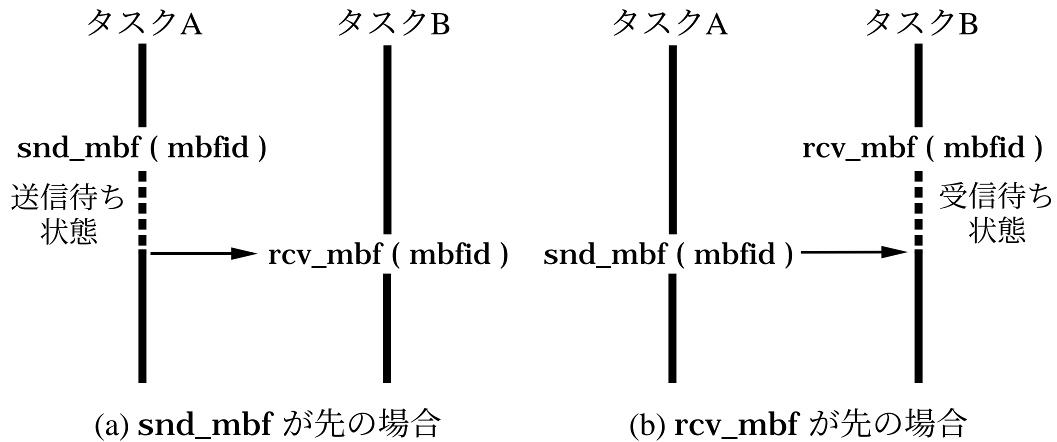
TFN_CRE_MBF	–0x89	cre_mbfの機能コード
TFN_ACRE_MBF	–0xc7	acre_mbfの機能コード
TFN_DEL_MBF	–0x8a	del_mbfの機能コード
TFN_SND_MBF	–0x8d	snd_mbfの機能コード
TFN_PSND_MBF	–0x8e	psnd_mbfの機能コード
TFN_TSND_MBF	–0x8f	tsnd_mbfの機能コード
TFN_RCV_MBF	–0x91	rcv_mbfの機能コード
TFN_PRCV_MBF	–0x92	prcv_mbfの機能コード
TFN_TRCV_MBF	–0x93	trcv_mbfの機能コード
TFN_REF_MBF	–0x94	ref_mbfの機能コード

【補足説明】

メッセージバッファ領域のサイズを0にした場合の，メッセージバッファの動作を図4-2の例を用いて説明する．この図で，タスクAとタスクBは非同期に実行しているものとする．

- もしタスクAが先にsnd_mbfを呼び出した場合には，タスクBがrcv_mbfを呼び出すまでタスクAは待ち状態となる．この時タスクAは，メッセージバッファへの送信待ち状態になっている（図4-2 (a)）．
- 逆にタスクBが先にrcv_mbfを呼び出した場合には，タスクAがsnd_mbfを呼び出すまでタスクBは待ち状態となる．この時タスクBは，メッセージバッファからの受信待ち状態になっている（図4-2 (b)）．
- タスクAがsnd_mbfを呼び出し，タスクBがrcv_mbfを呼び出した時点で，タスクAからタスクBへメッセージの受渡しが行われる．その後は，両タスクとも実行できる状態となる．

メッセージバッファへの送信を待っているタスクは，待ち行列につながれている順序でメッセージを送信する．例えば，あるメッセージバッファに対して40



(a) **snd_mbf** が先の場合 (b) **rcv_mbf** が先の場合

図4-2. メッセージバッファによる同期通信

バイトのメッセージを送信しようとしているタスクAと，10バイトのメッセージを送信しようとしているタスクBが，この順で待ち行列につながれている時に，別のタスクによるメッセージの受信により20バイトの空き領域ができたとする．このような場合でも，タスクAがメッセージを送信するまで，タスクBはメッセージを送信できない．ただし，このような場合にタスクBに先にメッセージを送信させる指定を，実装独自にメッセージバッファ属性に追加することは許される．

メッセージバッファは，可変長のメッセージをコピーして受渡しする．データキューとの違いは，メッセージの長さが可変であることである．一方，メールボックスとの違いは，メッセージをコピーすることである．

メッセージバッファは，リングバッファで実装することを想定している．

カーネルのエラーログ用（サービスコールを呼び出した処理に通知できないエラーの記録用）にメッセージバッファを用いる場合，ID番号が（–4）のメッセージバッファを用いることができる．さらに，デバッグサポート機能との通信用などに，カーネル内部でメッセージバッファを用いる場合には，ID番号が（–3）～（–2）のメッセージバッファを用いることができる．また，アプリケーションプログラムからこれらのメッセージバッファにアクセスする方法を限定することができる．

【μITRON3.0仕様との相違】

メッセージバッファでメッセージの送信を待っているタスクが，待ち行列につながれている順序でメッセージを送信するか，メッセージを送信できるものから先に送信するかが実装依存となっていたのを，前者に標準化した．

CRE_MBF　メッセージバッファの生成（静的API）
cre_mbf　メッセージバッファの生成
acre_mbf　メッセージバッファの生成（ID番号自動割付け）

【静的API】

```
CRE_MBF ( ID mbfid, { ATR mbfatr, UINT maxmsz, SIZE mbfsz,
                    VP mbf } ) ;
```

【C言語API】

```
ER ercd = cre_mbf ( ID mbfid, T_CMBF *pk_cmbf ) ;
ER_ID mbfid = acre_mbf ( T_CMBF *pk_cmbf ) ;
```

【パラメータ】

ID	mbfid	生成対象のメッセージバッファの ID 番号（acre_mbf以外）
T_CMBF *	pk_cmbf	メッセージバッファ生成情報を入れたパケットへのポインタ（CRE_MBF ではパケットの内容を直接記述する）

pk_cmbfの内容（T_CMBF型）

ATR	mbfatr	メッセージバッファ属性
UINT	maxmsz	メッセージの最大サイズ（バイト数）
SIZE	mbfsz	メッセージバッファ領域のサイズ（バイト数）
VP	mbf	メッセージバッファ領域の先頭番地

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cre_mbfの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acre_mbfの場合

ER_ID	mbfid	生成したメッセージバッファの ID 番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mbfidが不正あるいは使用できない；cre_mbfのみ）
E_NOID	ID 番号不足（割付け可能なメッセージバッファ ID がない；acre_mbfのみ）
E_NOMEM	メモリ不足（メッセージバッファ領域などが確保できない）
E_RSATR	予約属性（mbfatrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_cmbf, maxmsz, mbfsz, mbfが不正）
E_OBJ	オブジェクト状態エラー（対象メッセージバッファが登

録済み；cre_mbfのみ）

【機能】

mbfidで指定されるID番号を持つメッセージバッファを, pk_cmbfで指定されるメッセージバッファ生成情報に基づいて生成する．mbfatrはメッセージバッファの属性，maxmszはメッセージバッファに送信できるメッセージの最大サイズ（バイト数），mbfszはメッセージバッファ領域のサイズ（バイト数），mbfはメッセージバッファ領域の先頭番地である．

CRE_MBFにおいては，mbfidは自動割付け対応整数値パラメータ，mbfatrとmbfszはプリプロセッサ定数式パラメータである．

acre_mbfは, 生成するメッセージバッファのID番号をメッセージバッファが登録されていないID番号の中から割り付け，割り付けたID番号を返値として返す．

mbfatrには，（TA_TFIFO ‖ TA_TPRI）の指定ができる．メッセージバッファの送信待ち行列は，TA_TFIFO（＝0x00）が指定された場合にはFIFO順，TA_TPRI（＝0x01）が指定された場合にはタスクの優先度順となる．

mbfで指定された番地からmbfszバイトのメモリ領域を, メッセージバッファ領域として使用する．メッセージバッファ領域内には，メッセージを管理するための情報も置くため, メッセージバッファ領域のすべてをメッセージを格納するために使えるわけではない．TSZ_MBFを用いると，アプリケーションプログラムから, mbfszに指定すべきサイズの目安を知ることができる. mbfにNULL（＝0）が指定された場合には, mbfszで指定されたサイズのメモリ領域を，カーネルが確保する．mbfszに0を指定することも可能である．

maxmszに0が指定された場合や, 実装定義の最大値よりも大きい値が指定された場合には，E_PARエラーを返す．また，mbfszに実装定義の最大値よりも大きい値が指定された場合にも，E_PARエラーを返す．

【補足説明】

メッセージバッファの受信待ち行列は，常にFIFO順となる．また，メッセージバッファにメッセージを入れる順序も，FIFO順のみをサポートしている．

mbfにNULLが指定された場合にカーネルが確保するメッセージバッファ領域のサイズは，mbfszに指定されたサイズ以上であれば，それよりも大きくてもよい．

【μITRON3.0仕様との相違】

メッセージバッファ属性のTA_TPRIは, μITRON3.0仕様では受信待ち行列をタスクの優先度順とする指定であったが, 送信待ち行列をタスクの優先度順とする指定に変更した．これは，送信待ち行列をタスクの優先度順とする機能の方が有用性が高いためである．

メッセージバッファ生成情報に，メッセージバッファ領域の先頭番地（mbf）を追加し，拡張情報を削除した．パラメータの名称をbufszからmbfszに変更し, メッセージバッファ生成情報を入れたパケット中でのmaxmszとmbfszの順

序を交換した．また，maxmszのデータ型をINTからUINTに，mbfszのデータ型をINTからSIZEにそれぞれ変更した．

acre_mbfは新設のサービスコールである．

del_mbf　　メッセージバッファの削除

【C言語API】

ER ercd = del_mbf (ID mbfid) ;

【パラメータ】

ID	mbfid	削除対象のメッセージバッファのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mbfidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象メッセージバッファが未登録）

【機能】

mbfidで指定されるメッセージバッファを削除する．メッセージバッファ領域をカーネルで確保した場合には，その領域を解放する．

【補足説明】

対象メッセージバッファに入っていたメッセージは破棄される．対象メッセージバッファで送信または受信を待っているタスクがある場合の扱いについては，3.8節を参照すること．

---

snd_mbf メッセージバッファへの送信
psnd_mbf メッセージバッファへの送信（ポーリング）
tsnd_mbf メッセージバッファへの送信（タイムアウトあり）

【C言語API】

```
ER ercd = snd_mbf ( ID mbfid, VP msg, UINT msgsz ) ;
ER ercd = psnd_mbf ( ID mbfid, VP msg, UINT msgsz ) ;
ER ercd = tsnd_mbf ( ID mbfid, VP msg, UINT msgsz, TMO tmout ) ;
```

【パラメータ】

ID	mbfid	送信対象のメッセージバッファのID番号
VP	msg	送信メッセージの先頭番地
UINT	msgsz	送信メッセージのサイズ（バイト数）
TMO	tmout	タイムアウト指定（tsnd_mbfのみ）

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mbfidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象メッセージバッファが未登録）
E_PAR	パラメータエラー（msg，msgsz，tmoutが不正）
E_RLWAI	待ち状態の強制解除（待ち状態の間に rel_wai を受付；psnd_mbf以外）
E_TMOUT	ポーリング失敗またはタイムアウト（snd_mbf以外）
E_DLT	待ちオブジェクトの削除（待ち状態の間に対象メッセージバッファが削除；psnd_mbf以外）

【機能】

mbfidで指定されるメッセージバッファに，msgで指定される番地からmsgszで指定されるバイト数のメモリ領域に格納されたメッセージを送信する．具体的な処理内容は次の通りである．

対象メッセージバッファで受信を待っているタスクがある場合には，受信待ち行列の先頭のタスクが受信メッセージを格納する領域に送信メッセージをコピーし，そのタスクを待ち解除する．この時，待ち解除されたタスクに対しては，待ち状態に入ったサービスコールの返値として，送信メッセージのサイズ（msgsz）を返す．

受信を待っているタスクがない場合は，自タスクより優先してメッセージを送信できるタスクが待っているかどうかによって処理が異なる．対象メッセージバッファで送信を待っているタスクがない場合や，メッセージバッファ属性にTA_TPRI（＝0x01）が指定されており，送信を待っているタスクの優先度がい

ずれも自タスクの優先度よりも低い場合には，送信メッセージをメッセージバッファの末尾にコピーする．この条件を満たさない場合や，メッセージバッファ領域に送信メッセージを格納するために必要な空き領域がない場合には，自タスクを送信待ち行列につなぎ，メッセージバッファへの送信待ち状態に移行させる．

他のタスクがすでに送信待ち行列につながっている場合，自タスクを送信待ち行列につなぐ処理は次のように行う．メッセージバッファ属性にTA_TFIFO（＝0x00）が指定されている場合には，自タスクを送信待ち行列の末尾につなぐ．TA_TPRI（＝0x01）が指定されている場合には，自タスクを優先度順で送信待ち行列につなぐ．同じ優先度のタスクの中では，自タスクを最後につなぐ．

メッセージバッファで送信を待っているタスクが，rel_waiやter_tskにより待ち解除されたり，タイムアウトにより待ち解除された結果，送信待ち行列の先頭のタスクが変化する時には，新たに先頭になったタスクから順に可能ならメッセージを送信させる処理を行う必要がある．具体的な処理内容は，rcv_mbfによってメッセージバッファからメッセージが取り出された後の処理と同様であるため，rcv_mbfの機能説明を参照すること．また，メッセージバッファで送信を待っているタスクの優先度が，chg_priやミューテックスの操作によって変更された結果，メッセージバッファの送信待ち行列の先頭のタスクが変化する時にも，同様の処理が必要である．

psnd_mbfは，snd_mbfの処理をポーリングで行うサービスコール，tsnd_mbfは，snd_mbfにタイムアウトの機能を付け加えたサービスコールである．tmoutには，正の値のタイムアウト時間に加えて，TMO_POL（＝0）とTMO_FEVR（＝–1）を指定することができる．

msgszが，メッセージバッファの最大メッセージサイズよりも大きい場合には，E_PARエラーを返す．また，msgszに0が指定された場合にも，E_PARエラーを返す．

【補足説明】

tsnd_mbfは，tmoutにTMO_POLが指定された場合，E_CTXエラーにならない限りはpsnd_mbfと全く同じ動作をする．また，tmoutにTMO_FEVRが指定された場合は，snd_mbfと全く同じ動作をする．

【μITRON3.0仕様との相違】

msgszのデータ型をINTからUINTに変更した．

rcv_mbf メッセージバッファからの受信
prcv_mbf メッセージバッファからの受信（ポーリング）
trcv_mbf メッセージバッファからの受信（タイムアウトあり）

【C言語API】

ER_UINT msgsz = rcv_mbf (ID mbfid, VP msg) ;
ER_UINT msgsz = prcv_mbf (ID mbfid, VP msg) ;
ER_UINT msgsz = trcv_mbf (ID mbfid, VP msg, TMO tmout) ;

【パラメータ】

ID	mbfid	受信対象のメッセージバッファのID番号
VP	msg	受信メッセージを格納する先頭番地
TMO	tmout	タイムアウト指定（trcv_mbfのみ）

【リターンパラメータ】

ER_UINT	msgsz	受信メッセージのサイズ（バイト数，正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mbfidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象メッセージバッファが未登録）
E_PAR	パラメータエラー（msg，tmoutが不正）
E_RLWAI	待ち状態の強制解除（待ち状態の間に rel_wai を受付；prcv_mbf以外）
E_TMOUT	ポーリング失敗またはタイムアウト（rcv_mbf以外）
E_DLT	待ちオブジェクトの削除（待ち状態の間に対象メッセージバッファが削除；prcv_mbf以外）

【機能】

mbfidで指定されるメッセージバッファからメッセージを受信し，msgで指定される番地以降に格納する．受信したメッセージのバイト数は，msgszに返す．具体的な処理内容は次の通りである．

対象メッセージバッファにメッセージが格納されている場合には，その先頭のメッセージをmsgで指定された番地以降にコピーし，そのメッセージサイズをmsgszに返す．コピーしたメッセージは，メッセージバッファ領域から削除する．メッセージバッファで送信を待っているタスクがある場合には，メッセージを削除した結果，メッセージバッファ領域に，送信待ち行列の先頭のタスクが送信しようとしているメッセージを格納するために必要な空き領域ができたかを調べる．格納できる場合には，そのタスクが送信しようとしているメッセージをメッセージバッファの末尾にコピーし，そのタスクを待ち解除する．この時，待ち解除されたタスクに対しては，待ち状態に入ったサービスコール

の返値としてE_OKを返す．さらに，送信を待っているタスクが残っている場合には，新たに送信待ち行列の先頭になったタスクに対して同じ処理を繰り返す．

メッセージが格納されていない場合で，対象メッセージバッファで送信を待っているタスクがある場合には（このような状況が起こるのは，メッセージバッファ領域のサイズが，メッセージのサイズによっては一つのメッセージも格納できないほど小さい場合のみである），送信待ち行列の先頭のタスクが送信しようとしているメッセージをmsgで指定された番地以降にコピーし，そのタスクを待ち解除する．また，コピーしたメッセージのサイズをmsgszに返す．待ち解除されたタスクに対しては，待ち状態に入ったサービスコールの返値としてE_OKを返す．

メッセージが格納されていない場合で，送信を待っているタスクもない場合には，自タスクを受信待ち行列につなぎ，メッセージバッファからの受信待ち状態に移行させる．他のタスクがすでに受信待ち行列につながっている場合には，自タスクを受信待ち行列の末尾につなぐ．

prcv_mbfは，rcv_mbfの処理をポーリングで行うサービスコール，trcv_mbfは，rcv_mbfにタイムアウトの機能を付け加えたサービスコールである．tmoutには，正の値のタイムアウト時間に加えて，TMO_POL（＝0）とTMO_FEVR（＝–1）を指定することができる．

【補足説明】

これらのサービスコールにより，複数のタスクが待ち解除される場合，メッセージバッファの送信待ち行列につながれていた順序で待ち解除する．そのため，実行可能状態に移行したタスクで同じ優先度を持つものの間では，送信待ち行列の中で前につながれていたタスクの方が高い優先順位を持つことになる．

trcv_mbfは，tmoutにTMO_POLが指定された場合，E_CTXエラーにならない限りはprcv_mbfと全く同じ動作をする．また，tmoutにTMO_FEVRが指定された場合は，rcv_mbfと全く同じ動作をする．

【μITRON3.0仕様との相違】

受信メッセージのサイズ（msgsz）を，サービスコールの返値として返すこととした．パラメータの順序を変更した．また，msgszのデータ型をINTからUINT（実際には，ER_UINT）に変更した．

ref_mbf メッセージバッファの状態参照

【C言語API】

ER ercd = ref_mbf (ID mbfid, T_RMBF *pk_rmbf) ;

【パラメータ】

ID	mbfid	状態参照対象のメッセージバッファのID番号
T_RMBF *	pk_rmbf	メッセージバッファ状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rmbfの内容（T_RMBF型）

ID	stskid	メッセージバッファの送信待ち行列の先頭のタスクのID番号
ID	rtskid	メッセージバッファの受信待ち行列の先頭のタスクのID番号
UINT	smsgcnt	メッセージバッファに入っているメッセージの数
SIZE	fmbfsz	メッセージバッファ領域の空き領域のサイズ（バイト数，最低限の管理領域を除く）

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（mbfidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象メッセージバッファが未登録）
E_PAR	パラメータエラー（pk_rmbfが不正）

【機能】

mbfidで指定されるメッセージバッファに関する状態を参照し，pk_rmbfで指定されるパケットに返す．

stskidには，対象メッセージバッファの送信待ち行列の先頭のタスクのID番号を返す．送信を待っているタスクがない場合には，TSK_NONE（＝0）を返す．

rtskidには，対象メッセージバッファの受信待ち行列の先頭のタスクのID番号を返す．受信を待っているタスクがない場合には，TSK_NONE（＝0）を返す．

smsgcntには，対象メッセージバッファに現在入っているメッセージの個数を返す．

fmbfszには，対象メッセージバッファ領域の空き領域のサイズ（バイト数）から，最低限の管理領域のサイズを引いた値を返す．具体的には，対象メッセージバッファ領域に，最大サイズのメッセージを格納するのに十分な空き領域が

ない場合には，空き領域に格納できるメッセージの最大サイズをfmbfszに返す．対象メッセージバッファ領域に，最大サイズのメッセージを格納できる空き領域がある場合には，メッセージバッファ領域にあとどの程度のメッセージを格納できるかの目安となる値をfmbfszに返す．

【補足説明】

fmbfszに返す値以下のサイズのメッセージであっても，対象メッセージバッファで送信を待っているタスクがある場合（stskid≠TSK_NONEの場合）には，待ち状態に入ることなく送信できるとは限らない．

rtskid≠TSK_NONEとsmsgcnt≠0が同時に成立することはない．また，stskid≠TSK_NONEの時には，fmbfszはメッセージの最大サイズよりも小さい値となる．

【μITRON3.0仕様との相違】

参照できる情報から拡張情報を削除した．待ちタスクの有無ではなく待ち行列の先頭のタスクのID番号を，次に受信されるメッセージのサイズに代えてメッセージバッファに入っているメッセージの数を返すこととした．これらに伴って，リターンパラメータの名称とデータ型を変更した．また，fmbfszに返す値から，最低限の管理領域のサイズを除くこととし，メッセージバッファ領域の空き領域が少ない場合に返す値の意味を厳密化した．

リターンパラメータの名称をfrbufszからfmbfszに変更し，そのデータ型をINTからSIZEに変更した．また，パラメータとリターンパラメータの順序を変更した．

4.5.3 ランデブ

ランデブ機能は，タスク間で同期通信を行うための機能で，あるタスクから別のタスクへの処理依頼と，後者のタスクから前者のタスクへの処理結果の返却を，一連の手順としてサポートする．双方のタスクが待ち合わせるためのオブジェクトを，ランデブポートと呼ぶ．ランデブ機能は，典型的にはクライアント／サーバモデルのタスク間通信を実現するために用いられるが，クライアント／サーバモデルよりも柔軟な同期通信モデルを提供するものである．

ランデブ機能には，ランデブポートを生成／削除する機能，ランデブポートに対して処理の依頼を行う機能（ランデブの呼出し），ランデブポートで処理依頼を受け付ける機能（ランデブの受付），処理結果を返す機能（ランデブの終了），受け付けた処理依頼を他のランデブポートに回送する機能（ランデブの回送）ランデブポートおよびランデブの状態を参照する機能が含まれる．ランデブポートはID番号で識別されるオブジェクトである．ランデブポートのID番号をランデブポートIDと呼ぶ．

ランデブポートに対して処理依頼を行う側のタスク（クライアント側のタスク）は，ランデブポートとランデブ条件，依頼する処理に関する情報を入れたメッセージ（これを呼出しメッセージと呼ぶ）を指定して，ランデブの呼出しを行う．一方，ランデブポートで処理依頼を受け付ける側のタスク（サーバ側のタスク）は，ランデブポートとランデブ条件を指定して，ランデブの受付を行う．

ランデブ条件は，ビットパターンで指定する．あるランデブポートに対して，呼び出したタスクのランデブ条件のビットパターンと，受け付けたタスクのランデブ条件のビットパターンをビット毎に論理積をとり，結果が0以外の場合にランデブが成立する．ランデブを呼び出したタスクは，ランデブが成立するまでランデブ呼出し待ち状態となる．逆に，ランデブを受け付けるタスクは，ランデブが成立するまでランデブ受付待ち状態となる．

ランデブが成立すると，ランデブを呼び出したタスクから受け付けたタスクへ，呼出しメッセージが渡される．ランデブを呼び出したタスクはランデブ終了待ち状態へ移行し，依頼した処理が完了するのを待つ．一方，ランデブを受け付けたタスクは待ち解除され，依頼された処理を行う．ランデブを受け付けたタスクが依頼された処理を完了すると，処理結果を返答メッセージの形で呼び出したタスクに渡し，ランデブを終了する．この時点で，ランデブを呼び出したタスクが，ランデブ終了待ち状態から待ち解除される．

ランデブポートは，ランデブ呼出し待ち状態のタスクをつなぐための呼出し待ち行列と，ランデブ受付待ち状態のタスクをつなぐための受付待ち行列を持つ．それに対して，ランデブが成立した後は，ランデブした双方のタスクはランデブポートから切り離される．すなわち，ランデブポートは，ランデブ終了待ち状態のタスクをつなぐための待ち行列は持たない．また，ランデブを受け付け，依頼された処理を実行しているタスクに関する情報も持たない．

カーネルは，同時に成立しているランデブを識別するために，オブジェクト番

号を付与する．ランデブのオブジェクト番号をランデブ番号と呼ぶ．ランデブ番号の付与方法は実装依存であるが，少なくとも，ランデブを呼び出したタスクを指定するための情報を含んでいなければならない．また，同じタスクが呼び出したランデブであっても，1回目のランデブと2回目のランデブで異なるランデブ番号を付与するなど，できる限りユニークにしなければならない．

ランデブ機能では，次のデータ型を用いる．

RDVPTN	ランデブ条件のビットパターン（符号無し整数）
RDVNO	ランデブ番号

ランデブ機能に関連して，次のカーネル構成定数を定義する．

TBIT_RDVPTN	ランデブ条件のビット数（RDVPTNの有効ビット数）

ランデブポート生成情報，ランデブポート状態，およびランデブ状態のパケット形式として，次のデータ型を定義する．

```
typedef struct t_cpor {
        ATR         poratr ;      /* ランデブポート属性 */
        UINT        maxcmsz ;     /* 呼出しメッセージの最大サイズ（バ
                                     イト数）*/
        UINT        maxrmsz ;     /* 返答メッセージの最大サイズ（バイ
                                     ト数）*/
        /* 実装独自に他のフィールドを追加してもよい */
} T_CPOR ;

typedef struct t_rpor {
        ID          ctskid ;      /* ランデブポートの呼出し待ち行列の
                                     先頭のタスクのID番号 */
        ID          atskid ;      /* ランデブポートの受付待ち行列の先
                                     頭のタスクのID番号 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RPOR ;

typedef struct t_rrdv {
        ID          wtskid ;      /* ランデブ終了待ち状態のタスクのID
                                     番号 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RRDV ;
```

ランデブ機能の各サービスコールの機能コードは次の通りである．

TFN_CRE_POR	–0x95	cre_porの機能コード
TFN_ACRE_POR	–0xc8	acre_porの機能コード
TFN_DEL_POR	–0x96	del_porの機能コード
TFN_CAL_POR	–0x97	cal_porの機能コード
TFN_TCAL_POR	–0x98	tcal_porの機能コード
TFN_ACP_POR	–0x99	acp_porの機能コード
TFN_PACP_POR	–0x9a	pacp_porの機能コード
TFN_TACP_POR	–0x9b	tacp_porの機能コード

TFN_FWD_POR	–0x9c	fwd_porの機能コード
TFN_RPL_RDV	–0x9d	rpl_rdvの機能コード
TFN_REF_POR	–0x9e	ref_porの機能コード
TFN_REF_RDV	–0x9f	ref_rdvの機能コード

【補足説明】

ランデブ機能は，ADAの言語仕様に導入されている同期／通信機能であり，その元になっているのはCSP（Communicating Sequential Processes）である．ただし，ADAのランデブ機能は言語仕様の一部であり，リアルタイムカーネル仕様であるμITRON4.0仕様が提供するランデブ機能とは位置付けが異なる．具体的には，リアルタイムカーネルが提供するランデブ機能は，言語のランデブ機能を実現するためのプリミティブに位置付けられるものであるが，ADAのランデブ機能とμITRON4.0仕様のランデブ機能にはいくつかの相違があり，ADAのランデブを実現するために，μITRON4.0仕様のランデブ機能を使えるとは限らない．

ランデブの動作を図4-3の例を用いて説明する．この図で，タスクAとタスクBは非同期に実行しているものとする．

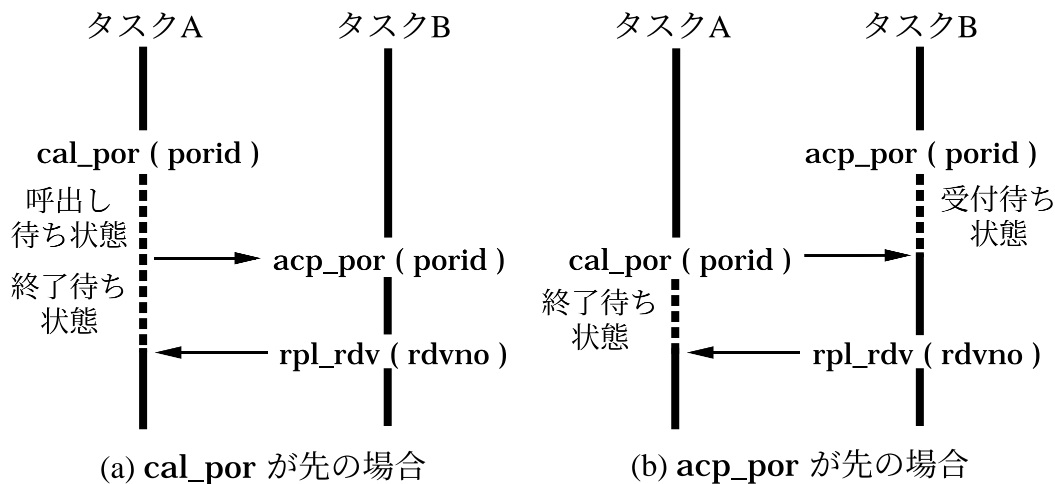


図4-3. ランデブの動作

- もしタスクAが先にcal_porを呼び出した場合には，タスクBがacp_porを呼び出すまでタスクAは待ち状態となる．この時タスクAは，ランデブの呼出し待ち状態になっている（図4-3 (a)）．
- 逆にタスクBが先にacp_porを呼び出した場合には，タスクAがcal_porを呼び出すまでタスクBは待ち状態となる．この時タスクBは，ランデブの受付待ち状態になっている（図4-3 (b)）．
- タスクAがcal_porを呼び出し，タスクBがacp_porを呼び出した時点でランデブが成立し，タスクAを待ち状態としたまま，タスクBが待ち解除される．この時タスクAは，ランデブの終了待ち状態になっている．

• タスクBがrpl_rdvを呼び出した時点で，タスクAは待ち解除される．その後は，両タスクとも実行できる状態となる．

ランデブ番号の具体的な付与方法の例として，ランデブ番号の下位ビットをランデブを呼び出したタスクのID番号，上位ビットをシーケンシャルな番号とする方法がある．例えば，ID番号が16ビットの場合には，シーケンシャルな番号も16ビットとし，ランデブ番号を32ビットの整数値とすることができる．

【μITRON3.0仕様との相違】

ポートに代えて，ランデブポートという用語を用いることとした．

ランデブ条件のビットパターンを入れるパラメータのデータ型を，UINTから専用のデータ型として新設したRDVPTNに変更した．また，ランデブ番号のデータ型を，RNOからRDVNOに変更した．

【仕様決定の理由】

ランデブ機能は，他の同期・通信機能を組み合わせて実現することも可能であるが，返答を伴う通信を行う場合には，それ専用の機能を用意した方が，アプリケーションプログラムが書きやすく，他の同期・通信機能を組み合わせるよりも効率を上げることができると考えられる．一例として，ランデブ機能は，メッセージの受渡しが終わるまで双方のタスクを待たせておくために，メッセージを格納するための領域が必要ないという利点がある．

同じタスクが呼び出したランデブであっても，ランデブ番号をできる限りユニークにしなければならないのは，次の理由による．ランデブが成立してランデブ終了待ち状態となっているタスクが，タイムアウトや待ち状態の強制解除などにより待ち解除された後，再度ランデブを呼び出してランデブが成立した場合を考える．この時，最初のランデブのランデブ番号と，後のランデブのランデブ番号が同一の値であると，最初のランデブを終了させようとした時に，ランデブ番号が同一であるために後のランデブが終了してしまう．2つのランデブに異なるランデブ番号を付与し，ランデブ終了待ち状態のタスクに待ち対象のランデブ番号を記憶しておけば，最初のランデブを終了させようとした時にエラーとすることができる．

CRE_POR	ランデブポートの生成（静的API）
cre_por	ランデブポートの生成
acre_por	ランデブポートの生成（ID番号自動割付け）

【静的API】

CRE_POR (ID porid, { ATR poratr, UINT maxcmsz, UINT maxrmsz }) ;

【C言語API】

ER ercd = cre_por (ID porid, T_CPOR *pk_cpor) ;

ER_ID porid = acre_por (T_CPOR *pk_cpor) ;

【パラメータ】

ID	porid	生成対象のランデブポートのID番号（acre_por以外）
T_CPOR *	pk_cpor	ランデブポート生成情報を入れたパケットへのポインタ（CRE_PORではパケットの内容を直接記述する）

pk_cporの内容（T_CPOR型）

ATR	poratr	ランデブポート属性
UINT	maxcmsz	呼出しメッセージの最大サイズ（バイト数）
UINT	maxrmsz	返答メッセージの最大サイズ（バイト数）

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cre_porの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acre_porの場合

ER_ID	porid	生成したランデブポートのID番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（poridが不正あるいは使用できない；cre_porのみ）
E_NOID	ID番号不足（割付け可能なランデブポートIDがない；acre_porのみ）
E_RSATR	予約属性（poratrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_cpor，maxcmsz，maxrmszが不正）
E_OBJ	オブジェクト状態エラー（対象ランデブポートが登録済み；cre_porのみ）

【機能】

poridで指定されるID番号を持つランデブポートを，pk_cporで指定されるラン

デブポート生成情報に基づいて生成する．poratrはランデブポートの属性，maxcmszは呼出しメッセージの最大サイズ（バイト数），maxrmszは返答メッセージの最大サイズ（バイト数）である．

CRE_PORにおいては，poridは自動割付け対応整数値パラメータ，poratrはプリプロセッサ定数式パラメータである．

acre_porは，生成するランデブポートのID番号をランデブポートが登録されていないID番号の中から割り付け，割り付けたID番号を返値として返す．

poratrには，（TA_TFIFO ‖ TA_TPRI）の指定ができる．ランデブポートの呼出し待ち行列は，TA_TFIFO（＝0x00）が指定された場合にはFIFO順，TA_TPRI（＝0x01）が指定された場合にはタスクの優先度順となる．

maxcmszまたはmaxrmszに，実装定義の最大値よりも大きい値が指定された場合には，E_PARエラーを返す．maxcmszとmaxrmszに0を指定することは可能である．

【補足説明】

ランデブポートの受付待ち行列は，常にFIFO順となる．

【μITRON3.0仕様との相違】

ランデブ属性（TA_TPRI）により，ランデブの呼出し待ち行列をタスクの優先度順とする機能を追加した．

ランデブポート生成情報から，拡張情報を削除した．また，maxcmszとmaxrmszのデータ型を，INTからUINTに変更した．

acre_porは新設のサービスコールである．

del_por ランデブポートの削除

【C言語API】

ER ercd = del_por (ID porid) ;

【パラメータ】

ID	porid	削除対象のランデブポートのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（poridが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象ランデブポートが未登録）

【機能】

poridで指定されるランデブポートを削除する．

【補足説明】

ランデブポートを削除しても，すでに成立したランデブには影響を与えない．ランデブを受け付け，依頼された処理を実行しているタスクには通知されないし，ランデブを呼び出して，ランデブ終了待ち状態となっているタスクもランデブ終了待ち状態のままである．また，ランデブポートが削除されていても，ランデブの終了は正常に実行することができる．

それに対して，対象ランデブポートで呼出しまたは受付を待っているタスクがある場合の扱いについては，3.8節を参照すること．

cal_por ランデブの呼出し
tcal_por ランデブの呼出し（タイムアウトあり）

【C言語API】

```
ER_UINT rmsgsz = cal_por ( ID porid, RDVPTN calptn, VP msg,
                            UINT cmsgsz ) ;
ER_UINT rmsgsz = tcal_por ( ID porid, RDVPTN calptn, VP msg,
                            UINT cmsgsz, TMO tmout ) ;
```

【パラメータ】

ID	porid	呼出し対象のランデブポートのID番号
RDVPTN	calptn	呼出し側のランデブ条件を示すビットパターン
VP	msg	呼出しメッセージの先頭番地／返答メッセージを格納する先頭番地
UINT	cmsgsz	呼出しメッセージのサイズ（バイト数）
TMO	tmout	タイムアウト指定（tcal_porのみ）

【リターンパラメータ】

ER_UINT	rmsgsz	返答メッセージのサイズ（バイト数，正の値または0）またはエラーコード

【エラーコード】

E_ID	不正ID番号（poridが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象ランデブポートが未登録）
E_PAR	パラメータエラー（calptn，msg，cmsgsz，tmoutが不正）
E_RLWAI	待ち状態の強制解除（待ち状態の間にrel_waiを受付）
E_TMOUT	ポーリング失敗またはタイムアウト（tcal_porのみ）
E_DLT	待ちオブジェクトの削除（ランデブの呼出し待ち状態の間に対象ランデブポートが削除）

【機能】

poridで指定されるランデブポートに対して，calptnで指定されるランデブ条件により，ランデブを呼び出す．msgで指定される番地からcmsgszで指定されるバイト数のメモリ領域に格納されたメッセージを，呼出しメッセージとする．また，返答メッセージをmsgで指定される番地以降に格納し，返答メッセージのバイト数をrmsgszに返す．具体的な処理内容は次の通りである．

対象ランデブポートにランデブ受付待ち状態のタスクがあり，そのタスクのランデブ条件とcalptnで指定されたランデブ条件が成立する場合には，ランデブを成立させる．ランデブ受付待ち状態のタスクが複数ある場合には，受付待ち行列の先頭のタスクから順にランデブ条件を調べ，条件が成立する最初のタスクとの間でランデブを成立させる．

この時，成立したランデブにランデブ番号を付与し，自タスクをランデブ終了待ち状態に移行させる．また，成立したランデブの相手タスク（ランデブ受付待ち状態であったタスク）が呼出しメッセージを格納する領域に，msgとcmsgszで指定される呼出しメッセージをコピーし，そのタスクを待ち解除する．待ち解除されたタスクに対しては，待ち状態に入ったサービスコールの返値として呼出しメッセージのサイズ（cmsgsz）を返し，付与したランデブ番号を成立したランデブ番号として返す．

対象ランデブポートにランデブ受付待ち状態のタスクがない場合や，ランデブ受付待ち状態のタスクがあってもランデブ条件が成立しない場合には，自タスクを呼出し待ち行列につなぎ，ランデブ呼出し待ち状態に移行させる．

他のタスクがすでに呼出し待ち行列につながっている場合，自タスクを呼出し待ち行列につなぐ処理は次のように行う．ランデブポート属性にTA_TFIFO（＝0x00）が指定されている場合には，自タスクを呼出し待ち行列の末尾につなぐ．TA_TPRI（＝0x01）が指定されている場合には，自タスクを優先度順で呼出し待ち行列につなぐ．同じ優先度のタスクの中では，自タスクを最後につなぐ．

tcal_porは，cal_porにタイムアウトの機能を付け加えたサービスコールである．tcal_porが呼び出されてから，tmoutで指定された時間が経過してもランデブが終了しない場合に，処理をキャンセルしてE_TMOUTエラーを返す．tmoutには，正の値のタイムアウト時間に加えて，TMO_FEVR（＝–1）を指定することができる．tmoutにTMO_POL（＝0）が指定された場合には，E_PARエラーを返す．

tcal_porが呼び出され，ランデブが成立した後にタイムアウトした場合，一旦成立したランデブを成立する前の状態に戻すことはできず，「サービスコールがエラーコードを返した場合には，サービスコールを呼び出したことによる副作用はない」という原則の例外となっている．この場合，ランデブ相手のタスクには，ランデブを終了させようとした時点でエラーが報告される．cal_porまたはtcal_porでランデブが成立した後のタスクに対してrel_waiが呼び出され，ランデブ終了待ち状態が強制解除された場合（この時，サービスコールはE_RLWAIエラーを返す）も同様である．それに対して，ランデブポートの削除はすでに成立したランデブには影響を与えないため，ランデブが成立した後にE_DLTエラーを返すことはない．

calptnに0が指定された場合には，E_PARエラーを返す．また，cmsgszが，対象ランデブポートの呼出しメッセージの最大サイズよりも大きい場合にも，E_PARエラーを返す．cmsgszに0を指定することは可能である．

【補足説明】

アプリケーションは，呼び出したランデブが回送される可能性がある場合には，期待する返答メッセージのサイズにかかわらず，msgで指定される番地から，対象ランデブポートの返答メッセージの最大サイズ分のメモリ領域を確保すべきである．また，確保したメモリ領域の内容が破壊されることを想定し

て，アプリケーションを記述すべきである．これは，ランデブを回送する際に，回送後の呼出しメッセージをmsgで指定される番地以降にコピーする場合があるためである．

tcal_porは，tmoutにTMO_FEVRが指定された場合，cal_porと全く同じ動作をする．

【μITRON3.0仕様との相違】

tcal_porのタイムアウト時間の解釈を変更した．それに伴って，pcal_porは意味を持たなくなったため，μITRON4.0仕様では廃止した．また，tmoutにTMO_POLが指定された場合には，E_PARエラーを返すこととした．

サイズ0の呼出しメッセージも許すことにした．

返答メッセージのサイズ（rmsgsz）を，サービスコールの返値として返すこととした．calptnのデータ型をUINTからRDVPTNに，cmsgszとrmsgszのデータ型をINTからUINT（実際には，rmsgszはER_UINT）にそれぞれ変更した．また，パラメータとリターンパラメータの順序を変更した．

【仕様決定の理由】

calptnに0が指定された場合をE_PARエラーとしたのは，この指定をした場合，決してランデブが成立せず，ランデブ呼出し待ち状態から抜けることができなくなるためである．

acp_por ランデブの受付
pacp_por ランデブの受付（ポーリング）
tacp_por ランデブの受付（タイムアウトあり）

【C言語API】

```
ER_UINT cmsgsz = acp_por ( ID porid, RDVPTN acpptn, RDVNO *p_rdvno,
                           VP msg ) ;
ER_UINT cmsgsz = pacp_por ( ID porid, RDVPTN acpptn,
                           RDVNO *p_rdvno, VP msg ) ;
ER_UINT cmsgsz = tacp_por ( ID porid, RDVPTN acpptn,
                           RDVNO *p_rdvno, VP msg, TMO tmout ) ;
```

【パラメータ】

ID	porid	受付対象のランデブポートのID番号
RDVPTN	acpptn	受付側のランデブ条件を示すビットパターン
VP	msg	呼出しメッセージを格納する先頭番地
TMO	tmout	タイムアウト指定（tacp_porのみ）

【リターンパラメータ】

ER_UINT	cmsgsz	呼出しメッセージのサイズ（バイト数，正の値または0）またはエラーコード
RDVNO	rdvno	成立したランデブ番号

【エラーコード】

E_ID	不正ID番号（poridが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象ランデブポートが未登録）
E_PAR	パラメータエラー（acpptn，msg，tmoutが不正）
E_RLWAI	待ち状態の強制解除（待ち状態の間に rel_wai を受付；pacp_por以外）
E_TMOUT	ポーリング失敗またはタイムアウト（acp_por以外）
E_DLT	待ちオブジェクトの削除（待ち状態の間に対象ランデブポートが削除；pacp_por以外）

【機能】

poridで指定されるランデブポートに対して，acpptnで指定されるランデブ条件により，ランデブを受け付ける．呼出しメッセージはmsgで指定される番地以降に格納し，呼出しメッセージのバイト数をcmsgszに，成立したランデブ番号をrdvnoに返す．具体的な処理内容は次の通りである．

対象ランデブポートにランデブ呼出し待ち状態のタスクがあり，acpptnで指定されたランデブ条件とそのタスクのランデブ条件が成立する場合には，ランデブを成立させる．ランデブ呼出し待ち状態のタスクが複数ある場合には，呼出し待ち行列の先頭のタスクから順にランデブ条件を調べ，条件が成立する最初

のタスクとの間でランデブを成立させる．

この時，成立したランデブにランデブ番号を付与し，それをrdvnoに返す．また，成立したランデブの相手タスク（ランデブ呼出し待ち状態であったタスク）の呼出しメッセージをmsgで指定された番地以降にコピーし，そのメッセージサイズをcmsgszに返す．ランデブの相手タスクは，ランデブ呼出し待ち行列から外し，ランデブ終了待ち状態に移行させる．

対象ランデブポートにランデブ呼出し待ち状態のタスクがない場合や，ランデブ呼出し待ち状態のタスクがあってもランデブ条件が成立しない場合には，自タスクを受付待ち行列につなぎ，ランデブ受付待ち状態に移行させる．他のタスクがすでに受付待ち行列につながっている場合には，自タスクを受付待ち行列の末尾につなぐ．

pacp_porは，acp_porの処理をポーリングで行うサービスコール，tacp_porは，acp_porにタイムアウトの機能を付け加えたサービスコールである．tmoutには，正の値のタイムアウト時間に加えて，TMO_POL（＝0）とTMO_FEVR（＝–1）を指定することができる．

acpptnに0が指定された場合には，E_PARエラーを返す．

【補足説明】

acp_porにより他のタスクとのランデブが成立したタスクが，そのランデブを終了させる前に，再度acp_porによりランデブを受け付けることも可能である．新たなランデブの受付は，成立しているランデブと異なるランデブポートを対象としたものであっても，同じランデブポートを対象としたものであってもよい．したがって，同じランデブポートを対象とした場合には，一つのタスクが同一のランデブポートに対して複数のランデブを行っている状態になる．さらに，成立しているランデブの相手タスクが，タイムアウトや待ち状態の強制解除などにより待ち解除された後，再度ランデブを呼び出す場合を考えると，一つのタスクが同一のランデブポートに対して同一のタスクを相手に複数のランデブを行っている状態も起こりうる．

tacp_porは，tmoutにTMO_POLが指定された場合，E_CTXエラーにならない限りはpacp_porと全く同じ動作をする．また，tmoutにTMO_FEVRが指定された場合は，acp_porと全く同じ動作をする．

【μITRON3.0仕様との相違】

呼出しメッセージのサイズ（cmsgsz）を，サービスコールの返値として返すことにした．acpptnのデータ型をUINTからRDVPTNに，rdvnoのデータ型をRNOからRDVNOに，cmsgszのデータ型をINTからUINT（実際には，ER_UINT）にそれぞれ変更した．また，パラメータとリターンパラメータの順序を変更した．

【仕様決定の理由】

acpptnに0が指定された場合をE_PARエラーとしたのは，この指定がされた場合，決してランデブが成立せず，ランデブ受付待ち状態から抜けることができなくなるためである．

fwd_por　　ランデブの回送

【C言語API】

ER ercd = fwd_por (ID porid, RDVPTN calptn, RDVNO rdvno, VP msg, UINT cmsgsz) ;

【パラメータ】

ID	porid	回送先のランデブポートのID番号
RDVPTN	calptn	呼出し側のランデブ条件を示すビットパターン
RDVNO	rdvno	回送するランデブ番号
VP	msg	呼出しメッセージの先頭番地
UINT	cmsgsz	呼出しメッセージのサイズ（バイト数）

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（poridが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象ランデブポートが未登録）
E_PAR	パラメータエラー（calptn，msg，cmsgszが不正）
E_ILUSE	サービスコール不正使用（回送先のランデブポートの返答メッセージの最大サイズが大きすぎる）
E_OBJ	オブジェクト状態エラー（rdvnoが不正）

【機能】

rdvnoで指定されるランデブ番号を付与されたランデブを，poridで指定されるランデブポートに対して，calptnで指定されるランデブ条件により回送する．msgで指定される番地からcmsgszで指定されるバイト数のメモリ領域に格納されたメッセージを，回送後の呼出しメッセージとする．

fwd_porを呼び出すと，rdvnoで指定されるランデブを呼び出したタスク（以下，これをタスクAと表記する）が，poridで指定されるランデブポートに対して，fwd_porのパラメータとして指定されたランデブ条件と呼出しメッセージにより，ランデブを呼び出したのと同じ結果になる．

fwd_porの具体的な処理内容は次の通りである．

回送先のランデブポートにランデブ受付待ち状態のタスクがあり，そのタスクのランデブ条件とcalptnで指定されたランデブ条件が成立する場合には，タスクAとそのタスクとの間でランデブを成立させる．ランデブ受付待ち状態のタスクが複数ある場合には，受付待ち行列の先頭のタスクから順にランデブ条件を調べ，条件が成立する最初のタスクとの間でランデブを成立させる．

この時，新たに成立したランデブにランデブ番号を付与し，タスクAを新しいランデブに対するランデブ終了待ち状態に移行させる．また，成立したランデ

ブの相手タスク（ランデブ受付待ち状態であったタスク）が呼出しメッセージを格納する領域に，msg と cmsgsz で指定される呼出しメッセージをコピーし，そのタスクを待ち解除する．待ち解除されたタスクに対しては，待ち状態に入ったサービスコールの返値として呼出しメッセージのサイズ（cmsgsz）を返し，付与したランデブ番号を成立したランデブ番号として返す．

回送先のランデブポートにランデブ受付待ち状態のタスクがない場合や，ランデブ受付待ち状態のタスクがあってもランデブ条件が成立しない場合には，タスクAを回送先のランデブポートの呼出し待ち行列につなぎ，ランデブ呼出し待ち状態に移行させる．この時，タスクAが返答メッセージを格納する領域に，msgとcmsgszで指定される呼出しメッセージをコピーする．

他のタスクがすでに呼出し待ち行列につながっている場合，タスクAを呼出し待ち行列につなぐ処理は，次のように行う．回送先のランデブポート属性にTA_TFIFO（＝0x00）が指定されている場合には，タスクAを呼出し待ち行列の末尾につなぐ．TA_TPRI（＝0x01）が指定されている場合には，タスクAを優先度順で呼出し待ち行列につなぐ．同じ優先度のタスクの中では，タスクAを最後につなぐ．

回送先のランデブポートの返答メッセージの最大サイズは，回送するランデブが成立したランデブポートの返答メッセージの最大サイズ以下でなければならない．この条件が満たされない場合には，E_ILUSEエラーを返す．

cmsgszが，回送先のランデブポートの呼出しメッセージの最大サイズよりも大きい場合や，回送するランデブが成立したランデブポートの返答メッセージの最大サイズよりも大きい場合には，E_PARエラーを返す．cmsgszに0を指定することは可能である．

rdvnoに，他のタスクが受け付けたランデブ番号を指定することも可能である．言い換えると，fwd_por を呼び出してランデブを回送するタスクは，ランデブを受け付けたタスクとは別のタスクであってもよい．

rdvno で指定されたランデブを呼び出したタスクが，指定されたランデブの終了待ち状態でない場合には，E_OBJ エラーを返す．また，rdvno に指定された値が，ランデブ番号として解釈できない場合にも，E_OBJエラーを返す．

calptnに0が指定された場合には，E_PARエラーを返す．

【補足説明】

fwd_porの処理が終わった後は，タスクAがランデブを呼び出した後と同じ状態となることから，ランデブが回送された履歴を記憶しておく必要はない．したがって，回送されてきたランデブをさらに回送することも可能である．

fwd_porの処理はすぐに終了し，fwd_porを呼び出したタスクが待ち状態に入ることはない．msg と cmsgsz で指定される呼出しメッセージは，fwd_por の処理で他の領域にコピーされるため，fwd_por からリターンした後は，アプリケーションは呼出しメッセージが格納されていた領域を別の目的に使うことができる．また，fwd_porを呼び出したタスクは，fwd_porの処理が終わった後には，

回送するランデブが成立したランデブポート，回送先のランデブポート，回送したランデブ，（もしあれば）新たに成立したランデブのいずれとも無関係となる．

tcal_porに対するタイムアウト指定は，tcal_porが呼び出されてからランデブが終了するまでの時間に適用される．そのため，タスクAがtcal_porによりランデブを呼び出していた場合，ランデブの回送後もタイムアウト指定は継続して有効である．

回送先のランデブポートとして，回送するランデブが成立したランデブポートを指定することも可能である．この場合は，一旦受け付けたランデブを，受け付ける前の状態に戻すことになる．ただし，ランデブ条件と呼出しメッセージは，fwd_porで指定されたものを使う．

ランデブを呼び出したタスクが，ランデブが成立した後に，タイムアウトや待ち状態の強制解除などによりランデブ終了待ち状態から待ち解除された場合でも，ランデブを受け付けたタスクには通知されない．この場合，ランデブを受け付けたタスクがfwd_porを呼び出してランデブを回送しようとすると，E_OBJエラーとなる．ランデブの相手タスクが，指定するランデブで終了待ち状態であるかどうかは，ref_rdvを用いて調べることができる．

fwd_porを用いたサーバ分配タスクの動作イメージを図4-4に示す．

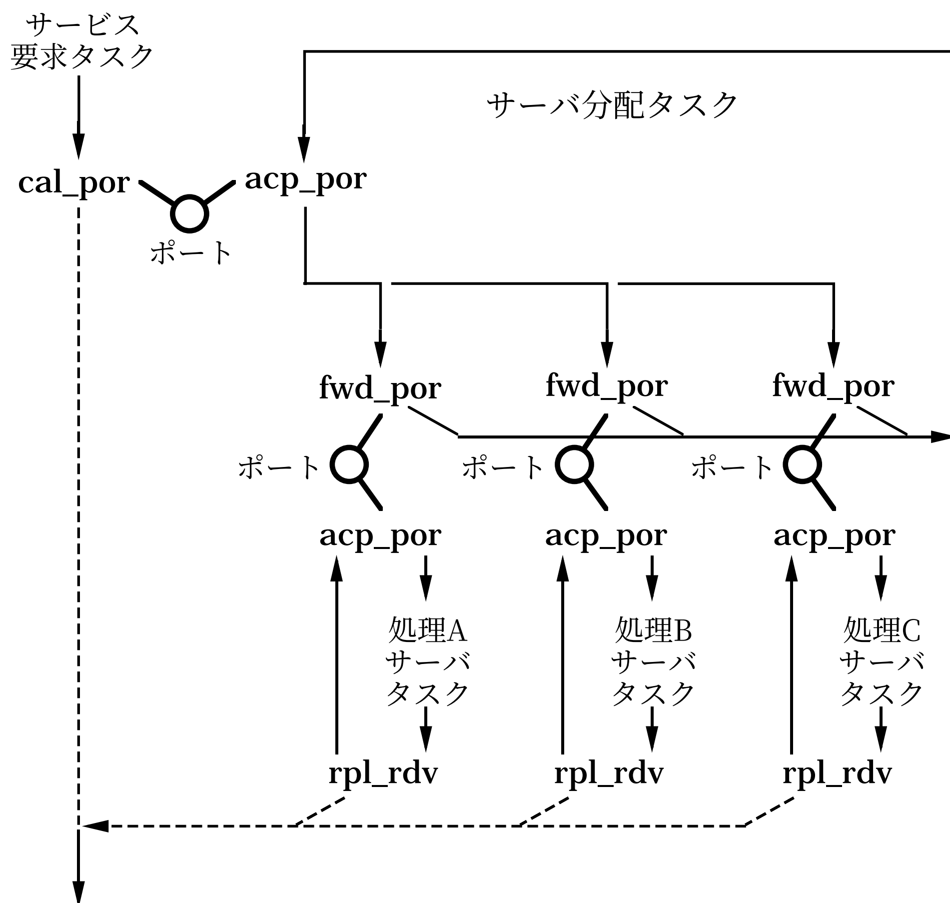


図4-4. fwd_porを使ったサーバ分配タスク

【μITRON3.0仕様との相違】

タスクAがランデブ呼出し待ち状態になる時に，msgとcmsgszで指定される呼出しメッセージを保存する領域を，タスクAが返答メッセージを格納する領域とすることを規定した．

tcal_porのタイムアウト時間の解釈を変更したことに伴って，fwd_porにおけるタイムアウトの扱いが変更になった．

ランデブを受け付けたタスクとは別のタスクがランデブを回送できることを明確にした．

サイズ0の呼出しメッセージも許すことにした．

calptnのデータ型をUINTからRDVPTNに，rdvnoのデータ型をRNOからRDVNOに，cmsgszのデータ型をINTからUINTにそれぞれ変更した．

【仕様決定の理由】

ランデブが回送された履歴を記憶しておく必要がない仕様としたのは，システムで持つべき状態の数を減らすためである．履歴を記憶しておく必要がある用途では，fwd_porを使って回送する代わりに，cal_porを使ってランデブを（多段に）呼び出せばよい．

回送先のランデブポートの返答メッセージの最大サイズが，回送するランデブが成立したランデブポートの返答メッセージの最大サイズより大きい場合にエラーとするのは，次の理由による．タスクAは，返答メッセージを格納するための領域として，最初に呼び出したランデブポートの返答メッセージの最大サイズを確保しておかなければならないとしているが，回送先のランデブポートの返答メッセージの最大サイズがこれより大きいと，タスクAが確保した領域に返答メッセージが収まらなくなる可能性があるためである．

cmsgszが，回送するランデブが成立したランデブポートの返答メッセージの最大サイズよりも大きい場合にエラーとするのは，タスクAがランデブ呼出し待ち状態になる時に，msgとcmsgszで指定される呼出しメッセージを，タスクAが返答メッセージを格納するための確保した領域にコピーするためである．

rpl_rdv　ランデブの終了

【C言語API】

ER ercd = rpl_rdv (RDVNO rdvno, VP msg, UINT rmsgsz) ;

【パラメータ】

RDVNO	rdvno	終了させるランデブ番号
VP	msg	返答メッセージの先頭番地
UINT	rmsgsz	返答メッセージのサイズ（バイト数）

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	パラメータエラー（msg，rmsgszが不正）
E_OBJ	オブジェクト状態エラー（rdvnoが不正）

【機能】

rdvnoで指定されるランデブ番号を付与されたランデブを終了させる．msgで指定される番地からrmsgszで指定されるバイト数のメモリ領域に格納されたメッセージを，返答メッセージとする．

具体的には，rdvnoで指定されたランデブを呼び出したタスクが，指定されたランデブの終了待ち状態である場合には，相手タスクが返答メッセージを格納する領域にmsgとrmsgszで指定される返答メッセージをコピーし，相手タスクを待ち解除する．また，待ち解除された相手タスクに対しては，待ち状態に入ったサービスコールの返値として返答メッセージのサイズ（rmsgsz）を返す．

rdvnoで指定されたランデブを呼び出したタスクが，指定されたランデブの終了待ち状態でない場合には，E_OBJエラーを返す．また，rdvnoに指定された値が，ランデブ番号として解釈できない場合にも，E_OBJエラーを返す．

rdvnoに，他のタスクが受け付けたランデブ番号を指定することも可能である．言い換えると，rpl_rdvを呼び出してランデブを終了させるタスクは，ランデブを受け付けたタスクとは別のタスクであってもよい．

rmsgszが，ランデブが成立したランデブポートの返答メッセージの最大サイズよりも大きい場合には，E_PARエラーを返す．rmsgszに0を指定することは可能である．

【補足説明】

ランデブを呼び出したタスクが，ランデブが成立した後に，タイムアウトや待ち状態の強制解除などによりランデブ終了待ち状態から待ち解除された場合でも，ランデブを受け付けたタスクには通知されない．この場合，ランデブを受け付けたタスクがrpl_rdvを呼び出してランデブを終了させたようとすると，E_OBJエラーとなる．ランデブの相手タスクが，指定するランデブで終了待ち

状態であるかどうかは，ref_rdvを用いて調べることができる．

ランデブが成立した後は，ランデブした双方のタスクともランデブポートから切り離されるが，ランデブポートの返答メッセージの最大サイズは，返答メッセージのサイズ（rmsgsz）が最大サイズ以下であるかを調べるために参照する必要があるため，ランデブに関連付けて記憶しておく必要がある．実装方法の例として，呼出し待ち状態となっているタスクのTCBまたはTCBから参照可能な領域（スタック領域など）に入れておく方法が考えられる．

【μITRON3.0仕様との相違】

ランデブを受け付けたタスクとは別のタスクがランデブを終了できることを明確にした．

サイズ0の返答メッセージも許すことにした．

rdvnoのデータ型をRNOからRDVNOに, rmsgszのデータ型をINTからUINTにそれぞれ変更した．

【仕様決定の理由】

rpl_rdvのパラメータにランデブポートのID番号を指定しないのは, ランデブが成立した後は，ランデブを呼び出したタスクはランデブポートから切り離されるためである．

rdvnoが不正の場合に，E_PARエラーではなくE_OBJエラーとしているのは，rdvnoの不正を静的に検出することはできないためである．

ref_por ランデブポートの状態参照

【C言語API】

ER ercd = ref_por (ID porid, T_RPOR *pk_rpor) ;

【パラメータ】

ID	porid	状態参照対象のランデブポートのID番号
T_RPOR *	pk_rpor	ランデブポート状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rporの内容（T_RPOR型）

ID	ctskid	ランデブポートの呼出し待ち行列の先頭のタスクのID番号
ID	atskid	ランデブポートの受付待ち行列の先頭のタスクのID番号

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（poridが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象ランデブポートが未登録）
E_PAR	パラメータエラー（pk_rporが不正）

【機能】

porid で指定されるランデブポートに関する状態を参照し，pk_rpor で指定されるパケットに返す．

ctskidには，対象ランデブポートの呼出し待ち行列の先頭のタスクのID番号を返す．ランデブ呼出し待ち状態で待っているタスクがない場合には，TSK_NONE（＝0）を返す．

atskidには，対象ランデブポートの受付待ち行列の先頭のタスクのID番号を返す．ランデブ受付待ち状態で待っているタスクがない場合には，TSK_NONE（＝0）を返す．

【μITRON3.0仕様との相違】

参照できる情報から拡張情報を削除した．また，待ちタスクの有無ではなく待ち行列の先頭のタスクのID番号を返すこととした．これに伴って，リターンパラメータの名称とデータ型を変更した．また，パラメータとリターンパラメータの順序を変更した．

ref_rdv　ランデブの状態参照

【C言語API】

```
ER ercd = ref_rdv ( RDVNO rdvno, T_RRDV *pk_rrdv ) ;
```

【パラメータ】

RDVNO	rdvno	状態参照対象のランデブ番号
T_RRDV *	pk_rrdv	ランデブ状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rrdvの内容（T_RRDV型）

ID	wtskid	ランデブ終了待ち状態のタスクのID番号

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_PAR	パラメータエラー（pk_rrdvが不正）

【機能】

rdvnoで指定されるランデブ番号を付与されたランデブに関する状態を参照し，pk_rrdvで指定されるパケットに返す.

rdvnoで指定されたランデブを呼び出したタスクが，指定されたランデブの終了待ち状態である場合には，そのタスクのID番号をwtskidに返す．そのタスクが指定されたランデブの終了待ち状態でない場合や，rdvnoに指定された値がランデブ番号として解釈できない場合，wtskidにはTSK_NONE（＝0）を返す.

【補足説明】

rpl_rdvやfwd_porが, このサービスコールでwtskidにタスクのID番号が返るランデブ番号を指定して呼びされた場合，E_OBJエラーになることはない.

【μITRON3.0仕様との相違】

新設のサービスコールである. ITRON2のランデブ機能には, このサービスコールに対応するrdv_stsが用意されていた.

4.6 メモリプール管理機能

メモリプール管理機能は，ソフトウェアによって動的なメモリ管理を行うための機能である．固定長メモリプール，可変長メモリプールの各機能が含まれる．

【補足説明】

μITRON4.0仕様では，多重論理空間やハードウェアのメモリ管理ユニット（MMU）を操作するための機能は規定されない．

4.6.1 固定長メモリプール

固定長メモリプールは，固定されたサイズのメモリブロックを動的に管理するためのオブジェクトである．固定長メモリプール機能には，固定長メモリプールを生成／削除する機能，固定長メモリプールに対してメモリブロックを獲得／返却する機能，固定長メモリプールの状態を参照する機能が含まれる．固定長メモリプールはID番号で識別されるオブジェクトである．固定長メモリプールのID番号を固定長メモリプールIDと呼ぶ．

固定長メモリプールは，固定長メモリプールとして利用するメモリ領域（これを固定長メモリプール領域，または単にメモリプール領域と呼ぶ）と，メモリブロックの獲得を待つタスクの待ち行列を持つ．固定長メモリプールからメモリブロックを獲得するタスクは，メモリプール領域に空きがなくなった場合，次にメモリブロックが返却されるまで固定長メモリブロックの獲得待ち状態となる．固定長メモリブロックの獲得待ち状態になったタスクは，その固定長メモリプールの待ち行列につながれる．

固定長メモリプール機能に関連して，次のカーネル構成マクロを定義する．

SIZE mpfsz = TSZ_MPF (UINT blkcnt, UINT blksz)

サイズがblkszバイトのメモリブロックをblkcnt個獲得できるのに必要な固定長メモリプール領域のサイズ（バイト数）

固定長メモリプール生成情報および固定長メモリプール状態のパケット形式として，次のデータ型を定義する．

```
typedef struct t_cmpf {
        ATR      mpfatr ;    /* 固定長メモリプール属性 */
        UINT     blkcnt ;    /* 獲得できるメモリブロック数（個
                                数）*/
        UINT     blksz ;     /* メモリブロックのサイズ（バイト
                                数）*/
        VP       mpf ;       /* 固定長メモリプール領域の先頭番
                                地 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_CMPF ;

typedef struct t_rmpf {
```

```
        ID          wtskid ;      /* 固定長メモリプールの待ち行列の先
                                     頭のタスクのID番号 */
        UINT        fblkcnt ;     /* 固定長メモリプールの空きメモリブ
                                     ロック数（個数）*/
        /* 実装独自に他のフィールドを追加してもよい */
} T_RMPF ;
```

固定長メモリプール機能の各サービスコールの機能コードは次の通りである．

TFN_CRE_MPF	–0x45	cre_mpfの機能コード
TFN_ACRE_MPF	–0xc9	acre_mpfの機能コード
TFN_DEL_MPF	–0x46	del_mpfの機能コード
TFN_GET_MPF	–0x49	get_mpfの機能コード
TFN_PGET_MPF	–0x4a	pget_mpfの機能コード
TFN_TGET_MPF	–0x4b	tget_mpfの機能コード
TFN_REL_MPF	–0x47	rel_mpfの機能コード
TFN_REF_MPF	–0x4c	ref_mpfの機能コード

【スタンダードプロファイル】

スタンダードプロファイルでは，固定長メモリプールを動的に生成／削除する機能（cre_mpf，acre_mpf，del_mpf），固定長メモリプールの状態を参照する機能（ref_mpf）を除いて，固定長メモリプール機能をサポートしなければならない．

スタンダードプロファイルでは，TSZ_MPFを定義する必要はない．

【補足説明】

固定長メモリプールの場合，何種類かのサイズのメモリブロックが必要となる場合には，サイズ毎に複数のメモリプールを用意する必要がある．

CRE_MPF 固定長メモリプールの生成（静的API） 【S】
cre_mpf 固定長メモリプールの生成
acre_mpf 固定長メモリプールの生成（ID番号自動割付け）

【静的API】

CRE_MPF (ID mpfid, { ATR mpfatr, UINT blkcnt, UINT blksz, VP mpf }) ;

【C言語API】

ER ercd = cre_mpf (ID mpfid, T_CMPF *pk_cmpf) ;

ER_ID mpfid = acre_mpf (T_CMPF *pk_cmpf) ;

【パラメータ】

ID	mpfid	生成対象の固定長メモリプールの ID 番号（acre_mpf以外）
T_CMPF *	pk_cmpf	固定長メモリプール生成情報を入れたパケットへのポインタ（CRE_MPF ではパケットの内容を直接記述する）

pk_cmpfの内容（T_CMPF型）

ATR	mpfatr	固定長メモリプール属性
UINT	blkcnt	獲得できるメモリブロック数（個数）
UINT	blksz	メモリブロックのサイズ（バイト数）
VP	mpf	固定長メモリプール領域の先頭番地

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cre_mpfの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acre_mpfの場合

ER_ID	mpfid	生成した固定長メモリプールの ID 番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mpfidが不正あるいは使用できない；cre_mpfのみ）
E_NOID	ID 番号不足（割付け可能な固定長メモリプール ID がない；acre_mpfのみ）
E_NOMEM	メモリ不足（メモリプール領域などが確保できない）
E_RSATR	予約属性（mpfatrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_cmpf，blkcnt，blksz，mpfが不正）
E_OBJ	オブジェクト状態エラー（対象固定長メモリプールが登録済み；cre_mpfのみ）

【機能】

mpfidで指定されるID番号を持つ固定長メモリプールを, pk_cmpfで指定される固定長メモリプール生成情報に基づいて生成する．mpfatrは固定長メモリプールの属性, blkcntは固定長メモリプールから獲得できるメモリブロックの個数, blkszは獲得するメモリブロックのサイズ（バイト数）, mpfは固定長メモリプール領域の先頭番地である．

CRE_MPFにおいては，mpfidは自動割付け対応整数値パラメータ，mpfatrはプリプロセッサ定数式パラメータである．

acre_mpfは, 生成する固定長メモリプールのID番号を固定長メモリプールが登録されていないID番号の中から割り付け，割り付けたID番号を返値として返す．

mpfatrには，（TA_TFIFO ‖ TA_TPRI）の指定ができる．固定長メモリプールの待ち行列は，TA_TFIFO（＝0x00）が指定された場合にはFIFO順，TA_TPRI（＝0x01）が指定された場合にはタスクの優先度順となる．

mpfで指定された番地から, blkszバイトのメモリブロックをblkcnt個獲得できるのに必要なサイズのメモリ領域を，メモリプール領域として使用する．TSZ_MPFを用いると，アプリケーションプログラムから，blkszバイトのメモリブロックをblkcnt個獲得できるのに必要なサイズを知ることができる．mpfにNULL（＝0）が指定された場合には，必要なサイズのメモリ領域をカーネルが確保する．

blkcntまたはblkszに0が指定された場合や, 実装定義の最大値よりも大きい値が指定された場合には，E_PARエラーを返す．

【スタンダードプロファイル】

スタンダードプロファイルでは，mpfにNULL以外の値が指定された場合の機能はサポートする必要がない．

【μITRON3.0仕様との相違】

固定長メモリプール生成情報に，メモリプール領域の先頭番地（mpf）を追加し，拡張情報を削除した．パラメータの名称をmpfcntからblkcntに，blfszからblkszにそれぞれ変更し，それらのデータ型をINTからUINTに変更した．

acre_mpfは新設のサービスコールである．

del_mpf 固定長メモリプールの削除

【C言語API】

```
ER ercd = del_mpf ( ID mpfid ) ;
```

【パラメータ】

ID	mpfid	削除対象の固定長メモリプールのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mpfidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象固定長メモリプールが未登録）

【機能】

mpfidで指定される固定長メモリプールを削除する．メモリプール領域をカーネルで確保した場合には，その領域を解放する．

【補足説明】

対象固定長メモリプールからメモリブロックの獲得を待っているタスクがある場合の扱いについては，3.8節を参照すること．

get_mpf	固定長メモリブロックの獲得	【S】【B】
pget_mpf	固定長メモリブロックの獲得（ポーリング）	【S】【B】
tget_mpf	固定長メモリブロックの獲得（タイムアウトあり）	【S】

【C言語API】

ER ercd = get_mpf (ID mpfid, VP *p_blk) ;
ER ercd = pget_mpf (ID mpfid, VP *p_blk) ;
ER ercd = tget_mpf (ID mpfid, VP *p_blk, TMO tmout) ;

【パラメータ】

ID	mpfid	メモリブロック獲得対象の固定長メモリプールのID番号
TMO	tmout	タイムアウト指定（tget_mpfのみ）

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード
VP	blk	獲得したメモリブロックの先頭番地

【エラーコード】

E_ID	不正ID番号（mpfidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象固定長メモリプールが未登録）
E_PAR	パラメータエラー（p_blk，tmoutが不正）
E_RLWAI	待ち状態の強制解除（待ち状態の間に rel_wai を受付；pget_mpf以外）
E_TMOUT	ポーリング失敗またはタイムアウト（get_mpf以外）
E_DLT	待ちオブジェクトの削除（待ち状態の間に対象固定長メモリプールが削除；pget_mpf以外）

【機能】

mpfid で指定される固定長メモリプールから，固定長メモリプール生成時に指定されたサイズのメモリブロックを獲得し，その先頭番地を blk に返す．具体的には，対象固定長メモリプール領域に空きメモリブロックがある場合には，その内のいずれかを選んで獲得された状態とし，その先頭番地を blk に返す．空きメモリブロックがない場合には，自タスクを待ち行列につなぎ，固定長メモリブロックの獲得待ち状態に移行させる．

他のタスクがすでに待ち行列につながっている場合，自タスクを待ち行列につなぐ処理は次のように行う．固定長メモリプール属性に TA_TFIFO（＝ 0x00）が指定されている場合には，自タスクを待ち行列の末尾につなぐ．TA_TPRI（＝ 0x01）が指定されている場合には，自タスクを優先度順で待ち行列につなぐ．同じ優先度のタスクの中では，自タスクを最後につなぐ．

pget_mpf は，get_mpf の処理をポーリングで行うサービスコール，tget_mpf は，

get_mpfにタイムアウトの機能を付け加えたサービスコールである. tmoutには, 正の値のタイムアウト時間に加えて，TMO_POL（＝0）とTMO_FEVR（＝–1）を指定することができる.

【補足説明】

獲得するメモリブロックのサイズは, 固定長メモリプール生成時に指定されたメモリブロックのサイズ以上であれば, それよりも大きくてもよい. またこれらのサービスコールは, 獲得したメモリブロックのクリアは行わないため, その内容は不定となる.

tget_mpfは，tmoutにTMO_POLが指定された場合，E_CTXエラーにならない限りはpget_mpfと全く同じ動作をする．また，tmoutにTMO_FEVRが指定された場合は，get_mpfと全く同じ動作をする.

【μITRON3.0仕様との相違】

サービスコールの名称を，get_blf，pget_blf，tget_blfから，それぞれget_mpf, pget_mpf, tget_mpfに変更した．リターンパラメータの名称をblfからblkに変更した．また，パラメータとリターンパラメータの順序を変更した.

rel_mpf 固定長メモリブロックの返却 【S】【B】

【C言語API】

ER ercd = rel_mpf (ID mpfid, VP blk) ;

【パラメータ】

ID	mpfid	メモリブロック返却対象の固定長メモリプールのID番号
VP	blk	返却するメモリブロックの先頭番地

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mpfidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象固定長メモリプールが未登録）
E_PAR	パラメータエラー（blk が不正，異なるメモリプールへの返却，獲得したメモリブロックの先頭番地以外の返却）

【機能】

mpfidで指定される固定長メモリプールに対して，blkを先頭番地とするメモリブロックを返却する．

対象固定長メモリプールでメモリブロックの獲得を待っているタスクがある場合には，返却したメモリブロックを待ち行列の先頭のタスクに獲得させ，そのタスクを待ち解除する．この時，待ち解除されたタスクに対しては，待ち状態に入ったサービスコールの返値としてE_OKを返し，固定長メモリブロックから獲得したメモリブロックの先頭番地としてblkの値を返す．

メモリブロックを返却する対象の固定長メモリプールは，メモリブロックの獲得を行った固定長メモリプールと同じものでなければならない．メモリブロックを返却する対象の固定長メモリプールが，メモリブロックの獲得を行った固定長メモリプールと異なっている場合には，E_PARエラーを返す．

また，返却するメモリブロックの先頭番地は，get_mpf，pget_mpf，tget_mpfのいずれかのサービスコールが，獲得したメモリブロックの先頭番地として返したもので，まだ返却されていないものでなければならない．blk にそれ以外の番地が指定された場合の振舞いは未定義である．エラーを報告する場合には，E_PARエラーを返す．

【μITRON3.0仕様との相違】

サービスコールの名称を，rel_blfからrel_mpfに変更した．パラメータの名称をblfからblkに変更した．

ref_mpf 固定長メモリプールの状態参照

【C言語API】

ER ercd = ref_mpf (ID mpfid, T_RMPF *pk_rmpf) ;

【パラメータ】

ID	mpfid	状態参照対象の固定長メモリプールのID番号
T_RMPF *	pk_rmpf	固定長メモリプール状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rmpfの内容（T_RMPF型）

ID	wtskid	固定長メモリプールの待ち行列の先頭のタスクのID番号
UINT	fblkcnt	固定長メモリプールの空きメモリブロック数（個数）

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（mpfidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象固定長メモリプールが未登録）
E_PAR	パラメータエラー（pk_rmpfが不正）

【機能】

mpfidで指定される固定長メモリプールに関する状態を参照し，pk_rmpfで指定されるパケットに返す．

wtskidには，対象固定長メモリプールの待ち行列の先頭のタスクのID番号を返す．メモリブロックの獲得を待っているタスクがない場合には，TSK_NONE（＝0）を返す．

fblkcntには，対象固定長メモリプール領域内の空きメモリブロックの個数を返す．

【補足説明】

wtskid ≠ TSK_NONE と fblkcnt ≠ 0が同時に成立することはない．

【μITRON3.0仕様との相違】

参照できる情報から拡張情報を削除した．また，待ちタスクの有無ではなく，待ち行列の先頭のタスクのID番号を返すこととした．これに伴って，リターンパラメータの名称とデータ型を変更した．

リターンパラメータの名称をfrbcntからfblkcntに変更し，そのデータ型をINTからUINTに変更した．また，パラメータとリターンパラメータの順序を変更

した．

4.6.2 可変長メモリプール

可変長メモリプールは，任意のサイズのメモリブロックを動的に管理するためのオブジェクトである．可変長メモリプール機能には，可変長メモリプールを生成／削除する機能，可変長メモリプールに対してメモリブロックを獲得／返却する機能，可変長メモリプールの状態を参照する機能が含まれる．可変長メモリプールはID番号で識別されるオブジェクトである．可変長メモリプールのID番号を可変長メモリプールIDと呼ぶ.

可変長メモリプールは，可変長メモリプールとして利用するメモリ領域（これを可変長メモリプール領域，または単にメモリプール領域と呼ぶ）と，メモリブロックの獲得を待つタスクの待ち行列を持つ．可変長メモリプールからメモリブロックを獲得するタスクは，メモリプール領域の空き領域が足りなくなった場合，十分なサイズのメモリブロックが返却されるまで可変長メモリブロックの獲得待ち状態となる．可変長メモリブロックの獲得待ち状態になったタスクは，その可変長メモリプールの待ち行列につながれる．

可変長メモリプール機能に関連して，次のカーネル構成マクロを定義する．

SIZE mplsz = TSZ_MPL (UINT blkcnt, UINT blksz)

サイズがblkszバイトのメモリブロックをblkcnt個獲得できるのに必要な可変長メモリプール領域のサイズ（目安のバイト数）

このマクロは，あくまでもメモリプール領域のサイズを決める際の目安として使うためのものである．このマクロを使って，異なるサイズのメモリブロックを獲得できるのに必要なサイズを決定することはできない．また，フラグメンテーションが起こった場合などには，指定した数のメモリブロックが獲得できない場合もある．

可変長メモリプール生成情報および可変長メモリプール状態のパケット形式として，次のデータ型を定義する．

```
typedef struct t_cmpl {
        ATR      mplatr ;    /* 可変長メモリプール属性 */
        SIZE     mplsz ;     /* 可変長メモリプール領域のサイズ
                                （バイト数）*/
        VP       mpl ;       /* 可変長メモリプール領域の先頭番
                                地 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_CMPL ;

typedef struct t_rmpl {
        ID       wtskid ;    /* 可変長メモリプールの待ち行列の先
                                頭のタスクのID番号 */
        SIZE     fmplsz ;    /* 可変長メモリプールの空き領域の合
                                計サイズ（バイト数）*/
        UINT     fblksz ;    /* すぐに獲得可能な最大メモリブロッ
                                クサイズ（バイト数）*/
        /* 実装独自に他のフィールドを追加してもよい */
```

```
} T_RMPL ;
```

可変長メモリプール機能の各サービスコールの機能コードは次の通りである．

TFN_CRE_MPL	–0xa1	cre_mplの機能コード
TFN_ACRE_MPL	–0xca	acre_mplの機能コード
TFN_DEL_MPL	–0xa2	del_mplの機能コード
TFN_GET_MPL	–0xa5	get_mplの機能コード
TFN_PGET_MPL	–0xa6	pget_mplの機能コード
TFN_TGET_MPL	–0xa7	tget_mplの機能コード
TFN_REL_MPL	–0xa3	rel_mplの機能コード
TFN_REF_MPL	–0xa8	ref_mplの機能コード

【スタンダードプロファイル】

スタンダードプロファイルでは，可変長メモリプール機能はサポートする必要がない．

【補足説明】

可変長メモリプールでメモリブロックの獲得を待っているタスクは，待ち行列につながれている順序でメモリブロックを獲得する．例えば，ある可変長メモリプールに対して400バイトのメモリブロックを獲得しようとしているタスクAと，100バイトのメモリブロックを獲得しようとしているタスクBが，この順で待ち行列につながれている時に，別のタスクからのメモリブロックの返却により200バイトの連続空きメモリ領域ができたとする．このような場合でも，タスクAがメモリブロックを獲得するまで，タスクBはメモリブロックを獲得できない．ただし，実装独自に，このような場合にタスクBに先にメモリブロックを獲得させる指定を，可変長メモリプール属性に追加することは許される．

【μITRON3.0仕様との相違】

可変長メモリプールでメモリブロックの獲得を待っているタスクが，待ち行列につながれている順序でメモリブロックを獲得するか，メモリブロックを獲得できるものから先に獲得するかが実装依存となっていたのを，前者に標準化した．

CRE_MPL 可変長メモリプールの生成（静的API）
cre_mpl 可変長メモリプールの生成
acre_mpl 可変長メモリプールの生成（ID番号自動割付け）

【静的API】

CRE_MPL (ID mplid, { ATR mplatr, SIZE mplsz, VP mpl }) ;

【C言語API】

ER ercd = cre_mpl (ID mplid, T_CMPL *pk_cmpl) ;

ER_ID mplid = acre_mpl (T_CMPL *pk_cmpl) ;

【パラメータ】

ID	mplid	生成対象の可変長メモリプールのID番号（acre_mpl以外）
T_CMPL *	pk_cmpl	可変長メモリプール生成情報を入れたパケットへのポインタ（CRE_MPLではパケットの内容を直接記述する）

pk_cmplの内容（T_CMPL型）

ATR	mplatr	可変長メモリプール属性
SIZE	mplsz	可変長メモリプール領域のサイズ（バイト数）
VP	mpl	可変長メモリプール領域の先頭番地

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cre_mplの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acre_mplの場合

ER_ID	mplid	生成した可変長メモリプールのID番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mplidが不正あるいは使用できない；cre_mplのみ）
E_NOID	ID番号不足（割付け可能な可変長メモリプールIDがない；acre_mplのみ）
E_NOMEM	メモリ不足（メモリプール領域などが確保できない）
E_RSATR	予約属性（mplatrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_cmpl，mplsz，mplが不正）
E_OBJ	オブジェクト状態エラー（対象可変長メモリプールが登録済み；cre_mplのみ）

【機能】

mplidで指定されるID番号を持つ可変長メモリプールを，pk_cmplで指定される可変長メモリプール生成情報に基づいて生成する．mplatrは可変長メモリプールの属性，mplszは可変長メモリプール領域のサイズ（バイト数），mplは可変長メモリプール領域の先頭番地である．

CRE_MPLにおいては，mplidは自動割付け対応整数値パラメータ，mplatrはプリプロセッサ定数式パラメータである．

acre_mplは，生成する可変長メモリプールのID番号を可変長メモリプールが登録されていないID番号の中から割り付け，割り付けたID番号を返値として返す．

mplatrには，（TA_TFIFO ‖ TA_TPRI）の指定ができる．可変長メモリプールの待ち行列は，TA_TFIFO（＝0x00）が指定された場合にはFIFO順，TA_TPRI（＝0x01）が指定された場合にはタスクの優先度順となる．

mplで指定された番地からmplszバイトのメモリ領域を，メモリプール領域として使用する．メモリプール領域内には，メモリブロックを管理するための情報も置くため，メモリプール領域のすべてをメモリブロックとして獲得できるわけではない．TSZ_MPLを用いると，アプリケーションプログラムから，mplszに指定すべきサイズの目安を知ることができる．mplにNULL（＝0）が指定された場合には，mplszで指定されたサイズのメモリ領域を，カーネルが確保する．

mplszに0が指定された場合や，実装定義の最大値よりも大きい値が指定された場合には，E_PARエラーを返す．

【補足説明】

mplにNULLが指定された場合にカーネルが確保するメモリプール領域のサイズは，mplszに指定されたサイズ以上であれば，それよりも大きくてもよい．

【μITRON3.0仕様との相違】

可変長メモリプール生成情報に，メモリプール領域の先頭番地（mpl）を追加し，拡張情報を削除した．また，mplszのデータ型を，INTからSIZEに変更した．

acre_mplは新設のサービスコールである．

del_mpl 可変長メモリプールの削除

【C言語API】

```
ER ercd = del_mpl ( ID mplid ) ;
```

【パラメータ】

ID	mplid	削除対象の可変長メモリプールのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mplidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象可変長メモリプールが未登録）

【機能】

mplidで指定される可変長メモリプールを削除する．メモリプール領域をカーネルで確保した場合には，その領域を解放する．

【補足説明】

対象可変長メモリプールからメモリブロックの獲得を待っているタスクがある場合の扱いについては，3.8節を参照すること．

get_mpl 可変長メモリブロックの獲得
pget_mpl 可変長メモリブロックの獲得（ポーリング）
tget_mpl 可変長メモリブロックの獲得（タイムアウトあり）

【C言語API】

```
ER ercd = get_mpl ( ID mplid, UINT blksz, VP *p_blk ) ;
ER ercd = pget_mpl ( ID mplid, UINT blksz, VP *p_blk ) ;
ER ercd = tget_mpl ( ID mplid, UINT blksz, VP *p_blk, TMO tmout ) ;
```

【パラメータ】

ID	mplid	メモリブロック獲得対象の可変長メモリプールのID番号
UINT	blksz	獲得するメモリブロックのサイズ（バイト数）
TMO	tmout	タイムアウト指定（tget_mplのみ）

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード
VP	blk	獲得したメモリブロックの先頭番地

【エラーコード】

E_ID	不正ID番号（mplidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象可変長メモリプールが未登録）
E_PAR	パラメータエラー (blksz, p_blk, tmoutが不正)
E_RLWAI	待ち状態の強制解除（待ち状態の間に rel_wai を受付；pget_mpl以外）
E_TMOUT	ポーリング失敗またはタイムアウト（get_mpl以外）
E_DLT	待ちオブジェクトの削除（待ち状態の間に対象可変長メモリプールが削除；pget_mpl以外）

【機能】

mplidで指定される可変長メモリプールから，blkszで指定されるサイズのメモリブロックを獲得し，その先頭番地をblkに返す．

具体的な処理内容は，自タスクより優先してメモリブロックを獲得できるタスクが待っているかどうかによって異なる．対象可変長メモリプールでメモリブロック獲得を待っているタスクがない場合や，可変長メモリプール属性にTA_TPRI（＝0x01）が指定されており，メモリブロック獲得を待っているタスクの優先度がいずれも自タスクの優先度よりも低い場合には，メモリプール領域からblkszバイトのメモリブロックを獲得する．この条件を満たさない場合や，メモリブロックを獲得するのに十分な空きメモリ領域がない場合には，自タスクを待ち行列につなぎ，可変長メモリブロックの獲得待ち状態に移行させる．

他のタスクがすでに待ち行列につながっている場合，自タスクを待ち行列につなぐ処理は次のように行う．可変長メモリプール属性にTA_TFIFO（＝0x00）が指定されている場合には，自タスクを待ち行列の末尾につなぐ．TA_TPRI（＝0x01）が指定されている場合には，自タスクを優先度順で待ち行列につなぐ．同じ優先度のタスクの中では，自タスクを最後につなぐ．

可変長メモリブロックの獲得を待っているタスクが，rel_waiやter_tskにより待ち解除されたり，タイムアウトにより待ち解除された結果，待ち行列の先頭のタスクが変化する時には，新たに先頭になったタスクから順に可能ならメモリブロックを獲得させる処理を行う必要がある．具体的な処理内容は，rel_mplによってメモリブロックが返却された後の処理と同様であるため，rel_mplの機能説明を参照すること．また，可変長メモリブロックの獲得を待っているタスクの優先度が，chg_priやミューテックスの操作によって変更された結果，可変長メモリプールの待ち行列の先頭のタスクが変化する時にも，同様の処理が必要である．

pget_mplは，get_mplの処理をポーリングで行うサービスコール，tget_mplは，get_mplにタイムアウトの機能を付け加えたサービスコールである．tmoutには，正の値のタイムアウト時間に加えて，TMO_POL（＝0）とTMO_FEVR（＝–1）を指定することができる．

blkszに0が指定された場合には，E_PARエラーを返す．また，生成時に指定する可変長メモリプールのメモリ領域のサイズよりも大きい値がblkszに指定された場合，実装依存でE_PARエラーを返すことができる．

【補足説明】

獲得するメモリブロックのサイズは，blkszに指定されたサイズ以上であれば，それよりも大きくてもよい．またこれらのサービスコールは，獲得したメモリブロックのクリアは行わないため，その内容は不定となる．

tget_mplは，tmoutにTMO_POLが指定された場合，E_CTXエラーにならない限りはpget_mplと全く同じ動作をする．また，tmoutにTMO_FEVRが指定された場合は，get_mplと全く同じ動作をする．

【μITRON3.0仕様との相違】

サービスコールの名称を，get_blk，pget_blk，tget_blkから，それぞれget_mpl，pget_mpl，tget_mplに変更した．blkszのデータ型を，INTからUINTに変更した．また，パラメータとリターンパラメータの順序を変更した．

rel_mpl 可変長メモリブロックの返却

【C言語API】

ER ercd = rel_mpl (ID mplid, VP blk) ;

【パラメータ】

ID	mplid	メモリブロック返却対象の可変長メモリプールのID番号
VP	blk	返却するメモリブロックの先頭番地

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mplidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象可変長メモリプールが未登録）
E_PAR	パラメータエラー（blk が不正，異なるメモリプールへの返却，獲得したメモリブロックの先頭番地以外の返却）

【機能】

mplidで指定される可変長メモリプールに対して，blkを先頭番地とするメモリブロックを返却する．

対象可変長メモリプールでメモリブロックの獲得を待っているタスクがある場合には，メモリブロックを返却した結果，待ち行列の先頭のタスクが獲得しようとしているサイズのメモリブロックが獲得できるようになったかを調べる．獲得できる場合には，そのタスクにメモリブロックを獲得させ，そのタスクを待ち解除する．この時，待ち解除されたタスクに対しては，待ち状態に入ったサービスコールの返値としてE_OKを返し，可変長メモリブロックから獲得したメモリブロックの先頭番地として獲得したメモリブロックの先頭番地を返す．さらに，メモリブロックの獲得を待っているタスクが残っている場合には，新たに待ち行列の先頭になったタスクに対して同じ処理を繰り返す．

メモリブロックを返却する対象の可変長メモリプールは，メモリブロックの獲得を行った可変長メモリプールと同じものでなければならない．メモリブロックを返却する対象の可変長メモリプールが，メモリブロックの獲得を行った可変長メモリプールと異なっている場合には，E_PARエラーを返す．

また，返却するメモリブロックの先頭番地は，get_mpl，pget_mpl，tget_mplのいずれかのサービスコールが，獲得したメモリブロックの先頭番地として返したもので，まだ返却されていないものでなければならない．blkにそれ以外の番地が指定された場合の振舞いは未定義である．エラーを報告する場合には，E_PARエラーを返す．

【補足説明】

このサービスコールにより，複数のタスクが待ち解除される場合，可変長メモリプールの待ち行列につながれていた順序で待ち解除する．そのため，実行可能状態に移行したタスクで同じ優先度を持つものの間では，待ち行列の中で前につながれていたタスクの方が高い優先順位を持つことになる．

【μITRON3.0仕様との相違】

サービスコールの名称を，rel_blkからrel_mplに変更した．

ref_mpl 可変長メモリプールの状態参照

【C言語API】

ER ercd = ref_mpl (ID mplid, T_RMPL *pk_rmpl) ;

【パラメータ】

ID	mplid	状態参照対象の可変長メモリプールのID番号
T_RMPL *	pk_rmpl	可変長メモリプール状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rmplの内容（T_RMPL型）

ID	wtskid	可変長メモリプールの待ち行列の先頭のタスクのID番号
SIZE	fmplsz	可変長メモリプールの空き領域の合計サイズ（バイト数）
UINT	fblksz	すぐに獲得可能な最大メモリブロックサイズ（バイト数）

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（mplidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象可変長メモリプールが未登録）
E_PAR	パラメータエラー（pk_rmplが不正）

【機能】

mplidで指定される可変長メモリプールに関する状態を参照し，pk_rmplで指定されるパケットに返す．

wtskidには，対象可変長メモリプールの待ち行列の先頭のタスクのID番号を返す．メモリブロックの獲得を待っているタスクがない場合には，TSK_NONE（＝0）を返す．

fmplszには，対象可変長メモリプールの現在の空き領域の合計サイズ（バイト数）を返す．

fblkszには，対象可変長メモリプールからすぐに獲得できる最大のメモリブロックサイズ（バイト数）を返す．すぐに獲得できる最大のメモリブロックサイズが，UINT型で表せる最大値よりも大きい場合には，UINT型で表せる最大値をfblkszに返す．

【補足説明】

カーネル内部で動的なメモリ管理を用いている場合に，アプリケーションプロ

グラムから，カーネルの管理するメモリの空き領域などを参照するためのAPIとして，このサービスコールを流用することができる．具体的には，ID番号が(–4)の可変長メモリプールを指定してこのサービスコールを呼び出すと，カーネルの管理するメモリに関する情報を参照できることとする．ただしこの場合，参照できる情報の内，wtskidは意味を持たない．さらに，カーネルがメモリを複数の単位に分けて管理している場合には，ID番号が（–3）～（–2）の可変長メモリプールを同様の目的に使うことができる．

【μITRON3.0仕様との相違】

参照できる情報から拡張情報を削除した．また，待ちタスクの有無ではなく，待ち行列の先頭のタスクのID番号を返すこととした．これに伴って，リターンパラメータの名称とデータ型を変更した．

リターンパラメータの名称を，frszからfmplszに，maxszからfblkszにそれぞれ変更し，fmplszのデータ型をINTからSIZEに，fblkszのデータ型をINTからUINTにそれぞれ変更した．また，パラメータとリターンパラメータの順序を変更した．

4.7 時間管理機能

時間管理機能は，時間に依存した処理を行うための機能である．システム時刻管理，周期ハンドラ，アラームハンドラ，オーバランハンドラの各機能が含まれる．周期ハンドラ，アラームハンドラ，オーバランハンドラを総称して，タイムイベントハンドラと呼ぶ．

【補足説明】

タイムイベントハンドラを実行するコンテキストと状態については，次のように整理できる．

- タイムイベントハンドラは，それぞれ独立したコンテキストで実行する（3.5.1節参照）．タイムイベントハンドラを実行するコンテキストは，非タスクコンテキストに分類される（3.5.2節参照）．
- タイムイベントハンドラ（オーバランハンドラを除く）の優先順位は，isig_tim を呼び出した割込みハンドラの優先順位以下で，ディスパッチャの優先順位よりも高い．オーバランハンドラの優先順位は，ディスパッチャの優先順位よりも高い（3.5.3節参照）．
- タイムイベントハンドラの実行開始直後は，CPUロック解除状態になっている．タイムイベントハンドラからリターンする際には，CPUロック解除状態にしなければならない（3.5.4節参照）．
- タイムイベントハンドラの起動と，そこからのリターンによって，ディスパッチ禁止／許可状態は変化しない．タイムイベントハンドラ内でディスパッチ禁止／許可状態を変化させた場合には，タイムイベントハンドラからリターンする前に元の状態に戻さなければならない（3.5.5節を参照）．

【μITRON3.0仕様との相違】

周期起動ハンドラに代えて，周期ハンドラという用語を用いることとした．オーバランハンドラは新たに導入した機能である．また，自タスクの実行を遅延する機能（dly_tsk）をタスク付属同期機能に分類することにした．ret_tmrは廃止した（3.9節参照）．

4.7.1 システム時刻管理

システム時刻管理機能は，システム時刻を操作するための機能である．システム時刻を設定／参照する機能，タイムティックを供給してシステム時刻を更新する機能が含まれる．

システム時刻は，システム初期化時に0に初期化し（3.7節参照），以降，アプリケーションによってタイムティックを供給するサービスコール（isig_tim）が呼び出される度に更新する．isig_tim が呼び出される度にシステム時刻をどれだけ更新するか，言い換えるとアプリケーションが isig_tim を呼び出すべき周期は，実装定義で定める．ただし，システム時刻を更新する機構をカーネル内部に持つことも可能で，その場合には，isig_timをサポートする必要はない．

システム時刻に依存して，タイムアウト処理，dly_tskによる時間経過待ち状態からの解除，周期ハンドラの起動，アラームハンドラの起動の各処理を行う．同じタイムティックで行うべき処理が複数ある場合，それらの実行順序は実装依存とする．

システム時刻管理機能に関連して，次のカーネル構成定数を定義する．

TIC_NUME	タイムティックの周期の分子
TIC_DENO	タイムティックの周期の分母

これらのカーネル構成定数は，システム時刻に依存して行われる処理のおおよその時間精度をアプリケーションが参照するためのもので，TIC_NUME/TIC_DENOがタイムティックの周期を示す（時間単位はシステム時刻と同じ）．システム時刻の更新を一定周期で行わない場合，これらのカーネル構成定数には，イベントの発生時刻の時間精度が同等になるタイムティックの周期を定義する．

システム時刻管理機能の各サービスコールの機能コードは次の通りである．

TFN_SET_TIM	–0x4d	set_timの機能コード
TFN_GET_TIM	–0x4e	get_timの機能コード
TFN_ISIG_TIM	–0x7d	isig_timの機能コード

【スタンダードプロファイル】

スタンダードプロファイルでは，システム時刻管理機能をサポートしなければならない．ただし，システム時刻を更新する機構をカーネル内部に持つ場合には，タイムティックを供給するサービスコール（isig_tim）をサポートする必要はない．

【補足説明】

システムコンフィギュレーションファイルないしはアプリケーションが用意するヘッダファイルでTIC_NUMEとTIC_DENOを定義することで，アプリケーションがisig_timを呼び出す周期をカーネルに知らせる方法もある．

【μITRON3.0仕様との相違】

システムクロックに代えて，システム時刻という用語を用いることとした．タイムティックを供給するサービスコールを新たに導入した．これは，タイマのハードウェアに依存しない形でカーネルを提供できるようにするためである．

システム時刻のビット数に関する推奨値（μITRON3.0仕様では，48ビット）を定めないことにした．また，システム時刻の起点として絶対的な日時（μITRON3.0仕様では，1985年1月1日0時（GMT））を推奨することに代えて，システム初期化時にシステム時刻を0に初期化するものと規定した．

set_tim システム時刻の設定 【S】

【C言語API】

```
ER ercd = set_tim ( SYSTIM *p_systim ) ;
```

【パラメータ】

SYSTIM	systim	システム時刻に設定する時刻

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	パラメータエラー（p_systim，systimが不正）

【機能】

現在のシステム時刻を，systimで指定される時刻に設定する．

【補足説明】

このサービスコールによってシステム時刻を変更した場合にも，相対時間を用いて指定されたイベントの発生する実時刻は変化しない．言い換えると，相対時間を用いて指定されたイベントの発生するシステム時刻は変化する（2.1.9節参照）．

【μITRON3.0仕様との相違】

systimのデータ型をSYSTIMEからSYSTIMに変更した．また，systimを格納するパケットを廃止し，C言語APIの引数の名称をpk_timからp_systimに変更した．

【仕様決定の理由】

systimをポインタによって渡しているのは，SYSTIM型が構造体として定義されている時に，そのままパラメータとして渡すと効率が悪くなる場合を考慮したためである．

get_tim　システム時刻の参照　【S】

【C言語API】

```
ER ercd = get_tim ( SYSTIM *p_systim ) ;
```

【パラメータ】

なし

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード
SYSTIM	systim	現在のシステム時刻

【エラーコード】

E_PAR	パラメータエラー（p_systimが不正）

【機能】

現在のシステム時刻を呼び出し，systimに返す.

【μITRON3.0仕様との相違】

systimのデータ型をSYSTIMEからSYSTIMに変更した. また, systimを格納するパケットを廃止し，C言語APIの引数の名称をpk_timからp_systimに変更した.

isig_tim タイムティックの供給 【S】

【C言語API】

ER ercd = isig_tim ( ) ;

【パラメータ】

なし

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

特記すべきエラーはない

【機能】

システム時刻を更新する．

【スタンダードプロファイル】

システム時刻を更新する機構をカーネル内部に持つ場合には，このサービスコールをサポートする必要はない．

【補足説明】

このサービスコールは，システム時刻に依存して行うべき処理のきっかけとなるものであり，該当する処理がこのサービスコール内で行われるとは限らない．言い換えると，このサービスコールからリターンした時点で，該当する処理が完了しているとは限らない．

【μITRON3.0仕様との相違】

新設のサービスコールである．

4.7.2 周期ハンドラ

周期ハンドラは，一定周期で起動されるタイムイベントハンドラである．周期ハンドラ機能には，周期ハンドラを生成／削除する機能，周期ハンドラの動作を開始／停止する機能，周期ハンドラの状態を参照する機能が含まれる．周期ハンドラはID番号で識別されるオブジェクトである．周期ハンドラのID番号を周期ハンドラIDと呼ぶ.

周期ハンドラの起動周期と起動位相は，周期ハンドラの生成時に，周期ハンドラ毎に設定することができる．カーネルは，周期ハンドラの操作時に，設定された起動周期と起動位相から，周期ハンドラを次に起動すべき時刻を決定する．周期ハンドラの生成時には，周期ハンドラを生成した時刻に起動位相を加えた時刻を，次に起動すべき時刻とする．周期ハンドラを起動すべき時刻になると，その周期ハンドラの拡張情報（exinf）をパラメータとして，周期ハンドラを起動する．またこの時，周期ハンドラの起動すべき時刻に起動周期を加えた時刻を，次に起動すべき時刻とする．また，周期ハンドラの動作を開始する時に，次に起動すべき時刻を決定しなおす場合がある.

周期ハンドラの起動位相は，起動周期以下であることを原則とする．起動位相に，起動周期よりも長い時間が指定された場合の振舞いは，実装依存とする.

周期ハンドラは，動作している状態か動作していない状態かのいずれかの状態をとる．周期ハンドラが動作していない状態の時には，周期ハンドラを起動すべき時刻となっても周期ハンドラを起動せず，次に起動すべき時刻の決定のみを行う．周期ハンドラの動作を開始するサービスコール（sta_cyc）が呼び出されると，周期ハンドラを動作している状態に移行させ，必要なら周期ハンドラを次に起動すべき時刻を決定しなおす．周期ハンドラの動作を停止するサービスコール（stp_cyc）が呼び出されると，周期ハンドラを動作していない状態に移行させる．周期ハンドラを生成した後にどちらの状態になるかは，周期ハンドラ属性によって決めることができる.

周期ハンドラの起動位相は，周期ハンドラを生成するサービスコールが呼び出された時刻（静的APIで生成する場合にはシステム初期化時）を基準に，周期ハンドラを最初に起動する時刻を指定する相対時間と解釈する．周期ハンドラの起動周期は，周期ハンドラを（起動した時刻ではなく）起動すべきであった時刻を基準に，周期ハンドラを次に起動する時刻を指定する相対時間と解釈する．そのため，周期ハンドラが起動される時刻の間隔は，個々には起動周期よりも短くなる場合があるが，長い期間で平均すると起動周期に一致する.

周期ハンドラのC言語による記述形式は次の通りとする.

```
void cychdr ( VP_INT exinf )
{
        周期ハンドラ本体
}
```

周期ハンドラ生成情報および周期ハンドラ状態のパケット形式として，次のデータ型を定義する.

```
typedef struct t_ccyc {
        ATR         cycatr ;      /* 周期ハンドラ属性 */
        VP_INT      exinf ;       /* 周期ハンドラの拡張情報 */
        FP          cychdr ;      /* 周期ハンドラの起動番地 */
        RELTIM      cyctim ;      /* 周期ハンドラの起動周期 */
        RELTIM      cycphs ;      /* 周期ハンドラの起動位相 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_CCYC ;

typedef struct t_rcyc {
        STAT        cycstat ;     /* 周期ハンドラの動作状態 */
        RELTIM      lefttim ;     /* 周期ハンドラを次に起動する時刻ま
                                     での時間 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RCYC ;
```

周期ハンドラ機能の各サービスコールの機能コードは次の通りである.

TFN_CRE_CYC	–0x4f	cre_cycの機能コード
TFN_ACRE_CYC	–0xcb	acre_cycの機能コード
TFN_DEL_CYC	–0x50	del_cycの機能コード
TFN_STA_CYC	–0x51	sta_cycの機能コード
TFN_STP_CYC	–0x52	stp_cycの機能コード
TFN_REF_CYC	–0x53	ref_cycの機能コード

【スタンダードプロファイル】

スタンダードプロファイルでは，周期ハンドラを動的に生成／削除する機能（cre_cyc，acre_cyc，del_cyc），周期ハンドラの状態を参照する機能（ref_cyc）を除いて，周期ハンドラ機能をサポートしなければならない.

スタンダードプロファイルでは，周期ハンドラの起動位相を保存する機能（周期ハンドラ属性のTA_PHS指定）をサポートする必要はない.

【補足説明】

起動位相を保存するとは，周期ハンドラを生成した時刻から周期ハンドラを起動すべき時刻までの時間を起動周期で割った余りが一定になるように，周期ハンドラを起動すべき時刻を決定することをいう．図4-5に，周期ハンドラ属性にTA_STAを指定せずに周期ハンドラを生成した後に，sta_cycによって周期ハンドラの動作を開始した場合の，周期ハンドラの起動の様子を示す．起動位相を保存する場合には，周期ハンドラを次に起動すべき時刻は，常に周期ハンドラを生成した時刻を基準に決定する（図4-5 (a)）．それに対して，起動位相を保存しない場合には，sta_cycが呼び出された時刻を基準として，周期ハンドラを次に起動すべき時刻を決定する（図4-5 (b)）.

周期ハンドラの起動は，システム時刻に依存して行われる処理である．そのため周期ハンドラの起動は，起動すべき時刻以降の最初のタイムティックで行う．周期ハンドラの起動位相は，周期ハンドラを生成するサービスコールが呼び出された時刻からの相対時間であるため，周期ハンドラの初回の起動は，周

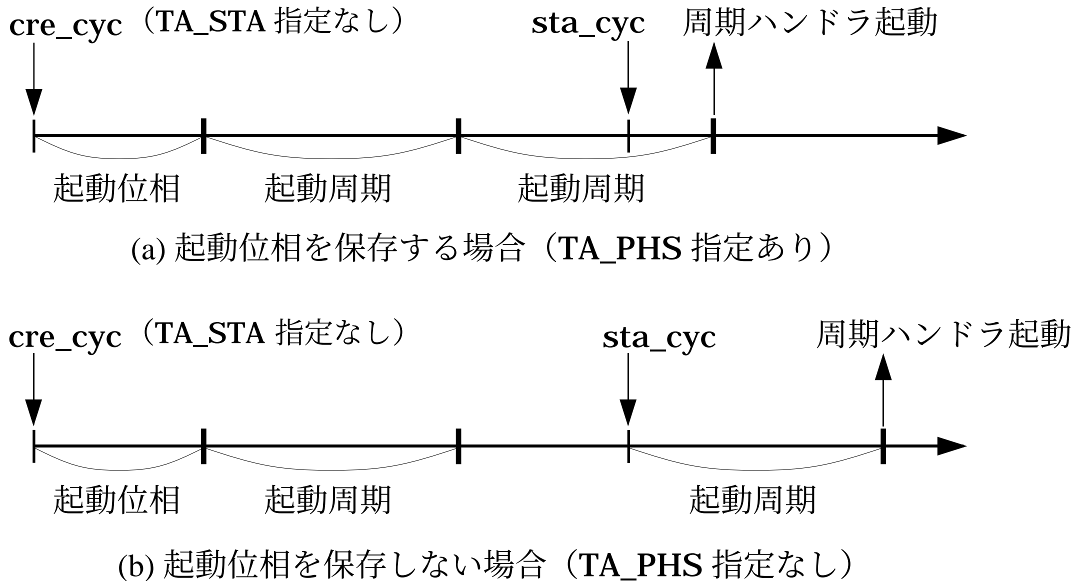


図4-5. 起動位相の保存

期ハンドラを生成するサービスコールが呼び出されてから，指定された起動位相以上の時間が経過した後に行うことを保証しなければならない（周期ハンドラが動作している状態の時．以下同じ）．周期ハンドラの起動周期は，周期ハンドラを起動すべき時刻からの相対時間である．すなわち，周期ハンドラの *n* 回目の起動は，周期ハンドラを生成するサービスコールが呼び出されてから，（起動位相＋起動周期×（*n*–1））以上の時間が経過した後に行うことを保証しなければならない．例えば，タイムティックの周期が10ミリ秒のシステムにおいて，起動位相が15ミリ秒，起動周期が25ミリ秒の周期ハンドラを静的APIで生成すると，周期起動ハンドラが起動されるシステム時刻は，20ミリ秒，40ミリ秒，70ミリ秒，90ミリ秒，120ミリ秒，…のようになる．相対時間でイベントの発生する時刻を指定する場合の扱いについては，2.1.9節を参照すること．

この仕様上は，周期ハンドラが動作していない状態の時にも，次に起動すべき時刻の管理を行うこととしているが，同じ振舞いをするのであれば，実装上はこの管理を省略してもよい．

【μITRON3.0仕様との相違】

周期起動ハンドラに代えて，周期ハンドラという用語を用いることとした．周期ハンドラをID番号で識別するオブジェクトとし，周期ハンドラの定義（def_cyc）を生成（cre_cyc），周期ハンドラ番号を周期ハンドラIDと呼ぶことにした．また，周期ハンドラの削除を行うサービスコール（del_cyc）を新設した．

周期ハンドラの活性制御を行うサービスコール（act_cyc）の機能を，周期ハンドラの動作を開始するサービスコール（sta_cyc）と停止するサービスコール（stp_cyc）に分割した．

CRE_CYC	周期ハンドラの生成（静的API）	【S】
cre_cyc	周期ハンドラの生成	
acre_cyc	周期ハンドラの生成（ID番号自動割付け）	

【静的API】

CRE_CYC (ID cycid, { ATR cycatr, VP_INT exinf, FP cychdr,
RELTIM cyctim, RELTIM cycphs }) ;

【C言語API】

ER ercd = cre_cyc (ID cycid, T_CCYC *pk_ccyc) ;
ER_ID cycid = acre_cyc (T_CCYC *pk_ccyc) ;

【パラメータ】

ID	cycid	生成対象の周期ハンドラのID番号（acre_cyc以外）
T_CCYC *	pk_ccyc	周期ハンドラ生成情報を入れたパケットへのポインタ（CRE_CYCではパケットの内容を直接記述する）

pk_ccycの内容（T_CCYC型）

ATR	cycatr	周期ハンドラ属性
VP_INT	exinf	周期ハンドラの拡張情報
FP	cychdr	周期ハンドラの起動番地
RELTIM	cyctim	周期ハンドラの起動周期
RELTIM	cycphs	周期ハンドラの起動位相

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cre_cycの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acre_cycの場合

ER_ID	cycid	生成した周期ハンドラのID番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（cycidが不正あるいは使用できない；cre_cycのみ）
E_NOID	ID 番号不足（割付け可能な周期ハンドラ ID がない；acre_cycのみ）
E_RSATR	予約属性（cycatrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_ccyc，cychdr，cyctim，cycphsが不正）
E_OBJ	オブジェクト状態エラー（対象周期ハンドラが登録済

み；cre_cycのみ）

【機能】

cycidで指定されるID番号を持つ周期ハンドラを，pk_ccycで指定される周期ハンドラ生成情報に基づいて生成する．cycatrは周期ハンドラの属性，exinfは周期ハンドラを起動する時にパラメータとして渡す拡張情報，cychdrは周期ハンドラの起動番地，cyctimは周期ハンドラを起動する周期，cycphsは周期ハンドラを起動する位相である．

CRE_CYCにおいては，cycidは自動割付け対応整数値パラメータ，cycatrはプリプロセッサ定数式パラメータである．

acre_cycは，生成する周期ハンドラのID番号を周期ハンドラが登録されていないID番号の中から割り付け，割り付けたID番号を返値として返す．

cycatrには，（(TA_HLNG ‖ TA_ASM）｜［TA_STA］｜［TA_PHS]）の指定ができる．TA_HLNG（＝0x00）が指定された場合には高級言語用のインタフェースで，TA_ASM（＝0x01）が指定された場合にはアセンブリ言語用のインタフェースで周期ハンドラを起動する．TA_STA（＝0x02）が指定された場合には，周期ハンドラを生成した後に，周期ハンドラを動作している状態とする．そうでない場合には，周期ハンドラを動作していない状態とする．TA_PHS（＝0x04）が指定された場合には，周期ハンドラの動作を開始する時に，周期ハンドラの起動位相を保存して，次に起動すべき時刻を決定する．周期ハンドラの動作を開始する時の振舞いに関しては，sta_cycの機能説明を参照すること．

周期ハンドラを最初に起動すべき時刻は，これらのサービスコールが呼び出された時刻（静的APIの場合にはシステム初期化の時刻）に，指定された起動位相を加えた時刻とする．

cyctimに0が指定された場合には，E_PARエラーを返す．cycphsにcyctimよりも大きい値が指定された場合の振舞いは実装依存である．エラーを報告する場合には，E_PARエラーを返す．

【スタンダードプロファイル】

スタンダードプロファイルでは，cycatrにTA_PHSが指定された場合の機能とTA_ASMが指定された場合の機能はサポートする必要がない．

【補足説明】

cycatrにTA_STAとTA_PHSのいずれも指定されない場合には，周期ハンドラの起動位相（cycphs）は意味を持たない．

【μITRON3.0仕様との相違】

周期ハンドラの定義（def_cyc）に代えて，生成（cre_cyc）と呼ぶことにした．周期ハンドラの起動位相を指定する機能を追加し，周期ハンドラ生成情報に周期ハンドラの起動位相（cycphs）を追加した．周期ハンドラを生成した後の動作状態を指定する方法を変更した．

周期ハンドラ生成情報を入れたパケット中でのcycatrとexinfの順序を交換し

た．また，exinfのデータ型をVPからVP_INTに，cyctimのデータ型をCYCTIMEからRELTIMに変更した．

acre_cycは新設のサービスコールである．

del_cyc　　周期ハンドラの削除

【C言語API】

```
ER ercd = del_cyc ( ID cycid ) ;
```

【パラメータ】

ID	cycid	削除対象の周期ハンドラのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（cycidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象周期ハンドラが未登録）

【機能】

cycidで指定される周期ハンドラを削除する．

【補足説明】

対象周期ハンドラが動作している場合には，周期ハンドラを動作していない状態に移行させる．

【μITRON3.0仕様との相違】

新設のサービスコールである．μITRON3.0仕様では，周期ハンドラを定義するサービスコール（def_cyc）が，周期ハンドラの登録を解除する機能を持っている．

sta_cyc 周期ハンドラの動作開始 【S】【B】

【C言語API】

ER ercd = sta_cyc (ID cycid) ;

【パラメータ】

ID	cycid	動作開始対象の周期ハンドラのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（cycidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象周期ハンドラが未登録）

【機能】

cycidで指定される周期ハンドラを，動作している状態に移行させる．

周期ハンドラ属性にTA_PHS（＝0x04）が指定されていない場合には，このサービスコールが呼び出された時刻に，周期ハンドラの起動周期を加えた時刻を，周期ハンドラを次に起動すべき時刻とする．

対象周期ハンドラに，周期ハンドラ属性にTA_PHSが指定されていない動作している状態の周期ハンドラが指定された場合には，周期ハンドラを次に起動すべき時刻の再設定のみを行う．対象周期ハンドラに，TA_PHSが指定されている動作している状態の周期ハンドラが指定された場合には，何もしない．

【μITRON3.0仕様との相違】

周期ハンドラの活性制御を行うサービスコール（act_cyc）の機能を，周期ハンドラの動作を開始するサービスコール（sta_cyc）と停止するサービスコール（stp_cyc）に分割した．μITRON3.0仕様のact_cycでカウントを初期化する指定（TCY_INI）は，周期ハンドラ属性でTA_PHSを指定しない場合に対応する．

stp_cyc　周期ハンドラの動作停止　【S】【B】

【C言語API】

ER ercd = stp_cyc (ID cycid) ;

【パラメータ】

ID	cycid	動作停止対象の周期ハンドラのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（cycidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象周期ハンドラが未登録）

【機能】

cycidで指定される周期ハンドラを動作していない状態に移行させる．対象周期ハンドラに，動作していない状態の周期ハンドラが指定された場合には，何もしない．

【μITRON3.0仕様との相違】

周期ハンドラの活性制御を行うサービスコール（act_cyc）の機能を，周期ハンドラの動作を開始するサービスコール（sta_cyc）と停止するサービスコール（stp_cyc）に分割した．

ref_cyc　　周期ハンドラの状態参照

【C言語API】

ER ercd = ref_cyc (ID cycid, T_RCYC *pk_rcyc) ;

【パラメータ】

ID	cycid	状態参照対象の周期ハンドラのID番号
T_RCYC *	pk_rcyc	周期ハンドラ状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rcycの内容（T_RCYC型）

STAT	cycstat	周期ハンドラの動作状態
RELTIM	lefttim	周期ハンドラを次に起動する時刻までの時間

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（cycidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象周期ハンドラが未登録）
E_PAR	パラメータエラー（pk_rcycが不正）

【機能】

cycidで指定される周期ハンドラに関する状態を参照し，pk_rcycで指定されるパケットに返す．

cycstatには，対象周期ハンドラが動作している状態か動作していない状態かによって，次のいずれかの値を返す．

TCYC_STP	0x00	周期ハンドラが動作していない
TCYC_STA	0x01	周期ハンドラが動作している

lefttimには，対象周期ハンドラが動作している状態の場合に，対象周期ハンドラを次に起動する時刻までの相対時間を返す．具体的には，対象周期ハンドラを次に起動する時刻から現在時刻を減じた値を返す．ただし，lefttimに返す値は，対象周期ハンドラが起動されるまでの時間以下でなければならない．そのため，次のタイムティックで周期ハンドラが起動される場合には，lefttimに0を返す．対象周期ハンドラが動作していない状態の場合にlefttimに返す値は実装依存である．

【μITRON3.0仕様との相違】

参照できる情報から拡張情報を削除した．周期ハンドラの動作状態を参照する方法を変更した．lefttimのデータ型をCYCTIMEからRELTIMに変更した．また，パラメータとリターンパラメータの順序を変更した．

4.7.3 アラームハンドラ

アラームハンドラは，指定した時刻に起動されるタイムイベントハンドラである．アラームハンドラ機能には，アラームハンドラを生成／削除する機能，アラームハンドラの動作を開始／停止する機能，アラームハンドラの状態を参照する機能が含まれる．アラームハンドラはID番号で識別されるオブジェクトである．アラームハンドラのID番号をアラームハンドラIDと呼ぶ．

アラームハンドラを起動する時刻（これをアラームハンドラの起動時刻と呼ぶ）は，アラームハンドラ毎に設定することができる．アラームハンドラの起動時刻になると，そのアラームハンドラの拡張情報（exinf）をパラメータとして，アラームハンドラを起動する．

アラームハンドラの生成直後には，アラームハンドラの起動時刻は設定されておらず，アラームハンドラの動作は停止している．アラームハンドラの動作を開始するサービスコール（sta_alm）が呼び出されると，アラームハンドラの起動時刻を，サービスコールが呼び出された時刻から指定された相対時間後に設定する．アラームハンドラの動作を停止するサービスコール（stp_alm）が呼び出されると，アラームハンドラの起動時刻の設定を解除する．また，アラームハンドラを起動する時にも，アラームハンドラの起動時刻の設定を解除し，アラームハンドラの動作を停止する．

アラームハンドラのC言語による記述形式は次の通りとする．

```
void almhdr ( VP_INT exinf )
{
        アラームハンドラ本体
}
```

アラームハンドラ生成情報およびアラームハンドラ状態のパケット形式として，次のデータ型を定義する．

```
typedef struct t_calm {
        ATR         almatr ;     /* アラームハンドラ属性 */
        VP_INT      exinf ;      /* アラームハンドラの拡張情報 */
        FP          almhdr ;     /* アラームハンドラの起動番地 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_CALM ;

typedef struct t_ralm {
        STAT        almstat ;    /* アラームハンドラの動作状態 */
        RELTIM      lefttim ;    /* アラームハンドラの起動時刻までの
                                     時間 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RALM ;
```

アラームハンドラ機能の各サービスコールの機能コードは次の通りである．

TFN_CRE_ALM	–0xa9	cre_almの機能コード
TFN_ACRE_ALM	–0xcc	acre_almの機能コード
TFN_DEL_ALM	–0xaa	del_almの機能コード

TFN_STA_ALM	–0xab	sta_almの機能コード
TFN_STP_ALM	–0xac	stp_almの機能コード
TFN_REF_ALM	–0xad	ref_almの機能コード

【スタンダードプロファイル】

スタンダードプロファイルでは，アラームハンドラ機能はサポートする必要がない．

【補足説明】

アラームハンドラの起動は，システム時刻に依存して行われる処理である．そのため，アラームハンドラの起動は，起動時刻以降の最初のタイムティックで行う．また，アラームハンドラの起動は，アラームハンドラの動作を開始するサービスコールが呼び出されてから，指定された以上の時間が経過した後に行うことを保証しなければならない（2.1.9節参照）．

アラームハンドラの起動時刻の設定解除は，アラームハンドラを起動する時（すなわちアラームハンドラの実行前）に行われる．そのため，非タスクコンテキストからアラームハンドラの動作を開始するサービスコールを実装独自に呼出し可能とした場合には，起動されたアラームハンドラ内で，それ自身の起動時刻を再設定して，アラームハンドラの動作を開始することができる．

【μITRON3.0仕様との相違】

アラームハンドラをID番号で識別するオブジェクトとし，アラームハンドラの定義（def_alm）を生成（cre_alm），アラームハンドラ番号をアラームハンドラIDと呼ぶことにした．また，アラームハンドラの削除を行うサービスコール（del_alm）を新設した．

アラームハンドラを静的に生成する場合を考慮し，アラームハンドラの起動時刻の設定をアラームハンドラの登録時に行わず，新設のアラームハンドラの動作を開始するサービスコール（sta_alm）で設定することとした．また，アラームハンドラの動作を停止するサービスコール（stp_alm）を新設した．

アラームハンドラの起動時刻を絶対時刻で指定する機能は廃止した．

CRE_ALM アラームハンドラの生成（静的API）
cre_alm アラームハンドラの生成
acre_alm アラームハンドラの生成（ID番号自動割付け）

【静的API】

CRE_ALM (ID almid, { ATR almatr, VP_INT exinf, FP almhdr }) ;

【C言語API】

ER ercd = cre_alm (ID almid, T_CALM *pk_calm) ;
ER_ID almid = acre_alm (T_CALM *pk_calm) ;

【パラメータ】

ID	almid	生成対象のアラームハンドラのID番号（acre_alm以外）
T_CALM *	pk_calm	アラームハンドラ生成情報を入れたパケットへのポインタ（CRE_ALMではパケットの内容を直接記述する）

pk_calmの内容（T_CALM型）

ATR	almatr	アラームハンドラ属性
VP_INT	exinf	アラームハンドラの拡張情報
FP	almhdr	アラームハンドラの起動番地

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cre_almの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acre_almの場合

ER_ID	almid	生成したアラームハンドラのID番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（almidが不正あるいは使用できない；cre_almのみ）
E_NOID	ID番号不足（割付け可能なアラームハンドラIDがない；acre_almのみ）
E_RSATR	予約属性（almatrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_calm，almhdrが不正）
E_OBJ	オブジェクト状態エラー（対象アラームハンドラが登録済み；cre_almのみ）

【機能】

almidで指定されるID番号を持つアラームハンドラを, pk_calmで指定されるア

ラームハンドラ生成情報に基づいて生成する．almatrはアラームハンドラの属性，exinfはアラームハンドラを起動する時にパラメータとして渡す拡張情報，almhdrはアラームハンドラの起動番地である．

CRE_ALMにおいては，almidは自動割付け対応整数値パラメータ，almatrはプリプロセッサ定数式パラメータである．

acre_almは，生成するアラームハンドラのID番号をアラームハンドラが登録されていないID番号の中から割り付け，割り付けたID番号を返値として返す．

アラームハンドラの生成直後には，アラームハンドラの起動時刻は設定されておらず，アラームハンドラの動作は停止している．

almatrには，（TA_HLNG ‖ TA_ASM）の指定ができる．TA_HLNG（＝0x00）が指定された場合には高級言語用のインタフェースで，TA_ASM（＝0x01）が指定された場合にはアセンブリ言語用のインタフェースでアラームハンドラを起動する．

【μITRON3.0仕様との相違】

アラームハンドラの定義（def_alm）に代えて，生成（cre_alm）と呼ぶことにした．アラームハンドラを静的に生成する場合を考慮し，アラームハンドラの起動時刻の設定をアラームハンドラの登録時には行わないこととした．

アラームハンドラ生成情報を入れたパケット中でのalmatrとexinfの順序を交換した．また，exinfのデータ型を，VPからVP_INTに変更した．

acre_almは新設のサービスコールである．

del_alm アラームハンドラの削除

【C言語API】

ER ercd = del_alm (ID almid) ;

【パラメータ】

ID	almid	削除対象のアラームハンドラのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（almidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象アラームハンドラが未登録）

【機能】

almidで指定されるアラームハンドラを削除する．

【補足説明】

対象アラームハンドラが動作している場合には，対象アラームハンドラの起動時刻の設定を解除し，アラームハンドラの動作を停止する．

【μITRON3.0仕様との相違】

新設のサービスコールである．μITRON3.0仕様では，アラームハンドラを定義するサービスコール（def_alm）が，アラームハンドラの登録を解除する機能を持っている．

sta_alm　　アラームハンドラの動作開始

【C言語API】

ER ercd = sta_alm (ID almid, RELTIM almtim) ;

【パラメータ】

ID	almid	動作開始対象のアラームハンドラのID番号
RELTIM	almtim	アラームハンドラの起動時刻（相対時間）

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（almidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象アラームハンドラが未登録）
E_PAR	パラメータエラー（almtimが不正）

【機能】

almidで指定されるアラームハンドラの起動時刻を，サービスコールが呼び出された時刻からalmtimで指定された相対時間後に設定し，アラームハンドラの動作を開始する．

対象アラームハンドラに，すでに動作しているアラームハンドラが指定された場合には，以前の起動時刻の設定を解除し，新しい起動時刻を設定する．

almtimは，サービスコールが呼び出された時刻を基準に，アラームハンドラの起動時刻を指定する相対時間と解釈する．

【μITRON3.0仕様との相違】

新設のサービスコールである．μITRON3.0仕様では，アラームハンドラを定義するサービスコール（def_alm）が，アラームハンドラの起動時刻を設定する機能を持っている．

stp_alm アラームハンドラの動作停止

【C言語API】

```
ER ercd = stp_alm ( ID almid ) ;
```

【パラメータ】

ID	almid	動作停止対象のアラームハンドラのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（almidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象アラームハンドラが未登録）

【機能】

almidで指定されるアラームハンドラの起動時刻の設定を解除し，アラームハンドラの動作を停止する．対象アラームハンドラに，動作していないアラームハンドラが指定された場合には，何もしない．

【μITRON3.0仕様との相違】

新設のサービスコールである．μITRON3.0仕様では，アラームハンドラの登録を解除する以外の方法で，アラームハンドラの動作を停止することはできない．

ref_alm アラームハンドラの状態参照

【C言語API】

ER ercd = ref_alm (ID almid, T_RALM *pk_ralm) ;

【パラメータ】

ID	almid	状態参照対象のアラームハンドラのID番号
T_RALM *	pk_ralm	アラームハンドラ状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_ralmの内容（T_RALM型）

STAT	almstat	アラームハンドラの動作状態
RELTIM	lefttim	アラームハンドラの起動時刻までの時間

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（almidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象アラームハンドラが未登録）
E_PAR	パラメータエラー（pk_ralmが不正）

【機能】

almidで指定されるアラームハンドラに関する状態を参照し，pk_ralmで指定されるパケットに返す．

almstatには，対象アラームハンドラが動作しているかどうか（言い換えると，対象アラームハンドラに起動時刻が設定されているかどうか）によって，次のいずれかの値を返す．

TALM_STP	0x00	アラームハンドラが動作していない
TALM_STA	0x01	アラームハンドラが動作している

lefttimには，対象アラームハンドラが動作している場合に，対象アラームハンドラの起動時刻までの相対時間を返す．具体的には，対象アラームハンドラの起動時刻から現在時刻を減じた値を返す．ただし，lefttimに返す値は，対象アラームハンドラが起動されるまでの時間以下でなければならない．そのため，次のタイムティックでアラームハンドラが起動される場合には，lefttimに0を返す．対象アラームハンドラが動作していない場合にlefttimに返す値は実装依存である．

【μITRON3.0仕様との相違】

参照できる情報に，アラームハンドラの動作状態（almstat）を追加し，拡張情報を削除した．lefttimのデータ型をALMTIMEからRELTIMに変更した．また，パラメータとリターンパラメータの順序を変更した．

4.7.4 オーバランハンドラ

オーバランハンドラは，タスクが設定された時間を越えてプロセッサを使用した場合に起動されるタイムイベントハンドラである．オーバランハンドラ機能には，オーバランハンドラを定義する機能，オーバランハンドラの動作を開始／停止する機能，オーバランハンドラの状態を参照する機能が含まれる．

オーバランハンドラの起動条件として設定される時間（これをタスクの上限プロセッサ時間と呼ぶ）は，タスク毎に設定することができる．カーネルは，上限プロセッサ時間が設定されたタスクに対して，上限プロセッサ時間が設定されて以降にそのタスクが使用した累積プロセッサ時間（これをタスクの使用プロセッサ時間と呼ぶ）を管理し，タスクの使用プロセッサ時間が上限プロセッサ時間を越えると，オーバランハンドラを起動する．システムに定義できるオーバランハンドラは1つだけであるため，オーバランハンドラには，起動の原因となったタスクのID番号（tskid）と拡張情報（exinf）をパラメータとして渡す．

タスクの生成直後には，タスクの上限プロセッサ時間は設定されていない．オーバランハンドラの動作を開始するサービスコール（sta_ovr）が呼び出されると，指定されたタスクに対して指定された上限プロセッサ時間を設定し，使用プロセッサ時間を0にクリアする．オーバランハンドラの動作を停止するサービスコール（stp_ovr）が呼び出されると，指定されたタスクの上限プロセッサ時間の設定を解除する．タスクの上限プロセッサ時間の設定は，そのタスクが原因となってオーバランハンドラを起動する時や，そのタスクが終了する時にも解除する．

タスクが使用したプロセッサ時間には，少なくとも，そのタスク（タスク例外処理ルーチンを含む）とそのタスクから呼び出したサービスコールの実行時間が含まれる．逆に，他のタスク（タスク例外処理ルーチンを含む）と他のタスクから呼び出したサービスコールの実行時間は含まれない．タスクディスパッチにかかる時間やタスクの実行中に起動された割込みハンドラの実行時間など，その他の処理の実行時間が含まれるかどうかは実装依存とする．また，使用したプロセッサ時間の計測精度も実装依存である．ただし，オーバランハンドラが起動されるのは，タスクの使用プロセッサ時間が，設定した上限プロセッサ時間を越えた場合に限る．

オーバランハンドラ機能では，次のデータ型を用いる．

OVRTIM　　プロセッサ時間（符号無し整数，時間単位は実装定義）

オーバランハンドラのC言語による記述形式は次の通りとする．

```
void ovrhdr ( ID tskid, VP_INT exinf )
{
        オーバランハンドラ本体
}
```

オーバランハンドラ生成情報およびオーバランハンドラ状態のパケット形式として，次のデータ型を定義する．

```
typedef struct t_dovr {
        ATR         ovratr ;      /* オーバランハンドラ属性 */
        FP          ovrhdr ;      /* オーバランハンドラの起動番地 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_DOVR ;

typedef struct t_rovr {
        STAT        ovrstat ;     /* オーバランハンドラの動作状態 */
        OVRTIM      leftotm ;     /* 残りのプロセッサ時間 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_ROVR ;
```

オーバランハンドラ機能の各サービスコールの機能コードは次の通りである．

TFN_DEF_OVR	–0xb1	def_ovrの機能コード
TFN_STA_OVR	–0xb2	sta_ovrの機能コード
TFN_STP_OVR	–0xb3	stp_ovrの機能コード
TFN_REF_OVR	–0xb4	ref_ovrの機能コード

【スタンダードプロファイル】

スタンダードプロファイルでは，オーバランハンドラ機能はサポートする必要がない．

【補足説明】

オーバランハンドラの起動は，システム時刻に依存して行われる処理ではない．そのため，オーバランハンドラは，タイムティックに同期して起動されるとは限らない（タイムティックに同期してオーバランハンドラを起動する実装も禁止されてはいない）．

タスクの上限プロセッサ時間の設定解除は，オーバランハンドラを起動する時（すなわちオーバランハンドラの実行前）に行われる．そのため，非タスクコンテキストからオーバランハンドラの動作を開始するサービスコールを実装独自に呼出し可能とした場合には，オーバランハンドラ起動の原因となったタスクに対して，オーバランハンドラ内で上限プロセッサ時間を再設定することができる．

タスクの使用プロセッサ時間が上限プロセッサ時間を越えた場合の処理を，そのタスクのコンテキストで処理させたい場合には，オーバランハンドラからタスク例外処理を要求し，タスク例外処理ルーチンで必要な処理を行う方法で実現することができる．

【μITRON3.0仕様との相違】

オーバランハンドラ機能は，μITRON4.0仕様において新たに導入した機能である．

DEF_OVR オーバランハンドラの定義（静的API）
def_ovr オーバランハンドラの定義

【静的API】

DEF_OVR ({ ATR ovratr, FP ovrhdr }) ;

【C言語API】

ER ercd = def_ovr (T_DOVR *pk_dovr) ;

【パラメータ】

T_DOVR *	pk_dovr	オーバランハンドラ定義情報を入れたパケットへのポインタ（DEF_OVR ではパケットの内容を直接記述する）

pk_dovrの内容（T_DOVR型）

ATR	ovratr	オーバランハンドラ属性
FP	ovrhdr	オーバランハンドラの起動番地

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_RSATR	予約属性（ovratrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_dovr，ovrhdrが不正）

【機能】

pk_dovr で指定されるオーバランハンドラ定義情報に基づいて，オーバランハンドラを定義する．ovratr はオーバランハンドラの属性，ovrhdr はオーバランハンドラの起動番地である．

DEF_OVRにおいては，ovratrはプリプロセッサ定数式パラメータである．

pk_dovrにNULL（＝0）が指定されると，すでに定義されているオーバランハンドラの定義を解除し，オーバランハンドラが定義されていない状態にする．この時，すべてのタスクの上限プロセッサ時間の設定を解除する．また，すでにオーバランハンドラが定義されている状態で，再度オーバランハンドラが定義された場合には，以前の定義を解除し，新しい定義に置き換える．この時は，タスクの上限プロセッサ時間の設定解除は行わない．

ovratrには，（TA_HLNG ‖ TA_ASM）の指定ができる．TA_HLNG（＝0x00）が指定された場合には高級言語用のインタフェースで，TA_ASM（＝0x01）が指定された場合にはアセンブリ言語用のインタフェースでオーバランハンドラを起動する．

【仕様決定の理由】

オーバランハンドラの定義を解除する時に，タスクの上限プロセッサ時間の設

定を解除するのは，オーバランハンドラが定義されていない時には，タスクの上限プロセッサ時間が設定されていない状態を保つためである（オーバランハンドラが定義されていない時には，タスクに対して上限プロセッサ時間を設定することができないため，この状態が保たれる）．

sta_ovr　　オーバランハンドラの動作開始

【C言語API】

ER ercd = sta_ovr (ID tskid, OVRTIM ovrtim) ;

【パラメータ】

ID	tskid	動作開始対象のタスクのID番号
OVRTIM	ovrtim	設定するタスクの上限プロセッサ時間

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（tskidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象タスクが未登録）
E_PAR	パラメータエラー（ovrtimが不正）
E_OBJ	オブジェクト状態エラー（オーバランハンドラが定義されていない）

【機能】

tskidで指定されるタスクに対して，オーバランハンドラの動作を開始する．具体的には，対象タスクの上限プロセッサ時間を，ovrtimで指定される時間に設定する．また，対象タスクの使用プロセッサ時間を0にクリアする．

対象タスクに，すでに上限プロセッサ時間が設定されているタスクが指定された場合には，以前の上限プロセッサ時間の設定を解除し，新しい上限プロセッサ時間を設定する．この時にも，使用プロセッサ時間を0にクリアする．

tskidにTSK_SELF（＝0）が指定されると，自タスクを対象タスクとする．

stp_ovr　　オーバランハンドラの動作停止

【C言語API】

ER ercd = stp_ovr (ID tskid) ;

【パラメータ】

ID	tskid	動作停止対象のタスクのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（tskidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象タスクが未登録）
E_OBJ	オブジェクト状態エラー（オーバランハンドラが定義されていない）

【機能】

tskidで指定されるタスクに対して，オーバランハンドラの動作を停止する．具体的には，対象タスクの上限プロセッサ時間の設定を解除する．対象タスクに，上限プロセッサ時間が設定されていないタスクが指定された場合には，何もしない．

tskidにTSK_SELF（＝0）が指定されると，自タスクを対象タスクとする．

ref_ovr　　オーバランハンドラの状態参照

【C言語API】

```
ER ercd = ref_ovr ( ID tskid, T_ROVR *pk_rovr ) ;
```

【パラメータ】

ID	tskid	状態参照対象のタスクのID番号
T_ROVR *	pk_rovr	オーバランハンドラ状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rovrの内容（T_ROVR型）

STAT	ovrstat	オーバランハンドラの動作状態
OVRTIM	leftotm	残りのプロセッサ時間

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（tskidが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象タスクが未登録）
E_PAR	パラメータエラー（pk_rovrが不正）
E_OBJ	オブジェクト状態エラー（オーバランハンドラが定義されていない）

【機能】

tskidで指定されるタスクの，オーバランハンドラに関する状態を参照し，pk_rovrで指定されるパケットに返す．

ovrstatには，対象タスクに対してオーバランハンドラが動作しているかどうかを返す．具体的には，対象タスクに上限プロセッサ時間が設定されているかどうかによって，次のいずれかの値を返す．

TOVR_STP	0x00	上限プロセッサ時間が設定されていない
TOVR_STA	0x01	上限プロセッサ時間が設定されている

leftotmには，対象タスクに対して上限プロセッサ時間が設定されている場合に，対象タスクを原因としてオーバランハンドラが起動されるまでの残りプロセッサ時間を返す．具体的には，対象タスクの上限プロセッサ時間から使用プロセッサ時間を減じた値を返す．ただし，leftotmに返す値は，対象タスクを原因としてオーバランハンドラが起動されるまでに対象タスクが使用できるプロセッサ時間以下でなければならない．そのため，オーバランハンドラが起動される直前には，leftotmには0が返される可能性がある．対象タスクに対して上限プロセッサ時間が設定されていない場合にleftotmに返す値は実装依存である．

tskidにTSK_SELF（＝0）が指定されると，自タスクを対象タスクとする．

4.8 システム状態管理機能

システム状態管理機能は，システムの状態を変更／参照するための機能である．タスクの優先順位を回転する機能，実行状態のタスクIDを参照する機能，CPUロック状態へ移行／解除する機能，タスクディスパッチを禁止／解除する機能，コンテキストやシステム状態を参照する機能が含まれる．

システム状態のパケット形式として，次のデータ型を定義する．

```
typedef struct t_rsys {
        /* 実装独自のフィールドが入る */
} T_RSYS ;
```

システム状態管理機能の各サービスコールの機能コードは次の通りである．

TFN_ROT_RDQ	–0x55	rot_rdqの機能コード
TFN_IROT_RDQ	–0x79	irot_rdqの機能コード
TFN_GET_TID	–0x56	get_tidの機能コード
TFN_IGET_TID	–0x7a	iget_tidの機能コード
TFN_LOC_CPU	–0x59	loc_cpuの機能コード
TFN_ILOC_CPU	–0x7b	iloc_cpuの機能コード
TFN_UNL_CPU	–0x5a	unl_cpuの機能コード
TFN_IUNL_CPU	–0x7c	iunl_cpuの機能コード
TFN_DIS_DSP	–0x5b	dis_dspの機能コード
TFN_ENA_DSP	–0x5c	ena_dspの機能コード
TFN_SNS_CTX	–0x5d	sns_ctxの機能コード
TFN_SNS_LOC	–0x5e	sns_locの機能コード
TFN_SNS_DSP	–0x5f	sns_dspの機能コード
TFN_SNS_DPN	–0x60	sns_dpnの機能コード
TFN_REF_SYS	–0x61	ref_sysの機能コード

【スタンダードプロファイル】

スタンダードプロファイルでは，システムの状態を参照する機能（ref_sys）を除いて，システム状態管理機能をサポートしなければならない．

【μITRON3.0仕様との相違】

システム状態管理機能という分類を，新たに設けることにした．

rot_rdq タスクの優先順位の回転 【S】【B】
irot_rdq 【S】【B】

【C言語API】

ER ercd = rot_rdq (PRI tskpri) ;
ER ercd = irot_rdq (PRI tskpri) ;

【パラメータ】

PRI	tskpri	優先順位を回転する対象の優先度

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	パラメータエラー（tskpriが不正）

【機能】

tskpri で指定される優先度のタスクの優先順位を回転する．すなわち，対象優先度を持った実行できる状態のタスクの中で，最も高い優先順位を持つタスクを，同じ優先度を持つタスクの中で最低の優先順位とする．

tskpriにTPRI_SELF（＝0）が指定されると，自タスクのベース優先度を対象優先度とする．ただし，非タスクコンテキストからの呼出しでこの指定が行われた場合には，E_PARエラーを返す．

【補足説明】

このサービスコールを周期的に呼び出すことにより，ラウンドロビン方式を実現することができる．対象優先度を持った実行できる状態のタスクがない場合や，一つしかない場合には，何もしない（エラーとはしない）．

ディスパッチ許可状態で，対象優先度に自タスクの現在優先度を指定してこのサービスコールが呼び出されると，自タスクの実行順位は同じ優先度を持つタスクの中で最低となる．そのため，このサービスコールを用いて，実行権の放棄を行うことができる．ディスパッチ禁止状態では，同じ優先度を持つタスクの中で最高の優先順位を持ったタスクが実行されているとは限らないため，この方法で自タスクの実行順位が同じ優先度を持つタスクの中で最低となるとは限らない．自タスクの現在優先度がベース優先度に一致している場合（ミューテックス機能を使わない場合には，この条件は常に成り立つ）には，tskpriにTPRI_SELFを指定しても同じ結果となる．

【μITRON3.0仕様との相違】

非タスクコンテキストから呼び出された場合に，実行状態のタスクの優先度を対象優先度とする指定は規定しないこととした．これに伴って，TPRI_RUNをTPRI_SELFに名称変更した．また，ミューテックスの導入に伴って，TPRI_SELFは自タスクのベース優先度を指定するものとした．

get_tid 実行状態のタスクIDの参照 【S】【B】
iget_tid 【S】

【C言語API】

```
ER ercd = get_tid ( ID *p_tskid ) ;
ER ercd = iget_tid ( ID *p_tskid ) ;
```

【パラメータ】

なし

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード
ID	tskid	実行状態のタスクのID番号

【エラーコード】

E_PAR	パラメータエラー（p_tskidが不正）

【機能】

実行状態のタスク（タスクコンテキストから呼び出された場合は，自タスクに一致する）のID番号を参照し，tskidに返す．非タスクコンテキストから呼び出された場合で，実行状態のタスクがない時には，tskidにTSK_NONE（＝0）を返す．

【補足説明】

アプリケーションが用意したタスクがどれも実行できる状態にない場合に，カーネルが用意したタスク（一般的にはアイドルタスクと呼ばれる）を実行するような実装では，アイドルタスクが実行状態の時にこのサービスコールが呼び出された場合，アイドルタスクのID番号ではなく，TSK_NONEを返す．

【µITRON3.0仕様との相違】

自タスクのID番号を返すサービスコールから，実行状態のタスクのID番号を返すものに変更した．具体的には，非タスクコンテキストから呼び出した場合の振舞いが変更になった．

【仕様決定の理由】

tskidをサービスコールの返値としなかったのは，タスクのID番号が負の値になる場合に対応するためである．

loc_cpu　　CPUロック状態への移行　　【S】【B】
iloc_cpu　　【S】

【C言語API】

```
ER ercd = loc_cpu ( ) ;
ER ercd = iloc_cpu ( ) ;
```

【パラメータ】

なし

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

特記すべきエラーはない

【機能】

CPUロック状態に移行する．CPUロック状態で呼び出された場合には何もしない．

【補足説明】

loc_cpu（またはiloc_cpu）が複数回呼び出されても，unl_cpu（またはiunl_cpu）が一回呼び出されるとCPUロック解除状態となる．したがって，loc_cpu（またはiloc_cpu）とunl_cpu（またはiunl_cpu）の対をネストさせたい場合には，次のような方法で対処することが必要である．

```
{
        BOOL cpu_locked = sns_loc ( ) ;

        if ( !cpu_locked )
                loc_cpu ( ) ;
        CPUロック状態で行うべき処理
        if ( !cpu_locked )
                unl_cpu ( ) ;
}
```

【μITRON3.0仕様との相違】

CPUロック状態の意味を変更した（3.5.4節参照）．また，非タスクコンテキストからも呼び出せるものとした．

unl_cpu CPUロック状態の解除 【S】【B】
iunl_cpu 【S】

【C言語API】

ER ercd = unl_cpu ( ) ;
ER ercd = iunl_cpu ( ) ;

【パラメータ】

なし

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

特記すべきエラーはない

【機能】

CPUロック解除状態に移行する．CPUロック解除状態で呼び出された場合には何もしない．

【μITRON3.0仕様との相違】

CPUロック状態の意味を変更した（3.5.4節参照）．これにより，このサービスコールを呼び出しても，ディスパッチ許可状態になるとは限らないこととなった．また，非タスクコンテキストからも呼び出せるものとした．

dis_dsp　　ディスパッチの禁止　　【S】【B】

【C言語API】

ER ercd = dis_dsp ( ) ;

【パラメータ】

なし

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

特記すべきエラーはない

【機能】

ディスパッチ禁止状態に移行する．ディスパッチ禁止状態で呼び出された場合には何もしない．

【補足説明】

dis_dspが複数回呼び出されても，ena_dspが一回呼び出されるとディスパッチ許可状態となる．したがって，dis_dspとena_dspの対をネストさせたい場合には，次のような方法で対処することが必要である．

```
{
	BOOL dispatch_disabled = sns_dsp ( ) ;

	if ( !dispatch_disabled )
		dis_dsp ( ) ;
	ディスパッチ禁止状態で行うべき処理
	if ( !dispatch_disabled )
		ena_dsp ( ) ;
}
```

【μITRON3.0仕様との相違】

ディスパッチ禁止状態の意味を変更した（3.5.5節参照）．

ena_dsp　　ディスパッチの許可　　【S】【B】

【C言語API】

ER ercd = ena_dsp ( ) ;

【パラメータ】

なし

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

特記すべきエラーはない

【機能】

ディスパッチ許可状態に移行する．ディスパッチ許可状態で呼び出された場合には何もしない．

【μITRON3.0仕様との相違】

ディスパッチ禁止状態の意味を変更した（3.5.5節参照）．

sns_ctx コンテキストの参照 【S】

【C言語API】

```
BOOL state = sns_ctx ( ) ;
```

【パラメータ】

なし

【リターンパラメータ】

BOOL	state	コンテキスト

【機能】

非タスクコンテキストから呼び出された場合にTRUE，タスクコンテキストから呼び出された場合にFALSEを返す.

【μITRON3.0仕様との相違】

新設のサービスコールである.

sns_loc　CPUロック状態の参照　【S】

【C言語API】

```
BOOL state = sns_loc ( ) ;
```

【パラメータ】

なし

【リターンパラメータ】

BOOL	state	CPUロック状態

【機能】

システムが，CPUロック状態の場合にTRUE，CPUロック解除状態の場合にFALSEを返す．

【μITRON3.0仕様との相違】

新設のサービスコールである．

sns_dsp ディスパッチ禁止状態の参照 【S】

【C言語API】

```
BOOL state = sns_dsp ( ) ;
```

【パラメータ】

なし

【リターンパラメータ】

BOOL	state	ディスパッチ禁止状態

【機能】

システムが，ディスパッチ禁止状態の場合にTRUE，ディスパッチ許可状態の場合にFALSEを返す．

【μITRON3.0仕様との相違】

新設のサービスコールである．

sns_dpn　ディスパッチ保留状態の参照　【S】

【C言語API】

BOOL state = sns_dpn ( ) ;

【パラメータ】

なし

【リターンパラメータ】

BOOL　state　ディスパッチ保留状態

【機能】

システムが，ディスパッチ保留状態の場合にTRUE，そうでない場合にFALSEを返す．すなわち，ディスパッチャよりも優先順位の高い処理が実行されているか，CPUロック状態であるか，ディスパッチ禁止状態であるかのいずれかの場合に，TRUEを返す．

【補足説明】

このサービスコールがFALSEを返す状態では，自タスクを広義の待ち状態にする可能性のあるサービスコールを呼び出すことができる．

【μITRON3.0仕様との相違】

新設のサービスコールである．

ref_sys　　システムの状態参照

【C言語API】

ER ercd = ref_sys (T_RSYS *pk_rsys) ;

【パラメータ】

T_RSYS *　pk_rsys　システム状態を返すパケットへのポインタ

【リターンパラメータ】

ER　ercd　正常終了（E_OK）またはエラーコード

pk_rsysの内容（T_RSYS型）

（実装独自の情報が入る）

【エラーコード】

E_PAR　パラメータエラー（pk_rsysが不正）

【機能】

システムの状態を参照し，pk_rsysで指定されるパケットに返す．具体的にどのような情報が参照できるかは，実装定義である．

【補足説明】

このサービスコールで参照できるようにする情報の候補としては，システム状態を参照する他のサービスコール（get_tid，sns_ctx，sns_loc，sns_dsp，sns_dpn）で参照できる情報，実行状態のタスクの優先度，割込みの禁止／許可状態，割込みマスク，プロセッサの実行モード，その他ターゲットプロセッサのアーキテクチャに依存した情報などが挙げられる．

【μITRON3.0仕様との相違】

μITRON3.0仕様で参照できるシステム状態（sysstat）は，他のサービスコール（sns_ctx，sns_loc，sns_dsp）で参照できるため，μITRON4.0仕様では規定しないこととした．

4.9 割込み管理機能

割込み管理機能は，外部割込みによって起動される割込みハンドラおよび割込みサービスルーチンを管理するための機能である．割込みハンドラを定義する機能，割込みサービスルーチンを生成／削除する機能，割込みサービスルーチンの状態を参照する機能，割込みを禁止／許可する機能，割込みマスクを変更／参照する機能が含まれる．割込みサービスルーチンはID番号で識別されるオブジェクトである．割込みサービスルーチンのID番号を割込みサービスルーチンIDと呼ぶ．

割込み管理機能では，次のデータ型を用いる．

INHNO	割込みハンドラ番号
INTNO	割込み番号
I*XXXX*	割込みマスク

割込みマスクのデータ型の一部の*XXXX*は，実装定義で，ターゲットプロセッサのアーキテクチャに応じた適切な文字列に定める．

割込みハンドラの記述形式は，実装定義である．

割込みサービスルーチンを起動する際には，その割込みサービスルーチンの拡張情報（exinf）をパラメータとして渡す．割込みサービスルーチンのC言語による記述形式は次の通りとする．

```
void isr ( VP_INT exinf )
{
        割込みサービスルーチン本体
}
```

割込みハンドラ定義情報，割込みサービスルーチン定義情報，および割込みサービスルーチン状態のパケット形式として，次のデータ型を定義する．

```
typedef struct t_dinh {
        ATR         inhatr ;      /* 割込みハンドラ属性 */
        FP          inthdr ;      /* 割込みハンドラの起動番地 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_DINH ;

typedef struct t_cisr {
        ATR         isratr ;      /* 割込みサービスルーチン属性 */
        VP_INT      exinf ;       /* 割込みサービスルーチンの拡張情
                                     報 */
        INTNO       intno ;       /* 割込みサービスルーチンを付加する
                                     割込み番号 */
        FP          isr ;         /* 割込みサービスルーチンの起動番
                                     地 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_CISR ;

typedef struct t_risr {
        /* 実装独自のフィールドが入る */
```

```
} T_RISR ;
```

割込み管理機能の各サービスコールの機能コードは次の通りである．

TFN_DEF_INH	–0x65	def_inhの機能コード
TFN_CRE_ISR	–0x66	cre_isrの機能コード
TFN_ACRE_ISR	–0xcd	acre_isrの機能コード
TFN_DEL_ISR	–0x67	del_isrの機能コード
TFN_REF_ISR	–0x68	ref_isrの機能コード
TFN_DIS_INT	–0x69	dis_intの機能コード
TFN_ENA_INT	–0x6a	ena_intの機能コード
TFN_CHG_I*XX*	–0x6b	chg_i*xx*の機能コード
TFN_GET_I*XX*	–0x6c	get_i*xx*の機能コード

【スタンダードプロファイル】

スタンダードプロファイルでは，割込みハンドラを定義する静的 API（DEF_INH）をサポートしなければならない．ただし，割込みサービスルーチンを追加する静的API（ATT_ISR）をサポートした場合には，DEF_INH をサポートする必要はない．

【補足説明】

割込みハンドラを実行するコンテキストと状態については，次のように整理できる．

- 割込みハンドラは，それぞれ独立したコンテキストで実行する（3.5.1 節参照）．割込みハンドラを実行するコンテキストは，非タスクコンテキストに分類される（3.5.2節参照）．
- 割込みハンドラの優先順位は，ディスパッチャの優先順位よりも高い（3.5.3節参照）．
- 割込みハンドラの実行開始直後に，CPU ロック状態か CPU ロック解除状態のいずれになっているかは実装依存であるが，割込みハンドラ内でCPUロック解除状態にする方法を明示しなければならない．また，CPUロック解除状態にした後に，割込みハンドラから正しくリターンするための方法も明示しなければならない（3.5.4節参照）．
- 割込みハンドラの起動と，そこからのリターンによって，ディスパッチ禁止／許可状態は変化しない．割込みハンドラ内でディスパッチ禁止／許可状態を変化させた場合には，割込みハンドラからリターンする前に元の状態に戻さなければならない（3.5.5節参照）．

また，割込みサービスルーチンを実行するコンテキストと状態については，次のように整理できる．

- 割込みサービスルーチンは，それぞれ独立したコンテキストで実行する（3.5.1節参照）．割込みサービスルーチンを実行するコンテキストは，非タスクコンテキストに分類される（3.5.2節参照）．
- 割込みサービスルーチンの優先順位は，ディスパッチャの優先順位よりも高

い（3.5.3節参照）.

- 割込みサービスルーチンの実行開始直後は, CPUロック解除状態になっている．割込みサービスルーチンからリターンする際には，CPUロック解除状態にしなければならない（3.5.4節参照）.
- 割込みサービスルーチンの起動と，そこからのリターンによって，ディスパッチ禁止／許可状態は変化しない．割込みサービスルーチン内でディスパッチ禁止／許可状態を変化させた場合には, 割込みサービスルーチンからリターンする前に元の状態に戻さなければならない（3.5.5節参照）.

【μITRON3.0仕様との相違】

CPUロック状態へ移行する機能（loc_cpu）とそれを解除する機能（unl_cpu）をシステム状態管理機能に分類することにした. ret_intとret_wupは廃止した（3.9節参照）.

割込みマスクを入れるパラメータおよびリターンパラメータのデータ型を, UINTから専用のデータ型として新設した*IXXXX*に変更した.

DEF_INH 割込みハンドラの定義（静的API） 【S】
def_inh 割込みハンドラの定義

【静的API】

DEF_INH (INHNO inhno, { ATR inhatr, FP inthdr }) ;

【C言語API】

ER ercd = def_inh (INHNO inhno, T_DINH *pk_dinh) ;

【パラメータ】

INHNO	inhno	定義対象の割込みハンドラ番号
T_DINH *	pk_dinh	割込みハンドラ定義情報を入れたパケットへのポインタ（DEF_INHではパケットの内容を直接記述する）

pk_dinhの内容（T_DINH型）

ATR	inhatr	割込みハンドラ属性
FP	inthdr	割込みハンドラの起動番地

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_RSATR	予約属性（inhatrが不正あるいは使用できない）
E_PAR	パラメータエラー（inhno，pk_dinh，inthdrが不正）

【機能】

inhnoで指定される割込みハンドラ番号に対して，pk_dinhで指定される割込みハンドラ定義情報に基づいて，割込みハンドラを定義する．inhatrは割込みハンドラの属性，inthdrは割込みハンドラの起動番地である．

DEF_INHにおいては，inhnoは自動割付け非対応整数値パラメータ，inhatrはプリプロセッサ定数式パラメータである．

inhnoの具体的な意味は実装定義であるが，一般的な実装ではプロセッサの割込みベクトル番号に対応する．割込みベクトルを持たないプロセッサにおいては，指定することができる割込みハンドラ番号が一つのみとなる場合もある．

pk_dinhにNULL（＝0）が指定されると，すでに定義されている割込みハンドラの定義を解除する．また，すでに割込みハンドラが定義されている割込みハンドラ番号に対して，再度割込みハンドラを定義した場合には，以前の定義を解除し，新しい定義に置き換える．

inhatrに指定できる値とその意味は実装定義である．

【スタンダードプロファイル】

ATT_ISRをサポートした場合には，DEF_INHをサポートする必要はない．

【μITRON3.0仕様との相違】

割込みハンドラを示す略称を，intからinhに変更した．これにより，サービスコールの名称をdef_intからdef_inhにするなどの変更を行った．inhatrに指定できる属性とその意味を実装定義とした．

ATT_ISR 割込みサービスルーチンの追加（静的API）
cre_isr 割込みサービスルーチンの生成
acre_isr 割込みサービスルーチンの生成（ID番号自動割付け）

【静的API】

ATT_ISR ({ ATR isratr, VP_INT exinf, INTNO intno, FP isr }) ;

【C言語API】

ER ercd = cre_isr (ID isrid, T_CISR *pk_cisr) ;
ER_ID isrid = acre_isr (T_CISR *pk_cisr) ;

【パラメータ】

ID	isrid	生成対象の割込みサービスルーチンの ID 番号（cre_isrのみ）
T_CISR *	pk_cisr	割込みサービスルーチン生成情報を入れたパケットへのポインタ（ATT_ISRではパケットの内容を直接記述する）

pk_cisrの内容（T_CISR型）

ATR	isratr	割込みサービスルーチン属性
VP_INT	exinf	割込みサービスルーチンの拡張情報
INTNO	intno	割込みサービスルーチンを付加する割込み番号
FP	isr	割込みサービスルーチンの起動番地

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cre_isrの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acre_isrの場合

ER_ID	isrid	生成した割込みサービスルーチンのID番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（isridが不正あるいは使用できない；cre_isrのみ）
E_NOID	ID 番号不足（割付け可能な割込みサービスルーチン ID がない；acre_isrのみ）
E_RSATR	予約属性（isratrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_cisr，intno，isrが不正）
E_OBJ	オブジェクト状態エラー（対象割込みサービスルーチンが登録済み；cre_isrのみ）

【機能】

isridで指定されるID番号を持つ割込みサービスルーチンを，pk_cisrで指定される割込みサービスルーチン生成情報に基づいて生成する．isratrは割込みサービスルーチンの属性，exinfは割込みサービスルーチンを起動する時にパラメータとして渡す拡張情報，intnoは割込みサービスルーチンを起動する割込みを指定する割込み番号，isrは割込みサービスルーチンの起動番地である．

ATT_ISRは，isridを指定せずに割込みサービスルーチンを追加する．追加した割込みサービスルーチンはID番号を持たない．ATT_ISRにおいては，isratrはプリプロセッサ定数式パラメータ，intnoは自動割付け非対応整数値パラメータである．

acre_isrは，生成する割込みサービスルーチンのID番号を割込みサービスルーチンが登録されていないID番号の中から割り付け，割り付けたID番号を返値として返す．

isratrには，（TA_HLNG ‖ TA_ASM）の指定ができる．TA_HLNG（＝0x00）が指定された場合には高級言語用のインタフェースで，TA_ASM（＝0x01）が指定された場合にはアセンブリ言語用のインタフェースで割込みサービスルーチンを起動する．

【スタンダードプロファイル】

スタンダードプロファイルでは，ATT_ISRをサポートすればDEF_INHをサポートする必要はないが，その際には，isratrにTA_ASMが指定された場合の機能はサポートする必要がない．

【補足説明】

同じ割込み番号を指定して，複数の割込みサービスルーチンを登録することができる．同じ割込み番号に対して登録された複数の割込みサービスルーチンの起動方法については，3.3.2節を参照すること．

【μITRON3.0仕様との相違】

新設のサービスコールである．

del_isr 割込みサービスルーチンの削除

【C言語API】

ER ercd = del_isr (ID isrid) ;

【パラメータ】

ID	isrid	削除対象の割込みサービスルーチンのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（isridが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象割込みサービスルーチンが未登録）

【機能】

isridで指定される割込みサービスルーチンを削除する.

【補足説明】

ATT_ISRで追加した割込みサービスルーチンは，ID番号を持たないために，削除することができない.

【μITRON3.0仕様との相違】

新設のサービスコールである.

ref_isr　　割込みサービスルーチンの状態参照

【C言語API】

ER ercd = ref_isr (ID isrid, T_RISR *pk_risr) ;

【パラメータ】

ID	isrid	状態参照対象の割込みサービスルーチンの ID番号
T_RISR *	pk_risr	割込みサービスルーチン状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_risrの内容（T_RISR型）

（実装独自の情報が入る）

【エラーコード】

E_ID	不正ID番号（isridが不正あるいは使用できない）
E_NOEXS	オブジェクト未生成（対象割込みサービスルーチンが未登録）
E_PAR	パラメータエラー（pk_risrが不正）

【機能】

isridで指定される割込みサービスルーチンの状態を参照し，pk_risrで指定されるパケットに返す．具体的にどのような情報が参照できるかは，実装定義である．

【μITRON3.0仕様との相違】

新設のサービスコールである．

dis_int 割込みの禁止

【C言語API】

ER ercd = dis_int (INTNO intno) ;

【パラメータ】

INTNO	intno	割込みを禁止する割込み番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	パラメータエラー（intnoが不正）

【機能】

intnoで指定される割込みを禁止する．intnoの具体的な意味は実装定義であるが，一般的な実装ではIRCへの割込み要求入力ラインに対応する．

【補足説明】

このサービスコールは，IRCを操作することで実現することを想定している．このサービスコールを呼び出したことで，CPUロック状態やディスパッチ禁止状態へ移行することはない．したがって，このサービスコールを使ってすべての割込みを禁止しても，ディスパッチは起こる．また，他のタスクへのディスパッチ後も，割込みを禁止した状態は保たれる．

【μITRON3.0仕様との相違】

IRCを操作して実現することを想定し，intnoの意味をμITRON3.0仕様よりも強く規定した．また，intnoのデータ型をUINTからINTNOに変更した．

ena_int　割込みの許可

【C言語API】

ER ercd = ena_int (INTNO intno) ;

【パラメータ】

INTNO	intno	割込みを許可する割込み番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	パラメータエラー（intnoが不正）

【機能】

intnoで指定される割込みを許可する．intnoの具体的な意味は実装定義であるが，一般的な実装ではIRCへの割込み要求入力ラインに対応する．

【補足説明】

このサービスコールは，IRCを操作することで実現することを想定している．このサービスコールを呼び出したことで，CPUロック解除状態やディスパッチ許可状態へ移行することはない．したがって，このサービスコールを使ってある割込みを許可しても，その割込みが受け付けられる状態になるとは限らない．

【μITRON3.0仕様との相違】

IRCを操作して実現することを想定し，intnoの意味をμITRON3.0仕様よりも強く規定した．また，intnoのデータ型をUINTからINTNOに変更した．

chg_i*xx* 割込みマスクの変更

【C言語API】

ER ercd = chg_i*xx* (I*XXXX* i*xxxx*) ;

【パラメータ】

I*XXXX*	i*xxxx*	変更後の割込みマスク

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	パラメータエラー（i*xxxx*が不正）

【機能】

プロセッサの割込みマスク（割込みレベル，割込み優先度などと呼ばれる場合もある）を，i*xxxx*で指定される値に変更する．

サービスコールの名称およびパラメータの名称の一部の*xx*および*xxxx*は，実装定義で，ターゲットプロセッサのアーキテクチャに応じた適切な文字列に定める．

i*xxxx*の値によっては，このサービスコールを呼び出したことで，CPUロック状態とCPUロック解除状態の間，ディスパッチ禁止状態とディスパッチ許可状態の間の移行を行うことがある．どのような値でどのような移行を行うかは，実装定義である．

【補足説明】

CPUロック／ロック解除状態をプロセッサの割込みマスクを使って管理している実装では，プロセッサの割込みマスクを変更すると，CPUロック状態とCPUロック解除状態の間の移行を行うことがある．ディスパッチ禁止／許可状態についても同様である．これらの状態をプロセッサの割込みマスクと変数を併用して管理している実装では，このサービスコールによってプロセッサの割込みマスクを変更すると，変数の値もそれと整合させる必要がある．

【μITRON3.0仕様との相違】

i*xxxx*のデータ型をUINTからI*XXXX*に変更した．

get_i*xx* 割込みマスクの参照

【C言語API】

ER ercd = get_i*xx* (I*XXXX* *p_i*xxxx*) ;

【パラメータ】

なし

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード
I*XXXX*	i*xxxx*	現在の割込みマスク

【エラーコード】

E_PAR	パラメータエラー（p_i*xxxx*が不正）

【機能】

プロセッサの割込みマスク（割込みレベル，割込み優先度などと呼ばれる場合もある）を参照しi*xxxx*に返す．

サービスコールの名称およびパラメータの名称の一部の*xx*および*xxxx*は，実装定義で，ターゲットプロセッサのアーキテクチャに応じた適切な文字列に定める．

【μITRON3.0仕様との相違】

サービスコールの名称を，ref_i*xx*からget_i*xx*に変更した．i*xxxx*のデータ型をUINTからI*XXXX*に変更した．

4.10 サービスコール管理機能

サービスコール管理機能は，拡張サービスコールの定義と呼出しを行うための機能である．

拡張サービスコールは，システム全体を1つのモジュールにリンクしない場合に他のモジュールを呼び出すための機能である．拡張サービスコールが呼び出されると，アプリケーションによって定義された拡張サービスコールルーチンを起動する．

拡張サービスコールルーチンのC言語による記述形式は次の通りとする．

```
ER_UINT svcrtn ( VP_INT par1, VP_INT par2, ...)
{
        拡張サービスコールルーチン本体
}
```

ここで，拡張サービスコールルーチンが受け取るパラメータ（par1，par2，...）は，必要な数だけ記述すればよい．実装定義で，受け取れるパラメータの数に上限が設けられている場合がある．ただし，少なくとも1つはパラメータを受け取ることができる．

拡張サービスコール定義情報のパケット形式として，次のデータ型を定義する．

```
typedef struct t_dsvc {
        ATR         svcatr ;      /* 拡張サービスコール属性 */
        FP          svcrtn ;      /* 拡張サービスコールルーチンの起動
                                     番地 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_DSVC ;
```

サービスコール管理機能のサービスコールの機能コードは次の通りである．なお，cal_svcは機能コードを持たない．

TFN_DEF_SVC –0x6d def_svcの機能コード

【スタンダードプロファイル】

スタンダードプロファイルでは，サービスコール管理機能はサポートする必要がない．

【補足説明】

拡張サービスコールルーチンを実行するコンテキストと状態については，次のように整理できる．

- 拡張サービスコールルーチンは，拡張サービスコール毎および拡張サービスコールが呼び出されたコンテキスト毎に，それぞれ独立したコンテキストで実行する（3.5.1節参照）．拡張サービスコールルーチンを実行するコンテキストは，呼び出したコンテキストがタスクコンテキストであればタスクコンテキスト，非タスクコンテキストであれば非タスクコンテキストに分類される（3.5.2節参照）．

- 拡張サービスコールルーチンの優先順位は，拡張サービスコールを呼び出した処理の優先順位よりも高く，拡張サービスコールを呼び出した処理よりも高い優先順位を持つ他のいずれの処理の優先順位よりも低い（3.5.3節参照）．
- 拡張サービスコールルーチンの起動と，そこからのリターンによって，CPUロック／ロック解除状態とディスパッチ禁止／許可状態は変化しない（3.5.4節と3.5.5節を参照）．
- 実装定義で，タスク例外処理を禁止して実行する場合がある（4.3節参照）．

【μITRON3.0仕様との相違】

サービスコール管理機能という分類を，新たに設けることにした．

拡張SVCに代えて拡張サービスコール，拡張SVCハンドラに代えて拡張サービスコールルーチンという用語を用いることにした．また，拡張サービスコールを実行するコンテキストと状態について，μITRON3.0仕様よりは強く規定した．

DEF_SVC　拡張サービスコールの定義（静的API）
def_svc　拡張サービスコールの定義

【静的API】

DEF_SVC (FN fncd, { ATR svcatr, FP svcrtn }) ;

【C言語API】

ER ercd = def_svc (FN fncd, T_DSVC *pk_dsvc) ;

【パラメータ】

FN	fncd	定義対象の機能コード
T_DSVC *	pk_dsvc	拡張サービスコール定義情報を入れたパケットへのポインタ（DEF_SVCではパケットの内容を直接記述する）

pk_dsvcの内容（T_DSVC型）

ATR	svcatr	拡張サービスコール属性
FP	svcrtn	拡張サービスコールルーチンの起動番地

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_RSATR	予約属性（svcatrが不正あるいは使用できない）
E_PAR	パラメータエラー（fncd，pk_dsvc，svcatrが不正）

【機能】

fncdで指定される機能コードに対して，pk_dsvcで指定される拡張サービスコール定義情報に基づいて，拡張サービスコールを定義する．svcatrは拡張サービスコールの属性，svcrtnは拡張サービスコールルーチンの起動番地である．

DEF_SVCにおいては，fncdは自動割付け非対応整数値パラメータ，svcatrはプリプロセッサ定数式パラメータである．

この静的APIおよびサービスコールでは，正の値のfncdに対して拡張サービスコールを定義することができる．fncdに負の値が指定された場合には，E_PARエラーを返す．

pk_dsvcにNULL（＝0）が指定されると，すでに定義されている拡張サービスコールの定義を解除し，拡張サービスコールが未定義の状態にする．また，すでに拡張サービスコールが定義されている機能コードに対して，再度拡張サービスコールを定義した場合には，以前の定義を解除し，新しい定義に置き換える．

svcatrには，（TA_HLNG ‖ TA_ASM）の指定ができる．TA_HLNG（＝0x00）が指定された場合には高級言語用のインタフェースで，TA_ASM（＝0x01）が指

定された場合にはアセンブリ言語用のインタフェースで拡張サービスコールルーチンを起動する.

【μITRON3.0仕様との相違】

パラメータの名称をsvchdrからsvcrtnに変更した.

cal_svc　サービスコールの呼出し

【C言語API】

ER_UINT ercd = cal_svc (FN fncd, VP_INT par1, VP_INT par2, ...) ;

【パラメータ】

FN	fncd	呼び出すサービスコールの機能コード
VP_INT	par1	サービスコールへ第1パラメータ
VP_INT	par2	サービスコールへ第2パラメータ
...	...	（必要な数のパラメータを渡す）

【リターンパラメータ】

ER_UINT	ercd	サービスコールからの返値

【エラーコード】

E_RSFN　予約機能コード（fncdが不正あるいは使用できない）

【機能】

fncdで指定される機能コードのサービスコールを，par1，par2，...のパラメータで呼び出し，呼び出したサービスコールの返値を返す．

実装定義で，サービスコールに渡せるパラメータの数に，1以上の上限を設けることができる．呼び出すサービスコールのパラメータのデータ型がVP_INTでない場合には，実装で別に定義される場合を除いては，値を保存するように型変換して渡す．サービスコールの返値のデータ型がER，BOOL，ER_BOOL，ER_IDのいずれかの場合には，値を保存するようにER_UINTに型変換したものを返す．

このサービスコールで，拡張サービスコール以外の標準のサービスコールを呼び出せるかどうかは実装定義である．呼び出せない場合には，E_RSFNエラーを返す．

【補足説明】

標準のサービスコールは負の値の機能コードを持つため，拡張サービスコールと区別することができる．なお，cal_svc自身には機能コードが割り付けられていないため，cal_svcを使ってcal_svc自身を呼び出すことはできない．

【μITRON3.0仕様との相違】

新設のサービスコールである．

4.11 システム構成管理機能

システム構成管理機能には，CPU例外ハンドラを定義する機能，システムのコンフィギュレーション情報やバージョン情報を参照する機能，初期化ルーチンを定義する機能が含まれる．初期化ルーチンは，システム初期化時に実行されるルーチンである．初期化ルーチンの実行されるタイミングやコンテキストについては，3.7節を参照すること．

システム構成管理機能では，次のデータ型を用いる．

EXCNO　　CPU例外ハンドラ番号

CPU例外ハンドラの記述形式は，実装定義である．

初期化ルーチンを起動する際には，その初期化ルーチンの拡張情報（exinf）をパラメータとして渡す．初期化ルーチンのC言語による記述形式は次の通りとする．

```
void inirtn ( VP_INT exinf )
{
        初期化ルーチン本体
}
```

CPU例外ハンドラ定義情報，コンフィギュレーション情報，およびバージョン情報のパケット形式として，次のデータ型を定義する．

```
typedef struct t_dexc {
        ATR         excatr ;        /* CPU例外ハンドラ属性 */
        FP          exchdr ;        /* CPU例外ハンドラの起動番地 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_DEXC ;

typedef struct t_rcfg {
        /* 実装独自のフィールドが入る */
} T_RCFG ;

typedef struct t_rver {
        UH          maker ;         /* カーネルのメーカコード */
        UH          prid ;          /* カーネルの識別番号 */
        UH          spver ;         /* ITRON仕様のバージョン番号 */
        UH          prver ;         /* カーネルのバージョン番号 */
        UH          prno[4] ;       /* カーネル製品の管理情報 */
} T_RVER ;
```

システム構成管理機能の各サービスコールの機能コードは次の通りである．

TFN_DEF_EXC	–0x6e	def_excの機能コード
TFN_REF_CFG	–0x6f	ref_cfgの機能コード
TFN_REF_VER	–0x70	ref_verの機能コード

【スタンダードプロファイル】

スタンダードプロファイルでは，CPU 例外ハンドラを定義する静的API（DEF_EXC），初期化ルーチンを追加する静的API（ATT_INI）をサポートしな

ければならない.

【補足説明】

CPU例外ハンドラを実行するコンテキストと状態については, 次のように整理できる.

- CPU例外ハンドラ内で呼出し可能なサービスコールは実装定義である（3.4.2節参照）.
- CPU例外ハンドラは，CPU例外毎およびCPU例外が発生したコンテキスト毎に，それぞれ独立したコンテキストで実行する（3.5.1節参照）．CPU例外ハンドラを実行するコンテキストは, CPU例外が発生したコンテキストがタスクコンテキストであればタスクコンテキストと非タスクコンテキストのいずれに分類されるか実装定義であり, 非タスクコンテキストであれば非タスクコンテキストに分類される（3.5.2節参照）.
- CPU 例外ハンドラの優先順位は，CPU 例外が発生した処理の優先順位と，ディスパッチャの優先順位のいずれよりも高い（3.5.3節参照）.
- CPU例外ハンドラの起動と，そこからのリターンによって，CPUロック／ロック解除状態とディスパッチ禁止／許可状態は変化しない. CPU例外ハンドラ内でCPUロック／ロック解除状態やディスパッチ禁止／許可状態を変化させた場合には, CPU例外ハンドラからリターンする前に元の状態に戻さなければならない（3.5.4節と3.5.5節を参照）.

DEF_EXC　CPU例外ハンドラの定義（静的API）【S】
def_exc　CPU例外ハンドラの定義

【静的API】

DEF_EXC (EXCNO excno, { ATR excatr, FP exchdr }) ;

【C言語API】

ER ercd = def_exc (EXCNO excno, T_DEXC *pk_dexc) ;

【パラメータ】

EXCNO	excno	定義対象のCPU例外ハンドラ番号
T_DEXC *	pk_dexc	CPU　例外ハンドラ定義情報を入れたパケットへのポインタ（DEF_EXCではパケットの内容を直接記述する）

pk_dexcの内容（T_DEXC型）

ATR	excatr	CPU例外ハンドラ属性
FP	exchdr	CPU例外ハンドラの起動番地

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_RSATR	予約属性（excatrが不正あるいは使用できない）
E_PAR	パラメータエラー（excno，pk_dexc，exchdrが不正）

【機能】

excnoで指定されるCPU例外ハンドラ番号に対して, pk_dexcで指定されるCPU例外ハンドラ定義情報に基づいて, CPU例外ハンドラを定義する. excatrはCPU例外ハンドラの属性，exchdrはCPU例外ハンドラの起動番地である.

DEF_EXCにおいては, excnoは自動割付け非対応整数値パラメータ, excatrはプリプロセッサ定数式パラメータである.

excnoの具体的な意味は実装定義であるが，一般的な実装では，プロセッサで区別される例外の種類に対応する.

pk_dexcにNULL（＝0）が指定されると，すでに定義されているCPU例外ハンドラの定義を解除する．また，すでにCPU例外ハンドラが定義されているCPU例外ハンドラ番号に対して，再度CPU例外ハンドラを定義した場合には，以前の定義を解除し，新しい定義に置き換える.

excatrに指定できる値とその意味は実装定義である.

【μITRON3.0仕様との相違】

このサービスコールを，CPU例外ハンドラを定義するものと定めた．CPU例外ハンドラを識別するオブジェクト番号をCPU例外ハンドラ番号（excno）とし，

そのデータ型をEXCNOとした．また，excatrに指定できる値とその意味を実装定義とした．

ref_cfg　　コンフィギュレーション情報の参照

【C言語API】

ER ercd = ref_cfg (T_RCFG *pk_rcfg) ;

【パラメータ】

T_RCFG *	pk_rcfg	コンフィギュレーション情報を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rcfgの内容（T_RCFG型）

（実装独自の情報が入る）

【エラーコード】

E_PAR	パラメータエラー（pk_rcfgが不正）

【機能】

システムの静的な情報やコンフィギュレーションで指定された情報を参照し，pk_rcfgで指定されるパケットに返す. 具体的にどのような情報が参照できるかは，実装定義である.

【補足説明】

このサービスコールで参照できるようにする情報の候補としては，カーネル構成定数に定義されている値，各オブジェクトのID番号の範囲，メモリマップの概要や使用可能なメモリ量，周辺チップや入出力装置などに関する情報，時間を指定するデータ型の時間単位や精度などが挙げられる.

ref_ver バージョン情報の参照

【C言語API】

ER ercd = ref_ver (T_RVER *pk_rver) ;

【パラメータ】

T_RVER *	pk_rver	バージョン情報を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rverの内容（T_RVER型）

UH	maker	カーネルのメーカコード
UH	prid	カーネルの識別番号
UH	spver	ITRON仕様のバージョン番号
UH	prver	カーネルのバージョン番号
UH	prno[4]	カーネル製品の管理情報

【エラーコード】

E_PAR	パラメータエラー（pk_rverが不正）

【機能】

使用しているカーネルのバージョン情報を参照し，pk_rverで指定されるパケットに返す．具体的には，次の情報を参照することができる．

makerは，このカーネルを実装したメーカコードである．具体的なメーカコードの割付けについては，5.5節を参照すること．

pridは，カーネルの種類を区別するための番号である．pridの具体的な値の割付けは，カーネルを実装したメーカに任される．したがって，makerとpridの組でカーネルの種類を一意に識別することができる．

spverでは，上位4ビットでTRON仕様の種類と，下位12ビットでカーネルが準拠する仕様のバージョン番号をあらわす．spverの上位4ビットの値の具体的な割付けは次の通りである．

0x0	TRON共通の仕様（TADなど）
0x1	ITRON仕様（ITRON1，ITRON2）
0x2	BTRON仕様
0x3	CTRON仕様
0x5	µITRON仕様（µITRON2.0，µITRON3.0，µITRON4.0）
0x6	µBTRON仕様

spverの下位12ビットには，仕様のバージョン番号の上位3桁を，上位から4ビットずつに入れる．すなわち，バージョン番号の上位3桁をBCD形式で入れたことになる．検討中の仕様や暫定仕様では，バージョン番号に英文字が使われることがあるが，その場合にはその英文字を16進数と解釈した値を入れる．

ITRON仕様のバージョン番号の形式については，5.4節を参照すること．

prverは，カーネル実装上のバージョン番号をあらわす．prverの具体的な値の割付けは，カーネルを実装したメーカに任される．

prnoは，カーネル製品の管理情報や製品番号などを入れるために使用するためのリターンパラメータである．prnoの具体的な値の意味は，カーネルを実装したメーカに任される．

【補足説明】

例えば，μITRON4.0仕様のVer. 4.02.10に準拠するカーネルのspverは0x5402，Ver. 4.A1.01に準拠するカーネルのspverは0x54A1となる．このように，ドラフト版の仕様に準拠して実装されたカーネルでは，仕様書の新旧関係とspverの大小関係が必ずしも一致しない．

このサービスコールが返す情報の内，prno以外の4つの情報は，それぞれカーネル構成定数TKERNEL_MAKER，TKERNEL_PRID，TKERNEL_SPVER，TKERNEL_PRVERによって参照することができる．

【μITRON3.0仕様との相違】

サービスコールの名称を，get_verからref_verに変更した．参照できる情報から，CPU情報とバリエーション記述子を削除し，prverのフォーマットを規定しないこととした．また，リターンパラメータの名称をidからpridに変更した．

【仕様決定の理由】

バージョン番号の上位3桁より下位の桁はspverに入れていないが，これは，これらの桁が仕様の表記に関するバージョンを示すものであり，仕様の内容には関係しないためである．

ATT_INI　初期化ルーチンの追加（静的API）　【S】

【静的API】

ATT_INI ({ ATR iniatr, VP_INT exinf, FP inirtn }) ;

【パラメータ】

ATR	iniatr	初期化ルーチン属性
VP_INT	exinf	初期化ルーチンの拡張情報
FP	inirtn	初期化ルーチンの起動番地

（実装独自に他の情報を追加してもよい）

【機能】

初期化ルーチンを，指定される各パラメータに基づいて追加する．iniatrは初期化ルーチンの属性, exinfは初期化ルーチンを起動する時にパラメータとして渡す拡張情報，inirtnは初期化ルーチンの起動番地である．

ATT_INIにおいては，iniatrはプリプロセッサ定数式パラメータである．

追加された初期化ルーチンは，システム初期化時に，静的APIの処理の一環として実行する．詳しくは，3.7節を参照すること．

iniatrには，（TA_HLNG || TA_ASM）の指定ができる．TA_HLNG（＝0x00）が指定された場合には高級言語用のインタフェースで，TA_ASM（＝0x01）が指定された場合にはアセンブリ言語用のインタフェースで初期化ルーチンを起動する．

【スタンダードプロファイル】

スタンダードプロファイルでは，iniatr に TA_ASM が指定された場合の機能はサポートする必要がない．

【補足説明】

システムコンフィギュレーションファイル中には，複数のATT_INIを記述することができる．ATT_INI を複数記述した場合の実行順序については，3.7 節を参照すること．

【μITRON3.0仕様との相違】

新たに導入した機能である．

第5章 付属規定

5.1 μITRON4.0仕様準拠の条件

5.1.1 基本的な考え方

μITRON仕様は，弱い標準化の方針に基づいて策定された仕様である．アプリケーションプログラムの移植性よりも，多様なハードウェアやアプリケーションに対する適応化を重視し，ソフトウェア技術者の教育を容易にするための標準化を狙っている．そのため，リアルタイムカーネルとしての最低限の要件を満たしていれば，仕様で定められている機能をどこまで持たせるかや，実装独自の拡張機能を持たせるかどうかは，実装毎に自由に決定することができる．

具体的には，実装がμITRON4.0仕様に準拠しているというためには，以下の条件を満たしていなければならない．

(a) μITRON4.0 仕様に準拠しているというために必要な最低限の機能を持っていること（5.1.2節参照）．

(b) μITRON4.0 仕様で規定されている機能と同等の機能を持つ場合には，その機能仕様が，μITRON4.0仕様の規定に合致していること．ただし，コンフィギュレータを用意しない場合には, μITRON4.0仕様の静的APIの仕様に合致している必要はない．

(c) μITRON4.0仕様で規定されていない機能を持つ場合には，その機能仕様が，μITRON4.0 仕様が実装独自の拡張方法に関して設けている規定に合致していること．ただし，実装が複数種類の API をサポートする場合には，μITRON4.0仕様準拠のAPI以外のAPIに対してはこの規定は適用されない．

また，サービスコールの機能をサブセット化する場合や機能に制限を設ける場合，μITRON4.0仕様で規定されていない実装独自の機能を持つ場合には，実装について説明する製品マニュアルなどで，それらの点を明示しなければならない．

プロファイル規定は，高級言語で記述されたアプリケーションプログラムの移植性を保つために，カーネルが最低限満たすべき機能条件を定めるものである．μITRON4.0仕様に準拠した実装があるプロファイル規定に準拠するためには，そのプロファイル規定に含まれるすべての機能を持っており，プロファイル規定に関するすべての規定に合致していることが必要である．プロファイル規定に含まれていない機能や実装独自の拡張機能を持っていてもよいが，プロファイル規定に含まれる機能のみで動作するように記述されたアプリケーションプログラムが，そのまま動作することが必要である．

なお，実装したカーネルをアプリケーションに組み込む場合には，アプリケーションプログラムが必要としている機能だけを組み込むことができる．

【スタンダードプロファイル】

スタンダードプロファイルは，μITRON4.0仕様におけるプロファイル規定の一つである．

【補足説明】

実装が μITRON4.0 仕様に準拠しているというための条件を例を使って説明する．例えば，実装がセマフォの機能を持つ場合には，サービスコールの名称と機能，パラメータとリターンパラメータの種類・順序・名称・データ型，メインエラーコードの種類と名称などが，μITRON4.0 仕様に規定されているセマフォ機能に合致していることが必要である．その際に，アプリケーションプログラムの移植性を犠牲にして，サービスコールの機能をサブセット化することは許される（μITRON4.0仕様で規定されている機能と同等の機能を持たないと見なせるため）．また，優先度継承機能を持った計数型セマフォなど，同等の機能がμITRON4.0仕様に規定されていない機能については，実装独自に機能仕様を定めることができる．さらに，サブセット化や実装独自の機能仕様で必要になる場合には，パラメータやリターンパラメータを追加または削除することも許される．

この節の条件は，アプリケーションに組み込まれた状態について規定するものではない．スタンダードプロファイルに準拠したカーネルであっても，アプリケーションに組み込む際には，アプリケーションプログラムが必要としているものだけに機能を絞り込んだり，ID番号や優先度の範囲を制限することができる．

5.1.2 μITRON4.0仕様準拠の最低機能

μITRON4.0 仕様に準拠しているというために必要な最低限の機能は次の通りである．

(a) タスクを生成できること．タスクは少なくとも，実行状態，実行可能状態，休止状態の3つの状態を持つこと．

(b) μITRON4.0 仕様のスケジューリング規則に従ったタスクスケジューリングを行うこと．ただし，優先度毎のタスクを1つに制限することや，優先度を1段階に制限することは許される．

(c) 割込みハンドラ（または割込みサービスルーチン）を登録できること．

(d) タスクおよび割込みハンドラ（または割込みサービスルーチン）から，タスクを起動する（休止状態から実行できる状態にする）手段が用意されていること．

(e) 自タスクを終了する（実行状態から休止状態にする）手段が用意されていること．

【補足説明】

例えば，以下のサービスコールおよび静的APIを用意し，タスクスケジューリ

ングの規則が仕様に従っていれば，上記の最低限の機能を満たすことができる．

CRE_TSK	タスクの生成（静的API）
act_tsk／iact_tsk	タスクの起動
ext_tsk	自タスクの終了
DEF_INH	割込みハンドラの定義（静的API）

この中で，割込みハンドラの定義（DEF_INH）は，割込みサービスルーチンの追加生成（ATT_ISR）で代えることができる．コンフィギュレータを用意しない場合には，静的APIの仕様に合致している必要はなく，それと同等の手段が用意されていればよい．また，act_tsk，iact_tskでは起動要求のキューイングをサポートする必要はなく，ext_tskはメインルーチンからのリターンで代用することができる．

【μITRON3.0仕様との相違】

タスクが最低限持つべき状態を，実行状態，実行可能状態，待ち状態の3つから，実行状態，実行可能状態，休止状態の3つに変更した．また，最低限サポートすべきサービスコール（レベルR）は規定しないこととした．

5.1.3 μITRON4.0仕様に対する拡張

μITRON4.0仕様で規定されていない機能を実現するために，実装独自にサービスコールを追加する場合には，サービスコール名称の前に“v”を付加しなければならない．実装独自の静的APIの名称もこれに準ずる．ただし，非タスクコンテキストから呼び出せるサービスコールを実装独自に追加する場合の名称は，この規則の例外となる（3.6.3節参照）．また，実装独自のサービスコールの機能コードの値は，そのために用意された範囲内で定めなければならない．

実装独自にメインエラーコードを追加する場合には，その名称をEV_*XXXXX*の形とし，メインエラーコードの値をそのために用意された範囲内で定めなければならない．また，実装独自にデータ型，エラーコード以外の定数やマクロを定義する場合にも，名称中に“V”を入れるなどの方法で，μITRON4.0仕様で定義されたものと区別することが推奨される．

μITRON4.0仕様では，オブジェクト属性とサービスコールの動作モードを指定する定数には，8ビットで表現できる値が割り付けられている．また，一部の例外を除いて，オブジェクトの状態をあらわす定数にも8ビットで表現できる値が割り付けられている．これらのパラメータまたはリターンパラメータの値の下位8ビットは，ITRON仕様の将来の拡張のために予約されている．これらのパラメータまたはリターンパラメータに用いるための定数を実装独自に追加する場合には，この仕様に規定されている定数で用いられているビットと，予約されている下位8ビットを除いた，残りのビットを用いた値を割り付けなければならない．

その他，オブジェクト登録情報やオブジェクト状態を入れるためのパケット形

式など，実装独自に拡張する方法について規定がある場合には，その規定に合致していることが必要である．

5.2 ベーシックプロファイル

ベーシックプロファイルは，μITRON3.0/μITRON4.0/T-Kernelにおいて同等の機能を持つサービスコールを規定したもので，アプリケーションの移植をより簡単にすることを目的としたプロファイルである．
ベーシックプロファイル抽出の原則は次の通り．

・μITRON4.0仕様のスタンダードプロファイルにある機能を対象とする．
・μITRON3.0/T-Kernel仕様にない機能は対象外とする．
・μITRON3.0/μITRON4.0/T-Kernel仕様において機能が対応するものを対象とする．(ただし，名称や細かい仕様の点で異なるものがある．
　xxx_msg/xxx_mbx, wai_flgなど)
・μITRON3.0でレベル[R]/[S]に分類されるものを対象とする．ただし，レベル[E]に分類される次の機能は，μITRON4.0仕様で基本機能と位置づけられるため，対象とする．(固定長メモリプール，周期ハンドラ)
・sta_tskはタスクを起動する共通サービスコールとして対象とする．

(1) タスク管理機能

sta_tsk	タスクの起動（起動コード指定）
ext_tsk	自タスクの終了
ter_tsk	タスクの強制終了
chg_pri	タスク優先度の変更

(2) タスク付属同期機能

slp_tsk	起床待ち
wup_tsk／iwup_tsk	タスクの起床
can_wup	タスク起床要求のキャンセル
rel_wai／irel_wai	待ち状態の強制解除
sus_tsk	強制待ち状態への移行
rsm_tsk	強制待ち状態からの再開
dly_tsk	自タスクの遅延

(4) 同期・通信機能

セマフォ

sig_sem／isig_sem	セマフォ資源の返却
wai_sem	セマフォ資源の獲得

pol_sem	セマフォ資源の獲得（ポーリング）
イベントフラグ	
set_flg／iset_flg	イベントフラグのセット
clr_flg	イベントフラグのクリア
wai_flg	イベントフラグ待ち
pol_flg	イベントフラグ待ち（ポーリング）
メールボックス	
snd_mbx	メールボックスへの送信
rcv_mbx	メールボックスからの受信
prcv_mbx	メールボックスからの受信（ポーリング）

(6) メモリプール管理機能

固定長メモリプール	
get_mpf	固定長メモリブロックの獲得
pget_mpf	固定長メモリブロックの獲得（ポーリング）
rel_mpf	固定長メモリブロックの返却

(7) 時間管理機能

周期ハンドラ	
sta_cyc	周期ハンドラの動作開始
stp_cyc	周期ハンドラの動作停止

(8) システム状態管理機能

rot_rdq／irot_rdq	タスクの優先順位の回転
get_tid	実行状態のタスクIDの参照
loc_cpu	CPUロック状態への移行
unl_cpu	CPUロック状態の解除
dis_dsp	ディスパッチの禁止
ena_dsp	ディスパッチの許可

5.3 自動車制御用プロファイル

μITRON4.0仕様の自動車制御用プロファイルは，自動車制御応用を主なターゲットとして定められたμITRON4.0仕様のプロファイル規定の1つである．カーネルのオーバヘッドやメモリ使用量の削減を目指して，機能のサブセット化およびメモリ使用量を削減するための機能追加を行っている．

具体的には，スタンダードプロファイルと比較して，自動車制御用プロファイルでは，次の機能をサポートする必要はない．

- タイムアウト付きのサービスコール
- タスクの優先度順の待ち行列
- 強制待ち状態
- タスク例外処理機能

- メールボックス
- 固定長メモリプール
- その他，いくつかのサービスコール

逆に，メモリ使用量を削減するために追加された機能として，制約タスクの機能がある．制約タスクは，文字どおり，通常のタスクに比べて機能が制限されたものであるため，制限されて使えなくなった機能を使おうとした場合にエラーが返る（E_NOSPT エラーが返る）ことに依存していない限りは，通常のタスクに置き換えても同じ振舞いをする．この意味で，制約タスクの機能が追加されているにも関わらず，自動車制御用プロファイルはスタンダードプロファイルに対する下位互換性を有している．

5.3.1 制約タスク

制約タスクは，タスクの持つ機能を制限することで，複数のタスクを同一のスタック空間を用いて動作させることを可能にし，それによりタスクのスタックのためのメモリ領域の削減を図るものである．具体的には，制約タスクは，通常のタスクと比較して以下の制限を持つ．

- 待ち状態に入ることができない．
- 優先度を変更することができない．また，タスク生成時の初期優先度の指定に，式を用いることができない．
- タスクのメインルーチンからのリターン以外の方法で，タスクを終了することはできない．

タスクが制約タスクであるかどうかは，タスクの生成時に，タスク属性によって指定する．

【補足説明】

制約タスクにおいても，タスクの生成時のパラメータの1つであるスタック領域のサイズの指定は有効である．例えば，同じ優先度を持った複数の制約タスクに同じスタック領域を割り付ける場合，カーネルで確保するスタック領域のサイズを，各タスクのスタックサイズの最大値にすることが必要である．スタンダードプロファイルと同様に，スタック領域の先頭番地の（NULL 以外の）指定はサポートする必要がない．

5.3.2 自動車制御用プロファイルに含まれる機能

自動車制御用プロファイルは，制約タスクの機能を除いては，スタンダードプロファイルに含まれている機能のみを含む．自動車制御用プロファイルでサポートしなければならない静的APIおよびサービスコールは，次の通りである．

(1) タスク管理機能

CRE_TSK	タスクの生成（静的API）
act_tsk／iact_tsk	タスクの起動

can_act	タスク起動要求のキャンセル
ext_tsk	自タスクの終了
ter_tsk	タスクの強制終了
chg_pri	タスク優先度の変更
get_pri	タスク優先度の参照

(2) タスク付属同期機能

slp_tsk	起床待ち
wup_tsk／iwup_tsk	タスクの起床
can_wup	タスク起床要求のキャンセル
rel_wai／irel_wai	待ち状態の強制解除

(4) 同期・通信機能

セマフォ

CRE_SEM	セマフォの生成（静的API）
sig_sem／isig_sem	セマフォ資源の返却
wai_sem	セマフォ資源の獲得
pol_sem	セマフォ資源の獲得（ポーリング）

イベントフラグ

CRE_FLG	イベントフラグの生成（静的API）
set_flg／iset_flg	イベントフラグのセット
clr_flg	イベントフラグのクリア
wai_flg	イベントフラグ待ち
pol_flg	イベントフラグ待ち（ポーリング）

データキュー

CRE_DTQ	データキューの生成（静的API）
psnd_dtq／ipsnd_dtq	データキューへの送信（ポーリング）
fsnd_dtq／ifsnd_dtq	データキューへの強制送信
rcv_dtq	データキューからの受信
prcv_dtq	データキューからの受信（ポーリング）

(7) 時間管理機能

システム時刻管理

isig_tim	タイムティックの供給

※システム時刻を更新する機構をカーネル内部に持つ場合には，isig_timをサポートする必要はない．

周期ハンドラ

CRE_CYC	周期ハンドラの生成（静的API）
sta_cyc	周期ハンドラの動作開始
stp_cyc	周期ハンドラの動作停止

(8) システム状態管理機能

get_tid／iget_tid	実行状態のタスクIDの参照

loc_cpu／iloc_cpu	CPUロック状態への移行
unl_cpu／iunl_cpu	CPUロック状態の解除
dis_dsp	ディスパッチの禁止
ena_dsp	ディスパッチの許可
sns_ctx	コンテキストの参照
sns_loc	CPUロック状態の参照
sns_dsp	ディスパッチ禁止状態の参照
sns_dpn	ディスパッチ保留状態の参照

(9) 割込み管理機能

DEF_INH	割込みハンドラの定義（静的API）

※ ATT_ISRをサポートした場合には，DEF_INHをサポートする必要はない．

(11) システム構成管理機能

DEF_EXC	CPU例外ハンドラの定義（静的API）
ATT_INI	初期化ルーチンの追加（静的API）

これらの静的APIおよびサービスコールの中で，自動車制御用プロファイルでサポートしなければならない機能が，スタンダードプロファイルに比べて制限ないしは拡張されているものは次の通りである．

• CRE_TSK

タスク属性にTA_RSTR（＝0x04）を指定することができる．TA_RSTRが指定されると，制約タスクを生成する．TA_RSTRが指定された場合には，itskpriは自動割付け非対応整数値パラメータとなる．

• CRE_SEM，CRE_FLG，CRE_DTQ

各オブジェクトの属性にTA_TPRIが指定された場合の機能はサポートする必要がない．

• ext_tsk

制約タスクから呼び出された場合の振舞いは実装定義である．

• ter_tsk，chg_pri

対象タスクが制約タスクである場合には，E_NOSPTエラーを返す．

• slp_tsk，wai_sem，wai_flg，rcv_dtq

制約タスクから呼び出された場合には，E_NOSPTエラーを返す．

【補足説明】

自動車制御用プロファイルの範囲では，イベントフラグ属性にTA_TFIFO，TA_TPRIのいずれが指定されたとしても，その振舞いは同じである．また，タスクがデータキューへの送信待ち状態になることはないため，データキュー属性のTA_TFIFO，TA_TPRIの指定は意味を持たない．したがって，イベントフラグ属性とデータキュー属性にTA_TPRIが指定された場合の機能をサポートする必要がないという制限は，実質的には，TA_TPRIが指定された場合にエ

ラーとしても良いことのみを示している．

5.4 仕様のバージョン番号

ITRON仕様のバージョン番号は，次の形式とする．

Ver. *X.YY.ZZ*［*.WW*］

この内，*X*はITRON仕様のメジャーなバージョンを示す．カーネル仕様の場合の割付けは次の通りである．

1	ITRON1
2	ITRON2またはμITRON（Ver. 2.0）
3	μITRON3.0
4	μITRON4.0

*YY*は, 仕様の内容に関する変更や追加を伴うようなバージョンの区別を示す番号である．仕様の公開後は，仕様のバージョンアップにあわせて，*YY*＝00，01，02，...の順に増加させる．仕様の呼称を変更するような大幅なバージョンアップの場合には，*YY*を不連続に増加させることもある．一方，検討中の仕様や暫定仕様（ドラフト）の場合は，*YY*のいずれかの桁を‘A’，‘B’，‘C’のいずれかとする．

バージョン番号の内，*X.YY*の部分は，カーネル構成マクロTKERNEL_SPVERとref_verサービスコールのリターンパラメータspverとして参照することができる．*YY*が‘A’，‘B’，‘C’を含む場合は，16進数の‘A’，‘B’，‘C’を用いる．

*ZZ*は, 仕様書の表記に関するバージョンの区別を示す番号である．仕様書の構成や章立ての変更，ミスプリントの修正などがあった場合に，*ZZ*＝00，01，02，...の順に増加させる．

仕様書の表記に関してさらにマイナーな区別を行いたい場合には，*WW*の桁を設ける場合がある．*WW*が省略されている場合には，*WW*が00であるものとみなす．

5.5 メーカコード

カーネル構成マクロTKERNEL_MAKERとref_verサービスコールのリターンパラメータmakerとして参照できるメーカコードは，トロン協会が割り付ける．

仕様書の発行時点で割り付けられているメーカコードは次の通りである．

0x0000	メーカコードなし（実験システムなど）
0x0001	東京大学 坂村研究室
0x0002	大学・研究機関
0x0008	個人（または個人事業者）
0x0009	富士通株式会社
0x000a	株式会社日立製作所

0x000b	松下電器産業株式会社
0x000c	三菱電機株式会社
0x000d	日本電気株式会社
0x000e	沖電気工業株式会社
0x000f	株式会社東芝
0x0010	アルプス電気株式会社
0x0011	株式会社ワコム
0x0012	パーソナルメディア株式会社
0x0013	ソニー株式会社
0x0014	Motorola, Inc.
0x0015	National Semiconductor Corporation
0x0101	オムロン株式会社
0x0102	セイコープレシジョン株式会社
0x0103	株式会社システムアルゴ
0x0104	東京コンピュータサービス株式会社
0x0105	ヤマハ株式会社
0x0106	株式会社モアソンジャパン
0x0107	東芝情報システム株式会社
0x0108	株式会社ミスポ
0x0109	スリーエース・コンピュータ株式会社
0x010a	ファームウェアシステム株式会社
0x010b	イーソル株式会社
0x010c	U S Software Corporation
0x010d	株式会社ACCESS
0x010e	富士通デバイス株式会社
0x010f	Mentor Graphics Corporation
0x0110	エルミック・ウェスコム株式会社
0x0111	ウェブテクノロジー株式会社
0x0112	株式会社エーアイコーポレーション
0x0113	株式会社グレープシステム
0x0114	株式会社日立情報制御ソリューションズ
0x0115	株式会社ルネサス テクノロジ
0x0116	茨城日立情報サービス株式会社
0x0117	NECエレクトロニクス株式会社
0x0118	特定非営利活動法人TOPPERSプロジェクト
0x0119	株式会社日新システムズ

大学・研究機関が実装したカーネルはメーカコードを0x0002に，個人（または個人事業者）が実装したカーネルはメーカコードを 0x0008 にする．さらに，カーネルの実装者を識別できるようにするために，大学・研究機関の研究室および個人（または個人事業者）に対して，カーネルの識別番号（カーネル構成マクロ TKERNEL_PRID と ref_ver サービスコールのリターンパラメータ prid と

して参照できる値）の上位8ビットに用いる値を割り付ける．

第6章 付録

6.1 仕様と仕様書の利用条件

μITRON4.0仕様および仕様書の利用条件は次の通りである．

仕様の利用条件

μITRON4.0仕様は，オープンな仕様である．誰でも自由に，μITRON4.0仕様に準拠したソフトウェアを開発・使用・配布・販売することができる．

ただし，トロン協会は，μITRON4.0仕様に準拠したソフトウェアの製品マニュアルなどに，以下の文言（ないしは同じ趣旨の文言）を入れることを強く推奨する．

- TRONは“The Real-time Operating system Nucleus”の略称です．
- ITRONは“Industrial TRON”の略称です．
- μITRONは“Micro Industrial TRON”の略称です．
- TRON，ITRON，およびμITRONは，特定の商品ないしは商品群を指す名称ではありません．

また，μITRON4.0仕様に準拠したソフトウェアの製品マニュアルなどに，以下の文言（ないしは同じ趣旨の文言）を入れることを推奨する．

> μITRON4.0仕様は，トロン協会が定めたオープンなリアルタイムカーネル仕様です．μITRON4.0仕様の仕様書は，トロン協会のホームページ（http://www.assoc.tron.org/）から入手することができます．

仕様書を改変して製品マニュアルを作成する許諾を受けた場合（後述）や，ITRON仕様準拠製品登録制度に登録する場合（6.2節参照）には，これらの2つの推奨に従うことが条件となる．

仕様書の利用条件

μITRON4.0仕様書の著作権はトロン協会に属する．

トロン協会は，μITRON4.0仕様書の全文または一部分を改変することなく複写し，無料または実費程度の費用で再配布することを許諾する．ただし，μITRON4.0仕様書の一部分を再配布する場合には，μITRON4.0仕様書からの抜粋である旨，抜粋した箇所，およびμITRON4.0仕様書の全文を入手する方法を明示しなければならない．

トロン協会の書面による事前の許可がない限り，μITRON4.0仕様書を改変することは堅く禁ずる．

トロン協会は，トロン協会会員に対してのみ，μITRON4.0仕様書を改変して製品マニュアルを作成し，それを配布・販売することを許諾している．許諾条件

やその申請方法については，トロン協会に問い合わせること．

無保証

トロン協会は，µITRON4.0仕様および仕様書に関して，いかなる保証も行わない．また，µITRON4.0仕様および仕様書を利用したことによって直接的または間接的に生じたいかなる損害も補償しない．

また，トロン協会は，µITRON4.0仕様書を予告なく改訂する場合がある．

6.2 仕様の保守体制と参考情報

ITRON仕様の保守体制と問い合わせ先

ITRON仕様および仕様書の作成・保守は，トロン協会ITRON仕様検討グループが行っている．仕様ならびに仕様書に関する問い合わせ先は，次の通りである．

(社)トロン協会 ITRON仕様検討グループ
〒108-0073 東京都港区三田1丁目3番39号 勝田ビル5階
TEL: 03-3454-3191　FAX: 03-3454-3224

ITRON仕様準拠製品登録制度

トロン協会では，ITRON仕様の普及と発展を促進するため，ITRON仕様準拠製品登録制度を設けている．この制度は，各社で開発されたITRON仕様準拠の製品の一覧を作成・保守し，トロン協会の広報活動などを通じて，ITRON仕様ならびに準拠製品の普及を図ることを目的としたものである．なお，この制度は，いわゆる検定制度とは異なり，登録された製品がITRON仕様に準拠していることを認証するためのものではない．

この制度に登録されている製品の一覧は，トロン協会のホームページで閲覧することができる．また，ITRON仕様準拠製品登録制度への登録を希望される方は，トロン協会まで連絡されたい．

参考文献

トロンプロジェクト全般の紹介資料として，「THE TRON PROJECT」がトロン協会により定期的に作成され，無料で配布されている．この資料には，トロンの各サブプロジェクトや応用プロジェクトの活動状況の紹介や，トロンプロジェクトの歴史，トロンプロジェクトに関する参考文献リストが含まれている．

トロンプロジェクトの最新情報については，パーソナルメディア社が隔月で発行しているTRON関連技術情報誌「TRONWARE」が参考になる．

ITRON仕様に関する教科書として，トロン協会ITRON仕様検討グループでは，

ITRON標準ガイドブックを編集・発行している．

坂村 健 監修，「μITRON4.0 標準ガイドブック」，パーソナルメディア，2001（ISBN4-89362-191-2）．

6.3 仕様策定の経緯とバージョン履歴

仕様策定の経緯

トロン協会のITRON仕様検討グループ（旧 ITRON部会）では，ハードリアルタイムサポート研究会（1996年11月～1998年3月）とRTOS自動車応用技術委員会（1997年6月～1998年3月）での検討結果を受けて，次世代のμITRON仕様を策定することを目的に，μITRON4.0仕様研究会(1998年4月～2001年4月）を設けて仕様検討を行い, 1999年6月にμITRON4.0仕様の初版であるVer.4.00.00を公開した．

このVer.4.00.00には，ITRON TCP/IP API仕様Ver.1.0の検討，JTRON2.0仕様の検討, μITRON4.0仕様研究会 デバイスドライバ設計ガイドラインWGにおける検討の中から出てきた要求事項やアイデアも反映された．

その後, 継続的に仕様書の見直しを行ない, 以下に記すように2001年, 2004年にそれぞれVer.4.01.00とVer.4.02.00を発行し,細部の不都合, 誤り, 曖昧な文言の修正の面での仕様書の改善を図ってきた．

今回のVer.4.03.00においては，過去２回において行った観点からの仕様書の見直し以外に，従来からあった「スタンダードプロファイル」と「自動車制御用プロファイル」に加えて，μITRON3.0，μITRON4.0，T-Kernel間でアプリケーションの移植を容易にするためのプロファイル「ベーシックプロファイル」を新たに規定した．また，実装定義を要する項目の一覧表を掲載して利用者への利便性の向上を図った．

μITRON Ver.4.00.00以降の仕様改訂履歴を以下に記す．

バージョン履歴

1999年6月30日	Ver. 4.00.00	初版公開，坂村健 監修／高田広章 編
2001年5月31日	Ver. 4.01.00	改訂版公開，〃
2004年3月30日	Ver. 4.02.00	〃 〃
2006年12月5日	Ver. 4.03.00	〃 坂村健 監修／トロン協会 編 ベーシックプロファイル新設等を含む

尚，Ver.4.03.00への改訂を行ったITRON仕様メンテナンスWGにご参加を戴

いたメンバーを以下に記載致しますと共に，そのご協力に対して謝意を表します．

金田一勉（エルミック・ウェスコム株式会社）
金子健（イーソル株式会社）
小林康浩（富士通株式会社）
竹内透（富士通株式会社）
鈴木克義（茨城日立情報サービス株式会社）
高倉規彰（NECエレクトロニクス株式会社）
檜原弘樹（ NEC東芝スペースシステム株式会社）
乾聡之（株式会社ルネサス ソリューションズ）
福井昭也（株式会社ルネサス ソリューションズ）
山田真二郎（株式会社ルネサス ソリューションズ）
リーダ： 宮本博暢（株式会社東芝）
（会社名アルファベット順）

第7章 リファレンス

7.1 サービスコール一覧

(1) タスク管理機能

ER ercd = cre_tsk (ID tskid, T_CTSK *pk_ctsk) ;
ER_ID tskid = acre_tsk (T_CTSK *pk_ctsk) ;
ER ercd = del_tsk (ID tskid) ;
ER ercd = act_tsk (ID tskid) ;
ER ercd = iact_tsk (ID tskid) ;
ER_UINT actcnt = can_act (ID tskid) ;
ER ercd = sta_tsk (ID tskid, VP_INT stacd) ;
void ext_tsk ( ) ;
void exd_tsk ( ) ;
ER ercd = ter_tsk (ID tskid) ;
ER ercd = chg_pri (ID tskid, PRI tskpri) ;
ER ercd = get_pri (ID tskid, PRI *p_tskpri) ;
ER ercd = ref_tsk (ID tskid, T_RTSK *pk_rtsk) ;
ER ercd = ref_tst (ID tskid, T_RTST *pk_rtst) ;

(2) タスク付属同期機能

ER ercd = slp_tsk ( ) ;
ER ercd = tslp_tsk (TMO tmout) ;
ER ercd = wup_tsk (ID tskid) ;
ER ercd = iwup_tsk (ID tskid) ;
ER_UINT wupcnt = can_wup (ID tskid) ;
ER ercd = rel_wai (ID tskid) ;
ER ercd = irel_wai (ID tskid) ;
ER ercd = sus_tsk (ID tskid) ;
ER ercd = rsm_tsk (ID tskid) ;
ER ercd = frsm_tsk (ID tskid) ;
ER ercd = dly_tsk (RELTIM dlytim) ;

(3) タスク例外処理機能

ER ercd = def_tex (ID tskid, T_DTEX *pk_dtex) ;
ER ercd = ras_tex (ID tskid, TEXPTN rasptn) ;
ER ercd = iras_tex (ID tskid, TEXPTN rasptn) ;
ER ercd = dis_tex ( ) ;
ER ercd = ena_tex ( ) ;
BOOL state = sns_tex ( ) ;
ER ercd = ref_tex (ID tskid, T_RTEX *pk_rtex) ;

(4) 同期・通信機能

セマフォ

```
ER ercd = cre_sem ( ID semid, T_CSEM *pk_csem ) ;
ER_ID semid = acre_sem ( T_CSEM *pk_csem ) ;
ER ercd = del_sem ( ID semid ) ;
ER ercd = sig_sem ( ID semid ) ;
ER ercd = isig_sem ( ID semid ) ;
ER ercd = wai_sem ( ID semid ) ;
ER ercd = pol_sem ( ID semid ) ;
ER ercd = twai_sem ( ID semid, TMO tmout ) ;
ER ercd = ref_sem ( ID semid, T_RSEM *pk_rsem ) ;
```

イベントフラグ

```
ER ercd = cre_flg ( ID flgid, T_CFLG *pk_cflg ) ;
ER_ID flgid = acre_flg ( T_CFLG *pk_cflg ) ;
ER ercd = del_flg ( ID flgid ) ;
ER ercd = set_flg ( ID flgid, FLGPTN setptn ) ;
ER ercd = iset_flg ( ID flgid, FLGPTN setptn ) ;
ER ercd = clr_flg ( ID flgid, FLGPTN clrptn ) ;
ER ercd = wai_flg ( ID flgid, FLGPTN waiptn, MODE wfmode,
                        FLGPTN *p_flgptn ) ;
ER ercd = pol_flg ( ID flgid, FLGPTN waiptn, MODE wfmode,
                        FLGPTN *p_flgptn ) ;
ER ercd = twai_flg ( ID flgid, FLGPTN waiptn, MODE wfmode,
                        FLGPTN *p_flgptn, TMO tmout ) ;
ER ercd = ref_flg ( ID flgid, T_RFLG *pk_rflg ) ;
```

データキュー

```
ER ercd = cre_dtq ( ID dtqid, T_CDTQ *pk_cdtq ) ;
ER_ID dtqid = acre_dtq ( T_CDTQ *pk_cdtq ) ;
ER ercd = del_dtq ( ID dtqid ) ;
ER ercd = snd_dtq ( ID dtqid, VP_INT data ) ;
ER ercd = psnd_dtq ( ID dtqid, VP_INT data ) ;
ER ercd = ipsnd_dtq ( ID dtqid, VP_INT data ) ;
ER ercd = tsnd_dtq ( ID dtqid, VP_INT data, TMO tmout ) ;
ER ercd = fsnd_dtq ( ID dtqid, VP_INT data ) ;
ER ercd = ifsnd_dtq ( ID dtqid, VP_INT data ) ;
ER ercd = rcv_dtq ( ID dtqid, VP_INT *p_data ) ;
ER ercd = prcv_dtq ( ID dtqid, VP_INT *p_data ) ;
ER ercd = trcv_dtq ( ID dtqid, VP_INT *p_data, TMO tmout ) ;
ER ercd = ref_dtq ( ID dtqid, T_RDTQ *pk_rdtq ) ;
```

メールボックス

```
ER ercd = cre_mbx ( ID mbxid, T_CMBX *pk_cmbx ) ;
```

```
ER_ID mbxid = acre_mbx ( T_CMBX *pk_cmbx ) ;
ER ercd = del_mbx ( ID mbxid ) ;
ER ercd = snd_mbx ( ID mbxid, T_MSG *pk_msg ) ;
ER ercd = rcv_mbx ( ID mbxid, T_MSG **ppk_msg ) ;
ER ercd = prcv_mbx ( ID mbxid, T_MSG **ppk_msg ) ;
ER ercd = trcv_mbx ( ID mbxid, T_MSG **ppk_msg, TMO tmout ) ;
ER ercd = ref_mbx ( ID mbxid, T_RMBX *pk_rmbx ) ;
```

(5) 拡張同期・通信機能

ミューテックス

```
ER ercd = cre_mtx ( ID mtxid, T_CMTX *pk_cmtx ) ;
ER_ID mtxid = acre_mtx ( T_CMTX *pk_cmtx ) ;
ER ercd = del_mtx ( ID mtxid ) ;
ER ercd = loc_mtx ( ID mtxid ) ;
ER ercd = ploc_mtx ( ID mtxid ) ;
ER ercd = tloc_mtx ( ID mtxid, TMO tmout ) ;
ER ercd = unl_mtx ( ID mtxid ) ;
ER ercd = ref_mtx ( ID mtxid, T_RMTX *pk_rmtx ) ;
```

メッセージバッファ

```
ER ercd = cre_mbf ( ID mbfid, T_CMBF *pk_cmbf ) ;
ER_ID mbfid = acre_mbf ( T_CMBF *pk_cmbf ) ;
ER ercd = del_mbf ( ID mbfid ) ;
ER ercd = snd_mbf ( ID mbfid, VP msg, UINT msgsz ) ;
ER ercd = psnd_mbf ( ID mbfid, VP msg, UINT msgsz ) ;
ER ercd = tsnd_mbf ( ID mbfid, VP msg, UINT msgsz, TMO tmout ) ;
ER_UINT msgsz = rcv_mbf ( ID mbfid, VP msg ) ;
ER_UINT msgsz = prcv_mbf ( ID mbfid, VP msg ) ;
ER_UINT msgsz = trcv_mbf ( ID mbfid, VP msg, TMO tmout ) ;
ER ercd = ref_mbf ( ID mbfid, T_RMBF *pk_rmbf ) ;
```

ランデブ

```
ER ercd = cre_por ( ID porid, T_CPOR *pk_cpor ) ;
ER_ID porid = acre_por ( T_CPOR *pk_cpor ) ;
ER ercd = del_por ( ID porid ) ;
ER_UINT rmsgsz = cal_por ( ID porid, RDVPTN calptn, VP msg,
                           UINT cmsgsz ) ;
ER_UINT rmsgsz = tcal_por ( ID porid, RDVPTN calptn, VP msg,
                           UINT cmsgsz, TMO tmout ) ;
ER_UINT cmsgsz = acp_por ( ID porid, RDVPTN acpptn, RDVNO *p_rdvno,
                           VP msg ) ;
ER_UINT cmsgsz = pacp_por ( ID porid, RDVPTN acpptn,
                           RDVNO *p_rdvno, VP msg ) ;
```

ER_UINT cmsgsz = tacp_por (ID porid, RDVPTN acpptn,
RDVNO *p_rdvno, VP msg, TMO tmout) ;
ER ercd = fwd_por (ID porid, RDVPTN calptn, RDVNO rdvno, VP msg,
UINT cmsgsz) ;
ER ercd = rpl_rdv (RDVNO rdvno, VP msg, UINT rmsgsz) ;
ER ercd = ref_por (ID porid, T_RPOR *pk_rpor) ;
ER ercd = ref_rdv (RDVNO rdvno, T_RRDV *pk_rrdv) ;

(6) メモリプール管理機能

固定長メモリプール

ER ercd = cre_mpf (ID mpfid, T_CMPF *pk_cmpf) ;
ER_ID mpfid = acre_mpf (T_CMPF *pk_cmpf) ;
ER ercd = del_mpf (ID mpfid) ;
ER ercd = get_mpf (ID mpfid, VP *p_blk) ;
ER ercd = pget_mpf (ID mpfid, VP *p_blk) ;
ER ercd = tget_mpf (ID mpfid, VP *p_blk, TMO tmout) ;
ER ercd = rel_mpf (ID mpfid, VP blk) ;
ER ercd = ref_mpf (ID mpfid, T_RMPF *pk_rmpf) ;

可変長メモリプール

ER ercd = cre_mpl (ID mplid, T_CMPL *pk_cmpl) ;
ER_ID mplid = acre_mpl (T_CMPL *pk_cmpl) ;
ER ercd = del_mpl (ID mplid) ;
ER ercd = get_mpl (ID mplid, UINT blksz, VP *p_blk) ;
ER ercd = pget_mpl (ID mplid, UINT blksz, VP *p_blk) ;
ER ercd = tget_mpl (ID mplid, UINT blksz, VP *p_blk, TMO tmout) ;
ER ercd = rel_mpl (ID mplid, VP blk) ;
ER ercd = ref_mpl (ID mplid, T_RMPL *pk_rmpl) ;

(7) 時間管理機能

システム時刻管理

ER ercd = set_tim (SYSTIM *p_systim) ;
ER ercd = get_tim (SYSTIM *p_systim) ;
ER ercd = isig_tim ( ) ;

周期ハンドラ

ER ercd = cre_cyc (ID cycid, T_CCYC *pk_ccyc) ;
ER_ID cycid = acre_cyc (T_CCYC *pk_ccyc) ;
ER ercd = del_cyc (ID cycid) ;
ER ercd = sta_cyc (ID cycid) ;
ER ercd = stp_cyc (ID cycid) ;
ER ercd = ref_cyc (ID cycid, T_RCYC *pk_rcyc) ;

アラームハンドラ

ER ercd = cre_alm (ID almid, T_CALM *pk_calm) ;

ER_ID almid = acre_alm (T_CALM *pk_calm) ;
ER ercd = del_alm (ID almid) ;
ER ercd = sta_alm (ID almid, RELTIM almtim) ;
ER ercd = stp_alm (ID almid) ;
ER ercd = ref_alm (ID almid, T_RALM *pk_ralm) ;

オーバランハンドラ

ER ercd = def_ovr (T_DOVR *pk_dovr) ;
ER ercd = sta_ovr (ID tskid, OVRTIM ovrtim) ;
ER ercd = stp_ovr (ID tskid) ;
ER ercd = ref_ovr (ID tskid, T_ROVR *pk_rovr) ;

(8) システム状態管理機能

ER ercd = rot_rdq (PRI tskpri) ;
ER ercd = irot_rdq (PRI tskpri) ;
ER ercd = get_tid (ID *p_tskid) ;
ER ercd = iget_tid (ID *p_tskid) ;
ER ercd = loc_cpu ( ) ;
ER ercd = iloc_cpu ( ) ;
ER ercd = unl_cpu ( ) ;
ER ercd = iunl_cpu ( ) ;
ER ercd = dis_dsp ( ) ;
ER ercd = ena_dsp ( ) ;
BOOL state = sns_ctx ( ) ;
BOOL state = sns_loc ( ) ;
BOOL state = sns_dsp ( ) ;
BOOL state = sns_dpn ( ) ;
ER ercd = ref_sys (T_RSYS *pk_rsys) ;

(9) 割込み管理機能

ER ercd = def_inh (INHNO inhno, T_DINH *pk_dinh) ;
ER ercd = cre_isr (ID isrid, T_CISR *pk_cisr) ;
ER_ID isrid = acre_isr (T_CISR *pk_cisr) ;
ER ercd = del_isr (ID isrid) ;
ER ercd = ref_isr (ID isrid, T_RISR *pk_risr) ;
ER ercd = dis_int (INTNO intno) ;
ER ercd = ena_int (INTNO intno) ;
ER ercd = chg_ixx (IXXXX ixxxx) ;
ER ercd = get_ixx (IXXXX *p_ixxxx) ;

(10) サービスコール管理機能

ER ercd = def_svc (FN fncd, T_DSVC *pk_dsvc) ;
ER_UINT ercd = cal_svc (FN fncd, VP_INT par1, VP_INT par2, ...) ;

(11) システム構成管理機能

ER ercd = def_exc (EXCNO excno, T_DEXC *pk_dexc) ;
ER ercd = ref_cfg (T_RCFG *pk_rcfg) ;
ER ercd = ref_ver (T_RVER *pk_rver) ;

7.2 静的API一覧

(1) タスク管理機能

CRE_TSK (ID tskid, { ATR tskatr, VP_INT exinf, FP task, PRI itskpri,
SIZE stksz, VP stk }) ;

(3) タスク例外処理機能

DEF_TEX (ID tskid, { ATR texatr, FP texrtn }) ;

(4) 同期・通信機能

CRE_SEM (ID semid, { ATR sematr, UINT isemcnt, UINT maxsem }) ;
CRE_FLG (ID flgid, { ATR flgatr, FLGPTN iflgptn }) ;
CRE_DTQ (ID dtqid, { ATR dtqatr, UINT dtqcnt, VP dtq }) ;
CRE_MBX (ID mbxid, { ATR mbxatr, PRI maxmpri, VP mprihd }) ;

(5) 拡張同期・通信機能

CRE_MTX (ID mtxid, { ATR mtxatr, PRI ceilpri }) ;
CRE_MBF (ID mbfid, { ATR mbfatr, UINT maxmsz, SIZE mbfsz,
VP mbf }) ;
CRE_POR (ID porid, { ATR poratr, UINT maxcmsz, UINT maxrmsz }) ;

(6) メモリプール管理機能

CRE_MPF (ID mpfid, { ATR mpfatr, UINT blkcnt, UINT blksz, VP mpf }) ;
CRE_MPL (ID mplid, { ATR mplatr, SIZE mplsz, VP mpl }) ;

(7) 時間管理機能

CRE_CYC (ID cycid, { ATR cycatr, VP_INT exinf, FP cychdr,
RELTIM cyctim, RELTIM cycphs }) ;
CRE_ALM (ID almid, { ATR almatr, VP_INT exinf, FP almhdr }) ;
DEF_OVR ({ ATR ovratr, FP ovrhdr }) ;

(9) 割込み管理機能

DEF_INH (INHNO inhno, { ATR inhatr, FP inthdr }) ;
ATT_ISR ({ ATR isratr, VP_INT exinf, INTNO intno, FP isr }) ;

(10) サービスコール管理機能

DEF_SVC (FN fncd, { ATR svcatr, FP svcrtn }) ;

(11) システム構成管理機能

DEF_EXC (EXCNO excno, { ATR excatr, FP exchdr }) ;
ATT_INI ({ ATR iniatr, VP_INT exinf, FP inirtn }) ;

ITRON仕様共通静的API
INCLUDE (文字列) ;

7.3 スタンダードプロファイルの静的APIとサービスコール

(1) タスク管理機能

CRE_TSK	タスクの生成（静的API）
act_tsk／iact_tsk	タスクの起動
can_act	タスク起動要求のキャンセル
ext_tsk	自タスクの終了
ter_tsk	タスクの強制終了
chg_pri	タスク優先度の変更
get_pri	タスク優先度の参照

(2) タスク付属同期機能

slp_tsk	起床待ち
tslp_tsk	起床待ち（タイムアウトあり）
wup_tsk／iwup_tsk	タスクの起床
can_wup	タスク起床要求のキャンセル
rel_wai／irel_wai	待ち状態の強制解除
sus_tsk	強制待ち状態への移行
rsm_tsk	強制待ち状態からの再開
frsm_tsk	強制待ち状態からの強制再開
dly_tsk	自タスクの遅延

(3) タスク例外処理機能

DEF_TEX	タスク例外処理ルーチンの定義（静的API）
ras_tex／iras_tex	タスク例外処理の要求
dis_tex	タスク例外処理の禁止
ena_tex	タスク例外処理の許可
sns_tex	タスク例外処理禁止状態の参照

(4) 同期・通信機能

セマフォ

CRE_SEM	セマフォの生成（静的API）
sig_sem／isig_sem	セマフォ資源の返却
wai_sem	セマフォ資源の獲得
pol_sem	セマフォ資源の獲得（ポーリング）
twai_sem	セマフォ資源の獲得（タイムアウトあり）

イベントフラグ

CRE_FLG	イベントフラグの生成（静的API）

set_flg／iset_flg イベントフラグのセット
clr_flg イベントフラグのクリア
wai_flg イベントフラグ待ち
pol_flg イベントフラグ待ち（ポーリング）
twai_flg イベントフラグ待ち（タイムアウトあり）

データキュー

CRE_DTQ データキューの生成（静的API）
snd_dtq データキューへの送信
psnd_dtq／ipsnd_dtq データキューへの送信（ポーリング）
tsnd_dtq データキューへの送信（タイムアウトあり）
fsnd_dtq／ifsnd_dtq データキューへの強制送信
rcv_dtq データキューからの受信
prcv_dtq データキューからの受信（ポーリング）
trcv_dtq データキューからの受信（タイムアウトあり）

メールボックス

CRE_MBX メールボックスの生成（静的API）
snd_mbx メールボックスへの送信
rcv_mbx メールボックスからの受信
prcv_mbx メールボックスからの受信（ポーリング）
trcv_mbx メールボックスからの受信（タイムアウトあり）

(6) メモリプール管理機能

固定長メモリプール

CRE_MPF 固定長メモリプールの生成（静的API）
get_mpf 固定長メモリブロックの獲得
pget_mpf 固定長メモリブロックの獲得（ポーリング）
tget_mpf 固定長メモリブロックの獲得（タイムアウトあり）
rel_mpf 固定長メモリブロックの返却

(7) 時間管理機能

システム時刻管理

set_tim システム時刻の設定
get_tim システム時刻の参照
isig_tim タイムティックの供給

※システム時刻を更新する機構をカーネル内部に持つ場合には，isig_timをサポートする必要はない．

周期ハンドラ

CRE_CYC 周期ハンドラの生成（静的API）
sta_cyc 周期ハンドラの動作開始
stp_cyc 周期ハンドラの動作停止

(8) システム状態管理機能

rot_rdq／irot_rdq	タスクの優先順位の回転
get_tid／iget_tid	実行状態のタスクIDの参照
loc_cpu／iloc_cpu	CPUロック状態への移行
unl_cpu／iunl_cpu	CPUロック状態の解除
dis_dsp	ディスパッチの禁止
ena_dsp	ディスパッチの許可
sns_ctx	コンテキストの参照
sns_loc	CPUロック状態の参照
sns_dsp	ディスパッチ禁止状態の参照
sns_dpn	ディスパッチ保留状態の参照

(9) 割込み管理機能

DEF_INH	割込みハンドラの定義（静的API）

※ATT_ISR をサポートした場合には，DEF_INH をサポートする必要はない．

(11) システム構成管理機能

DEF_EXC	CPU例外ハンドラの定義（静的API）
ATT_INI	初期化ルーチンの追加（静的API）

7.4 データ型

μITRON4.0仕様で規定しているデータ型は次の通り（パケットのためのデータ型を除く）．

B	符号付き8ビット整数
H	符号付き16ビット整数
W	符号付き32ビット整数
D	符号付き64ビット整数
UB	符号無し8ビット整数
UH	符号無し16ビット整数
UW	符号無し32ビット整数
UD	符号無し64ビット整数
VB	データタイプが定まらない8ビットの値
VH	データタイプが定まらない16ビットの値
VW	データタイプが定まらない32ビットの値
VD	データタイプが定まらない64ビットの値
VP	データタイプが定まらないものへのポインタ
FP	プログラムの起動番地（ポインタ）
INT	プロセッサに自然なサイズの符号付き整数

UINT	プロセッサに自然なサイズの符号無し整数
BOOL	真偽値（TRUEまたはFALSE）
FN	機能コード（符号付き整数）
ER	エラーコード（符号付き整数）
ID	オブジェクトのID番号（符号付き整数）
ATR	オブジェクト属性（符号無し整数）
STAT	オブジェクトの状態（符号無し整数）
MODE	サービスコールの動作モード（符号無し整数）
PRI	優先度（符号付き整数）
SIZE	メモリ領域のサイズ（符号無し整数）
TMO	タイムアウト指定（符号付き整数，時間単位は実装定義）
RELTIM	相対時間（符号無し整数，時間単位は実装定義）
SYSTIM	システム時刻（符号無し整数，時間単位は実装定義）
VP_INT	データタイプが定まらないものへのポインタまたはプロセッサに自然なサイズの符号付き整数
ER_BOOL	エラーコードまたは真偽値
ER_ID	エラーコードまたはID番号（負のID番号は表現できない）
ER_UINT	エラーコードまたは符号無し整数（符号無し整数の有効ビット数はUINTより1ビット短い）
TEXPTN	タスク例外要因のビットパターン（符号無し整数）
FLGPTN	イベントフラグのビットパターン（符号無し整数）
T_MSG	メールボックスのメッセージヘッダ
T_MSG_PRI	メールボックスの優先度付きメッセージヘッダ
RDVPTN	ランデブ条件のビットパターン（符号無し整数）
RDVNO	ランデブ番号
OVRTIM	プロセッサ時間（符号無し整数，時間単位は実装定義）
INHNO	割込みハンドラ番号
INTNO	割込み番号
I*XXXX*	割込みマスク
EXCNO	CPU例外ハンドラ番号

これらの中で，次のデータ型は定義が標準化されている．

```
typedef struct t_msg_pri {
        T_MSG       msgque ;     /* メッセージヘッダ */
        PRI         msgpri ;     /* メッセージ優先度 */
} T_MSG_PRI ;
```

【スタンダードプロファイル】

スタンダードプロファイルで定義しなければならないデータ型と，それらの有効ビット数と時間単位は次の通り（パケットのためのデータ型を除く）．

B	符号付き8ビット整数
H	符号付き16ビット整数
W	符号付き32ビット整数
UB	符号無し8ビット整数
UH	符号無し16ビット整数
UW	符号無し32ビット整数
VB	データタイプが定まらない8ビットの値
VH	データタイプが定まらない16ビットの値
VW	データタイプが定まらない32ビットの値
VP	データタイプが定まらないものへのポインタ
FP	プログラムの起動番地（ポインタ）
INT	プロセッサに自然なサイズの符号付き整数（16ビット以上）
UINT	プロセッサに自然なサイズの符号無し整数（16ビット以上）
BOOL	真偽値（TRUEまたはFALSE）
FN	機能コード（符号付き整数，16ビット以上）
ER	エラーコード（符号付き整数，8ビット以上）
ID	オブジェクトのID番号（符号付き整数，16ビット以上）
ATR	オブジェクト属性（符号無し整数，8ビット以上）
STAT	オブジェクトの状態（符号無し整数，16ビット以上）
MODE	サービスコールの動作モード（符号無し整数，8 ビット以上）
PRI	優先度（符号付き整数，16ビット以上）
SIZE	メモリ領域のサイズ（符号無し整数，ポインタと同じビット数）
TMO	タイムアウト指定（符号付き整数，16ビット以上，時間単位は1ミリ秒）
RELTIM	相対時間（符号無し整数，16ビット以上，時間単位は1ミリ秒）
SYSTIM	システム時刻（符号無し整数，16ビット以上，時間単位は1ミリ秒）
VP_INT	データタイプが定まらないものへのポインタまたはプロセッサに自然なサイズの符号付き整数（VPとINTの大きい方と同じビット数）
ER_BOOL	エラーコードまたは真偽値（符号付き整数，ERとBOOLの大きい方と同じビット数）
ER_ID	エラーコードまたはID番号（符号付き整数，ERとIDの

	大きい方と同じビット数，負のID番号は表現できない）
ER_UINT	エラーコードまたは符号無し整数（符号付き整数，ERとUINTの大きい方と同じビット数，符号無し整数の有効ビット数はUINTより1ビット短い）
TEXPTN	タスク例外要因のパターン（符号無し整数，16ビット以上）
FLGPTN	イベントフラグのパターン（符号無し整数，16ビット以上）
T_MSG	メールボックスのメッセージヘッダ
T_MSG_PRI	メールボックスの優先度付きメッセージヘッダ
INHNO	割込みハンドラ番号（DEF_INHをサポートする場合）
INTNO	割込み番号（ATT_ISRをサポートする場合）
EXCNO	CPU例外ハンドラ番号

7.5 パケット形式

(1) タスク管理機能

タスク生成情報のパケット形式

```
typedef struct t_ctsk {
        ATR         tskatr ;        /* タスク属性 */
        VP_INT      exinf ;         /* タスクの拡張情報 */
        FP          task ;          /* タスクの起動番地 */
        PRI         itskpri ;       /* タスクの起動時優先度 */
        SIZE        stksz ;         /* タスクのスタックサイズ（バイト
                                       数）*/
        VP          stk ;           /* タスクのスタック領域の先頭番地 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_CTSK ;
```

タスク状態のパケット形式

```
typedef struct t_rtsk {
        STAT        tskstat ;       /* タスク状態 */
        PRI         tskpri ;        /* タスクの現在優先度 */
        PRI         tskbpri ;       /* タスクのベース優先度 */
        STAT        tskwait ;       /* 待ち要因 */
        ID          wobjid ;        /* 待ち対象のオブジェクトのID番号 */
        TMO         lefttmo ;       /* タイムアウトするまでの時間 */
        UINT        actcnt ;        /* 起動要求キューイング数 */
        UINT        wupcnt ;        /* 起床要求キューイング数 */
        UINT        suscnt ;        /* 強制待ち要求ネスト数 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RTSK ;
```

タスク状態（簡易版）のパケット形式

```
typedef struct t_rtst {
```

```
		STAT		tskstat ;	/* タスク状態 */
		STAT		tskwait ;	/* 待ち要因 */
		/* 実装独自に他のフィールドを追加してもよい */
	} T_RTST ;
```

(3) タスク例外処理機能

タスク例外処理ルーチン定義情報のパケット形式

```
	typedef struct t_dtex {
		ATR		texatr ;	/* タスク例外処理ルーチン属性 */
		FP		texrtn ;	/* タスク例外処理ルーチンの起動番
						地 */
		/* 実装独自に他のフィールドを追加してもよい */
	} T_DTEX ;
```

タスク例外処理状態のパケット形式

```
	typedef struct t_rtex {
		STAT		texstat ;	/* タスク例外処理の状態 */
		TEXPTN		pndptn ;	/* 保留例外要因 */
		/* 実装独自に他のフィールドを追加してもよい */
	} T_RTEX ;
```

(4) 同期・通信機能

セマフォ生成情報のパケット形式

```
	typedef struct t_csem {
		ATR		sematr ;	/* セマフォ属性 */
		UINT		isemcnt ;	/* セマフォの資源数の初期値 */
		UINT		maxsem ;	/* セマフォの最大資源数 */
		/* 実装独自に他のフィールドを追加してもよい */
	} T_CSEM ;
```

セマフォ状態のパケット形式

```
	typedef struct t_rsem {
		ID		wtskid ;	/* セマフォの待ち行列の先頭のタスク
						のID番号 */
		UINT		semcnt ;	/* セマフォの現在の資源数 */
		/* 実装独自に他のフィールドを追加してもよい */
	} T_RSEM ;
```

イベントフラグ生成情報のパケット形式

```
	typedef struct t_cflg {
		ATR		flgatr ;	/* イベントフラグ属性 */
		FLGPTN		iflgptn ;	/* イベントフラグのビットパターンの
						初期値 */
		/* 実装独自に他のフィールドを追加してもよい */
	} T_CFLG ;
```

イベントフラグ状態のパケット形式

```
	typedef struct t_rflg {
		ID		wtskid ;	/* イベントフラグの待ち行列の先頭の
						タスクのID番号 */
```

```
        FLGPTN    flgptn ;      /* イベントフラグの現在のビットパ
                                   ターン */
        /* 実装独自に他のフィールドを追加してもよい */
    } T_RFLG ;
```

データキュー生成情報のパケット形式

```
    typedef struct t_cdtq {
        ATR       dtqatr ;      /* データキュー属性 */
        UINT      dtqcnt ;      /* データキュー領域の容量（データの
                                   個数）*/
        VP        dtq ;         /* データキュー領域の先頭番地 */
        /* 実装独自に他のフィールドを追加してもよい */
    } T_CDTQ ;
```

データキュー状態のパケット形式

```
    typedef struct t_rdtq {
        ID        stskid ;      /* データキューの送信待ち行列の先頭
                                   のタスクのID番号 */
        ID        rtskid ;      /* データキューの受信待ち行列の先頭
                                   のタスクのID番号 */
        UINT      sdtqcnt ;     /* データキューに入っているデータの
                                   数 */
        /* 実装独自に他のフィールドを追加してもよい */
    } T_RDTQ ;
```

メールボックス生成情報のパケット形式

```
    typedef struct t_cmbx {
        ATR       mbxatr ;      /* メールボックス属性 */
        PRI       maxmpri ;     /* 送信されるメッセージの優先度の最
                                   大値 */
        VP        mprihd ;      /* 優先度別のメッセージキューヘッダ
                                   領域の先頭番地 */
        /* 実装独自に他のフィールドを追加してもよい */
    } T_CMBX ;
```

メールボックス状態のパケット形式

```
    typedef struct t_rmbx {
        ID        wtskid ;      /* メールボックスの待ち行列の先頭の
                                   タスクのID番号 */
        T_MSG *   pk_msg ;      /* メッセージキューの先頭のメッセー
                                   ジパケットの先頭番地 */
        /* 実装独自に他のフィールドを追加してもよい */
    } T_RMBX ;
```

(5) 拡張同期・通信機能

ミューテックス生成情報のパケット形式

```
    typedef struct t_cmtx {
        ATR       mtxatr ;      /* ミューテックス属性 */
        PRI       ceilpri ;     /* ミューテックスの上限優先度 */
        /* 実装独自に他のフィールドを追加してもよい */
```

```
} T_CMTX ;
```

ミューテックス状態のパケット形式

```
typedef struct t_rmtx {
        ID          htskid ;     /* ミューテックスをロックしているタ
                                    スクのID番号 */
        ID          wtskid ;     /* ミューテックスの待ち行列の先頭の
                                    タスクのID番号 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RMTX ;
```

メッセージバッファ生成情報のパケット形式

```
typedef struct t_cmbf {
        ATR         mbfatr ;     /* メッセージバッファ属性 */
        UINT        maxmsz ;     /* メッセージの最大サイズ（バイト
                                    数）*/
        SIZE        mbfsz ;      /* メッセージバッファ領域のサイズ
                                    （バイト数）*/
        VP          mbf ;        /* メッセージバッファ領域の先頭番
                                    地 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_CMBF ;
```

メッセージバッファ状態のパケット形式

```
typedef struct t_rmbf {
        ID          stskid ;     /* メッセージバッファの送信待ち行列
                                    の先頭のタスクのID番号 */
        ID          rtskid ;     /* メッセージバッファの受信待ち行列
                                    の先頭のタスクのID番号 */
        UINT        smsgcnt ;    /* メッセージバッファに入っている
                                    メッセージの数 */
        SIZE        fmbfsz ;     /* メッセージバッファ領域の空き領域
                                    のサイズ（バイト数，最低限の管理
                                    領域を除く）*/
        /* 実装独自に他のフィールドを追加してもよい */
} T_RMBF ;
```

ランデブポート生成情報のパケット形式

```
typedef struct t_cpor {
        ATR         poratr ;     /* ランデブポート属性 */
        UINT        maxcmsz ;    /* 呼出しメッセージの最大サイズ（バ
                                    イト数）*/
        UINT        maxrmsz ;    /* 返答メッセージの最大サイズ（バイ
                                    ト数）*/
        /* 実装独自に他のフィールドを追加してもよい */
} T_CPOR ;
```

ランデブポート状態のパケット形式

```
typedef struct t_rpor {
        ID          ctskid ;     /* ランデブポートの呼出し待ち行列の
                                    先頭のタスクのID番号 */
```

```
        ID          atskid ;     /* ランデブポートの受付待ち行列の先
                                    頭のタスクのID番号 */
        /* 実装独自に他のフィールドを追加してもよい */
    } T_RPOR ;
```

ランデブ状態のパケット形式

```
    typedef struct t_rrdv {
        ID          wtskid ;     /* ランデブ終了待ち状態のタスクのID
                                    番号 */
        /* 実装独自に他のフィールドを追加してもよい */
    } T_RRDV ;
```

(6) メモリプール管理機能

固定長メモリプール生成情報のパケット形式

```
    typedef struct t_cmpf {
        ATR         mpfatr ;     /* 固定長メモリプール属性 */
        UINT        blkcnt ;     /* 獲得できるメモリブロック数（個
                                    数）*/
        UINT        blksz ;      /* メモリブロックのサイズ（バイト
                                    数）*/
        VP          mpf ;        /* 固定長メモリプール領域の先頭番
                                    地 */
        /* 実装独自に他のフィールドを追加してもよい */
    } T_CMPF ;
```

固定長メモリプール状態のパケット形式

```
    typedef struct t_rmpf {
        ID          wtskid ;     /* 固定長メモリプールの待ち行列の先
                                    頭のタスクのID番号 */
        UINT        fblkcnt ;    /* 固定長メモリプールの空きメモリブ
                                    ロック数（個数）*/
        /* 実装独自に他のフィールドを追加してもよい */
    } T_RMPF ;
```

可変長メモリプール生成情報のパケット形式

```
    typedef struct t_cmpl {
        ATR         mplatr ;     /* 可変長メモリプール属性 */
        SIZE        mplsz ;      /* 可変長メモリプール領域のサイズ
                                    （バイト数）*/
        VP          mpl ;        /* 可変長メモリプール領域の先頭番
                                    地 */
        /* 実装独自に他のフィールドを追加してもよい */
    } T_CMPL ;
```

可変長メモリプール状態のパケット形式

```
    typedef struct t_rmpl {
        ID          wtskid ;     /* 可変長メモリプールの待ち行列の先
                                    頭のタスクのID番号 */
        SIZE        fmplsz ;     /* 可変長メモリプールの空き領域の合
                                    計サイズ（バイト数）*/
```

```
		UINT		fblksz ;	/* すぐに獲得可能な最大メモリブロッ
						クサイズ（バイト数）*/
		/* 実装独自に他のフィールドを追加してもよい */
	} T_RMPL ;
```

(7) 時間管理機能

周期ハンドラ生成情報のパケット形式

```
	typedef struct t_ccyc {
		ATR		cycatr ;	/* 周期ハンドラ属性 */
		VP_INT		exinf ;		/* 周期ハンドラの拡張情報 */
		FP		cychdr ;	/* 周期ハンドラの起動番地 */
		RELTIM		cyctim ;	/* 周期ハンドラの起動周期 */
		RELTIM		cycphs ;	/* 周期ハンドラの起動位相 */
		/* 実装独自に他のフィールドを追加してもよい */
	} T_CCYC ;
```

周期ハンドラ状態のパケット形式

```
	typedef struct t_rcyc {
		STAT		cycstat ;	/* 周期ハンドラの動作状態 */
		RELTIM		lefttim ;	/* 周期ハンドラを次に起動すべき時刻
						までの時間 */
		/* 実装独自に他のフィールドを追加してもよい */
	} T_RCYC ;
```

アラームハンドラ生成情報のパケット形式

```
	typedef struct t_calm {
		ATR		almatr ;	/* アラームハンドラ属性 */
		VP_INT		exinf ;		/* アラームハンドラの拡張情報 */
		FP		almhdr ;	/* アラームハンドラの起動番地 */
		/* 実装独自に他のフィールドを追加してもよい */
	} T_CALM ;
```

アラームハンドラ状態のパケット形式

```
	typedef struct t_ralm {
		STAT		almstat ;	/* アラームハンドラの動作状態 */
		RELTIM		lefttim ;	/* アラームハンドラの起動時刻までの
						時間 */
		/* 実装独自に他のフィールドを追加してもよい */
	} T_RALM ;
```

オーバランハンドラ定義情報のパケット形式

```
	typedef struct t_dovr {
		ATR		ovratr ;	/* オーバランハンドラ属性 */
		FP		ovrhdr ;	/* オーバランハンドラの起動番地 */
		/* 実装独自に他のフィールドを追加してもよい */
	} T_DOVR ;
```

オーバランハンドラ状態のパケット形式

```
	typedef struct t_rovr {
		STAT		ovrstat ;	/* オーバランハンドラの動作状態 */
```

```
		OVRTIM	leftotm ;	/* 残りのプロセッサ時間 */
		/* 実装独自に他のフィールドを追加してもよい */
	} T_ROVR ;
```

(8) システム状態管理機能

システム状態のパケット形式

```
	typedef struct t_rsys {
		/* 実装独自のフィールドが入る */
	} T_RSYS ;
```

(9) 割込み管理機能

割込みハンドラ定義情報のパケット形式

```
	typedef struct t_dinh {
		ATR		inhatr ;	/* 割込みハンドラ属性 */
		FP		inthdr ;	/* 割込みハンドラの起動番地 */
		/* 実装独自に他のフィールドを追加してもよい */
	} T_DINH ;
```

割込みサービスルーチン生成情報のパケット形式

```
	typedef struct t_cisr {
		ATR		isratr ;	/* 割込みサービスルーチン属性 */
		VP_INT	exinf ;		/* 割込みサービスルーチンの拡張情
						報 */
		INTNO	intno ;		/* 割込みサービスルーチンを付加する
						割込み番号 */
		FP		isr ;		/* 割込みサービスルーチンの起動番
						地 */
		/* 実装独自に他のフィールドを追加してもよい */
	} T_CISR ;
```

割込みサービスルーチン状態のパケット形式

```
	typedef struct t_risr {
		/* 実装独自のフィールドが入る */
	} T_RISR ;
```

(10) サービスコール管理機能

拡張サービスコール定義情報のパケット形式

```
	typedef struct t_dsvc {
		ATR		svcatr ;	/* 拡張サービスコール属性 */
		FP		svcrtn ;	/* 拡張サービスコールルーチンの起動
						番地 */
		/* 実装独自に他のフィールドを追加してもよい */
	} T_DSVC ;
```

(11) システム構成管理機能

CPU例外ハンドラ定義情報のパケット形式

```
	typedef struct t_dexc {
		ATR		excatr ;	/* CPU例外ハンドラ属性 */
```

```
		FP		exchdr ;	/* CPU例外ハンドラの起動番地 */
		/* 実装独自に他のフィールドを追加してもよい */
	} T_DEXC ;
```

コンフィギュレーション情報のパケット形式

```
	typedef struct t_rcfg {
		/* 実装独自のフィールドが入る */
	} T_RCFG ;
```

バージョン情報のパケット形式

```
	typedef struct t_rver {
		UH		maker ;		/* カーネルのメーカコード */
		UH		prid ;		/* カーネルの識別番号 */
		UH		spver ;		/* ITRON仕様のバージョン番号 */
		UH		prver ;		/* カーネルのバージョン番号 */
		UH		prno[4] ;	/* カーネル製品の管理情報 */
	} T_RVER ;
```

7.6 定数とマクロ

(1) 一般

NULL	0	無効ポインタ
TRUE	1	真
FALSE	0	偽
E_OK	0	正常終了

(2) オブジェクト属性

TA_NULL	0	オブジェクト属性を指定しない
TA_HLNG	0x00	高級言語用のインタフェースで処理単位を起動
TA_ASM	0x01	アセンブリ言語用のインタフェースで処理単位を起動
TA_TFIFO	0x00	タスクの待ち行列をFIFO順に
TA_TPRI	0x01	タスクの待ち行列をタスクの優先度順に
TA_MFIFO	0x00	メッセージのキューをFIFO順に
TA_MPRI	0x02	メッセージのキューをメッセージの優先度順に
TA_ACT	0x02	タスクを起動された状態で生成
TA_RSTR	0x04	制約タスク
TA_WSGL	0x00	イベントフラグを複数のタスクが待つことを許さない
TA_WMUL	0x02	イベントフラグを複数のタスクが待つことを許

		す
TA_CLR	0x04	待ち解除時にイベントフラグをクリア
TA_INHERIT	0x02	ミューテックスが優先度継承プロトコルをサポート
TA_CEILING	0x03	ミューテックスが優先度上限プロトコルをサポート
TA_STA	0x02	周期ハンドラを動作している状態で生成
TA_PHS	0x04	周期ハンドラの位相を保存

(3) タイムアウト指定

TMO_POL	0	ポーリング
TMO_FEVR	–1	永久待ち
TMO_NBLK	–2	ノンブロッキング

(4) サービスコールの動作モード

TWF_ANDW	0x00	イベントフラグのAND待ち
TWF_ORW	0x01	イベントフラグのOR待ち

(5) オブジェクトの状態

TTS_RUN	0x01	実行状態
TTS_RDY	0x02	実行可能状態
TTS_WAI	0x04	待ち状態
TTS_SUS	0x08	強制待ち状態
TTS_WAS	0x0c	二重待ち状態
TTS_DMT	0x10	休止状態

TTW_SLP	0x0001	起床待ち状態
TTW_DLY	0x0002	時間経過待ち状態
TTW_SEM	0x0004	セマフォ資源の獲得待ち状態
TTW_FLG	0x0008	イベントフラグ待ち状態
TTW_SDTQ	0x0010	データキューへの送信待ち状態
TTW_RDTQ	0x0020	データキューからの受信待ち状態
TTW_MBX	0x0040	メールボックスからの受信待ち状態
TTW_MTX	0x0080	ミューテックスのロック待ち状態
TTW_SMBF	0x0100	メッセージバッファへの送信待ち状態
TTW_RMBF	0x0200	メッセージバッファからの受信待ち状態
TTW_CAL	0x0400	ランデブの呼出し待ち状態
TTW_ACP	0x0800	ランデブの受付待ち状態
TTW_RDV	0x1000	ランデブの終了待ち状態
TTW_MPF	0x2000	固定長メモリブロックの獲得待ち状態
TTW_MPL	0x4000	可変長メモリブロックの獲得待ち状態

TTEX_ENA	0x00	タスク例外処理許可状態
TTEX_DIS	0x01	タスク例外処理禁止状態
TCYC_STP	0x00	周期ハンドラが動作していない
TCYC_STA	0x01	周期ハンドラが動作している
TALM_STP	0x00	アラームハンドラが動作していない
TALM_STA	0x01	アラームハンドラが動作している
TOVR_STP	0x00	上限プロセッサ時間が設定されていない
TOVR_STA	0x01	上限プロセッサ時間が設定されている

(6) その他の定数

TSK_SELF	0	自タスク指定
TSK_NONE	0	該当するタスクが無い
TPRI_SELF	0	自タスクのベース優先度の指定
TPRI_INI	0	タスクの起動時優先度の指定

(7) マクロ

ER ercd = ERCD (ER mercd, ER sercd)

メインエラーコードとサブエラーコードからエラーコードを生成する.

ER mercd = MERCD (ER ercd)

エラーコードからメインエラーコードを取り出す.

ER sercd = SERCD (ER ercd)

エラーコードからサブエラーコードを取り出す.

7.7 構成定数とマクロ

(1) 優先度の範囲

TMIN_TPRI	タスク優先度の最小値（＝1）
TMAX_TPRI	タスク優先度の最大値
TMIN_MPRI	メッセージ優先度の最小値（＝1）
TMAX_MPRI	メッセージ優先度の最大値

(2) バージョン情報

TKERNEL_MAKER	カーネルのメーカコード
TKERNEL_PRID	カーネルの識別番号
TKERNEL_SPVER	ITRON仕様のバージョン番号
TKERNEL_PRVER	カーネルのバージョン番号

(3) キューイング／ネスト回数の最大値

TMAX_ACTCNT　タスクの起動要求キューイング数の最大値
TMAX_WUPCNT　タスクの起床要求キューイング数の最大値
TMAX_SUSCNT　タスクの強制待ち要求ネスト数の最大値

(4) ビットパターンのビット数

TBIT_TEXPTN　タスク例外要因のビット数
TBIT_FLGPTN　イベントフラグのビット数
TBIT_RDVPTN　ランデブ条件のビット数

(5) タイムティックの周期

TIC_NUME　タイムティックの周期の分子
TIC_DENO　タイムティックの周期の分母

(6) 必要なメモリ領域のサイズ

SIZE dtqsz = TSZ_DTQ (UINT dtqcnt)

dtqcnt 個のデータを格納するのに必要なデータキュー領域のサイズ（バイト数）

SIZE mprihdsz = TSZ_MPRIHD (PRI maxmpri)

送信されるメッセージの優先度の最大値がmaxmpriのメールボックスに必要な優先度別メッセージキューヘッダ領域のサイズ（バイト数）

SIZE mbfsz = TSZ_MBF (UINT msgcnt, UINT msgsz)

サイズがmsgszバイトのメッセージをmsgcnt個バッファリングするのに必要なメッセージバッファ領域のサイズ（目安のバイト数）

SIZE mpfsz = TSZ_MPF (UINT blkcnt, UINT blksz)

サイズがblkszバイトのメモリブロックをblkcnt個獲得できるのに必要な固定長メモリプール領域のサイズ（バイト数）

SIZE mplsz = TSZ_MPL (UINT blkcnt, UINT blksz)

サイズがblkszバイトのメモリブロックをblkcnt個獲得できるのに必要な可変長メモリプール領域のサイズ（目安のバイト数）

(7) その他

TMAX_MAXSEM　セマフォの最大資源数の最大値

7.8 エラーコード一覧

E_SYS　–5　システムエラー

E_NOSPT	–9	未サポート機能
E_RSFN	–10	予約機能コード
E_RSATR	–11	予約属性
E_PAR	–17	パラメータエラー
E_ID	–18	不正ID番号
E_CTX	–25	コンテキストエラー
E_MACV	–26	メモリアクセス違反
E_OACV	–27	オブジェクトアクセス違反
E_ILUSE	–28	サービスコール不正使用
E_NOMEM	–33	メモリ不足
E_NOID	–34	ID番号不足
E_OBJ	–41	オブジェクト状態エラー
E_NOEXS	–42	オブジェクト未生成
E_QOVR	–43	キューイングオーバフロー
E_RLWAI	–49	待ち状態の強制解除
E_TMOUT	–50	ポーリング失敗またはタイムアウト
E_DLT	–51	待ちオブジェクトの削除
E_CLS	–52	待ちオブジェクトの状態変化
E_WBLK	–57	ノンブロッキング受付け
E_BOVR	–58	バッファオーバフロー

7.9 機能コード一覧

	–0	–1	–2	–3
–0x01	予約	予約	予約	予約
–0x05	cre_tsk	del_tsk	act_tsk	can_act
–0x09	sta_tsk	ext_tsk	exd_tsk	ter_tsk
–0x0d	chg_pri	get_pri	ref_tsk	ref_tst
–0x11	slp_tsk	tslp_tsk	wup_tsk	can_wup
–0x15	rel_wai	sus_tsk	rsm_tsk	frsm_tsk
–0x19	dly_tsk	予約	def_tex	ras_tex
–0x1d	dis_tex	ena_tex	sns_tex	ref_tex
–0x21	cre_sem	del_sem	sig_sem	予約
–0x25	wai_sem	pol_sem	twai_sem	ref_sem
–0x29	cre_flg	del_flg	set_flg	clr_flg
–0x2d	wai_flg	pol_flg	twai_flg	ref_flg
–0x31	cre_dtq	del_dtq	予約	予約
–0x35	snd_dtq	psnd_dtq	tsnd_dtq	fsnd_dtq
–0x39	rcv_dtq	prcv_dtq	trcv_dtq	ref_dtq
–0x3d	cre_mbx	del_mbx	snd_mbx	予約
–0x41	rcv_mbx	prcv_mbx	trcv_mbx	ref_mbx
–0x45	cre_mpf	del_mpf	rel_mpf	予約
–0x49	get_mpf	pget_mpf	tget_mpf	ref_mpf
–0x4d	set_tim	get_tim	cre_cyc	del_cyc
–0x51	sta_cyc	stp_cyc	ref_cyc	予約
–0x55	rot_rdq	get_tid	予約	予約
–0x59	loc_cpu	unl_cpu	dis_dsp	ena_dsp
–0x5d	sns_ctx	sns_loc	sns_dsp	sns_dpn
–0x61	ref_sys	予約	予約	予約
–0x65	def_inh	cre_isr	del_isr	ref_isr
–0x69	dis_int	ena_int	chg_*ixx*	get_*ixx*
–0x6d	def_svc	def_exc	ref_cfg	ref_ver
–0x71	iact_tsk	iwup_tsk	irel_wai	iras_tex
–0x75	isig_sem	iset_flg	ipsnd_dtq	ifsnd_dtq
–0x79	irot_rdq	iget_tid	iloc_cpu	iunl_cpu
–0x7d	isig_tim	予約	予約	予約
–0x81	cre_mtx	del_mtx	unl_mtx	予約
–0x85	loc_mtx	ploc_mtx	tloc_mtx	ref_mtx
–0x89	cre_mbf	del_mbf	予約	予約
–0x8d	snd_mbf	psnd_mbf	tsnd_mbf	予約

–0x91	rcv_mbf	prcv_mbf	trcv_mbf	ref_mbf
–0x95	cre_por	del_por	cal_por	tcal_por
–0x99	acp_por	pacp_por	tacp_por	fwd_por
–0x9d	rpl_rdv	ref_por	ref_rdv	予約
–0xa1	cre_mpl	del_mpl	rel_mpl	予約
–0xa5	get_mpl	pget_mpl	tget_mpl	ref_mpl
–0xa9	cre_alm	del_alm	sta_alm	stp_alm
–0xad	ref_alm	予約	予約	予約
–0xb1	def_ovr	sta_ovr	stp_ovr	ref_ovr
–0xb5	予約	予約	予約	予約
–0xb9	予約	予約	予約	予約
–0xbd	予約	予約	予約	予約
–0xc1	acre_tsk	acre_sem	acre_flg	acre_dtq
–0xc5	acre_mbx	acre_mtx	acre_mbf	acre_por
–0xc9	acre_mpf	acre_mpl	acre_cyc	acre_alm
–0xcd	acre_isr	予約	予約	予約
–0xd1	予約	予約	予約	予約
–0xd5	予約	予約	予約	予約
–0xd9	予約	予約	予約	予約
–0xdd	予約	予約	予約	予約
–0xe1	実装独自のサービスコール			
–0xe5	実装独自のサービスコール			
–0xe9	実装独自のサービスコール			
–0xed	実装独自のサービスコール			
–0xf1	実装独自のサービスコール			
–0xf5	実装独自のサービスコール			
–0xf9	実装独自のサービスコール			
–0xfd	実装独自のサービスコール			

7.10 実装毎に規定すべき事項（実装定義）一覧表

表7-1. 実装毎に規定すべき事項（実装定義）一覧表

番号	章・節，ページ	内容
1	2.1.2　APIの構成要素　p.23	データ型 ITRON仕様でビット数が規定されていないデータ型に対して，C言語の型のビット数よりも少ない有効ビット数ないしは型で表現できる範囲よりも狭い有効範囲を，実装定義に定めることができる．
2	2.1.6　サービスコールの返値とエラーコード　p.26	サブエラーコードは実装定義とする．
3	2.1.11　静的APIの文法とパラメータ　p.35	静的APIの処理において検出すべきエラーと，エラーを検出した場合の扱いについては，実装定義とする．
4	2.3.1　ITRON仕様共通データ型　p.42	TMO　タイムアウト指定（符号付き整数，時間単位は実装定義）
5	2.3.1　ITRON仕様共通データ型　p.42	RELTIM　相対時間（符号無し整数，時間単位は実装定義）
6	2.3.1　ITRON仕様共通データ型　p.42	SYSTIM　システム時刻（符号無し整数，時間単位は実装定義）
7	2.3.1　ITRON仕様共通データ型　p.42	SYSTIMは，システム時刻の表現に必要なビット数が整数型として扱えるビット数よりも多い場合には，構造体として定義してもよい．構造体として定義する場合の構造体の内容については，実装定義とする．
8	2.3.2　ITRON仕様共通定数　p.44	未サポートエラークラスに属するエラーの検出は，実装定義で省略することができる．
9	2.3.2　ITRON仕様共通定数　p.44	パラメータエラークラスに属するエラーの検出は，実装定義で省略することができる．
10	2.3.2　ITRON仕様共通定数　p.44	呼出しコンテキストエラークラスに属するエラーの検出は，実装定義で省略することができる．
11	3.3.1　割込みハンドラと割込みサービスルーチン　p.55	割込みハンドラの記述方法は，一般にはプロセッサの割込みアーキテクチャや用いるIRCなどに依存するため，実装定義とする．
12	3.3.1　割込みハンドラと割込みサービスルーチン　p.56	割込みハンドラを登録するためのAPIと割込みサービスルーチンを登録するためのAPIの両方のAPIを併用した場合の振舞いは実装定義とする．
13	3.3.1　割込みハンドラと割込みサービスルーチン　p.56	カーネルは，ある優先度よりも高い優先度を持つ割込みを管理しないものとすることができる．どの優先度より高い優先度を持つものをカーネルの管理外の割込みとするかは，実装定義である．

表7-1. 実装毎に規定すべき事項（実装定義）一覧表

14	3.3.1　割込みハンドラと割込みサービスルーチン　p.57	割込みハンドラの出入口処理で必要な処理として，割込みハンドラ内で使用するレジスタの保存と復帰，スタックの切替え，タスクディスパッチ処理，プロセッサレベルでの割込みハンドラからの復帰が挙げられる．この中で実際にどの処理が必要かは，場合によって異なる．また，これらの処理の中で，どれをカーネルが用意する出入口処理で行い，どれをアプリケーションが登録する割込みハンドラで行わなければならないとするかは，実装定義である．
15	3.4.2　CPU 例外ハンドラで行える操作　p.59	CPU 例外ハンドラの記述方法は，一般にはプロセッサの CPU 例外アーキテクチャやカーネルの実装などに依存するため，実装定義とする．
16	3.4.2　CPU 例外ハンドラで行える操作　p.59	CPU 例外ハンドラ内で呼出し可能なサービスコールは実装定義である．
17	3.4.2　CPU 例外ハンドラで行える操作　p.59	CPU例外ハンドラ内で次の操作を行う方法を用意しなければならない．これらの操作を行う方法も，実装定義である． (a)CPU 例外が発生したコンテキストや状態の読出し．具体的には，CPU 例外が発生した処理で sns_*yyy* を呼び出した場合の結果を，CPU 例外ハンドラ内で取り出せること．
18	3.4.2　CPU 例外ハンドラで行える操作　p.59	CPU例外ハンドラ内で次の操作を行う方法を用意しなければならない．これらの操作を行う方法も，実装定義である．… (b)CPU 例外が発生したタスクの ID 番号の読出し（例外を発生させたのがタスクである場合のみ）．
19	3.4.2　CPU 例外ハンドラで行える操作　p.59	CPU例外ハンドラ内で次の操作を行う方法を用意しなければならない．これらの操作を行う方法も，実装定義である．… (c) タスク例外処理の要求．具体的には，CPU 例外ハンドラ内で，ras_tex と同等の操作ができること．
20	3.5.2　タスクコンテキストと非タスクコンテキスト p.61	タスクコンテキストを実行中に CPU 例外が発生した場合には，CPU 例外ハンドラが実行されるコンテキストがタスクコンテキストと非タスクコンテキストのいずれに分類されるかは実装定義とする．
21	3.5.3　処理の優先順位とサービスコールの不可分性 p.62	割込みハンドラの優先順位は，ディスパッチャの優先順位よりも高い．割込みハンドラおよび割込みサービスルーチン相互間の優先順位は，それらを起動する外部割込みの優先度に対応して定めることを基本に，実装定義で定める．
22	3.5.3　処理の優先順位とサービスコールの不可分性 p.62	タイムイベントハンドラ（オーバランハンドラを除く）の優先順位は，isig_tim を呼び出した割込みハンドラの優先順位以下で，ディスパッチャの優先順位よりも高いという範囲内で，実装定義で定める．

表7-1. 実装毎に規定すべき事項（実装定義）一覧表

23	3.5.3　処理の優先順位とサービスコールの不可分性 p.62	オーバランハンドラの優先順位は，ディスパッチャの優先順位よりも高いという範囲内で，実装定義で定める．
24	3.5.3　処理の優先順位とサービスコールの不可分性 p.62	割込みハンドラやタイムイベントハンドラとの間の優先順位は，実装定義である．
25	3.5.4　CPU ロック状態 p.63	割込みハンドラ内で CPU ロック解除状態にするための方法を，実装定義で用意することが必要である．
26	3.5.4　CPU ロック状態 p.63	CPU ロック解除状態にした後に，割込みハンドラから正しくリターンするための方法も実装定義とする．
27	3.6.3　呼び出せるサービスコールの追加　p.71	E_CTX エラーの検出は，実装定義で省略することができる．
28	3.7 システム初期化手順 p.72	ハードウェア依存の初期化処理の最後で，カーネルの初期化処理を呼び出す．カーネルの初期化処理を呼び出す方法は，実装定義である．
29	3.7 システム初期化手順 p.72	静的 API の処理中にエラーを検出した場合の扱いについては，実装定義とする．
30	3.7 システム初期化手順 p.72	初期化ルーチンは，カーネルの管理外の割込みを除くすべての割込みを禁止した状態で実行する．カーネルの管理外の割込みを禁止するかどうかは，実装定義である．
31	3.7 システム初期化手順 p.72	初期化ルーチン内でサービスコールを呼び出せるか否か，呼び出せる場合にはどのようなサービスコールが呼び出せるかも，実装定義である．
32	3.8　オブジェクトの登録とその解除　p.72	カーネルに登録できるオブジェクトの最大数や ID 番号の範囲は，実装定義である．
33	3.8　オブジェクトの登録とその解除　p.72	サービスコールを用いて生成できるオブジェクトの最大数や ID 番号の範囲の指定方法も，実装定義である．
34	3.8　オブジェクトの登録とその解除　p.73	自動割付けされる ID 番号の範囲の指定方法は，実装定義である．
35	3.9 処理単位の記述形式 p.73	割込みサービスルーチン，タイムイベントハンドラ（周期ハンドラ，アラームハンドラ，オーバランハンドラ），拡張サービスコールルーチン，タスク，タスク例外処理ルーチンの各処理単位をアセンブリ言語で記述する方法については，実装定義とする．
36	3.9 処理単位の記述形式 p.73	割込みハンドラおよび CPU 例外ハンドラを記述する方法と，これらをカーネルに登録する場合のオブジェクト属性の指定方法については実装定義とする
37	CRE_TSK/cre_tsk/acre_tsk p.84	stkszに実装定義の最大値よりも大きい値が指定された場合には，E_PAR エラーを返す．

表7-1. 実装毎に規定すべき事項（実装定義）一覧表

38	act_tsk　p.87	非タスクコンテキストから呼び出された場合で，サービスコールを遅延実行する場合には，E_QOVRエラーを返すことを，実装定義で省略することができる．
39	ext_tsk　p.90	サービスコール内のエラーを検出するか否かと，もし検出した場合の振る舞いは実装定義とする．
40	exd_tsk　p.91	サービスコール内のエラーを検出するか否かと，もし検出した場合の振る舞いは実装定義とする．
41	wup_tsk/iwup_tsk　p.104	非タスクコンテキストから呼び出された場合で，サービスコールを遅延実行する場合には，E_OBJエラーとE_QOVRエラーを返すことを，実装定義で省略することができる．
42	rel_wai/irel_wai　p.107	非タスクコンテキストから呼び出された場合で，サービスコールを遅延実行する場合には，E_OBJエラーを返すことを，実装定義で省略することができる．
43	4.3 タスク例外処理機能 p.113	実装定義で，拡張サービスコールルーチンの起動によりタスク例外処理を禁止し，そこからのリターンにより起動前の状態に戻すことができる
44	ras_tex/iras_tex　p.120	非タスクコンテキストから呼び出された場合で，サービスコールを遅延実行する場合には，E_OBJエラーを返すことを，実装定義で省略することができる．
45	sig_sem/isig_sem　p.131	非タスクコンテキストから呼び出された場合で，サービスコールを遅延実行する場合には，E_QOVRエラーを返すことを，実装定義で省略することができる．
46	CRE_DTQ/cre_dtq/acre_dtq p.151	dtqcntに実装定義の最大値よりも大きい値が指定された場合には，E_PARエラーを返す．
47	snd_dtq/psnd_dtq/ipsnd_dtq/tsnd_dtq　p.154	非タスクコンテキストから呼び出された場合で，サービスコールを遅延実行する場合には，E_TMOUTエラーを返すことを，実装定義で省略することができる．
48	4.4.4　メールボックス p.160	T_MSG型(メールボックスのメッセージヘッダ)の定義やサイズは，実装定義である．
49	4.5.1　ミューテックス p.171	厳密な優先度制御規則に加えて，現在優先度を変更する状況を限定した優先度制御規則（これを簡略化した優先度制御規則と呼ぶ）を規定し，どちらを採用するかは実装定義とする．
50	CRE_MBF/cre_mbf/acre_mbf p.185	maxmszに0が指定された場合や,実装定義の最大値よりも大きい値が指定された場合には，E_PARエラーを返す．
51	CRE_MBF/cre_mbf/acre_mbf p.185	また，mbfszに実装定義の最大値よりも大きい値が指定された場合にも，E_PARエラーを返す．
52	CRE_POR/cre_por/acre_por p.199	maxcmszまたはmaxrmszに，実装定義の最大値よりも大きい値が指定された場合には，E_PARエラーを返す．

表7-1. 実装毎に規定すべき事項（実装定義）一覧表

53	CRE_MPF/cre_mpf/acre_mpf p.217	blkcnt または blksz に 0 が指定された場合や，実装定義の最大値よりも大きい値が指定された場合には，E_PAR エラーを返す．
54	CRE_MPL/cre_mpl/acre_mpl p.227	mplsz に 0 が指定された場合や, 実装定義の最大値よりも大きい値が指定された場合には，E_PAR エラーを返す．
55	4.7.1　システム時刻管理 p.235	isig_tim が呼び出される度にシステム時刻をどれだけ更新するか，言い換えるとアプリケーションが isig_tim を呼び出すべき周期は，実装定義で定める．
56	4.7.4　オーバランハンドラ p.258	OVRTIM　プロセッサ時間（符号無し整数，時間単位は実装定義
57	ref_sys　p.277	システムの状態を参照し，pk_rsys で指定されるパケットに返す．具体的にどのような情報が参照できるかは，実装定義である．
58	4.9 割込み管理機能　p.278	割込みマスクのデータ型の一部の *XXXX* は，実装定義で，ターゲットプロセッサのアーキテクチャに応じた適切な文字列に定める．
59	4.9 割込み管理機能　p.278	割込みハンドラの記述形式は，実装定義である．
60	DEF_INH/def_inh　p.281	inhno の具体的な意味は実装定義であるが，一般的な実装ではプロセッサの割込みベクトル番号に対応する．
61	DEF_INH/def_inh　p.281	inhatr に指定できる値とその意味は実装定義である．
62	ref_isr　p.286	isrid で指定される割込みサービスルーチンの状態を参照し，pk_risr で指定されるパケットに返す．具体的にどのような情報が参照できるかは，実装定義である．
63	dis_int　p.287	intno で指定される割込みを禁止する．intno の具体的な意味は実装定義であるが，一般的な実装では IRC への割込み要求入力ラインに対応する．
64	ena_int　p.288	intno で指定される割込みを許可する．intno の具体的な意味は実装定義であるが，一般的な実装では IRC への割込み要求入力ラインに対応する．
65	chg_ixx　p.289	サービスコールの名称およびパラメータの名称の一部の *xx* および *xxxx* は，実装定義で，ターゲットプロセッサのアーキテクチャに応じた適切な文字列に定める．
66	chg_ixx　p.289	i*xxxx* の値によっては，このサービスコールを呼び出したことで，CPU ロック状態と CPU ロック解除状態の間，ディスパッチ禁止状態とディスパッチ許可状態の間の移行を行うことがある．どのような値でどのような移行を行うかは，実装定義である．
67	get_ixx　p.289	サービスコールの名称およびパラメータの名称の一部の *xx* および *xxxx* は，実装定義で，ターゲットプロセッサのアーキテクチャに応じた適切な文字列に定める．

表7-1. 実装毎に規定すべき事項（実装定義）一覧表

68	4.10　サービスコール管理機能 p.291	拡張サービスコールルーチンが受け取るパラメータ（par1，par2，...）は，必要な数だけ記述すればよい．実装定義で，受け取れるパラメータの数に上限が設けられている場合がある．
69	4.10　サービスコール管理機能 p.292	拡張サービスコールルーチンは，実装定義で，タスク例外処理を禁止して実行する場合がある．
70	cal_svc　p.295	実装定義で，サービスコールに渡せるパラメータの数に，1以上の上限を設けることができる．
71	cal_svc　p.295	このサービスコールで，拡張サービスコール以外の標準のサービスコールを呼び出せるかどうかは実装定義である．
72	4.11 システム構成管理機能 p.296	CPU例外ハンドラの記述形式は，実装定義である．
73	4.11 システム構成管理機能 p.297	CPU例外ハンドラ内で呼出し可能なサービスコールは実装定義である．
74	4.11 システム構成管理機能 p.297	CPU例外が発生したコンテキストがタスクコンテキストであればタスクコンテキストと非タスクコンテキストのいずれに分類されるか実装定義
75	DEF_EXC/def_exc p.298	excnoの具体的な意味は実装定義であるが，一般的な実装では，プロセッサで区別される例外の種類に対応する．
76	DEF_EXC/def_exc　p.298	excatrに指定できる値とその意味は実装定義である．
77	ref_cfg　p.300	システムの静的な情報やコンフィギュレーションで指定された情報を参照し，pk_rcfgで指定されるパケットに返す．具体的にどのような情報が参照できるかは，実装定義である．
78	5.2.2 自動車プロファイル p.312	ext_tsk 制約タスクから呼び出された場合の振舞いは実装定義である．
79	7.4 データ型　p.330	TMO タイムアウト指定（符号付き整数，時間単位は実装定義）
80	7.4 データ型　p.330	RELTIM 相対時間（符号無し整数，時間単位は実装定義）
81	7.4 データ型　p.330	SYSTIM システム時刻（符号無し整数，時間単位は実装定義）
82	7.4 データ型　p.330	OVRTIM プロセッサ時間（符号無し整数，時間単位は実装定義）

索引

この索引は，μITRON4.0仕様の本体（第2章～第5章）で用いている用語の索引で，用語が定義または説明されているページを参照するものである．

A

API 22

C

CPU例外ハンドラ ....58，296
CPUロック解除状態 63
CPUロック状態 63

F

FCFS 53

I

ID番号（オブジェクトの） 24
IRC 55
ITRON仕様共通規定 21
ITRON仕様共通静的API 48
ITRON仕様共通定数 43
ITRON仕様共通データ型 41
ITRON仕様共通マクロ 47

T

TCB 116

あ

アラームハンドラ 250

い

一般定数式パラメータ 34
イベントフラグ 135

え

エラークラス 26
エラーコード 26

お

オーバランハンドラ 258
オブジェクト 24
オブジェクト状態エラークラス 45
オブジェクト属性 28

オブジェクト番号 24

か

カーネル構成定数 74
カーネル構成マクロ 74
カーネルの管理外の割込み 56
返値（サービスコールの） 26
拡張サービスコール 291
拡張サービスコールルーチン 60, 291
拡張情報 28
可変長メモリプール 224
簡略化した優先度制御規則 171, 349

き

起床待ち状態 103
起床要求キューイング数 101
起動（タスクの） 51
起動コード（タスクの） 79
起動された状態 51
起動要求キューイング数 79
機能コード 26
キューイング（タスクに対する起床要求の） 101
キューイング（タスクに対する起動要求の） 79
休止状態 51
強制待ち状態 51
強制待ち要求ネスト数 101

け

警告クラス 46
現在優先度 79
厳密な優先度制御規則 171

こ

広義の待ち状態 50
コールバック 22
固定長メモリプール 214
コンテキスト 49
コンフィギュレータ 30

さ

サービスコール 22
再開（強制待ちからの） 51
削除（オブジェクトの） 72

サブエラーコード 26
サブセット化（サービスコールの機能の） 305

し

時間経過待ち状態 112
資源不足エラークラス 45
システムオブジェクト 25
システムコール 22
システムコンフィギュレーションファイル 30
システム時刻 ....30，235
自タスク 49
実行可能状態 50
実行状態 50
実行できる状態 50
実時刻 30
実装依存 21
実装定義 21
実装独自 21
自動車制御用プロファイル 309
自動割付け（コンフィギュレータによる） 34
自動割付け（サービスコールによる） 72
自動割付け結果ヘッダファイル 24
自動割付け対応整数値パラメータ 34
自動割付け非対応整数値パラメータ 34
周期ハンドラ 240
終了（タスクの） 51
上限プロセッサ時間 258
状態遷移（タスクの） 51
使用プロセッサ時間 258
初期化ルーチン ....72，296
処理単位 59

す

推移的な優先度継承 171
スケジューラ 49
スケジューリング 49
スケジューリング規則 53
スタンダードプロファイル 306

せ

制限（サービスコールの機能の） 305
整数値パラメータ 35

生成（オブジェクトの）……24，72
静的API……22
静的APIの処理……72
制約タスク……310
セマフォ……126

そ

相対時間……30
ソフトウェア部品識別名……36

た

大域脱出……115
タイムアウト……28
タイムイベントハンドラ……60，235
タイムティック……235
タスク……49
タスクコンテキスト……60
タスクコンテキスト専用のサービスコール……68
タスク状態……50
タスクスケジューラ……49
タスクスケジューリング……49
タスクディスパッチ……49
タスクディスパッチャ……49
タスク例外処理許可状態……113
タスク例外処理禁止状態……113
タスク例外処理ルーチン……58，113
タスク例外要因……113

ち

遅延実行（サービスコールの）……69

つ

追加（オブジェクトの）……72

て

定義（オブジェクトの）……25
定数……23
ディスパッチ……49
ディスパッチ許可状態……64
ディスパッチ禁止状態……64
ディスパッチ保留状態……66
ディスパッチャ……49
出入口処理（CPU例外ハンドラの）……58

出入口処理（割込みハンドラの） 56
データ型 23
データキュー 147

と

登録（オブジェクトの） 24

な

内部エラークラス 44
内部識別子 41

に

二重待ち状態 51

ね

ネスト（タスクに対する強制待ち要求の） 101

の

ノンブロッキング 29

は

パケット 23
パラメータ 23
パラメータエラークラス 44

ひ

引数 23
非タスクコンテキスト 60
非タスクコンテキスト専用のサービスコール 67

ふ

不可分性（サービスコールの） 62
プリエンプト 51
プリプロセッサ定数式パラメータ 34
プロファイル規定 305

へ

ベーシックプロファイル 308
ベース優先度 79
ヘッダファイル 24
返答メッセージ（ランデブの） 194

ほ

ポーリング 29

保留例外要因 113

ま

マクロ 24
待ち解除 51
待ち解除エラークラス 46
待ち状態 50

み

未サポートエラークラス 44
未定義 21
未登録状態 51
ミューテックス 170

め

メインエラーコード .... 26，44
メールボックス 159
メッセージパケット 159
メッセージバッファ 181
メッセージヘッダ 159
メモリプール 214

ゆ

ユーザオブジェクト 25
優先順位 49
優先順位（処理単位の）........ 61
優先順位（タスクの）........ 53
優先度 25
優先度（タスクの）........ 79
優先度逆転 170
優先度継承プロトコル 170
優先度上限プロトコル 170

よ

呼出しコンテキストエラークラス 44
呼出しメッセージ（ランデブの）........ 194
弱い標準化 305

ら

ラウンドロビン方式 .... 53，267
ランデブ 194
ランデブポート 194
ランデブ番号 195

り

リターンパラメータ 23

わ

割込み 55
割込みサービスルーチン ....55，278
割込み番号 57
割込みハンドラ ....55，278
割込みハンドラ番号 57

静的API索引

この索引は，µITRON4.0仕様の静的APIのアルファベット順の索引である．

ATT_INI 初期化ルーチンの追加.....303
ATT_ISR 割込みサービスルーチンの追加.....283
CRE_ALM アラームハンドラの生成.....252
CRE_CYC 周期ハンドラの生成.....243
CRE_DTQ データキューの生成.....150
CRE_FLG イベントフラグの生成.....137
CRE_MBF メッセージバッファの生成.....184
CRE_MBX メールボックスの生成.....162
CRE_MPF 固定長メモリプールの生成.....216
CRE_MPL 可変長メモリプールの生成.....226
CRE_MTX ミューテックスの生成.....174
CRE_POR ランデブポートの生成.....198
CRE_SEM セマフォの生成.....128
CRE_TSK タスクの生成.....83
DEF_EXC CPU例外ハンドラの定義.....298
DEF_INH 割込みハンドラの定義.....281
DEF_OVR オーバランハンドラの定義.....260
DEF_SVC 拡張サービスコールの定義.....293
DEF_TEX タスク例外処理ルーチンの定義.....118
INCLUDE ファイルのインクルード.....48

マイクロアイトロン
μITRON4.0 仕様 Ver.4.03.00

1999 年 6 月 30 日 Ver.4.00.00 第 1 刷 発行
2001 年 5 月 31 日 Ver.4.01.00 第 1 刷 発行
2004 年 4 月 30 日 Ver.4.02.00 第 1 刷 発行
2006 年 12 月 5 日 Ver.4.03.00 第 1 刷 発行

監修　坂村 健

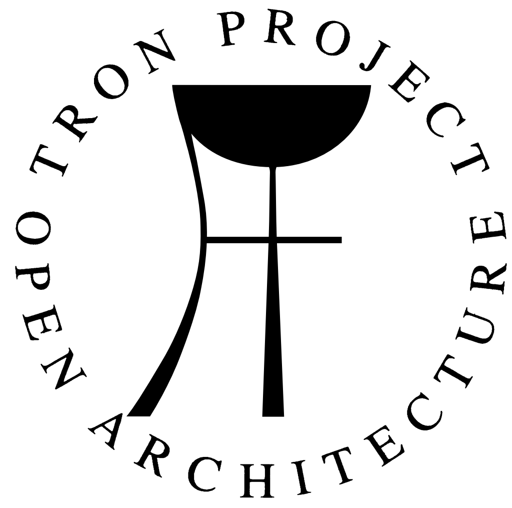
編集／発行　社団法人トロン協会

〒108-0073 東京都港区三田 1-3-39 勝田ビル 5F

電話 03-3454-3191

印刷　ホクエツ印刷株式会社

©社団法人トロン協会 1999～2006


TRON PROJECT
OPEN ARCHITECTURE