TOPPERS/SSP 新世代カーネル

統合仕様書

Ver. 1.4.0

最終更新日：2012/05/16

＜はじめに＞

本ドキュメントは，TOPPERS プロジェクトから配布されている「TOPPERS 新世代カーネル統合仕様書 Release 1.4.0」をリッチテキスト形式で再編集したものです. .

原文を変更しないように編集していますが，再編集時の不注意により記載内容が原文と異なる可能性があります．その場合は原文自体に誤りがある場合も含めて常に原文を正としてください．また，その箇所を TOPPERS プロジェクトのユーザ ML もしくは開発者 ML（TOPPERS プロジェクト会員のみアクセス可能）にてお知らせいただけると幸いです.

TOPPERS 準会員 杉本 明加

このドキュメントは，TOPPERS 新世代カーネルに属する一連のリアルタイムカーネルの仕様を，統合的に記述するものである．現時点で，TOPPERS/ASP カーネル，TOPPERS/FMP カーネル，TOPPERS/HRP2 カーネル，TOPPERS/SSP カーネルの仕様に関しては記述が完成しているが，未完成部分も残っている．特に動的生成対応カーネルについては，仕様検討が不十分なところが多い．

なお，TOPPERS/SSP カーネルは，Release 1.1.0 の時点では，この仕様に準拠していない．今後のリリースでは，この仕様に準拠するよう修正される予定である．

＜ライセンス条項＞

＜目次＞

1. TOPPERS 新世代カーネルの概要

TOPPERS 新世代カーネルとは，TOPPERS プロジェクトにおいて ITRON 仕様をベースとして開発している一連のリアルタイムカーネルの総称である．この章では，TOPPERS 新世代カーネル仕様の位置付けと設計方針，それに属する各カーネルの適用対象領域と設計方針について述べる．

1.1. TOPPERS 新世代カーネル仕様の位置付け

TOPPERSプロジェクトでは，2000年に公開したTOPPERS/JSPカーネルを始めとして，μITRON4.0仕様およびその保護機能拡張（μITRON4.0/PX 仕様）に準拠したリアルタイムカーネルを開発してきた．

μITRON4.0 仕様は 1999 年に，μITRON4.0/PX 仕様は 2002 年に公表されたが，それ以降現在までの間に，大きな仕様改訂は実施されていない．その間に，組込みシステムおよびソフトウェアのますますの大規模化・複雑化，これまで以上に高い信頼性・安全性に対する要求，小さい消費エネルギー下での高い性能要求など，組込みシステム開発を取り巻く状況は刻々変化している．リアルタイムカーネルに対しても，マルチプロセッサへの対応，発展的な保護機能のサポート，機能安全対応，省エネルギー制御機能のサポートなど，新しい要求が生じている．

TOPPERS プロジェクトでは，リアルタイムカーネルに対するこのような新しい要求に対応するために，μITRON4.0 仕様を発展させる形で，TOPPERS 新世代カーネル仕様を策定することになった．

ただし，ITRON 仕様が，各社が開発するリアルタイムカーネルを標準化することを目的に，リアルタイムカーネルの「標準仕様」を規定することを目指しているのに対して，TOPPERS 新世代カーネル仕様は，TOPPERS プロジェクトにおいて開発している一連のリアルタイムカーネルの「実装仕様」を記述するものであり，ITRON 仕様とは異なる目的・位置付けを持つものである．

1.2. TOPPERS 新世代カーネル仕様の設計方針

TOPPERS 新世代カーネル仕様を設計するにあたり，次の方針を設定する．

(1) μITRON4.0 仕様をベースに拡張・改良を加える

TOPPERS 新世代カーネル仕様は，多くの技術者の尽力により作成され，多くの実装・使用実績があるμITRON4.0 仕様をベースとする．ただし，μITRON4.0 仕様の策定時以降の状況の変化を考慮し，μITRON4.0 仕様で不十分と考えられる点については積極的に拡張・改良する．μITRON4.0 仕様への準拠性にはこだわらない．

(2) ソフトウェアの再利用性を重視する

μITRON4.0 仕様の策定時点と比べると，組込みソフトウェアの大規模化が進展している一方で，ハードウェアの性能向上も著しい．そのため，ソフトウェアの再利用性を向上させるためには，少々のオーバヘッドは許容される状況にある．

そこで，TOPPERS 新世代カーネル仕様では，μITRON4.0 仕様においてオーバヘッド削減のために実装定義または実装依存としていたような項目についても，ターゲットシステムに依存する項目とするのではなく，強く規定する方針とする．

(3) 高信頼・安全なシステム構築を支援する

TOPPERS 新世代カーネル仕様は，高信頼・安全な組込みシステム構築を支援するものとする．

安全性の面では，アプリケーションプログラムに問題がある場合でも，リーゾナブルなオーバヘッドでそれを救済できるなら，救済するような仕様とする．また，アプリケーションプログラムの誤動作を検出する機能や，システムの自己診断のための機能についても，順次取り込んでいく．

(4) アプリケーションシステム構築に必要な機能は積極的に取り込む

上記の方針を満たした上で，多くのアプリケーションシステムに共通に必要となる機能については，積極的にカーネルに取り込む．

カーネル単体の信頼性を向上させるためには，カーネルの機能は少なくした方が楽である．しかし，アプリケーションシステム構築に必要となる機能は，カーネルがサポートしていなければアプリケーションプログラムで実現しなければならず，システム全体の信頼性を考えると，多くのアプリケーションシステムに共通に必要となる機能については，カーネルに取り込んだ方が有利である．

1.3. TOPPERS/ASP カーネルの適用対象領域と仕様設計方針

TOPPERS/ASP カーネル（ASP は，Advanced Standard Profile の略．以下，ASP カーネル）は，TOPPERS 新世代カーネルの出発点となるリアルタイムカーネルである．保護機能を持ったカーネルやマルチプロセッサ対応のカーネルは，ASP カーネルを拡張する形で開発する．

ASP カーネルは，20 年以上に渡る ITRON 仕様の技術開発成果をベースとして，完成度の高いリアルタイムカーネルを実現するものである．完成度を高めるという観点から，カーネル本体の仕様については，枯れた技術で実装できる範囲に留める．

ASP カーネルの主な適用対象は，高い信頼性・安全性・リアルタイム性を要求される組込みシステム

とする．ソフトウェア規模の面では，プログラムサイズ（バイナリコード）が数十 KB～1MB 程度のシステムを主な適用対象とする．それより大規模なシステムには，保護機能を持ったリアルタイムカーネルを適用すべきと考えられる．

ASP カーネルの機能は，カーネル内で動的なメモリ管理が不要な範囲に留める．これは，高い信頼性・安全性・リアルタイム性を要求される組込みシステムでは，システム稼働中に発生するメモリ不足への対処が難しいためである．この方針から，カーネルオブジェクトは静的に生成することとし，動的なオブジェクト生成機能は設けない．ただし，アプリケーションプログラムが動的なメモリ管理をするためのカーネル機能である固定長メモリプール機能はサポートする．

1.4. TOPPERS/FMP カーネルの適用対象領域と仕様設計方針

TOPPERS/FMP カーネル（FMP は，Flexible Multiprocessor Profile の略．以下，FMP カーネル）は，ASP カーネルを，マルチプロセッサ対応に拡張したリアルタイムカーネルである
.

FMP カーネルの適用対象となるターゲットハードウェアは，ホモジニアスなマルチプロセッサシステムである．各プロセッサが全く同一のものである必要はないが，すべてのプロセッサでバイナリコードを共有することから，同じバイナリコードを実行できることが必要である．

FMP カーネルでは，タスクを実行するプロセッサを静的に決定するのが基本であり，カーネルは自動的に負荷分散する機能を持たないが，タスクをマイグレーションさせるサービスコールを備えている．これを用いて，アプリケーションで動的な負荷分散を実現することが可能である．

FMP カーネルの機能は，ASP カーネルと同様に，カーネル内で動的なメモリ管理が不要な範囲に留める．

1.5. TOPPERS/HRP2 カーネルの適用対象領域と仕様設計方針

TOPPERS/HRP2 カーネル（HRP は，High Reliable system Profile の略．2 はバージョン番号を示す．以下，HRP2 カーネル）は，さらに高い信頼性・安全性を要求される組込みシステムや，より大規模な組込みシステム向けに適用できるように，ASP カーネルを拡張したリアルタイムカーネルである．

HRP2 カーネルの適用対象となるターゲットハードウェアは，特権モードと非特権モードを備え，メモリ保護のために MMU（Memory Management Unit）または MPU（Memory Protection Unit）を持つプロセッサを用いたシステムである．

HRP2 カーネルの主な適用対象は，ソフトウェア規模の面では，プログラムサイズ（バイナリコード）が数百 KB 以上のシステムである．

HRP2 カーネルの機能は，ASP カーネルと同様に，カーネル内で動的なメモリ管理が不要な範囲に留める．具体的には，ASP カーネルに対して，メモリ保護機能とオブジェクトアクセス保護機能，拡張サービスコール機能，ミューテックス機能，オーバランハンドラ機能を追加し，メールボックス機能を削除している．

1.6. TOPPERS/SSP カーネルの適用対象領域と仕様設計方針

TOPPERS/SSP カーネル（SSP は，Smallest Set Profile の略．以下，SSP カーネル）は，小規模システムに用いるために，ASP カーネルをベースに可能な限り機能を絞り込んだリアルタイムカーネルである．

SSP カーネルの機能は，μITRON4.0 仕様の「仕様準拠の最低条件」の考え方を踏襲し，メモリ使用量を最小化するように定めている．具体的には，SSP カーネルにおいては，タスクは待ち状態を持たない（言い換えると，制約タスクのみをサポートする）のが最大の特徴である．また，ASP カーネルに対して下位互換性を持つように配慮しているが，システム全体のメモリ使用量を最小化するために有用な機能は，ASP カーネルに対して追加している．

TOPPERS/SSP カーネルの主な適用対象は，プログラムサイズ（バイナリコード）が数 KB～数十 KB 程度の極めて小規模な組込みシステムである．

1.7. TOPPERS/ASP Safety カーネルの適用対象領域と仕様設計方針

TOPPERS/ASP Safety カーネル（以下，ASP Safety カーネル）は，小規模な安全関連システムに用いるために，ASP カーネルの機能を徹底的な検証が可能な範囲にサブセット化したものである．メールボックスのように安全性の観点から問題のある機能や，タスク例外処理機能のように使用頻度に比べて検証にコストのかかる機能はサポートしない．

ASP Safety カーネルの主な適用対象は，特に高い安全性を要求される組込みシステムとする．ソフトウェア規模の面では，プログラムサイズ（バイナリコード）が数十 KB～1MB 程度のシステムを主な適用対象とする．それより大規模なシステムには，保護機能を持ったカーネルを適用すべきと考えられる．

2. 主要な概念と共通定義

2.1. 仕様の位置付け

2.1.1. 仕様の位置付け

この仕様は，TOPPERS 新世代カーネルに属する各カーネルの仕様を，統合的に記述することを目標としている．また，TOPPERS 新世代カーネル上で動作する各種のシステムサービスに共通に適用される事項についても規定する．

2.1.2. カーネルの機能セット

TOPPERS 新世代カーネルは，ASP カーネルをベースとして，保護機能，マルチプロセッサ，カーネルオブジェクトの動的生成，機能安全などに対応した一連のカーネルで構成される．

この仕様では，TOPPERS 新世代カーネルを構成する一連のカーネルの仕様を統合的に記述するが，言うまでもなく，カーネルの種類によってサポートする機能は異なる．サポートする機能をカーネルの種類毎に記述する方法もあるが，カーネルの種類はユーザ要求に対応して増える可能性もあり，その度に仕様書を修正するのは得策ではない．

そこでこの仕様では，サポートする機能を，カーネルの種類毎ではなく，カーネルの対応する機能セット毎に記述する．具体的には，保護機能を持ったカーネルを保護機能対応カーネル，マルチプロセッサに対応したカーネルをマルチプロセッサ対応カーネル，カーネルオブジェクトの動的生成機能を持ったカーネルを動的生成対応カーネルと呼ぶことにする．

【TOPPERS/ASP カーネルにおける規定】
ASP カーネルは，保護機能対応カーネル，マルチプロセッサ対応カーネル，動的生成対応カーネルのいずれでもない．ただし，動的生成機能拡張パッケージを用いると，動的生成対応カーネルとなる．

【TOPPERS/FMP カーネルにおける規定】
FMP カーネルは，マルチプロセッサ対応カーネルであり，保護機能対応カーネル，動的生成対応カーネルではない．

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルは，保護機能対応カーネルであり，マルチプロセッサ対応カーネル，動的生成対応カーネルではない．

【TOPPERS/SSP カーネルにおける規定】
SSP カーネルは，保護機能対応カーネル，マルチプロセッサ対応カーネル，動的生成対応カーネルの

いずれでもない.

【μITRON4.0 仕様，μITRON4.0/PX 仕様との関係】

μITRON4.0 仕様は，カーネルオブジェクトの動的生成機能を持っているが，保護機能を持っておらず，マルチプロセッサにも対応していない．μITRON4.0/PX 仕様は，μITRON4.0 仕様に対して保護機能を追加するための仕様であり，カーネルオブジェクトの動的生成機能と保護機能を持っているが，マルチプロセッサには対応していない.

2.1.3. ターゲット非依存の規定とターゲット定義の規定

TOPPERS 新世代カーネルは，アプリケーションプログラムの再利用性を向上させるために，ターゲットハードウェアや開発環境の違いをできる限り隠蔽することを目指している．ただし，ターゲットハードウェアや開発環境の制限によって実現できない機能が生じたり，逆にターゲットハードウェアの特徴を活かすためには機能拡張が不可欠になる場合がある．また，同一のターゲットハードウェアであっても，アプリケーションシステムによって使用方法が異なる場合があり，ターゲットシステム毎に仕様の細部に違いが生じることは避けられない.

そこで，TOPPERS 新世代カーネルの仕様は，ターゲットシステムによらずに定めるターゲット非依存（target-independent）の規定と，ターゲットシステム毎に定めるターゲット定義（target-defined）の規定に分けて記述する．この仕様書は，ターゲット非依存の規定について記述するものであり，この仕様書で「ターゲット定義」とした事項は，ターゲットシステム毎に用意するドキュメントにおいて規定する.

また，この仕様書でターゲット非依存に規定した事項であっても，ターゲットハードウェアや開発環境の制限によって実現できない場合や，実現するためのオーバヘッドが大きくなる場合には，この仕様書の規定を逸脱する場合がある．このような場合には，ターゲットシステム毎に用意するドキュメントでその旨を明記する.

2.1.4. 想定するソフトウェア構成

この仕様では，アプリケーションシステムを構成するソフトウェアを，アプリケーションプログラム（以下，単にアプリケーションと呼ぶ），システムサービス，カーネルの 3 階層に分けて考える（図 2-1）．カーネルとシステムサービスをあわせて，ソフトウェアプラットフォームと呼ぶ.

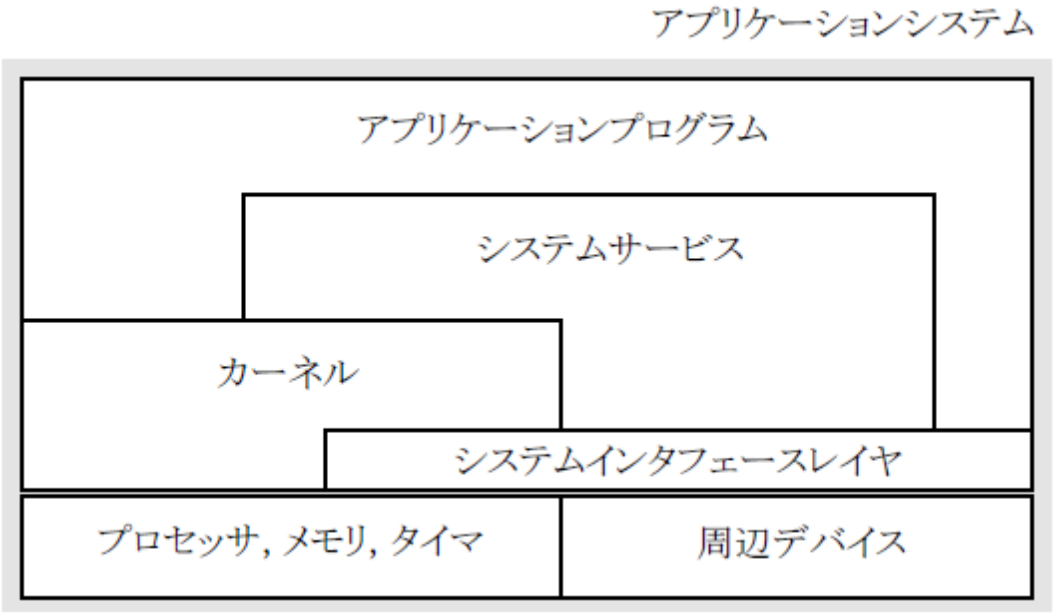

図 2-1 想定するソフトウェア構成

カーネルは，コンピュータの持つ最も基本的なハードウェア資源であるプロセッサ，メモリ，タイマを抽象化し，上位階層のソフトウェア（アプリケーションおよびシステムサービス）に論理的なプログラム実行環境を提供するソフトウェアである．

システムサービスは，各種の周辺デバイスを抽象化するソフトウェアで，ファイルシステムやネットワークプロトコルスタック，各種のデバイスドライバなどが含まれる．

また，この仕様では，プロセッサと各種の周辺デバイスの接続方法を隠蔽するためのソフトウェア階層として，システムインタフェースレイヤ（SIL）を規定する．

システムインタフェースレイヤ，カーネル，各種のシステムサービス（これらをモジュールと呼ぶ）を，上位階層のソフトウェアから使うためのインタフェースを，API（Application Programming Interface）と呼ぶ．

この仕様書では，第 3 章においてシステムインタフェースレイヤの API 仕様を，第 4 章においてカーネルの API 仕様を規定する．システムサービスの API 仕様は，システムサービス毎の仕様書で規定される．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様では，カーネルとアプリケーションの中間にあるソフトウェアをソフトウェア部品と呼んでいたが，TOPPERS 組込みコンポーネントシステム（TECS）においてはカーネルもソフトウェア部品の 1 つと捉えることから，この仕様ではシステムサービスと呼ぶことにした．

2.1.5. 想定するハードウェア構成

この仕様では，カーネルがサポートするハードウェア構成として，以下のことを想定している．これらに合致しないターゲットハードウェアでカーネルを動作させることは可能であるが，合致しない部分への適応はアプリケーションの責任になる．

(a) メモリ番地は，常に同一のメモリを指すこと（オーバレイのように，異なるメモリを同一のメモリ番地でアクセスすることがないこと）．マルチプロセッサ対応カーネルにおいては，同一のメモリに対しては，各プロセッサから同一の番地でアクセスできること．

(b) マルチプロセッサ対応カーネルにおいては，各プロセッサが同一の機械語命令を実行できること．

2.1.6. 想定するプログラミング言語

この仕様における API 仕様は，ISO/IEC 9899:1990（以下，C90 と呼ぶ）または ISO/IEC 9899:1999（以下，C99 と呼ぶ）に準拠した C 言語を，フリースタンディング環境で用いることを想定して規定している．

ただし，C90 の規定に加えて，以下のことを仮定している．

- 16 ビットおよび 32 ビットの整数型があること
- ポインタが格納できるサイズの整数型があること

2.2. 2.2 API の構成要素とコンベンション

2.2.1. API の構成要素

(1) サービスコール

上位階層のソフトウェアから，下位階層のソフトウェアを呼び出すインタフェースをサービスコール（service call）と呼ぶ．カーネルのサービスコールを，システムコール（system call）と呼ぶ場合もある．

(2) コールバック

下位階層のソフトウェアから，上位階層のソフトウェアを呼び出すインタフェースをコールバック（callback）と呼ぶ．

(3) 静的 API

オブジェクトの生成情報や初期状態などを定義するために，システムコンフィギュレーションファイル中に記述するインタフェースを，静的 API（static API）と呼ぶ．

(4) 構成マクロ

下位階層のソフトウェアに関する各種の情報を取り出すために，上位階層のソフトウェアが用いるマクロを，構成マクロ（configuration macro）と呼ぶ．

2.2.2. パラメータとリターンパラメータ

サービスコールやコールバックに渡すデータをパラメータ（parameter），それらが返すデータをリターンパラメータ（return parameter）と呼ぶ．また，静的 API に渡すデータもパラメータと呼ぶ．

オブジェクトを生成するサービスコールなど，パラメータの数が多い場合やターゲット定義のパラメータを追加する可能性がある場合には，複数のパラメータを 1 つの構造体に入れ，その領域へのポインタをパラメータとして渡す．また，パラメータのサイズが大きい場合にも，パラメータを入れた領域へのポインタをパラメータとして渡す場合がある．

C 言語 API では，リターンパラメータは，関数の返値とするか，リターンパラメータを入れる領域へのポインタをパラメータとして渡すことで実現する．オブジェクトの状態を参照するサービスコールなど，リターンパラメータの数が多い場合やターゲット定義のリターンパラメータを追加する可能性がある場合には，複数のリターンパラメータを 1 つの構造体に入れて返すこととし，その領域へのポインタをパラメータとして渡す．

複数のパラメータまたはリターンパラメータを入れるための構造体を，パケット（packet）と呼ぶ．

サービスコールやコールバックに，パケットを置く領域へのポインタやリターンパラメータを入れる領域へのポインタを渡す場合，別に規定がない限りは，サービスコールやコールバックの処理が完了した後は，それらの領域が参照されることはなく，別の目的に使用できる．

2.2.3. 返値とエラーコード

一部の例外を除いて，サービスコールおよびコールバックの返値は，処理が正常終了したかを表す符号付き整数とする．処理が正常終了した場合には，E_OK（＝0）または正の値が返るものとし，値の意味はサービスコールまたはコールバック毎に定める．処理が正常終了しなかった場合には，その原因を表す負の値が返る．処理が正常終了しなかった原因を表す値を，エラーコード（error code）と呼ぶ．

エラーコードは，いずれも負の値のメインエラーコードとサブエラーコードで構成される．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構成するマクロ（ERCD）と，エラーコードからメインエラーコードを取り出すマクロ（MERCD），サブエラーコードを取り出すマクロ（SERCD）が用意されている．

メインエラーコードの名称・意味・値は，カーネルとシステムサービスで共通に定める（「2.14.4TOPPERS 共通エラーコード」の節を参照）．サービスコールおよびコールバックの機能説明中の「E_XXXXX エラーとなる」または「E_XXXXX エラーが返る」という記述は，メインエラーコードとして E_XXXXX が返ることを意味する．

サブエラーコードは，エラーの原因をより詳細に表すために用いる．カーネルはサブエラーコードを

使用せず，サブエラーコードとして常に-1 が返る．サブエラーコードの名称・意味・値は，サブエラーコードを使用するシステムサービスの API 仕様において規定する．

サービスコールが負の値のエラーコード（警告を表すものを除く）を返した場合には，サービスコールによる副作用がないのが原則である．ただし，そのような実装ができない場合にはこの原則の例外とし，サービスコールの機能説明にその旨を記述する．

サービスコールが複数のエラーを検出するべき状況では，その内のいずれか 1 つのエラーを示すエラーコードが返る．

コールバックが複数のエラーを検出するべき状況では，その内のいずれか 1 つのエラーを示すエラーコードを返せばよい．

なお，静的 API は返値を持たない．静的 API の処理でエラーが検出された場合の扱いについては，「2.12.5 コンフィギュレータの処理モデル」の節および「2.12.6 静的 API のパラメータに関するエラー検出」の節を参照すること．

2.2.4. 機能コード

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などに用いるためのサービスコールを識別するための番号を，機能コード（function code）と呼ぶ．機能コードは符号付きの整数値とし，カーネルのサービスコールには負の値を割り付け，拡張サービスコールには正の値を用いる．

2.2.5. ヘッダファイル

カーネルやシステムサービスを用いるために必要な定義を含むファイル．

ヘッダファイルは，原則として，複数回インクルードしてもエラーにならないように対処されている．具体的には，ヘッダファイルの先頭で特定の識別子（例えば，kernel.h なら”TOPPERS_KERNEL_H”）がマクロ定義され，ヘッダファイルの内容全体をその識別子が定義されていない場合のみ有効とする条件ディレクティブが付加されている．

2.3. 主な概念

2.3.1. オブジェクトと処理単位

(1) オブジェクト

カーネルまたはシステムサービスが管理対象とするソフトウェア資源を，オブジェクト（object）と呼ぶ．特に，カーネルが管理対象とするソフトウェア資源を，カーネルオブジェクト（kernel object）と

呼ぶ.

オブジェクトは，種類毎に，番号によって識別する．カーネルまたはシステムサービスで，オブジェクトに対して任意に識別番号を付与できる場合には，1 から連続する正の整数値でオブジェクトを識別する．この場合に，オブジェクトの識別番号を，オブジェクトの ID 番号（ID number）と呼ぶ．そうでない場合，すなわちカーネルまたはシステムサービスの内部または外部からの条件によって識別番号が決まる場合には，オブジェクトの識別番号を，オブジェクト番号（object number）と呼ぶ．識別する必要のないオブジェクトには，識別番号を付与しない場合がある．

オブジェクト属性（object attribute）は，オブジェクトの動作モードや初期状態を定めるもので，オブジェクトの登録時に指定する．オブジェクト属性に TA_XXXX が指定されている場合，そのオブジェクトを，TA_XXXX 属性のオブジェクトと呼ぶ．複数の属性を指定する場合には，オブジェクト属性を渡すパラメータに，指定する属性値のビット毎論理和（C 言語の”|”）を渡す．また，指定すべきオブジェクト属性がない場合には，TA_NULL を指定する．

(2) 処理単位

オブジェクトの中には，プログラムが対応付けられるものがある．プログラムが対応付けられるオブジェクト（または，対応付けられるプログラム）を，処理単位（processing unit）と呼ぶ．処理単位に対応付けられるプログラムは，アプリケーションまたはシステムサービスで用意し，カーネルが実行制御する．

処理単位の実行を要求することを起動 (activate)，処理単位の実行を開始することを実行開始 (start) と呼ぶ.

拡張情報（extended information）は，処理単位が呼び出される時にパラメータとして渡される情報で，処理単位の登録時に指定する．拡張情報は，カーネルやシステムサービスの動作には影響しない.

(3) タスク

カーネルが実行順序を制御するプログラムの並行実行の単位をタスク（task）と呼ぶ．タスクは，処理単位の 1 つである.

サービスコールの機能説明において，サービスコールを呼び出したタスクを，自タスク (invoking task) と呼ぶ．拡張サービスコールから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を呼び出したタスクが自タスクである.

カーネルには，静的 API により，少なくとも 1 つのタスクを登録しなければならない．タスクが登録されていない場合には，コンフィギュレータがエラーを報告する.

【補足説明】

タスクが呼び出した拡張サービスコールが実行されている間は，「サービスコールを呼び出した処理単位」は拡張サービスコールであり，「自タスク」とは一致しない．そのため，保護機能対応カーネルにおいて，「サービスコールを呼び出した処理単位の属する保護ドメイン」と「自タスクの属する保護ドメイン」は，異なるものを指す．

(4) ディスパッチとスケジューリング

プロセッサが実行するタスクを切り換えることを，タスクディスパッチまたは単にディスパッチ（dispatching）と呼ぶ．それに対して，次に実行すべきタスクを決定する処理を，タスクスケジューリングまたは単にスケジューリング（scheduling）と呼ぶ．

ディスパッチが起こるべき状態（すなわち，スケジューリングによって，現在実行しているタスクとは異なるタスクが，実行すべきタスクに決定されている状態）となっても，何らかの理由でディスパッチを行わないことを，ディスパッチの保留（pend dispatching）という．ディスパッチを行わない理由が解除された時点で，ディスパッチが起こる．

(5) 割込みと CPU 例外

プロセッサが実行中の処理とは独立に発生するイベントによって起動される例外処理のことを，外部割込みまたは単に割込み（interrupt）と呼ぶ．それに対して，プロセッサが実行中の処理に依存して起動される例外処理を，CPU 例外（CPU exception）と呼ぶ．

周辺デバイスからの割込み要求をプロセッサに伝える経路を遮断し，割込み要求が受け付けられるのを抑止することを，割込みのマスク（mask interrupt）または割込みの禁止（disable interrupt）という．マスクが解除された時点で，まだ割込み要求が保持されていれば，その時点で割込み要求を受け付ける．

マスクすることができない割込みを，NMI（non-maskable interrupt）と呼ぶ．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様において，未定義のまま使われていた割込みと CPU 例外という用語を定義した．

(6) タイムイベントとタイムイベントハンドラ

時間の経過をきっかけに発生するイベントをタイムイベント（time event）と呼ぶ．タイムイベントにより起動され，カーネルが実行制御する処理単位を，タイムイベントハンドラ（time event handler）と呼ぶ．

2.3.2. サービスコールとパラメータ

(1) 優先順位と優先度

優先順位（precedence）とは，処理単位の実行順序を説明するための仕様上の概念である．複数の処理単位が実行できる場合には，その中で最も優先順位の高い処理単位が実行される．

優先度（priority）は，タスクなどの処理単位の優先順位や，メッセージなどの配送順序を決定するために，アプリケーションが処理単位やメッセージなどに与える値である．優先度は，符号付きの整数型である PRI 型で表し，1 から連続した正の値を用いるのを原則とする．優先度は，値が小さいほど優先度が高い（すなわち，先に実行または配送される）ものとする．

(2) システム時刻と相対時間

カーネルが管理する時刻を，システム時刻（system time）と呼ぶ．システム時刻は，符号無しの整数型である SYSTIM 型で表し，単位はミリ秒とする．システム時刻は，タイムティック（time tick）を通知するためのタイマ割込みが発生する毎に更新される．

イベントを発生させる時刻を指定する場合には，基準時刻（base time）からの相対時間（relative time）によって指定する．基準時刻は，別に規定がない限りは，相対時間を指定するサービスコールを呼び出した時刻となる．

相対時間は，符号無しの整数型である RELTIM 型で表し，単位はシステム時刻と同一，すなわちミリ秒とする．相対時間には，少なくとも，16 ビットの符号無しの整数型（uint16_t 型）に格納できる任意の値を指定することができるが，RELTIM 型（uint_t 型に定義される）に格納できる任意の値を指定できるとは限らない．相対時間に指定できる最大値は，構成マクロ TMAX_RELTIM に定義されている．

イベントを発生させる時刻を相対時間で指定した場合，イベントの処理が行われるのは，基準時刻から相対時間によって指定した以上の時間が経過した後となる．ただし，基準時刻を定めるサービスコールを呼び出した時に，タイムティックを通知するためのタイマ割込みがマスクされている場合（タイマ割込みより優先して実行される割込み処理が実行されている場合を含む）は，相対時間によって指定した以上の時間が経過した後となることは保証されない．

イベントが発生する時刻を参照する場合には，基準時刻からの相対時間として返される．基準時刻は，相対時間を返すサービスコールを呼び出した時刻となる．

イベントが発生する時刻が相対時間で返された場合，イベントの処理が行われるのは，基準時刻から相対時間として返された以上の時間が経過した後となる．ただし，相対時間を返すサービスコールを呼び出した時に，タイムティックを通知するためのタイマ割込みがマスクされている場合（タイマ割込みより優先して実行される割込み処理が実行されている場合を含む）は，相対時間として返された以上の時間が経過した後となることは保証されない．

【補足説明】

相対時間に 0 を指定した場合，基準時刻後の最初のタイムティックでイベントの処理が行われる．また，1 を指定した場合，基準時刻後の 2 回目以降のタイムティックでイベントの処理が行われる．これは基準時刻後の最初のタイムティックは，基準時刻の直後に発生する可能性があるため，ここでイベントの処理を行うと，基準時刻からの経過時間が 1 以上という仕様を満たせないためである．

同様に，相対時間として 0 が返された場合，基準時刻後の最初のタイムティックでイベントの処理が行われる．また，1 が返された場合，基準時刻後の 2 回目以降のタイムティックでイベントの処理が行われる．

【μITRON4.0 仕様との関係】

相対時間（RELTIM 型）とシステム時刻（SYSTIM 型）の時間単位は，μITRON4.0 仕様では実装定義としていたが，この仕様ではミリ秒と規定した．また，相対時間の解釈について，より厳密に規定した．

TMAX_RELTIM は，μITRON4.0 仕様に規定されていないカーネル構成マクロである．

(3) タイムアウトとポーリング

サービスコールの中で待ち状態が指定した時間以上継続した場合に，サービスコールの処理を取りやめて，サービスコールからリターンすることを，タイムアウト（timeout）という．タイムアウトしたサービスコールからは，E_TMOUT エラーが返る．

タイムアウトを起こすまでの時間（タイムアウト時間）は，符号付きの整数型である TMO 型で表し，単位はシステム時刻と同一，すなわちミリ秒とする．タイムアウト時間に正の値を指定した場合には，タイムアウトを起こすまでの相対時間を表す．すなわち，タイムアウトの処理が行われるのは，サービスコールを呼び出してから指定した以上の時間が経過した後となる．

ポーリング（polling）を行うサービスコールとは，サービスコールの中で待ち状態に遷移すべき状況になった場合に，サービスコールの処理を取りやめてリターンするサービスコールのことをいう．ここで，サービスコールの処理を取りやめてリターンすることを，ポーリングに失敗したという．ポーリングに失敗したサービスコールからは，E_TMOUT エラーが返る．

ポーリングを行うサービスコールでは，待ち状態に遷移することはないのが原則である．そのため，ポーリングを行うサービスコールは，ディスパッチ保留状態であっても呼び出せる場合がある．ただし，サービスコールの中で待ち状態に遷移する状況が複数ある場合，ある状況でポーリング動作をしても，他の状況では待ち状態に遷移する場合がある．このような場合の振舞いは，該当するサービスコール毎に規定する．

タイムアウト付きのサービスコールは，別に規定がない限りは，タイムアウト時間に TMO_POL（＝0）

を指定した場合にはポーリングを行い，TMO_FEVR（＝-1）を指定した場合にはタイムアウトを起こさないものとする．

【補足説明】
エラーコードに関する原則により，サービスコールがタイムアウトした場合やポーリングに失敗した場合には，サービスコールによる副作用がないのが原則である．ただし，そのような実装ができない場合にはこの原則の例外とし，どのような副作用があるかをサービスコール毎に規定する．

タイムアウト付きのサービスコールを，タイムアウト時間を TMO_POL として呼び出した場合には，ディスパッチ保留状態で呼び出すと E_CTX エラーとなることを除いては，ポーリングを行うサービスコールと同じ振舞いをする．また，タイムアウト時間を TMO_FEVR として呼び出した場合には，タイムアウトなしのサービスコールと全く同じ振舞いをする．

【μITRON4.0 仕様との関係】
タイムアウト時間（TMO 型）の時間単位は，μITRON4.0 仕様では実装定義としていたが，この仕様ではミリ秒と規定した．

【仕様決定の理由】
ディスパッチ保留状態において，ポーリングを行うサービスコールを呼び出せる場合があるのに対して，タイムアウト付きのサービスコールをタイムアウト時間を TMO_POL として呼び出すとエラーになるのは，ディスパッチ保留状態では，別に規定がない限り，自タスクを広義の待ち状態に遷移させる可能性のあるサービスコール（タイムアウト付きのサービスコールはこれに該当）を呼び出すことはできないと規定されているためである．

(4) ノンブロッキング
サービスコールの中で待ち状態に遷移すべき状況になった時，サービスコールの処理を継続したままサービスコールからリターンする場合，そのサービスコールをノンブロッキング（non-blocking）という．処理を継続したままリターンする場合，サービスコールからは E_WBLK エラーが返る．E_WBLK は警告を表すエラーコードであり，サービスコールによる副作用がないという原則は適用されない．

サービスコールから E_WBLK エラーが返った場合には，サービスコールの処理は継続しているため，サービスコールに渡したパラメータまたはリターンパラメータを入れる領域はまだ参照される可能性があり，別の目的に使用することはできない．継続している処理が完了した場合や，何らかの理由で処理が取りやめられた場合には，コールバックを呼び出すなどの方法で，サービスコールを呼び出したソフトウェアに通知するものとする．

ノンブロッキングの指定は，タイムアウト時間に TMO_NBLK（＝-2）を指定することによって行う．ノンブロッキングの指定を行えるサービスコールは，指定した場合の振舞いをサービスコール毎に規定

する．

【補足説明】
ノンブロッキングは，システムサービスでサポートすることを想定した機能である．カーネルは，ノンブロッキングの指定を行えるサービスコールをサポートしていない．

2.3.3. 保護機能

この節では，保護機能に関連する主な概念について説明する．この節の内容は，保護機能対応カーネルにのみ適用される．

(1) アクセス保護
保護機能対応カーネルは，処理単位が，許可されたカーネルオブジェクトに対して，許可された種別のアクセスを行うことのみを許し，それ以外のアクセスを防ぐアクセス保護機能を提供する．

アクセス制御の用語では，処理単位が主体（subject），カーネルオブジェクトが対象（object）ということになる．

(2) メモリオブジェクト
保護機能対応カーネルにおいては，メモリ領域をカーネルオブジェクトとして扱い，アクセス保護の対象とする．カーネルがアクセス保護の対象とする連続したメモリ領域を，メモリオブジェクト（memory object）と呼ぶ．メモリオブジェクトは，互いに重なりあうことはない．

メモリオブジェクトは，その先頭番地によって識別する．言い換えると，先頭番地がオブジェクト番号となる．

メモリオブジェクトの先頭番地とサイズには，ターゲットハードウェアでメモリ保護が実現できるように，ターゲット定義の制約が課せられる．

(3) 保護ドメイン
保護機能を提供するために用いるカーネルオブジェクトの集合を，保護ドメイン（protection domain）と呼ぶ．保護ドメインは，保護ドメイン ID と呼ぶ ID 番号によって識別する．

カーネルオブジェクトは，たかだか 1 つの保護ドメインに属する．処理単位は，いずれか 1 つの保護ドメインに属さなければならないのに対して，それ以外のカーネルオブジェクトは，いずれの保護ドメインにも属さないことができる．いずれの保護ドメインにも属さないカーネルオブジェクトを，無所属のカーネルオブジェクト（independent kernel object）と呼ぶ．

処理単位がカーネルオブジェクトにアクセスできるかどうかは，処理単位が属する保護ドメインによ

り決まるのが原則である．すなわち，カーネルオブジェクトに対するアクセス権は，処理単位ではなく，保護ドメイン単位で管理される．このことから，ある保護ドメインに属する処理単位がアクセスできることを，単に，その保護ドメインからアクセスできるという．

ただし，タスクのユーザスタック領域は，ターゲット定義での変更がない限りは，そのタスク（とカーネルドメインに属する処理単位）のみがアクセスできる（「2.11.6 ユーザタスクのユーザスタック領域」の節を参照）．これは，「処理単位がカーネルオブジェクトにアクセスできるかどうかは，処理単位が属する保護ドメインにより決まる」という原則の例外となっている．

デフォルトでは，保護ドメインに属するカーネルオブジェクトは，同じ保護ドメイン（とカーネルドメイン）のみからアクセスできる．また，無所属のカーネルオブジェクトは，すべての保護ドメインからアクセスできる．

(4) カーネルドメインとユーザドメイン

システムには，カーネルドメイン（kernel domain）と呼ばれる保護ドメインが 1 つ存在する．カーネルドメインに属する処理単位は，プロセッサの特権モードで実行される．また，すべてのカーネルオブジェクトに対して，すべての種別のアクセスを行うことが許可される．この仕様で，「ある保護ドメイン（またはタスク）のみからアクセスできる」といった場合でも，カーネルドメインドメインからはアクセスすることができる．

カーネルドメイン以外の保護ドメインを，ユーザドメイン（user domain）と呼ぶ．ユーザドメインに属する処理単位は，プロセッサの非特権モードで実行される．また，どのカーネルオブジェクトに対してどの種別のアクセスを行えるかを制限することができる．

ユーザドメインには，1 から連続する正の整数値の保護ドメイン ID が付与される．カーネルドメインの保護ドメイン ID は，TDOM_KERNEL（＝-1）である．

この仕様では，システムに登録できるユーザドメインの数は，32 個以下に制限する．これを超える数のユーザドメインを登録した場合には，コンフィギュレータがエラーを報告する．

【補足説明】

ユーザドメインは，システムコンフィギュレーションファイル中にユーザドメインの囲みを記述することで，カーネルに登録する（「2.12.3 保護ドメインの指定」の節を参照）．ユーザドメインを動的に生成する機能は，現時点では用意していない．

保護機能対応でないカーネルは，カーネルドメインのみをサポートしているとみなすこともできる．

【μITRON4.0/PX 仕様との関係】

μITRON4.0/PX 仕様のシステムドメイン（system domain）は，現時点ではサポートしない．システムドメインは，それに属する処理単位が，プロセッサの特権モードで実行され，カーネルオブジェクトに対するアクセスを制限することができる保護ドメインである．

(5) システムタスクとユーザタスク

カーネルドメインに属するタスクをシステムタスク（system task），ユーザドメインに属するタスクをユーザタスク（user task）と呼ぶ．

【補足説明】

特権モードで実行されるタスクをシステムタスク，非特権モードで実行されるタスクをユーザタスクと定義する方法もあるが，ユーザタスクであっても，サービスコールの実行中は特権モードで実行されるため，上記の定義とした．

μITRON4.0/PX 仕様のシステムドメインに属するタスクは，システムタスクと呼ぶことになる．

(6) アクセス許可パターン

あるカーネルオブジェクトに対するある種別のアクセスが，どの保護ドメインに属する処理単位に許可されているかを表現するビットパターンを，アクセス許可パターン（access permission pattern）と呼ぶ．アクセス許可パターンの各ビットは，1 つのユーザドメインに対応する．カーネルドメインには，すべてのアクセスが許可されているため，カーネルドメインに対応するビットは用意されていない．

アクセス許可パターンは，符号無し 32 ビット整数に定義されるデータ型（ACPTN）で保持し，値が 1 のビットに対応するユーザドメインにアクセスが許可されていることを表す．そのため，2 つのアクセス許可パターンのビット毎論理和（C 言語の”|”）を求めることで，アクセスを許可されているユーザドメインの和集合（union）を得ることができる．また，2 つのアクセス許可パターンのビット毎論理積（C 言語の”&”）を求めることで，アクセスを許可されているユーザドメインの積集合（intersection）を得ることができる．

アクセス許可パターンの指定に用いるために，指定したユーザドメインのみにアクセスを許可することを示すアクセス許可パターンを構成するマクロ（TACP）が用意されている．また，カーネルドメインのみにアクセスを許可することを示すアクセス許可パターンを表す定数（TACP_KERNEL）と，すべての保護ドメインにアクセスを許可することを示すアクセス許可パターンを表す定数（TACP_SHARED）が用意されている．

(7) アクセス許可ベクタ

カーネルオブジェクトに対するアクセスは，カーネルオブジェクトの種類毎に，通常操作 1，通常操作 2，管理操作，参照操作の 4 つの種別に分類されている．あるカーネルオブジェクトに対する 4 つの種別

のアクセスに関するアクセス許可パターンをひとまとめにしたものを，アクセス許可ベクタ（access permissionvector）と呼び，次のように定義されるデータ型（ACVCT）で保持する．

```
typedef struct acvct {
        ACPTN   acptn1;     /*  通常操作 1 のアクセス許可パターン  */
        ACPTN   acptn2;     /*  通常操作 2 のアクセス許可パターン  */
        ACPTN   acptn3;     /*  管理操作のアクセス許可パターン  */
        ACPTN   acptn4;     /*  参照操作のアクセス許可パターン  */
} ACVCT;
```

【補足説明】

カーネルオブジェクトの種類毎のアクセスの種別の分類については，「5.8 カーネルオブジェクトに対するアクセスの種別」の節を参照すること．

【μITRON4.0/PX 仕様との関係】

μITRON4.0/PX 仕様では，アクセス許可ベクタを，1 つまたは 2 つのアクセス許可パターンで構成することも許しているが，この仕様では 4 つで構成するものと決めている．

(8) サービスコールの呼出し方法

保護機能対応カーネルでは，サービスコールは，ソフトウェア割込みによって呼び出すのが基本である．サービスコール呼出しを通常の方法で記述した場合，ソフトウェア割込みによって呼び出すコードが生成される．

一般に，ソフトウェア割込みによるサービスコール呼出しはオーバヘッドが大きい．そのため，カーネルドメインに属する処理単位からは，関数呼出しによってサービスコールを呼び出すことで，オーバヘッドを削減することができる．そこで，カーネルドメインに属する処理単位から関数呼出しによってサービスコールを呼び出せるように，以下の機能が用意されている．

カーネルドメインに属する処理単位が実行する関数のみを含んだソースファイルでは，カーネルヘッダファイル（kernel.h）をインクルードする前に，TOPPERS_SVC_CALL をマクロ定義することで，サービスコール呼出しを通常の方法で記述した場合に，関数呼出しによって呼び出すコードが生成される．

また，カーネルドメインに属する処理単位が実行する関数と，ユーザドメインに属する処理単位が実行する関数の両方を含んだソースファイルでは，関数呼出しによってサービスコールを呼び出すための名称を作るマクロ（SVC_CALL）を用いることで，関数呼出しによって呼び出すコードが生成される．例えば，act_tsk を関数呼出しによって呼び出す場合には，次のように記述すればよい．

```
ercd = SVC_CALL(act_tsk)(tskid);
```

【補足説明】
拡張サービスコールを，関数呼出しによって呼び出す方法は用意されていない．カーネルドメインに属する処理単位が，関数呼出しによって，拡張サービスコールとして登録した関数を呼び出すことはできるが，その場合には，処理単位が呼び出した通常の関数であるとみなされ，拡張サービスコールであるとは扱われない．

2.3.4. マルチプロセッサ対応

この節では，マルチプロセッサ対応に関連する主な概念について説明する．この節の内容は，マルチプロセッサ対応カーネルにのみ適用される．

(1) クラス
マルチプロセッサに対応するために用いるカーネルオブジェクトの集合を，クラス（class）と呼ぶ．クラスは，クラス ID と呼ぶ ID 番号によって識別する．

カーネルオブジェクトは，いずれか 1 つのクラスに属するのが原則である．カーネルオブジェクトが属するクラスは，オブジェクトの登録時に決定し，登録後に変更することはできない．

【補足説明】
処理単位を実行するプロセッサを静的に決定する機能分散型のマルチプロセッサシステムでは，プロセッサ毎にクラスを設ける方法が典型的である．それに対して，対称型のマルチプロセッサシステムで，処理単位のマイグレーションを許す場合には，プロセッサ毎のクラスに加えて，どのプロセッサでも実行できるクラスを（システム中に 1 つまたは初期割付けプロセッサ毎に）設ける方法が典型的である．

カーネルオブジェクトはいずれか 1 つのクラスに属するという原則に関わらず，以下のオブジェクトはいずれのクラスにも属さない．

- オーバランハンドラ
- 拡張サービスコール
- グローバル初期化ルーチン
- グローバル終了処理ルーチン

マルチプロセッサ対応でないカーネルは，カーネルによって規定された 1 つのクラスのみをサポートしているとみなすこともできる．

(2) プロセッサ
たかだか 1 つの処理単位のみを同時に実行できるハードウェアの単位を，プロセッサ（processor）と呼ぶ．プロセッサは，プロセッサ ID と呼ぶ ID 番号によって識別する．

複数のプロセッサを持つシステム構成をマルチプロセッサ（multiprocessor）と呼び，同時に複数の処理単位を実行することができる．

システムの初期化時と終了時に特別な役割を果たすプロセッサを，マスタプロセッサ（master processor）と呼び，システムに 1 つ存在する．どのプロセッサをマスタプロセッサとするかは，ターゲット定義である．マスタプロセッサ以外のプロセッサを，スレーブプロセッサ（slave processor）と呼ぶ．なお，カーネル動作状態では，マスタプロセッサとスレーブプロセッサの振舞いに違いはない．

(3) 処理単位の割付けとマイグレーション

処理単位は，後述のマイグレーションが発生しない限りは，いずれか 1 つのプロセッサに割り付けられて実行される．処理単位を実行するプロセッサを，割付けプロセッサと呼ぶ．また，処理単位が登録時に割り付けられるプロセッサを，初期割付けプロセッサと呼ぶ．

処理単位によっては，処理単位の登録後に，割付けプロセッサを変更することが可能である．処理単位の登録後に割付けプロセッサを変更することを，処理単位のマイグレーション（migration）と呼ぶ．

割付けプロセッサを変更できる処理単位に対しては，処理単位を割り付けることができるプロセッサ（これを，割付け可能プロセッサと呼ぶ）を制限することができる．

(4) クラスの持つ属性とカーネルオブジェクト

タスクの初期割付けプロセッサや割付け可能プロセッサなど，カーネルオブジェクトをマルチプロセッサ上で実現する際に設定すべき属性は，そのカーネルオブジェクトが属するクラスによって定まる．

各クラスが持ち，それに属するカーネルオブジェクトに適用される属性は，次の通りである．

- 初期割付けプロセッサ
- 割付け可能プロセッサ（複数のプロセッサを指定可能，初期割付けプロセッサを含む）
- ATT_MOD／ATA_MOD において，オブジェクトモジュール中に含まれる標準のセクションが配置されるメモリリージョン
- オブジェクト生成に必要なメモリ領域（オブジェクトの管理ブロック，タスクのスタック領域やデータキューのデータキュー管理領域など）の配置場所
- その他の管理情報（ロック単位など）

使用できるクラスの ID 番号とその属性は，ターゲット定義である．

【仕様決定の理由】

クラスを導入することで，カーネルオブジェクト毎に上記の属性を設定できるようにしなかったのは，これらの属性をアプリケーション設計者が個別に設定するよりも，ターゲット依存部の実装者が有益な組み合わせをあらかじめ用意しておく方が良いと考えたためである．

(5) ローカルタイマ方式とグローバルタイマ方式

システム時刻の管理方式として，プロセッサ毎にシステム時刻を持つローカルタイマ方式と，システム全体で 1 つのシステム時刻を持つグローバルタイマ方式の 2 つの方式がある．どちらの方式を用いることができるかは，ターゲット定義である．

ローカルタイマ方式では，プロセッサ毎のシステム時刻は，それぞれのプロセッサが更新する．異なるプロセッサのシステム時刻を同期させる機能は，カーネルでは用意しない．

グローバルタイマ方式では，システム中の 1 つのプロセッサがシステム時刻を更新する．これを，システム時刻管理プロセッサと呼ぶ．どのプロセッサをシステム時刻管理プロセッサとするかは，ターゲット定義である．

【補足説明】

システム時刻管理プロセッサが，マスタプロセッサと一致している必要はない．

【未決定事項】

ローカルタイマ方式の場合に，プロセッサ毎に異なるタイムティックの周期を設定したい場合が考えられるが，現時点の実装ではサポートしておらず，TIC_NUME と TIC_DENO の扱いも未決定であるため，今後の課題とする．

2.3.5. その他

(1) オブジェクトモジュール

プログラムのオブジェクトコードとデータを含むファイルを，オブジェクトモジュール（object module）と呼ぶ．オブジェクトファイルとライブラリは，オブジェクトモジュールである．

(2) メモリリージョン

オブジェクトモジュールに含まれるセクションの配置対象となる同じ性質を持った連続したメモリ領域をメモリリージョン（memory region）と呼ぶ．

メモリリージョンは，文字列によって識別する．メモリリージョンを識別する文字列を，メモリリージョン名と呼ぶ．

【補足説明】

この仕様では，メモリ領域（memory area）という用語は，連続したメモリの範囲という一般的な意味で使っている．

(3) 標準のセクション

コンパイラに特別な指定をしない場合に出力するセクションを，標準のセクション（standard sections）と呼ぶ．また，ターゲット定義で，コンパイラが出力しないセクションを，標準のセクションと扱う場合もある．

(4) 保護ドメイン毎の標準セクション

保護機能対応カーネルにおいては，保護ドメイン毎に，標準のセクションを配置するためのセクションが登録される．また，無所属の標準のセクションを配置するためのセクションが登録される．これらのセクションを，保護ドメイン毎の標準セクションと呼ぶ（standard sections for each protection domain）．保護ドメイン毎の標準セクションのセクション名は，ターゲット定義で別に規定がない限りは，標準のセクション名と保護ドメイン名（カーネルドメインの場合は”kernel”，無所属の場合は”shared”）を”_”でつないだものとする．例えば，カーネルドメインの”.text”セクションのセクション名は，.text_kernel”とする．

2.4. 処理単位の種類と実行順序

2.4.1. 処理単位の種類

カーネルが実行を制御する処理単位の種類は次の通りである．

(a) タスク
  (a.1) タスク例外処理ルーチン
(b) 割込みハンドラ
  (b.1) 割込みサービスルーチン
  (b.2) 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
(e) 初期化ルーチン
(f) 終了処理ルーチン

ここで，タイムイベントハンドラとは，時間の経過をきっかけに起動される処理単位である周期ハンドラ，アラームハンドラ，オーバランハンドラの総称である．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，オーバランハンドラと拡張サービスコールをサポートしていない．ただし，オーバランハンドラ機能拡張パッケージを用いると，オーバランハンドラ機能を追加することができる．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，オーバランハンドラと拡張サービスコールをサポートしていない．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，タスク例外処理ルーチン，タイムイベントハンドラ，拡張サービスコールをサポートしていない．

2.4.2. 処理単位の実行順序

処理単位の実行順序を規定するために，ここでは，処理単位の優先順位を規定する．また，ディスパッチが起こるタイミングを規定するために，ディスパッチを行うカーネル内の処理であるディスパッチャの優先順位についても規定する．

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．タスク間では，高い優先度を持つ方が優先順位が高く，同じ優先度を持つタスク間では，先に実行できる状態となった方が優先順位が高い．詳しくは，「2.6.3 タスクのスケジューリング規則」の節を参照すること．

タスク例外処理ルーチンの優先順位は，例外が要求されたタスクと同じであるが，タスクよりも先に実行される．

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も高い．割込みハンドラ間では，高い割込み優先度を持つ方が優先順位が高く，同じ割込み優先度を持つ割込みハンドラ間では，先に実行開始された方が優先順位が高い．同じ割込み優先度を持つ割込みハンドラ間での実行開始順序は，この仕様では規定しない．詳しくは，「2.7.2 割込み優先度」の節を参照すること．

割込みサービスルーチンとタイムイベントハンドラの優先順位は，それを呼び出す割込みハンドラと同じである．

CPU 例外ハンドラの優先順位は，CPU 例外がタスクまたはタスク例外処理ルーチンで発生した場合には，ディスパッチャの優先順位と同じであるが，ディスパッチャよりも先に実行される．CPU 例外がその他の処理単位で発生した場合には，CPU 例外ハンドラの優先順位は，その処理単位の優先順位と同じであるが，その処理単位よりも先に実行される．

拡張サービスコールの優先順位は，それを呼び出した処理単位と同じであるが，それを呼び出した処理単位よりも先に実行される．

初期化ルーチンは，カーネルの動作開始前に，システムコンフィギュレーションファイル中に初期化ルーチンを登録する静的 API を記述したのと同じ順序で実行される．終了処理ルーチンは，カーネルの動作終了後に，終了処理ルーチンを登録する静的 API を記述したのと逆の順序で実行される．

マルチプロセッサ対応カーネルでは，初期化ルーチンには，クラスに属さないグローバル初期化ルーチンと，クラスに属するローカル初期化ルーチンがある．グローバル初期化ルーチンがマスタプロセッ

サで実行された後に，各プロセッサでローカル初期化ルーチンが実行される．また，終了処理ルーチンには，クラスに属さないグローバル終了処理ルーチンと，クラスに属するローカル終了処理ルーチンがある．ローカル終了処理ルーチンが各プロセッサで実行された後に，マスタプロセッサでグローバル終了処理ルーチンが実行される．

【仕様決定の理由】

終了処理ルーチンを，登録する静的 API を記述したのと逆順で実行するのは，終了処理は初期化の逆の順序で行うのがよいためである（システムコンフィギュレーションファイルを分割すると，終了処理ルーチンを登録する静的 API だけ逆順に記述するのは難しい）．

2.4.3. カーネル処理の不可分性

カーネルのサービスコール処理やディスパッチャ，割込みハンドラと CPU 例外ハンドラの入口処理と出口処理などのカーネル処理は不可分に実行されるのが基本である．実際には，カーネル処理の途中でアプリケーションが実行される場合はあるが，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，カーネル処理が不可分に実行された場合と同様に振る舞うのが原則である．これを，カーネル処理の不可分性という．

ただし，マルチプロセッサ対応カーネルにおいては，カーネル処理が実行されているプロセッサ以外のプロセッサから，カーネル処理の途中の状態が観測できる場合がある．具体的には，1 つのサービスコールにより複数のオブジェクトの状態が変化する場合に，一部のオブジェクトの状態のみが変化し，残りのオブジェクトの状態が変化していない過渡的な状態が観測できる場合がある．

【補足説明】

マルチプロセッサ対応でないカーネルでは，1 つのサービスコールにより複数のタスクが実行できる状態になる場合，新しく実行状態となるべきタスクへのディスパッチは，すべてのタスクの状態遷移が完了した後に行われる．例えば，低優先度のタスク A が発行したサービスコールにより，中優先度のタスク B と高優先度のタスク C がこの順で待ち解除される場合，タスク B とタスク C が待ち解除された後に，タスク C へのディスパッチが行われる．

マルチプロセッサ対応カーネルでは，上のことは，1 つのプロセッサ内では成り立つが，他のプロセッサに割り付けられたタスクに対しては成り立たない．例えば，プロセッサ 1 で低優先度のタスク A が実行されている時に，他のプロセッサ 2 で実行されているタスクが発行したサービスコールにより，プロセッサ 1 に割り付けられた中優先度のタスク B と高優先度のタスク C がこの順で待ち解除される場合，タスク C が待ち解除される前に，タスク B へディスパッチされる場合がある．

2.4.4. 処理単位を実行するプロセッサ

マルチプロセッサ対応カーネルでは，処理単位を実行するプロセッサ（割付けプロセッサ）は，その処理単位が属するクラスの初期割付けプロセッサと割付け可能プロセッサから，次のように決まる．

タスク，周期ハンドラ，アラームハンドラは，登録時に，属するクラスの初期割付けプロセッサに割り付けられる．また，割付けプロセッサを変更するサービスコール（mact_tsk／imact_tsk，mig_tsk，msta_cyc，msta_alm／imsta_alm）によって，割付けプロセッサを，クラスの割付け可能プロセッサのいずれかに変更することができる．

割込みハンドラ，CPU 例外ハンドラ，ローカル初期化ルーチン，ローカル終了処理ルーチンは，属するクラスの初期割付けプロセッサで実行される．クラスの割付け可能プロセッサの情報は用いられない．

割込みサービスルーチンは，属するクラスの割付け可能プロセッサのいずれか（オプション設定によりすべて）で実行される．クラスの初期割付けプロセッサの情報は用いられない．

以上を整理すると，次の表の通りとなる．この表の中で，「○」はその情報が使用されることを，「－」はその情報が使用されないことを示す．

	初期割付けプロセッサ	割付け可能プロセッサ
タスク（タスク例外処理ルーチンを含む）	○	○
割込みハンドラ	○	－
割込みサービスルーチン	－	○
周期ハンドラ	○	○
アラームハンドラ	○	○
CPU 例外ハンドラ	○	－
ローカル初期化ルーチン	○	－
ローカル終了処理ルーチン	○	－

オーバランハンドラ，拡張サービスコール，グローバル初期化ルーチン，グローバル終了処理ルーチンは，いずれのクラスにも属さない．オーバランハンドラは，オーバランを起こしたタスクの割付けプロセッサによって実行される．拡張サービスコールは，それを呼び出した処理単位の割付けプロセッサによって実行される．グローバル初期化ルーチンとグローバル終了処理ルーチンは，マスタプロセッサによって実行される．

2.5. システム状態とコンテキスト

2.5.1. カーネル動作状態と非動作状態

カーネルの初期化が完了した後，カーネルの終了処理が開始されるまでの間を，カーネル動作状態と呼ぶ．それ以外の状態，すなわちカーネルの初期化完了前（初期化ルーチンの実行中を含む）と終了処

理開始後（終了処理ルーチンの実行中を含む）を，カーネル非動作状態と呼ぶ．プロセッサは，カーネル動作状態かカーネル非動作状態のいずれかの状態を取る．

カーネル非動作状態では，原則として，NMI を除くすべての割込みがマスクされる．

カーネル非動作状態では，システムインタフェースレイヤの API とカーネル非動作状態を参照するサービスコール（sns_ker）のみを呼び出すことができる．カーネル非動作状態で，その他のサービスコールを呼び出した場合の動作は，保証されない．

マルチプロセッサ対応カーネルでは，プロセッサ毎に，カーネル動作状態かカーネル非動作状態のいずれかの状態を取る．

2.5.2. タスクコンテキストと非タスクコンテキスト

処理単位が実行される環境（用いるスタック領域やプロセッサの動作モードなど）をコンテキストと呼ぶ．

カーネル動作状態において，処理単位が実行されるコンテキストは，タスクコンテキストと非タスクコンテキストに分類される．

タスク（タスク例外処理ルーチンを含む）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．また，タスクコンテキストから呼び出した拡張サービスコール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．

割込みハンドラ（割込みサービスルーチンおよびタイムイベントハンドラを含む）と 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．また，非タスクコンテキストから呼び出した拡張サービスコール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で実行される処理単位は，別に規定がない限り，タスクのスタック領域を用いて実行される．非タスクコンテキストで実行される処理単位は，別に規定がない限り，非タスクコンテキスト用スタック領域を用いて実行される．

タスクコンテキストからは，非タスクコンテキスト専用のサービスコールを呼び出すことはできない．逆に，非タスクコンテキストからは，タスクコンテキスト専用のサービスコールを呼び出すことはできない．いずれも，呼び出した場合には E_CTX エラーとなる．

2.5.3. カーネルの振舞いに影響を与える状態

カーネル動作状態において，プロセッサは，カーネルの振舞いに影響を与える状態として，次の状態

を持つ.

- 全割込みロックフラグ（全割込みロック状態と全割込みロック解除状態）
- CPU ロックフラグ（CPU ロック状態と CPU ロック解除状態）
- 割込み優先度マスク（割込み優先度マスク全解除状態と全解除でない状態）
- ディスパッチ禁止フラグ（ディスパッチ禁止状態とディスパッチ許可状態）

これらの状態は，それぞれ独立な状態である．すなわち，プロセッサは上記の状態の任意の組合せを取ることができ，それぞれの状態を独立に変化させることができる．

2.5.4. 全割込みロック状態と全割込みロック解除状態

プロセッサは，NMI を除くすべての割込みをマスクするための全割込みロックフラグを持つ．全割込みロックフラグがセットされた状態を全割込みロック状態，クリアされた状態を全割込みロック解除状態と呼ぶ．すなわち，全割込みロック状態では，NMI を除くすべての割込みがマスクされる．

全割込みロック状態では，システムインタフェースレイヤの API とカーネル非動作状態を参照するサービスコール（sns_ker），カーネルを終了するサービスコール（ext_ker）のみを呼び出すことができ，その他のサービスコールを呼び出すことはできない．全割込みロック状態で，その他のサービスコールを呼び出した場合の動作は，保証されない．また，全割込みロック状態では，実行中の処理単位からリターンしてはならない．リターンした場合の動作は保証されない．

マルチプロセッサ対応カーネルでは，プロセッサ毎に，全割込みロックフラグを持つ．すなわち，プロセッサ毎に，全割込みロック状態か全割込みロック解除状態のいずれかの状態を取る．

2.5.5. CPU ロック状態と CPU ロック解除状態

プロセッサは，カーネル管理の割込み（「2.7.7 カーネル管理外の割込み」の節を参照）をすべてマスクするための CPU ロックフラグを持つ．CPU ロックフラグがセットされた状態を CPU ロック状態，クリアされた状態を CPU ロック解除状態と呼ぶ．すなわち，CPU ロック状態では，すべてのカーネル管理の割込みがマスクされ，ディスパッチが保留される．

NMI 以外にカーネル管理外の割込みを設けない場合には，全割込みロックフラグと CPU ロックフラグの機能は同一となるが，両フラグは独立に存在する．

CPU ロック状態で呼び出すことができるサービスコールは次の通り．

- システムインタフェースレイヤの API
- loc_cpu／iloc_cpu，unl_cpu／iunl_cpu
- unl_spn／iunl_spn（マルチプロセッサ対応カーネルのみ）
- dis_int，ena_int
- sns_yyy，xsns_yyy（CPU 例外ハンドラからのみ）
- get_utm
- ext_tsk，ext_ker
- cal_svc（保護機能対応カーネルのみ）

CPU ロック状態で，その他のサービスコールを呼び出した場合には，E_CTX エラーとなる．

マルチプロセッサ対応カーネルでは，プロセッサ毎に，CPU ロックフラグを持つ．すなわち，プロセッサ毎に，CPU ロック状態か CPU ロック解除状態のいずれかの状態を取る．

【補足説明】
マルチプロセッサ対応カーネルにおいて，あるプロセッサが CPU ロック状態にある間は，そのプロセッサにおいてのみ，すべてのカーネル管理の割込みがマスクされ，ディスパッチが保留される．それに対して他のプロセッサにおいては，割込みはマスクされず，ディスパッチも起こるため，CPU ロック状態を使って他のプロセッサで実行される処理単位との排他制御を実現することはできない．

2.5.6. 割込み優先度マスク

プロセッサは，割込み優先度を基準に割込みをマスクするための割込み優先度マスクを持つ．割込み優先度マスクが TIPM_ENAALL（＝0）の時は，いずれの割込み要求もマスクされない．この状態を割込み優先度マスク全解除状態と呼ぶ．割込み優先度マスクが TIPM_ENAALL（＝0）以外の時は，割込み優先度マスクと同じかそれより低い割込み優先度を持つ割込みはマスクされ，ディスパッチは保留される．この状態を割込み優先度マスクが全解除でない状態と呼ぶ．

割込み優先度マスクが全解除でない状態では，別に規定がない限りは，自タスクを広義の待ち状態に遷移させる可能性のあるサービスコールを呼び出すことはできない．呼び出した場合には，E_CTX エラーとなる．

マルチプロセッサ対応カーネルでは，プロセッサ毎に，割込み優先度マスクを持つ．

2.5.7. ディスパッチ禁止状態とディスパッチ許可状態

プロセッサは，ディスパッチを保留するためのディスパッチ禁止フラグを持つ．ディスパッチ禁止フラグがセットされた状態をディスパッチ禁止状態，クリアされた状態をディスパッチ許可状態と呼ぶ．

すなわち，ディスパッチ禁止状態では，ディスパッチは保留される．

ディスパッチ禁止状態では，別に規定がない限りは，自タスクを広義の待ち状態に遷移させる可能性のあるサービスコールを呼び出すことはできない．呼び出した場合には，E_CTX エラーとなる．

マルチプロセッサ対応カーネルでは，プロセッサ毎に，ディスパッチ禁止フラグを持つ．すなわち，プロセッサ毎に，ディスパッチ禁止状態かディスパッチ許可状態のいずれかの状態を取る．

【補足説明】
マルチプロセッサ対応カーネルにおいて，あるプロセッサがディスパッチ禁止状態にある間は，そのプロセッサにおいてのみ，ディスパッチが保留される．それに対して他のプロセッサにおいては，ディスパッチが起こるため，ディスパッチ禁止状態を使って他のプロセッサで実行されるタスクとの排他制御を実現することはできない．

2.5.8. ディスパッチ保留状態

非タスクコンテキストの実行中，CPU ロック状態，割込み優先度マスクが全解除でない状態，ディスパッチ禁止状態では，ディスパッチが保留される．これらの状態を総称して，ディスパッチ保留状態と呼ぶ．

マルチプロセッサ対応カーネルでは，プロセッサ毎に，ディスパッチ保留状態かそうでない状態のいずれかの状態を取る．

【補足説明】
全割込みロック状態はカーネルが管理しておらず，ディスパッチが保留されることをカーネルが保証できないため，ディスパッチ保留状態に含めていない．

2.5.9. カーネル管理外の状態

全割込みロック状態，カーネル管理外の割込みハンドラ実行中（「2.7.7 カーネル管理外の割込み」の節を参照），カーネル管理外の CPU 例外ハンドラ実行中（「2.8.4 カーネル管理外の CPU 例外」の節を参照）を総称して，カーネル管理外の状態と呼ぶ．

それぞれの節で規定する通り，カーネル管理外の状態では，システムインタフェースレイヤの API と sns_ker, ext_ker のみ（カーネル管理外の CPU 例外ハンドラからは，それに加えて xsns_dpn と xsns_xpn）を呼び出すことができ，その他のサービスコールを呼び出すことはできない．カーネル管理外の状態から，その他のサービスコールを呼び出した場合の動作は，保証されない．

カーネル管理外の状態では，少なくとも，カーネル管理の割込みはマスクされている．カーネル管理外の割込み（の一部）もマスクされている場合もある．保護機能対応カーネルでは，カーネル管理外の状態になるのは，特権モードで実行している間に限られる．

2.5.10. 処理単位の開始・終了とシステム状態

各処理単位が実行開始されるシステム状態の条件（実行開始条件），各処理単位の実行開始時にカーネルによって行われるシステム状態の変更処理（実行開始時処理），各処理単位からのリターン前（または終了前）にアプリケーションが設定しておくべきシステム状態（リターン前または終了前），各処理単位からのリターン時（または終了時）にカーネルによって行われるシステム状態の変更処理（リターン時処理または終了時処理）は，次の表の通りである．

	CPU ロックフラグ	割込み優先度マスク	ディスパッチ禁止フラグ
タスク			
実行開始条件	解除	全解除	許可
実行開始時処理	そのまま	そのまま	そのまま
終了前	原則解除(*1)	原則全解除(*1)	原則許可(*1)
終了時処理	解除する	全解除する	許可する
タスク例外処理ルーチン			
実行開始条件	解除	全解除	任意
実行開始時処理	そのまま	そのまま	そのまま
リターン前	原則解除(*1)	原則全解除(*1)	元に戻す
リターン時処理	解除する	全解除する	元に戻す(*4)
カーネル管理の割込みハンドラ/割込みサービスルーチン/タイムイベントハンドラ】			
実行開始条件	解除	自優先度より低い	任意
実行開始時処理	そのまま	自優先度に(*2)	そのまま
リターン前	原則解除(*1)	変更不可(*3)	変更不可(*3)
リターン時処理	解除する	元に戻す(*5)	そのまま
CPU 例外ハンドラ			
実行開始条件	任意	任意	任意
実行開始時処理	そのまま(*6)	そのまま	そのまま
リターン前	原則元に(*1)	変更不可(*3)	変更不可(*3)
リターン時処理	元に戻す	元に戻す(*5)	そのまま
拡張サービスコール			
実行開始条件	任意	任意	任意
実行開始時処理	そのまま	そのまま	そのまま
リターン前	任意	任意	任意
リターン時処理	そのまま	そのまま	そのまま

この表の中で「原則(*1)」とは，処理単位からのリターン前（または終了前）に，アプリケーションが指定された状態に設定しておくことが原則であるが，この原則に従わなくても，リターン時（または終了時）にカーネルによって状態が設定されるため，支障がないことを意味する．

「自優先度に(*2)」 とは，割込みハンドラと割込みサービスルーチンの場合にはそれを要求した割込みの割込み優先度，周期ハンドラとアラームハンドラの場合にはタイマ割込みの割込み優先度，オーバランハンドラの場合にはオーバランタイマ割込みの割込み優先度に変更することを意味する．

「変更不可(*3)」 とは，その処理単位中で，そのシステム状態を変更する API が用意されていないことを示す．

保護機能対応カーネルでは，タスク例外処理ルーチンからのリターン時にディスパッチ禁止フラグを元に戻す処理(*4)は，タスクにディスパッチ禁止フラグの変更を許可している場合にのみ行われる．カーネルは，ディスパッチ禁止フラグの元の状態をユーザスタック上に保存する．アプリケーションがユーザスタック上に保存されたディスパッチ禁止フラグの状態を書き換えた場合，タスク例外処理ルーチンからのリターン時には，書き換えた後のディスパッチ禁止フラグの状態に変更される（すなわち，元に戻されるとは限らない）．

また，タスクにディスパッチ禁止フラグの変更を許可していない場合で，タスク例外処理ルーチン中で拡張サービスコールを用いてディスパッチ禁止フラグを変更した場合，カーネルは元の状態に戻さない．このことから，タスク例外処理ルーチンからの終了前に，ディスパッチ禁止フラグを元の状態に戻すのは，アプリケーションの責任とする．

【補足説明】
マルチプロセッサ対応カーネルにおいて，タスクがタスク例外処理ルーチンを実行中にマイグレーションされた場合，マイグレーション先のプロセッサにおいて，割込み優先度マスクとディスパッチ禁止フラグが元に戻される．

【仕様決定の理由】
保護機能対応カーネルにおいて，タスク例外処理ルーチンからのリターン時にディスパッチ禁止フラグを元に戻す処理(*4)が，タスクにディスパッチ禁止フラグの変更を許可している場合にのみ行われるのは，タスクがユーザスタック上の状態を書き換えることで，許可していない状態変更を起こせてしまうことを防止するためである．

割込みハンドラや CPU 例外ハンドラで，その処理単位中で割込み優先度マスクを変更する API が用意されていないにもかかわらず，処理単位からのリターン時に元の状態に戻す(*5)のは，プロセッサによっては，割込み優先度マスクがステータスレジスタ等に含まれており，API を用いずに変更できてしま

う場合があるためである．

CPU 例外ハンドラの実行開始時には，CPU ロックフラグは変更されない(*6)ことから，CPU ロック状態で CPU 例外が発生した場合，CPU 例外ハンドラの実行開始直後は CPU ロック状態となっている．CPU ロック状態で CPU 例外が発生した場合，起動される CPU 例外ハンドラはカーネル管理外の CPU 例外ハンドラであり（xsns_dpn，xsns_xpn とも true を返す），CPU 例外ハンドラ中で iunl_cpu を呼び出して CPU ロック状態を解除しようとした場合の動作は保証されない．ただし，保証されないにも関わらず iunl_cpu を呼び出した場合も考えられるため，リターン時には元に戻すこととしている．

2.6. タスクの状態遷移とスケジューリング規則

2.6.1. 基本的なタスク状態

カーネルに登録したタスクは，実行できる状態，休止状態，広義の待ち状態のいずれかの状態を取る．また，実行できる状態と広義の待ち状態を総称して，起動された状態と呼ぶ．さらに，タスクをカーネルに登録していない仮想的な状態を，未登録状態と呼ぶ．

(a) 実行できる状態（runnable）
タスクを実行できる条件が，プロセッサが使用できるかどうかを除いて，揃っている状態．実行できる状態は，さらに，実行状態と実行可能状態に分類される．

(a.1) 実行状態（running）
タスクが実行されている状態．または，そのタスクの実行中に，割込みまたは CPU 例外により非タスクコンテキストの実行が開始され，かつ，タスクコンテキストに戻った後に，そのタスクの実行を再開するという状態．

(a.2) 実行可能状態（ready）
タスク自身は実行できる状態にあるが，それよりも優先順位の高いタスクが実行状態にあるために，そのタスクが実行されない状態．

(b) 休止状態（dormant）
タスクが実行すべき処理がない状態．タスクの実行を終了した後，次に起動するまでの間は，タスクは休止状態となっている．タスクが休止状態にある時には，タスクの実行を再開するための情報（実行再開番地やレジスタの内容など）は保存されていない．

(c) 広義の待ち状態（blocked）
タスクが，処理の途中で実行を止められている状態．タスクが広義の待ち状態にある時には，タスクの実行を再開するための情報（実行再開番地やレジスタの内容など）は保存されており，タスクが実行

を再開する時には，広義の待ち状態に遷移する前の状態に戻される．広義の待ち状態は，さらに，（狭義の）待ち状態，強制待ち状態，二重待ち状態に分類される．

(c.1) （狭義の）待ち状態（waiting）
タスクが何らかの条件が揃うのを待つために，自ら実行を止めている状態．

(c.2) 強制待ち状態（suspended）
他のタスクによって，強制的に実行を止められている状態．ただし，自タスクを強制待ち状態にすることも可能である．

(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．すなわち，タスクが何らかの条件が揃うのを待つために自ら実行を止めている時に，他のタスクによって強制的に実行を止められている状態．

単にタスクが「待ち状態である」といった場合には，二重待ち状態である場合を含み，「待ち状態でない」といった場合には，二重待ち状態でもないことを意味する．また，単にタスクが「強制待ち状態である」といった場合には，二重待ち状態である場合を含み，「強制待ち状態でない」といった場合には，二重待ち状態でもないことを意味する．

(d) 未登録状態（non-existent）
タスクをカーネルに登録していない仮想的な状態．タスクの生成前と削除後は，タスクは未登録状態にあるとみなす．

カーネルによっては，これらのタスク状態以外に，過渡的な状態が存在する場合がある．過渡的な状態については，「2.6.6 ディスパッチ保留状態で実行中のタスクに対する強制待ち」の節を参照すること．

【TOPPERS/ASP カーネルにおける規定】
ASP カーネルでは，タスクが未登録状態になることはない．また，上記のタスク状態以外の過渡的な状態になることもない．ただし，動的生成機能拡張パッケージでは，タスクが未登録状態になる．

【TOPPERS/FMP カーネルにおける規定】
FMP カーネルでは，タスクが未登録状態になることはない．上記のタスク状態以外の過渡的な状態として，タスクが強制待ち状態［実行継続中］になることがある．詳しくは，「2.6.6 ディスパッチ保留状態で実行中のタスクに対する強制待ち」の節を参照すること．

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，タスクが未登録状態になることはない．また，上記のタスク状態以外の過渡的な状態になることもない．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，タスクが広義の待ち状態と未登録状態になることはない．また，上記のタスク状態以外の過渡的な状態になることもない．

2.6.2. タスクの状態遷移

タスクの状態遷移を図 2-2 に示す．

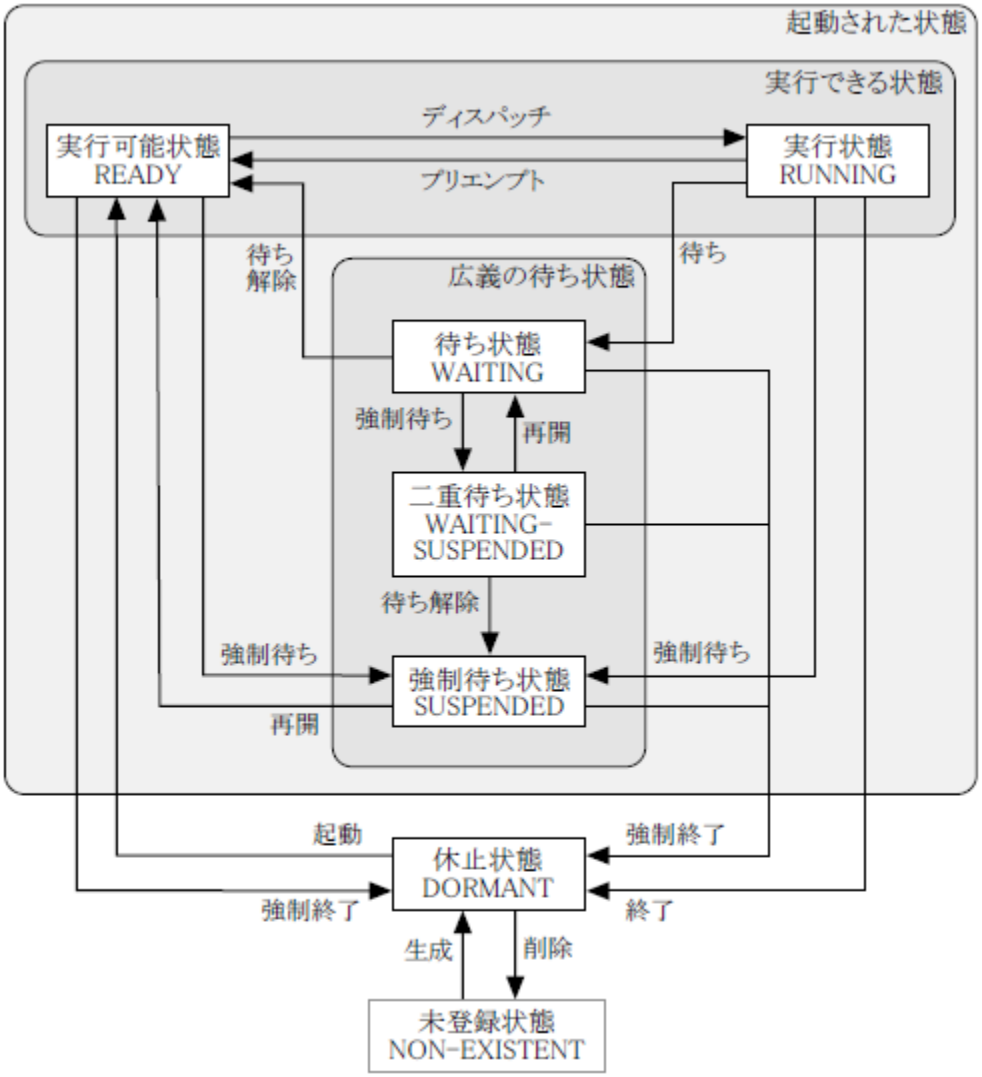

図 2-2 タスクの状態遷移

未登録状態のタスクをカーネルに登録することを，タスクを生成する（create）という．生成されたタスクは，休止状態に遷移する．また，タスク生成時の属性指定により，生成と同時にタスクを起動し，実行できる状態にすることもできる．逆に，登録されたタスクを未登録状態に遷移させることを，タスクを削除する（delete）という．

休止状態のタスクを，実行できる状態にすることを，タスクを起動する（activate）という．起動されたタスクは，実行できる状態になる．逆に，起動された状態のタスクを，休止状態（または未登録状態）に遷移させることを，タスクを終了する（terminate）という．

実行できる状態になったタスクは，まずは実行可能状態に遷移するが，そのタスクの優先順位が実行状態のタスクよりも高い場合には，ディスパッチ保留状態でない限りはただちにディスパッチが起こり，実行状態へ遷移する．この時，それまで実行状態であったタスクは実行可能状態に遷移する．この時，実行状態に遷移したタスクは，実行可能状態に遷移したタスクをプリエンプトしたという．逆に，実行可能状態に遷移したタスクは，プリエンプトされたという．

タスクを待ち解除するとは，タスクが待ち状態（二重待ち状態を除く）であれば実行できる状態に，二重待ち状態であれば強制待ち状態に遷移させることをいう．また，タスクを強制待ちから再開するとは，タスクが強制待ち状態（二重待ち状態を除く）であれば実行できる状態に，二重待ち状態であれば待ち状態に遷移させることをいう．

【補足説明】
タスクの実行開始とは，タスクが起動された後に最初に実行される（実行状態に遷移する）時のことをいう．

2.6.3. タスクのスケジューリング規則

実行できるタスクは，優先順位の高いものから順に実行される．すなわち，ディスパッチ保留状態でない限りは，実行できるタスクの中で最も高い優先順位を持つタスクが実行状態となり，他は実行可能状態となる．

タスクの優先順位は，タスクの優先度とタスクが実行できる状態になった順序から，次のように定まる．優先度の異なるタスクの間では，優先度の高いタスクが高い優先順位を持つ．優先度が同一のタスクの間では，先に実行できる状態になったタスクが高い優先順位を持つ．すなわち，同じ優先度を持つタスクは，FCFS（First Come First Served）方式でスケジューリングされる．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を変更することも可能である．

最も高い優先順位を持つタスクが変化した場合には，ディスパッチ保留状態でない限りはただちにディスパッチが起こり，最も高い優先順位を持つタスクが実行状態となる．ディスパッチ保留状態においては，実行状態のタスクは切り換わらず，最も高い優先順位を持つタスクは実行可能状態にとどまる．
マルチプロセッサ対応カーネルでは，プロセッサ毎に，上記のスケジューリング規則を適用して，タスクスケジューリングを行う．すなわち，プロセッサがディスパッチ保留状態でない限りは，そのプロセッサに割り付けられた実行できるタスクの中で最も高い優先順位を持つタスクが実行状態となり，他は実行可能状態となる．そのため，実行状態のタスクは，プロセッサ毎に存在する．

2.6.4. 待ち行列と待ち解除の順序

タスクが待ち解除される順序の管理のために，待ち状態のタスクがつながれているキューを，待ち行列と呼ぶ．また，タスクが同期・通信オブジェクトの待ち行列につながれている場合に，そのオブジェクトを，タスクの待ちオブジェクトと呼ぶ．

待ち行列にタスクをつなぐ順序には，FIFO 順とタスクの優先度順がある．どちらの順序でつなぐかは，待ち行列毎に規定される．多くの待ち行列において，どちらの順序でつなぐかを，オブジェクト属性により指定できる．

FIFO 順の待ち行列においては，新たに待ち状態に遷移したタスクは待ち行列の最後につながれる．それに対してタスクの優先度順の待ち行列においては，新たに待ち状態に遷移したタスクは，優先度の高い順に待ち行列につながれる．同じ優先度のタスクが待ち行列につながれている場合には，新たに待ち状態に遷移したタスクが，同じ優先度のタスクの中で最後につながれる．

待ち解除の条件がタスクによって異なる場合には，待ち行列の先頭のタスクは待ち解除の条件を満たさないが，後方のタスクが待ち解除の条件を満たす場合がある．このような場合の振舞いとして，次の 2 つのケースがある．どちらの振舞いをするかは，待ち行列毎に規定される．

(a) 待ち解除の条件を満たしたタスクの中で，待ち行列の前方につながれたものから順に待ち解除される．すなわち，待ち行列の前方に待ち解除の条件を満たさないタスクがあっても，後方のタスクが待ち解除の条件を満たしていれば，先に待ち解除される．

(b) タスクの待ち解除は，待ち行列につながれている順序で行われる．すなわち，待ち行列の前方に待ち解除の条件を満たさないタスクがあると，後方のタスクが待ち解除の条件を満たしても，待ち解除されない．

ここで，(b)の振舞いをする待ち行列においては，待ち行列につながれたタスクの強制終了，タスク優先度の変更（待ち行列がタスクの優先度順の場合のみ），待ち状態の強制解除が行われた場合に，タスクの待ち解除が起こることがある．

具体的には，これらの操作により新たに待ち行列の先頭になったタスクが，待ち解除の条件を満たしていれば，ただちに待ち解除される．さらに，この待ち解除により新たに待ち行列の先頭になったタスクに対しても，同じ処理が繰り返される．

2.6.5. タスク例外処理マスク状態と待ち禁止状態

保護機能対応カーネルにおいて，ユーザタスクについては特権モードで実行している間（特権モードを実行している間に，実行可能状態や広義の待ち状態になっている場合を含む．また，サービスコール

を呼び出して，実行可能状態や広義の待ち状態になっている場合も含む．タスクの実行開始前は含まない)，システムタスクについては拡張サービスコールを実行している間（拡張サービスコールを実行している間に，実行可能状態や広義の待ち状態になっている場合を含む）は，タスク例外処理ルーチンの実行は開始されない．これらの状態を，タスク例外処理マスク状態と呼ぶ．

タスクは，タスク例外処理マスク状態である時に，基本的なタスク状態と重複して，待ち禁止状態になることができる．

待ち禁止状態とは，タスクが待ち状態に入ることが一時的に禁止された状態である．待ち禁止状態にあるタスクが，サービスコールを呼び出して待ち状態に遷移しようとした場合，サービスコールはE_RLWAI エラーとなる．

タスクを待ち禁止状態に遷移させるサービスコールは，対象タスクがタスク例外処理マスク状態である場合に，対象タスクを待ち禁止状態に遷移させる．その後，タスクがタスク例外処理マスク状態でなくなる時点（ユーザタスクについては特権モードから戻る時点，システムタスクについて拡張サービスコールからリターンする時点）で，待ち禁止状態が解除される．また，タスクの待ち禁止状態を解除するサービスコールによっても，待ち禁止状態を解除することができる．

【仕様決定の理由】

タスク例外処理ルーチンでは，タスクの本体のための例外処理（例えば，タスクに対して終了要求があった時の処理）を行うことを想定しており，タスクから呼び出した拡張サービスコールのための例外処理を行うことは想定していない．そのため，拡張サービスコールを実行している間にタスク例外処理が要求された場合に，すぐにタスク例外処理ルーチンを実行すると，拡張サービスコールのための例外処理が行われないことになる．

また，ユーザタスクの場合には，特権モードを実行中にタスク例外処理ルーチンを実行すると，システムスタックに情報を残したまま非特権モードに戻ることになる．この状態で，タスク例外処理ルーチンから大域脱出すると，システムスタック上に不要な情報が残ってしまう．

これらの理由から，タスクが拡張サービスコールを実行している間は，タスク例外処理マスク状態とし，タスク例外処理ルーチンの実行を開始しないこととする．さらに，ユーザタスクについては，特権モードを実行している間（拡張サービスコールを実行している間を含む）を，タスク例外処理マスク状態とする．

対象タスクに，タスク例外処理ルーチンをすみやかに実行させたい場合には，タスク例外処理の要求に加えて，待ち状態の強制解除を行う（必要に応じて，強制待ち状態からの再開も行う）．保護機能対応でないカーネルにおいては，この方法により，対象タスクが正常に待ち解除されるのを待たずに，タスク例外処理ルーチンを実行させることができる．

それに対して，保護機能対応カーネルにおいては，対象タスクがタスク例外処理マスク状態で実行している間は，タスク例外処理ルーチンの実行が開始されない．そのため，対象タスクに対して待ち状態の強制解除を行っても，その後に対象タスクが待ち状態に入ると，タスク例外処理ルーチンがすみやかに実行されないことになる．

待ち禁止状態は，この問題を解決するために導入したものである．タスク例外処理の要求（ras_tex／iras_tex）に加えて，待ち禁止状態への遷移（dis_wai／idis_wai）と待ち状態の強制解除（rel_wai／irel_wai）をこの順序で行うことで，対象タスクが正常に待ち解除されるのを待たずに，タスク例外処理ルーチンを実行させることができる．

タスク例外処理マスク状態を，ユーザタスクについても拡張サービスコールを実行している間とせず，特権モードで実行している間とした理由は，拡張サービスコールを実行している間とした場合に次のような問題があるためである．

ユーザタスクが，ソフトウェア割込みにより自タスクを待ち状態に遷移させるサービスコールを呼び出した直後に割込みが発生し，その割込みハンドラの中で iras_tex，idis_wai，irel_wai が呼び出されると，この時点では待ち解除もされず待ち禁止状態にもならないために，割込みハンドラからのリターン後に待ち状態に入ってしまう．ソフトウェア割込みによりすべての割込みが禁止されないターゲットプロセッサでは，ソフトウェア割込みの発生とサービスコールの実行を不可分にできないため，このような状況を防ぐことができない．

なお，拡張サービスコールは，待ち状態に入るサービスコールから E_RLWAI が返された場合には，実行中の処理を取りやめて，E_RLWAI を返値としてリターンするように実装すべきである．

【μITRON4.0 仕様，μITRON4.0/PX 仕様との関係】
待ち禁止状態は，μITRON4.0 仕様にはない概念であり，μITRON4.0/PX 仕様で導入された．ただし，μITRON4.0/PX 仕様では，タスクの待ち状態を強制解除するサービスコールが，タスクを待ち禁止状態へ遷移させる機能も持つこととしている．その結果μITRON4.0/PX 仕様は，待ち状態を強制解除するサービスコールの仕様において，μITRON4.0 仕様との互換性がなくなっている．

この仕様では，待ち状態の強制解除と待ち禁止状態への遷移を別々のサービスコールで行うこととした．これにより，待ち状態を強制解除するサービスコールの仕様が，μITRON4.0 仕様と互換になっている．一方，μITRON4.0/PX 仕様とは互換性がない．

2.6.6. ディスパッチ保留状態で実行中のタスクに対する強制待ち

ディスパッチ保留状態において，実行状態のタスクを強制待ち状態へ遷移させるサービスコールを呼

び出した場合，実行状態のタスクの切換えは，ディスパッチ保留状態が解除されるまで保留される．

この間，それまで実行状態であったタスクは，実行状態と強制待ち状態の間の過渡的な状態にあると考える．この状態を，強制待ち状態［実行継続中］と呼ぶ．一方，ディスパッチ保留状態が解除された後に実行すべきタスクは，実行可能状態にとどまる．

タスクが強制待ち状態［実行継続中］にある時に，ディスパッチ保留状態が解除されると，ただちにディスパッチが起こり，タスクは強制待ち状態に遷移する．

過渡的な状態も含めたタスクの状態遷移を図 2-3 に示す．

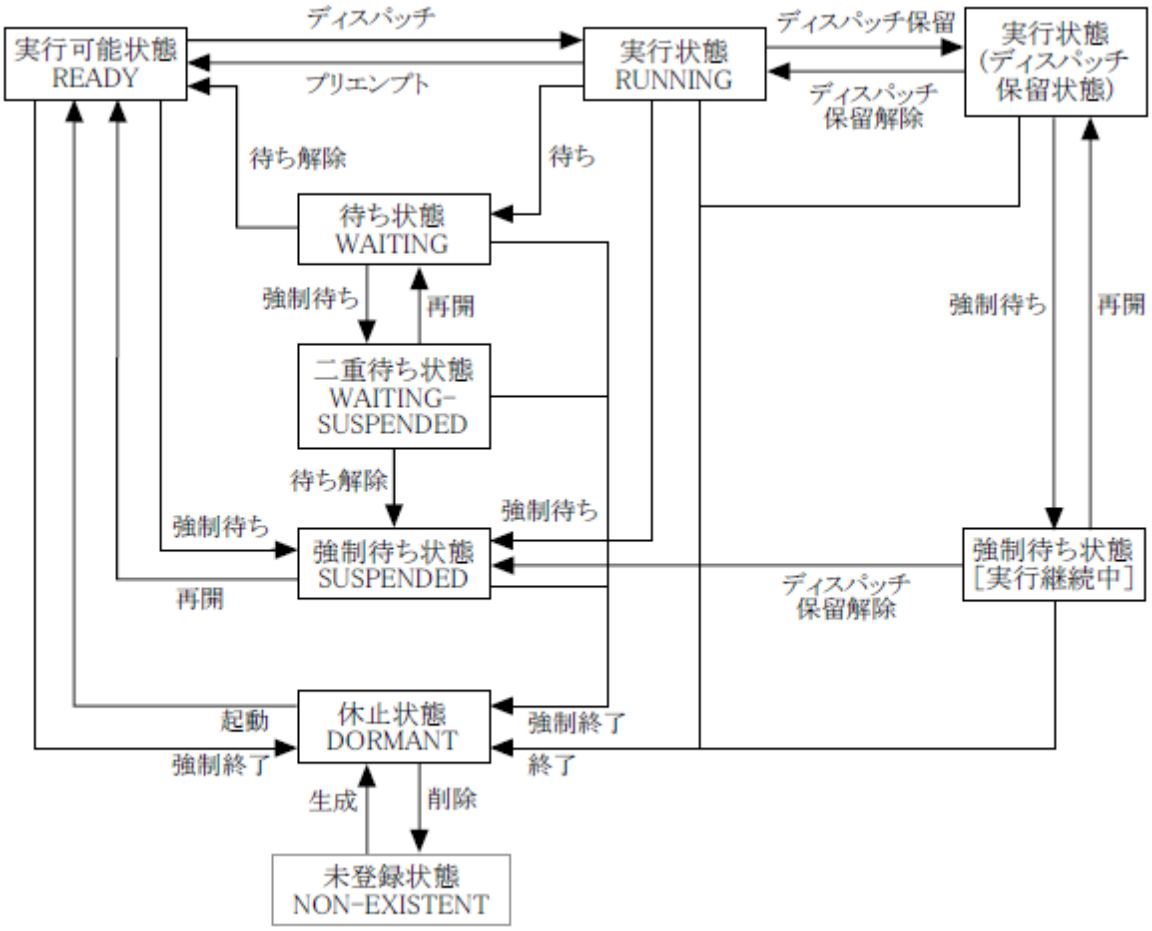

図 2-3 過渡的な状態も含めたタスクの状態遷移

タスクが強制待ち状態［実行継続中］である時の扱いは次の通りである．

(a) プロセッサを占有して実行を継続する．

強制待ち状態［実行継続中］のタスクは，プロセッサを占有して，そのまま継続して実行される．

(b) 実行状態のタスクに関する情報を参照するサービスコールでは，実行状態であるものと扱う．

実行状態のタスクに関する情報を参照するサービスコール（get_tid／iget_tid，get_did，sns_tex）では，強制待ち状態［実行継続中］のタスクが，それを実行するプロセッサにおいて実行状態のタスクであるものと扱う．具体的には，強制待ち状態［実行継続中］のタスクが実行されている時に get_tid／iget_tid を発行すると，そのタスクの ID 番号を参照する．また，get_did を発行するとそのタスクが属する保護ドメインの ID 番号を，sns_tex を発行するとそのタスクのタスク例外処理禁止フラグを参照する．

(c) その他のサービスコールでは，強制待ち状態であるものと扱う．

その他のサービスコールでは，強制待ち状態［実行継続中］のタスクは，強制待ち状態であるものと扱う．

なお，TOPPERS 新世代カーネルでは，ディスパッチ保留状態において，実行状態のタスクを強制終了させるサービスコールはサポートしていない．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，ディスパッチ保留状態において実行状態のタスクを強制待ち状態へ遷移させるサービスコールはサポートしていないため，タスクが強制待ち状態［実行継続中］になることはない．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，ディスパッチ保留状態において実行状態のタスクを強制待ち状態へ遷移させるサービスコールを，他のプロセッサから呼び出すことができるため，タスクが強制待ち状態［実行継続中］になる場合がある．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，ディスパッチ保留状態において実行状態のタスクを強制待ち状態へ遷移させるサービスコールはサポートしていないため，タスクが強制待ち状態［実行継続中］になることはない．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，タスクが広義の待ち状態になることはないため，タスクが強制待ち状態［実行継続中］になることもない．

2.6.7. 制約タスク

制約タスク（restricted task）は，複数のタスクでスタック領域を共有することによるメモリ使用量の削減を目的に，通常のタスクに対して，広義の待ち状態を持たないなどの機能制限を加えたものである．具体的には，制約タスクには以下の機能制限がある．

(a) 広義の待ち状態に入ることができない．

(b) サービスコールにより優先度を変更することができない.
(c) 対象優先度の中の先頭のタスクが制約タスクである場合には，タスクの優先順位の回転（rot_rdq／irot_rdq）を行うことができない.
(d) マルチプロセッサ対応カーネルでは，割付けプロセッサを変更することができない.

制約タスクに対して，機能制限により使用できなくなったサービスコールを呼び出した場合には，E_NOSPT エラーとなる．E_NOSPT エラーが返ることに依存している場合を除いては，制約タスクを通常のタスクに置き換えることができる.

【未決定事項】
現状では，制約タスクの優先度を変更するサービスコールは設けていないが，制約タスクが，自タスクの優先度を，起動時優先度（SSP カーネルにおいては，実行時優先度）と同じかそれよりも高い値に変更することは許してもよい．ただし，優先度の変更後は，同じ優先度内で最高優先順位としなければならないため，chg_pri とは振舞いが異なることになる．自タスクの優先度を起動時優先度と同じかそれよりも高い値に変更するサービスコールを設けるかどうかは，今後の課題である.

【TOPPERS/ASP カーネルにおける規定】
ASP カーネルでは，制約タスクをサポートしていない．ただし，制約タスク拡張パッケージを用いると，制約タスクの機能を追加することができる.

【TOPPERS/FMP カーネルにおける規定】
FMP カーネルでは，制約タスクをサポートしていない.

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，制約タスクをサポートしていない.

【TOPPERS/SSP カーネルにおける規定】
SSP カーネルでは，制約タスクのみをサポートする．そのため，すべてのタスクと非タスクコンテキストがスタック領域を共有することができ，すべての処理単位で同一のスタック領域を使用している．このスタック領域を，共有スタック領域と呼ぶ.

【μITRON4.0 仕様との関係】
制約タスクは，μITRON4.0 仕様の自動車制御プロファイルで導入された機能である．この仕様における制約タスクは，μITRON4.0 仕様の制約タスクよりも機能制限が少なくなっている.

2.7. 割込み処理モデル

TOPPERS 新世代カーネルにおける割込み処理のモデルは，TOPPERS 標準割込み処理モデルに準拠

している．

TOPPERS 標準割込み処理モデルの概念図を図 2-4 に示す．この図は，割込み処理モデルの持つすべての機能が，ハードウェア（プロセッサおよび割込みコントローラ）で実現されているとして描いた概念図である．実際のハードウェアで不足している機能については，カーネル内の割込み処理のソフトウェアで実現される．

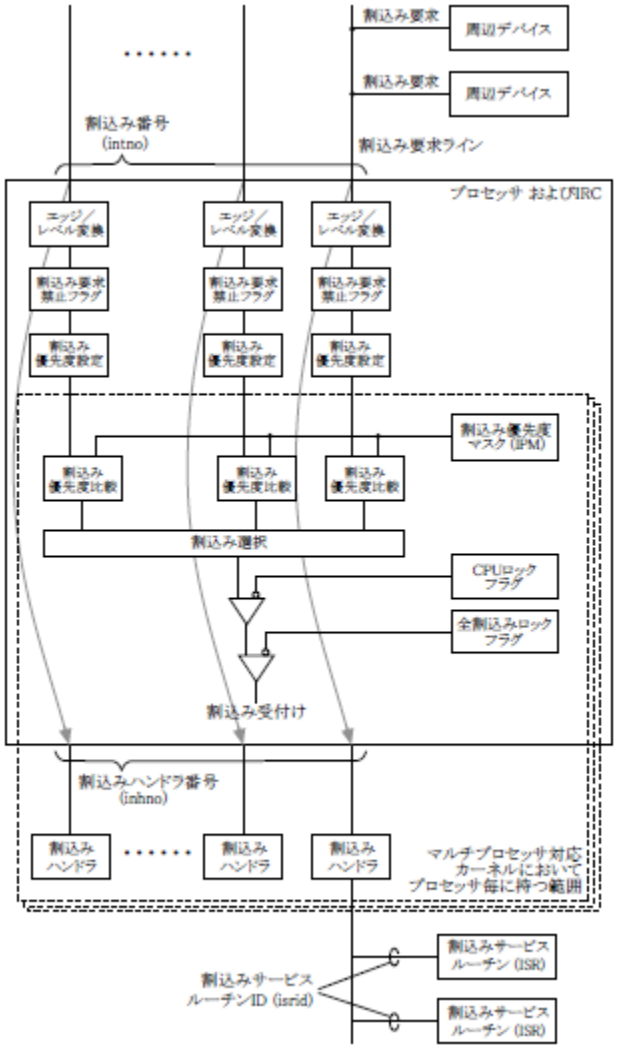

図 2-4 TOPPERS 標準割込み処理モデルの概念図

【μITRON4.0 仕様との関係】

割込み処理モデルは，μITRON4.0 仕様から大幅に拡張している．

2.7.1. 割込み処理の流れ

周辺デバイス（以下，デバイスと呼ぶ）からの割込み要求は，割込みコントローラ（IRC）を経由して，

プロセッサに伝えられる．デバイスから割込みコントローラに割込み要求を伝えるための信号線を，割込み要求ラインと呼ぶ．一般には，1 つの割込み要求ラインに，複数のデバイスからの割込み要求が接続される．

プロセッサは，デバイスからの割込み要求を受け付ける条件が満たされた場合，割込み要求を受け付ける．受け付けた割込み要求が，カーネル管理の割込みである場合には，カーネル内の割込みハンドラの入口処理（割込み入口処理）を経由して，カーネル内の割込みハンドラを実行する．

カーネル内の割込みハンドラは，アプリケーションが割込み要求ラインに対して登録した割込みサービスルーチン（ISR）を呼び出す．割込みサービスルーチンは，プロセッサの割込みアーキテクチャや割込みコントローラに依存せず，割込みを要求したデバイスのみに依存して記述するのが原則である．1 つの割込み要求ラインに対して複数のデバイスが接続されることから，1 つの割込み要求ライン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．

ただし，カーネルが標準的に用意している割込みハンドラで対応できない特殊なケースも考えられる．このような場合に対応するために，アプリケーションが用意した割込みハンドラをカーネルに登録することもできる．

カーネルが用いるタイマデバイスからの割込み要求の場合，カーネル内の割込みハンドラにより，タイムイベントの処理が行われる．具体的には，タイムアウト処理等が行われることに加えて，アプリケーションが登録したタイムイベントハンドラが呼び出される．

なお，受け付けた割込み要求に対して，割込みサービスルーチンも割込みハンドラも登録していない場合の振舞いは，ターゲット定義である．

2.7.2. 割込み優先度

割込み要求は，割込み処理の優先順位を指定するための割込み優先度を持つ．プロセッサは，割込み優先度マスクの現在値よりも高い割込み優先度を持つ割込み要求のみを受け付ける．逆に言うと，割込み優先度マスクの現在値と同じか，それより低い割込み優先度を持つ割込みは，マスクされる．

プロセッサは，割込み要求を受け付けると，割込み優先度マスクを，受け付けた割込み要求の割込み優先度に設定する（ただし，受け付けた割込みが NMI である場合には例外とする）．また，割込み処理からのリターンにより，割込み優先度マスクを，割込み要求を受け付ける前の値に戻す．

これらのことから，他の方法で割込みをマスクしていない限り，ある割込み要求の処理中は，それと同じかそれより低い割込み優先度を持つ割込み要求は受け付けられず，それより高い割込み優先度を持つ割込み要求は受け付けられることになる．つまり，割込み優先度は，多重割込みを制御するためのも

のと位置付けることができる．それに対して，同時に発生している割込み要求の中で，割込み優先度の高い割込み要求が先に受け付けられるとは限らない．

割込み優先度は，PRI 型で表現し，値が小さいほど優先度が高いものとするが，優先度に関する原則には従わず，-1 から連続した負の値を用いる．

割込み優先度の段階数は，ターゲット定義である．プロセッサが割込み優先度マスクを実現するための機能を持たないか，実現するために大きいオーバヘッドを生じる場合には，ターゲット定義で，割込み優先度の段階数を 1 にする（すなわち，多重割込みを許さない）場合がある．

【仕様決定の理由】
割込み優先度に-1 から連続した負の値を用いるのは，割込み優先度とタスク優先度を比較できるようになることと，いずれの割込みもマスクしない割込み優先度マスクの値を 0 にできるためである．

2.7.3. 割込み要求ラインの属性

各割込み要求ラインは，以下の属性を持つ．なお，1 つの割込み要求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る場合，それらの割込み要求は同一の属性を持つ．それらの割込み要求に別々の属性を設定することはできない．

(1) 割込み要求禁止フラグ
割込み要求ライン毎に，割込みをマスクするための割込み要求禁止フラグを持つ．割込み要求禁止フラグをセットすると，その割込み要求ラインによって伝えられる割込み要求はマスクされる．

プロセッサが割込み要求禁止フラグを実現するための機能を持たないか，実現するために大きいオーバヘッドを生じる場合には，ターゲット定義で，割込み要求禁止フラグをサポートしない場合がある．また，プロセッサの持つ割込み要求禁止フラグの機能がこの仕様に合致しない場合には，ターゲット定義で，割込み要求禁止フラグをサポートしないか，振舞いが異なるものとする場合がある．

アプリケーションが，割込み要求禁止フラグを動的にセット／クリアする機能を用いると，次の理由でソフトウェアの再利用性が下がる可能性があるため，注意が必要である．プロセッサによっては，この割込み処理モデルに合致した割込み要求禁止フラグの機能を実現できない場合がある．また，割込み要求禁止フラグをセットすることで，複数のデバイスからの割込みがマスクされる場合がある．ソフトウェアの再利用性を上げるためには，あるデバイスからの割込みのみをマスクしたい場合には，そのデバイス自身の機能を使ってマスクを実現すべきである．

(2) 割込み優先度
割込み要求ライン毎に，割込み優先度を設定することができる．割込み要求の割込み優先度とは，そ

の割込み要求を伝える割込み要求ラインに対して設定された割込み優先度のことである．

(3) トリガモード

割込み要求ラインに対する割込み要求が，レベルトリガであるかエッジトリガであるかを設定することができる．エッジトリガの場合には，さらに，ポジティブエッジトリガかネガティブエッジトリガか両エッジトリガかを設定できる場合もある．また，レベルトリガの場合には，ローレベルトリガかハイレベルトリガかを設定できる場合もある．

プロセッサがトリガモードを設定するための機能を持たないか，設定するために大きいオーバヘッドを生じる場合には，ターゲット定義で，トリガモードの設定をサポートしない場合がある．

属性が設定されていない割込み要求ラインに対しては，割込み要求禁止フラグがセットされ，割込み要求はマスクされる．また，割込み要求禁止フラグをクリアすることもできない．

2.7.4. 割込みを受け付ける条件

NMI 以外の割込み要求は，次の 4 つの条件が揃った場合に受け付けられる．

(a) 割込み要求ラインに対する割込み要求禁止フラグがクリアされていること
(b) 割込み要求ラインに設定された割込み優先度が，割込み優先度マスクの現在値よりも高い（優先度の値としては小さい）こと
(c) 全割込みロックフラグがクリアされていること
(d) 割込み要求がカーネル管理の割込みである場合には，CPU ロックフラグがクリアされていること

これらの条件が揃った割込み要求が複数ある場合に，どの割込み要求が最初に受け付けられるかは，この仕様では規定しない．すなわち，割込み優先度の高い割込み要求が先に受け付けられるとは限らない．

2.7.5. 割込み番号と割込みハンドラ番号

割込み要求ラインを識別するための番号を，割込み番号と呼ぶ．割込み番号は，符号無しの整数型である INTNO 型で表し，ターゲットハードウェアの仕様から決まる自然な番号付けを基本として，ターゲット定義で付与される．そのため，1 から連続した正の値であるとは限らない．

それに対して，アプリケーションが用意した割込みハンドラをカーネルに登録する場合に，割込みハンドラの登録対象となる割込みを識別するための番号を，割込みハンドラ番号と呼ぶ．割込みハンドラ番号は，符号無しの整数型である INHNO 型で表し，ターゲットハードウェアの仕様から決まる自然な番号付けを基本として，ターゲット定義で付与される．そのため，1 から連続した正の値であるとは限らない．

割込みハンドラ番号は，割込み番号と 1 対 1 に対応するのが基本である（両者が一致する場合が多い）．

ただし，割込みを要求したデバイスが割込みベクタを生成してプロセッサに渡すアーキテクチャなど

では，割込み番号と割込みハンドラ番号の対応を，カーネルが管理していない場合がある．そこで，ターゲット定義で，割込み番号に対応しない割込みハンドラ番号や，割込みハンドラ番号に対応しない割込み番号を設ける場合もある．ただし，割込みサービスルーチンの登録対象にできる割込み番号は，割込みハンドラ番号との 1 対 1 の対応関係をカーネルが管理しているもののみである．

2.7.6. マルチプロセッサにおける割込み処理

この節では，マルチプロセッサにおける割込み処理について説明する．この節の内容は，マルチプロセッサ対応カーネルにのみ適用される．

マルチプロセッサ対応カーネルでは，TOPPERS 標準割込み処理モデルの構成要素中で，図 2-4 の破線に囲まれた部分はプロセッサ毎に持ち，それ以外の部分はシステム全体で 1 つのみ持つ．すなわち，全割込みロックフラグ，CPU ロックフラグ，割込み優先度マスクはプロセッサ毎に持つのに対して，割込み要求ラインおよびその属性（割込み要求禁止フラグ，割込み優先度，トリガモード）はシステム全体で共通に持つ．

割込み番号は，割込み要求ラインを識別するための番号であることから，割込み要求ラインが複数のプロセッサに接続されている場合でも，1 つの割込み要求ラインには 1 つの割込み番号を付与する．逆に，複数のプロセッサが同じ種類のデバイスを持っている場合でも，別のデバイスからの割込み要求ラインには異なる割込み番号を付与する（図 2-5）．図 2-5 において，ローカル IRC は個々のプロセッサに対する割込みを制御するための回路であり，グローバル IRC はデバイスからの割込みをプロセッサに分配するための回路である．グローバル IRC は，必ず備わっているとは限らない．

割込み要求禁止フラグは，この仕様上はシステム全体で共通に持つこととしているが，実際のターゲットハードウェア（特に，グローバル IRC を備えていないもの）では，プロセッサ毎に持っている場合がある．そのため，ターゲット定義で，あるプロセッサで割込み要求禁止フラグを動的にセット／クリアしても，他のプロセッサに対しては割込みがマスク／マスク解除されない場合があるものとする．

複数のプロセッサに接続された割込み要求ラインに対して登録された割込みサービスルーチンは，それらのプロセッサのいずれによっても実行することができる．ただし，その内のどのプロセッサで割込みサービスルーチンを実行するかは，割込みサービスルーチンが属するクラスの割付け可能プロセッサにより決定される（「2.4.4 処理単位を実行するプロセッサ」の節を参照）．

割込みサービスルーチンが属するクラスの割付け可能プロセッサは，登録対象の割込み要求ラインが接続されたプロセッサの集合に含まれていなければならない．また，同一の割込み要求ラインに対して登録する割込みサービスルーチンは，同一のクラスに属していなければならない．

それに対して，割込みハンドラはプロセッサ毎に登録する．そのため，同じ割込み要求に対応する割込みハンドラであっても，プロセッサ毎に異なる割込みハンドラ番号を付与する（図 2-5）．割込みハンドラが属するクラスの初期割付けプロセッサは，割込みが要求されるプロセッサと一致していなければ

ならない.

【補足説明】

マルチプロセッサ対応カーネルにおける割込み番号の付与方法は，複数のプロセッサに接続された割込み要求ラインに対しては，割込み番号の上位ビットを 0 とし，1 つのプロセッサのみに接続された割込み要求ラインに対しては，割込み番号の上位ビットに，接続されたプロセッサの ID 番号を含める方法を基本とする．また，割込みハンドラ番号の付与方法は，割込みハンドラ番号の上位ビットに，その割込みハンドラを実行するプロセッサの ID 番号を含める方法を基本とする（図 2-5）.

1 つのプロセッサのみに接続された割込み要求ラインに対して登録された割込みサービスルーチンは，そのプロセッサのみを割付け可能プロセッサとするクラスに属していなければならない.

【使用上の注意】

複数のプロセッサで実行することができる割込みサービスルーチンは，それらのプロセッサのいずれかで実行されるものと設定した場合でも，複数回の割込み要求により，異なるプロセッサで同時に実行される可能性がある.

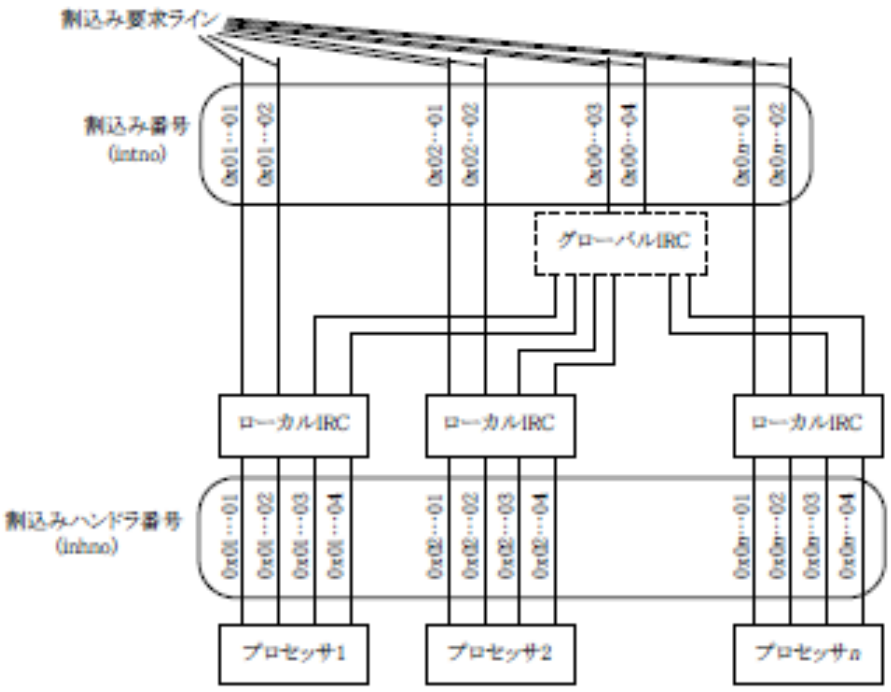

図 2-5 マルチプロセッサ対応カーネルにおける割込み番号と割込みハンドラ番号

2.7.7. カーネル管理外の割込み

高い割込み応答性を求められるアプリケーションでは，カーネル内で割込みをマスクすることにより，割込み応答性の要求を満たせなくなる場合がある．このような要求に対応するために，カーネル内では，

ある割込み優先度（これを，TMIN_INTPRI と書く）よりも高い割込み優先度を持つ割込みをマスクしないこととしている．TMIN_INTPRI を固定するか設定できるようにするか，設定できるようにする場合の設定方法は，ターゲット定義である．

TMIN_INTPRI よりも高い割込み優先度を持ち，カーネル内でマスクしない割込みを，カーネル管理外の割込みと呼ぶ．また，カーネル管理外の割込みによって起動される割込みハンドラを，カーネル管理外の割込みハンドラと呼ぶ．NMI は，カーネル管理外の割込みとして扱う．NMI 以外にカーネル管理外の割込みを設けるか（設けられるようにするか）どうかは，ターゲット定義である．

それに対して，TMIN_INTPRI と同じかそれよりも低い割込み優先度を持つ割込みをカーネル管理の割込み，カーネル管理の割込みによって起動される割込みハンドラをカーネル管理の割込みハンドラと呼ぶ．

カーネル管理外の割込みハンドラは，カーネル内の割込み入口処理を経由せずに実行するのが基本である．ただし，すべての割込みで同じ番地に分岐するプロセッサでは，カーネル内の割込み入口処理を全く経由せずにカーネル管理外の割込みハンドラを実行することができず，入口処理の一部分を経由してカーネル管理外の割込みハンドラが実行されることになる．

カーネル管理外の割込みハンドラが実行開始される時のシステム状態とコンテキスト，割込みハンドラの終了時に行われる処理，割込みハンドラの記述方法は，ターゲット定義である．カーネル管理外の割込みハンドラからは，システムインタフェースレイヤの API と sns_ker，ext_ker のみを呼び出すことができ，その他のサービスコールを呼び出すことはできない．カーネル管理外の割込みハンドラから，その他のサービスコールを呼び出した場合の動作は，保証されない．

2.7.8. カーネル管理外の割込みの設定方法

カーネル管理外の割込みの設定方法は，ターゲット定義で，次の 3 つの方法のいずれかが採用される．

(a.1) NMI 以外にカーネル管理外の割込みを設けない
(a.2) カーネル構築時に特定の割込みをカーネル管理外にすると決める

これら場合には，カーネル管理外とする割込みはカーネル構築時（ターゲット依存部の実装時やカーネルのコンパイル時）に決まるため，カーネル管理外とする割込みをアプリケーション側で設定する必要はない．ここで，カーネル管理外とされた割込みに対して，カーネルの API により割込みハンドラを登録できるかと，割込み要求ラインの属性を設定できるかは，ターゲット定義である．割込みハンドラを登録できる場合には，それを定義する API において，カーネル管理外であることを示す割込みハンドラ属性（TA_NONKERNEL）を指定する．また，割込み要求ラインの属性を設定できる場合には，設定する割込み優先度を TMIN_INTPRI よりも高い値とする．

(b) カーネル管理外とする割込みをアプリケーションで設定できるようにする

この場合には，カーネル管理外とする割込みの設定は，次の方法で行う．まずカーネル管理外とする割込みハンドラを定義する API において，カーネル管理外であることを示す割込みハンドラ属性（TA_NONKERNEL）を指定する．また，カーネル管理外とする割込みの割込み要求ラインに対して設定する割込み優先度を，TMIN_INTPRI よりも高い値とする．

いずれの場合にも，カーネル管理の割込みの割込み要求ラインに対して設定する割込み優先度は，TMIN_INTPRI より高い値であってはならない．また，カーネル管理外の割込みに対して，割込みサービスルーチンを登録することはできない．

2.8. CPU 例外処理モデル

プロセッサが検出する CPU 例外の種類や，CPU 例外検出時のプロセッサの振舞いは，プロセッサによって大きく異なる．そのため，CPU 例外ハンドラをターゲットハードウェアに依存せずに記述することは，少なくとも現時点では困難である．そこでこの仕様では，CPU 例外の処理モデルを厳密に標準化するのではなく，ターゲットハードウェアに依存せずに決められる範囲で規定する．

2.8.1. CPU 例外処理の流れ

アプリケーションは，プロセッサが検出する CPU 例外の種類毎に，CPU 例外ハンドラを登録することができる．プロセッサが CPU 例外の発生を検出すると，カーネル内の CPU 例外ハンドラの入口処理（CPU 例外入口処理）を経由して，発生した CPU 例外に対して登録した CPU 例外ハンドラが呼び出される．

CPU 例外ハンドラの登録対象となる CPU 例外を識別するための番号を，CPU 例外ハンドラ番号と呼ぶ．CPU 例外ハンドラ番号は，符号無しの整数型である EXCNO 型で表し，ターゲットハードウェアの仕様から決まる自然な番号付けを基本として，ターゲット定義で付与される．そのため，1 から連続した正の値であるとは限らない．

マルチプロセッサ対応カーネルでは，異なるプロセッサで発生する CPU 例外は，異なる CPU 例外であると扱う．すなわち，同じ種類の CPU 例外であっても，異なるプロセッサの CPU 例外には異なる CPU 例外ハンドラ番号を付与し，プロセッサ毎に CPU 例外ハンドラを登録する．CPU 例外ハンドラが属するクラスの初期割付けプロセッサは，CPU 例外が発生するプロセッサと一致していなければならない．

CPU 例外ハンドラにおいては，CPU 例外が発生した状態からのリカバリ処理を行う．どのようなリカバリ処理を行うかは，一般には CPU 例外の種類やそれが発生したコンテキストおよび状態に依存するが，大きく次の 4 つの方法が考えられる．

(a) カーネルに依存しない形で CPU 例外の原因を取り除き，実行を継続する．

(b) CPU 例外を起こしたタスクよりも優先度の高いタスクを起動または待ち解除し，そのタスクでリカバリ処理を行う（例えば，CPU 例外を起こしたタスクを強制終了し，再度起動する）．ただし，CPU 例外を起こしたタスクが最高優先度の場合には，この方法でリカバリ処理を行うことはできない（リカバリ処理を行うタスクを最高優先度とし，タスクの起動または待ち解除後に優先順位を回転させることで，リカバリ処理を行える可能性があるが，CPU 例外を起こしたタスクが制約タスクの場合には適用できないなど，推奨できる方法ではない）．

(c) CPU 例外を起こしたタスクにタスク例外処理を要求し，タスク例外処理ルーチンでリカバリ処理を行う（例えば，CPU 例外を起こしたタスクを終了する）．

(d) システム全体に対してリカバリ処理を行う（例えば，システムを再起動する）．

この中で(a)と(d)の方法は，カーネルの機能を必要としないため，CPU 例外が発生したコンテキストおよび状態に依存せずに常に行える．それに対して(b)と(c)の方法は，CPU 例外ハンドラからそのためのサービスコールを呼び出せることが必要であり，それが行えるかどうかは，CPU 例外が発生したコンテキストおよび状態に依存する．

なお，発生した CPU 例外に対して，CPU 例外ハンドラを登録していない場合の振舞いは，ターゲット定義である．

【使用上の注意】

CPU 例外入口処理で CPU 例外が発生し，それを処理するための CPU 例外ハンドラの入口処理で同じ原因で CPU 例外が発生すると，CPU 例外が繰り返し発生し，アプリケーションが登録した CPU 例外ハンドラまで処理が到達しない状況が考えられる．このような状況が発生するかどうかはターゲットによるが，これが許容できない場合には，CPU 例外入口処理を経由せずに，アプリケーションが用意した CPU 例外ハンドラを直接実行するようにしなければならない．

【補足説明】

マルチプロセッサ対応カーネルにおける CPU 例外ハンドラ番号の付与方法は，CPU 例外ハンドラ番号の上位ビットに，その CPU 例外が発生するプロセッサの ID 番号を含める方法を基本とする．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様では，CPU 例外からのリカバリ処理の方法については，記述されていない．

2.8.2. CPU 例外ハンドラから呼び出せるサービスコール

CPU 例外ハンドラからは，CPU 例外発生時のディスパッチ保留状態を参照するサービスコール（xsns_dpn）と，CPU 例外発生時にタスク例外処理ルーチンを実行開始できない状態であったかを参照

するサービスコール（xsns_xpn）を呼び出すことができる．

xsns_dpn は，CPU 例外がタスクコンテキストで発生し，そのタスクがディスパッチできる状態であった場合に false を返す．xsns_dpn が false を返した場合，その CPU 例外ハンドラから，非タスクコンテキストから呼び出せるすべてのサービスコールを呼び出すことができ，(b)の方法によるリカバリ理が可能である．ただし，CPU 例外を起こしたタスクが最高優先度の場合には，この方法でリカバリ処理を行うことはできない．

xsns_xpn は，CPU 例外がタスクコンテキストで発生し，そのタスクがタスク例外処理ルーチンを実行できる状態であった場合に false を返す．xsns_xpn が false を返した場合，その CPU 例外ハンドラから，非タスクコンテキストから呼び出せるすべてのサービスコールを呼び出すことができ，©の方法によるリカバリ処理が可能である．

xsns_dpn と xsns_xpn のいずれのサービスコールも true を返した場合，その CPU 例外ハンドラからは，xsns_dpn と xsns_xpn に加えて，システムインタフェースレイヤの API と sns_ker，ext_ker のみを呼び出すことができ，その他のサービスコールを呼び出すことはできない．いずれのサービスコールも true を返したにもかかわらず，その他のサービスコールを呼び出した場合の動作は，保証されない．この場合には，(b)と(c)の方法によるリカバリ処理は行うことはできず，(a)または(d)の方法によるリカバリ処理を行うしかないことになる．

【μITRON4.0 仕様との関係】
CPU 例外ハンドラで行える操作に関しては，μITRON4.0 仕様を見直し，全面的に修正した．

2.8.3. エミュレートされた CPU 例外ハンドラ

エラーコードによってアプリケーションに通知できないエラーをカーネルが検出した場合に，アプリケーションが登録したエラー処理を，カーネルが呼び出す場合がある．この場合に，カーネルが検出するエラーを CPU 例外と同等に扱うものとし，エミュレートされた CPU 例外と呼ぶ．また，エラー処理のためのプログラムを CPU 例外ハンドラと同等に扱うものとし，エミュレートされた CPU 例外ハンドラと呼ぶ．

具体的には，エミュレートされた CPU 例外ハンドラに対しても CPU 例外ハンドラ番号が付与され，CPU 例外ハンドラと同じ方法で登録できる．また，エミュレートされた CPU 例外ハンドラからも，CPU 例外ハンドラから呼び出せるサービスコールを呼び出すことができ，CPU 例外ハンドラと同様のリカバリ処理を行うことができる．

【μITRON4.0 仕様との関係】
エミュレートされた CPU 例外および CPU 例外ハンドラは，μITRON4.0 仕様に定義されていない概

念である．

2.8.4. カーネル管理外の CPU 例外

カーネル内のクリティカルセクションの実行中（これを，カーネル実行中と呼ぶ），全割込みロック状態，CPU ロック状態，カーネル管理外の割込みハンドラ実行中のいずれかで発生した CPU 例外を，カーネル管理外の CPU 例外と呼ぶ．また，それによって起動される CPU 例外ハンドラを，カーネル管理外の CPU 例外ハンドラと呼ぶ．さらに，カーネル管理外の CPU 例外ハンドラ実行中に発生した CPU 例外も，カーネル管理外の CPU 例外とする．

それに対して，カーネル管理外の CPU 例外以外の CPU 例外をカーネル管理の CPU 例外，カーネル管理の CPU 例外によって起動される CPU 例外ハンドラをカーネル管理の CPU 例外ハンドラと呼ぶ．
カーネル管理外の CPU 例外ハンドラからは，システムインタフェースレイヤの API と sns_ker，ext_ker，xsns_dpn，xsns_xpn のみを呼び出すことができ，その他のサービスコールを呼び出すことはできない．カーネル管理外の CPU 例外ハンドラから，その他のサービスコールを呼び出した場合の動作は，保証されない．

カーネル管理外の CPU 例外ハンドラにおいては，xsns_dpn と xsns_xpn のいずれのサービスコールも true を返す．そのため，カーネル管理外の CPU 例外からは，(a)または(d)の方法によるリカバリ処理しか行えない．

【補足説明】
カーネル管理外の CPU 例外は，カーネル管理外の割込みと異なり，特定の CPU 例外をカーネル外とするわけではない．同じ CPU 例外であっても，CPU 例外が起こる状況によって，カーネル管理となる場合とカーネル管理外となる場合がある．

2.9. システムの初期化と終了

2.9.1. システム初期化手順

システムのリセット後，最初に実行するプログラムを，スタートアップモジュールと呼ぶ．スタートアップモジュール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のが基本であるが，スタートアップモジュールで行うべき処理を明確にするために，カーネルの配布パッケージの中に，標準のスタートアップモジュールが用意されている．

標準のスタートアップモジュールは，プロセッサのモードとスタックポインタ等の初期化，NMI を除くすべての割込みのマスク（全割込みロック状態と同等の状態にする），ターゲットシステム依存の初期化フックの呼出し，非初期化データセクション（bss セクション）のクリア，初期化データセクション（data

セクション）の初期化，ソフトウェア環境（ライブラリなど）依存の初期化フックの呼出しを行った後，カーネルの初期化処理へ分岐する．ここで呼び出すターゲットシステム依存の初期化フックでは，リセット後に速やかに行うべき初期化処理を行うことが想定されている．

マルチプロセッサ対応カーネルでは，すべてのプロセッサがスタートアップモジュールを実行し，カーネルの初期化処理へ分岐する．ただし，共有リソースの初期化処理（非初期化データセクションのクリア，初期化データセクションの初期化，ソフトウェア環境依存の初期化フックの呼出しなど）は，マスタプロセッサのみで実行する．各プロセッサがカーネルの初期化処理へ分岐するのは，共有リソースの初期化処理が完了した後でなければならないため，スレーブプロセッサは，カーネルの初期化処理へ分岐する前に，マスタプロセッサによる共有リソースの初期化処理の完了を待ち合わせる必要がある．

カーネルの初期化処理においては，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，カーネルが用いるデバイスの初期化など）と静的 API の処理（オブジェクトの登録など）が行われる．静的 API のパラメータに関するエラーは，コンフィギュレータによって検出されるのが原則であるが，コンフィギュレータで検出できないエラーが，この処理中に検出される場合もある．

静的 API の処理順序によりシステムの規定された振舞いが変化する場合には，システムコンフィギュレーションファイルにおける静的 API の記述順と同じ順序で静的 API が処理された場合と，同じ振舞いとなる．例えば，静的 API によって同じ優先度のタスクを複数生成・起動した場合，静的 API の記述順が先のタスクが高い優先順位を持つ．それに対して，周期ハンドラの動作開始順序は，同じタイムティックで行うべき処理が複数ある場合の処理順序が規定されないことから（「4.6.1 システム時刻管理」の節を参照），静的 API の記述順となるとは限らない．

次に，静的 API（ATT_INI）により登録した初期化ルーチンが，システムコンフィギュレーションファイルにおける静的 API の記述順と同じ順序で実行される．マルチプロセッサ対応カーネルでは，すべてのプロセッサがカーネル自身の初期化処理と静的 API の処理を完了した後に，マスタプロセッサがグローバル初期化ルーチンを実行する．グローバル初期化ルーチンの実行が完了した後に，各プロセッサは，自プロセッサに割り付けられたローカル初期化ルーチンを実行する．すなわち，ローカル初期化ルーチンは，初期割付けプロセッサにより実行される．

以上が終了すると，カーネル非動作状態から動作状態に遷移し（2.5.1 カーネル動作状態と非動作状態）の節を参照），カーネルの動作が開始される．具体的には，システム状態が，全割込みロック解除状態・CPU ロック解除状態・割込み優先度マスク全解除状態・ディスパッチ許可状態に設定され（すなわち，割込みがマスク解除され），タスクの実行が開始される．

マルチプロセッサ対応カーネルでは，すべてのプロセッサがローカル初期化ルーチンの実行を完了した後に，カーネル非動作状態から動作状態に遷移し，カーネルの動作が開始される．マルチプロセッサ対応カーネルにおけるシステム初期化の流れと，各プロセッサが同期を取るタイミングを，図 2-6 に示

す.

【μITRON4.0 仕様との関係】

μITRON4.0 仕様においては, 初期化ルーチンの実行は静的 API の処理に含まれるものとしていたが, この仕様では，初期化ルーチンを登録する静的 API の処理は，初期化ルーチンを登録することのみを意味し，初期化ルーチンの実行は含まないものとした.

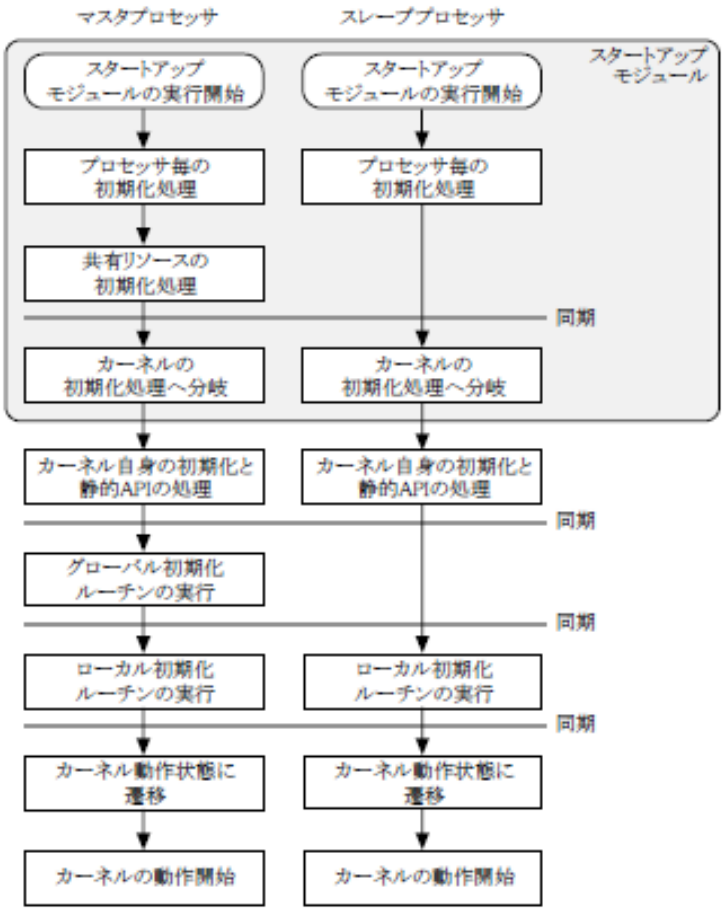

図 2-6 マルチプロセッサ対応カーネルにおけるシステム初期化の流れ

2.9.2. システム終了手順

カーネルを終了させるサービスコール（ext_ker）を呼び出すと，カーネル動作状態から非動作状態に遷移する（「2.5.1 カーネル動作状態と非動作状態」の節を参照）. 具体的には，NMI を除くすべての割込みがマスクされ，タスクの実行が停止される.

マルチプロセッサ対応カーネルでは，カーネルを終了させるサービスコール（ext_ker）は，どのプロセッサからでも呼び出すことができる. 1 つのプロセッサでカーネルを終了させるサービスコールを呼び出すと，そのプロセッサがカーネル動作状態から非動作状態に遷移した後，他のプロセッサに対してカ

ーネル終了処理の開始を要求する．複数のプロセッサから，カーネルを終了させるサービスコール（ext_ker）を呼び出してもよい．

次に，静的 API（ATT_TER）により登録した終了処理ルーチンが，システムコンフィギュレーションファイルにおける静的 API の記述順と逆の順序で実行される．

マルチプロセッサ対応カーネルでは，すべてのプロセッサがカーネル非動作状態に遷移した後に，各プロセッサが，自プロセッサに割り付けられたローカル終了処理ルーチンを実行する．すなわち，ローカル終了処理ルーチンは，初期割付けプロセッサにより実行される．すべてのプロセッサでローカル処理ルーチンの実行が完了した後に，マスタプロセッサがグローバル終了処理ルーチンを実行する．

以上が終了すると，ターゲットシステム依存の終了処理が呼び出される．ターゲットシステム依存の終了処理は，カーネルの管理外であり，アプリケーションで用意するのが基本であるが，カーネルの配布パッケージの中に，ターゲットシステム毎に標準的なルーチンが用意されている．標準のターゲットシステム依存の終了処理では，ソフトウェア環境（ライブラリなど）依存の終了処理フックを呼び出す．

マルチプロセッサ対応カーネルでは，すべてのプロセッサで，ターゲットシステム依存の終了処理が呼び出される．マルチプロセッサ対応カーネルにおけるシステム終了処理の流れと，各プロセッサが同期を取るタイミングを，図 2-7 に示す．

【使用上の注意】

マルチプロセッサ対応カーネルで，あるプロセッサからカーネルを終了させるサービスコール（ext_ker）を呼び出しても，他のプロセッサがカーネル動作状態で割込みをマスクしたまま実行し続けると，カーネルが終了しない．

プロセッサが割込みをマスクしたまま実行し続けないようにするのは，アプリケーションの責任である．例えば，ある時間を超えて割込みをマスクしたまま実行し続けていないかを，ウォッチドッグタイマを用いて監視する方法が考えられる．割込みをマスクしたまま実行し続けていた場合には，そのプロセッサからもカーネルを終了させるサービスコール（ext_ker）を呼び出すことで，カーネルを終了させることができる．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様には，システム終了に関する規定はない．

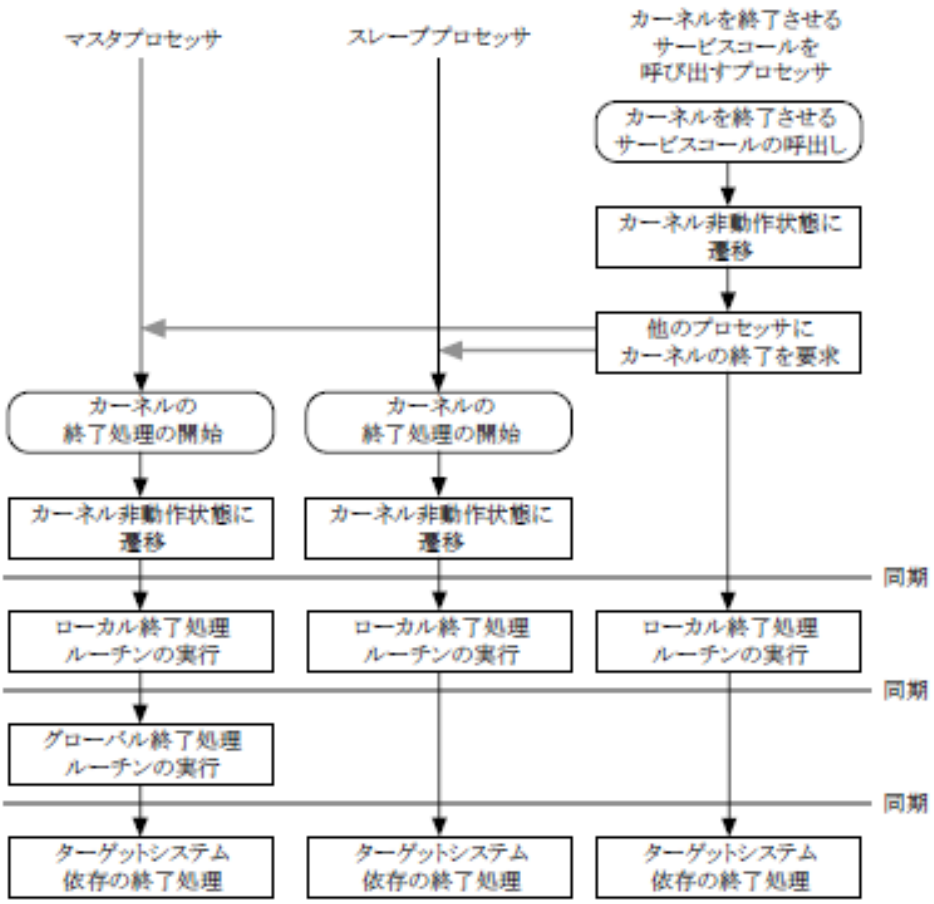

図 2-7 マルチプロセッサ対応カーネルにおけるシステム終了処理の流れ

2.10. オブジェクトの登録とその解除

2.10.1. ID 番号で識別するオブジェクト

ID 番号で識別するオブジェクトは，オブジェクトを生成する静的 API（CRE_YYY），サービスコール（acre_yyy），またはオブジェクトを追加する静的 API（ATT_YYY，ATA_YYY）によってカーネルに登録する. オブジェクトを追加する静的 API によって登録されたオブジェクトは ID 番号を持たないため，ID 番号を指定して操作することができない.

オブジェクトを生成する静的 API（CRE_YYY）は，生成するオブジェクトに ID 番号を割り付け，ID 番号を指定するパラメータとして記述した識別名を，割り付けた ID 番号にマクロ定義する．同じ識別名のオブジェクトが生成済みの場合には，E_OBJ エラーとなる.

オブジェクトを生成するサービスコール（acre_yyy）は，割付け可能な ID 番号の数を指定する静的 API（AID_YYY）によって確保された ID 番号の中から，使用されていない ID 番号を 1 つ選び，生成するオブジェクトに割り付ける．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

通知する．使用されていない ID 番号が残っていない場合には，E_NOID エラーとなる．

割付け可能な ID 番号の数を指定する静的 API（AID_YYY）は，システムコンフィギュレーションファイル中に複数記述することができる．その場合，各静的 API で指定した数の合計の数の ID 番号が確保される．

オブジェクトを生成するサービスコール（acre_yyy）によって登録したオブジェクトは，オブジェクトを削除するサービスコール（del_yyy）によって登録を解除することができる．登録解除したオブジェクトの ID 番号は，未使用の状態に戻され，その ID 番号を用いて新しいオブジェクトを登録することができる．この場合に，登録解除前のオブジェクトに対して行うつもりの操作が，新たに登録したオブジェクトに対して行われないように，注意が必要である．

オブジェクトを生成または追加する静的 API によって登録したオブジェクトは，登録を解除することができない．登録を解除しようとした場合には，E_OBJ エラーとなる．

タスク以外の処理単位は，その処理単位が実行されている間でも，登録解除することができる．この場合，登録解除された処理単位に実行が強制的に終了させられることはなく，処理単位が自ら実行を終了するまで，処理単位の実行は継続される．

同期・通信オブジェクトを削除した時に，そのオブジェクトを待っているタスクがあった場合，それらのタスクは待ち解除され，待ち状態に遷移させたサービスコールは E_DLT エラーとなる．複数のタスクが待ち解除される場合には，待ち行列につながれていた順序で待ち解除される．削除した同期・通信オブジェクトが複数の待ち行列を持つ場合には，別の待ち行列で待っていたタスクの間の待ち解除の順序は，該当するサービスコール毎に規定する．

オブジェクトを再初期化するサービスコール（ini_yyy）は，指定したオブジェクトを削除した後に，同じパラメータで再度生成したのと等価の振舞いをする．ただし，オブジェクトを生成または追加する静的 API によって登録したオブジェクトも，再初期化することができる．

なお，動的生成対応カーネル以外では，オブジェクトを生成するサービスコール（acre_yyy），割付け可能な ID 番号の数を指定する静的 API（AID_YYY），オブジェクトを削除するサービスコール（del_yyy）は，サポートされない．

【μITRON4.0 仕様との関係】

ID 番号を指定してオブジェクトを生成するサービスコール（cre_yyy）を廃止した．また，オブジェクトを生成または追加する静的 API によって登録したオブジェクトは，登録解除できないこととした．

μITRON4.0 仕様では，割付け可能な ID 番号の数を指定する静的 API（AID_YYY）は規定されてい

ない.

複数の待ち行列を持つ同期・通信オブジェクトを削除した時に，別の待ち行列で待っていたタスクの間の待ち解除の順序は，μITRON4.0 仕様では実装依存とされている.

【μITRON4.0/PX 仕様との関係】
アクセス許可ベクタを指定してオブジェクトを生成する静的 API（CRA_YYY）は廃止し，オブジェクトの登録後にアクセス許可ベクタを設定する静的 API（SAC_YYY）をサポートすることとした．これにあわせて, アクセス許可ベクタを指定してオブジェクトを登録するサービスコール（cra_yyy, acra_yyy, ata_yyy）も廃止した.

【仕様決定の理由】
ID 番号を指定してオブジェクトを生成するサービスコール（cre_yyy）とアクセス許可ベクタを指定してオブジェクトを登録するサービスコール（cra_yyy，acra_yyy，ata_yyy）を廃止したのは，必要性が低いと考えたためである．静的 API についても，サービスコールに整合するよう変更した.

2.10.2. オブジェクト番号で識別するオブジェクト

オブジェクト番号で識別するオブジェクトは，オブジェクトを定義する静的 API（DEF_YYY）またはサービスコール（def_yyy）によってカーネルに登録する.

オブジェクトを定義するサービスコール（def_yyy）によって登録したオブジェクトは，同じサービスコールを，オブジェクトの定義情報を入れたパケットへのポインタを NULL として呼び出すことによって，登録を解除することができる．登録解除したオブジェクト番号は，オブジェクト登録前の状態に戻され，同じオブジェクト番号に対して新たにオブジェクトを定義することができる．登録解除されていないオブジェクト番号に対して再度オブジェクトを登録しようとした場合には，E_OBJ エラーとなる.
オブジェクトを定義する静的 API によって登録したオブジェクトは, 登録を解除することができない. 登録を解除しようとした場合には，E_OBJ エラーとなる.

なお，動的生成対応カーネル以外では，オブジェクトを定義するサービスコール（def_yyy）はサポートされない.

【μITRON4.0 仕様との関係】
この仕様では，オブジェクトの定義を変更したい場合には，一度登録解除した後に，新たにオブジェクトを定義する必要がある. また, オブジェクトを定義する静的 API によって登録したオブジェクトは, この仕様では，登録解除できないこととした.

2.10.3. 識別番号を持たないオブジェクト

識別する必要がないために，識別番号を持たないオブジェクトは，オブジェクトを追加する静的 API（ATT_YYY）によってカーネルに登録する．

2.10.4. オブジェクト生成に必要なメモリ領域

カーネルオブジェクトを生成する際に，サイズが一定でないメモリ領域を必要とする場合には，カーネルオブジェクトを生成する静的 API およびサービスコールに，使用するメモリ領域の先頭番地を渡すパラメータを設けている．このパラメータを NULL とした場合，必要なメモリ領域は，コンフィギュレータまたはカーネルにより確保される．

オブジェクト生成に必要なメモリ領域の中で，カーネルの内部で用いるものを，カーネルの用いるオブジェクト管理領域と呼ぶ．この仕様では，以下のメモリ領域が，カーネルの用いるオブジェクト管理領域に該当する．

- データキュー管理領域
- 優先度データキュー管理領域
- 優先度別のメッセージキューヘッダ領域
- 固定長メモリプール管理領域

【補足説明】
カーネルオブジェクトを生成する際には，管理ブロックなどを置くためのメモリ領域も必要になるが，サイズが一定のメモリ領域はコンフィギュレータにより確保されるため，カーネルオブジェクトを生成する静的 API およびサービスコールにそれらのメモリ領域の先頭番地を渡すパラメータを設けていない．

2.10.5. オブジェクトが属する保護ドメインの設定

保護機能対応カーネルにおいて，カーネルオブジェクトが属する保護ドメインは，オブジェクトの登録時に決定し，登録後に変更することはできない．

カーネルオブジェクトを静的 API によって登録する場合には，オブジェクトを登録する静的 API を，そのオブジェクトを属させる保護ドメインの囲みの中に記述する．無所属のオブジェクトを登録する静的 API は，保護ドメインの囲みの外に記述する（「2.12.3 保護ドメインの指定」の節を参照）．

カーネルオブジェクトをサービスコールによって登録する場合には，オブジェクト属性に TA_DOM(domid)を指定することにより，オブジェクトを属させる保護ドメインを設定する．ここで domid は，そのオブジェクトを属させる保護ドメインの ID 番号であり，TDOM_KERNEL（＝-1）を指

定することでカーネルドメインに属させることができる．また，domid に TDOM_SELF（＝0）を指定するか，オブジェクト属性に TA_DOM(domid)を指定しないことで，自タスクが属する保護ドメインに属させることができる．さらに，無所属のオブジェクトを登録する場合には，domid に TDOM_NONE（＝-2）を指定する．

ただし，特定の保護ドメインのみに属することができるカーネルオブジェクトを登録するサービスコールの中には，オブジェクトを属させる保護ドメインをオブジェクト属性で設定する必要がないものもある．

割付け可能な ID 番号の数を指定する静的 API（AID_YYY）で確保した ID 番号は，どの保護ドメインに属するオブジェクトにも（また，無所属のオブジェクトにも）割り付けられる．これらの静的 API は，保護ドメインの囲みの外に記述しなければならない．保護ドメインの囲みの中に記述した場合には，E_RSATR エラーとなる．

【補足説明】
この仕様では，カーネルオブジェクトの属する保護ドメインを参照する機能は用意していない．

【仕様決定の理由】
カーネルオブジェクトをサービスコールによって登録する場合に，オブジェクトを属させる保護ドメインをオブジェクト属性で指定することにしたのは，保護機能対応でないカーネルとの互換性のためには，サービスコールのパラメータを増やさない方が望ましいためである．

2.10.6. オブジェクトが属するクラスの設定

マルチプロセッサ対応カーネルにおいて，カーネルオブジェクトが属するクラスは，オブジェクトの登録時に決定し，登録後に変更することはできない．

カーネルオブジェクトを静的 API によって登録する場合には，オブジェクトを登録する静的 API を，そのオブジェクトを属させるクラスの囲みの中に記述する．クラスに属さないオブジェクトを登録する静的 API は，クラスの囲みの外に記述する（2.12.4 クラスの指定）の節を参照）．

カーネルオブジェクトをサービスコールによって登録する場合には，オブジェクト属性に TA_CLS(clsid)を指定することにより，オブジェクトを属させるクラスを設定する．ここで clsid は，そのオブジェクトを属させるクラスの ID 番号であり，clsid に TCLS_SELF（＝0）を指定するか，オブジェクト属性に TA_CLS(clsid)を指定しないことで，自タスクが属するクラスに属させることができる．

割付け可能な ID 番号の数を指定する静的 API（AID_YYY）で確保した ID 番号は，静的 API を囲むクラスに属するオブジェクトにのみ割り付けられる．これらの静的 API は，確保した ID 番号を割り付

けるオブジェクトの属すべきクラスの囲みの中に記述しなければならない．クラスの囲みの外に記述した場合には，E_RSATR エラーとなる．

【補足説明】
この仕様では，カーネルオブジェクトの属するクラスを参照する機能は用意していない．

【仕様決定の理由】
カーネルオブジェクトをサービスコールによって登録する場合に，オブジェクトを属させるクラスをオブジェクト属性で指定することにしたのは，マルチプロセッサ対応でないカーネルとの互換性のためには，サービスコールのパラメータを増やさない方が望ましいためである．

2.10.7. オブジェクトの状態参照

ID 番号で識別するオブジェクトのすべてと，オブジェクト番号で識別するオブジェクトの一部に対して，オブジェクトの状態を参照するサービスコール（ref_yyy，get_yyy）を用意する．

オブジェクトの状態を参照するサービスコールでは，オブジェクトの登録時に指定し，その後に変化しない情報（例えば，タスクのタスク属性や初期優先度）を参照するための機能は用意しないことを原則とする．自タスクの拡張情報の参照するサービスコール（get_inf）は，この原則に対する例外である．

2.11. オブジェクトのアクセス保護

この節では，カーネルオブジェクトのアクセス保護について述べる．この節の内容は，保護機能対応カーネルにのみ適用される．

2.11.1. オブジェクトのアクセス保護とアクセス違反の通知

カーネルオブジェクトに対するアクセスは，そのオブジェクトに対して設定されたアクセス許可ベクタによって保護される．ただし，アクセス許可ベクタを持たないオブジェクトに対するアクセスは，システム状態に対するアクセス許可ベクタによって保護される．また，オブジェクトを登録するサービスコールと，特定のオブジェクトに関連しないシステムの状態に対するアクセスについては，システム状態のアクセス許可ベクタによって保護される．

アクセス許可ベクタによって許可されていないアクセス（アクセス違反）は，カーネルによって検出され，以下の方法によって通知される．

サービスコールにより，メモリオブジェクト以外のカーネルオブジェクトに対して，許可されていないアクセスを行おうとした場合，サービスコールから E_OACV エラーが返る．また，メモリオブジェク

トに対して，許可されていない管理操作または参照操作を行おうとした場合も，サービスコールから E_OACV エラーが返る．

メモリオブジェクトに対して，通常のメモリアクセスにより，許可されていない書込みアクセスまたは読出しアクセス（実行アクセスを含む）を行おうとした場合，CPU 例外ハンドラが起動される．どの CPU 例外ハンドラが起動されるかは，ターゲット定義である．ターゲットによっては，エミュレートされた CPU 例外ハンドラの場合もある．また，ターゲット定義で，アクセス違反の状況に応じて異なる CPU 例外ハンドラが起動される場合もある．この（これらの）CPU 例外ハンドラを，メモリアクセス違反ハンドラと呼ぶ．

メモリオブジェクトに対して，サービスコールを通じて，許可されていない書込みアクセスまたは読出しアクセスを行おうとした場合，サービスコールから E_MACV エラーが返るか，メモリアクセス違反ハンドラが起動される．E_MACV エラーが返るかメモリアクセス違反ハンドラされるかは，ターゲット定義である．

メモリアクセス違反ハンドラでは，アクセス違反を発生させたアクセスに関する情報（アクセスした番地，アクセスの種別，アクセスした命令の番地など）を参照する方法を，ターゲット定義で用意する．

メモリオブジェクトとしてカーネルに登録されていないメモリ領域に対して，カーネルドメイン以外の保護ドメインから，書込みアクセスまたは読出しアクセス（実行アクセスを含む）を行おうとした場合には，メモリオブジェクトに対するアクセスが許可されていない場合と同様に扱われる．

【未決定事項】

マルチプロセッサ対応カーネルにおいて，システム状態のアクセス許可ベクタをシステム全体で 1 つ持つかプロセッサ毎に持つかは，今後の課題である．

【μITRON4.0/PX 仕様との関係】

μITRON4.0/PX 仕様では，アクセス保護の実装定義の制限について規定しているが，この仕様では，メモリオブジェクトに対するアクセス許可ベクタのターゲット定義の制限以外については規定していない．

【仕様決定の理由】

オブジェクトを登録するサービスコールを，そのオブジェクトのアクセス許可ベクタによって保護しないのは，オブジェクトを登録する前には，アクセス許可ベクタが設定されていないためである．

2.11.2. メモリオブジェクトに対するアクセス許可ベクタの制限

メモリオブジェクトの書込みアクセスと読出しアクセス（実行アクセスを含む）に対して設定できる

アクセス許可パターンは，ターゲット定義で制限される場合がある．

ただし，少なくとも，次の 5 つの組み合わせの設定は，行うことができる．

(a) メモリオブジェクトが属する保護ドメインのみに，読出しアクセス（実行アクセスを含む）のみを許可する．これを，専有リードオンリー（private　read only）と呼ぶ．

(b) メモリオブジェクトが属する保護ドメインのみに，書込みアクセスと読出しアクセス（実行アクセスを含む）を許可する．これを，専有リードライト（private read/write）と呼ぶ．

(c) すべての保護ドメインに，読出しアクセス（実行アクセスを含む）のみを許可する．これを，共有リードオンリー（shared read only）と呼ぶ．

(d) すべての保護ドメインに，書込みアクセスと読出しアクセス（実行アクセスを含む）を許可する．これを，共有リードライト（shared read/write）と呼ぶ．

(e) メモリオブジェクトが属する保護ドメインに，書込みアクセスと読出しアクセス（実行アクセスを含む）を許可し，他の保護ドメインには，読出しアクセス（実行アクセスを含む）のみを許可する．これを，共有リード専有ライト（shared read private write）と呼ぶ．

また，ターゲット定義で，1 つの保護ドメインに登録できるメモリオブジェクトの数が制限される場合がある．

2.11.3. デフォルトのアクセス許可ベクタ

静的 API によりカーネルオブジェクトを登録した直後は，次に規定されるデフォルトのアクセス許可ベクタが設定される．

保護ドメインに属するカーネルオブジェクトに対しては，4 つの種別のアクセスがいずれも，その保護ドメインのみに許可される．すなわち，カーネルドメインに属するオブジェクトに対しては，4 つのアクセス許可パターンがいずれも TACP_KERNEL に，ユーザドメインに属するオブジェクトに対しては，4 つのアクセス許可パターンがいずれも TACP(domid)（domid はオブジェクトが属する保護ドメインの ID 番号）に設定される．

無所属のカーネルオブジェクトに対しては，4 つの種別のアクセスがいずれも，すべての保護ドメインに許可される．すなわち，4 つのアクセス許可パターンがいずれも，TACP_SHARED に設定される．
システム状態のアクセス許可ベクタは，4 つの種別のアクセスがいずれも，カーネルドメインのみに許可される．すなわち，4 つのアクセス許可パターンがいずれも，TACP_KERNEL に設定される．

【未決定事項】
サービスコールによりカーネルオブジェクトを登録した直後のアクセス許可ベクタについては，今後

の課題である．

2.11.4. アクセス許可ベクタの設定

アクセス許可ベクタをデフォルト以外の値に設定するために，カーネルオブジェクトのアクセス許可ベクタを設定する静的 API（SAC_YYY）とサービスコール（sac_yyy）が用意されている．また，システム状態のアクセス許可ベクタを設定する静的 API（SAC_SYS）とサービスコール（sac_sys）が用意されている．

ただし，静的 API によって登録したオブジェクトは，サービスコール（sac_yyy）によってアクセス許可ベクタを設定することができない．アクセス許可ベクタを設定しようとした場合には，E_OBJ エラーとなる．

メモリオブジェクトに対しては，アクセス許可ベクタを設定する静的 API は用意されておらず，オブジェクトの登録と同時にアクセス許可ベクタを設定する静的 API（ATA_YYY）が用意されている．

オブジェクトに対するアクセスが許可されているかは，そのオブジェクトにアクセスするサービスコールを呼び出した時点でチェックされる．そのため，アクセス許可ベクタを変更しても，変更以前に呼び出されたサービスコールの振舞いには影響しない．例えば，待ち行列を持つ同期・通信オブジェクトのアクセス許可ベクタを変更しても，呼び出した時点ですでに待ち行列につながれているタスクには影響しない．また，ミューテックスのアクセス許可ベクタを変更しても，呼び出した時点ですでにミューテックスをロックしていたタスクには影響しない．

なお，動的生成対応カーネル以外では，アクセス許可ベクタを設定するサービスコール（sac_yyy）はサポートされない．

この仕様では，カーネルオブジェクトに設定されたアクセス許可ベクタを参照する機能は用意していない．

【μITRON4.0/PX 仕様との関係】
アクセス許可ベクタを指定してオブジェクトを生成する静的 API（CRA_YYY）は廃止し，オブジェクトの登録後にアクセス許可ベクタを設定する静的 API（SAC_YYY）をサポートすることとした．
静的 API によって登録したオブジェクトは，サービスコール（sac_yyy）によってアクセス許可ベクタを設定することができないこととした．

オブジェクトの状態参照するサービスコール（ref_yyy）により，オブジェクトに設定されたアクセス許可ベクタを参照する機能サポートしないこととした．これは，オブジェクトの登録時に指定し，その後に変化しない情報を参照するための機能は用意しないという原則に合わせるための修正である．

2.11.5. カーネルの管理領域のアクセス保護

カーネルが動作するために，カーネルの内部で用いるメモリ領域を，カーネルの管理領域と呼ぶ．ユーザタスクからカーネルを保護するためには，カーネルの管理領域にアクセスできるのは，カーネルドメインのみでなければならない．そのため，カーネルの管理領域は，4 つの種別のアクセスがカーネルドメインのみに許可されたメモリオブジェクト（これを，カーネル専用のメモリオブジェクトと呼ぶ）の中に置かれる．

カーネルの用いるオブジェクト管理領域（カーネルの管理領域に該当する．「2.10.4 オブジェクト生成に必要なメモリ領域」の節を参照）として，カーネル専用のメモリオブジェクトに含まれないメモリ領域を指定した場合，E_OBJ エラーとなる．また，カーネルの用いるオブジェクト管理領域の先頭番地に NULL を指定した場合，必要なメモリ領域が，カーネル専用のメモリオブジェクトの中に確保される．

システムタスクのスタック領域，ユーザタスクのシステムスタック領域，非タスクコンテキスト用のスタック領域は，カーネルの用いるオブジェクト管理領域には該当しないが，カーネルドメインの実行中にのみアクセスされるため，カーネルの用いるオブジェクト管理領域と同様の扱いとなる．一方，ユーザタスクのユーザスタック領域と固定長メモリプール領域は，ユーザドメインの実行中にもアクセスされるため，カーネルの用いるオブジェクト管理領域とは異なる扱いとなる．

2.11.6. ユーザタスクのユーザスタック領域

ユーザタスクが非特権モードで実行する間に用いるスタック領域を，システムスタック領域（「4.1 タスク管理機能」の節を参照）と対比させて，ユーザスタック領域と呼ぶ．ユーザスタック領域は，そのタスクと同じ保護ドメインに属する 1 つのメモリオブジェクトとしてカーネルに登録されるが，他のメモリオブジェクトとは異なり，次のように扱われる．

タスクのユーザスタック領域に対しては，そのタスクのみが書込みアクセスおよび読出しアクセスを行うことができる．そのため，書込みアクセスと読出しアクセス（実行アクセスを含む）に対するアクセス許可パターンは意味を持たない．ユーザスタック領域に対して実行アクセスを行えるかどうかは，ターゲット定義である．

ただし，上記の仕様を実現するために大きいオーバヘッドを生じる場合には，ターゲット定義で，タスクのユーザスタック領域を，そのタスクが属する保護ドメインのみからアクセスできるものとする場合がある．

【μITRON4.0/PX 仕様との関係】
この仕様では，タスクのユーザスタック領域は，そのタスクのみがアクセスできるものとした．

2.12. システムコンフィギュレーション手順

2.12.1. システムコンフィギュレーションファイル

カーネルやシステムサービスが管理するオブジェクトの生成情報や初期状態などを記述するファイルを，システムコンフィギュレーションファイル（systemconfiguration file）と呼ぶ．また，システムコンフィギュレーションファイルを解釈して，カーネルやシステムサービスの構成・初期化情報を含むファイルなどを生成するツールを，コンフィギュレータ（configurator）と呼ぶ．

システムコンフィギュレーションファイルには，カーネルの静的 API，システムサービスの静的 API，保護ドメインの囲み，クラスの囲み，コンフィギュレータに対する INCLUDE ディレクティブ，C 言語プリプロセッサのインクルードディレクティブ（#include）と条件ディレクティブ（#if，#ifdef など）のみを記述することができる．

コンフィギュレータに対する INCLUDE ディレクティブは，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して記述するために用いるもので，その文法は次のいずれかである（両者の違いは，指定されたファイルを探すディレクトリの違いのみ）．

```
INCLUDE("ファイル名");
INCLUDE(<ファイル名>);
```

コンフィギュレータは，INCLUDE ディレクティブによって指定されたファイルの記述を，システムコンフィギュレーションファイルの一部分として解釈する．

すなわち，INCLUDE ディレクティブによって指定されたファイル中には，カーネルの静的 API，システムサービスの静的 API，コンフィギュレータに対する INCLUDE ディレクティブ，C 言語プリプロセッサのインクルードディレクティブと条件ディレクティブのみを記述することができる．

C 言語プリプロセッサのインクルードディレクティブは，静的 API のパラメータを解釈するために必要な C 言語のヘッダファイルを指定するために用いる．また，条件ディレクティブは，有効とする静的 API を選択するために用いることができる．ただし，インクルードディレクティブは，コンフィギュレータが生成するファイルでは先頭に集められる．そのため，条件ディレクティブの中にインクルードディレクティブを記述しても，インクルードディレクティブは常に有効となる．また，1 つの静的 API の記述の途中に，条件ディレクティブを記述することはできない．

コンフィギュレータは，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，その記述順に解釈する．そのため例えば，タスクを生成する静的 API の前に，そのタスクにタスク例外処理ルーチンを

定義する静的 API が記述されていた場合，タスク例外処理ルーチンを定義する静的 API が E_NOEXS エラーとなる．

【μITRON4.0 仕様との関係】
システムコンフィギュレーションファイルにおける C 言語プリプロセッサのディレクティブの扱いを全面的に見直し，コンフィギュレータに対する INCLUDE ディレクティブを設けた．また，共通静的 API を廃止した．μITRON4.0 仕様における#include ディレクティブの役割は，この仕様では INCLUDE ディレクティブに置き換わる．逆に，μITRON4.0 仕様における INCLUDE 静的 API の役割は，この仕様では#include ディレクティブに置き換わる．

2.12.2. 静的 API の文法とパラメータ

静的 API は，次に述べる例外を除いては，C 言語の関数呼出しと同様の文法で記述する．すなわち，静的 API の名称に続けて，静的 API の各パラメータを”,”で区切って列挙したものを”(“と”)”で囲んで記述し，最後に”;”を記述する．ただし，静的 API のパラメータに構造体（または構造体へのポインタ）を記述する場合には，構造体の各フィールドを”,”で区切って列挙したものを”{”と”}”で囲んだ形で記述する．
サービスコールに対応する静的 API の場合，静的 API のパラメータは，対応するサービスコールのパラメータと同一とすることを原則とする．

静的 API のパラメータは，次の 4 種類に分類される．

(a) オブジェクト識別名
オブジェクトの ID 番号を指定するパラメータ．オブジェクトの名称を表す単一の識別名のみを記述することができる．

コンフィギュレータは，オブジェクト生成のための静的 API（CRE_YYY）を処理する際に，オブジェクトに ID 番号を割り付け，構成・初期化ヘッダファイルに，指定された識別名を割り付けた ID 番号にマクロ定義する C 言語プリプロセッサのディレクティブ（#define）を生成する．

オブジェクト生成以外の静的 API が，オブジェクトの ID 番号をパラメータに取る場合（カーネルの静的 API では，SAC_TSK や DEF_TEX の tskid パラメータ等がこれに該当する）には，パラメータとして記述する識別名は，生成済みのオブジェクトの名称を表す識別名でなければならない．そうでない場合には，コンフィギュレータがエラーを報告する．

静的 API の整数定数式パラメータの記述に，オブジェクト識別名を使用することはできない．

(b) 整数定数式パラメータ
オブジェクト番号や機能コード，オブジェクト属性，サイズや数，優先度など，整数値を指定するパ

ラメータ．プログラムが配置される番地に依存せずに値の決まる整数定数式を記述することができる．

整数定数式の解釈に必要な定義や宣言等は，システムコンフィギュレーションファイルから C 言語プリプロセッサのインクルードディレクティブによってインクルードするファイルに含まれていなければならない．

(c) 一般定数式パラメータ

処理単位のエントリ番地，メモリ領域の先頭番地，拡張情報など，番地を指定する可能性のあるパラメータ．任意の定数式を記述することができる．

定数式の解釈に必要な定義や宣言等は，システムコンフィギュレーションファイルから C 言語プリプロセッサのインクルードディレクティブによってインクルードするファイルに含まれていなければならない．

(d) 文字列パラメータ

オブジェクトモジュール名やセクション名など，文字列を指定するパラメータ．任意の文字列を，C 言語の文字列の記法で記述することができる．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様においては，静的 API のパラメータを次の 4 種類に分類していたが，コンフィギュレータの仕組みを見直したことに伴い全面的に見直した．

(A) 自動割付け対応整数値パラメータ
(B) 自動割付け非対応整数値パラメータ
(C) プリプロセッサ定数式パラメータ
(D) 一般定数式パラメータ

この仕様の(a)が，おおよそμITRON4.0 仕様の(A)に相当するが，(a)には整数値を記述できない点が異なる．(b)～(c)と(B)～(D)の間には単純な対応関係がないが，記述できる定数式の範囲には，(B)⊂(C)⊂(b)⊂(c)＝(D)の関係がある．

μITRON4.0 仕様では，静的 API のパラメータは基本的には(D)とし，コンフィギュレータが値を知る必要があるパラメータを(B)，構成・初期化ファイルに生成する C 言語プリプロセッサの条件ディレクティブ（#if）中に含めたい可能性のあるパラメータを(C)としていた．

それに対して，この仕様におけるコンフィギュレータの処理モデル（「2.12.5 コンフィギュレータの処理モデル」の節を参照）では，コンフィギュレータのパス 2 において定数式パラメータの値を知ることができるため，(B)～(D)の区別をする必要がない．そのため，静的 API のパラメータは基本的には(b)とし，パス 2 で値を知ることのできない定数式パラメータのみを(c)としている．

2.12.3. 保護ドメインの指定

保護機能対応カーネルでは，オブジェクトを登録する静的 API 等を，そのオブジェクトが属する保護ドメインの囲みの中に記述する．無所属のオブジェクトを登録する静的 API は，保護ドメインの囲みの外に記述する．保護ドメインに属すべきオブジェクトを登録する静的 API 等を，保護ドメインの囲みの外に記述した場合には，コンフィギュレータが E_RSATR エラーを報告する．

ユーザドメインの囲みの文法は次の通り．

```
DOMAIN(保護ドメイン名){
    ユーザドメインに属するオブジェクトを登録する静的 API 等
```

保護ドメイン名には，ユーザドメインの名称を表す単一の識別名のみを記述することができる．

コンフィギュレータは，ユーザドメインの囲みを処理する際に，ユーザドメインに保護ドメイン ID を割り付け，構成・初期化ヘッダファイルに，指定された保護ドメイン名を割り付けた保護ドメイン ID にマクロ定義する C 言語プリプロセッサのディレクティブ（#define）を生成する．また，ユーザドメインの囲みの中およびそれ以降に記述する静的 API の整数定数式パラメータの記述に保護ドメイン名を記述すると，割り付けた保護ドメイン ID の値に評価される．

ユーザドメインの囲みの中を空にすることで，ユーザドメインへの保護ドメイン ID の割付けのみを行うことができる．

カーネルドメインの囲みの文法は次の通り．

```
KERNEL_DOMAIN{
    カーネルドメインに属するオブジェクトを登録する静的 API 等
}
```

同じ保護ドメイン名を指定したユーザドメインの囲みや，カーネルドメインの囲みを，複数回記述してもよい．保護機能対応でないカーネルで保護ドメインの囲みを記述した場合や，保護ドメインの囲みの中に保護ドメインの囲みを記述した場合には，コンフィギュレータがエラーを報告する．

【μITRON4.0/PX 仕様との関係】
ユーザドメインの囲みの文法を変更した．

【仕様決定の理由】
保護ドメインに属すべきオブジェクトを登録する静的 API 等を保護ドメインの囲みの外に記述した場合のエラーコードを E_RSATR としたのは，オブジェクトを動的に登録する API においては，オブジェクトの属する保護ドメインを，オブジェクト属性によって指定するためである．

2.12.4. クラスの指定

マルチプロセッサ対応カーネルでは，オブジェクトを登録する静的 API 等を，そのオブジェクトが属するクラスの囲みの中に記述する．クラスに属すべきオブジェクトを登録する静的 API 等を，クラスの囲みの外に記述した場合には，コンフィギュレータが E_RSATR エラーを報告する．

クラスの囲みの文法は次の通り．

```
CLASS(クラス ID) {
    クラスに属するオブジェクトを登録する静的 API 等
}
```

クラス ID には，静的 API の整数定数式パラメータと同等の定数式を記述することができる．使用できないクラス ID を指定した場合には，コンフィギュレータが E_ID エラーを報告する．

同じクラス ID を指定したクラスの囲みを複数回記述してもよい．マルチプロセッサ対応でないカーネルでクラスの囲みを記述した場合や，クラスの囲みの中にクラスの囲みを記述した場合には，コンフィギュレータがエラーを報告する．

なお，保護機能とマルチプロセッサの両方に対応するカーネルでは，保護ドメインの囲みとクラスの囲みはどちらが外側になっていてもよい．

【仕様決定の理由】
クラスに属すべきオブジェクトを登録する静的 API 等をクラスの囲みの外に記述した場合のエラーコードを E_RSATR としたのは，オブジェクトを動的に登録する API においては，オブジェクトの属するクラスを，オブジェクト属性によって指定するためである．

2.12.5. コンフィギュレータの処理モデル

コンフィギュレータは，次の 3 つないしは 4 つのパスにより，システムコンフィギュレーションファイルを解釈し，構成・初期化情報を含むファイルなどを生成する（図 2-8）．

最初のパス 1 では，システムコンフィギュレーションファイルを解釈し，そこに含まれる静的 API の整数定数式パラメータの値を C コンパイラを用いて求めるために，パラメータ計算用 C 言語ファイル（cfg1_out.c）を生成する．この時，システムコンフィギュレーションファイルに含まれる C 言語プリプロセッサのインクルードディレクティブは，パラメータ計算用 C 言語ファイルの先頭に集めて生成する．また，条件ディレクティブは，順序も含めて，そのままの形でパラメータ計算用 C 言語ファイルに出力する．システムコンフィギュレーションファイルに文法エラーや未サポートの記述があった場合には，この段階で検出される．

次に，C コンパイラおよび関連ツールを用いて，パラメータ計算用 C 言語ファイルをコンパイルし，ロードモジュールを生成する．また，それを S レコードフォーマットの形（cfg1_out.srec）に変換し，ロードモジュール中の各シンボルとアドレスの対応表を含むシンボルファイル（cfg1_out.syms）を生成する．静的 API のパラメータに解釈できない式が記述された場合には，この段階でエラーが検出される．

コンフィギュレータのパス 2 では，パス 1 で生成されたオブジェクトファイルを S レコードフォーマットの形に変換したものとシンボルファイルから，C 言語プリプロセッサの条件ディレクティブによりどの静的 API が有効となったかと，それらの静的 API の整数定数式パラメータの値を取り出し，カーネルおよびシステムサービスの構成・初期化ファイル（kernel_cfg.c など）と構成・初期化ヘッダファイル（kernel_cfg.h など）を生成する．構成・初期化ヘッダファイルには，登録できるオブジェクトの数（動的生成対応カーネル以外では，静的 API によって登録されたオブジェクトの数に一致）やオブジェクトの ID 番号などの定義を出力する．静的 API の整数定数式パラメータに不正がある場合には，この段階でエラーが検出される．

パス 2 で生成されたこれらのファイルを，他のソースファイルとあわせてコンパイルし，アプリケーションのロードモジュールを生成する．また，それを S レコードフォーマットの形（system.srec）に変換し，ロードモジュール中の各シンボルと番地の対応表を含むシンボルファイル（system.syms）を生成する．

コンフィギュレータのパス 3 では，パス 1 で生成されたロードモジュールを S レコードフォーマットの形に変換したものとシンボルファイル，パス 2 で生成されたロードモジュールを S レコードフォーマットの形に変換したものとシンボルファイルから，静的 API パラメータの値などを取り出し，妥当性のチェックを行う．静的 API の一般定数式パラメータに不正がある場合には，この段階でエラーが検出される．

保護機能対応カーネルにおいては，メモリ保護のための設定情報を生成するために，パス 3 ではじめて得られる情報が必要となる．そこで，そのようなメモリ保護のための設定情報は，パス 3 においてメモリ構成・初期化ファイル（kernel_mem.c）に生成する．生成したメモリ構成・初期化ファイルは，他のソースファイルとあわせてコンパイルし，アプリケーションの最終的なロードモジュールを生成する．

そのため，パス 2 で生成されたファイルから生成したロードモジュールは，仮のロードモジュールという位置付けになる．ここで，仮のロードモジュールと最終的なロードモジュールでサイズが変化してはならないため，パス 3 でメモリ構成・初期化ファイルに生成するのと同じサイズのデータ構造を，パス 2 において仮のメモリ構成・初期化ファイル（kernel_mem2.c）に生成し，これも含めて仮のロードモジュールを生成しておく．また，仮のロードモジュールを S レコードフォーマットの形に変換したもの（cfg2_out.srec），仮のロードモジュール中の各シンボルと番地の対応表を含むシンボルファイル（cfg2_out.syms）も，混乱を避けるためにファイル名を変更しておく（図 2-9）．

パス 3 でメモリ保護のための設定情報を生成した場合には，パス 4 を実行する．コンフィギュレータのパス 4 では，パス 1 で生成されたロードモジュールを S レコードフォーマットの形に変換したものとシンボルファイル，パス 3 で生成されたロードモジュールを S レコードフォーマットの形に変換したものとシンボルファイルから，生成したロードモジュールの妥当性のチェックを行う．この段階で検出されるエラーは，コンフィギュレーション処理の不具合を示すものである．

【μITRON4.0 仕様との関係】

コンフィギュレータの処理モデルは全面的に変更した．

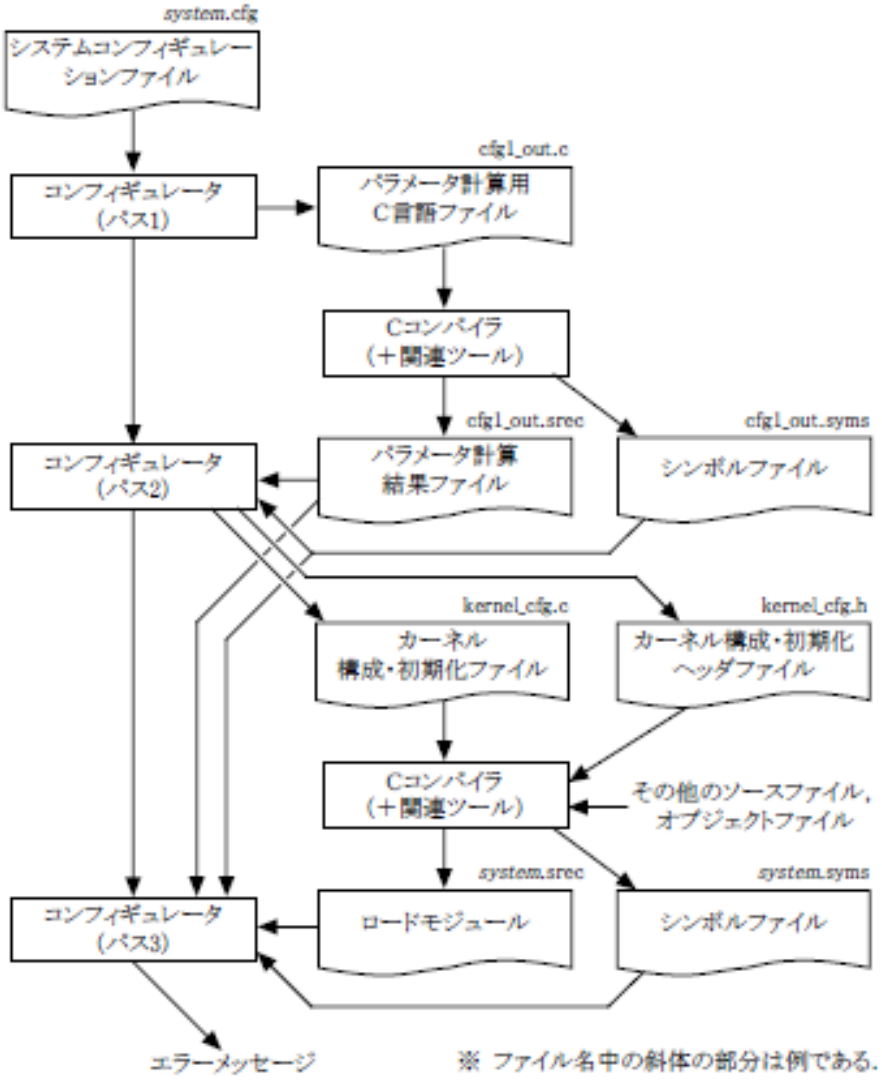

図 2-8 コンフィグレータの処理モデル

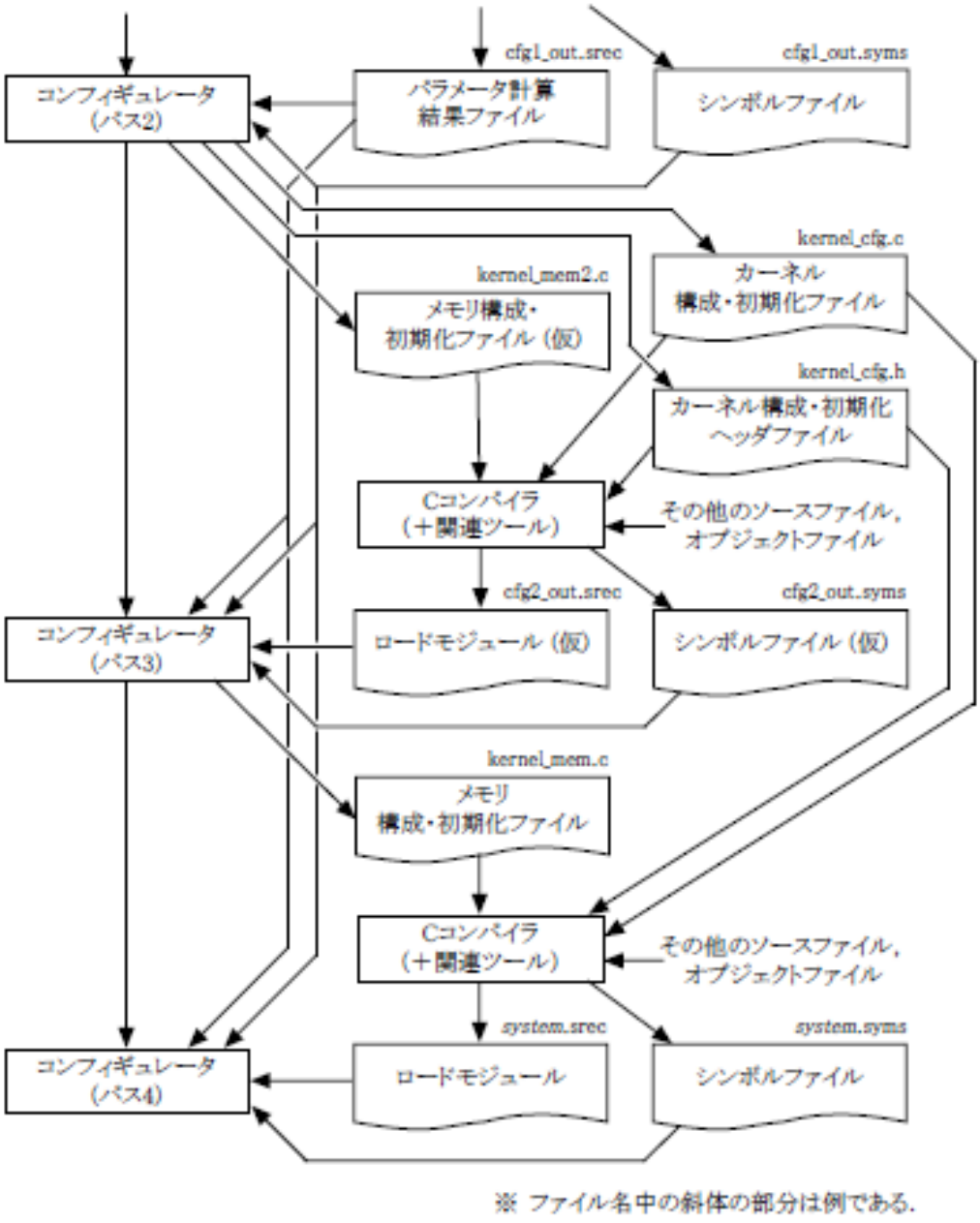

図 2-9 最終的なロードモジュールのパス 3 での生成

2.12.6. 静的 API のパラメータに関するエラー検出

静的 API のパラメータに関するエラー検出は，同じものがサービスコールとして呼ばれた場合と同等とすることを原則とする．言い換えると，サービスコールによっても検出できないエラーは，静的 API においても検出しない．静的 API の機能説明中の「E_XXXXX エラーとなる」または「E_XXXXX エラーが返る」という記述は，コンフィギュレータがそのエラーを検出することを意味する．

ただし，エラーの種類によっては，サービスコールと同等のエラー検出を行うことが難しいため，そのようなものについては例外とする．例えば，メモリ不足をコンフィギュレータによって検出するのは

容易ではない．

逆に，オブジェクト属性については，サービスコールより強力なエラーチェックを行える可能性がある．例えば，タスク属性に TA_STA と記述されている場合，サービスコールではエラーを検出できないが，コンフィギュレータでは検出できる可能性がある．ただし，このようなエラー検出を完全に行おうとするとコンフィギュレータが複雑になるため，このようなエラーを検出することは必須とせず，検出できた場合には警告として報告する．

【μITRON4.0 仕様との関係】
μITRON4.0 仕様では，静的 API のパラメータに関するエラー検出について規定されていない．

2.12.7. オブジェクトの ID 番号の指定

コンフィギュレータのオプション機能として，アプリケーション設計者がオブジェクトの ID 番号を指定するための次の機能を用意する．

コンフィギュレータのオプション指定により，オブジェクト識別名と ID 番号の対応表を含むファイルを渡すと，コンフィギュレータはそれに従ってオブジェクトに ID 番号を割り付ける．それに従った ID 番号割付けができない場合（ID 番号に抜けができる場合など）には，コンフィギュレータはエラーを報告する．

またコンフィギュレータは，オプション指定により，オブジェクト識別名とコンフィギュレータが割り付けた ID 番号の対応表を含むファイルを，コンフィギュレータに渡すファイルと同じフォーマットで生成する．

【μITRON4.0 仕様との関係】
μITRON4.0 仕様では，オブジェクト生成のための静的 API の ID 番号を指定するパラメータに整数値を記述できるため，このような機能は用意されていない．

2.13. TOPPERS ネーミングコンベンション

この節では，TOPPERS ソフトウェアの API の構成要素の名称に関するネーミングコンベンションについて述べる．このネーミングコンベンションは，モジュール間のインタフェースに関わる名称に適用することを想定しているが，モジュール内部の名称に適用してもよい．

2.13.1. モジュール識別名

異なるモジュールの API の構成要素の名称が衝突することを避けるために，各モジュールに対して，それを識別するためのモジュール識別名を定める．モジュール識別名は，英文字と数字で構成し，2～8 文字程度の長さとする．

カーネルのモジュール識別名は”kernel”，システムインタフェースレイヤのモジュール識別名は”sil”とする．

API の構成要素の名称には，モジュール識別名を含めることを原則とするが，カーネルの API など，頻繁に使用されて衝突のおそれが少ない場合には，モジュール識別名を含めない名称を使用する．

以下では，モジュール識別名の英文字を英小文字としたものを www，英大文字としたものを WWW と表記する．

2.13.2. データ型名

各サイズの整数型など，データの意味を定めない基本データ型の名称は，英小文字，数字，”_”で構成する．データ型であることを明示するために，末尾が“_t”である名称とする．
複合データ型やデータの意味を定めるデータ型の名称は，英大文字，数字，“_”で構成する．データ型であることを明示するために，先頭が”T_”または末尾が”_T”である名称とする場合もある．
データ型の種類毎に，次のネーミングコンベンションを定める．

(A)　パケットのデータ型

T_CYYY	acre_yyy に渡すパケットのデータ型
T_DYYY	def_yyy に渡すパケットのデータ型
T_RYYY	ref_yyy に渡すパケットのデータ型
T_WWW_CYYY	www_acre_yyy に渡すパケットのデータ型
T_WWW_DYYY	www_def_yyy に渡すパケットのデータ型
T_WWW_RYYY	www_ref_yyy に渡すパケットのデータ型

2.13.3. 関数名

関数の名称は，英小文字，数字，”_”で構成する．関数の種類毎に，次のネーミングコンベンションを定める．

(A)　サービスコール

サービスコールは，xxx_yyy または www_xxx_yyy の名称とする．ここで，xxx は操作の方法，yyy は操作の対象を表す．xxx_yyy または www_xxx_yyy から派生したサー
ビスコールは，それぞれ zxxx_yyy または www_zxxx_yyy の名称とする．ここで z は，派生したことを表す文字である．派生したことを表す文字を 2 つ付加する場合には，zzxxx_yyy または www_zzxxx_yyy

の名称となる．

非タスクコンテキスト専用のサービスコールの名称は，派生したことを表す文字として”i”を付加し，ixxx_yyy，izxxx_yyy，www_ixxx_yyy，www_izxxx_yyy といった名称とする．

【補足説明】

サービスコールの名称を構成する省略名（xxx，yyy，z）の元になった英語については，「5.9 省略名の元になった英語」の節を参照すること．

(B)　コールバック

コールバックの名称は，サービスコールのネーミングコンベンションに従う．

2.13.4. 変数名

変数（const 修飾子のついたものを含む）の名称は，英小文字，数字，”_”で構成する．データ型が異なる変数には，異なる名称を付けることを原則とする．変数の名称に関して，次のガイドラインを設ける．

～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）
～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_t 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～	～の残り

また，ポインタ変数（関数ポインタを除く）の名称に関して，次のガイドラインを設ける．

p_～	ポインタ
pp_～	ポインタを入れる領域へのポインタ
pk_～	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

変数の種類毎に，次のネーミングコンベンションを定める．

(A)　パケットへのポインタ

pk_cyyy	acre_yyy に渡すパケットへのポインタ
pk_dyyy	def_yyy に渡すパケットへのポインタ
pk_ryyy	ref_yyy に渡すパケットへのポインタ
pk_www_cyyy	www_acre_yyy に渡すパケットへのポインタ
pk_www_dyyy	www_def_yyy に渡すパケットへのポインタ
pk_www_ryyy	www_ref_yyy に渡すパケットへのポインタ

2.13.5. 定数名

定数（C 言語プリプロセッサのマクロ定義によるもの）の名称は，英大文字，数字，”_”で構成する．

定数の種類毎に，次のネーミングコンベンションを定める．

(A)　メインエラーコード

メインエラーコードは，先頭が”E_”である名称とする．

(B)　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

(C)　その他の定数

その他の定数は，先頭が TUU_または TUU_WWW_である名称とする．ここで UU は，定数の種類またはデータ型を表す．同じパラメータまたはリターンパラメータに用いられる定数の名称については，UU を同一にすることを原則とする．

また，定数の名称に関して，次のガイドラインを設ける．

TA_～	オブジェクトの属性値
TSZ_～	～のサイズ
TBIT_～	～のビット数
TMAX_～	～の最大値

2.13.6. マクロ名

マクロ（C 言語プリプロセッサのマクロ定義によるもの）の名称は，それが表す構成要素のネーミングコンベンションに従う．すなわち，関数を表すマクロは関数のネーミングコンベンションに，定数を表すマクロは定数のネーミングコンベンションに従う．ただし，簡単な関数を表すマクロや，副作用があるなどの理由でマクロであることを明示したい場合には，英大文字，数字，”_”で構成する場合もある．

マクロの種類毎に，次のネーミングコンベンションを定める．

(A)　構成マクロ

構成マクロの名称は，英大文字，数字，”_”で構成し，次のガイドラインを設ける．

TSZ_～	～のサイズ
TBIT_～	～のビット数
TMAX_～	～の最大値
TMIN_～	～の最小値

2.13.7. 静的 API 名

静的 API の名称は，英大文字，数字，”_”で構成し，対応するサービスコールの名称中の英小文字を英大文字で置き換えたものとする．対応するサービスコールがない場合には，サービスコールのネーミングコンベンションに従って定めた名称中の英小文字を英大文字で置き換えたものとする．

2.13.8. ファイル名

ファイルの名称は，英小文字，数字，”_”，”.”で構成する．英大文字と英小文字を区別しないファイルシステムに対応するために，英大文字は使用しない．また，”-“も使用しない．

ファイルの種類毎に，次のネーミングコンベンションを定める

.

(B)　ヘッダファイル

モジュールを用いるために必要な定義を含むヘッダファイルは，そのモジュールのモジュール識別名の末尾に”.h”を付加した名前（すなわち，www.h）とする．

2.13.9. モジュール内部の名称の衝突回避

モジュール内部の名称が，他のモジュール内部の名称と衝突することを避けるために，次のガイドラインを設ける．

モジュール内部に閉じて使われる関数や変数などの名称で，オブジェクトファイルのシンボル表に登録されて外部から参照できる名称は，C 言語レベルで，先頭が_www_または_WWW_である名称とする．例えば，カーネルの内部シンボルは，　C 言語レベルで，先頭が”_kernel_”または”_KERNEL_”である名称とする．

また，モジュールを用いるために必要な定義を含むヘッダファイル中に用いる名称で，それをインクルードする他のモジュールで使用する名称と衝突する可能性のある名称は，”TOPPERS_”で始まる名称とする．

2.14. TOPPERS 共通定義

TOPPERS ソフトウェアに共通に用いる定義を，TOPPERS 共通定義と呼ぶ．

2.14.1. TOPPERS 共通ヘッダファイル

TOPPERS 共通定義（共通データ型，共通定数，共通マクロ）は，TOPPERS 共通ヘッダファイル（t_stddef.h）およびそこからインクルードされるファイルに含まれている．TOPPERS 共通定義を用いる場合には，TOPPERS 共通ヘッダファイルをインクルードする．

TOPPERS 共通ヘッダファイルは，カーネルヘッダファイル（kernel.h）やシステムインタフェースレイヤヘッダファイル（sil.h）からインクルードされるため，これらのファイルをインクルードする場合には，TOPPERS 共通ヘッダファイルを直接インクルードする必要はない．

2.14.2. TOPPERS 共通データ型

C90 に規定されているデータ型以外で，TOPPERS ソフトウェアで共通に用いるデータ型は次の通りである．

int8_t	符号付き 8 ビット整数（オプション，C99 準拠）
uint8_t	符号無し 8 ビット整数（オプション，C99 準拠）
int16_t	符号付き 16 ビット整数（C99 準拠）
uint16_t	符号無し 16 ビット整数（C99 準拠）
int32_t	符号付き 32 ビット整数（C99 準拠）
uint32_t	符号無し 32 ビット整数（C99 準拠）
int64_t	符号付き 64 ビット整数（オプション，C99 準拠）
uint64_t	符号無し 64 ビット整数（オプション，C99 準拠）
int128_t	符号付き 128 ビット整数（オプション，C99 準拠）
uint128_t	符号無し 128 ビット整数（オプション，C99 準拠）

int_least8_t	8 ビット以上の符号付き整数（C99 準拠）
uint_least8_t	int_least8_t 型と同じサイズの符号無し整数（C99 準拠）
float32_t	IEEE754 準拠の 32 ビット単精度浮動小数点数（オプション）
double64_t	IEEE754 準拠の 64 ビット倍精度浮動小数点数（オプション）

bool_t	真偽値（true または false）
int_t	16 ビット以上の符号付き整数
uint_t	int_t 型と同じサイズの符号無し整数
long_t	32 ビット以上かつ int_t 型以上のサイズの符号付き整数
ulong_t	long_t 型と同じサイズの符号無し整数

intptr_t	ポインタを格納できるサイズの符号付き整数（C99 準拠
uintptr_t	intptr_t 型と同じサイズの符号無し整数（C99 準拠）

FN	機能コード（符号付き整数，int_t に定義）
ER	正常終了（E_OK）またはエラーコード（符号付き整数，int_t に定義）
ID	オブジェクトの ID 番号（符号付き整数，int_t に定義）
ATR	オブジェクト属性（符号無し整数，uint_t に定義）
STAT	オブジェクトの状態（符号無し整数，uint_t に定義）
MODE	サービスコールの動作モード（符号無し整数，uint_t に定義）
PRI	優先度（符号付き整数，int_t に定義）
SIZE	メモリ領域のサイズ（符号無し整数，ポインタを格納できるサイズの符号無し整数型に定義）

TMO	タイムアウト指定（符号付き整数，単位はミリ秒，int_t に定義）
RELTIM	相対時間（符号無し整数，単位はミリ秒，uint_t に定義）
SYSTIM	システム時刻（符号無し整数，単位はミリ秒，ulong_t に定義）
SYSUTM	性能評価用システム時刻（符号無し整数，単位はマイクロ秒，

FP	プログラムの起動番地（型の定まらない関数ポインタ）

ER_BOOL	エラーコードまたは真偽値（符号付き整数，int_t に定義）
ER_ID	エラーコードまたは ID 番号（符号付き整数，int_t に定義，負の ID 番号は格納できない）
ER_UINT	エラーコードまたは符号無し整数（符号付き整数，int_t に定義，符号無し整数を格納する場合の有効ビット数は uint_t より 1 ビット短い

MB_T	オブジェクト管理領域を確保するためのデータ型

ACPTN	アクセス許可パターン(符号無し 32 ビット整数, uint32_t に定義)
ACVCT	アクセス許可ベクタ

ここで，データ型が「A または B」とは，A か B のいずれかの値を取ることを示す．

例えば ER_BOOL は，エラーコードまたは真偽値のいずれかの値を取る．

int8_t，uint8_t，int64_t，uint64_t，int128_t，uint128_t，float32_t， double64_t が使用できるかどうかは，ターゲットシステムに依存する．これらが使用できるかどうかは，それぞれ，INT8_MAX，UINT8_MAX，INT64_MAX， UINT64_MAX，INT128_MAX，UINT128_MAX，FLOAT32_MAX，DOUBLE64_MAX がマクロ定義されているかどうかで判別することができる．IEEE754 準拠の浮動小数点数がサポートされていないターゲットシステムでは, float32_t と double64_t は使用できないものとする．

【μITRON4.0 仕様との関係】

B，UB，H，UH，W，UW，D，UD，VP_INT に代えて，C99 準拠の int8_t，uint8_t， int16_t, uint16_t, int32_t, uint32_t, int64_t, uint64_t, intptr_t を用いることにした. また, uintptr_t, int128_t, uint128_t を用意することにした．

VP は，void *と等価であるため，用意しないことにした．また，ターゲットシステムにより振舞いが一定しないことから，VB，VH，VW，VD に代わるデータ型は用意しないことにした．

INT，UINT に代えて，C99 の型名と相性が良い int_t，uint_t を用いることにした．また，32 ビット以上かつ int_t 型（または uint_t 型）以上のサイズが保証される整数型として，long_t，ulong_t を用意し，8 ビット以上のサイズで必ず存在する整数型として，C99 準拠の int_least8_t，uint_least8_t を導入することにした．int_least16_t，uint_least16_t，int_least32_t，uint_least32_t を導入しなかったのは，16 ビットおよび 32 ビットの整数型があることを仮定しており，それぞれ int16_t, uint16_t, int32_t, uint32_t で代用できるためである．

TECS との整合性を取るために，BOOL に代えて，bool_t を用いることにした．また，IEEE754 準拠の単精度浮動小数点数を表す型として float32_t, IEEE754 準拠の 64 ビットを表す型として double64_t を導入した．

性能評価用システム時刻のためのデータ型として SYSUTM を，オブジェクト管理領域を確保するためのデータ型として MB_T を用意することにした

2.14.3. TOPPERS 共通定数

C90 に規定されている定数以外で，TOPPERS ソフトウェアで共通に用いる定数は次の通りである（一部，C90 に規定されているものも含む）．

(1) 一般定数

NULL	-	無効ポインタ
true	1	真
false	0	偽
E_OK	0	正常終了

【μITRON4.0 仕様との関係】

BOOL を bool_t に代えたことから，TRUE および FALSE に代えて，true および false を用いることにした．

(2) 整数型に格納できる最大値と最小値

INT8_MAX	int8_t に格納できる最大値（オプション，C99 準拠）
INT8_MIN	int8_t に格納できる最小値（オプション，C99 準拠）
UINT8_MAX	uint8_t に格納できる最大値（オプション，C99 準拠）
INT16_MAX	int16_t に格納できる最大値（C99 準拠）
INT16_MIN	int16_t に格納できる最小値（C99 準拠）
UINT16_MAX	uint16_t に格納できる最大値（C99 準拠）
INT32_MAX	int32_t に格納できる最大値（C99 準拠）
INT32_MIN	int32_t に格納できる最小値（C99 準拠）
UINT32_MAX	uint32_t に格納できる最大値（C99 準拠）
INT64_MAX	int64_t に格納できる最大値（オプション，C99 準拠）
INT64_MIN	int64_t に格納できる最小値（オプション，C99 準拠）
UINT64_MAX	uint64_t に格納できる最大値（オプション，C99 準拠）
INT128_MAX	int128_t に格納できる最大値（オプション，C99 準拠）
INT128_MIN	int128_t に格納できる最小値（オプション，C99 準拠）
UINT128_MAX	uint128_t に格納できる最大値（オプション，C99 準拠）

INT_LEAST8_MAX	int_least8_t に格納できる最大値（C99 準拠）
INT_LEAST8_MIN	int_least8_t に格納できる最小値（C99 準拠）
UINT_LEAST8_MAX	uint_least8_t に格納できる最大値（C99 準拠）
INT_MAX	int_t に格納できる最大値（C90 準拠）
INT_MIN	int_t に格納できる最小値（C90 準拠）
UINT_MAX	uint_t に格納できる最大値（C90 準拠）
LONG_MAX	long_t に格納できる最大値（C90 準拠）
LONG_MIN	long_t に格納できる最小値（C90 準拠）
ULONG_MAX	ulong_t に格納できる最大値（C90 準拠）

FLOAT32_MIN	float32_t に格納できる最小の正規化された正の浮動小数点数（オプション）
FLOAT32_MAX	float32_t に格納できる表現可能な最大の有限浮動小数点数（オプション）
DOUBLE64_MIN	double64_t に格納できる最小の正規化された正の浮動小数点数（オプション）
DOUBLE64_MAX	double64_t に格納できる表現可能な最大の有限浮動小数点数（オプション）

(3) 整数型のビット数

CHAR_BIT	char 型のビット数（C90 準拠）

(4) オブジェクト属性

TA_NULL	0U	オブジェクト属性を指定しない

(5) 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

(6) アクセス許可パターン

TACP_KERNEL	0U	カーネルドメインのみにアクセスを許可
TACP_SHARED	~0U	べての保護ドメインにアクセスを許可

2.14.4. TOPPERS 共通エラーコード

TOPPERS ソフトウェアで共通に用いるメインエラーコードは次の通りである．

(A) 内部エラークラス（EC_SYS，-5～-8）

E_SYS	-5	システムエラー

(B) 未サポートエラークラス（EC_NOSPT，-9～-16）

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

(C) パラメータエラークラス（EC_PAR，-17～-24）

E_PAR	-17	パラメータエラー
E_ID	-18	不正 ID 番号

(D) 呼出しコンテキストエラークラス（EC_CTX，-25～-32）

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)　資源不足エラークラス（EC_NOMEM，-33～-40）

E_NOMEM	-33	メモリ不足
E_NOID	-34	ID 番号不足
E_NORES	-35	資源不足

(F)　オブジェクト状態エラークラス（EC_OBJ，-41～-48）

E_OBJ	-41	オブジェクト状態エラー
E_NOEXS	-42	オブジェクト未登録
E_QOVR	-43	キューイングオーバフロー

(G)　待ち解除エラークラス（EC_RLWAI，-49～-56）

E_RLWAI	-49	待ち禁止状態または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または再初期化
E_CLS	-52	待ちオブジェクトの状態変化

(H)　警告クラス（EC_WARN，-57～-64）

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

このエラークラスに属するエラーコードは，警告を表すエラーコードであり，サービスコールがエラーコードを返した場合には副作用がないという原則の例外となる．

【μITRON4.0 仕様との関係】

E_NORES は，μITRON4.0 仕様に規定されていないエラーコードである．

2.14.5. TOPPERS 共通マクロ

(1) 整数定数を作るマクロ

INT8_C(val)	int_least8_t 型の定数を作るマクロ（C99 準拠）
UINT8_C(val)	uint_least8_t 型の定数を作るマクロ（C99 準拠）
INT16_C(val)	int16_t 型の定数を作るマクロ（C99 準拠）
UINT16_C(val)	uint16_t 型の定数を作るマクロ（C99 準拠）
INT32_C(val)	int32_t 型の定数を作るマクロ（C99 準拠）
UINT32_C(val)	uint32_t 型の定数を作るマクロ（C99 準拠）
INT64_C(val)	int64_t 型の定数を作るマクロ（オプション，C99 準拠）
UINT64_C(val)	uint64_t 型の定数を作るマクロ（オプション，C99 準拠）
INT128_C(val)	int128_t 型の定数を作るマクロ（オプション，C99 準拠）
UINT128_C(val)	uint128_t 型の定数を作るマクロ（オプション，C99 準拠）

UINT_C(val)	uint_t 型の定数を作るマクロ
ULONG_C(val)	ulong_t 型の定数を作るマクロ

【仕様決定の理由】

C99 に用意されていない UINT_C と ULONG_C を導入したのは，アセンブリ言語からも参照する定数を記述するためである．C 言語のみで用いる定数をこれらのマクロを使って記述する必要はない．

(2) 型に関する情報を取り出すためのマクロ

offsetof(structure, field)	構造体 structure 中のフィールド field のバイト位置を返すマクロ（C90 準拠）
alignof(type)	型 type のアラインメント単位を返すマクロ
ALIGN_TYPE(addr, type)	番地 addr が型 type に対してアラインしているかどうかを返すマクロ

(3) assert マクロ

assert(exp)	exp が成立しているかを検査するマクロ（C90 準拠）

(4)　コンパイラの拡張機能のためのマクロ

inline	インライン関数
Inline	ファイルローカルなインライン関数
asm	インラインアセンブラ
Asm	インラインアセンブラ（最適化抑止）
throw()	例外を発生しない関数
NoReturn	リターンしない関数

(5) エラーコード構成・分解マクロ

ERCD(mercd, sercd)	メインエラーコード mercd とサブエラーコード sercd から，エラーコードを構成するためのマクロ
MERCD(ercd)	エラーコード ercd からメインエラーコードを抽出するためのマクロ
SERCD(ercd)	エラーコード ercd からサブエラーコードを抽出するためのマクロ

(6) アクセス許可パターン構成マクロ

TACP(domid)	domid で指定されるユーザドメインのみにアクセスを許可するアクセス許可パターンを構成するためのマクロ

ここで，TACP のパラメータ（domid）には，ユーザドメインの ID 番号のみを指定することができる．TDOM_SELF，TDOM_KERNEL，TDOM_NONE を指定した場合の動作は，保証されない．

2.14.6. TOPPERS 共通構成マクロ

(1) 相対時間の範囲

TMAX_RELTIM	相対時間に指定できる最大値

2.15. カーネル共通定義

カーネルの複数の機能で共通に用いる定義を，カーネル共通定義と呼ぶ．

2.15.1. カーネルヘッダファイル

カーネルを用いるために必要な定義は，カーネルヘッダファイル（kernel.h）およびそこからインクルードされるファイルに含まれている．カーネルを用いる場合には，カーネルヘッダファイルをインクルードする．

ただし，カーネルを用いるために必要な定義の中で，コンフィギュレータによって生成されるものは，

カーネル構成・初期化ヘッダファイル（kernel_cfg.h）に含まれる．具体的には，登録できるオブジェクトの数（TNUM_YYY）やオブジェクトの ID 番号などの定義が，これに該当する．これらの定義を用いる場合には，カーネル構成・初期化ヘッダファイルをインクルードする．

μITRON4.0 仕様で規定されており，この仕様で廃止されたデータ型および定数を用いる場合には，ITRON 仕様互換ヘッダファイル（itron.h）をインクルードする．

【μITRON4.0 仕様との関係】

この仕様では，コンフィギュレータが生成するヘッダファイルに，オブジェクトの ID 番号の定義に加えて，登録できるオブジェクトの数（TNUM_YYY）の定義が含まれることとした．これに伴い，ヘッダファイルの名称を，μITRON4.0 仕様の自動割付け結果ヘッダファイル（kernel_id.h）から，カーネル構成・初期化ヘッダファイル（kernel_cfg.h）に変更した．

2.15.2. カーネル共通定数

(1) オブジェクト属性

TA_TPRI	0x01U	タスクの待ち行列をタスクの優先度順に

【μITRON4.0 仕様との関係】

値が 0 のオブジェクト属性（TA_HLNG，TA_TFIFO，TA_MFIFO，TA_WSGL）は，デフォルトの扱いにして廃止した．これは，「(tskatr & TA_HLNG) != 0U」のような間違いを防ぐためである．TA_ASM は，有効な使途がないために廃止した． TA_MPRI は，メールボックス機能でのみ使用するため，カーネル共通定義から外した．

(2) 保護ドメイン ID

TDOM_SELF	0	自タスクの属する保護ドメイン
TDOM_KERNEL	-1	カーネルドメイン
TDOM_NONE	-2	無所属（保護ドメインに属さない）

(3) その他のカーネル共通定数

TCLS_SELF	0	自タスクの属するクラス

TPRC_NONE	0	割付けプロセッサの指定がない
TPRC_INI	0	初期割付けプロセッサ

TSK_SELF	0	自タスク指定
TSK_NONE	0	該当するタスクがない

TIPM_ENAALL	0	割込み優先度マスク全解除

(4) カーネルで用いるメインエラーコード

「2.14.4TOPPERS 共通エラーコード」の節で定義したメインエラーコードの中で, E_CLS, E_WBLK, E_BOVR の 3 つは，カーネルでは使用しない.

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは, サービスコールから, E_RSFN, E_RSATR, E_MACV, E_OACV， E_NOMEM, E_NOID，E_NORES，E_NOEXS が返る状況は起こらない．E_RSATR は，コンフィギュレータによって検出される. ただし, 動的生成機能拡張パッケージでは, E_RSATR, E_NOMEM, E_NOID, E_NOEXS が返る状況が起こる.

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは, サービスコールから, E_RSFN, E_RSATR, E_MACV, E_OACV， E_NOMEM, E_NOID，E_NORES，E_NOEXS が返る状況は起こらない．E_RSATR と E_NORES は，コンフィギュレータによって検出される.

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，サービスコールから，E_RSATR，E_NOID，E_NORES， E_NOEXS が返る状況は起こらない．E_RSATR は，コンフィギュレータによって検出される.

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，サービスコールから，E_RSFN，E_RSATR，E_MACV，E_OACV， E_ILUSE, E_NOMEM，E_NOID，E_NORES，E_NOEXS，E_RLWAI，E_TMOUT，E_DLT が返る状況は起こらない．E_RSATR は，コンフィギュレータによって検出される.

2.15.3. カーネル共通マクロ

(1) スタック領域をアプリケーションで確保するためのデータ型とマクロ

スタック領域をアプリケーションで確保するために，次のデータ型とマクロを用意している.

STK_T	スタック領域を確保するためのデータ型

COUNT_STK_T(sz)	サイズ sz のスタック領域を確保するために必要な STK_T 型の配列の要素数
ROUND_STK_T(sz)	要素数 COUNT_STK_T(sz)の STK_T 型の配列のサイズ（sz を，STK_T 型のサイズの倍数になるように大きい方に丸めた値）

これらを用いてスタック領域を確保する方法は次の通り．

```
STK_T      <スタック領域の変数名>[COUNT_STK_T(<スタック領域のサイズ>)];
```

この方法で確保したスタック領域を，サービスコールまたは静的 API に渡す場合には，スタック領域の先頭番地に<スタック領域の変数名>を，スタック領域のサイズに ROUND_STK_T(<スタック領域のサイズ>)を指定する．

ただし，保護機能対応カーネルにおいては，上の方法によりタスクのユーザスタック領域を確保することはできない．詳しくは，「4.1 タスク管理機能」の節の CRE_TSK の機能の項を参照すること．

(2) オブジェクト属性を作るマクロ

保護機能対応カーネルでは，オブジェクトが属する保護ドメインを指定するためのオブジェクト属性を作るマクロとして，次のマクロを用意している．

TA_DOM(domid)	domid で指定される保護ドメインに属する

マルチプロセッサ対応カーネルでは，オブジェクトが属するクラスを指定するためのオブジェクト属性を作るマクロとして，次のマクロを用意している．

TA_CLS(clsid)	svc で指定されるサービスコールを関数呼出しによって呼び出すための名称

(3) サービスコールの呼出し方法を指定するマクロ

保護機能対応カーネルでは，サービスコールの呼出し方法を指定するためのマクロとして，次のマクロを用意している．

SVC_CALL(svc)	svc で指定されるサービスコールを関数呼出しによって呼び出すための名称

2.15.4. カーネル共通構成マクロ

(1) サポートする機能

TOPPERS_SUPPORT_PROTECT	保護機能対応のカーネル
TOPPERS_SUPPORT_MULTI_PRC	マルチプロセッサ対応のカーネル
TOPPERS_SUPPORT_DYNAMIC_CRE	動的生成対応のカーネル

【未決定事項】

マクロ名は，今後変更する可能性がある．

(2) 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

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，タスク優先度の最大値（TMAX_TPRI）は 16 に固定されている．ただし，タスク優先度拡張パッケージを用いると，TMAX_TPRI を 256 に拡張することができる．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，タスク優先度の最大値（TMAX_TPRI）は 16 に固定されている．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，タスク優先度の最大値（TMAX_TPRI）は 16 に固定されている．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，タスク優先度の最大値（TMAX_TPRI）は 16 に固定されている．

【μITRON4.0 仕様との関係】

メッセージ優先度の最小値（TMIN_MPRI）と最大値（TMAX_MPRI）は，メールボックス機能でのみ使用するため，カーネル共通定義から外した．

(3) プロセッサの数

マルチプロセッサ対応カーネルでは，プロセッサの数を知るためのマクロとして，次の構成マクロを用意している．

TNUM_PRCID	プロセッサの数

(4) 特殊な役割を持ったプロセッサ

マルチプロセッサ対応カーネルでは，特殊な役割を持ったプロセッサを知るためのマクロとして，次の構成マクロを用意している．

TOPPERS_MASTER_PRCID	マスタプロセッサの ID 番号
TOPPERS_SYSTIM_PRCID	システム時刻管理プロセッサの ID 番号（グローバルタイマ方式の場合のみ）

(5) タイマ方式

マルチプロセッサ対応カーネルでは，システム時刻の方式を知るためのマクロとして，次の構成マクロを用意している．

TOPPERS_SYSTIM_LOCAL	ローカルタイマ方式の場合にマクロ定義
TOPPERS_SYSTIM_GLOBAL	グローバルタイマ方式の場合にマクロ定義

(6) バージョン情報

TKERNEL_MAKER	カーネルのメーカコード（＝0x0118）
TKERNEL_PRID	カーネルの識別番号
TKERNEL_SPVER	カーネル仕様のバージョン番号
TKERNEL_PRVER	カーネルのバージョン番号

カーネルのメーカコード（TKERNEL_MAKER）は，TOPPERS プロジェクトから配布するカーネルでは，TOPPERS プロジェクトを表す値（0x0118）に設定されている．カーネルの識別番号（TKERNEL_PRID）は，TOPPERS カーネルの種類を表す．

0x0001	TOPPERS/JSP カーネル
0x0002	予約（IIMP カーネル）
0x0003	予約（IDL カーネル）
0x0004	TOPPERS/FI4 カーネル
0x0005	TOPPERS/FDMP カーネル
0x0006	TOPPERS/HRP カーネル
0x0007	TOPPERS/ASP カーネル
0x0008	TOPPERS/FMP カーネル
0x0009	TOPPERS/SSP カーネル

カーネル仕様のバージョン番号（TKERNEL_SPVER）は，上位 8 ビット（0xf5）が TOPPERS 新世代カーネル仕様であることを，中位 4 ビットがメジャーバージョン番号，下位 4 ビットがマイナーバージョン番号を表す．

カーネルのバージョン番号（TKERNEL_PRVER）は，上位 4 ビットがメジャーバージョン番号，中位 8 ビットがマイナーバージョン番号，下位 4 ビットがパッチレベルを表す.

3. システムインタフェースレイヤ API 仕様

3.1. システムインタフェースレイヤの概要

システムインタフェースレイヤ（この章では，SIL と略記する）は，デバイスを直接操作するプログラムが用いるための機能である．ITRON デバイスドライバ設計ガイドラインの一部分として検討されたものをベースに，TOPPERS プロジェクトにおいて修正を加えて用いている．

SIL の機能は，プロセッサの特権モードで実行されているプログラムが使用することを想定している．非特権モードで実行されているプログラムから SIL の機能を呼び出した場合の動作は，次の例外を除いては保証されない．

- ・微少時間待ちの機能を呼び出すこと
- ・エンディアンの取得のためのマクロを参照すること
- ・メモリ空間アクセス関数により，アクセスを許可されたメモリ領域にアクセスすること
- ・I/O 空間アクセス関数により，アクセスを許可された I/O 領域にアクセスすること

3.2. SIL ヘッダファイル

SIL を用いるために必要な定義は，SIL ヘッダファイル（sil.h）およびそこからインクルードされるファイルに含まれている．SIL を用いる場合には，SIL ヘッダファイルをインクルードする．

3.3. 全割込みロック状態の制御

デバイスを扱うプログラムの中では，すべての割込み（NMI を除く，以下同じ）をマスクしたい場合がある．カーネルで制御できる CPU ロック状態は，カーネル管理外の割込み（NMI 以外にカーネル管理外の割込みがあるかはターゲット定義）をマスクしないため，このような場合に用いることはできない．

そこで，SIL では，すべての割込みをマスクする全割込みロック状態を制御するための以下の機能を用意している．

(1) SIL_PRE_LOC

全割込みロック状態の制御に必要な変数を宣言するマクロ．通常は，型と変数名を並べたもので，最後に";"を含まない．

このマクロは，SIL_LOC_INT，SIL_UNL_INT を用いる関数またはブロックの先頭の変数宣言部に記述しなければならない．SIL_LOC_INT，SIL_UNL_INT を 1 つの関

数内でネストして用いることは可能であるが，その場合には，ネストレベル毎にブロックを作り，そのブロックの先頭の変数宣言部に SIL_PRE_LOC を記述しなければならない．そのように記述しなかった場合の動作は保証されない．

(2) SIL_LOC_INT()

全割込みロックフラグをセットすることで，NMI を除くすべての割込みをマスクし，全割込みロック状態に遷移する．

(3) SIL_UNL_INT()

全割込みロックフラグを，対応する SIL_LOC_INT を実行する前の状態に戻す．
SIL_LOC_INT を実行せずに SIL_UNL_INT を呼び出した場合の動作は保証されない．

なお，全割込みロック状態で呼び出せるサービスコールなどの制限事項については，「2.5.4 全割込みロック状態と全割込みロック解除状態」の節を参照すること．

【補足説明】

全割込みロック状態の制御機能の使用例は次の通り．

```
{
    SIL_PRE_LOC;

    SIL_LOC_INT();
    // この間は NMI を除くすべての割込みがマスクされる．
    // この間にサービスコールを呼び出してはならない（一部例外あり）．
    SIL_UNL_INT();
}
```

3.4. SIL スピンロック

マルチプロセッサシステムにおいて，カーネルの機能を用いずに，他のプロセッサとの間でも排他制御を実現したい場合がある．そこで SIL では，割込みのマスクとプロセッサ間ロックの取得により排他制御を行うためのスピンロックの機能を用意している．これを，カーネルのスピンロック機能と区別するために，SIL スピンロックと呼ぶ．

プロセッサ間ロックを取得している間は，全割込みロック状態にすることですべての割込み（NMI を除く）がマスクされる．ロックが他のプロセッサに取得されている場合には，ロックが取得できるまでループによって待つ．ロックの取得を待つ間は，割込みはマスクされない（ロックの取得を試みる前にマスクしていた割込みは，マスク解除されない）．プロセッサ間ロックを取得し割込みをマスクすることを，SIL スピンロックを取得するという．また，プロセッサ間ロックを返却し割込みをマスク解除する

ことを，SIL スピンロックを返却するという．

SIL で取得・返却するプロセッサ間ロックは，システムに唯一存在する．

(1) SIL_PRE_LOC

全割込みロック状態の制御に必要な変数を宣言するマクロであるが，SIL スピンロックの取得・解放にも兼用する．

このマクロは，SIL_LOC_SPN，SIL_UNL_SPN を用いる関数またはブロックの先頭の変数宣言部に記述しなければならない．SIL_LOC_SPN，SIL_UNL_SPN を，同じ関数内の SIL_LOC_INT，SIL_UNL_INT とネストして用いることは可能であるが，その場合には，ネストレベル毎にブロックを作り，そのブロックの先頭の変数宣言部に SIL_PRE_LOC を記述しなければならない．そのように記述しなかった場合の動作は保証されない．

(2) SIL_LOC_SPN()

SIL スピンロックが取得されていない状態である場合には，プロセッサ間ロックの取得を試みる．ロックが他のプロセッサに取得されている状態である場合や，他のプロセッサがロックの取得に成功した場合には，ロックが返却されるまでループによって待ち，返却されたらロックの取得を試みる．ロックの取得に成功した場合には，全割込みロックフラグをセットし，全割込みロック状態に遷移する．

(3) SIL_UNL_SPN()

プロセッサ間ロックを返却し，全割込みロックフラグを対応する SIL_LOC_SPN を実行する前の状態に戻す．

SIL スピンロックを取得している状態で SIL_LOC_SPN を呼び出した場合の動作は保証されない．逆に，SIL スピンロックを取得していない状態で SIL_UNL_SPN を呼び出した場合の動作も保証されない．

なお，SIL スピンロック取得中は全割込みロック状態となっているため，SIL スピンロック取得中に呼び出せるサービスコールなどについては，「2.5.4 全割込みロック状態と全割込みロック解除状態」の節の制限事項が適用される．

なお，マルチプロセッサシステム以外では，SIL_LOC_SPN と SIL_UNL_SPN は用意されていない．

【使用上の注意】

全割込ロック状態や CPU ロック状態で SIL_LOC_SPN を呼び出すことはできるが，割込みがマスクされている時間が長くなるために，そのような使い方は避けるべきである．

【補足説明】

SIL スピンロック機能の使用例は次の通り.

```
{
        SIL_PRE_LOC;

        SIL_LOC_SPN();
        // この間は SIL スピンロックを取得している.
        // この間は NMI を除くすべての割込みがマスクされる.
        // この間にサービスコールを呼び出してはならない (一部例外あり).
        SIL_UNL_SPN();
}
```

3.5. 微少時間待ち

デバイスをアクセスする際に，微少な時間待ちを入れなければならない場合がある．そのような場合に, NOP 命令をいくつか入れるなどの方法で対応すると, ポータビリティを損なうことになる. そこで, SIL では，微少な時間待ちを行うための以下の機能を用意している.

(1) void sil_dly_nse(ulong_t dlytim)

dlytim で指定された以上の時間（単位はナノ秒)，ループなどによって待つ．指定した値によっては，指定した時間よりもかなり長く待つ場合があるので注意すること.

3.6. エンディアンの取得

プロセッサのバイトエンディアンを取得するためのマクロとして，SIL では，以下のマクロを定義している.

(1) SIL_ENDIAN_BIG，SIL_ENDIAN_LITTLE

ビッグエンディアンプロセッサでは SIL_ENDIAN_BIG を，リトルエンディアンプロセッサでは SIL_ENDIAL_LITTLE を，マクロ定義している.

3.7. メモリ空間アクセス関数

メモリ空間にマッピングされたデバイスレジスタや，デバイスとの共有メモリをアクセスするために，SIL では，以下の関数を用意している.

(1) uint8_t sil_reb_mem(const uint8_t *mem)

mem で指定されるアドレスから 8 ビット単位で読み出した値を返す.

(2) void sil_wrb_mem(uint8_t *mem, uint8_t data)
mem で指定されるアドレスに data で指定される値を 8 ビット単位で書き込む.

(3) uint16_t sil_reh_mem(const uint16_t *mem)
mem で指定されるアドレスから 16 ビット単位で読み出した値を返す.

(4) void sil_wrh_mem(uint16_t *mem, uint16_t data)
mem で指定されるアドレスに data で指定される値を 16 ビット単位で書き込む.

(5) uint16_t sil_reh_lem(const uint16_t *mem)
mem で指定されるアドレスから 16 ビット単位でリトルエンディアンで読み出した値を返す．リトルエンディアンプロセッサでは，sil_reh_mem と一致する．ビッグエンディアンプロセッサでは，sil_reh_mem が返す値を，エンディアン変換した値を返す.

(6) void sil_wrh_lem(uint16_t *mem, uint16_t data)
mem で指定されるアドレスに data で指定される値を 16 ビット単位でリトルエンディアンで書き込む.リトルエンディアンプロセッサでは，sil_wrh_mem と一致する．ビッグエンディアンプロセッサでは，data をエンディアン変換した値を，sil_wrh_mem で書き込むのと同じ結果となる.

(7) uint16_t sil_reh_bem(const uint16_t *mem)
mem で指定されるアドレスから 16 ビット単位でビッグエンディアンで読み出した値を返す．ビッグエンディアンプロセッサでは，sil_reh_mem と一致する．リトルエンディアンプロセッサでは，sil_reh_mem が返す値を，エンディアン変換した値を返す.

(8) void sil_wrh_bem(uint16_t *mem, uint16_t data)
mem で指定されるアドレスに data で指定される値を 16 ビット単位でビッグエンディアンで書き込む.ビッグエンディアンプロセッサでは，sil_wrh_mem と一致する．リトルエンディアンプロセッサでは，data をエンディアン変換した値を，sil_wrh_mem で書き込むのと同じ結果となる.

(9) uint32_t sil_rew_mem(const uint32_t *mem)
mem で指定されるアドレスから 32 ビット単位で読み出した値を返す.

(10) void sil_wrw_mem(uint32_t *mem, uint32_t data)
mem で指定されるアドレスに data で指定される値を 32 ビット単位で書き込む.

(11) uint32_t sil_rew_lem(const uint32_t *mem)
mem で指定されるアドレスから 32 ビット単位でリトルエンディアンで読み出した値を返す．リトルエンディアンプロセッサでは，sil_rew_mem と一致する．ビッグエンディアンプロセッサでは，

sil_rew_mem が返す値を，エンディアン変換した値を返す.

(12)　void sil_wrw_lem(uint32_t *mem, uint32_t data)
mem で指定されるアドレスに data で指定される値を 32 ビット単位でリトルエンディアンで書き込む. リトルエンディアンプロセッサでは，sil_wrw_mem と一致する．ビッグエンディアンプロセッサでは，data をエンディアン変換した値を，sil_wrw_mem で書き込むのと同じ結果となる.

(13)　uint32_t sil_rew_bem(const uint32_t *mem)
mem で指定されるアドレスから 32 ビット単位でビッグエンディアンで読み出した値を返す．ビッグエンディアンプロセッサでは，sil_rew_mem と一致する．リトルエンディアンプロセッサでは，sil_rew_mem が返す値を，エンディアン変換した値を返す.

(14)　void sil_wrw_bem(uint32_t *mem, uint32_t data)
mem で指定されるアドレスに data で指定される値を 32 ビット単位でビッグエンディアンで書き込む. ビッグエンディアンプロセッサでは，sil_wrw_mem と一致する．リトルエンディアンプロセッサでは，data をエンディアン変換した値を，sil_wrw_mem で書き込むのと同じ結果となる.

3.8. I/O 空間アクセス関数

メモリ空間とは別に I/O 空間を持つプロセッサでは，I/O 空間にあるデバイスレジスタをアクセスするために，メモリ空間アクセス関数と同等の以下の関数を用意している.

(1) uint8_t sil_reb_iop(const uint8_t *iop)
(2) void sil_wrb_iop(uint8_t *iop, uint8_t data)
(3) uint16_t sil_reh_iop(const uint16_t *iop)
(4) void sil_wrh_iop(uint16_t *iop, uint16_t data)
(5) uint16_t sil_reh_lep(const uint16_t *iop)
(6) void sil_wrh_lep(uint16_t *iop, uint16_t data)
(7) uint16_t sil_reh_bep(const uint16_t *iop)
(8) void sil_wrh_bep(uint16_t *iop, uint16_t data)
(9) uint32_t sil_rew_iop(const uint32_t *iop)
(10) void sil_wrw_iop(uint32_t *iop, uint32_t data)
(11) uint32_t sil_rew_lep(const uint32_t *iop)
(12) void sil_wrw_lep(uint32_t *iop, uint32_t data)
(13) uint32_t sil_rew_bep(const uint32_t *iop)
(14) void sil_wrw_bep(uint32_t *iop, uint32_t data)

3.9. プロセッサ ID の参照

マルチプロセッサシステムにおいては，プログラムがどのプロセッサで実行されているかを参照するために，以下の関数を用意している．

(1) void sil_get_pid(ID *p_prcid)

この関数を呼び出したプログラムを実行しているプロセッサの ID 番号を参照し，p_prcid で指定したメモリ領域に返す．

【使用上の注意】

タスクは, sil_get_pid を用いて, 自タスクを実行しているプロセッサを正しく参照できるとは限らない．これは，sil_get_pid を呼び出し，自タスクを実行しているプロセッサの ID 番号を参照した直後に割込みが発生した場合，sil_get_pid から戻ってきた時には自タスクを実行しているプロセッサが変化している可能性があるためである．

4. カーネル API 仕様

この章では，カーネルの API 仕様について規定する．

カーネルの API の種別と API をサポートするカーネルの種類を表すために，次の記号を用いる．

〔T〕	タスクコンテキスト専用のサービスコールを示す．非タスクコンテキストから呼び出すと，E_CTX エラーとなる
〔I〕	非タスクコンテキスト専用のサービスコールを示す．タスクコンテキストから呼び出すと，E_CTX エラーとなる
〔TI〕	タスクコンテキストからも非タスクコンテキストからも呼び出すことのできるサービスコールを示す．
〔S〕	静的 API を示す
〔P〕	保護機能対応カーネルのみでサポートされている API を示す．保護機能対応でないカーネルでは，この API はサポートされない．
〔p〕	保護機能対応でないカーネルのみでサポートされている API を示す．保護機能対応カーネルでは，この API はサポートされない
〔M〕	マルチプロセッサ対応カーネルのみでサポートされている API を示す．マルチプロセッサ対応でないカーネルでは，この API はサポートされない
〔D〕	動的生成対応カーネルのみでサポートされている API を示す．動的生成対応でないカーネルでは，この API はサポートされない

.

また，エラーコードが返る条件を表すために，次の記号を用いる．

〔s〕	サービスコールのみで返るエラーコードを示す．静的 API では，このエラーコードは返らない
〔S〕	静的 API のみで返るエラーコードを示す．サービスコールでは，このエラーコードは返らない．
〔P〕	保護機能対応カーネルのみで返るエラーコードを示す．保護機能対応でないカーネルでは，このエラーコードは返らない
〔D〕	動的生成対応カーネルのみで返るエラーコードを示す．動的生成対応でないカーネルでは，このエラーコードは返らない．

【μITRON4.0 仕様との関係】

TOPPERS 共通データ型に従い，パラメータのデータ型を次の通り変更した．これらの変更については，個別の API 仕様では記述しない．

- INT → int_t
- UINT → uint_t
- VP → void *
- VP_INT → intptr_t

【μITRON4.0/PX 仕様との関係】
ID 番号で識別するオブジェクトのアクセス許可ベクタをデフォルト以外に設定する場合には，オブジェクトを生成した後に設定することとし，アクセス許可ベクタを設定する静的 API（SAC_YYY）を新設した. 逆に, アクセス許可ベクタを指定してオブジェクトを生成する機能（CRA_YYY, cra_yyy, acra_yyy）は廃止した．これらの変更については，個別の API 仕様では記述しない.

4.1. タスク管理機能

タスクは，プログラムの並行実行の単位で，カーネルが実行を制御する処理単位である．タスクは，タスク ID と呼ぶ ID 番号によって識別する.

タスク管理機能に関連して，各タスクが持つ情報は次の通り.

- タスク属性
- タスク状態
- ベース優先度
- 現在優先度
- 起動要求キューイング数
- 割付けプロセッサ（マルチプロセッサ対応カーネルの場合）
- 次回起動時の割付けプロセッサ（マルチプロセッサ対応カーネルの場合）
- 拡張情報
- メインルーチンの先頭番地
- 起動時優先度
- 実行時優先度（TOPPERS/SSP カーネルの場合）
- スタック領域
- システムスタック領域（保護機能対応カーネルの場合）
- アクセス許可ベクタ（保護機能対応カーネルの場合）
- 属する保護ドメイン（保護機能対応カーネルの場合）
- 属するクラス（マルチプロセッサ対応カーネルの場合）

タスクのベース優先度は，タスクの現在優先度を決定するために使われる優先度であり，タスクの起動時に起動時優先度に初期化される.

タスクの現在優先度は，タスクの実行順位を決定するために使われる優先度である．単にタスクの優先度と言った場合には，現在優先度のことを指す．タスクがミューテックスをロックしていない間は，タスクの現在優先度はベース優先度に一致する．ミューテックスをロックしている間のタスクの現在優先度については，「4.4.6 ミューテックス」の節を参照すること．

タスクの起動要求キューイング数は，処理されていないタスクの起動要求の数であり，タスクの生成時に 0 に初期化される．

割付けプロセッサは，マルチプロセッサ対応カーネルにおいて，タスクを実行するプロセッサで，タスクの生成時に，タスクが属するクラスによって定まる初期割付けプロセッサに初期化される．

次回起動時の割付けプロセッサは，マルチプロセッサ対応カーネルにおいて，タスクが次に起動される時に割り付けられるプロセッサで，タスクの生成時に未設定の状態に初期化される．タスクの起動時に，次回起動時の割付けプロセッサが設定されていれば，タスクの割付けプロセッサがそのプロセッサに変更され，次回起動時の割付けプロセッサは未設定の状態に戻される．次回起動時の割付けプロセッサが未設定の場合には，タスクの割付けプロセッサは変更されない（つまり，タスクが前に実行されていたのと同じプロセッサで実行される）．

保護機能対応カーネルにおいては，スタック領域の扱いは，ユーザタスクとシステムタスクで異なる．ユーザタスクのスタック領域は，ユーザタスクが非特権モードで実行する間に用いるスタック領域であり，ユーザスタック領域と呼ぶ．その扱いについては，「2.11.6 ユーザタスクのユーザスタック領域」の節を参照すること．システムタスクのスタック領域は，カーネルの用いるオブジェクト管理領域と同様に扱われる．

システムスタック領域は，保護機能対応カーネルにおいて，ユーザタスクがサービスコール（拡張サービスコールを含む）を呼び出し，特権モードで実行する間に用いるスタック領域である．システムスタック領域は，カーネルの用いるオブジェクト管理領域と同様に扱われる．

タスク属性には，次の属性を指定することができる．

TA_ACT	0x02U	タスクの生成時にタスクを起動する
TA_RSTR	0x04U	生成するタスクを制約タスクとする

TA_ACT を指定しない場合，タスクの生成直後には，タスクは休止状態となるまた，ターゲットによっては，ターゲット定義のタスク属性を指定できる場合がある．ターゲット定義のタスク属性として，次の属性を予約している．

TA_FPU	－	FPU レジスタをコンテキストに含める

C 言語によるタスクの記述形式は次の通り．

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

exinf には，タスクの拡張情報が渡される．ext_tsk を呼び出さず，タスクのメインルーチンからリターンした場合，ext_tsk を呼び出した場合と同じ動作をする．

タスク管理機能に関連するカーネル構成マクロは次の通り．

TMAX_ACTCNT	タスクの起動要求キューイング数の最大値
TNUM_TSKID	登録できるタスクの数（動的生成対応でないカーネルでは，静的 API によって登録されたタスクの数に一致）

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，TMAX_ACTCNT は 1 に固定されている．また，制約タスクはサポートしていない．ただし，制約タスク拡張パッケージを用いると，制約タスクの機能を追加することができる．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，TMAX_ACTCNT は 1 に固定されている．また，制約タスクはサポートしていない．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，TMAX_ACTCNT は 1 に固定されている．また，制約タスクはサポートしていない．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，タスクの起動要求のキューイングはサポートしておらず，タスクに対して起動要求キューイング数の情報を持たない．また，TMAX_ACTCNT を定義していない．

また，制約タスクのみをサポートすることから，すべてのタスクでスタック領域を共有しており，タスク毎にスタック領域の情報を持たない．

SSP カーネルにおける追加機能として，タスクに対して，実行時優先度の情報を持つ．SSP カーネルにおいては，タスクが起動された後，最初に実行状態になる時に，タスクのベース優先度が，タスクの実行時優先度に設定される．実行時優先度の機能は，起動時優先度よりも高い優先度でタスクを実行することで，同時期に共有スタック領域を使用している状態になるタスクの組み合わせを限定し，スタッ

ク領域を節約するための機能である．

タスクの実行時優先度は，実行時優先度を定義する静的 API（DEF_EPR）によって設定する．実行時優先度を定義しない場合，タスクの実行時優先度は，起動時優先度と同じ値に設定される．

〔実行時優先度によるスタック領域の節約〕
いずれのタスクにも実行時優先度が設定されていない場合には，すべてのタスクが同時期に共有スタック領域を使用している状態になる可能性があるため，すべてのタスクのスタック領域のサイズの和に，非タスクコンテキスト用のスタック領域のサイズを加えたものが，共有スタック領域に必要なサイズとなる．

タスク A に対して実行時優先度が設定されており，タスク A の起動時優先度よりも高く，タスク A の実行時優先度と同じかそれよりも低い起動時優先度を持つタスク B がある場合，タスク A とタスク B は同時期に共有スタック領域を使用している状態にならない．そのため，タスク A とタスク B の内，サイズが小さい方のスタック領域のサイズは，共有スタック領域のサイズに加える必要がなくなり，スタック領域を節約できることになる．

【μITRON4.0 仕様との関係】
この仕様では，自タスクの拡張情報の参照するサービスコール（get_inf）をサポートし，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），タスクを終了と同時に削除するサービスコール（exd_tsk），タスクの状態を参照するサービスコールの簡易版（ref_tst）はサポートしないこととした．

TNUM_TSKID は，μITRON4.0 仕様に規定されていないカーネル構成マクロである．

CRE_TSK　タスクの生成〔S〕
acre_tsk　タスクの生成〔TD〕

【静的 API】

* 保護機能対応でないカーネルの場合

```
CRE_TSK(ID tskid, {ATR tskatr, intptr_t exinf, TASK task,
                                PRI itskpri, SIZE stksz, STK_T *stk})
```

* 保護機能対応カーネルの場合

```
CRE_TSK(ID tskid, {ATR tskatr, intptr_t exinf, TASK task,
        PRI itskpri, SIZE stksz, STK_T *stk, SIZE sstksz, STK_T *sstk})
※ sstksz および sstk の記述は省略することができる．
```

【C 言語 API】

```
ER_ID tskid = acre_tsk(const T_CTSK *pk_ctsk)
```

【パラメータ】

ID	tskid	生成するタスクの ID 番号（CRE_TSK の場合）
T_CTSK *	pk_ctsk	タスクの生成情報を入れたパケットへのポインタ（静的 API を除く）

＊タスクの生成情報（パケットの内容）

ATR	tskatr	タスク属性
intptr_t	exinf	タスクの拡張情報
TASK	task	タスクのメインルーチンの先頭番地
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）
STK_T *	stk	タスクのスタック領域の先頭番地
SIZE	sstksz	タスクのシステムスタック領域のサイズ（バイト数，保護機能対応カーネルの場合，静的 API
STK_T *	sstk	タスクのシステムスタック領域の先頭番地（保護機能対応カーネルの場合，静的 API においては省略可）

【リターンパラメータ】

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

【エラーコード】

E_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_RSATR	予約属性（tskatr が不正または使用できない，属する保護ドメインかクラスが不正）
E_PAR	パラメータエラー（task，itskpri，stksz，stk，sstksz，sstk が不正，その他の条件については機能の項を参照すること）
E_OACV〔sP〕	オブジェクトアクセス違反（システム状態に対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（pk_ctsk が指すメモリ領域への読出しアクセスが許可されていない）
E_NOID〔sD〕	ID 番号不足（割り付けられるタスク ID がない）
E_NOMEM	メモリ不足（スタック領域やシステムスタック領域が確保できない）
E_OBJ	オブジェクト状態エラー（tskid で指定したタスクが登録済み：CRE_TSK の場合，その他の条件については機能の項を参照すること）

【機能】

各パラメータで指定したタスク生成情報に従って，タスクを生成する．具体的な振舞いは以下の通り．

まず，stk と stksz からタスクが用いるスタック領域が設定される．stksz に 0 を指定した時や，ターゲット定義の最小値よりも小さい値を指定した時には，E_PAR エラーとなる．また，保護機能対応カーネルで，生成するタスクがユーザタスクの場合には，sstk と sstksz からシステムスタック領域が設定される．この場合，sstksz に 0 を指定した時や，ターゲット定義の最小値よりも小さい値を指定した時には，E_PAR エラーとなる．

次に，生成されたタスクに対してタスク生成時に行うべき初期化処理が行われ，生成されたタスクは休止状態になる．さらに，tskatr に TA_ACT を指定した場合には，タスク起動時に行うべき初期化処理が行われ，生成されたタスクは実行できる状態になる．

静的 API においては，tskid はオブジェクト識別名，tskatr，itskpri，stksz は整数定数式パラメータ，exinf，task，stk は一般定数式パラメータである．コンフィギュレータは，静的 API のメモリ不足（E_NOMEM）エラーを検出することができない．

itskpri は，TMIN_TPRI 以上，TMAX_TPRI 以下でなければならない．

〔stk に NULL を指定した場合〕

stk を NULL とした場合，stksz で指定したサイズのスタック領域が，コンフィギュレータまたはカーネルにより確保される．stksz にターゲット定義の制約に合致しないサイズを指定した時には，ターゲッ

ト定義の制約に合致するように大きい方に丸めたサイズで確保される．

保護機能対応カーネルにおいて，生成するタスクがユーザタスクの場合，コンフィギュレータまたはカーネルにより確保されるスタック領域（ユーザスタック領域）は，「2.11.6 ユーザタスクのユーザスタック領域」の節の規定に従って，メモリオブジェクトとしてカーネルに登録される．

静的 API において，生成するタスクが制約タスクの場合（tskatr に TA_RSTR を指定した場合），コンフィギュレータは，生成する制約タスクの起動時優先度毎にスタック領域を確保し，同じ起動時優先度を持つ制約タスクにそのスタック領域を共有させる．確保するスタック領域のサイズは，コンフィギュレータがスタック領域を確保し（stk に NULL を指定して生成され），同じ起動時優先度を持つ制約タスクのスタック領域のサイズ（stksz）の最大値となる．マルチプロセッサ対応カーネルでは，以上のスタック領域の確保処理を，制約タスクの初期割付けプロセッサ毎に行う．

〔stk に NULL 以外を指定した場合〕
stk に NULL 以外を指定した場合，stk と stksz で指定したスタック領域は，アプリケーションで確保しておく必要がある．スタック領域をアプリケーションで確保する方法については，「2.15.3 カーネル共通マクロ」の節を参照すること．その方法に従わず，stk や stksz にターゲット定義の制約に合致しない先頭番地やサイズを指定した時には，E_PAR エラーとなる．

保護機能対応カーネルにおいて，生成するタスクがシステムタスクの場合に，stk と stksz で指定したスタック領域がカーネル専用のメモリオブジェクトに含まれない場合，E_OBJ エラーとなる．

保護機能対応カーネルにおいて，生成するタスクがユーザタスクの場合，stk と stksz で指定したスタック領域（ユーザスタック領域）は，「2.11.6 ユーザタスクのユーザスタック領域」の節の規定に従って，メモリオブジェクトとしてカーネルに登録される．そのため，上の方法を用いてスタック領域を確保しても，ターゲット定義の制約に合致する先頭番地とサイズとなるとは限らず，スタック領域をアプリケーションで確保する方法は，ターゲット定義である．また，stk と stksz で指定したスタック領域が，登録済みのメモリオブジェクトとメモリ領域が重なる場合には，E_OBJ エラーとなる．

〔sstk と sstksz の扱い〕

保護機能対応カーネルにおける sstk と sstksz の扱いは，生成するタスクがユーザタスクの場合とシステムタスクの場合で異なる．

生成するタスクがユーザタスクの場合の扱いは次の通り．

sstk の記述を省略するか，sstk を NULL とした場合，sstksz で指定したサイズのシステムスタック領域が，コンフィギュレータまたはカーネルにより確保される．sstksz にターゲット定義の制約に合致し

ないサイズを指定した時には，ターゲット定義の制約に合致するように大きい方に丸めたサイズで確保される．sstksz の記述も省略した場合には，ターゲット定義のデフォルトのサイズで確保される．

sstk に NULL 以外を指定した場合，sstk と sstksz で指定したスタック領域は，アプリケーションで確保しておく必要がある．スタック領域をアプリケーションで確保する方法については，「2.15.3 カーネル共通マクロ」の節を参照すること．その方法に従わず，sstk や sstksz にターゲット定義の制約に合致しない先頭番地やサイズを指定した時には，E_PAR エラーとなる．また，stk と stksz で指定したシステムスタック領域がカーネル専用のメモリオブジェクトに含まれない場合，E_OBJ エラーとなる．

生成するタスクがシステムタスクの場合の扱いは次の通り．

sstk に指定することができるのは，NULL のみである．sstk に NULL 以外を指定した場合には，E_PAR エラーとなる．

sstksz に 0 以外の値を指定した場合で，stk が NULL の場合には，コンフィギュレータまたはカーネルにより確保されるスタック領域のサイズに，sstksz が加えられる．stksz に sstksz を加えた値が，ターゲット定義の制約に合致しないサイズになる時には，ターゲット定義の制約に合致するように大きい方に丸めたサイズで確保される．

sstksz に 0 以外の値を指定した場合で，stk が NULL でない場合には，E_PAR エラーとなる．

sstksz に 0 を指定した場合，これらの処理は行わず，E_PAR エラーにもならない．

【TOPPERS/ASP カーネルにおける規定】
ASP カーネルでは，CRE_TSK のみをサポートする．ただし，動的生成機能拡張パッケージでは，acre_tsk もサポートする．

【TOPPERS/FMP カーネルにおける規定】
FMP カーネルでは，CRE_TSK のみをサポートする．

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，CRE_TSK のみをサポートする．

【TOPPERS/SSP カーネルにおける規定】
SSP カーネルでは，CRE_TSK のみをサポートする．

SSP カーネルでは，複数のタスクに対して，同じ起動時優先度を設定することはできない．設定した場合には，コンフィギュレータが E_PAR エラーを報告する．

SSP カーネルでは，制約タスクのみをサポートするため，タスク属性に TA_RSTR を指定しない場合でも，生成されるタスクは制約タスクとなる．

SSP カーネルでは，stk には NULL を指定しなくてはならず，その場合でも，コンフィギュレータはタスクのスタック領域を確保しない．これは，SSP カーネルでは，すべての処理単位が共有スタック領域を使用し，タスク毎にスタック領域を持たないためである．stk に NULL 以外を指定した場合には，E_PAR エラーとなる．

共有スタック領域の設定方法については，DEF_STK の項を参照すること．

【μITRON4.0 仕様との関係】
task のデータ型を TASK に，stk のデータ型を STK_T *に変更した．COUNT_STK_T と ROUND_STK_T を新設し，スタック領域をアプリケーションで確保する方法を規定した．

【μITRON4.0/PX 仕様との関係】
sstk のデータ型を STK_T *に変更した．システムスタック領域をアプリケーションで確保する方法を規定した．

【未決定事項】
サービスコール（acre_tsk）により，stk に NULL を指定して制約タスクを生成した場合のスタック領域の確保方法については，今後の課題である．

【仕様決定の理由】
保護機能対応カーネルにおいて，sstksz および sstk の記述は省略することができることとしたのは，保護機能対応でないカーネル用のシステムコンフィギュレーションファイルを，保護機能対応カーネルにも変更なしに使えるようにするためである．

AID_TSK　　割付け可能なタスク ID の数の指定〔SD〕

【静的 API】

```
AID_TSK(uint_t notsk)
```

【パラメータ】

uint_t	notsk	割付け可能なタスク ID の数

【エラーコード】

E_RSATR	予約属性（属する保護ドメインまたはクラスが不正）

【機能】

notsk で指定した数のタスク ID を, タスクを生成するサービスコールによって割付け可能なタスク ID として確保する.

notsk は整数定数式パラメータである.

SAC_TSK　タスクのアクセス許可ベクタの設定〔SP〕
sac_tsk　タスクのアクセス許可ベクタの設定〔TPD〕

【静的 API】

```
SAC_TSK(ID tskid, {ACPTN acptn1,
                   ACPTN acptn2, ACPTN acptn3, ACPTN acptn4})
```

【C 言語 API】

```
ER ercd = sac_tsk(ID tskid, const ACVCT *p_acvct)
```

【パラメータ】

ID	tskid	対象タスクの ID 番号
ACVCT *	p_acvct	アクセス許可ベクタを入れたパケットへのポインタ（静的 API を除く）

＊アクセス許可ベクタ（パケットの内容）

ACPTN	acptn1	通常操作 1 のアクセス許可パターン
ACPTN	acptn2	通常操作 2 のアクセス許可パターン
ACPTN	acptn3	管理操作のアクセス許可パターン
ACPTN	acptn4	参照操作のアクセス許可パターン

【リターンパラメータ】

ER	tskid	正常終了（E_OK）またはエラーコード

【エラーコード】

E_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（tskid が不正）
E_RSATR	予約属性（属する保護ドメインかクラスが不正：SAC_TSK の場合）
E_NOEXS〔D〕	オブジェクト未登録（対象タスクが未登録）
E_OACV〔sP〕	オブジェクトアクセス違反（対象タスクに対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（p_acvct が指すメモリ領域への読出しアクセスが許可されていない）
E_OBJ	オブジェクト状態エラー（対象タスクは静的 API で生成された：sac_tsk の場合，対象タスクに対してアクセス許可ベクタが設定済み：SAC_TSK の場合）

【機能】

tskid で指定したタスク（対象タスク）のアクセス許可ベクタ（4 つのアクセス許可パターンの組）を，各パラメータで指定した値に設定する．

静的 API においては，tskid はオブジェクト識別名，acptn1～acptn4 は整数定数式パラメータである．

SAC_TSK は，対象タスクが属する保護ドメインの囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．

sac_tsk において tskid に TSK_SELF（＝0）を指定すると，自タスクが対象タスクとなる．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，SAC_TSK，sac_tsk をサポートしない．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，SAC_TSK，sac_tsk をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，SAC_TSK のみをサポートする．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，SAC_TSK，sac_tsk をサポートしない．

DEF_EPR　　タスクの実行時優先度の定義〔S〕

【静的 API】

```
DEF_EPR(ID tskid, { PRI exepri })
```

【パラメータ】

ID	tskid	対象タスクの ID 番号
PRI	exepri	タスクの実行時優先度

【エラーコード】

E_ID	不正 ID 番号（tskid が不正）
E_PAR	パラメータエラー（exepri が不正）
E_ILUSE	サービスコール不正使用（exepri が，自タスクの起動時優先度よりも低い場合）
E_OBJ	オブジェクト状態エラー（対象タスクに対して実行優先度が設定済み）

【サポートするカーネル】

DEF_EPR は，TOPPERS/SSP カーネルのみがサポートする静的 API である．他のカーネルは，DEF_EPR をサポートしない．

【機能】

tskid で指定したタスク（対象タスク）の実行時優先度を，exepri で指定した優先度に設定する．

tskid はオブジェクト識別名，exepri は整数定数式パラメータである．

exepri は，TMIN_TPRI 以上，TMAX_TPRI 以下でなければならない．また，exepri は，対象タスクの起動時優先度と同じかそれよりも高くなければならない．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様に定義されていない静的 API である．

del_tsk　　タスクの削除〔TD〕

【C 言語 API】

```
ER ercd = del_tsk(ID tskid)
```

【パラメータ】

ID	tskid	対象タスクの ID 番号

【リターンパラメータ】

ER_ID	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（tskid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象タスクが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象タスクに対する管理操作が許可されていない）
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，対象タスクは静的 API で生成された）

【機能】

指定したタスク（対象タスク）を削除する．具体的な振舞いは以下の通り．

対象タスクが休止状態である場合には，対象タスクの登録が解除され，そのタスク ID が未使用の状態に戻される．また，タスクの生成時にタスクのスタック領域およびシステムスタック領域がカーネルによって確保された場合は，それらのメモリ領域が解放される．

対象タスクが休止状態でない場合には，E_OBJ エラーとなる．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，del_tsk をサポートしない．ただし，動的生成機能拡張パッケージでは，del_tsk をサポートする．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，del_tsk をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，del_tsk をサポートしない．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，del_tsk をサポートしない．

act_tsk　　タスクの起動〔T〕
iact_tsk　　タスクの起動〔I〕

【C 言語 API】

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

【パラメータ】

ID	tskid	対象タスクの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し：act_tsk の場合，タスクコンテキストからの呼出し：iact_tsk の場合，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（tskid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象タスクが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象タスクに対する通常操作 1 が許可されていない：act_tsk の場合）
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が TMAX_ACTCNT に一致）

【機能】

tskid で指定したタスク（対象タスク）に対して起動要求を行う．具体的な振舞いは以下の通り．

対象タスクが休止状態である場合には，対象タスクに対してタスク起動時に行うべき初期化処理が行われ，対象タスクは実行できる状態になる．

対象タスクが休止状態でない場合には，対象タスクの起動要求キューイング数に 1 が加えられる．起動要求キューイング数に 1 を加えると TMAX_ACTCNT を超える場合には，E_QOVR エラーとなる．

act_tsk において tskid に TSK_SELF（＝0）を指定すると，自タスクが対象タスクとなる．

【補足説明】

マルチプロセッサ対応カーネルでは，act_tsk／iact_tsk は，対象タスクの次回起動時の割付けプロセッサを変更しない.

---

mact_tsk　割付けプロセッサ指定でのタスクの起動〔TM〕
imact_tsk　割付けプロセッサ指定でのタスクの起動〔IM〕

【C 言語 API】

```
ER ercd = mact_tsk(ID tskid, ID prcid)
ER ercd = imact_tsk(ID tskid, ID prcid)
```

【パラメータ】

ID	tskid	対象タスクの ID 番号
ID	prcid	タスクの割付け対象のプロセッサの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼し：mact_tsk の場合，タスクコンテキストからの呼出し：imact_tsk の場合，CPU ロック状態からの呼出し）
E_NOSPT	未サポート機能（対象タスクが制約タスク）
E_ID	不正 ID 番号（tskid，prcid が不正）
E_PAR	パラメータエラー（対象タスクは prcid で指定したプロセッサに割り付けられない）
E_NOEXS〔D〕	オブジェクト未登録（対象タスクが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象タスクに対する通常操作 1 が許可されていない：mact_tsk の場合）
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が TMAX_ACTCNT に一致）

【機能】

prcid で指定したプロセッサを割付けプロセッサとして，tskid で指定したタスク（対象タスク）に対して起動要求を行う．具体的な振舞いは以下の通り.

対象タスクが休止状態である場合には, 対象タスクの割付けプロセッサが prcid で指定したプロセッサ

に変更された後，対象タスクに対してタスク起動時に行うべき初期化処理が行われ，対象タスクは実行できる状態になる．

対象タスクが休止状態でない場合には，対象タスクの起動要求キューイング数に 1 が加えられ，次回起動時の割付けプロセッサが prcid で指定したプロセッサに変更される．起動要求キューイング数に 1 を加えると TMAX_ACTCNT を超える場合には，E_QOVR エラーとなる．

mact_tsk において tskid に TSK_SELF（＝0）を指定すると，自タスクが対象タスクとなる．

対象タスクの属するクラスの割付け可能プロセッサが，prcid で指定したプロセッサを含んでいない場合には，E_PAR エラーとなる．

prcid に TPRC_INI（＝0）を指定すると，対象タスクの割付けプロセッサを，それが属するクラスの初期割付けプロセッサとする．

【補足説明】
TMAX_ACTCNT が 2 以上の場合でも，対象タスクが次に起動される時の割付けプロセッサは，キューイングされない．すなわち，プロセッサ A に割り付けられた休止状態でないタスクを対象として，プロセッサ B を割付けプロセッサとして mact_tsk を呼び出し，さらにプロセッサ C を割付けプロセッサとして mact_tsk を呼び出すと，対象タスクの次回起動時の割付けプロセッサがプロセッサ C に変更され，対象タスクがプロセッサ B で実行されることはない．なお，TMAX_ACTCNT が 1 の場合には，プロセッサ C を割付けプロセッサとした 2 回目の mact_tsk が E_QOVR エラーとなるため，次回起動時の割付けプロセッサはプロセッサ B のまま変更されない．

【TOPPERS/ASP カーネルにおける規定】
ASP カーネルでは，mact_tsk，imact_tsk をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，mact_tsk，imact_tsk をサポートしない．

【TOPPERS/SSP カーネルにおける規定】
SSP カーネルでは，mact_tsk，imact_tsk をサポートしない．

【μITRON4.0 仕様との関係】
μITRON4.0 仕様に定義されていない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〔T〕

【C 言語 API】

ER_UINT　　actcnt = can_act(ID tskid)

【パラメータ】

ID	tskid	対象タスクの 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数（正の値または 0）またはエラーコード

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（tskid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象タスクが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象タスクに対する通常操作 1 が許可されていない）

【機能】

tskid で指定したタスク（対象タスク）に対する処理されていない起動要求をすべてキャンセルし，キャンセルした起動要求の数を返す．具体的な振舞いは以下の通り．

対象タスクの起動要求キューイング数が 0 に設定され，0 に設定する前の起動要求キューイング数が，サービスコールの返値として返される．また，マルチプロセッサ対応カーネルにおいては，対象タスクの次回起動時の割付けプロセッサが未設定状態に戻される．

tskid に TSK_SELF（＝0）を指定すると，自タスクが対象タスクとなる．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，can_act をサポートしない．

mig_tsk　　タスクの割付けプロセッサの変更〔TM〕

【C 言語 API】

```
ER ercd = mig_tsk(ID tskid, ID prcid)
```

【パラメータ】

ID	tskid	対象タスクの ID 番号
ID	prcid	タスクの割付け対象のプロセッサの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し，対象タスクが自タスクでディスパッチ保留状態からの呼出し）
E_NOSPT	未サポート機能（対象タスクが制約タスク）
E_ID	不正 ID 番号（tskid，prcid が不正）
E_PAR	パラメータエラー（対象タスクは prcid で指定したプロセッサに割り付けられない）
E_NOEXS〔D〕	オブジェクト未登録（対象タスクが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象タスクに対する通常操作 1 が許可されていない）
E_OBJ	オブジェクト状態エラー（対象タスクが自タスクと異なるプロセッサに割り付けられている）

【機能】

tskid で指定したタスクの割付けプロセッサを，prcid で指定したプロセッサに変更する．具体的な振舞いは以下の通り．

対象タスクが，自タスクが割り付けられたプロセッサに割り付けられている場合には，対象タスクを prcid で指定したプロセッサに割り付ける．対象タスクが実行できる状態の場合には，prcid で指定したプロセッサに割り付けられた同じ優先度のタスクの中で，最も優先順位が低い状態となる．

対象タスクが，自タスクが割付けられたプロセッサと異なるプロセッサに割り付けられている場合に

は，E_OBJ エラーとなる．

tskid に TSK_SELF（＝0）を指定すると，自タスクが対象タスクとなる．

ディスパッチ保留状態で，対象タスクを自タスクとして mig_tsk を呼び出すと，E_CTX エラーとなる．

prcid に TPRC_INI（＝0）を指定すると，対象タスクの割付けプロセッサを，それが属するクラスの初期割付けプロセッサに変更する．

【補足説明】
この仕様では，タスクをマイグレーションさせることができるのは，そのタスクと同じプロセッサに割り付けられたタスクのみである．そのため，CPU ロック状態やディスパッチ禁止状態を用いて，他のタスクへのディスパッチが起こらないようにすることで，自タスクが他のプロセッサへマイグレーションされるのを防ぐことができる．

対象タスクが，最初から prcid で指定したプロセッサに割り付けられている場合には，割付けプロセッサの変更は起こらないが，優先順位が同一優先度のタスクの中で最低となる．

【TOPPERS/ASP カーネルにおける規定】
ASP カーネルでは，mig_tsk をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，mig_tsk をサポートしない．

【TOPPERS/SSP カーネルにおける規定】
SSP カーネルでは，mig_tsk をサポートしない．

【μITRON4.0 仕様との関係】
μITRON4.0 仕様に定義されていないサービスコールである．

ext_tsk 自タスクの終了〔T〕

【C 言語 API】

```
ER ercd = ext_tsk()
```

【パラメータ】
なし

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_SYS	システムエラー（カーネルの誤動作）
E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し）

【機能】

自タスクを終了させる．具体的な振舞いは以下の通り．

自タスクに対してタスク終了時に行うべき処理が行われ，自タスクは休止状態になる．さらに，自タスクの起動要求キューイング数が 0 でない場合には，自タスクに対してタスク起動時に行うべき処理が行われ，自タスクは実行できる状態になる．またこの時，起動要求キューイング数から 1 が減ぜられる．

ext_tsk は，CPU ロック解除状態，割込み優先度マスク全解除状態，ディスパッチ許可状態で呼び出すのが原則であるが，そうでない状態で呼び出された場合には，CPU ロック解除状態，割込み優先度マスク全解除状態，ディスパッチ許可状態に遷移させた後，自タスクを終了させる．

ext_tsk が正常に処理された場合，ext_tsk からはリターンしない．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，ext_tsk をサポートしない．自タスクを終了させる場合には，タスクのメインルーチンからリターンする．

【μITRON4.0 仕様との関係】

ext_tsk を非タスクコンテキストから呼び出した場合に，E_CTX エラーが返ることとした．μITRON4.0 仕様においては，ext_tsk からはリターンしないと規定されている．

ter_tsk　　タスクの強制終了〔T〕

【C 言語 API】

```
ER ercd = ter_tsk(ID tskid)
```

【パラメータ】

ID	tskid	対象タスクの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（tskid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象タスクが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象タスクに対する通常操作 2 が許可されていない）
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，対象タスクが自タスクと異なるプロセッサに割り付けられている）

【機能】

tskid で指定したタスク（対象タスク）を終了させる．具体的な振舞いは以下の通り．

対象タスクが休止状態でない場合には，対象タスクに対してタスク終了時に行うべき処理が行われ，対象タスクは休止状態になる．さらに，対象タスクの起動要求キューイング数が 0 でない場合には，対象タスクに対してタスク起動時に行うべき処理が行われ，対象タスクは実行できる状態になる．またこの時，起動要求キューイング数から 1 が減ぜられる．

対象タスクが休止状態である場合には，E_OBJ エラーとなる．また，対象タスクが自タスクの場合には，E_ILUSE エラーとなる．

マルチプロセッサ対応カーネルでは，対象タスクは，自タスクと同じプロセッサに割り付けられているタスクに限られる. 対象タスクが自タスクと異なるプロセッサに割り付けられている場合には, E_OBJ エラーとなる．

【TOPPERS/FMP カーネルにおける使用上の注意】

現時点の FMP カーネルの実装では，デッドロック回避のためのリトライ処理により，サービスコールの処理時間に上限がないため，注意が必要である（ロック方式にも依存する）．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，ter_tsk をサポートしない．

chg_pri　　タスクのベース優先度の変更〔T〕

【C 言語 API】

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

【パラメータ】

ID	tskid	対象タスクの ID 番号
PRI	tskpri	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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_NOSPT	未サポート機能（対象タスクが制約タスク）
E_ID	不正 ID 番号（tskid が不正）
E_PAR	パラメータエラー（tskpri が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象タスクが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象タスクに対する通常操作 2 が許可されていない）
E_ILUSE	サービスコール不正使用（tskpri が，対象タスクがロックしているかロックを待っている優先度上限ミューテックスの上限優先度よりも高い場合）

【機能】

tskid で指定したタスク（対象タスク）のベース優先度を，tskpri で指定した優先度に変更する．具体的な振舞いは以下の通り．

対象タスクが休止状態でない場合には，対象タスクのベース優先度が，tskpri で指定した優先度に変更される．それに伴って，対象タスクの現在優先度も変更される．

対象タスクが，優先度上限ミューテックスをロックしていない場合には，次の処理が行われる．対象タスクが実行できる状態の場合には，同じ優先度のタスクの中で最低優先順位となる．対象タスクが待ち状態で，タスクの優先度順の待ち行列につながれている場合には，対象タスクの変更後の現在優先度に従って，その待ち行列中での順序が変更される．待ち行列中に同じ現在優先度のタスクがある場合には，対象タスクの順序はそれらの中で最後になる．

対象タスクが，優先度上限ミューテックスをロックしている場合には，対象タスクの現在優先度が変更されることはなく，優先順位も変更されない．

対象タスクが休止状態である場合には，E_OBJ エラーとなる．

tskid に TSK_SELF（＝0）を指定すると，自タスクが対象タスクとなる．また，tskpri に TPRI_INI（＝0）を指定すると，対象タスクのベース優先度が，起動時優先度に変更される．

tskpri は，TPRI_INI であるか，TMIN_TPRI 以上，TMAX_TPRI 以下でなければならない．また，対象タスクが優先度上限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，tskpri は，それらのミューテックスの上限優先度と同じかそれより低くなければならない．

【TOPPERS/SSP カーネルにおける規定】
SSP カーネルでは，chg_pri をサポートしない．

【μITRON4.0 仕様との関係】
対象タスクが，同じ優先度のタスクの中で最低の優先順位となる（対象タスクが待ち状態で，タスクの優先度順の待ち行列につながれている場合には，同じ優先度のタスクの中での順序が最後になる）条件を変更した．

---

get_pri　　タスク優先度の参照〔T〕

【C 言語 API】

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

【パラメータ】

ID	tskid	対象タスクの ID 番号
PRI	p_tskpri	現在優先度を入れるメモリ領域へのポインタ

【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
PRI	tskpri	現在優先度

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（tskid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象タスクが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象タスクに対する参照操作が許可されていない）
E_MACV〔P〕	メモリアクセス違反（p_tskpri が指すメモリ領域への書込みアクセスが許可されていない）
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 で指定したタスク（対象タスク）の現在優先度を参照する．具体的な振舞いは以下の通り．

対象タスクが休止状態でない場合には，対象タスクの現在優先度が，p_tskpri で指定したメモリ領域に返される．対象タスクが休止状態である場合には，E_OBJ エラーとなる．

tskid に TSK_SELF（＝0）を指定すると，自タスクが対象タスクとなる．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，get_pri をサポートしない．

get_inf　　自タスクの拡張情報の参照〔T〕

【C 言語 API】

```
ER ercd = get_inf(intptr_t *p_exinf)
```

【パラメータ】

intptr_t *	p_exinf	拡張情報を入れるメモリ領域へのポインタ

【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
intptr_t	exinf	拡張情報

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_MACV〔P〕	メモリアクセス違反（p_exinf が指すメモリ領域への書込みアクセスが許可されていない）

【機能】

自タスクの拡張情報を参照する．参照した拡張情報は，p_exinf で指定したメモリ領域に返される．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，get_inf をサポートしない．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様に定義されていないサービスコールである．

---

ref_tsk　　タスクの状態参照〔T〕

【C 言語 API】

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

ER ercd = chg_pri(ID tskid, PRI tskpri)

【パラメータ】

ID	tskid	対象タスクの ID 番号
T_RTSK *	pk_rtsk	タスクの現在状態を入れる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

＊タスクの現在状態（パケットの内容）

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	タスクの待ち要因
ID	wobjid	タスクの待ち対象のオブジェクトの ID
TMO	lefttmo	タスクがタイムアウトするまでの時間
uint_t	actcnt	タスクの起動要求キューイング数
uint_t	wupcnt	タスクの起床要求キューイング数
bool_t	texmsk	タスクがタスク例外処理マスク状態か否か（保護機能対応カーネルの場合）
bool_t	waifbd	タスクが待ち禁止状態か否か（保護機能対応カーネルの場合）
uint_t	svclevel	タスクの拡張サービスコールのネストレベル（保護機能対応カーネルの場合）
ID	prcid	タスクの割付けプロセッサの ID（マルチプロセッサ対応カーネルの場合）
ID	actprc	タスクの次回起動時の割付けプロセッサの ID（マルチプロセッサ対応カーネルの場合）

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（tskid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象タスクが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象タスクに対する参照操作が許可されていない）
E_MACV〔P〕	メモリアクセス違反（pk_rtsk が指すメモリ領域への書込

【機能】

tskid で指定したタスク（対象タスク）の現在状態を参照する．参照した現在状態は，pk_rtsk で指定したメモリ領域に返される．

tskstat には，対象タスクの現在のタスク状態を表す次のいずれかの値が返される．

TTS_RUN	0x01U	実行状態
TTS_RDY	0x02U	実行可能状態
TTS_WAI	0x04U	待ち状態
TTS_SUS	0x08U	強制待ち状態
TTS_WAS	0x0cU	二重待ち状態
TTS_DMT	0x10U	休止状態
TTS_RUN	0x01U	実行状態

マルチプロセッサ対応カーネルでは，対象タスクが自タスクの場合にも，tskstat が TTS_SUS となる場合がある．この状況は，自タスクに対して ref_tsk を発行するのと同じタイミングで，他のプロセッサで実行されているタスクから同じタスクに対して sus_tsk が発行された場合に発生する可能性がある．

対象タスクが休止状態でない場合には，tskpri には対象タスクの現在優先度が，tskbpri には対象タスクのベース優先度が返される．対象タスクが休止状態である場合には，tskpri と tskbpri の値は保証されない．

対象タスクが待ち状態である場合には，tskwait には，対象タスクが何を待っている状態であるかを表す次のいずれかの値が返される．

TTW_SLP	0x0001U	起床待ち
TTW_DLY	0x0002U	時間経過待ち
TTW_SEM	0x0004U	セマフォの資源獲得待ち
TTW_FLG	0x0008U	イベントフラグ待ち
TTW_SDTQ	0x0010U	データキューへの送信待ち
TTW_RDTQ	0x0020U	データキューからの受信待ち
TTW_SPDQ	0x0100U	優先度データキューへの送信待ち
TTW_RPDQ	0x0200U	優先度データキューからの受信待ち
TTW_MBX	0x0040U	メールボックスからの受信待ち
TTW_MTX	0x0080U	ミューテックスのロック待ち状態
TTW_MPF	0x2000U	固定長メモリブロックの獲得待ち

対象タスクが待ち状態でない場合には，tskwait の値は保証されない．

対象タスクが起床待ち状態および時間経過待ち状態以外の待ち状態である場合には，wobjid に，対象タスクが待っているオブジェクトの ID 番号が返される．対象タスクが待ち状態でない場合や，起床待ち状態または時間経過待ち状態である場合には，wobjid の値は保証されない．

対象タスクが時間経過待ち状態以外の待ち状態である場合には，lefttmo に，タスクがタイムアウトを

起こすまでの相対時間が返される．タスクがタイムアウトを起こさない場合には，TMO_FEVR（＝-1）が返される．

対象タスクが時間経過待ち状態である場合には，lefttmo に，タスクの遅延時間が経過して待ち解除されるまでの相対時間が返される．ただし，返されるべき相対時間が TMO 型に格納することができない場合がありうる．この場合には，相対時間（RELTIM 型，uint_t 型に定義される）を TMO 型（int_t 型に定義される）に型キャストした値が返される．

対象タスクが待ち状態でない場合には，lefttmo の値は保証されない．

actcnt には，対象タスクの起動要求キューイング数が返される．

対象タスクが休止状態でない場合には，wupcnt に，タスクの起床要求キューイング数が返される．対象タスクが休止状態である場合には，wupcnt の値は保証されない．

保護機能対応カーネルで，対象タスクが休止状態でない場合には，texmsk に，対象タスクがタスク例外処理マスク状態の場合に true，そうでない場合に false が返される．waifbd には，対象タスクが待ち禁止状態の場合に true，そうでない場合に false が返される．また svclevel には，対象タスクが拡張サービスコールを呼び出していない場合には 0，呼び出している場合には，実行中の拡張サービスコールがネスト段数が返される．対象タスクが休止状態である場合には，texmsk，waifbd，svclevel の値は保証されない．

マルチプロセッサ対応カーネルでは，prcid に，対象タスクの割付けプロセッサの ID 番号が返される．また actprc には，対象タスクの次回起動時の割付けプロセッサの ID 番号が返される．次回起動時の割付けプロセッサが未設定の場合には，actprc に TPRC_NONE（＝0）が返される．

tskid に TSK_SELF（＝0）を指定すると，自タスクが対象タスクとなる．

【補足説明】
対象タスクが時間経過待ち状態である場合に，lefttmo（TMO 型）に返される値を RELTIM 型に型キャストすることで，タスクが待ち解除されるまでの相対時間を正しく得ることができる．

【TOPPERS/ASP カーネルにおける規定】
ASP カーネルでは，tskwait に TTW_MTX が返ることはない．ただし，ミューテックス機能拡張パッケージを用いると，tskwait に TTW_MTX が返る場合がある．

【TOPPERS/FMP カーネルにおける規定】
FMP カーネルでは，tskwait に TTW_MTX が返ることはない．

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，tskwait に TTW_MBX が返ることはない．

【TOPPERS/SSP カーネルにおける規定】
SSP カーネルでは，ref_tsk をサポートしない．

【使用上の注意】
ref_tsk はデバッグ時向けの機能であり，その他の目的に使用することは推奨しない．これは，ref_tsk を呼び出し，対象タスクの現在状態を参照した直後に割込みが発生した場合，ref_tsk から戻ってきた時には対象タスクの状態が変化している可能性があるためである．

【μITRON4.0 仕様との関係】
対象タスクが時間経過待ち状態の時に lefttmo に返される値について規定した．また，参照できるタスクの状態から，強制待ち要求ネスト数（suscnt）を除外した．

マルチプロセッサ対応カーネルで参照できる情報として，割付けプロセッサの ID（prcid）と次回起動時の割付けプロセッサの ID（actprc）を追加した．

【μITRON4.0/PX 仕様との関係】
保護機能対応カーネルで参照できる情報として，タスク例外処理マスク状態か否か（texmsk），待ち禁止状態か否か（waifbd），拡張サービスコールのネストレベル（svclevel）を追加した．

4.2. タスク付属同期機能

タスク付属同期機能は，タスクとタスクの間，または非タスクコンテキストの処理とタスクの間で同期を取るために，タスク単独で持っている機能である．

タスク付属同期機能に関連して，各タスクが持つ情報は次の通り．

- 起床要求キューイング数

タスクの起床要求キューイング数は，処理されていないタスクの起床要求の数であり，タスクの起動時に 0 に初期化される．

タスク付属同期機能に関連するカーネル構成マクロは次の通り．

TMAX_WUPCNT	タスクの起床要求キューイング数の最大値

【TOPPERS/ASP カーネルにおける規定】
ASP カーネルでは，TMAX_WUPCNT は 1 に固定されている．

【TOPPERS/FMP カーネルにおける規定】
FMP カーネルでは，TMAX_WUPCNT は 1 に固定されている．

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，TMAX_WUPCNT は 1 に固定されている．

【TOPPERS/SSP カーネルにおける規定】
SSP カーネルでは，タスク付属同期機能をサポートしない．

【μITRON4.0 仕様との関係】
この仕様では，強制待ち要求をネストする機能をサポートしないこととした．言い換えると，強制待ち要求ネスト数の最大値を 1 に固定する．これに伴い，強制待ち状態から強制再開するサービスコール（frsm_tsk）とタスクの強制待ち要求ネスト数の最大値を表すカーネル構成マクロ（TMAX_SUSCNT）は廃止した．また，ref_tsk で参照できる情報（T_RTSK のフィールド）から，強制待ち要求ネスト数（suscnt）を除外した．

---

slp_tsk　　起床待ち〔T〕
tslp_tsk　　起床待ち（タイムアウト付き）〔T〕

【C 言語 API】

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

【パラメータ】

TMO	tmout	タイムアウト時間（tslp_tsk の場合

【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_CTX	コンテキストエラー（ディスパッチ保留状態からの呼出し）
E_NOSPT	未サポート機能（制約タスクからの呼出し）
E_PAR	パラメータエラー（tmout が不正：tslp_tsk の場合）
E_TMOUT	ポーリング失敗またはタイムアウト（slp_tsk を除く）
E_RLWAI	待ち禁止状態または待ち状態の強制解除

【機能】

自タスクを起床待ちさせる．具体的な振舞いは以下の通り．

自タスクの起床要求キューイング数が 0 でない場合には，起床要求キューイング数から 1 が減ぜられる．起床要求キューイング数が 0 の場合には，自タスクは起床待ち状態となる．

【補足説明】

自タスクの起床要求キューイング数が 0 でない場合には，自タスクは実行できる状態を維持し，自タスクの優先順位は変化しない．

---

wup_tsk　タスクの起床〔T〕
iwup_tsk　タスクの起床〔I〕

【C 言語 API】

ER ercd = wup_tsk(ID tskid)

ER ercd = iwup_tsk(ID tskid)

【パラメータ】

ID	tskid	対象タスクの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し：wup_tsk の場合，タスクコンテキストからの呼出し：iwup_tsk の場合，CPU ロック状態からの呼出し）
E_NOSPT	未サポート機能（対象タスクが制約タスク）
E_ID	不正 ID 番号（tskid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象タスクが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象タスクに対する通常操作 1 が許可されていない：wup_tsk の場合）
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が TMAX_WUPCNT に一致）

【機能】

tskid で指定したタスク（対象タスク）を起床する．具体的な振舞いは以下の通り．

対象タスクが起床待ち状態である場合には，対象タスクが待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_OK が返る．

対象タスクが起床待ち状態でなく，休止状態でもない場合には，対象タスクの起床要求キューイング数に 1 が加えられる．起床要求キューイング数に 1 を加えると TMAX_WUPCNT を超える場合には，E_QOVR エラーとなる．

対象タスクが休止状態である場合には，E_OBJ エラーとなる．

wup_tsk において tskid に TSK_SELF（＝0）を指定すると，自タスクが対象タスクとなる．

can_wup　　タスク起床要求のキャンセル〔T〕

【C 言語 API】

```
ER_UINT wupcnt = can_wup(ID tskid)
```

【パラメータ】

ID	tskid	対象タスクの 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数（正の値または 0）またはエラーコード

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_NOSPT	未サポート機能（対象タスクが制約タスク）
E_ID	不正 ID 番号（tskid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象タスクが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象タスクに対する通常操作 1 が許可されていない）
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 で指定したタスク（対象タスク）に対する処理されていない起床要求をすべてキャンセルし，キャンセルした起床要求の数を返す．具体的な振舞いは以下の通り．

対象タスクが休止状態でない場合には，対象タスクの起床要求キューイング数が 0 に設定され，0 に設定する前の起床要求キューイング数が，サービスコールの返値として返される．

対象タスクが休止状態である場合には，E_OBJ エラーとなる．

tskid に TSK_SELF（＝0）を指定すると，自タスクが対象タスクとなる．

---

rel_wai　強制的な待ち解除〔T〕
irel_wai　強制的な待ち解除〔I〕

【C 言語 API】

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

【パラメータ】

ID	tskid	対象タスクの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し：rel_wai の場合，タスクコンテキストからの呼出し：irel_wai の場合，CPU ロック状態からの呼出し）
E_NOSPT	未サポート機能（対象タスクが制約タスク）
E_ID	不正 ID 番号（tskid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象タスクが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象タスクに対する通常操作 2 が許可されていない：rel_wai の場合）
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 で指定したタスク（対象タスク）を，強制的に待ち解除する．具体的な振舞いは以下の通り．

対象タスクが待ち状態である場合には，対象タスクが待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_RLWAI が返る．

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　　強制待ち状態への遷移〔T〕

【C 言語 API】

```
ER ercd = sus_tsk(ID tskid)
```

【パラメータ】

ID	tskid	対象タスクの ID 番号

【リターンパラメータ】

ER_ID	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し，対象タスクが自タスクでディスパッチ保留状態からの呼出し）
E_NOSPT	未サポート機能（対象タスクが制約タスク）
E_ID	不正 ID 番号（tskid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象タスクが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象タスクに対する通常操作 2 が許可されていない）
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 で指定したタスク（対象タスク）を強制待ちにする．具体的な振舞いは以下の通り．

対象タスクが実行できる状態である場合には，対象タスクは強制待ち状態となる．また，待ち状態（二重待ち状態を除く）である場合には，二重待ち状態となる．

対象タスクが強制待ち状態または二重待ち状態である場合は E_QOVR エラー，休止状態である場合には E_OBJ エラーとなる．

マルチプロセッサ対応カーネルでは，対象タスクが自タスクの場合にも，E_QOVR エラーとなる場合がある．この状況は，自タスクに対して sus_tsk を発行するのと同じタイミングで，他のプロセッサで実行されているタスクから同じタスクに対して sus_tsk が発行された場合に発生する可能性がある．

tskid に TSK_SELF（＝0）を指定すると，自タスクが対象タスクとなる．

ディスパッチ保留状態で，対象タスクを自タスクとして sus_tsk を呼び出すと，E_CTX エラーとなる．

---

rsm_tsk　　強制待ち状態からの再開〔T〕

【C 言語 API】

```
ER ercd = rsm_tsk(ID tskid)
```

【パラメータ】

ID	tskid	対象タスクの ID 番号

【リターンパラメータ】

ER_ID	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_NOSPT	未サポート機能（対象タスクが制約タスク）
E_ID	不正 ID 番号（tskid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象タスクが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象タスクに対する通常操作 2 が許可されていない）

【機能】

tskid で指定したタスク（対象タスク）を，強制待ちから再開する．具体的な振舞いは以下の通り．

対象タスクが強制待ち状態である場合には，対象タスクは強制待ちから再開される．強制待ち状態でない場合には，E_OBJ エラーとなる．

---

dis_wai　待ち禁止状態への遷移〔TP〕
idis_wai　待ち禁止状態への遷移〔IP〕

【C 言語 API】

```
ER ercd = dis_wai(ID tskid)
ER ercd = idis_wai(ID tskid)
```

【パラメータ】

ID	tskid	対象タスクの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し：dis_wai の場合，タスクコンテキストからの呼出し：idis_wai の場合，CPU ロック状態からの呼出し）
E_NOSPT	未サポート機能（対象タスクが制約タスク）
E_ID	不正 ID 番号（tskid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象タスクが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象タスクに対する通常操作 2 が許可されていない：dis_wai の場合）
E_OBJ	オブジェクト状態エラー（対象タスクが休止状態，対象タスクがタスク例外処理マスク状態でない）
E_QOVR	キューイングオーバフロー（対象タスクが待ち禁止状態）

【機能】

tskid で指定したタスク（対象タスク）を待ち禁止状態にする．具体的な振舞いは以下の通り．

対象タスクがタスク例外処理マスク状態であり，待ち禁止状態でない場合には，対象タスクは待ち禁止状態になる．

対象タスクが休止状態である場合には，E_OBJ エラーとなる．また，対象タスクがタスク例外処理マスク状態でない場合には E_OBJ エラー，待ち禁止状態の場合には E_QOVR エラーとなる．

dis_wai において tskid に TSK_SELF（＝0）を指定すると，自タスクが対象タスクとなる．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，dis_wai をサポートしない．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，dis_wai をサポートしない．

【補足説明】

dis_wai は，対象タスクの待ち解除は行わない．対象タスクを待ち禁止状態にすることに加えて待ち解除したい場合には，dis_wai を呼び出した後に，rel_wai を呼び出せばよい．

【未決定事項】

マルチプロセッサ対応カーネルでは，対象タスクを，自タスクと同じプロセッサに割り付けられているタスクに限るなどの制限を導入する可能性があるが，現時点では未決定である．

【μITRON4.0/PX 仕様との関係】

μITRON4.0/PX 仕様に定義されていないサービスコールである．

---

ena_wai	待ち禁止状態の解除〔TP〕
iena_wai	待ち禁止状態の解除〔IP〕

【C 言語 API】

```
ER ercd = ena_wai(ID tskid)
ER ercd = iena_wai(ID tskid)
```

【パラメータ】

ID	tskid	対象タスクの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し：ena_wai の場合，タスクコンテキストからの呼出し：iena_wai の場合，CPU ロック状態からの呼出し）
E_NOSPT	未サポート機能（対象タスクが制約タスク）
E_ID	不正 ID 番号（tskid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象タスクが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象タスクに対する通常操作 2 が許可されていない：ena_wai の場合）
E_OBJ	ブジェクト状態エラー（対象タスクが休止状態，対象タスクが待ち禁止状態でない）

【機能】

tskid で指定したタスク（対象タスク）の待ち禁止状態を解除する．具体的な振舞いは以下の通り．

対象タスクが待ち禁止状態である場合には，待ち禁止状態は解除される．対象タスクが休止状態である場合や，待ち禁止状態でない場合には，E_OBJ エラーとなる．

ena_wai において tskid に TSK_SELF（＝0）を指定すると，自タスクが対象タスクとなる．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，ena_wai をサポートしない．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，ena_wai をサポートしない．

【未決定事項】

マルチプロセッサ対応カーネルでは，対象タスクを，自タスクと同じプロセッサに割り付けられているタスクに限るなどの制限を導入する可能性があるが，現時点では未決定である．

【μITRON4.0/PX 仕様との関係】

μITRON4.0/PX 仕様に定義されていないサービスコールである．

dly_tsk　　自タスクの遅延〔T〕

【C 言語 API】

```
ER ercd = dly_tsk(RELTIM dlytim)
```

【パラメータ】

ID	tskid	対象タスクの 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_CTX	コンテキストエラー（ディスパッチ保留状態からの呼出し）
E_NOSPT	未サポート機能（制約タスクからの呼出し）
E_PAR	パラメータエラー（dlytim が不正）
E_RLWAI	待ち禁止状態または待ち状態の強制解除

【機能】

dlytim で指定した時間，自タスクを遅延させる．具体的な振舞いは以下の通り．

自タスクは，dlytim で指定した時間が経過するまでの間，時間経過待ち状態となる．dly_tsk を呼び出してから dlytim で指定した相対時間後に，自タスクは待ち解除され，dly_tsk から E_OK が返る．

dlytim は，TMAX_RELTIM 以下でなければならない．

4.3. タスク例外処理機能

タスク例外処理ルーチンは，カーネルが実行を制御する処理単位で，タスクと同一のコンテキスト内で実行される．タスク例外処理ルーチンは，各タスクに 1 つのみ登録できるため，タスク ID によって識別する．

タスク例外処理機能に関連して，各タスクが持つ情報は次の通り．

- タスク例外処理ルーチン属性
- タスク例外処理禁止フラグ
- 保留例外要因
- タスク例外処理ルーチンの先頭番地

タスク例外処理ルーチン属性に指定できる属性はない．そのため，タスク例外処理ルーチン属性には，TA_NULL を指定しなければならない．

タスクは，タスク例外処理ルーチンの実行を保留するためのタスク例外処理禁止フラグを持つ．タスク例外処理禁止フラグがセットされた状態をタスク例外処理禁止状態，クリアされた状態をタスク例外処理許可状態と呼ぶ．タスク例外処理禁止フラグは，タスクの起動時に，セットした状態に初期化される．

タスクの保留例外要因は，タスクに対して要求された例外要因を蓄積するためのビットマップであり，タスクの起動時に 0 に初期化される．

タスク例外処理ルーチンは，「タスク例外処理許可状態である」「保留例外要因が 0 でない」「タスクが実行状態である」「タスクコンテキストが実行されている」「割込み優先度マスク全解除状態である」「CPU ロック状態でない」の 6 つの条件が揃った場合に実行が開始される．保護機能対応カーネルにおいては，さらに，「タスク例外処理マスク状態でない」という条件が追加される．タスク例外処理マスク状態については，「2.6.5 タスク例外処理マスク状態と待ち禁止状態」の節を参照すること．

タスク例外処理ルーチンの実行が開始される時，タスク例外処理禁止フラグはセットされ，保留例外要因は 0 にクリアされる．また，タスク例外処理ルーチンからのリターン時には，タスク例外処理禁止フラグはクリアされる．

保護機能対応カーネルでは，ユーザタスクのタスク例外処理ルーチンの実行開始時に，リターン先の番地やシステム状態等が，ユーザスタック上に保存される．ここで，ユーザスタック領域に十分な空きがない場合や，ユーザスタックポインタがユーザスタック領域以外を指している場合，カーネルは，エミュレートされた CPU 例外を発生させる．これを，タスク例外実行開始時スタック不正例外と呼ぶ．

逆に，タスク例外処理ルーチンからのリターン時には，リターン先の番地やシステム状態等が，ユーザスタック上から取り出される．ここで，ユーザスタック領域に積まれている情報が足りない場合や，ユーザスタックポインタがユーザスタック領域以外を指している場合，カーネルは，エミュレートされた CPU 例外を発生させる．これを，タスク例外リターン時スタック不正例外と呼ぶ．

タスク例外実行開始時スタック不正例外またはタスク例外リターン時スタック不正例外を起こしたタスクの実行を継続した場合の動作は保証されないため，アプリケーションは，これらの CPU 例外を処理する CPU 例外ハンドラで，「2.8.1CPU 例外処理の流れ」の節の(b)または(d)の方法でリカバリ処理を行う
必要がある．この方法に従わなかった場合の動作は，保証されない．

保護機能対応カーネルにおいて，タスク例外処理ルーチンは，タスクと同じ保護ドメインに属する．

タスク例外処理機能に用いるデータ型は次の通り．

TEXPTN	タスク例外要因のビットパターン（符号無し整数, uint_t に定義）

C 言語によるタスク例外処理ルーチンの記述形式は次の通り．

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

texptn にはタスク例外処理ルーチン起動時の保留例外要因が，exinf にはタスクの拡張情報が，それぞれ渡される．

タスク例外処理機能に関連するカーネル構成マクロは次の通り．

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

【補足説明】
保護機能対応でないカーネルでは，タスク例外処理ルーチンの実行開始条件の内，「CPU ロック状態でない」は省いても同じ結果になる．これは，CPU ロック状態で他の条件が揃うことはないためである．一方，保護機能対応カーネルでは，CPU ロック状態で拡張サービスコールからリターンした場合（より厳密には，タスク例外処理マスク状態が解除された場合）に，CPU ロック状態で他の条件が揃うことになる．

【TOPPERS/ASP カーネルにおける規定】
ASP カーネルでは，タスク例外要因のビット数（TBIT_TEXPTN）は 16 以上である．

【TOPPERS/FMP カーネルにおける規定】
FMP カーネルでは，タスク例外要因のビット数（TBIT_TEXPTN）は 16 以上である．

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，タスク例外要因のビット数（TBIT_TEXPTN）は 16 以上である．

【TOPPERS/SSP カーネルにおける規定】
SSP カーネルでは，タスク例外処理機能をサポートしない．

【μITRON4.0 仕様との関係】
割込み優先度マスク全解除状態でない場合には，タスク例外処理ルーチンの実行が開始されないという仕様に変更した．

【μITRON4.0/PX 仕様との関係】
ユーザタスクのタスク例外処理ルーチンの実行開始時とリターン時にユーザスタックが不正となる問題に関して，μITRON4.0/PX 仕様では考慮されていない．

【仕様変更の経緯】
この仕様の Release 1.2 以前では，タスク例外処理ルーチンの実行開始条件に「割込み優先度マスク全解除状態である」の条件がなかったが，Release1.3 以降で追加した．これは，マルチプロセッサ対応カーネルにおいて，他プロセッサで実行中のタスクに対してタスク例外処理を要求した場合に，割込み優先度マスクが全解除でないと，タスク例外処理ルーチンをただちに実行開始することができないためである．なお，ASP カーネル Release 1.6 以前と，FMP カーネル Release 1.1.1 以前のバージョンは，古い仕様に従って実装されている．

DEF_TEX　タスク例外処理ルーチンの定義〔S〕
def_tex　タスク例外処理ルーチンの定義〔TD〕

【静的 API】

```
DEF_TEX(ID tskid, {ATR texatr, TEXRTN texrtn })
```

【C 言語 API】

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

【パラメータ】

ID	tskid	対象タスクの ID 番号
T_DTEX	pk_dtex	タスク例外処理ルーチンの定義情報を入れたパケットへのポインタ（静的 API を除く）

＊タスク例外処理ルーチンの定義情報（パケットの内容）

ATR	texatr	タスク例外処理ルーチン属性
TEXRTN	texrtn	タスク例外処理ルーチンの先頭番地

【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_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（tskid が不正）
E_RSATR	予約属性（texatr が不正または使用できない，属する保護ドメインかクラスが不正）
E_NOEXS〔D〕	オブジェクト未登録（対象タスクが未登録）
E_OACV〔sP〕	オブジェクトアクセス違反（対象タスクに対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（pk_dtex が指すメモリ領域への読出しアクセスが許可されていない）
E_PAR	パラメータエラー（texrtn が不正）
E_OBJ	オブジェクト状態エラー（タスク例外処理ルーチンを定義済みのタスクに対する定義，タスク例外処理ルーチンを未定義のタスクに対する解除，対象タスクは静的 API で生成された：def_tex の場合）

【機能】

tskid で指定したタスク（対象タスク）に対して，各パラメータで指定したタスク例外処理ルーチン定義情報に従って，タスク例外処理ルーチンを定義する．

ただし，def_tex において pk_dtex を NULL にした場合には，対象タスクに対するタスク例外処理ルーチンの定義を解除する．また，対象タスクのタスク例外処理禁止フラグをセットし，保留例外要因を 0 に初期化する．

静的 API においては，tskid はオブジェクト識別名，texatr は整数定数式パラメータ，texrtn は一般

定数式パラメータである．

静的 API によって生成したタスクに対しては，タスク例外処理ルーチンの登録は DEF_TEX によって行わねばならず，def_tex によってタスク例外処理ルーチンを登録／登録解除することはできない．def_tex において，対象タスクが静的 API で生成したタスクである場合には，E_OBJ エラーとなる．

タスク例外処理ルーチンを定義する場合（DEF_TEX の場合および def_tex において pk_dtex を NULL 以外にした場合）で，対象タスクに対してすでにタスク例外処理ルーチンが定義されている場合には，E_OBJ エラーとなる．

保護機能対応カーネルにおいて，DEF_TEX は，対象タスクが属する保護ドメインの囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．また，def_tex でタスク例外処理ルーチンを定義する場合には，タスク例外処理ルーチンの属する保護ドメインを設定する必要はなく，タスク例外処理ルーチン属性に TA_DOM(domid)を指定した場合には E_RSATR エラーとなる．ただし，TA_DOM(TDOM_SELF)を指定した場合には，指定が無視され，E_RSATR エラーは検出されない．

タスク例外処理ルーチンの定義を解除する場合（def_tex において pk_dtex を NULL にした場合）で，対象タスクに対して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 エラーとなる．

def_tex において tskid に TSK_SELF（＝0）を指定すると，自タスクが対象タスクとなる．

【TOPPERS/ASP カーネルにおける規定】
ASP カーネルでは，DEF_TEX のみをサポートする．ただし，動的生成機能拡張パッケージでは，def_tex もサポートする．

【TOPPERS/FMP カーネルにおける規定】
FMP カーネルでは，DEF_TEX のみをサポートする．

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，DEF_TEX のみをサポートする．

【μITRON4.0 仕様との関係】
texrtn のデータ型を TEXRTN に変更した．

def_tex によって，定義済みのタスク例外処理ルーチンを再定義しようとした場合に，E_OBJ エラーとすることにした．

ras_tex　　タスク例外処理の要求〔T〕
iras_tex　　タスク例外処理の要求〔I〕

【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	対象タスクの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し：ras_tex の場合，タスクコンテキストからの呼出し：iras_tex の場合，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（tskid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象タスクが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象タスクに対する通常操作 2 が許可されていない：ras_tex の場合）
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，対象タスクに対してタスク例外処理ルーチンが定義されていない）

【機能】

tskid で指定したタスク（対象タスク）に対して，rasptn で指定したタスク例外要因のタスク例外処理を要求する．対象タスクの保留例外要因が，それまでの値と rasptn で指定した値のビット毎論理和（C 言語の"|"）に更新される．

対象タスクが休止状態である場合と，対象タスクに対して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 エラーとなる．

ras_tex において tskid に TSK_SELF（＝0）を指定すると，自タスクが対象タスクとなる．

rasptn が 0 の場合には，E_PAR エラーとなる．

dis_tex　　タスク例外処理の禁止〔T〕

【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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	オブジェクト状態エラー（自タスクに対してタスク例外処理ルーチンが定義されていない）

【機能】

自タスクのタスク例外処理禁止フラグをセットする．すなわち，自タスクをタスク例外処理禁止状態に遷移させる．

ena_tex　　タスク例外処理の許可〔T〕

【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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	オブジェクト状態エラー（自タスクに対してタスク例外処理ルーチンが定義されていない）

【機能】

自タスクのタスク例外処理禁止フラグをクリアする．すなわち，自タスクをタスク例外処理許可状態に遷移させる．

【補足説明】

タスク例外処理ルーチン中で ena_tex を呼び出すことにより，タスク例外処理ルーチンの多重起動を行うことができる．ただし，多重起動の最大段数を制限するのは，アプリケーションの責任である．

---

sns_tex　　タスク例外処理禁止状態の参照〔TI〕

【C 言語 API】

```
bool_t state = sns_tex()
```

【パラメータ】

なし

【リターンパラメータ】

bool_t	state	タスク例外処理禁止状態

【機能】

実行状態のタスクのタスク例外処理禁止フラグを参照する．具体的な振舞いは以下の通り．

実行状態のタスクが，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 が返る. sns_tex を非タスクコンテキストから呼び出した場合で, 実行状態のタスクがない場合には, true が返る.

マルチプロセッサ対応カーネルにおいては，サービスコールを呼び出した処理単位を実行しているプロセッサにおいて実行状態のタスクのタスク例外処理禁止フラグを参照する.

【補足説明】

sns_tex をタスクコンテキストから呼び出した場合，実行状態のタスクは自タスクに一致する.

ref_tex　　タスク例外処理の状態参照〔T〕

【C 言語 API】

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

【パラメータ】

ID	tskid	対象タスクの ID 番号
T_RTEX	pk_rtex	タスク例外処理の現在状態を入れる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

＊タスク例外処理の現在状態（パケットの内容）

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

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（tskid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象タスクが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象タスクに対する参照操作が許可されていない）
E_MACV〔P〕	メモリアクセス違反（pk_rtex が指すメモリ領域への書込みアクセスが許可されていない）
E_OBJ	オブジェクト状態エラー（対象タスクが休止状態，対象タスクに対してタスク例外処理ルーチンが定義されていない）

【機能】

tskid で指定したタスク（対象タスク）のタスク例外処理に関する現在状態を参照する．参照した現在状態は，pk_rtex で指定したパケットに返される．

texstat には，対象タスクの現在のタスク例外処理禁止フラグを表す次のいずれかの値が返される．

TTEX_ENA	0x01U	タスク例外処理許可状態
TTEX_DIS	0x02U	タスク例外処理禁止状態

pndptn には，対象タスクの現在の保留例外要因が返される．

tskid に TSK_SELF（＝0）を指定すると，自タスクが対象タスクとなる．

4.4. 同期・通信機能

【TOPPERS/SSP カーネルにおける規定】
SSP カーネルでは，同期・通信機能をサポートしない．.

【μITRON4.0 仕様との関係】
この仕様では，ランデブ機能はサポートしていない．今後の検討により，ランデブ機能をサポートすることに変更する可能性もある．

4.4.1. セマフォ

セマフォは，資源の数を表す 0 以上の整数値を取るカウンタ（資源数）を介して，排他制御やイベント通知を行うための同期・通信オブジェクトである．セマフォの資源数から 1 を減ずることを資源の獲得，資源数に 1 を加えることを資源の返却と呼ぶ．セマフォは，セマフォ ID と呼ぶ ID 番号によって識別する．

各セマフォが持つ情報は次の通り．

- セマフォ属性
- 資源数（の現在値）
- 待ち行列（セマフォの資源獲得待ち状態のタスクのキュー）
- 初期資源数
- 最大資源数
- アクセス許可ベクタ（保護機能対応カーネルの場合）
- 属する保護ドメイン（保護機能対応カーネルの場合）
- 属するクラス（マルチプロセッサ対応カーネルの場合）

待ち行列は，セマフォの資源が獲得できるまで待っている状態（セマフォの資源獲得待ち状態）のタスクが，資源を獲得できる順序でつながれているキューである．

セマフォの初期資源数は，セマフォを生成または再初期化した際の，資源数の初期値である．また，セマフォの最大資源数は，資源数が取りうる最大値である．資源数が最大資源数に一致している時に資源を返却しようとすると，E_QOVR エラーとなる．

セマフォ属性には，次の属性を指定することができる．

TA_TPRI	0x01U	待ち行列をタスクの優先度順にする

TA_TPRI を指定しない場合，待ち行列は FIFO 順になる．

セマフォ機能に関連するカーネル構成マクロは次の通り．

TMAX_ACTCNT	TMAX_MAXSEM
TNUM_SEMID	登録できるセマフォの数（動的生成対応でないカーネルでは，静的 API によって登録されたセマフォの数に一致）

【μITRON4.0 仕様との関係】
TNUM_SEMID は，μITRON4.0 仕様に規定されていないカーネル構成マクロである．

CRE_SEM　セマフォの生成〔S〕
acre_sem　セマフォの生成〔TD〕

【静的 API】

```
CRE_SEM(ID semid, {ATR sematr, uint_t isemcnt, uint_t maxsem })
```

【C 言語 API】

```
ER_ID semid = acre_sem(const T_CSEM *pk_csem)
```

【パラメータ】

ID	semid	生成するセマフォの ID 番号（CRE_SEM の場合）
T_CSEM *	pk_csem	セマフォの生成情報を入れたパケットへのポインタ（静的 API を除く）

＊セマフォの生成情報（パケットの内容）

ATR	sematr	セマフォ属性
uint_t	isemcnt	セマフォの初期資源数
uint_t	maxsem	セマフォの最大資源数

【リターンパラメータ】

ER_ID	semid	生成されたセマフォの ID 番号（正の値）またはエラーコード

【エラーコード】

E_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_RSATR	予約属性（sematr が不正または使用できない，属する保護ドメインかクラスが不正）
E_PAR	パラメータエラー（isemcnt，maxsem が不正）
E_OACV〔sP〕	オブジェクトアクセス違反（システム状態に対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（pk_csem が指すメモリ領域への読出しアクセスが許可されていない）
E_NOID〔sD〕	ID 番号不足（割り付けられるセマフォ ID がない）
E_OBJ	オブジェクト状態エラー（semid で指定したセマフォが登録済み：CRE_SEM の場合）

【機能】

各パラメータで指定したセマフォ生成情報に従って，セマフォを生成する．生成されたセマフォの資源数は初期資源数に，待ち行列は空の状態に初期化される．

静的 API においては，semid はオブジェクト識別名，isemcnt と maxsem は整数定数式パラメータである．

isemcnt は，0 以上で，maxsem 以下でなければならない．また，maxsem は，1 以上で，TMAX_MAXSEM 以下でなければならない．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，CRE_SEM のみをサポートする．ただし，動的生成機能拡張パッケージでは，acre_sem もサポートする．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，CRE_SEM のみをサポートする．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，CRE_SEM のみをサポートする．

AID_SEM　　割付け可能なセマフォ ID の数の指定〔SD〕

【静的 API】

```
AID_SEM(uint_t nosem)
```

【パラメータ】

uint_t	nosem	割付け可能なセマフォ ID の数

【エラーコード】

E_RSATR	予約属性（属する保護ドメインまたはクラスが不正）

【機能】

nosem で指定した数のセマフォ ID を, セマフォを生成するサービスコールによって割付け可能なセマフォ ID として確保する.

nosem は整数定数式パラメータである.

SAC_SEM　　セマフォのアクセス許可ベクタの設定〔SP〕
sac_sem　　セマフォのアクセス許可ベクタの設定〔TPD〕

【静的 API】

```
SAC_SEM(ID semid, {ACPTN acptn1, ACPTN acptn2
                   ACPTN acptn3, ACPTN acptn4})
```

【C 言語 API】

```
ER ercd = sac_sem(ID semid, const ACVCT *p_acvct)
```

【パラメータ】

ID	semid	対象セマフォの ID 番号
ACVCT *	p_acvct	アクセス許可ベクタを入れたパケットへのポインタ（静的 API を除く）

＊アクセス許可ベクタ（パケットの内容）

ACPTN	acptn1	通常操作 1 のアクセス許可パターン
ACPTN	acptn2	通常操作 2 のアクセス許可パターン
ACPTN	acptn3	管理操作のアクセス許可パターン
ACPTN	acptn4	参照操作のアクセス許可パターン

【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_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（semid が不正）
E_RSATR	予約属性（属する保護ドメインかクラスが不正：SAC_SEM の場合）
E_NOEXS〔D〕	オブジェクト未登録（対象セマフォが未登録）
E_OACV〔sP〕	オブジェクトアクセス違反（対象セマフォに対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（p_acvct が指すメモリ領域への読出しアクセスが許可されていない）
E_OBJ	オブジェクト状態エラー（対象セマフォは静的 API で生成された：sac_sem の場合，対象セマフォに対してアクセス許可ベクタが設定済み：SAC_SEM の場合）

【機能】

semid で指定したセマフォ（対象セマフォ）のアクセス許可ベクタ（4 つのアクセス許可パターンの組）を，各パラメータで指定した値に設定する．

静的 API においては，semid はオブジェクト識別名，acptn1～acptn4 は整数定数式パラメータである．

SAC_SEM は，対象セマフォが属する保護ドメインの囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，SAC_SEM，sac_sem をサポートしない．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，SAC_SEM，sac_sem をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，SAC_SEM のみをサポートする．

del_sem　　セマフォの削除〔TD〕

【C 言語 API】

```
ER ercd = del_sem(ID semid)
```

【パラメータ】

ID	semid	対象セマフォの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（semid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象セマフォが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象セマフォに対する管理操作が許可されていない）
E_OBJ	オブジェクト状態エラー（対象セマフォは静的 API で生成された）

【機能】

semid で指定したセマフォ（対象セマフォ）を削除する．具体的な振舞いは以下の通り．

対象セマフォの登録が解除され，そのセマフォ ID が未使用の状態に戻される．また，対象セマフォの待ち行列につながれたタスクは，待ち行列の先頭のタスクから順に待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_DLT エラーが返る．

【使用上の注意】

del_sem により複数のタスクが待ち解除される場合，サービスコールの処理時間およびカーネル内での割込み禁止時間が，待ち解除されるタスクの数に比例して長くなる．特に，多くのタスクが待ち解除される場合，カーネル内での割込み禁止時間が長くなるため，注意が必要である．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，del_sem をサポートしない．ただし，動的生成機能拡張パッケージでは，del_sem

をサポートする.

【TOPPERS/FMP カーネルにおける規定】
FMP カーネルでは，del_sem をサポートしない.

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，del_sem をサポートしない.

---

sig_sem　セマフォの資源の返却〔T〕
isig_sem　セマフォの資源の返却〔I〕

【C 言語 API】

```
ER ercd = sig_sem(ID semid)
ER ercd = isig_sem(ID semid)
```

【パラメータ】

ID	semid	対象セマフォの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し：sig_sem の場合，タスクコンテキストからの呼出し：isig_sem の場合，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（semid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象セマフォが未登録）操作 1 が許可されていない：sig_sem の場合）
E_QOVR	キューイングオーバフロー（資源数が最大資源数に一致）

【機能】
semid で指定したセマフォ（対象セマフォ）に資源を返却する．具体的な振舞いは以下の通り.

対象セマフォの待ち行列にタスクが存在する場合には，待ち行列の先頭のタスクが待ち解除される．この時，待ち解除されたタスクが資源を獲得したことになるため，対象セマフォの資源数は変化しない．待ち解除されたタスクには，待ち状態となったサービスコールから E_OK が返る.

待ち行列にタスクが存在しない場合には，対象セマフォの資源数に 1 が加えられる．資源数に 1 を加えるとそのセマフォの最大資源数を越える場合には，E_QOVR エラーとなる．

---

wai_sem　　セマフォの資源の獲得〔T〕
pol_sem　　セマフォの資源の獲得（ポーリング）〔T〕
twai_sem　　セマフォの資源の獲得（タイムアウト付き）〔T〕

【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	対象セマフォの ID 番号
TMO	tmout	タイムアウト時間（twai_sem の場合）

【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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し，ディスパッチ保留状態
E_NOSPT	未サポート機能（制約タスクからの呼出し：pol_sem を除く）
E_ID	不正 ID 番号（semid が不正）
E_PAR	パラメータエラー（tmout が不正：twai_sem の場合）
E_NOEXS〔D〕	オブジェクト未登録（対象セマフォが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象セマフォに対する通常操作 2 が許可されていない）
E_TMOUT	ポーリング失敗またはタイムアウト（wai_sem を除く）
E_RLWAI	待ち禁止状態または待ち状態の強制解除（pol_sem を除く）
E_DLT	待ちオブジェクトの削除または再初期化（pol_sem を除く）

【機能】

semid で指定したセマフォ（対象セマフォ）から資源を獲得する．具体的な振舞いは以下の通り．

対象セマフォの資源数が 1 以上の場合には，資源数から 1 が減ぜられる．資源数が 0 の場合には，自タスクはセマフォの資源獲得待ち状態となり，対象セマフォの待ち行列につながれる．

ini_sem　　セマフォの再初期化〔T〕

【C 言語 API】

```
ER ercd = ini_sem(ID semid)
```

【パラメータ】

ID	semid	対象セマフォの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（semid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象セマフォが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象セマフォに対する管理操作が許可されていない）

【機能】

semid で指定したセマフォ（対象セマフォ）を再初期化する．具体的な振舞いは以下の通り．

対象セマフォの資源数は，初期資源数に初期化される．また，対象セマフォの待ち行列につながれたタスクは，待ち行列の先頭のタスクから順に待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_DLT エラーが返る．

【使用上の注意】

ini_sem により複数のタスクが待ち解除される場合，サービスコールの処理時間およびカーネル内での割込み禁止時間が，待ち解除されるタスクの数に比例して長くなる．特に，多くのタスクが待ち解除される場合，カーネル内での割込み禁止時間が長くなるため，注意が必要である．

セマフォを再初期化した場合に，アプリケーションとの整合性を保つのは，アプリケーションの責任である．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様に定義されていないサービスコールである．

ref_sem　　セマフォの状態参照〔T〕

【C 言語 API】

```
ER ercd = ref_sem(ID semid, T_RSEM *pk_rsem)
```

【パラメータ】

ID	semid	対象セマフォの ID 番号
T_RSEM *	pk_rsem	セマフォの現在状態を入れる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

＊セマフォの現在状態（パケットの内容）

ID	wtskid	セマフォの待ち行列の先頭のタスクの ID 番号
uint_t	semcnt	セマフォの資源数

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（semid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象セマフォが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象セマフォに対する参照操作が許可されていない）
E_MACV〔P〕	メモリアクセス違反（pk_rsem が指すメモリ領域への書込みアクセスが許可されていない）

【機能】

semid で指定したセマフォ（対象セマフォ）の現在状態を参照する．参照した現在状態は，pk_rsem で指定したパケットに返される．

対象セマフォの待ち行列にタスクが存在しない場合，wtskid には TSK_NONE（＝0）が返る．

【使用上の注意】

ref_sem はデバッグ時向けの機能であり，その他の目的に使用することは推奨しない．これは，ref_sem を呼び出し，対象セマフォの現在状態を参照した直後に割込みが発生した場合，ref_sem から戻ってきた時には対象セマフォの状態が変化している可能性があるためである．

4.4.2. イベントフラグ

イベントフラグは，イベントの発生の有無を表すビットの集合（ビットパターン）を介して，イベント通知を行うための同期・通信オブジェクトである．イベントが発生している状態を 1，発生していない状態を 0 とし，ビットパターンにより複数のイベントの発生の有無を表す．イベントフラグは，イベントフラグ ID と呼ぶ ID 番号によって識別する．

1 つまたは複数のビットをセットする 1 にする（セットする）ことを，イベントフラグをセットするといい，0 にする（クリアする）ことを，イベントフラグをクリアするという．イベントフラグによりイベントを通知する側のタスクは，イベントフラグをセットまたはクリアすることで，イベントの発生を通知する．

イベントフラグによりイベントの通知を受ける側のタスクは，待ちビットパターンと待ちモードにより，どのビットがセットされるのを待つかを指定する．待ちモードに TWF_ORW（＝0x01U）を指定した場合，待ちビットパターンに含まれるいずれかのビットがセットされるのを待つ．待ちモードに TWF_ANDW（＝0x02U）を指定した場合，待ちビットパターンに含まれるすべてのビットがセットされるのを待つ．この条件を，イベントフラグの待ち解除の条件と呼ぶ．

各イベントフラグが持つ情報は次の通り．

- イベントフラグ属性
- ビットパターン（の現在値）
- 待ち行列（イベントフラグ待ち状態のタスクのキュー）
- 初期ビットパターン
- アクセス許可ベクタ（保護機能対応カーネルの場合）
- 属する保護ドメイン（保護機能対応カーネルの場合）
- 属するクラス（マルチプロセッサ対応カーネルの場合）

待ち行列は，イベントフラグが指定した待ち解除の条件を満たすまで待っている状態（イベントフラグ待ち状態）のタスクがつながれているキューである．待ち行列につながれたタスクの待ち解除は，待ち解除の条件を満たした中で，待ち行列の前方につながれたものから順に行われる（「2.6.4 待ち行列と待ち解除の順序」の節の(a)に該当）．

イベントフラグの初期ビットパターンは，イベントフラグを生成または再初期化した際の，ビットパターンの初期値である．

イベントフラグ属性には，次の属性を指定することができる．

TA_TPRI	0x01U	待ち行列をタスクの優先度順にする
TA_WMUL	0x02U	複数のタスクが待つのを許す
TA_CLR	0x04U	タスクの待ち解除時にイベントフラグをクリアする

TA_TPRI を指定しない場合，待ち行列は FIFO 順になる．TA_WMUL を指定しない場合，1 つのイベントフラグに複数のタスクが待つことを禁止する．

TA_CLR を指定した場合，タスクの待ち解除時に，イベントフラグのビットパターンを 0 にクリアする．TA_CLR を指定しない場合，タスクの待ち解除時にイベントフラグをクリアしない．

イベントフラグ機能に用いるデータ型は次の通り．

FLGPTN	イベントフラグのビットパターン(符号無し整数, uint_t に定義)

イベントフラグ機能に関連するカーネル構成マクロは次の通り．

TBIT_FLGPTN	イベントフラグのビット数（FLGPTN の有効ビット数）
TNUM_FLGID	登録できるイベントフラグの数 (動的生成対応でないカーネルでは，静的 API によって登録されたイベントフラグの数に一致)

【TOPPERS/ASP カーネルにおける規定】
ASP カーネルでは，イベントフラグのビット数（TBIT_FLGPTN）は 16 以上である．

【TOPPERS/FMP カーネルにおける規定】
FMP カーネルでは，イベントフラグのビット数（TBIT_FLGPTN）は 16 以上である．

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，イベントフラグのビット数（TBIT_FLGPTN）は 16 以上である．

【μITRON4.0 仕様との関係】
TNUM_FLGID は，μITRON4.0 仕様に規定されていないカーネル構成マクロである．

CRE_FLG　イベントフラグの生成〔S〕
acre_flg　イベントフラグの生成〔TD〕

【静的 API】

```
CRE_FLG(ID flgid, { ATR flgatr, FLGPTN iflgptn })
```

【C 言語 API】

```
ER_ID flgid = acre_flg(const T_CFLG *pk_cflg)
```

【パラメータ】

ID	flgid	生成するイベントフラグの ID 番号（CRE_FLG の場合）
T_CFLG *	pk_cflg	イベントフラグの生成情報を入れたパケットへのポインタ（静的 API を除く）

＊イベントフラグの生成情報（パケットの内容）

ATR	flgatr	イベントフラグ属性
uint_t	iflgptn	イベントフラグの初期ビットパターン

【リターンパラメータ】

ER_ID	flgid	生成され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_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_RSATR	予約属性（flgatr が不正または使用できない，属する保護ドメインかクラスが不正）
E_OACV〔sP〕	オブジェクトアクセス違反（システム状態に対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（pk_cflg が指すメモリ領域への読出しアクセスが許可されていない）
E_NOID〔sD〕	ID 番号不足（割り付けられるイベントフラグ ID がない）オブジェクト状態エラー（flgid で指定したイベントフラグが登録済み：CRE_FLG の場合）

【機能】

各パラメータで指定したイベントフラグ生成情報に従って，イベントフラグを生成する．生成されたイベントフラグのビットパターンは初期ビットパターンに，待ち行列は空の状態に初期化される．

静的 API においては，flgid はオブジェクト識別名，iflgptn は整数定数式パラメータである．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，CRE_FLG のみをサポートする．ただし，動的生成機能拡張パッケージでは，acre_flg もサポートする．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，CRE_FLG のみをサポートする．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，CRE_FLG のみをサポートする．

AID_FLG　　割付け可能なイベントフラグ ID の数の指定〔SD〕

【静的 API】

```
AID_FLG(uint_t noflg)
```

【パラメータ】

uint_t	noflg	割付け可能なイベントフラグ ID の数

【エラーコード】

E_RSATR	予約属性（属する保護ドメインまたはクラスが不正）

【機能】

noflg で指定した数のイベントフラグ ID を，イベントフラグを生成するサービスコールによって割付け可能なイベントフラグ ID として確保する．

noflg は整数定数式パラメータである．

SAC_FLG　　イベントフラグのアクセス許可ベクタの設定〔SP〕
sac_flg　　イベントフラグのアクセス許可ベクタの設定〔TPD〕

【静的 API】

```
SAC_FLG(ID flgid, {ACPTN acptn1, ACPTN acptn2,
                   ACPTN acptn3, ACPTN acptn4})
```

【C 言語 API】

```
ER ercd = sac_flg(ID flgid, const ACVCT *p_acvct)
```

【パラメータ】

ID	flgid	対象イベントフラグの ID 番号
ACVCT *	p_acvct	アクセス許可ベクタを入れたパケットへのポインタ（静的 API を除く）

＊アクセス許可ベクタ（パケットの内容）

ACPTN	acptn1	通常操作 1 のアクセス許可パターン
ACPTN	acptn2	通常操作 2 のアクセス許可パターン
ACPTN	acptn3	管理操作のアクセス許可パターン
ACPTN	acptn4	参照操作のアクセス許可パターン

【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_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（flgid が不正）
E_RSATR	予約属性（属する保護ドメインかクラスが不正：SAC_FLG の場合）
E_NOEXS〔D〕	オブジェクト未登録（対象イベントフラグが未登録）
E_OACV〔sP〕	オブジェクトアクセス違反（対象イベントフラグに対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（p_acvct が指すメモリ領域への読出しアクセスが許可されていない）
E_OBJ	オブジェクト状態エラー（対象イベントフラグは静的 API で生成された：sac_flg の場合，対象イベントフラグに対してアクセス許可ベクタが設定済み：SAC_FLG の場合）

【機能】

flgid で指定したイベントフラグ（対象イベントフラグ）のアクセス許可ベクタ（4 つのアクセス許可パターンの組）を，各パラメータで指定した値に設定する．

静的 API においては，flgid はオブジェクト識別名，acptn1～acptn4 は整数定数式パラメータである．

SAC_FLG は，対象イベントフラグが属する保護ドメインの囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．

【TOPPERS/ASP カーネルにおける規定】
ASP カーネルでは，SAC_FLG，sac_flg をサポートしない．

【TOPPERS/FMP カーネルにおける規定】
FMP カーネルでは，SAC_FLG，sac_flg をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，SAC_FLG のみをサポートする．

del_flg イベントフラグの削除〔TD〕

【C 言語 API】

```
ER ercd = del_flg(ID flgid)
```

【パラメータ】

ID	flgid	対象イベントフラグの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（flgid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象イベントフラグが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象イベントフラグに対する管理操作が許可されていない）
E_OBJ	オブジェクト状態エラー（対象イベントフラグは静的 API で生成された）

【機能】
flgid で指定したイベントフラグ（対象イベントフラグ）を削除する．具体的な振舞いは以下の通り．

対象イベントフラグの登録が解除され，そのイベントフラグ ID が未使用の状態に戻される．また，対象イベントフラグの待ち行列につながれたタスクは，待ち行列の先頭のタスクから順に待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_DLT エラーが返る．

【使用上の注意】

del_flg により複数のタスクが待ち解除される場合，サービスコールの処理時間およびカーネル内での割込み禁止時間が，待ち解除されるタスクの数に比例して長くなる．特に，多くのタスクが待ち解除される場合，カーネル内での割込み禁止時間が長くなるため，注意が必要である．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，del_flg をサポートしない．ただし，動的生成機能拡張パッケージでは，del_flg をサポートする．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，del_flg をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，del_flg をサポートしない．

---

set_flg　　イベントフラグのセット〔T〕
iset_flg イベントフラグのセット〔I〕

【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	対象イベントフラグの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し：set_flg の場合，タスクコンテキストからの呼出し：iset_flg の場合，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（flgid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象イベントフラグが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象イベントフラグに対する通常操作 1 が許可されていない：set_flg の場合）

【機能】

flgid で指定したイベントフラグ（対象イベントフラグ）の setptn で指定したビットをセットする．具体的な振舞いは以下の通り．

対象イベントフラグのビットパターンは，それまでの値と setptn で指定した値のビット毎論理和（C 言語の"|"）に更新される．対象イベントフラグの待ち行列にタスクが存在する場合には，待ち解除の条件を満たしたタスクが，待ち行列の前方につながれたものから順に待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_OK が返る．

ただし，対象イベントフラグが TA_CLR 属性である場合には，待ち解除の条件を満たしたタスクを 1 つ待ち解除した時点で，対象イベントフラグのビットパターンが 0 にクリアされるため，他のタスクが待ち解除されることはない．

【使用上の注意】

対象イベントフラグが，TA_WMUL 属性であり，TA_CLR 属性でない場合，set_flg または iset_flg により複数のタスクが待ち解除される場合がある．この場合，サービスコールの処理時間およびカーネル内での割込み禁止時間が，待ち解除されるタスクの数に比例して長くなる．特に，多くのタスクが待ち解除される場合，カーネル内での割込み禁止時間が長くなるため，注意が必要である．

clr_flg　イベントフラグのクリア〔T〕

【C 言語 API】

```
ER ercd = clr_flg(ID flgid, FLGPTN clrptn)
```

【パラメータ】

ID	flgid	対象イベントフラグの ID 番号
FLGPTN	clrptn	クリアするビットパターン（クリアしないビットを 1，クリアするビットを 0 とする

【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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（flgid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象イベントフラグが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象イベントフラグに対する通常操作 1 が許可されていない：clr_flg の場合）

【機能】

flgid で指定したイベントフラグ（対象イベントフラグ）の clrptn で指定した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は，それまでの値と clrptn で指定した値のビット毎論理積（C 言語の"&"）に更新される．

---

wai_flg　　イベントフラグ待ち〔T〕
pol_flg　　イベントフラグ待ち（ポーリング）〔T〕
twai_flg　　イベントフラグ待ち（タイムアウト付き）〔T〕

【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	対象イベントフラグの ID 番号
FLGPTN	waiptn	待ちビットパターン
MODE	wfmode	待ちモード
FLGPTN *	p_flgptn	待ち解除時のビットパターンを入れるメモリ領域へのポインタ
TMO	tmout	タイムアウト時間（twai_flg の場合）

【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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し，ディスパッチ保留状態からの呼出し：pol_flg を除く
E_NOSPT	未サポート機能（制約タスクからの呼出し：pol_flg を除く）
E_ID	不正 ID 番号（flgid が不正）
E_PAR	パラメータエラー（waiptn，wfmode が不正，tmout が不正：twai_flg の場合）
E_NOEXS〔D〕	オブジェクト未登録（対象イベントフラグが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象イベントフラグに対する通常操作 2 が許可されていない）
E_MACV〔P〕	メモリアクセス違反（p_flgptn が指すメモリ領域への書込みアクセスが許可されていない）
E_ILUSE	サービスコール不正使用（TA_WMUL 属性でないイベントフラグで待ちタスクあり）
E_TMOUT	ポーリング失敗またはタイムアウト（wai_flg を除く）
E_RLWAI	待ち禁止状態または待ち状態の強制解除（pol_flg を除く）
E_DLT	待ちオブジェクトの削除または再初期化（pol_flg を除く）

【機能】

flgid で指定したイベントフラグ（対象イベントフラグ）が，waiptn と wfmode で指定した待ち解除の条件を満たすのを待つ．具体的な振舞いは以下の通り．

対象イベントフラグが，waiptn と wfmode で指定した待ち解除の条件を満たしている場合には，対象イベントフラグのビットパターンの現在値が flgptn に返される．対象イベントフラグが TA_CLR 属性である場合には，対象イベントフラグのビットパターンが 0 にクリアされる．

待ち解除の条件を満たしていない場合には，自タスクはイベントフラグ待ち状態となり，対象イベントフラグの待ち行列につながれる．

ini_flg イベントフラグの再初期化〔T〕

【C 言語 API】

```
ER ercd = ini_flg(ID flgid)
```

【パラメータ】

ID	flgid	対象イベントフラグの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（flgid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象イベントフラグが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象イベントフラグに対する管理操作が許可されていない）

【機能】

flgid で指定したイベントフラグ（対象イベントフラグ）を再初期化する．具体的な振舞いは以下の通り．

対象イベントフラグのビットパターンは，初期ビットパターンに初期化される．また，対象イベントフラグの待ち行列につながれたタスクは，待ち行列の先頭のタスクから順に待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_DLT エラーが返る．

【使用上の注意】

ini_flg により複数のタスクが待ち解除される場合，サービスコールの処理時間およびカーネル内での割込み禁止時間が，待ち解除されるタスクの数に比例して長くなる．特に，多くのタスクが待ち解除される場合，カーネル内での割込み禁止時間が長くなるため，注意が必要である．

イベントフラグを再初期化した場合に，アプリケーションとの整合性を保つのは，アプリケーションの責任である．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様に定義されていないサービスコールである.

---

ref_flg　　イベントフラグの状態参照〔T〕

【C 言語 API】

```
ER ercd = ref_flg(ID flgid, T_RFLG *pk_rflg)
```

【パラメータ】

ID	flgid	対象イベントフラグの ID 番号
T_RFLG *	pk_rflg	イベントフラグの現在状態を入れる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

＊イベントフラグの現在状態（パケットの内容）

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
uint_t	flgptn	イベントフラグのビットパターン

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（flgid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象イベントフラグが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象イベントフラグに対す参照操作が許可されていない）
E_MACV〔P〕	メモリアクセス違反（pk_rflg が指すメモリ領域への書込みアクセスが許可されていない）

【機能】

flgid で指定したイベントフラグ（対象イベントフラグ）の現在状態を参照する．参照した現在状態は，pk_rflg で指定したパケットに返される．

対象イベントフラグの待ち行列にタスクが存在しない場合，wtskid には TSK_NONE（＝0）が返る．

【使用上の注意】

ref_flg はデバッグ時向けの機能であり，その他の目的に使用することは推奨しない．これは，ref_flg を呼び出し，対象イベントフラグの現在状態を参照した直後に割込みが発生した場合，ref_flg から戻ってきた時には対象イベントフラグの状態が変化している可能性があるためである．

4.4.3. 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，FIFO 順で送受信するための同期・通信オブジェクトである．より大きいサイズのメッセージを送受信したい場合には，メッセージを置いたメモリ領域へのポインタを 1 ワードのデータとして送受信する方法がある．データキューは，データキューID と呼ぶ ID 番号によって識別する．

各データキューが持つ情報は次の通り．

- データキュー属性
- データキュー管理領域
- 送信待ち行列（データキューへの送信待ち状態のタスクのキュー）
- 受信待ち行列（データキューからの受信待ち状態のタスクのキュー）
- アクセス許可ベクタ（保護機能対応カーネルの場合）
- 属する保護ドメイン（保護機能対応カーネルの場合）
- 属するクラス（マルチプロセッサ対応カーネルの場合）

データキュー管理領域は，データキューに送信されたデータを，送信された順に格納しておくためのメモリ領域である．データキュー生成時に，データキュー管理領域に格納できるデータ数を 0 とすることで，データキュー管理領域のサイズを 0 とすることができる．

保護機能対応カーネルにおいて，データキュー管理領域は，カーネルの用いるオブジェクト管理領域として扱われる．

送信待ち行列は，データキューに対してデータが送信できるまで待っている状態（データキューへの送信待ち状態）のタスクが，データを送信できる順序でつながれているキューである．また，受信待ち行列は，データキューからデータが受信できるまで待っている状態（データキューからの受信待ち状態）のタスクが，データを受信できる順序でつながれているキューである．

データキュー属性には，次の属性を指定することができる．

TA_TPRI	0x01U	送信待ち行列をタスクの優先度順にする

TA_TPRI を指定しない場合，送信待ち行列は FIFO 順になる．受信待ち行列は，FIFO 順に固定され

ている．

データキュー機能に関連するカーネル構成マクロは次の通り．

TNUM_DTQID	登録できるデータキューの数（動的生成対応でないカーネルでは，静的 API によって登録されたデータキューの数に一致）

【μITRON4.0 仕様との関係】
TNUM_DTQID は，μITRON4.0 仕様に規定されていないカーネル構成マクロである．

CRE_DTQ　データキューの生成〔S〕
acre_dtq　データキューの生成〔TD〕

【静的 API】

```
CRE_DTQ(ID dtqid, {ATR dtqatr, uint_t dtqcnt, void *dtqmb })
```

【C 言語 API】

```
ER_ID dtqid = acre_dtq(const T_CDTQ *pk_cdtq)
```

【パラメータ】

ID	dtqid	生成するデータキューの ID 番号（CRE_DTQ の場合）
T_CDTQ *	pk_cdtq	データキューの生成情報を入れたパケットへのポインタ（静的 API を除く）

＊データキューの生成情報（パケットの内容）

ATR	dtqatr	データキュー属性
uint_t	dtqcnt	データキュー管理領域に格納できるデータ数
void *	dtqmb	データキュー管理領域の先頭番地

【リターンパラメータ】

ER_ID	dtqid	生成されたデータキューの ID 番号（正の値）またはエラーコード

【エラーコード】

E_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_RSATR	予約属性（dtqatr が不正または使用できない，属する保護ドメインかクラスが不正）
E_NOSPT	未サポート機能（dtqmb がサポートされていない値）
E_PAR	パラメータエラー（dtqmb が不正）
E_OACV〔sP〕	オブジェクトアクセス違反（システム状態に対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（pk_cdtq が指すメモリ領域への読出しアクセスが許可されていない）
E_NOID〔sD〕	ID 番号不足（割り付けられるデータキューID がない）
E_NOMEM	メモリ不足（データキュー管理領域が確保できない）
E_OBJ	オブジェクト状態エラー（dtqid で指定したデータキューが登録済み：CRE_DTQ の場合，その他の条件については機能の項を参照すること）

【機能】

各パラメータで指定したデータキュー生成情報に従って，データキューを生成する．dtqcnt と dtqmb からデータキュー管理領域が設定され，格納されているデータがない状態に初期化される．また，送信待ち行列と受信待ち行列は，空の状態に初期化される．

静的 API においては，dtqid はオブジェクト識別名，dtqcnt は整数定数式パラメータ，dtqmb は一般定数式パラメータである．コンフィギュレータは，静的 API のメモリ不足（E_NOMEM）エラーを検出することができない．

dtqmb を NULL とした場合，dtqcnt で指定した数のデータを格納できるデータキュー管理領域を，コンフィギュレータまたはカーネルが確保する．

〔dtqmb に NULL 以外を指定した場合〕

dtqmb に NULL 以外を指定した場合，dtqmb を先頭番地とするデータキュー管理領域は，アプリケーションで確保しておく必要がある．データキュー管理領域をアプリケーションで確保するために，次のマクロを用意している．

TSZ_DTQMB(dtqcnt)	dtqcnt で指定した数のデータを格納できるデータキュー管理領域のサイズ（バイト数）
TCNT_DTQMB(dtqcnt)	dtqcnt で指定した数のデータを格納できるデータキュー管理領域を確保するために必要な MB_T 型の配列の要素数

これらを用いてデータキュー管理領域を確保する方法は次の通り．

```
MB_T　<データキュー管理領域の変数名>[TCNT_DTQMB(dtqcnt)];
```

この時，dtqmb には<データキュー管理領域の変数名>を指定する．

この方法に従わず，dtqmb にターゲット定義の制約に合致しない先頭番地を指定した時には，E_PAR エラーとなる．また，保護機能対応カーネルにおいて，dtqmb で指定したデータキュー管理領域がカーネル専用のメモリオブジェクトに含まれない場合，E_OBJ エラーとなる．

【TOPPERS/ASP カーネルにおける規定】
ASP カーネルでは，CRE_DTQ のみをサポートする．また，dtqmb には NULL のみを指定することができる．NULL 以外を指定した場合には，E_NOSPT エラーとなる．ただし，動的生成機能拡張パッケージでは，acre_dtq もサポートする．acre_dtq に対しては，dtqmb に NULL 以外を指定できないという制限はない．

【TOPPERS/FMP カーネルにおける規定】
FMP カーネルでは，CRE_DTQ のみをサポートする．また，dtqmb には NULL のみを指定することができる．NULL 以外を指定した場合には，E_NOSPT エラーとなる．

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，CRE_DTQ のみをサポートする．また，dtqmb には NULL のみを指定することができる．NULL 以外を指定した場合には，E_NOSPT エラーとなる．

【μITRON4.0 仕様との関係】
μITRON4.0/PX 仕様にあわせて，データキュー生成情報の最後のパラメータを，dtq（データキュー領域の先頭番地）から，dtqmb（データキュー管理領域の先頭番地）に改名した．また，TSZ_DTQ を TSZ_DTQMB に改名した．

TCNT_DTQMB を新設し，データキュー管理領域をアプリケーションで確保する方法を規定した．

AID_DTQ　　割付け可能なデータキューID の数の指定〔SD〕

【静的 API】

```
AID_DTQ(uint_t nodtq)
```

【パラメータ】

uint_t	nodtq	割付け可能なデータキューID の数

【エラーコード】

E_RSATR	予約属性（属する保護ドメインまたはクラスが不正）

【機能】

nodtq で指定した数のデータキューID を，データキューを生成するサービスコールによって割付け可能なデータキューID として確保する．

nodtq は整数定数式パラメータである．

SAC_DTQ　　データキューのアクセス許可ベクタの設定〔SP〕
sac_dtq　　データキューのアクセス許可ベクタの設定〔TPD〕

【静的 API】

```
AC_DTQ(ID dtqid, {ACPTN acptn1, ACPTN acptn2,
                                ACPTN acptn3, ACPTN acptn4 })
```

【C 言語 API】

```
ER ercd = sac_dtq(ID dtqid, const ACVCT *p_acvct)
```

【パラメータ】

ID	dtqid	対象データキューの ID 番号
ACVCT *	p_acvct	アクセス許可ベクタを入れたパケットへのポインタ（静的 API を除く）

＊アクセス許可ベクタ（パケットの内容）

ACPTN	acptn1	通常操作 1 のアクセス許可パターン
ACPTN	acptn2	通常操作 2 のアクセス許可パターン
ACPTN	acptn3	管理操作のアクセス許可パターン
ACPTN	acptn4	参照操作のアクセス許可パターン

【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_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（dtqid が不正）
E_RSATR	予約属性（属する保護ドメインかクラスが不正：SAC_DTQ の場合）
E_NOEXS〔D〕	オブジェクト未登録（対象データキューが未登録）
E_OACV〔sP〕	オブジェクトアクセス違反（対象データキューに対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（p_acvct が指すメモリ領域への読出しアクセスが許可されていない）
E_OBJ	オブジェクト状態エラー（対象データキューは静的 API で生成された：sac_dtq の場合，対象データキューに対してアクセス許可ベクタが設定済み：SAC_DTQ の場合）

【機能】

dtqid で指定したデータキュー（対象データキュー）のアクセス許可ベクタ（4 つのアクセス許可パターンの組）を，各パラメータで指定した値に設定する．

静的 API においては，dtqid はオブジェクト識別名，acptn1～acptn4 は整数定数式パラメータである．

SAC_DTQ は，対象データキューが属する保護ドメインの囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，SAC_DTQ，sac_dtq をサポートしない．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，SAC_DTQ，sac_dtq をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，SAC_DTQ のみをサポートする．

del_dtq　　データキューの削除〔TD〕

【C 言語 API】

```
ER ercd = del_dtq(ID dtqid)
```

【パラメータ】

ID	dtqid	対象データキューの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（dtqid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象データキューが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象データキューに対する管理操作が許可されていない）
E_OBJ	オブジェクト状態エラー（対象データキューは静的 API で生成された）

【機能】

dtqid で指定したデータキュー（対象データキュー）を削除する．具体的な振舞いは以下の通り．

対象データキューの登録が解除され，そのデータキューID が未使用の状態に戻される．また，対象データキューの送信待ち行列と受信待ち行列につながれたタスクは，それぞれの待ち行列の先頭のタスクから順に待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_DLT エラーが返る．

データキューの生成時に，データキュー管理領域がカーネルによって確保された場合は，そのメモリ領域が解放される．

【補足説明】

送信待ち行列と受信待ち行列の両方にタスクがつながれていることはないため，別の待ち行列で待っていたタスクの間の待ち解除の順序は，規定する必要がない．

【使用上の注意】

del_dtq により複数のタスクが待ち解除される場合，サービスコールの処理時間およびカーネル内での割込み禁止時間が，待ち解除されるタスクの数に比例して長くなる．特に，多くのタスクが待ち解除される場合，カーネル内での割込み禁止時間が長くなるため，注意が必要である．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，del_dtq をサポートしない．ただし，動的生成機能拡張パッケージでは，del_dtq をサポートする．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，del_dtq をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，del_dtq をサポートしない．

---

snd_dtq　　データキューへの送信〔T〕
psnd_dtq　　データキューへの送信（ポーリング）〔T〕
ipsnd_dtq　　データキューへの送信（ポーリング）〔I〕
tsnd_dtq　　データキューへの送信（タイムアウト付き）〔T〕

【C 言語 API】

```
ER ercd = snd_dtq(ID dtqid, intptr_t data)
ER ercd = psnd_dtq(ID dtqid, intptr_t data)
ER ercd = ipsnd_dtq(ID dtqid, intptr_t data)
ER ercd = tsnd_dtq(ID dtqid, intptr_t data, TMO tmout)
```

【パラメータ】

ID	dtqid	対象データキューの ID 番号
intptr_t	data	送信データ
TMO	tmout	タイムアウト時間（tsnd_dtq の場合）

【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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し：ipsnd_dtq を除く，タスクコンテキストからの呼出し：ipsnd_dtq の場合，CPU ロック状態からの呼出し，ディスパッチ保留状態からの呼出し：snd_dtq と tsnd_dtq の場合）
E_NOSPT	未サポート機能（制約タスクからの呼出し：snd_dtq と tsnd_dtq の場合）
E_ID	不正 ID 番号（dtqid が不正）
E_PAR	パラメータエラー（tmout が不正：tsnd_dtq の場合）
E_NOEXS〔D〕	オブジェクト未登録（対象データキューが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象データキューに対する通常操作 1 が許可されていない：ipsnd_dtq を除く）
E_TMOUT	ポーリング失敗またはタイムアウト（snd_dtq を除く）
E_RLWAI	待ち禁止状態または待ち状態の強制解除（snd_dtq と tsnd_dtq の場合）
E_DLT	待ちオブジェクトの削除または再初期化（snd_dtq と tsnd_dtq の場合）

【機能】

dtqid で指定したデータキュー（対象データキュー）に，data で指定したデータを送信する．具体的な振舞いは以下の通り．

対象データキューの受信待ち行列にタスクが存在する場合には，受信待ち行列の先頭のタスクが，data で指定したデータを受信し，待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_OK が返る．

対象データキューの受信待ち行列にタスクが存在せず，データキュー管理領域にデータを格納するスペースがある場合には，data で指定したデータが，FIFO 順でデータキュー管理領域に格納される．

対象データキューの受信待ち行列にタスクが存在せず，データキュー管理領域にデータを格納するスペースがない場合には，自タスクはデータキューへの送信待ち状態となり，対象データキューの送信待ち行列につながれる．

---

fsnd_dtq　データキューへの強制送信〔T〕
ifsnd_dtq　データキューへの強制送信〔I〕

【C 言語 API】

```
ER ercd = fsnd_dtq(ID dtqid, intptr_t data)
ER ercd = ifsnd_dtq(ID dtqid, intptr_t data)
```

【パラメータ】

ID	dtqid	対象データキューの ID 番号
intptr_t	data	送信データ

【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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し：fsnd_dtq の場合，タスクコンテキストからの呼出し：ifsnd_dtq の場合，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（dtqid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象データキューが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象データキューに対する通常操作 1 が許可されていない：fsnd_dtq の場合）
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管理領域のサイズが 0）

【機能】

dtqid で指定したデータキュー（対象データキュー）に，data で指定したデータを強制送信する．具体的な振舞いは以下の通り．

対象データキューの受信待ち行列にタスクが存在する場合には，受信待ち行列の先頭のタスクが，data で指定したデータを受信し，待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_OK が返る．

対象データキューの受信待ち行列にタスクが存在せず，データキュー管理領域にデータを格納するスペースがある場合には，data で指定したデータが，FIFO 順でデータキュー管理領域に格納される．

対象データキューの受信待ち行列にタスクが存在せず，データキュー管理領域にデータを格納するスペースがない場合には，データキュー管理領域の先頭に格納されたデータを削除し，空いたスペースを用いて，data で指定したデータが，FIFO 順でデータキュー管理領域に格納される．

対象データキューのデータキュー管理領域のサイズが 0 の場合には，E_ILUSE エラーとなる．

rcv_dtq　　データキューからの受信〔T〕
prcv_dtq　　データキューからの受信（ポーリング）〔T〕
trcv_dtq　　データキューからの受信（タイムアウト付き）〔T〕

【C 言語 API】

```
ER ercd = rcv_dtq(ID dtqid, intptr_t *p_data)
ER ercd = prcv_dtq(ID dtqid, intptr_t *p_data)
ER ercd = trcv_dtq(ID dtqid, intptr_t *p_data, TMO tmout)
```

【パラメータ】

ID	dtqid	対象データキューの ID 番号
intptr_t *	p_data	受信データを入れるメモリ領域へのポインタ
TMO	tmout	タイムアウト時間（trcv_dtq の場合）

【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
intptr_t	data	受信データ

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し，ディスパッチ保留状態からの呼出し：prcv_dtq を除く）
E_NOSPT	未サポート機能（制約タスクからの呼出し：prcv_dtq を除く）
E_ID	不正 ID 番号（dtqid が不正）
E_PAR	パラメータエラー（tmout が不正：trcv_dtq の場合）
E_NOEXS〔D〕	オブジェクト未登録（対象データキューが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象データキューに対する通常操作 2 が許可されていない）
E_MACV〔P〕	メモリアクセス違反（p_data が指すメモリ領域への書込みアクセスが許可されていない）
E_TMOUT	ポーリング失敗またはタイムアウト（rcv_dtq を除く）
E_RLWAI	待ち禁止状態または待ち状態の強制解除（prcv_dtq を除く）
E_DLT	待ちオブジェクトの削除または再初期化（prcv_dtq を除く）

【機能】

dtqid で指定したデータキュー（対象データキュー）からデータを受信する．受信したデータは，p_data

で指定したメモリ領域に返される．具体的な振舞いは以下の通り．

対象データキューのデータキュー管理領域にデータが格納されている場合には，データキュー管理領域の先頭に格納されたデータが取り出され，p_data で指定したメモリ領域に返される．また，送信待ち行列にタスクが存在する場合には，送信待ち行列の先頭のタスクの送信データが，FIFO 順でデータキュー管理領域に格納され，そのタスクは待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_OK が返る．

対象データキューのデータキュー管理領域にデータが格納されておらず，送信待ち行列にタスクが存在する場合には，送信待ち行列の先頭のタスクの送信データが，p_data で指定したメモリ領域に返される．送信待ち行列の先頭のタスクは，待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_OK が返る．

対象データキューのデータキュー管理領域にデータが格納されておらず，送信待ち行列にタスクが存在しない場合には，自タスクはデータキューからの受信待ち状態となり，対象データキューの受信待ち行列につながれる．

ini_dtq　　データキューの再初期化〔T〕

【C 言語 API】

```
ER ercd = ini_dtq(ID dtqid)
```

【パラメータ】

ID	dtqid	対象データキューの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（dtqid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象データキューが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象データキューに対する管理操作が許可されていない）

【機能】

dtqid で指定したデータキュー（対象データキュー）を再初期化する．具体的な振舞いは以下の通り．

対象データキューのデータキュー管理領域は，格納されているデータがない状態に初期化される．また，対象データキューの送信待ち行列と受信待ち行列につながれたタスクは，それぞれの待ち行列の先頭のタスクから順に待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_DLT エラーが返る．

【補足説明】

送信待ち行列と受信待ち行列の両方にタスクがつながれていることはないため，別の待ち行列で待っていたタスクの間の待ち解除の順序は，規定する必要がない．

【使用上の注意】

ini_dtq により複数のタスクが待ち解除される場合，サービスコールの処理時間およびカーネル内での割込み禁止時間が，待ち解除されるタスクの数に比例して長くなる．特に，多くのタスクが待ち解除される場合，カーネル内での割込み禁止時間が長くなるため，注意が必要である．

データキューを再初期化した場合に，アプリケーションとの整合性を保つのは，アプリケーションの責任である．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様に定義されていないサービスコールである．

ref_dtq　　データキューの状態参照〔T〕

【C 言語 API】

```
ER ercd = ref_dtq(ID dtqid, T_RDTQ *pk_rdtq)
```

【パラメータ】

ID	dtqid	対象データキューの ID 番号
T_RDTQ *	pk_rdtq	データキューの現在状態を入れる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

＊データキューの現在状態（パケットの内容）

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_t	sdtqcnt	データキュー管理領域に格納されているデータの数

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（dtqid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象データキューが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象データキューに対する参照操作が許可されていない）
E_MACV〔P〕	メモリアクセス違反（pk_rdtq が指すメモリ領域への書込みアクセスが許可されていない）

【機能】

dtqid で指定したデータキュー（対象データキュー）の現在状態を参照する．参照した現在状態は，pk_rdtq で指定したパケットに返される．

対象データキューの送信待ち行列にタスクが存在しない場合，stskid には TSK_NONE（＝0）が返る．また，受信待ち行列にタスクが存在しない場合，rtskid には TSK_NONE（＝0）が返る．

【使用上の注意】

ref_dtq はデバッグ時向けの機能であり，その他の目的に使用することは推奨しない．これは，ref_dtq を呼び出し，対象データキューの現在状態を参照した直後に割込みが発生した場合，ref_dtq から戻ってきた時には対象データキューの状態が変化している可能性があるためである．

4.4.4. 優先度データキュー

優先度データキューは，1 ワードのデータをメッセージとして，データの優先度順で送受信するための同期・通信カーネルオブジェクトである．より大きいサイズのメッセージを送受信したい場合には，メッセージを置いたメモリ領域へのポインタを 1 ワードのデータとして送受信する方法がある．優先度データキューは，優先度データキューID と呼ぶ ID 番号によって識別する．

各優先度データキューが持つ情報は次の通り．

- 優先度データキュー属性
- 優先度データキュー管理領域
- 送信待ち行列（優先度データキューへの送信待ち状態のタスクのキュー）
- 受信待ち行列（優先度データキューからの受信待ち状態のタスクのキュー）
- 送信できるデータ優先度の最大値
- アクセス許可ベクタ（保護機能対応カーネルの場合）
- 属する保護ドメイン（保護機能対応カーネルの場合）
- 属するクラス（マルチプロセッサ対応カーネルの場合）

優先度データキュー管理領域は，優先度データキューに送信されたデータを，データの優先度順に格納しておくためのメモリ領域である．優先度データキュー生成時に，優先度データキュー管理領域に格納できるデータ数を 0 とすることで，優先度データキュー管理領域のサイズを 0 とすることができる．

保護機能対応カーネルにおいて，優先度データキュー管理領域は，カーネルの用いるオブジェクト管理領域として扱われる．

送信待ち行列は，優先度データキューに対してデータが送信できるまで待っている状態（優先度データキューへの送信待ち状態）のタスクが，データを送信できる順序でつながれているキューである．また，受信待ち行列は，優先度データキューからデータが受信できるまで待っている状態（優先度データキューからの受信待ち状態）のタスクが，データを受信できる順序でつながれている
キューである．

優先度データキュー属性には，次の属性を指定することができる．

TA_TPRI	0x01U	送信待ち行列をタスクの優先度順にする

TA_TPRI を指定しない場合，送信待ち行列は FIFO 順になる．受信待ち行列は，FIFO 順に固定されている．

優先度データキュー機能に関連するカーネル構成マクロは次の通り．

TMIN_DPRI	データ優先度の最小値（＝1）
TMAX_DPRI	データ優先度の最大値
TNUM_PDQID	登録できる優先度データキューの数（動的生成対応でないカーネルでは，静的 API によって登録された優先度データキューの数に一致）

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，データ優先度の最大値（TMAX_DPRI）は 16 に固定されている．ただし，タスク優先度拡張パッケージでは，TMAX_DPRI を 256 に拡張する．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，データ優先度の最大値（TMAX_DPRI）は 16 に固定されている．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，データ優先度の最大値（TMAX_DPRI）は 16 に固定されている．

【使用上の注意】

データの優先度が使われるのは，データが優先度データキュー管理領域に格納される場合のみであり，データを送信するタスクが送信待ち行列につながれている間には使われない．そのため，送信待ち行列につながれているタスクが，優先度データキュー管理領域に格納されているデータよりも高い優先度のデータを送信しようとしている場合でも，最初に送信されるのは，優先度データキュー管理領域に格納されているデータである．また，TA_TPRI 属性の優先度データキューにおいても，送信待ち行列はタスクの優先度順となり，タスクが送信しようとしているデータの優先度順となるわけではない．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様に規定されていない機能である．

CRE_PDQ　　優先度データキューの生成〔S〕
acre_pdq　　優先度データキューの生成〔TD〕

【静的 API】

```
CRE_PDQ(ID pdqid, {ATR pdqatr, uint_t pdqcnt,
                   PRI maxdpri, void *pdqmb })
```

【C 言語 API】

```
ER_ID pdqid = acre_pdq(const T_CPDQ *pk_cpdq)
```

【パラメータ】

ID	pdqid	生成する優先度データキューの ID 番号（CRE_PDQ の場合）
T_CPDQ *	pk_cpdq	優先度データキューの生成情報を入れたパケットへのポインタ（静的 API を除く）

＊優先度データキューの生成情報（パケットの内容）

ATR	pdqatr	優先度データキュー属性
uint_t	pdqcnt	優先度データキュー管理領域に格納できるデータ数
PRI	maxdpri	優先度データキューに送信できるデータ優先度の最大値
void *	pdqmb	優先度データキュー管理領域の先頭番地

【リターンパラメータ】

ER_ID	pdqid	生成された優先度データキューの ID 番号（正の値）または エラーコード

【エラーコード】

E_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_RSATR	予約属性（pdqatr が不正または使用できない，属する保護ドメインかクラスが不正）
E_NOSPT	未サポート機能（pdqmb がサポートされていない値）パラメータエラー（pdqmb，maxdpri が不正）
E_OACV〔sP〕	オブジェクトアクセス違反（システム状態に対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（pk_cpdq が指すメモリ領域への読出しアクセスが許可されていない）
E_NOID〔sD〕	ID 番号不足（割り付けられる優先度データキューID がない）
E_NOMEM	メモリ不足（優先度データキュー管理領域が確保できない）
E_OBJ	オブジェクト状態エラー（pdqid で指定した優先度データキューが登録済み：CRE_PDQ の場合，その他の条件については機能の項を参照すること）

【機能】

各パラメータで指定した優先度データキュー生成情報に従って，優先度データキューを生成する．pdqcnt と pdqmb から優先度データキュー管理領域が設定され，格納されているデータがない状態に初期化される．また，送信待ち行列と受信待ち行列は，空の状態に初期化される．

静的 API においては，pdqid はオブジェクト識別名，pdqcnt と maxdpri は整数定数式パラメータ，pdqmb は一般定数式パラメータである．コンフィギュレータは，静的 API のメモリ不足（E_NOMEM）エラーを検出することができない．

pdqmb を NULL とした場合，pdqcnt で指定した数のデータを格納できる優先度データキュー管理領域を，コンフィギュレータまたはカーネルが確保する．

maxdpri は，TMIN_DPRI 以上，TMAX_DPRI 以下でなければならない．

〔pdqmb に NULL 以外を指定した場合〕

pdqmb に NULL 以外を指定した場合，pdqmb を先頭番地とする優先度データキュー管理領域は，アプリケーションで確保しておく必要がある．優先度データキュー管理領域をアプリケーションで確保するために，次のマクロを用意している．

TSZ_PDQMB(pdqcnt)	pdqcnt で指定した数のデータを格納できる優先度データキュー管理領域のサイズ（バイト数）
TCNT_PDQMB(pdqcnt)	pdqcnt で指定した数のデータを格納できる優先度データキュー管理領域を確保するために必要な MB_T 型の配列の要素数

これらを用いて優先度データキュー管理領域を確保する方法は次の通り．

```
MB_T　<優先度データキュー管理領域の変数名>[TCNT_PDQMB(pdqcnt)];
```

この時，pdqmb には<優先度データキュー管理領域の変数名>を指定する．

この方法に従わず，pdqmb にターゲット定義の制約に合致しない先頭番地を指定した時には，E_PAR エラーとなる．また，保護機能対応カーネルにおいて，pdqmb で指定した優先度データキュー管理領域がカーネル専用のメモリオブジェクトに含まれない場合，E_OBJ エラーとなる．

【TOPPERS/ASP カーネルにおける規定】
ASP カーネルでは，CRE_PDQ のみをサポートする．また，pdqmb には NULL のみを指定することができる．NULL 以外を指定した場合には，E_NOSPT エラーとなる．ただし，動的生成機能拡張パッケージでは，acre_pdq もサポートする．acre_pdq に対しては，pdqmb に NULL 以外を指定できないという制限はない．

【TOPPERS/FMP カーネルにおける規定】
FMP カーネルでは，CRE_PDQ のみをサポートする．また，pdqmb には NULL のみを渡すことができる．NULL 以外を指定した場合には，E_NOSPT エラーとなる．

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，CRE_PDQ のみをサポートする．また，pdqmb には NULL のみを渡すことができる．NULL 以外を指定した場合には，E_NOSPT エラーとなる．

AID_PDQ　　割付け可能な優先度データキューID の数の指定〔SD〕

【静的 API】

```
AID_PDQ(uint_t nopdq)
```

【パラメータ】

uint_t	nopdq	割付け可能な優先度データキューID の数

【エラーコード】

E_RSATR	予約属性（属する保護ドメインまたはクラスが不正）

【機能】

nopdq で指定した数の優先度データキューID を，優先度データキューを生成するサービスコールによって割付け可能な優先度データキューID として確保する．

nopdq は整数定数式パラメータである．

SAC_PDQ　　優先度データキューのアクセス許可ベクタの設定〔SP〕
sac_pdq　　優先度データキューのアクセス許可ベクタの設定〔TPD〕

【静的 API】

```
SAC_PDQ(ID pdqid, {ACPTN acptn1, ACPTN acptn2,
                   ACPTN acptn3, ACPTN acptn4})
```

【C 言語 API】

```
ER ercd = sac_pdq(ID pdqid, const ACVCT *p_acvct)
```

【パラメータ】

ID	pdqid	対象優先度データキューの ID 番号
ACVCT *	p_acvct	アクセス許可ベクタを入れたパケットへのポインタ（静的 API を除く）

＊アクセス許可ベクタ（パケットの内容）

ACPTN	acptn1	通常操作 1 のアクセス許可パターン
ACPTN	acptn2	通常操作 2 のアクセス許可パターン
ACPTN	acptn3	管理操作のアクセス許可パターン
ACPTN	acptn4	参照操作のアクセス許可パターン

【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_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（pdqid が不正）
E_RSATR	予約属性（属する保護ドメインかクラスが不正：SAC_PDQ の場合）
E_NOEXS〔D〕	オブジェクト未登録（対象優先度データキューが未登録）
E_OACV〔sP〕	オブジェクトアクセス違反（対象優先度データキューに対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（p_acvct が指すメモリ領域への読出しアクセスが許可されていない）
E_OBJ	オブジェクト状態エラー（対象優先度データキューは静的 API で生成された：sac_pdq の場合，対象優先度データキューに対してアクセス許可ベクタが設定済み：SAC_PDQ の場合）

【機能】

pdqid で指定した優先度データキュー（対象優先度データキュー）のアクセス許可ベクタ（4 つのアクセス許可パターンの組）を，各パラメータで指定した値に設定する．

静的 API においては，pdqid はオブジェクト識別名，acptn1～acptn4 は整数定数式パラメータである．

SAC_PDQ は，対象優先度データキューが属する保護ドメインの囲みの中に記述なければならない．そうでない場合には，E_RSATR エラーとなる．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，SAC_PDQ，sac_pdq をサポートしない．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，SAC_PDQ，sac_pdq をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，SAC_PDQ のみをサポートする．

del_pdq　　優先度データキューの削除〔TD〕

【C 言語 API】

```
ER ercd = del_pdq(ID pdqid)
```

【パラメータ】

ID	pdqid	対象優先度データキューの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（pdqid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象優先度データキューが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象優先度データキューに対する管理操作が許可されていない）
E_OBJ	オブジェクト状態エラー（対象優先度データキューは静的 API で生成された）

【機能】

pdqid で指定した優先度データキュー（対象優先度データキュー）を削除する．具体的な振舞いは以下の通り．

対象優先度データキューの登録が解除され，その優先度データキューID が未使用の状態に戻される．また，対象優先度データキューの送信待ち行列と受信待ち行列につながれたタスクは，それぞれの待ち行列の先頭のタスクから順に待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_DLT エラーが返る．

優先度データキューの生成時に，優先度データキュー管理領域がカーネルによって確保された場合は，そのメモリ領域が解放される．

【補足説明】

送信待ち行列と受信待ち行列の両方にタスクがつながれていることはないため，別の待ち行列で待っていたタスクの間の待ち解除の順序は，規定する必要がない．

【使用上の注意】

del_pdq により複数のタスクが待ち解除される場合，サービスコールの処理時間およびカーネル内での割込み禁止時間が，待ち解除されるタスクの数に比例して長くなる．特に，多くのタスクが待ち解除される場合，カーネル内での割込み禁止時間が長くなるため，注意が必要である．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，del_pdq をサポートしない．ただし，動的生成機能拡張パッケージでは，del_pdq をサポートする．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，del_pdq をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，del_pdq をサポートしない．

---

snd_pdq　　優先度データキューへの送信〔T〕
psnd_pdq　　優先度データキューへの送信（ポーリング）〔T〕
ipsnd_pdq　　優先度データキューへの送信（ポーリング）〔I〕
tsnd_pdq　　優先度データキューへの送信（タイムアウト付き）〔T〕

【C 言語 API】

```
ER ercd = snd_pdq(ID pdqid, intptr_t data, PRI datapri)
ER ercd = psnd_pdq(ID pdqid, intptr_t data, PRI datapri)
ER ercd = ipsnd_pdq(ID pdqid, intptr_t data, PRI datapri)
ER ercd = tsnd_pdq(ID pdqid, intptr_t data, PRI datapri, TMO tmout)
```

【パラメータ】

ID	pdqid	対象データキューの ID 番号
intptr_t	data	送信データ
PRI	datapri	送信データの優先度
TMO	tmout	タイムアウト時間（tsnd_pdq の場合）

【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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し：ipsnd_pdq を除く，タスクコンテキストからの呼出し：ipsnd_pdq の場合，CPU ロック状態からの呼出し，ディスパッチ保留状態からの呼出し：snd_pdq と tsnd_pdq の場合）
E_NOSPT	未サポート機能（制約タスクからの呼出し：snd_pdq と tsnd_pdq の場合）
E_ID	不正 ID 番号（pdqid が不正）
E_PAR	パラメータエラー（datapri が不正，tmout が不正：tsnd_pdq のみ）
E_NOEXS〔D〕	オブジェクト未登録（対象優先度データキューが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象優先度データキューに対する通常操作 1 が許可されていない：ipsnd_pdq を除く）
E_TMOUT	ポーリング失敗またはタイムアウト（snd_pdq を除く）
E_RLWAI	待ち禁止状態または待ち状態の強制解除（snd_pdq と tsnd_pdq の場合）
E_DLT	待ちオブジェクトの削除または再初期化（snd_pdq と tsnd_pdq の場合）

【機能】

pdqid で指定した優先度データキュー(対象優先度データキュー)に, data で指定したデータを, datapri で指定した優先度で送信する．具体的な振舞いは以下の通り．

対象優先度データキューの受信待ち行列にタスクが存在する場合には，受信待ち行列の先頭のタスクが，data で指定したデータを受信し，待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_OK が返る．

対象優先度データキューの受信待ち行列にタスクが存在せず，優先度データキュー管理領域にデータを格納するスペースがある場合には，data で指定したデータが，datapri で指定したデータの優先度順で優先度データキュー管理領域に格納される．

対象優先度データキューの受信待ち行列にタスクが存在せず，優先度データキュー管理領域にデータを格納するスペースがない場合には，自タスクは優先度データキューへの送信待ち状態となり，対象優先度データキューの送信待ち行列につながれる．

datapri は，TMIN_DPRI 以上で，対象データキューに送信できるデータ優先度の最大値以下でなければならない．

rcv_pdq	優先度データキューからの受信〔T〕
prcv_pdq	優先度データキューからの受信（ポーリング）〔T〕
trcv_pdq	優先度データキューからの受信（タイムアウト付き）〔T〕

【C 言語 API】

```
ER ercd = rcv_pdq(ID pdqid, intptr_t *p_data, PRI *p_datapri)
ER ercd = prcv_pdq(ID pdqid, intptr_t *p_data, PRI *p_datapri)
ER ercd = trcv_pdq(ID pdqid, intptr_t *p_data, PRI *p_datapri, TMO tmout)
```

【パラメータ】

ID	pdqid	対象優先度データキューの ID 番号
intptr_t *	p_data	受信データを入れるメモリ領域へのポインタ
PRI *	p_datapri	受信データの優先度を入れるメモリ領域へのポインタ
TMO	tmout	タイムアウト時間（trcv_pdq の場合）

【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
intptr_t	data	受信データ
PRI	datapri	受信データの優先度

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し，ディスパッチ保留状態からの呼出し：prcv_pdq を除く）
E_NOSPT	未サポート機能（制約タスクからの呼出し：prcv_pdq を除く）
E_ID	不正 ID 番号（pdqid が不正）
E_PAR	パラメータエラー（tmout が不正：trcv_pdq の場合）
E_NOEXS〔D〕	オブジェクト未登録（対象優先度データキューが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象優先度データキューに対する通常操作 2 が許可されていない）
E_MACV〔P〕	メモリアクセス違反（p_data または p_datapri が指すメモリ領域への書込みアクセスが許可されていない）
E_TMOUT	ポーリング失敗またはタイムアウト（rcv_pdq を除く）
E_RLWAI	待ち禁止状態または待ち状態の強制解除（prcv_pdq を除く）
E_DLT	待ちオブジェクトの削除または再初期化（prcv_pdq を除く）

【機能】

pdqid で指定した優先度データキュー（対象優先度データキュー）からデータを受信する．受信したデータは p_data で指定したメモリ領域に，その優先度は p_datapri で指定したメモリ領域に返される．具体的な振舞いは以下の通り．

対象優先度データキューの優先度データキュー管理領域にデータが格納されている場合には，優先度データキュー管理領域の先頭に格納されたデータが取り出され，p_data で指定したメモリ領域に返される．また，その優先度が p_datapri で指定したメモリ領域に返される．さらに，送信待ち行列にタスクが存在する場合には，送信待ち行列の先頭のタスクの送信データが，データの優先度順で優先度データキュー管理領域に格納され，そのタスクは待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_OK が返る．

対象優先度データキューの優先度データキュー管理領域にデータが格納されておらず，送信待ち行列にタスクが存在する場合には，送信待ち行列の先頭のタスクの送信データが，p_data で指定したメモリ領域に返される．また，その優先度が p_datapri で指定したメモリ領域に返される．送信待ち行列の先頭のタスクは，待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_OK が返る．

対象優先度データキューの優先度データキュー管理領域にデータが格納されておらず，送信待ち行列にタスクが存在しない場合には，自タスクは優先度データキューからの受信待ち状態となり，対象優先度データキューの受信待ち行列につながれる．

ini_pdq　　優先度データキューの再初期化〔T〕

【C 言語 API】

```
ER ercd = ini_pdq(ID pdqid)
```

【パラメータ】

ID	pdqid	対象優先度データキューの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（pdqid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象優先度データキューが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象優先度データキューに対する管理操作が許可されていない）

【機能】

pdqid で指定した優先度データキュー（対象優先度データキュー）を再初期化する．具体的な振舞いは以下の通り．

対象優先度データキューの優先度データキュー管理領域は，格納されているデータがない状態に初期化される．また，対象優先度データキューの送信待ち行列と受信待ち行列につながれたタスクは，それぞれの待ち行列の先頭のタスクから順に待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_DLT エラーが返る．

【補足説明】

送信待ち行列と受信待ち行列の両方にタスクがつながれていることはないため別の待ち行列で待っていたタスクの間の待ち解除の順序は，規定する必要がない．

【使用上の注意】

ini_pdq により複数のタスクが待ち解除される場合，サービスコールの処理時間およびカーネル内での割込み禁止時間が，待ち解除されるタスクの数に比例して長くなる．特に，多くのタスクが待ち解除される場合，カーネル内での割込み禁止時間が長くなるため，注意が必要である．

優先度データキューを再初期化した場合に，アプリケーションとの整合性を保つのは，アプリケーションの責任である．

ref_pdq　　優先度データキューの状態参照〔T〕

【C 言語 API】

```
ER ercd = ref_pdq(ID pdqid, T_RPDQ *pk_rpdq)
```

【パラメータ】

ID	pdqid	対象優先度データキューの ID 番号
T_RPDQ *	pk_rpdq	優先度データキューの現在状態を入れる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

＊優先度データキューの現在状態（パケットの内容）

ID	stskid	優先度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	優先度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_t	spdqcnt	優先度データキュー管理領域に格納されているデータの数

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（pdqid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象優先度データキューが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象優先度データキューに対する参照操作が許可されていない）

【機能】

pdqid で指定した優先度データキュー（対象優先度データキュー）の現在状態を参照する．参照した現在状態は，pk_rpdq で指定したパケットに返される．

対象優先度データキューの送信待ち行列にタスクが存在しない場合，stskid には TSK_NONE（＝0）が返る．また，受信待ち行列にタスクが存在しない場合，rtskid には TSK_NONE（＝0）が返る．

【使用上の注意】

ref_pdq はデバッグ時向けの機能であり，その他の目的に使用することは推奨しない．これは，ref_pdq を呼び出し，対象優先度データキューの現在状態を参照した直後に割込みが発生した場合，ref_pdq から戻ってきた時には対象優先度データキューの状態が変化している可能性があるためである．

4.4.5. メールボックス

メールボックスは，共有メモリ上に置いたメッセージを，FIFO 順またはメッセージの優先度順で送受

信するための同期・通信オブジェクトである．メールボックスは，メールボックス ID と呼ぶ ID 番号によって識別する．

各メールボックスが持つ情報は次の通り．

- メールボックス属性
- メッセージキュー
- 待ち行列（メールボックスからの受信待ち状態のタスクのキュー）
- 送信できるメッセージ優先度の最大値
- 優先度別のメッセージキューヘッダ領域
- 属するクラス（マルチプロセッサ対応カーネルの場合）

メッセージキューは，メールボックスに送信されたメッセージを，FIFO 順またはメッセージの優先度順につないでおくためのキューである．

待ち行列は，メールボックスからメッセージが受信できるまで待っている状態（メールボックスからの受信待ち状態）のタスクが，メッセージを受信できる順序でつながれているキューである．

メールボックス属性には，次の属性を指定することができる．

TA_TPR	0x01U	待ち行列をタスクの優先度順にする
TA_MPRI	0x02U	メッセージキューをメッセージの優先度順にする

TA_TPRI を指定しない場合，待ち行列は FIFO 順になる．TA_MPRI を指定しない場合，メッセージキューは FIFO 順になる．

優先度別のメッセージキューヘッダ領域は，TA_MPRI 属性のメールボックスに対して，メッセージキューを優先度別に設ける場合に使用する領域である．

カーネルは，メールボックスに送信されたメッセージをメッセージキューにつなぐために，メッセージの先頭のメモリ領域を使用する．そのためアプリケーションは，メールボックスに送信するメッセージの先頭に，カーネルが利用するためのメッセージヘッダを置かなければならない．メッセージヘッダのデータ型として，メールボックス属性に TA_MPRI が指定されているか否かにより，以下のいずれかを用いる．

T_MSG	TA_MPRI 属性でないメールボックス用のメッセージヘッダ
T_MSG_PRI	TA_MPRI 属性のメールボックス用のメッセージヘッダ

メッセージヘッダの領域は，メッセージがメッセージキューにつながれている間（すなわち，メールボックスに送信してから受信するまでの間），カーネルによって使用される．そのため，メッセージキューにつながれているメッセージのメッセージヘッダの領域をアプリケーションが書き換えた場合や，メッセージキューにつながれ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動作は保証されない．

TA_MPRI 属性のメールボックスにメッセージを送信する場合，アプリケーションは，メッセージの優先度を，T_MSG_PRI 型のメッセージヘッダ中の msgpri フィールドに設定する．

保護機能対応カーネルでは，メールボックス機能はサポートしない．

メールボックス機能に関連するカーネル構成マクロは次の通り．

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
TNUM_MBXID	登録できるメールボックスの数（動的生成対応でないカーネルでは，静的 API によって登録されたメールボックスの数に一致）

【補足説明】
TOPPERS 新世代カーネルの現時点の実装では，優先度別のメッセージキューヘッダ領域は用いていない．

【使用上の注意】
メールボックス機能は，μITRON4.0 仕様との互換性のために残した機能であり，保護機能対応カーネルではサポートしないため，使用することは推奨しない．メールボックス機能は，ほとんどの場合に，データキュー機能または優先度データキュー機能を用いて，メッセージを置いたメモリ領域へのポインタを送受信する方法で置き換えることができる．

【TOPPERS/ASP カーネルにおける規定】
ASP カーネルでは，メールボックス機能をサポートする．メッセージ優先度の最大値（TMAX_MPRI）は 16 に固定されている．ただし，タスク優先度拡張パッケージでは，TMAX_MPRI を 256 に拡張する．

【TOPPERS/FMP カーネルにおける規定】
FMP カーネルでは，メールボックス機能をサポートする．メッセージ優先度の最大値（TMAX_MPRI）は 16 に固定されている．

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，メールボックス機能をサポートしない．

【μITRON4.0 仕様との関係】

TNUM_MBXID は，μITRON4.0 仕様に規定されていないカーネル構成マクロである．

CRE_MBX　メールボックスの生成〔Sp〕
acre_mbx　メールボックスの生成〔TpD〕

【静的 API】

```
CRE_MBX(ID mbxid, {ATR mbxatr, PRI maxmpri, void *mprihd })
```

【C 言語 API】

```
ER_ID mbxid = acre_mbx(const T_CMBX *pk_cmbx)
```

【パラメータ】

ID	mbxid	生成するメールボックスの ID 番号（CRE_MBX の場合）
T_CPDQ *	pk_cmbx	メールボックスの生成情報を入れたパケットへのポインタ（静的 API を除く）

＊メールボックスの生成情報（パケットの内容）

ATR	mbxatr	メールボックス属性
PRI	maxmpri	優先度メールボックスに送信できるメッセージ優先度の最大値
void *	mprihd	優先度別のメッセージキューヘッダ領域の先頭番地

【リターンパラメータ】

ER_ID	mbxid	生成され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_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_RSATR	予約属性（mbxatr が不正または使用できない，属するクラスが不正）
E_NOSPT	未サポート機能（mprihd がサポートされていない値）
E_PAR	パラメータエラー（mprihd が不正）
E_NOID〔sD〕	ID 番号不足（割り付けられるメールボックス ID がない）
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が確保できない）
E_OBJ	オブジェクト状態エラー（mbxid で指定したメールボックスが登録済み：CRE_MBX の場合）

【機能】

各パラメータで指定したメールボックス生成情報に従って，メールボックスを生成する．メッセージキューはつながれているメッセージがない状態に初期化され，mprihd と maxmpri から優先度別のメッセージキューヘッダ領域が設定される．また，待ち行列は空の状態に初期化される．

静的 API においては，mbxid はオブジェクト識別名，maxmpri は整数定数式パラメータ，mprihd は一般定数式パラメータである．コンフィギュレータは，静的 API のメモリ不足（E_NOMEM）エラーを検出することができない．

mprihd を NULL とした場合，maxmpri の指定に合致したサイズの優先度別のメッセージキューヘッダ領域を，コンフィギュレータまたはカーネルが確保する．

maxmpri は，TMIN_MPRI 以上，TMAX_MPRI 以下でなければならない．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，CRE_MBX のみをサポートする．また，優先度別のメッセージキューヘッダ領域は使用しておらず, mprihd には NULL のみを指定することができる. NULL 以外を指定した場合には，E_NOSPT エラーとなる．ただし，動的生成機能拡張パッケージでは，acre_mbx もサポートする．acre_mbx に対しても，mprihd には NULL のみを指定することができる．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，CRE_MBX のみをサポートする．また，優先度別のメッセージキューヘッダ領域は使用しておらず，mprihd には NULL のみを渡すことができる．NULL 以外を指定した場合には，E_NOSPT エラーとなる．

AID_MBX　　割付け可能なメールボックス ID の数の指定〔SpD〕

【静的 API】

```
AID_MBX(uint_t nombx)
```

【パラメータ】

uint_t	nombx	割付け可能なメールボックス ID の数

【エラーコード】

E_RSATR	予約属性（属する保護ドメインまたはクラスが不正）

【機能】

nombx で指定した数のメールボックス ID を，メールボックスを生成するサービスコールによって割付け可能なメールボックス ID として確保する．

nombx は整数定数式パラメータである．

del_mbx　　メールボックスの削除〔TDp〕

【C 言語 API】

```
ER ercd = del_mbx(ID mbxid)
```

【パラメータ】

ID	mbxid	対象メールボックスの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（mbxid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象メールボックスが未登録）
E_OBJ	オブジェクト状態エラー（対象メールボックスは静的 API で生成された）

【機能】

mbxid で指定したメールボックス（対象メールボックス）を削除する．具体的な振舞いは以下の通り．

対象メールボックスの登録が解除され，そのメールボックス ID が未使用の状態に戻される．また，対象メールボックスの待ち行列につながれたタスクは，待ち行列の先頭のタスクから順に待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_DLT エラーが返る．

メールボックスの生成時に，優先度別のメッセージキューヘッダ領域がカーネルによって確保された場合は，そのメモリ領域が解放される．

【使用上の注意】

del_mbx により複数のタスクが待ち解除される場合，サービスコールの処理時間およびカーネル内での割込み禁止時間が，待ち解除されるタスクの数に比例して長くなる．特に，多くのタスクが待ち解除される場合，カーネル内での割込み禁止時間が長くなるため，注意が必要である．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，del_mbx をサポートしない．ただし，動的生成機能拡張パッケージでは，del_mbx をサポートする．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，del_mbx をサポートしない．

snd_mbx　　メールボックスへの送信〔Tp〕

【C 言語 API】

```
ER ercd = snd_mbx(ID mbxid, T_MSG *pk_msg)
```

【パラメータ】

ID	mbxid	対象メールボックスの ID 番号
T_MSG *	pk_msg	送信メッセージの先頭番地

【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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出
E_ID	不正 ID 番号（mbxid が不正）
E_PAR	パラメータエラー（メッセージヘッダ中の msgpri が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象メールボックスが未登録）

【機能】

mbxid で指定したメールボックス（対象メールボックス）に，pk_msg で指定したメッセージを送信する．具体的な振舞いは以下の通り．

対象メールボックスの待ち行列にタスクが存在する場合には，待ち行列の先頭のタスクが，pk_msg で指定したメッセージを受信し，待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_OK が返る．

対象メールボックスの待ち行列にタスクが存在しない場合には，pk_msg で指定したメッセージが，メールボックス属性の TA_MPRI 指定の有無によって指定される順序で，メッセージキューにつなぐ．

対象メールボックスが TA_MPRI 属性である場合には，pk_msg で指定したメッセージの先頭のメッセージヘッダ中の msgpri フィールドの値が，TMIN_MPRI 以上で，
対象メールボックスに送信できるメッセージ優先度の最大値以下でなければならない．

---

rcv_mbx　メールボックスからの受信〔Tp〕
prcv_mbx　メールボックスからの受信（ポーリング）〔Tp〕
trcv_mbx　メールボックスからの受信（タイムアウト付き）〔Tp〕

【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	対象メールボックスの ID 番号
T_MSG **	ppk_msg	受信メッセージの先頭番地を入れるメモリ領域へのポインタ
TMO	tmout	タイムアウト時間（trcv_mbx の場合）

【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 *	ppk_msg	受信メッセージの先頭番地

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し，ディスパッチ保留状態からの呼出し：prcv_mbx を除く）
E_NOSPT	未サポート機能（制約タスクからの呼出し：prcv_mbx を除く）
E_ID	不正 ID 番号（mbxid が不正）
E_PAR	パラメータエラー（tmout が不正：trcv_mbx の場合）
E_NOEXS〔D〕	オブジェクト未登録（対象メールボックスが未登録）
E_TMOUT	ポーリング失敗またはタイムアウト（rcv_mbx を除く）
E_RLWAI	待ち禁止状態または待ち状態の強制解除（prcv_mbx を除く）
E_DLT	待ちオブジェクトの削除または再初期化（prcv_mbx を除く）

【機能】

mbxid で指定したメールボックス（対象メールボックス）からメッセージを受信する．受信したメッセージの先頭番地は，ppk_msg で指定したメモリ領域に返される．具体的な振舞いは以下の通り．

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つながれている場合には，メッセージキューの先頭につながれたメッセージが取り出され，ppk_msg で指定したメモリ領域に返される．

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つながれていない場合には，自タスクはメールボックスからの受信待ち状態となり，対象メールボックスの待ち行列につながれる．

ini_mbx　　メールボックスの再初期化〔Tp〕

【C 言語 API】

```
ER ercd = ini_mbx(ID mbxid)
```

【パラメータ】

ID	mbxid	対象メールボックスの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（mbxid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象メールボックスが未登録）

【機能】

mbxid で指定したメールボックス（対象メールボックス）を再初期化する．具体的な振舞いは以下の通り．

対象メールボックスのメールボックス管理領域は，メッセージキューはつながれているメッセージがない状態に初期化される．また，対象メールボックスの待ち行列につながれたタスクは，待ち行列の先頭のタスクから順に待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_DLT エラーが返る．

【使用上の注意】

ini_mbx により複数のタスクが待ち解除される場合，サービスコールの処理時間およびカーネル内での割込み禁止時間が，待ち解除されるタスクの数に比例して長くなる．特に，多くのタスクが待ち解除される場合，カーネル内での割込み禁止時間が長くなるため，注意が必要である．

メールボックスを再初期化した場合に，アプリケーションとの整合性を保つのは，アプリケーションの責任である．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様に定義されていないサービスコールである．

ref_mbx　　メールボックスの状態参照〔Tp〕

【C 言語 API】

```
ER ercd = ref_mbx(ID mbxid, T_RMBX *pk_rmbx)
```

【パラメータ】

ID	mbxid	対象メールボックスの ID 番号
T_RMBX *	pk_rmbx	メールボックスの現在状態を入れる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

＊メールボックスの現在状態（パケットの内容）

ID	wtskid	データキューの送信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につながれたメッセージの先頭番地

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（mbxid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象メールボックスが未登録）

【機能】

mbxid で指定したメールボックス（対象メールボックス）の現在状態を参照する．参照した現在状態は，pk_rmbx で指定したパケットに返される．

対象メールボックスの待ち行列にタスクが存在しない場合，wtskid には TSK_NONE（＝0）が返る．また，メッセージキューにメッセージがつながれていない場合，pk_msg には NULL が返る．

【使用上の注意】

ref_mbx はデバッグ時向けの機能であり，その他の目的に使用することは推奨しない．これは，ref_mbx を呼び出し，対象メールボックスの現在状態を参照した直後に割込みが発生した場合，ref_mbx から戻ってきた時には対象メールボックスの状態が変化している可能性があるためである．

4.4.6. ミューテックス

ミューテックスは，タスク間の排他制御を行うための同期・通信オブジェクトである．タスクは，排他制御区間に入る時にミューテックスをロックし，排他制御区間を出る時にロック解除する．ミューテックスは，ミューテックス ID と呼ぶ ID 番号によって識別する．

ミューテックスは，排他制御に伴う優先度逆転の時間を最小限に抑えるための優先度上限プロトコル（priority ceiling protocol）をサポートする．ミューテックス属性により優先度上限ミューテックスであると指定することで，そのミューテックスの操作時に，優先度上限プロトコルに従った現在優先度の制御が行われる．

各ミューテックスが持つ情報は次の通り．

- ミューテックス属性
- ロック状態（ロックされている状態とロック解除されている状態）
- ミューテックスをロックしているタスク
- 待ち行列（ミューテックスのロック待ち状態のタスクのキュー）
- 上限優先度（優先度上限ミューテックスの場合）
- アクセス許可ベクタ（保護機能対応カーネルの場合）
- 属する保護ドメイン（保護機能対応カーネルの場合）
- 属するクラス（マルチプロセッサ対応カーネルの場合）

待ち行列は，ミューテックスをロックできるまで待っている状態（ミューテックスのロック待ち状態）のタスクが，ミューテックスをロックできる順序でつながれているキューである．

上限優先度は，優先度上限ミューテックスに対してのみ有効で，ミューテックスの生成時に，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ベース優先度の中で最も高い優先度（または，それより高い優先度）に設定する．

ミューテックス属性には，次の属性を指定することができる．

TA_TPRI	0x01U	送信待ち行列をタスクの優先度順にする
TA_CEILING	0x03U	優先度上限ミューテックスとする．待ち行列をタスクの優先度順にする

TA_TPRI，TA_CEILING のいずれも指定しない場合，待ち行列は FIFO 順になる．

ミューテックス機能に関連して，各タスクが持つ情報は次の通り．

- ロックしているミューテックスのリスト

ロックしているミューテックスのリストは，タスクの起動時に空に初期化される．

タスクの現在優先度は，そのタスクのベース優先度と，そのタスクがロックしている優先度上限ミューテックスの優先度上限の中で，最も高い優先度に設定される．

ミューテックス機能によりタスクの現在優先度が変化する場合には，次の処理が行われる．現在優先度を変化させるサービスコールの前後とも，当該タスクが実行できる状態である場合には，同じ優先度のタスクの中で最高優先順位となる．そのサービスコールにより，当該タスクが実行できる状態に遷移する場合には，同じ優先度のタスクの中で最低優先順位となる．そのサービスコールの後で，当該タス

クが待ち状態で，タスクの優先度順の待ち行列につながれている場合には，当該タスクの変更後の現在優先度に従って，その待ち行列中での順序が変更される．待ち行列中に同じ現在優先度のタスクがある場合には，当該タスクの順序はそれらの中で最後になる．

ミューテックス機能に関連して，タスクの終了時に行うべき処理として，タスクがロックしているミューテックスのロック解除がある．タスクの終了時にロックしているミューテックスが残っている場合，それらのミューテックスは，ロックしたのと逆の順序でロック解除される．

ミューテックス機能に関連するカーネル構成マクロは次の通り．

TNUM_MTXID	登録できるミューテックスの数（動的生成対応でないカーネルでは，静的 API によって登録されたミューテックスの数に一致）

【使用上の注意】

優先度上限プロトコルには，(a) 優先度の低いタスクの排他制御区間に最大 1 回しかブロックされない，(b) タスクの実行が開始された以降は優先度の低いタスクにブロックされないという利点があるが，これは，タスク間の同期に優先度上限ミューテックスのみを用い，他の方法でタスクのスケジューリングに関与しない場合に得られる利点である．

これらの利点を得るためには，タスクの優先順位の回転やディスパッチの禁止を行ってはならないことに加えて，優先度上限ミューテックスをロックしたタスクを待ち状態にしてはならない．特に，優先度上限ミューテックスに対して，タスクがロック待ち状態になる状況に注意が必要である（優先度上限プロトコルでは，タスクがミューテックスのロック待ち状態になることはない）．

例えば，着目するタスク A と，タスク A よりベース優先度の低いタスク B とタスク C，タスク A よりも高い上限優先度を持った優先度上限ミューテックスがある場合を考える．タスク A がミューテックスをロックし，タスク B とタスク C がミューテックスを待っている状況で，タスク A がミューテックスをロック解除すると，タスク B がミューテックスをロックして優先度が上がり，タスク B に切り換わる．

さらにタスク B がミューテックスをロック解除すると，タスク C がミューテックスをロックして優先度が上がり，タスク C に切り換わる．タスク A が実行されるのは，タスク C がミューテックスをロック解除した後である．この例では，タスク A が実行開始後に，タスク B とタスク C の排他制御区間にブロックされることになる．

優先度上限ミューテックスに対してタスクがロック待ち状態になる状況を回避するためには，優先度上限ミューテックスをロックする場合に，待ち状態にならない ploc_mtx を用いるのが安全である．

【補足説明】

この仕様で優先度上限プロトコルと呼んでいる方式は，オリジナルの priority ceiling protocol とは異

なるものである．この仕様の方式は，OSEK/VDX OS 仕様でも priority ceiling protocol と呼ばれているが，学術論文や他の OS では，immediate ceiling priority protocol，priority protection protocol，priority ceiling emulation，highest locker protocol などと呼ばれている．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，ミューテックス機能をサポートしない．ただし，ミューテックス機能拡張パッケージを用いると，ミューテックス機能を追加することができる．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，ミューテックス機能をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，ミューテックス機能をサポートする．

【未決定事項】

マルチプロセッサにおいては，タスク間の同期に優先度上限ミューテックスのみを用い，他の方法でタスクのスケジューリングに関与しない場合でも，優先度上限ミューテックスに対してタスクがロック待ち状態になる．マルチプロセッサ対応カーネルにおける優先度上限ミューテックスの扱いについては，今後の課題である．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様の厳密な優先度制御規則を採用し，簡略化した優先度制御規則はサポートしていない．また，μITRON4.0 仕様でサポートしている優先度継承プロトコル（priority inheritance protocol）は，現時点ではサポートしていない．

ミューテックス機能によりタスクの現在優先度が変化する場合の振舞いは，μITRON4.0 仕様では実装依存となっているが，この仕様では規定している．

TNUM_MTXID は，μITRON4.0 仕様に規定されていないカーネル構成マクロである．

CRE_MTX　ミューテックスの生成〔S〕
acre_mtx　ミューテックスの生成〔TD〕

【静的 API】

```
CRE_MTX(ID mtxid, {ATR mtxatr, PRI ceilpri })
```

【C 言語 API】

ER_ID mtxid = acre_mtx(const T_CMTX *pk_cmtx)

【パラメータ】

ID	mtxid	生成するミューテックスの ID 番号（CRE_MTX の場合）
T_CMTX *	pk_cmtx	ミューテックスの生成情報を入れたパケットへのポインタ（静的 API を除く）

＊ミューテックスの生成情報（パケットの内容）

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

【リターンパラメータ】

ER_ID	mtxid	生成され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_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_RSATR	予約属性（mtxatr が不正または使用できない，属する保護ドメインかクラスが不正）
E_PAR	パラメータエラー（ceilpri が不正）
E_OACV〔sP〕	オブジェクトアクセス違反（システム状態に対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（pk_cmtx が指すメモリ領域への読出しアクセスが許可されていない）
E_NOID〔sD〕	ID 番号不足（割り付けられるミューテックス ID がない）
E_OBJ	オブジェクト状態エラー（mtxid で指定したミューテックスが登録済み：CRE_MTX の場合）

【機能】

各パラメータで指定したミューテックス生成情報に従って，ミューテックスを生成する．生成されたミューテックスのロック状態はロックされていない状態に，待ち行列は空の状態に初期化される．

静的 API においては，mtxid はオブジェクト識別名，ceilpri は整数定数式パラメータである．優先度上限ミューテックス以外の場合には，ceilpri の指定を省略することができる．

優先度上限ミューテックスを生成する場合，ceilpri は，TMIN_TPRI 以上，TMAX_TPRI 以下でなければならない．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルのミューテックス機能拡張パッケージでは，CRE_MTX のみをサポートする．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，CRE_MTX のみをサポートする．

AID_MTX　　割付け可能なミューテックス ID の数の指定〔SD〕

【静的 API】

```
AID_MTX(uint_t nomtx)
```

【パラメータ】

uint_t	nomtx	割付け可能なミューテックス ID の数

【エラーコード】

E_RSATR	予約属性（属する保護ドメインまたはクラスが不正）

【機能】

nomtx で指定した数のミューテックス ID を，ミューテックスを生成するサービスコールによって割付け可能なミューテックス ID として確保する．

nomtx は整数定数式パラメータである．

SAC_MTX　　ミューテックスのアクセス許可ベクタの設定〔SP〕
sac_mtx　　ミューテックスのアクセス許可ベクタの設定〔TPD〕

【静的 API】

```
SAC_MTX(ID mtxid, {ACPTN acptn1, ACPTN acptn2
                   ACPTN acptn3, ACPTN acptn4})
```

【C 言語 API】

```
ER ercd = sac_mtx(ID mtxid, const ACVCT *p_acvct)
```

【パラメータ】

ID	mtxid	対象ミューテックスの ID 番号
ACVCT *	p_acvct	アクセス許可ベクタを入れたパケットへのポインタ（静的 API を除く）

＊アクセス許可ベクタ（パケットの内容）

ACPTN	acptn1	通常操作 1 のアクセス許可パターン
ACPTN	acptn2	通常操作 2 のアクセス許可パターン
ACPTN	acptn3	管理操作のアクセス許可パターン
ACPTN	acptn4	参照操作のアクセス許可パターン

【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_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_RSATR	予約属性（属する保護ドメインかクラスが不正：SAC_MTX の場合）
E_ID	不正 ID 番号（mtxid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象ミューテックスが未登録）
E_OACV〔sP〕	オブジェクトアクセス違反（対象ミューテックスに対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（p_acvct が指すメモリ領域への読出しアクセスが許可されていない）
E_OBJ	オブジェクト状態エラー（対象ミューテックスは静的 API で生成された：sac_mtx の場合，対象ミューテックスに対してアクセス許可ベクタが設定済み：SAC_MTX の場合）

【機能】

mtxid で指定したミューテックス（対象ミューテックス）のアクセス許可ベクタ（4 つのアクセス許可パターンの組）を，各パラメータで指定した値に設定する．

静的 API においては，mtxid はオブジェクト識別名，acptn1～acptn4 は整数定数式パラメータである．

SAC_MTX は，対象ミューテックスが属する保護ドメインの囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルのミューテックス機能拡張パッケージでは，SAC_MTX，sac_mtx をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，SAC_MTX のみをサポートする．

del_mtx　　ミューテックスの削除〔TD〕

【C 言語 API】

```
ER ercd = del_mtx(ID mtxid)
```

【パラメータ】

ID	mtxid	対象ミューテックスの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（mtxid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象ミューテックスが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象ミューテックスに対する管理操作が許可されていない
E_OBJ	オブジェクト状態エラー（対象ミューテックスは静的 API で生成された）

【機能】

mtxid で指定したミューテックス（対象ミューテックス）を削除する．具体的な振舞いは以下の通り．

対象ミューテックスの登録が解除され，そのミューテックス ID が未使用の状態に戻される．対象ミューテックスをロックしているタスクがある場合には，そのタスクがロックしているミューテックスのリストから対象ミューテックスが削除され，必要な場合にはそのタスクの現在優先度が変更される．また，対象ミューテックスの待ち行列につながれたタスクは，待ち行列の先頭のタスクから順に待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_DLT エラーが返る．

【使用上の注意】

対象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が通知されず，

そのミューテックスをロック解除する時点でエラーとなる．これが不都合な場合には，ミューテックスをロックした状態で，ミューテックスを削除すればよい．

del_mtx により複数のタスクが待ち解除される場合，サービスコールの処理時間およびカーネル内での割込み禁止時間が，待ち解除されるタスクの数に比例して長くなる．特に，多くのタスクが待ち解除される場合，カーネル内での割込み禁止時間が長くなるため，注意が必要である．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルのミューテックス機能拡張パッケージでは，del_mtx をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，del_mtx をサポートしない．

---

loc_mtx　　ミューテックスのロック〔T〕
ploc_mtx　　ミューテックスのロック（ポーリング）〔T〕
tloc_mtx　　ミューテックスのロック（タイムアウト付き）〔T〕

【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	対象ミューテックスの ID 番号
TMO	tmout	タイムアウト時間（twai_mtx の場合）

【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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し，ディスパッチ保留状態からの呼出し：pol_mtx を除く）
E_NOSPT	未サポート機能（制約タスクからの呼出し：pol_mtx を除く）
E_ID	不正 ID 番号（mtxid が不正）
E_PAR	パラメータエラー（tmout が不正：twai_mtx の場合）
E_NOEXS〔D〕	オブジェクト未登録（対象ミューテックスが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象ミューテックスに対する通常操作 1 が許可されていない）
E_ILUSE	サービスコール不正使用（対象ミューテックスを自タスクがロックしている，対象優先度上限ミューテックスの上限優先度より自タスクのベース優先度が高い）
E_TMOUT	ポーリング失敗またはタイムアウト（wai_mtx を除く）
E_RLWAI	待ち禁止状態または待ち状態の強制解除（pol_mtx を除く）
E_DLT	待ちオブジェクトの削除または再初期化（pol_mtx を除く）

【機能】

mtxid で指定したミューテックス（対象ミューテックス）をロックする．具体的な振舞いは以下の通り．

対象ミューテックスがロックされていない場合には，自タスクによってロックされている状態になる．自タスクがロックしているミューテックスのリストに対象ミューテックスが追加され，必要な場合には自タスクの現在優先度が変更される．

対象ミューテックスが自タスク以外のタスクによってロックされている場合には，自タスクはミューテックスのロック待ち状態となり，対象ミューテックスの待ち行列につながれる．

対象ミューテックスが自タスクによってロックされている場合には，E_ILUSE エラーとなる．また，対象ミューテックスが優先度上限ミューテックスで，その上限優先度より自タスクのベース優先度が高い場合にも，E_ILUSE エラーとなる．

unl_mtx　　ミューテックスのロック解除〔T〕

【C 言語 API】

```
ER ercd = unl_mtx(ID mtxid)
```

【パラメータ】

ID	mtxid	対象ミューテックスの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（mtxid が不正）
E_NOEXS〔D〕	オブジェクト未登録（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自タスクがロックしていない）

【機能】

mtxid で指定したミューテックス（対象ミューテックス）をロック解除する．具体的な振舞いは以下の通り．

まず，自タスクがロックしているミューテックスのリストから対象ミューテックスが削除され，必要な場合には自タスクの現在優先度が変更される．

対象ミューテックスの待ち行列にタスクが存在する場合には，待ち行列の先頭のタスクが待ち解除される．対象ミューテックスは，待ち解除されたタスクによってロックされている状態になる．待ち解除されたタスクがロックしているミューテックスのリストに対象ミューテックスが追加され，必要な場合にはそのタスクの現在優先度が変更される．待ち解除されたタスクには，待ち状態となったサービスコールから E_OK が返る．

待ち行列にタスクが存在しない場合には，対象ミューテックスはロックされていない状態になる．

対象ミューテックスが自タスクによってロックされていない場合には，E_ILUSE エラーとなる．

ini_mtx　　ミューテックスの再初期化〔T〕

【C 言語 API】

```
ER ercd = ini_mtx(ID mtxid)
```

【パラメータ】

ID	mtxid	対象ミューテックスの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（mtxid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象ミューテックスが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象ミューテックスに対する管理操作が許可されていない）

【機能】

mtxid で指定したミューテックス（対象ミューテックス）を再初期化する．具体的な振舞いは以下の通り．

対象ミューテックスのロック状態は，ロックされていない状態に初期化される．対象ミューテックスをロックしているタスクがある場合には，そのタスクがロックしているミューテックスのリストから対象ミューテックスが削除され，必要な場合にはそのタスクの現在優先度が変更される．また，対象ミューテックスの待ち行列につながれたタスクは，待ち行列の先頭のタスクから順に待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_DLT エラーが返る．

【使用上の注意】

対象ミューテックスをロックしているタスクには，ミューテックスが再初期化されたことが通知されず，そのミューテックスをロック解除する時点でエラーとなる．これが不都合な場合には，ミューテックスをロックした状態で，ミューテックスを再初期化すればよい．

ini_mtx により複数のタスクが待ち解除される場合，サービスコールの処理時間およびカーネル内での割込み禁止時間が，待ち解除されるタスクの数に比例して長くなる．特に，多くのタスクが待ち解除される場合，カーネル内での割込み禁止時間が長くなるため，注意が必要である．

ミューテックスを再初期化した場合に，アプリケーションとの整合性を保つのは，アプリケーションの責任である．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様に定義されていないサービスコールである．

ref_mtx　　ミューテックスの状態参照〔T〕

【C 言語 API】

```
ER ercd = ref_mtx(ID mtxid, T_RMTX *pk_rmtx)
```

【パラメータ】

ID	mtxid	対象ミューテックスの ID 番号
T_RMTX *	pk_rmtx	ミューテックスの現在状態を入れる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

＊ミューテックスの現在状態（パケットの内容）

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	pk_ wtskid	ミューテックスの待ち行列の先頭のタスクの ID 番号

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（mtxid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象ミューテックスが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象ミューテックスに対する参照操作が許可されていない）
E_MACV〔P〕	メモリアクセス違反（pk_rmtx が指すメモリ領域への書込みアクセスが許可されていない）

【機能】

mtxid で指定したミューテックス（対象ミューテックス）の現在状態を参照する．
参照した現在状態は，pk_rmtx で指定したパケットに返される．

対象ミューテックスがロックされていない場合，htskid には TSK_NONE（＝0）が返る．

対象ミューテックスの待ち行列にタスクが存在しない場合，wtskid には TSK_NONE（＝0）が返る．

【使用上の注意】

ref_mtx はデバッグ時向けの機能であり，その他の目的に使用することは推奨しない．これは，ref_mtx

を呼び出し，対象ミューテックスの現在状態を参照した直後に割込みが発生した場合，ref_mtx から戻ってきた時には対象ミューテックスの状態が変化している可能性があるためである．

4.4.7. メッセージバッファ

☆未完成

4.4.8. スピンロック

スピンロックは，マルチプロセッサ対応カーネルにおいて，割込みのマスクとプロセッサ間ロックの取得により，排他制御を行うための同期・通信オブジェクトである．スピンロックは，スピンロック ID と呼ぶ ID 番号によって識別する．

プロセッサ間ロックを取得している間は，CPU ロック状態にすることですべてのカーネル管理の割込みがマスクされ，ディスパッチが保留される．ロックが他のプロセッサに取得されている場合には，ロックが取得できるまでループによって待つ．ロックの取得を待つ間は，CPU ロック解除状態であり，割込みはマスクされない．プロセッサ間ロックを取得し，CPU ロック状態に遷移することを，スピンロックを取得するという．また，プロセッサ間ロックを返却し，CPU ロック状態を解除することを，スピンロックを返却するという．

タスクが取得したスピンロックを返却せずに終了した場合や，タスク例外処理ルーチン，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が取得したスピンロックを返却せずにリターンした場合には，カーネルによってスピンロックが返却される．また，スピンロックを取得していない状態で発生した CPU 例外によって呼び出された CPU 例外ハンドラが，取得したスピンロックを返却せずにリターンした場合には，カーネルによってスピンロックが返却される．一方，拡張サービスコールからのリターンでは，スピンロックは返却されない．

各スピンロックが持つ情報は次の通り．

- スピンロック属性
- ロック状態（取得されている状態と取得されていない状態）
- アクセス許可ベクタ（保護機能対応カーネルの場合）
- 属する保護ドメイン（保護機能対応カーネルの場合）
- 属するクラス

スピンロック属性に指定できる属性はない．そのためスピンロック属性には，TA_NULL を指定しなければならない．

スピンロック機能に関連するカーネル構成マクロは次の通り．

TNUM_SPNID	登録できるスピンロックの数（動的生成対応でないカーネルでは，静的 API によって登録されたスピンロックの数に一致）

【補足説明】

CPU ロック状態では，スピンロックを取得するサービスコールを呼び出すことができないため，スピンロックを取得しているプロセッサが，さらにスピンロックを取得することはできない．そのため，1 つの処理単位が，複数のスピンロックを取得した状態になることはできない．

スピンロックを取得した状態で CPU 例外が発生した場合，起動される CPU 例外ハンドラはカーネル管理外の CPU 例外ハンドラであり（xsns_dpn，xsns_xpn とも true を返す），CPU 例外ハンドラ中で iunl_spn を呼び出してスピンロックを返却しようとした場合の動作は保証されない．保証されないにも関わらず iunl_spn を呼び出した場合には，CPU 例外ハンドラからのリターン時に元の状態に戻らない．これは，CPU ロック状態の扱いと一貫していないため，注意が必要である．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，スピンロック機能をサポートしない．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，スピンロック機能をサポートする．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，スピンロック機能をサポートしない．

【μITRON4.0 仕様との関係】

スピンロック機能は，μITRON4.0 仕様に定義されていない機能である．

CRE_SPN　スピンロックの生成〔SM〕
acre_spn　スピンロックの生成〔TMD〕

【静的 API】

```
CRE_SPN(ID spnid, {ATR spnatr})
```

【C 言語 API】

```
ER_ID spnid = acre_spn(const T_CSPN *pk_cspn)
```

【パラメータ】

ID	spnid	生成するスピンロックの ID 番号（CRE_MBX の場合）
T_CSPN *	pk_cspn	スピンロックの生成情報を入れたパケットへのポインタ（静的 API を除く）

＊スピンロックの生成情報（パケットの内容）

ATR	spnatr	スピンロック属性

【リターンパラメータ】

ER_ID	spnid	生成されたスピンロックの ID 番号（正の値）またはエラーコード

【エラーコード】

E_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_RSATR	予約属性（spnatr が不正または使用できない，属する保護ドメインかクラスが不正）
E_OACV〔sP〕	オブジェクトアクセス違反（システム状態に対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（pk_cspn が指すメモリ領域への読出しアクセスが許可されていない）
E_NOID〔sD〕	ID 番号不足（割り付けられるスピンロック ID がない）
E_NORES	資源不足（スピンロックを実現するためのハードウェア資源がない：CRE_SPN の場合）
E_OBJ	オブジェクト状態エラー（spnid で指定したスピンロックが登録済み：CRE_SPN の場合）

【機能】

各パラメータで指定したスピンロック生成情報に従って，スピンロックを生成する．生成されたスピンロックのロック状態は，取得されていない状態に初期化される．

静的 API においては，spnid はオブジェクト識別名である．

スピンロックをハードウェアによって実現している場合には，ターゲット定義で，生成できるスピンロックの数に上限がある．この上限を超えてスピンロックを生成しようとした場合には，E_NORES エラーとなる．

【補足説明】

スピンロックを動的に生成する場合に，生成できるスピンロックの数の上限は AID_SPN によってチェックされるため，acre_spn で E_NORES エラーが返ることはない．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，CRE_SPN のみをサポートする

．

AID_SPN　　割付け可能なスピンロック ID の数の指定〔SMD〕

【静的 API】

```
AID_SPN(uint_t nospn)
```

【パラメータ】

uint_t	nospn	割付け可能なスピンロック ID の数

【エラーコード】

E_RSATR	予約属性（属する保護ドメインまたはクラスが不正）

【機能】

nospn で指定した数のスピンロック ID を，スピンロックを生成するサービスコールによって割付け可能なスピンロック ID として確保する．

nospn は整数定数式パラメータである．

SAC_SPN　　スピンロックのアクセス許可ベクタの設定〔SPM〕
sac_spn　　スピンロックのアクセス許可ベクタの設定〔TPMD〕

【静的 API】

```
SAC_SPN(ID spnid, {ACPTN acptn1, ACPTN acptn2,
                   ACPTN acptn3, ACPTN acptn4})
```

【C 言語 API】

```
ER ercd = sac_spn(ID spnid, const ACVCT *p_acvct)
```

【パラメータ】

ID	spnid	対象スピンロックの ID 番号
ACVCT *	p_acvct	アクセス許可ベクタを入れたパケットへのポインタ（静的 API を除く）

＊アクセス許可ベクタ（パケットの内容）

ACPTN	acptn1	通常操作 1 のアクセス許可パターン
ACPTN	acptn2	通常操作 2 のアクセス許可パターン
ACPTN	acptn3	管理操作のアクセス許可パターン
ACPTN	acptn4	参照操作のアクセス許可パターン

【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_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（spnid が不正）
E_RSATR	予約属性（属する保護ドメインかクラスが不正：SAC_SPN の場合）
E_NOEXS〔D〕	オブジェクト未登録（対象スピンロックが未登録）
E_OACV〔sP〕	オブジェクトアクセス違反（対象スピンロックに対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（p_acvct が指すメモリ領域への読出しアクセスが許可されていない）
E_OBJ	オブジェクト状態エラー（対象スピンロックは静的 API で生成された：sac_spn の場合，対象スピンロックに対してアクセス許可ベクタが設定済み：SAC_SPN の場合）

【機能】

spnid で指定したスピンロック（対象スピンロック）のアクセス許可ベクタ（4 つのアクセス許可パターンの組）を，各パラメータで指定した値に設定する．

静的 API においては，spnid はオブジェクト識別名，acptn1～acptn4 は整数定数式パラメータである．

SAC_SPN は，対象スピンロックが属する保護ドメインの囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，SAC_SPN，sac_spn をサポートしない．

del_spn　　スピンロックの削除〔TMD〕

【C 言語 API】

```
ER ercd = del_spn(ID spnid)
```

【パラメータ】

ID	spnid	対象スピンロックの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（spnid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象スピンロックが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象スピンロックに対する管理操作が許可されていない）
E_OBJ	オブジェクト状態エラー（対象スピンロックは静的 API で生成された）

【機能】

spnid で指定したスピンロック（対象スピンロック）を削除する．具体的な振舞いは以下の通り．

対象スピンロックの登録が解除され，そのスピンロック ID が未使用の状態に戻される．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，del_spn をサポートしない．

【未決定事項】

対象スピンロックが取得されている状態の場合の振舞いは，今後の課題である．

loc_spn　　スピンロックの取得〔TM〕
iloc_spn　　スピンロックの取得〔IM〕

【C 言語 API】

```
ER ercd = loc_spn(ID spnid)
ER ercd = iloc_spn(ID spnid)
```

【パラメータ】

ID	spnid	対象スピンロックの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し：loc_spn の場合，タスクコンテキストからの呼出し：iloc_spn の場合，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（spnid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象スピンロックが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象スピンロックに対する通常操作 1 が許可されていない：loc_spn の場合）

【機能】

spnid で指定したスピンロック（対象スピンロック）を取得する．具体的な振舞いは以下の通り．

対象スピンロックが取得されていない状態である場合には，プロセッサ間ロックの取得を試みる．ロックが他のプロセッサによって取得されている状態である場合や，他のプロセッサがロックの取得に成功した場合には，ロックが返却されるまでループによって待ち，返却されたらロックの取得を試みる．これを，ロックの取得に成功するまで繰り返す．

ロックの取得に成功した場合には，スピンロックは取得されている状態になる．また，CPU ロックフラグをセットして CPU ロック状態へ遷移し，サービスコールからリターンする．

なお，複数のプロセッサがロックの取得を待っている時に，どのプロセッサが最初にロックを取得できるかは，現時点ではターゲット定義とする．

【補足説明】

対象スピンロックが，loc_spn／iloc_spn を呼び出したプロセッサによって取得されている状態である場合には，スピンロックの取得により CPU ロック状態になっているため，loc_spn／iloc_spn は E_CTX エラーとなる．

プロセッサがロックを取得できる順序を，現時点ではターゲット定義としたが，リアルタイム性保証のためには，（ロックの取得待ちの間に割込みが発生しない限りは）loc_spn／iloc_spn を呼び出した順序でロックを取得できるとするのが望ましい．ただし，ターゲットハードウェアの制限で，そのような実装ができるとは限らないため，現時点ではターゲット定義としている．

---

try_spn　スピンロックの取得（ポーリング）〔TM〕
itry_spn　スピンロックの取得（ポーリング）〔IM〕

【C 言語 API】

```
ER ercd = try_spn(ID spnid)
ER ercd = itry_spn(ID spnid)
```

【パラメータ】

ID	spnid	対象スピンロックの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し：try_spn の場合，タスクコンテキストからの呼出し：itry_spn の場合，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（spnid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象スピンロックが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象スピンロックに対する通常操作 1 が許可されていない：try_spn の場合）
E_OBJ	オブジェクト状態エラー（対象スピンロックが取得されている状態）

【機能】

spnid で指定したスピンロック（対象スピンロック）の取得を試みる．具体的な振舞いは以下の通り．

対象スピンロックが取得されていない状態である場合には，プロセッサ間ロックの取得を試みる．ロ

ックの取得に成功した場合には，スピンロックは取得されている状態になる．また，CPU ロックフラグをセットして CPU ロック状態へ遷移し，サービスコールからリターンする．

対象スピンロックが他のプロセッサによって取得されている状態である場合や，ロックの取得に失敗した場合（他のプロセッサがロックの取得に成功した場合）には，E_OBJ エラーとする．

---

unl_spn　スピンロックの返却〔TM〕
iunl_spn　スピンロックの返却〔IM〕

【C 言語 API】

```
ER ercd = unl_spn(ID spnid)
ER ercd = iunl_spn(ID spnid)
```

【パラメータ】

ID	spnid	対象スピンロックの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し：unl_spn の場合，タスクコンテキストからの呼出し：iunl_spn の場合）
E_ID	不正 ID 番号（spnid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象スピンロックが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象スピンロックに対する通常操作 1 が許可されていない：unl_spn の場合）
E_ILUSE	サービスコール不正使用（対象スピンロックをロックしていない）

【機能】

spnid で指定したスピンロック（対象スピンロック）を返却する．具体的な振舞いは以下の通り．

対象スピンロックが，unl_spn／iunl_spn を呼び出したプロセッサによって取得されている状態である場合には，ロックを返却し，スピンロックを取得されていない状態とする．また，CPU ロックフラグをクリアし，CPU ロック解除状態へ遷移する．

対象スピンロックが，取得されていない状態である場合や，他のプロセッサによって取得されている状態である場合には，E_ILUSE エラーとなる．

ref_spn　　スピンロックの状態参照〔TM〕

【C 言語 API】

```
ER ercd = ref_spn(ID spnid, T_RSPN *pk_rspn)
```

【パラメータ】

ID	spnid	対象スピンロックの ID 番号
T_ RSPN *	pk_ rspn	スピンロックの現在状態を入れる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

＊スピンロックの現在状態（パケットの内容）

STAT	spnstat	ロック状態

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（spnid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象スピンロックが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象スピンロックに対する参照操作が許可されていない）
E_MACV〔P〕	メモリアクセス違反（pk_rspn が指すメモリ領域への書込みアクセスが許可されていない）

【機能】

spnid で指定したスピンロック（対象スピンロック）の現在状態を参照する．参照した現在状態は，pk_rspn で指定したパケットに返される．

spnstat には，対象スピンロックの現在のロック状態を表す次のいずれかの値が返される．

TSPN_UNL	0x01U	取得されていない状態
TSPN_LOC	0x02U	取得されている状態

【使用上の注意】

ref_spn はデバッグ時向けの機能であり，その他の目的に使用することは推奨しない．これは，ref_spn

を呼び出し，対象スピンロックの現在状態を参照した直後に割込みが発生した場合，ref_spn から戻ってきた時には対象スピンロックの状態が変化している可能性があるためである．

4.5. メモリプール管理機能

【TOPPERS/SSP カーネルにおける規定】
SSP カーネルでは，メモリプール管理機能をサポートしない．

【μITRON4.0 仕様との関係】
この仕様では，可変長メモリプール機能はサポートしないこととした．

【仕様決定の理由】
可変長メモリプール機能をサポートしないこととしたのは，メモリ割付けの処理時間とフラグメンテーションの発生を考えると，最適なメモリ管理アルゴリズムはアプリケーション依存となるため，カーネル内で実現するより，ライブラリとして実現する方が適切と考えたためである．

4.5.1. 固定長メモリプール

固定長メモリプールは，生成時に決めたサイズのメモリブロック（固定長メモリブロック）を動的に獲得・返却するための同期・通信オブジェクトである．固定長メモリプールは，固定長メモリプール ID と呼ぶ ID 番号で識別する．

各固定長メモリプールが持つ情報は次の通り．

- 固定長メモリプール属性
- 待ち行列（固定長メモリブロックの獲得待ち状態のタスクのキュー）
- 固定長メモリプール領域
- 固定長メモリプール管理領域
- アクセス許可ベクタ（保護機能対応カーネルの場合）
- 属する保護ドメイン（保護機能対応カーネルの場合）
- 属するクラス（マルチプロセッサ対応カーネルの場合）

待ち行列は，固定長メモリブロックが獲得できるまで待っている状態（固定長メモリブロックの獲得待ち状態）のタスクが，固定長メモリブロックを獲得できる順序でつながれているキューである．

固定長メモリプール領域は，その中から固定長メモリブロックを割り付けるためのメモリ領域である．

固定長メモリプール管理領域は，固定長メモリプール領域中の割当て済みの固定長メモリブロックと

未割当てのメモリ領域に関する情報を格納しておくためのメモリ領域である．

保護機能対応カーネルにおいて，固定長メモリプール管理領域は，カーネルの用いるオブジェクト管理領域として扱われる．

固定長メモリプール属性には，次の属性を指定することができる．

TA_TPRI	0x01U	待ち行列をタスクの優先度順にする

TA_TPRI を指定しない場合，待ち行列は FIFO 順になる．

固定長メモリプール機能に関連するカーネル構成マクロは次の通り．

TNUM_MPFID	登録できる固定長メモリプールの数（動的生成対応でないカーネルでは，静的 API によって登録された固定長メモリプールの数に一致）

【μITRON4.0 仕様との関係】

固定長メモリプール領域として確保すべき領域のサイズを返すカーネル構成マクロ（TSZ_MPF）は廃止した．これは，固定長メモリプール領域をアプリケーションで確保する方法を定めた結果，そのサイズは(blkcnt * ROUND_MPF_T(blksz))で求めることができるようになったためである．

TNUM_MPFID は，μITRON4.0 仕様に規定されていないカーネル構成マクロである．

CRE_MPF　固定長メモリプールの生成〔S〕
acre_mpf　固定長メモリプールの生成〔TD〕

【静的 API】

```
CRE_MPF(ID mpfid, {ATR mpfatr, uint_t blkcnt, uint_t blksz,
                                MPF_T *mpf, void *mpfmb })
```

【C 言語 API】

```
ER_ID mpfid = acre_mpf(const T_CMPF *pk_cmpf)
```

【パラメータ】

ID	mpfid	生成する固定長メモリプールの ID 番号（CRE_MPF の場合）
T_ CMPF *	pk_ cmpf	固定長メモリプールの生成情報を入れたパケットへのポインタ（静的 API を除く）

＊固定長メモリプールの生成情報（パケットの内容）

ATR	mpfatr	固定長メモリプール属性
uint_t	blkcnt	獲得できる固定長メモリブロックの数
uint_t	blksz	固定長メモリブロックのサイズ（バイト数）
MPF_T *	mpf	固定長メモリプール領域の先頭番地
void *	mpfmb	固定長メモリプール管理領域の先頭番地

【リターンパラメータ】

ER_ID	spnid	生成された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_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_RSATR	予約属性（mpfatr が不正または使用できない，属する保護ドメインかクラスが不正）
E_NOSPT	未サポート機能（mpfmb がサポートされていない値）
E_PAR	パラメータエラー（blkcnt，blksz，mpf，mpfmb が不正）
E_OACV〔sP〕	オブジェクトアクセス違反（システム状態に対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（pk_cmpf が指すメモリ領域への読出しアクセスが許可されていない）
E_NOID〔sD〕	ID 番号不足（割り付けられる固定長メモリプール ID がない）
E_NOMEM	メモリ不足（固定長メモリプール領域や固定長メモリプール管理領域が確保できない）
E_OBJ	オブジェクト状態エラー（mpfid で指定した固定長メモリプールが登録済み：CRE_MPF の場合，その他の条件については機能の項を参照すること）

【機能】

各パラメータで指定した固定長メモリプール生成情報に従って，固定長メモリプールを生成する．mpf，blkcnt，blksz から固定長メモリプール領域が，mpfmb と blkcnt から固定長メモリプール管理領域がそ

れぞれ設定され，メモリプール領域全体が未割当ての状態に初期化される．また，待ち行列は空の状態に初期化される．

静的 API においては，mpfid はオブジェクト識別名，blkcnt と blksz は整数定数式パラメータ，mpf と mpfmb は一般定数式パラメータである．コンフィギュレータは，静的 API のメモリ不足（E_NOMEM）エラーを検出することができない．

mpf を NULL とした場合，blkcnt と blksz から決まるサイズの固定長メモリプール領域が，コンフィギュレータまたはカーネルにより確保される．

保護機能対応カーネルでは，コンフィギュレータまたはカーネルにより確保される固定長メモリプール領域は，固定長メモリプールと同じ保護ドメインに属し，固定長メモリプールと同じアクセス許可ベクタを持ったメモリオブジェクト中に確保される．

mpfmb を NULL とした場合，blkcnt から決まるサイズの固定長メモリプール管理領域が，コンフィギュレータまたはカーネルにより確保される．

blkcnt と blksz は，0 より大きい値でなければならない．

〔mpf に NULL 以外を指定した場合〕

mpf に NULL 以外を指定した場合，mpf を先頭番地とする固定長メモリプール領域は，アプリケーションで確保しておく必要がある．固定長メモリプール領域をアプリケーションで確保するために，次のデータ型とマクロを用意している．

MPF_T	固定長メモリプール領域を確保するためのデータ型
COUNT_MPF_T(blksz)	固定長メモリブロックのサイズが blksz の固定長メモリプール領域を確保するために，固定長メモリブロック 1 つあたりに必要な MPF_T 型の配列の要素数
ROUND_MPF_T(blksz)	要素数 COUNT_MPF_T(blksz)の MPF_T 型の配列のサイズ（blksz を，MPF_T 型のサイズの倍数になるように大きい方に丸めた値）

これらを用いて固定長メモリプール領域を確保する方法は次の通り．

```
MPF_T  <固定長メモリプール領域の変数名>[(blkcnt) * COUNT_MPF_T(blksz)]
```

この時，mpf には<固定長メモリプール領域の変数名>を指定する．

これ以外の方法で固定長メモリプール領域を確保する場合には，先頭番地がターゲット定義の制約に合致しており，上記の配列と同じサイズのメモリ領域を確保しなければならない．mpf にターゲット定義の制約に合致しない先頭番地を指定した時には，E_PAR エラーとなる．

保護機能対応カーネルでは，アプリケーションで確保する固定長メモリプール領域は，カーネルに登録されたメモリオブジェクトに含まれていなければならない．指定した固定長メモリプール領域が，カーネルに登録されたメモリオブジェクトに含まれていない場合，E_OBJ エラーとなる．

〔mpfmb に NULL 以外を指定した場合〕

mpfmb に NULL 以外を指定した場合，mpfmb を先頭番地とする固定長メモリプール管理領域は，アプリケーションで確保しておく必要がある．固定長メモリプール管理領域をアプリケーションで確保するために，次のマクロを用意している．

TSZ_MPFMB(blkcnt)	blkcnt で指定した数の固定長メモリブロックを管理することができる固定長メモリプール管理領域のサイズ（バイト数）
TCNT_MPFMB(blkcnt)	blkcnt で指定した数の固定長メモリブロックを管理することができる固定長メモリプール管理領域を確保するために必要な MB_T 型の配列の要素数

これらを用いて固定長メモリプール管理領域を確保する方法は次の通り．

```
MB_T　<固定長メモリプール管理領域の変数名>[TCNT_MPFMB(blkcnt)];
```

この時，mpfmb には<固定長メモリプール管理領域の変数名>を指定する．

この方法に従わず，mpfmb にターゲット定義の制約に合致しない先頭番地を指定した時には，E_PAR エラーとなる．また，保護機能対応カーネルにおいて，mpfmb で指定した固定長メモリプール管理領域がカーネル専用のメモリオブジェクトに含まれない場合，E_OBJ エラーとなる．

【補足説明】

保護機能対応カーネルにおいて，固定長メモリプール領域をアプリケーションで確保する場合には，固定長メモリプール領域が属する保護ドメインとアクセス権の設定は変更されない．これらを適切に設定することは，アプリケーションの責任である．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，CRE_MPF のみをサポートする．また，mpfmb には NULL のみを指定すること

ができる．NULL 以外を指定した場合には，E_NOSPT エラーとなる．ただし，動的生成機能拡張パッケージでは，acre_mpf もサポートする．acre_mpf に対しては，mpfmb に NULL 以外を指定できないという制限はない．

【TOPPERS/FMP カーネルにおける規定】
FMP カーネルでは，CRE_MPF のみをサポートする．また，mpfmb には NULL のみを渡すことができる．NULL 以外を指定した場合には，E_NOSPT エラーとなる．

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，CRE_MPF のみをサポートする．また，mpfmb には NULL のみを渡すことができる．NULL 以外を指定した場合には，E_NOSPT エラーとなる．

【μITRON4.0 仕様との関係】
mpf のデータ型を MPF_T *に変更した．COUNT_MPF_T と ROUND_MPF_T を新設し，固定長メモリプール領域をアプリケーションで確保する方法を規定した．また，μITRON4.0/PX 仕様にあわせて，固定長メモリプール生成情報に，mpfmb を追加した．

【μITRON4.0/PX 仕様との関係】
TCNT_MPFMB を新設し，固定長メモリプール管理領域をアプリケーションで確保する方法を規定した．

AID_MPF　　割付け可能な固定長メモリプール ID の数の指定〔SD〕

【静的 API】

AID_MPF(uint_t nompf)

【パラメータ】

uint_t	nompf	割付け可能な固定長メモリプール ID の数

【エラーコード】

E_RSATR	予約属性（属する保護ドメインまたはクラスが不正）

【機能】
nompf で指定した数の固定長メモリプール ID を，固定長メモリプールを生成するサービスコールによって割付け可能な固定長メモリプール ID として確保する．

nompf は整数定数式パラメータである．

SAC_MPF　　固定長メモリプールのアクセス許可ベクタの設定〔SP〕
sac_mpf　　固定長メモリプールのアクセス許可ベクタの設定〔TPD〕

【静的 API】

```
SAC_MPF(ID mpfid, {ACPTN acptn1, ACPTN acptn2,
                   ACPTN acptn3, ACPTN acptn4})
```

【C 言語 API】

```
ER ercd = sac_mpf(ID mpfid, const ACVCT *p_acvct)
```

【パラメータ】

ID	mpfid	対象固定長メモリプールの ID 番号
ACVCT *	p_acvct	アクセス許可ベクタを入れたパケットへのポインタ（静的 API を除く）

＊アクセス許可ベクタ（パケットの内容）

ACPTN	acptn1	通常操作 1 のアクセス許可パターン
ACPTN	acptn2	通常操作 2 のアクセス許可パターン
ACPTN	acptn3	管理操作のアクセス許可パターン
ACPTN	acptn4	参照操作のアクセス許可パターン

【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_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（mpfid が不正）
E_RSATR	予約属性（属する保護ドメインかクラスが不正：SAC_MPF の場合）
E_NOEXS〔D〕	オブジェクト未登録（対象固定長メモリプールが未登録）
E_OACV〔sP〕	オブジェクトアクセス違反（対象固定長メモリプールに対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（p_acvct が指すメモリ領域への読出しアクセスが許可されていない）
E_OBJ	オブジェクト状態エラー（対象固定長メモリプールは静的 API で生成された：sac_mpf の場合，対象固定長メモリプールに対してアクセス許可ベクタが設定済み：SAC_MPF の場合）

【機能】

mpfid で指定した固定長メモリプール（対象固定長メモリプール）のアクセス許可ベクタ（4 つのアクセス許可パターンの組）を，各パラメータで指定した値に設定する．対象固定長メモリプールの固定長メモリプール領域がコンフィギュレータまたはカーネルにより確保されたものである場合には，固定長メモリプール領域のアクセス許可ベクタも，各パラメータで指定した値に設定する．

静的 API においては，mpfid はオブジェクト識別名，acptn1～acptn4 は整数定数式パラメータである．

SAC_MPF は，対象固定長メモリプールが属する保護ドメインの囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，SAC_MPF，sac_mpf をサポートしない．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，SAC_MPF，sac_mpf をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，SAC_MPF のみをサポートする．

del_mpf　　固定長メモリプールの削除〔TD〕

【C 言語 API】

```
ER ercd = del_mpf(ID mpfid)
```

【パラメータ】

ID	mpfid	対象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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（mpfid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象固定長メモリプールが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象固定長メモリプールに対する管理操作が許可されていない）
E_OBJ	オブジェクト状態エラー（対象固定長メモリプールは静的 API で生成された）

【機能】

mpfid で指定した固定長メモリプール（対象固定長メモリプール）を削除する．具体的な振舞いは以下の通り．

対象固定長メモリプールの登録が解除され，その固定長メモリプール ID が未使用の状態に戻される．また，対象固定長メモリプールの待ち行列につながれたタスクは，待ち行列の先頭のタスクから順に待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_DLT エラーが返る．

【使用上の注意】

del_mpf により複数のタスクが待ち解除される場合，サービスコールの処理時間およびカーネル内での割込み禁止時間が，待ち解除されるタスクの数に比例して長くなる．特に，多くのタスクが待ち解除される場合，カーネル内での割込み禁止時間が長くなるため，注意が必要である．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，del_mpf をサポートしない．ただし，動的生成機能拡張パッケージでは，del_mpf

をサポートする.

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，del_mpf をサポートしない.

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，del_mpf をサポートしない.

---

get_mpf　　固定長メモリブロックの獲得〔T〕
pget_mpf　　固定長メモリブロックの獲得（ポーリング）〔T〕
tget_mpf　　固定長メモリブロックの獲得（タイムアウト付き）〔T〕

【C 言語 API】

```
ER ercd = get_mpf(ID mpfid, void **p_blk)
ER ercd = pget_mpf(ID mpfid, void **p_blk)
ER ercd = tget_mpf(ID mpfid, void **p_blk, TMO tmout)
```

【パラメータ】

ID	mpfid	対象固定長メモリプールの ID 番号
void **	p_blk	獲得した固定長メモリブロックの先頭番地を入れるメモリ領域へのポインタ
TMO	tmout	タイムアウト時間（twai_mpf の場合）

【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
void *	blk	獲得した固定長メモリブロックの先頭番地

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し，ディスパッチ保留状態からの呼出し：pget_mpf を除く）
E_NOSPT	未サポート機能（制約タスクからの呼出し：pget_mpf を除く）
E_ID	不正 ID 番号（mpfid が不正）
E_PAR	パラメータエラー（tmout が不正：tget_mpf の場合）
E_NOEXS〔D〕	オブジェクト未登録（対象固定長メモリプールが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象固定長メモリプールに対する通常操作 1 が許可されていない）
E_MACV〔P〕	メモリアクセス違反（p_blk が指すメモリ領域への読出しアクセスが許可されていない）
E_TMOUT	ポーリング失敗またはタイムアウト（get_mpf を除く）
E_RLWAI	待ち禁止状態または待ち状態の強制解除（pget_mpf を除く）
E_DLT	待ちオブジェクトの削除または再初期化（pget_mpf を除く）

【機能】

mpfid で指定した固定長メモリプール（対象固定長メモリプール）から固定長メモリブロックを獲得し，その先頭番地を blk に返す．具体的な振舞いは以下の通り．

対象固定長メモリプールの固定長メモリプール領域の中に，固定長メモリブロックを割り付けることのできる未割当てのメモリ領域がある場合には，固定長メモリブロックが 1 つ割り付けられ，その先頭番地が blk に返される．

未割当てのメモリ領域がない場合には，自タスクは固定長メモリプールの獲得待ち状態となり，対象固定長メモリプールの待ち行列につながれる．

rel_mpf　　固定長メモリブロックの返却〔T〕

【C 言語 API】

```
ER ercd = rel_mpf(ID mpfid, void *blk)
```

【パラメータ】

ID	mpfid	対象固定長メモリプールの ID 番号
void *	blk	返却する固定長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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（mpfid が不正）
E_PAR	パラメータエラー（blk が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象固定長メモリプールが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象固定長メモリプールに対する通常操作 2 が許可されていない）

【機能】

mpfid で指定した固定長メモリプール（対象固定長メモリプール）に，blk で指定した固定長メモリブロックを返却する．具体的な振舞いは以下の通り．

対象固定長メモリプールの待ち行列にタスクが存在する場合には，待ち行列の先頭のタスクが，blk で指定した固定長メモリブロックを獲得し，待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_OK が返る．

待ち行列にタスクが存在しない場合には，blk で指定した固定長メモリブロックは，対象固定長メモリプールのメモリプール領域に返却される．

blk が，対象固定長メモリプールから獲得した固定長メモリブロックの先頭番地でない場合には，E_PAR エラーとなる．

ini_mpf　　固定長メモリプールの再初期化〔T〕

【C 言語 API】

```
ER ercd = ini_mpf(ID mpfid)
```

【パラメータ】

ID	mpfid	対象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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（mpfid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象固定長メモリプールが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象固定長メモリプールに対する管理操作が許可されていない）

【機能】

mpfid で指定した固定長メモリプール（対象固定長メモリプール）を再初期化する．具体的な振舞いは以下の通り．

対象固定長メモリプールのメモリプール領域全体が未割当ての状態に初期化される．また，対象固定長メモリプールの待ち行列につながれたタスクは，待ち行列の先頭のタスクから順に待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_DLT エラーが返る．

【使用上の注意】

ini_mpf により複数のタスクが待ち解除される場合，サービスコールの処理時間およびカーネル内での割込み禁止時間が，待ち解除されるタスクの数に比例して長くなる．特に，多くのタスクが待ち解除される場合，カーネル内での割込み禁止時間が長くなるため，注意が必要である．

固定長メモリプールを再初期化した場合に，アプリケーションとの整合性を保つのは，アプリケーションの責任である．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様に定義されていないサービスコールである．

ref_mpf　　固定長メモリプールの状態参照〔T〕

【C 言語 API】

```
ER ercd = ref_mpf(ID mpfid, T_RMPF *pk_rmpf)
```

【パラメータ】

ID	mpfid	対象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の現在状態を入れる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

＊固定長メモリプールの現在状態（パケットの内容）

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_t	fblkcnt	固定長メモリプール領域の空きメモリ領域に割り付けることができる固定長メモリブロックの数

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（mpfid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象固定長メモリプールが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象固定長メモリプールに対する参照操作が許可されていない）
E_MACV〔P〕	メモリアクセス違反（pk_rmpf が指すメモリ領域への書込みアクセスが許可されていない）

【機能】

mpfid で指定した固定長メモリプール（対象固定長メモリプール）の現在状態を参照する．参照した現在状態は，pk_rmpf で指定したパケットに返される．

対象固定長メモリプールの待ち行列にタスクが存在しない場合，wtskid には TSK_NONE（＝0）が返る．

【使用上の注意】

ref_mpf はデバッグ時向けの機能であり，その他の目的に使用することは推奨しない．これは，ref_mpf を呼び出し，対象固定長メモリプールの現在状態を参照した直後に割込みが発生した場合，ref_mpf から戻ってきた時には対象固定長メモリプールの状態が変化している可能性があるためである．

4.6. 時間管理機能

4.6.1. システム時刻管理

システム時刻は，カーネルによって管理され，タイムアウト処理，タスクの遅延，周期ハンドラの起動，アラームハンドラの起動に使用される時刻を管理するカーネルオブジェクトである．システム時刻は，符号無しの整数型である SYSTIM 型で表され，単位はミリ秒である．

システム時刻は，カーネルの初期化時に 0 に初期化される．タイムティックを通知するためのタイマ割込みが発生する毎にカーネルによって更新され，SYSTIM 型で表せる最大値（ULONG_MAX）を超えると 0 に戻される．タイムティックの周期は，ターゲット定義である．また，システム時刻の精度はターゲットに依存する．

マルチプロセッサ対応でないカーネルと，マルチプロセッサ対応カーネルでグローバルタイマ方式を用いている場合には，システム時刻は，システムに 1 つのみ存在する．マルチプロセッサ対応カーネルでローカルタイマ方式を用いている場合には，システム時刻は，プロセッサ毎に存在する．ローカルタイマ方式とグローバルタイマ方式については，「2.3.4 マルチプロセッサ対応」の節を参照すること．

マルチプロセッサ対応カーネルでローカルタイマ方式を用いている場合には，タイムアウト処理とタスクの遅延処理には，待ち解除されるタスクが割り付けられているプロセッサのシステム時刻が用いられる．また，周期ハンドラとアラームハンドラの起動には，それが割り付けられているプロセッサのシステム時刻が用いられる．これらの処理単位がマイグレーションする場合には，用いられるシステム時刻も変更される．この場合にも，イベントの処理が行われるのは，基準時刻から相対時間によって指定した以上の時間が経過した後となるという原則は維持される．

1 回のタイムティックの発生により，複数のイベントの処理を行うべき状況になった場合，それらの処理の間の処理順序は規定されない．

性能評価用システム時刻は，性能評価に使用することを目的とした，システム時刻よりも精度の高い時刻である．性能評価用システム時刻は，符号無しの整数型である SYSUTM 型で表され，単位はマイクロ秒である．ただし，実際の精度はターゲットに依存する．

マルチプロセッサ対応カーネルにおける性能評価用システム時刻の扱いは，ターゲット定義とする．

システム時刻管理機能に関連するカーネル構成マクロは次の通り．

TIC_NUME	タイムティックの周期（単位はミリ秒）の分子
TIC_DENO	タイムティックの周期（単位はミリ秒）の分母

TOPPERS_SUPPORT_GET_UTM	get_utm がサポートされている

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，時間管理機能をサポートしない．

【使用上の注意】

タイムティックを通知するためのタイマ割込みが長時間マスクされた場合（タイマ割込みより優先し

て実行される割込み処理が長時間続けて実行された場合を含む）や，シミュレーション環境においてシミュレータのプロセスが長時間スケジュールされなかった場合には，システム時刻が正しく更新されない可能性があるため，注意が必要である．

【μITRON4.0 仕様との関係】

システム時刻を設定するサービスコール（set_tim）を廃止した．また，タイムティックを供給する機能は，カーネル内に実現することとし，そのためのサービスコール（isig_tim）は廃止した．

【μITRON4.0/PX 仕様との関係】

システム時刻のアクセス許可ベクタは廃止し，システム状態のアクセス許可ベクタで代替することとした．そのため，システム時刻のアクセス許可ベクタを設定する静的 API（SAC_TIM）は廃止した．

---

get_tim　　システム時刻の参照〔T〕

【C 言語 API】

```
R ercd = get_tim(SYSTIM *p_systim)
```

【パラメータ】

SYSTIM *	p_systim	システム時刻を入れるメモリ領域へのポインタ

【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
SYSTIM	systim	システム時刻の現在値

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_OACV〔P〕	オブジェクトアクセス違反（システム状態に対する参照操作が許可されていない）
E_MACV〔P〕	メモリアクセス違反（p_systim が指すメモリ領域への書込みアクセスが許可されていない）

【機能】

システム時刻の現在値を参照する．参照したシステム時刻は，p_systim で指定したメモリ領域に返される．

マルチプロセッサ対応カーネルでローカルタイマ方式を用いている場合には，自タスクが割り付けられているプロセッサのシステム時刻の現在値を参照する．

【補足説明】

マルチプロセッサ対応カーネルでローカルタイマ方式を用いている場合に，他のプロセッサのシステム時刻の現在値を参照する機能は用意していない．

---

get_utm　　性能評価用システム時刻の参照〔TI〕

【C 言語 API】

```
ER ercd = get_utm(SYSUTM *p_sysutm)
```

【パラメータ】

SYSUTM	p_sysutm	性能評価用システム時刻を入れるメモリ領域へのポインタ

【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
SYSUTM	sysutm	性能評価用システム時刻の現在値

【エラーコード】

E_NOSPT	未サポート機能（get_utm がサポートされていない）
E_MACV〔P〕	メモリアクセス違反（p_sysutm が指すメモリ領域へ書込みアクセスが許可されていない）

【機能】

性能評価用システム時刻の現在値を参照する．参照した性能評価用システム時刻は，p_sysutm で指定したメモリ領域に返される．

get_utm は，任意の状態から呼び出すことができる．タスクコンテキストからも非タスクコンテキストからも呼び出すことができるし，CPU ロック状態であっても呼び出すことができる．

ターゲット定義で，get_utm がサポートされていない場合がある．get_utm がサポートされている場合には, TOPPERS_SUPPORT_GET_UTM がマクロ定義される. サポートされていない場合に get_utm を呼び出すと，E_NOSPT エラーが返るか，リンク時にエラーとなる．

【使用方法】

get_utm を使用してプログラムの処理時間を計測する場合には，次の手順を取る．処理時間を計測し

たいプログラムの実行直前と実行直後に，get_utm を用いて性能評価用システム時刻を読み出す．その差を求めることで，対象プログラムの処理時間に，get_utm 自身の処理時間を加えたものが得られる．

マルチプロセッサ対応カーネルにおいては，異なるプロセッサで読み出した性能評価用システム時刻の差を求めることで，処理時間が正しく計測できるとは限らない．

【使用上の注意】
get_utm は性能評価のための機能であり，その他の目的に使用することは推奨しない．

get_utm は，任意の状態から呼び出すことができるように，全割込みロック状態を用いて実装されている．そのため，get_utm を用いると，カーネル管理外の割込みの応答性が低下する．

システム時刻が正しく更新されない状況では，get_utm は誤った性能評価用システム時刻を返す可能性がある．システム時刻の更新が確実に行われることを保証できない場合には，get_utm が誤った性能評価用システム時刻を返す可能性を考慮に入れて使用しなければならない．

【μITRON4.0 仕様との関係】
μITRON4.0 仕様に定義されていないサービスコールである．

4.6.2. 周期ハンドラ

周期ハンドラは，指定した周期で起動されるタイムイベントハンドラである．周期ハンドラは，周期ハンドラ ID と呼ぶ ID 番号によって識別する．

各周期ハンドラが持つ情報は次の通り．

- 周期ハンドラ属性
- 周期ハンドラの動作状態
- 次に周期ハンドラを起動する時刻
- 拡張情報
- 周期ハンドラの先頭番地
- 起動周期
- 起動位相
- アクセス許可ベクタ（保護機能対応カーネルの場合）
- 属する保護ドメイン（保護機能対応カーネルの場合）
- 属するクラス（マルチプロセッサ対応カーネルの場合）

周期ハンドラの起動時刻は，後述する基準時刻から，以下の式で求められる相対時間後である．

起動位相＋起動周期×(n－1)　　n＝1,2,…

周期ハンドラの動作状態は，動作している状態と動作していない状態のいずれかをとる．周期ハンドラを動作している状態にすることを動作開始，動作していない状態にすることを動作停止という．

周期ハンドラが動作している状態の場合には，周期ハンドラを起動する時刻になると，周期ハンドラの起動処理が行われる．具体的には，拡張情報をパラメータとして，周期ハンドラが呼び出される．

保護機能対応カーネルにおいて，周期ハンドラが属することのできる保護ドメインは，カーネルドメインに限られる．

周期ハンドラ属性には，次の属性を指定することができる．

TA_STA	0x02U	待ち行列をタスクの優先度順にする
TA_PHS	0x04U	周期ハンドラを生成した時刻を基準時刻とする

TA_STA を指定しない場合，周期ハンドラの生成直後には，周期ハンドラは動作していない状態となる．

TA_PHS を指定しない場合には，周期ハンドラを動作開始した時刻が，周期ハンドラを起動する時刻の基準時刻となる．TA_PHS を指定した場合には，周期ハンドラを生成した時刻（静的 API で生成した場合にはカーネルの起動時刻）が，基準時刻となる．

次に周期ハンドラを起動する時刻は，周期ハンドラが動作している状態でのみ有効で，必要に応じて，カーネルの起動時，周期ハンドラの動作開始時，周期ハンドラの起動処理時に設定される．

マルチプロセッサ対応カーネルでグローバルタイマ方式を用いている場合には，周期ハンドラは，システム時刻管理プロセッサのみが割付け可能プロセッサであるクラスにのみ属することができる．すなわち，周期ハンドラは，システム時刻管理プロセッサによって実行される．

C 言語による周期ハンドラの記述形式は次の通り．

```
void cyclic_handler(intptr_t exinf)
{
    周期ハンドラ本体
}
```

exinf には，周期ハンドラの拡張情報が渡される．

周期ハンドラ機能に関連するカーネル構成マクロは次の通り．

TNUM_CYCID	登録できる周期ハンドラの数（動的生成対応でないカーネルでは，静的 API によって登録された周期ハンドラの数に一致）

【TOPPERS/ASP カーネルにおける規定】
ASP カーネルでは，TA_PHS 属性の周期ハンドラをサポートしない．

【TOPPERS/FMP カーネルにおける規定】
FMP カーネルでは，TA_PHS 属性の周期ハンドラをサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，TA_PHS 属性の周期ハンドラをサポートしない．

【μITRON4.0 仕様との関係】
TNUM_CYCID は，μITRON4.0 仕様に規定されていないカーネル構成マクロである．

CRE_CYC　周期ハンドラの生成〔S〕
acre_cyc　周期ハンドラの生成〔TD〕

【静的 API】

```
CRE_CYC(ID cycid, {ATR cycatr, intptr_t exinf, CYCHDR cychdr,
                   RELTIM cyctim, RELTIM cycphs })
```

【C 言語 API】

```
ER_ID cycid = acre_cyc(const T_CCYC *pk_ccyc)
```

【パラメータ】

ID	cycid	生成する周期ハンドラの ID 番号（CRE_CYC の場合）
T_CCYC *	pk_ ccyc	周期ハンドラの生成情報を入れたパケットへのポインタ（静的 API を除く）

＊周期ハンドラの生成情報（パケットの内容）

ATR	cycatr	周期ハンドラ属性
intptr_t	exinf	周期ハンドラの拡張情報
CYCHDR	cychdr	周期ハンドラの先頭番地
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

【リターンパラメータ】

ER_ID	cycid	生成された周期ハンドラの ID 番号（正の値）またはエラーコード

【エラーコード】

E_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_RSATR	予約属性（cycatr が不正または使用できない，属する保護ドメインかクラスが不正）
E_PAR	パラメータエラー（cychdr，cyctim，cycphs が不正）
E_OACV〔sP〕	オブジェクトアクセス違反（システム状態に対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（pk_ccyc が指すメモリ領域への読出しアクセスが許可されていない）
E_NOID〔sD〕	ID 番号不足（割り付けられる周期ハンドラ ID がない）
E_OBJ	オブジェクト状態エラー（cycid で指定した周期ハンドラが登録済み：CRE_CYC の場合）

【機能】

各パラメータで指定した周期ハンドラ生成情報に従って，周期ハンドラを生成する．具体的な振舞いは以下の通り．

cycatr に TA_STA を指定した場合，対象周期ハンドラは動作している状態となる．次に周期ハンドラを起動する時刻は，サービスコールを呼び出した時刻（静的 API の場合はカーネルの起動時刻）から，cycphs で指定した相対時間後に設定される．

cycatr に TA_STA を指定しない場合，対象周期ハンドラは動作していない状態に初期化される．

静的 API においては，cycid はオブジェクト識別名，cycatr，cyctim，cycphs は整数定数式パラメータ，exinf と cychdr は一般定数式パラメータである．

保護機能対応カーネルにおいて，CRE_CYC は，カーネルドメインの囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．また，acre_cyc で，生成する周期ハンドラが属する保護ドメインとしてカーネルドメイン以外を指定した場合には，E_RSATR エラーとなる．

cyctim は，0 より大きく，TMAX_RELTIM 以下の値でなければならない．また，cycphs は，TMAX_RELTIM 以下でなければならない．cycphs に cyctim より大きい値を指定してもよい．

マルチプロセッサ対応カーネルでグローバルタイマ方式を用いている場合で，生成する周期ハンドラの属するクラスの割付け可能プロセッサが，システム時刻管理プロセッサのみでない場合には，E_RSATR エラーとなる．

【補足説明】
静的 API において，cycatr に TA_STA を，cycphs に 0 を指定した場合，周期ハンドラが最初に呼び出されるのは，カーネル起動後最初のタイムティックになる．cycphs に 1 を指定した場合も同じ振舞いとなるため，静的 API で cycatr に TA_STA が指定されている場合には，cycphs に 0 を指定することは推奨されず，コンフィギュレータが警告メッセージを出力する．

【TOPPERS/ASP カーネルにおける規定】
ASP カーネルでは，CRE_CYC のみをサポートする．ただし，TA_PHS 属性の周期ハンドラはサポートしない．動的生成機能拡張パッケージでは，acre_cyc もサポートする．

【TOPPERS/FMP カーネルにおける規定】
FMP カーネルでは，CRE_CYC のみをサポートする．ただし，TA_PHS 属性の周期ハンドラはサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，CRE_CYC のみをサポートする．ただし，TA_PHS 属性の周期ハンドラはサポートしない．

【μITRON4.0 仕様との関係】
cychdr のデータ型を CYCHDR に変更した．また，cycphs に cyctim より大きい値を指定した場合の振舞いと，静的 API で cycphs に 0 を指定した場合の振舞いを規定した．

AID_CYC　　割付け可能な周期ハンドラ ID の数の指定〔SD〕

【静的 API】

```
AID_CYC(uint_t nocyc)
```

【パラメータ】

uint_t	nocyc	割付け可能な周期ハンドラ ID の数

【エラーコード】

E_RSATR	予約属性（属する保護ドメインまたはクラスが不正）

【機能】

nocyc で指定した数の周期ハンドラ ID を，周期ハンドラを生成するサービスコールによって割付け可能な周期ハンドラ ID として確保する．

nocyc は整数定数式パラメータである．

SAC_CYC　　周期ハンドラのアクセス許可ベクタの設定〔SP〕
sac_cyc　　周期ハンドラのアクセス許可ベクタの設定〔TPD〕

【静的 API】

【C 言語 API】

```
ER ercd = sac_cyc(ID cycid, const ACVCT *p_acvct)
```

【パラメータ】

ID	cycid	対象周期ハンドラの ID 番号
ACVCT *	p_acvct	アクセス許可ベクタを入れたパケットへのポインタ（静的 API を除く）

＊アクセス許可ベクタ（パケットの内容）

ACPTN	acptn1	通常操作 1 のアクセス許可パターン
ACPTN	acptn2	通常操作 2 のアクセス許可パターン
ACPTN	acptn3	管理操作のアクセス許可パターン
ACPTN	acptn4	参照操作のアクセス許可パターン

【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_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（cycid が不正）
E_RSATR	予約属性（属する保護ドメインかクラスが不正：SAC_CYC の場合）
E_NOEXS〔D〕	オブジェクト未登録（対象周期ハンドラが未登録）
E_OACV〔sP〕	オブジェクトアクセス違反（対象周期ハンドラに対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（p_acvct が指すメモリ領域への読出しアクセスが許可されていない）
E_OBJ	オブジェクト状態エラー（対象周期ハンドラは静的 API で生成された：sac_cyc の場合，対象周期ハンドラに対してアクセス許可ベクタが設定済み：SAC_CYC の場合）

【機能】

cycid で指定した周期ハンドラ（対象周期ハンドラ）のアクセス許可ベクタ（4 つのアクセス許可パターンの組）を，各パラメータで指定した値に設定する．

静的 API においては，cycid はオブジェクト識別名，acptn1～acptn4 は整数定数式パラメータである．

SAC_CYC は，対象周期ハンドラが属する保護ドメイン（この仕様ではカーネルドメインに限られる）の囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，SAC_CYC，sac_cyc をサポートしない．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，SAC_CYC，sac_cyc をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，SAC_CYC のみをサポートする．

del_cyc　　周期ハンドラの削除〔TD〕

【C 言語 API】

```
ER ercd = del_cyc(ID cycid)
```

【パラメータ】

ID	cycid	対象周期ハンドラの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（cycid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象周期ハンドラが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象周期ハンドラに対する管理操作が許可されていない）
E_OBJ	オブジェクト状態エラー（対象周期ハンドラは静的 API で生成された）

【機能】

cycid で指定した周期ハンドラ（対象周期ハンドラ）を削除する．具体的な振舞いは以下の通り．

対象周期ハンドラの登録が解除され，その周期ハンドラ ID が未使用の状態に戻される．対象周期ハンドラが動作している状態であった場合には，動作していない状態にされた後に，登録が解除される．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，del_cyc をサポートしない．ただし，動的生成機能拡張パッケージでは，del_cyc をサポートする．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，del_cyc をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，del_cyc をサポートしない．

sta_cyc　　周期ハンドラの動作開始〔T〕

【C 言語 API】

```
ER ercd = sta_cyc(ID cycid)
```

【パラメータ】

ID	cycid	対象周期ハンドラの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（cycid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象周期ハンドラが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象周期ハンドラに対する通常操作 1 が許可されていない）

【機能】

cycid で指定した周期ハンドラ（対象周期ハンドラ）を動作開始する．具体的な振舞いは以下の通り．

対象周期ハンドラが動作していない状態であれば，対象周期ハンドラは動作している状態となる．次に周期ハンドラを起動する時刻は，sta_cyc を呼び出して以降の最初の起動時刻に設定される．

対象周期ハンドラが動作している状態であれば，次に周期ハンドラを起動する時刻の再設定のみが行われる．

【補足説明】

TA_PHS 属性でない周期ハンドラの場合，次に周期ハンドラを起動する時刻は，sta_cyc を呼び出してから，対象周期ハンドラの起動位相で指定した相対時間後に設定される．

対象周期ハンドラが TA_PHS 属性で，動作している状態であれば，次に周期ハンドラを起動する時刻は変化しない．

【μITRON4.0 仕様との関係】

TA_PHS 属性でない周期ハンドラにおいて，sta_cyc を呼び出した後，最初に周期ハンドラが起動される時刻を変更した．μITRON4.0 仕様では，sta_cyc を呼び出してから周期ハンドラの起動周期で指定した相対時間後となっているが，この仕様では，起動位相で指定した相対時間後とした．

msta_cyc　　割付けプロセッサ指定での周期ハンドラの動作開始〔TM〕

【C 言語 API】

```
ER ercd = msta_cyc(ID cycid, ID prcid)
```

【パラメータ】

ID	cycid	対象周期ハンドラの ID 番号
ID	prcid	周期ハンドラの割付け対象のプロセッサの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_NOSPT	未サポート機能（グローバルタイマ方式を用いている場合）
E_ID	不正 ID 番号（cycid，prcid が不正）
E_PAR	パラメータエラー（対象周期ハンドラは prcid で指定したプロセッサに割り付けられない）
E_NOEXS〔D〕	オブジェクト未登録（対象周期ハンドラが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象周期ハンドラに対する通常操作 1 が許可されていない）

【機能】

prcid で指定したプロセッサを割付けプロセッサとして，cycid で指定した周期ハンドラ（対象周期ハンドラ）を動作開始する．具体的な振舞いは以下の通り．

対象周期ハンドラが動作していない状態であれば，対象周期ハンドラの割付けプロセッサが prcid で指定したプロセッサに変更された後，対象周期ハンドラは動作している状態となる．次に周期ハンドラを起動する時刻は，msta_cyc を呼び出して以降の最初の起動時刻に設定される．

対象周期ハンドラが動作している状態であれば，対象周期ハンドラの割付けプロセッサが prcid で指定したプロセッサに変更された後，次に周期ハンドラを起動する時刻の再設定が行われる．

対象周期ハンドラが実行中である場合には，割付けプロセッサを変更しても，実行中の周期ハンドラを実行するプロセッサは変更されない．対象周期ハンドラが変更後の割付けプロセッサで実行されるの

は，次に起動される時からである．

対象周期ハンドラの属するクラスの割付け可能プロセッサが，prcid で指定したプロセッサを含んでいない場合には，E_PAR エラーとなる．

prcid に TPRC_INI（＝0）を指定すると，対象周期ハンドラの割付けプロセッサを，それが属するクラスの初期割付けプロセッサとする．

グローバルタイマ方式を用いている場合，msta_cyc は E_NOSPT を返す．

【補足説明】

TA_PHS 属性でない周期ハンドラの場合，次に周期ハンドラを起動する時刻は，msta_cyc を呼び出してから，対象周期ハンドラの起動位相で指定した相対時間後に設定される．

【使用上の注意】

msta_cyc で実行中の周期ハンドラの割付けプロセッサを変更した場合，同じ周期ハンドラが異なるプロセッサで同時に実行される可能性がある．特に，対象周期ハンドラの起動位相が 0 の場合に，注意が必要である．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様に定義されていないサービスコールである．

stp_cyc　　周期ハンドラの動作停止〔T〕

【C 言語 API】

```
ER ercd = stp_cyc(ID cycid)
```

【パラメータ】

ID	cycid	対象周期ハンドラの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（cycid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象周期ハンドラが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象周期ハンドラに対する通常操作 2 が許可されていない）

【機能】

cycid で指定した周期ハンドラ（対象周期ハンドラ）を動作停止する．具体的な振舞いは以下の通り．

対象周期ハンドラが動作している状態であれば，動作していない状態になる．対象周期ハンドラが動作していない状態であれば，何も行われずに正常終了する．

ref_cyc　　周期ハンドラの状態参照〔T〕

【C 言語 API】

```
ER ercd = ref_cyc(ID cycid, T_RCYC *pk_rcyc)
```

【パラメータ】

ID	cycid	対象周期ハンドラの ID 番号
T_RCYC *	pk_rcyc	周期ハンドラの現在状態を入れる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

＊周期ハンドラの現在状態（パケットの内容）

STAT	cycstat	周期ハンドラの動作状態
RELTIM	lefttim	次に周期ハンドラを起動する時刻までの相対時間
ID	prcid	周期ハンドラの割付けプロセッサの ID（マルチプロセッサ対応カーネルの場合）

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（cycid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象周期ハンドラが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象周期ハンドラに対する参照操作が許可されていない）
E_MACV〔P〕	メモリアクセス違反（pk_rcyc が指すメモリ領域への書込みアクセスが許可されていない）

【機能】

cycid で指定した周期ハンドラ（対象周期ハンドラ）の現在状態を参照する．参照した現在状態は，pk_rcyc で指定したパケットに返される．

cycstat には，対象周期ハンドラの現在の動作状態を表す次のいずれかの値が返される．

TCYC_STP	0x01U	周期ハンドラが動作していない状態
TCYC_STA	0x02U	周期ハンドラが動作している状態

対象周期ハンドラが動作している状態である場合には，lefttim に，次に周期ハンドラ起動する時刻までの相対時間が返される．対象周期ハンドラが動作していない状態である場合には，lefttim の値は保証されない．

マルチプロセッサ対応カーネルでは，prcid に，対象周期ハンドラの割付けプロセッサの ID 番号が返される．

【使用上の注意】

ref_cyc はデバッグ時向けの機能であり，その他の目的に使用することは推奨しない．これは，ref_cyc を呼び出し，対象周期ハンドラの現在状態を参照した直後に割込みが発生した場合，ref_cyc から戻ってきた時には対象周期ハンドラの状態が変化している可能性があるためである．

【μITRON4.0 仕様との関係】

TCYC_STP と TCYC_STA を値を変更した．

4.6.3. アラームハンドラ

アラームハンドラは，指定した相対時間後に起動されるタイムイベントハンドラである．アラームハンドラは，アラームハンドラ ID と呼ぶ ID 番号によって識別する．

各アラームハンドラが持つ情報は次の通り．

- アラームハンドラ属性
- アラームハンドラの動作状態
- アラームハンドラを起動する時刻
- 拡張情報
- アラームハンドラの先頭番地
- アクセス許可ベクタ（保護機能対応カーネルの場合）
- 属する保護ドメイン（保護機能対応カーネルの場合）
- 属するクラス（マルチプロセッサ対応カーネルの場合）

アラームハンドラの動作状態は，動作している状態と動作していない状態のいずれかをとる．アラームハンドラを動作している状態にすることを動作開始，動作していない状態にすることを動作停止という．

アラームハンドラを起動する時刻は，アラームハンドラを動作開始する時に設定される．

アラームハンドラが動作している状態の場合には，アラームハンドラを起動する時刻になると，アラームハンドラの起動処理が行われる．具体的には，まず，アラームハンドラが動作していない状態にされる．その後に，拡張情報をパラメータとして，アラームハンドラが呼び出される．

保護機能対応カーネルにおいて，アラームハンドラが属することのできる保護ドメインは，カーネルドメインに限られる．

マルチプロセッサ対応カーネルでグローバルタイマ方式を用いている場合には，アラームハンドラは，割付け可能プロセッサがシステム時刻管理プロセッサのみであるクラスにのみ属することができる．すなわち，アラームハンドラは，システム時刻管理プロセッサによって実行される．

C 言語によるアラームハンドラの記述形式は次の通り．

```
void alarm_handler(intptr_t exinf)
{
    アラームハンドラ本体
}
```

exinf には，アラームハンドラの拡張情報が渡される．

アラームハンドラ機能に関連するカーネル構成マクロは次の通り.

TNUM_ALMID	登録できるアラームハンドラの数（動的生成対応でないカーネルでは，静的 API によって登録された周期ハンドラの数に一致）

【μITRON4.0 仕様との関係】

TNUM_ALMID は，μITRON4.0 仕様に規定されていないカーネル構成マクロである.

CRE_ALM　アラームハンドラの生成〔S〕
acre_alm　アラームハンドラの生成〔TD〕

【静的 API】

```
CRE_ALM(ID almid, {ATR almatr, intptr_t exinf, ALMHDR almhdr})
```

【C 言語 API】

```
ER_ID almid = acre_alm(const T_CALM *pk_calm)
```

【パラメータ】

ID	almid	生成するアラームハンドラの ID 番号（CRE_ALM の場合）
T_CSEM *	pk_calm	アラームハンドラの生成情報を入れたパケットへのポインタ（静的 API を除く）

＊アラームハンドラの生成情報（パケットの内容）

ATR	almatr	アラームハンドラ属性
intptr_t	exinf	アラームハンドラの拡張情報
ALMHDR	almhdr	アラームハンドラの先頭番地

【リターンパラメータ】

ER_ID	almid	生成され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_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_RSATR	予約属性（almatr が不正または使用できない，属する保護ドメインかクラスが不正）
E_PAR	パラメータエラー（almhdr が不正）
E_OACV〔sP〕	オブジェクトアクセス違反（システム状態に対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（pk_calm が指すメモリ領域への読出しアクセスが許可されていない）
E_NOID〔sD〕	ID 番号不足（割り付けられるアラームハンドラ ID がない）
E_OBJ	オブジェクト状態エラー（almid で指定したアラームハンドラが登録済み：CRE_ALM の場合）

【機能】

各パラメータで指定したアラームハンドラ生成情報に従って，アラームハンドラを生成する．対象アラームハンドラは，動作していない状態に初期化される．

静的 API においては，almid はオブジェクト識別名，almatr は整数定数式パラメータ，exinf と almhdr は一般定数式パラメータである．

保護機能対応カーネルにおいて，CRE_ALM は，カーネルドメインの囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．また，
acre_alm で，生成するアラームハンドラが属する保護ドメインとしてカーネルドメイン以外を指定した場合には，E_RSATR エラーとなる．

マルチプロセッサ対応カーネルでグローバルタイマ方式を用いている場合で，生成するアラームハンドラの属するクラスの割付け可能プロセッサが，システム時刻管理プロセッサのみでない場合には，E_RSATR エラーとなる．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，CRE_ALM のみをサポートする．ただし，動的生成機能拡張パッケージでは，acre_alm もサポートする．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，CRE_ALM のみをサポートする．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，CRE_ALM のみをサポートする．

【μITRON4.0 仕様との関係】

almhdr のデータ型を ALMHDR に変更した．

AID_ALM　　割付け可能なアラームハンドラ ID の数の指定〔SD〕

【静的 API】

```
AID_ALM(uint_t noalm)
```

【パラメータ】

uint_t	noalm	割付け可能なアラームハンドラ ID の数

【エラーコード】

E_RSATR	予約属性（属する保護ドメインまたはクラスが不正）

【機能】

noalm で指定した数のアラームハンドラ ID を，アラームハンドラを生成するサービスコールによって割付け可能なアラームハンドラ ID として確保する．

noalm は整数定数式パラメータである．

SAC_ALM　　アラームハンドラのアクセス許可ベクタの設定〔SP〕
sac_alm　　アラームハンドラのアクセス許可ベクタの設定〔TPD〕

【静的 API】

```
SAC_ALM(ID almid, {ACPTN acptn1, ACPTN acptn2,
                   ACPTN acptn3, ACPTN acptn4 })
```

【C 言語 API】

```
ER ercd = sac_alm(ID almid, const ACVCT *p_acvct)
```

【パラメータ】

ID	almid	対象アラームハンドラの ID 番号
ACVCT *	p_acvct	アクセス許可ベクタを入れたパケットへのポインタ（静的 API を除く）

＊アクセス許可ベクタ（パケットの内容）

ACPTN	acptn1	通常操作 1 のアクセス許可パターン
ACPTN	acptn2	通常操作 2 のアクセス許可パターン
ACPTN	acptn3	管理操作のアクセス許可パターン
ACPTN	acptn4	参照操作のアクセス許可パターン

【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_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（almid が不正）
E_RSATR	予約属性（属する保護ドメインかクラスが不正：SAC_ALM の場合）
E_NOEXS〔D〕	オブジェクト未登録（対象アラームハンドラが未登録）
E_OACV〔sP〕	オブジェクトアクセス違反（対象アラームハンドラに対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（p_acvct が指すメモリ領域への読出しアクセスが許可されていない）
E_OBJ	オブジェクト状態エラー（対象アラームハンドラは静的 API で生成された：sac_alm の場合，対象アラームハンドラに対してアクセス許可ベクタが設定済み：SAC_ALM の場合）

【機能】

almid で指定したアラームハンドラ（対象アラームハンドラ）のアクセス許可ベクタ（4 つのアクセス許可パターンの組）を，各パラメータで指定した値に設定する．

静的 API においては，almid はオブジェクト識別名，acptn1～acptn4 は整数定数式パラメータである．

SAC_ALM は，対象アラームハンドラが属する保護ドメイン（この仕様ではカーネルドメインに限られる）の囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，SAC_ALM，sac_alm をサポートしない．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，SAC_ALM，sac_alm をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，SAC_ALM のみをサポートする．

del_alm　　アラームハンドラの削除〔TD〕

【C 言語 API】

```
ER ercd = del_alm(ID almid)
```

【パラメータ】

ID	almid	対象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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（almid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象アラームハンドラが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象アラームハンドラに対する管理操作が許可されていない）
E_OBJ	オブジェクト状態エラー（対象アラームハンドラは静的 API で生成された）

【機能】

almid で指定したアラームハンドラ（対象アラームハンドラ）を削除する．具体的な振舞いは以下の通り．

対象アラームハンドラの登録が解除され，そのアラームハンドラ ID が未使用の状態に戻される．対象アラームハンドラが動作している状態であった場合には，登録解除の前に，アラームハンドラが動作していない状態となる．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，del_alm をサポートしない．ただし，動的生成機能拡張パッケージでは，del_alm をサポートする．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，del_alm をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，del_alm をサポートしない．

---

sta_alm　　アラームハンドラの動作開始〔T〕
ista_alm　　アラームハンドラの動作開始〔I〕

【C 言語 API】

```
ER ercd = sta_alm(ID almid, RELTIM almtim)
ER ercd = ista_alm(ID almid, RELTIM almtim)
```

【パラメータ】

ID	almid	対象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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し：sta_alm の場合，タスクコンテキストからの呼出し：ista_alm の場合，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（almid が不正）
E_PAR	パラメータエラー（almtim が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象アラームハンドラが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象アラームハンドラに対する通常操作 1 が許可されていない：sta_alm の場合）

【機能】

almid で指定したアラームハンドラ（対象アラームハンドラ）を動作開始する．具体的な振舞いは以下

の通り.

対象アラームハンドラが動作していない状態であれば，対象アラームハンドラは動作している状態となる．アラームハンドラを起動する時刻は，sta_alm を呼び出してから，almtim で指定した相対時間後に設定される.

対象アラームハンドラが動作している状態であれば，アラームハンドラを起動する時刻の再設定のみが行われる.

almtim は，TMAX_RELTIM 以下でなければならない.

---

msta_alm　割付けプロセッサ指定でのアラームハンドラの動作開始〔TM〕
imsta_alm　割付けプロセッサ指定でのアラームハンドラの動作開始〔IM〕

【C 言語 API】

```
ER ercd = msta_alm(ID almid, RELTIM almtim, ID prcid)
ER ercd = imsta_alm(ID almid, RELTIM almtim, ID prcid)
```

【パラメータ】

ID	almid	対象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）
ID	prcid	アラームハンドラの割付け対象のプロセッサの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し：msta_alm の場合，タスクコンテキストからの呼出し：imsta_alm の場合，CPU ロック状態からの呼出し）
E_NOSPT	未サポート機能（グローバルタイマ方式を用いている場合）
E_ID	不正 ID 番号（almid，prcid が不正）
E_PAR	パラメータエラー（almtim が不正，対象アラームハンドラは prcid で指定したプロセッサに割り付けられない）
E_NOEXS〔D〕	オブジェクト未登録（対象アラームハンドラが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象アラームハンドラに対する通常操作 1 が許可されていない：msta_alm の場合）

【機能】

prcid で指定したプロセッサを割付けプロセッサとして，almid で指定したアラームハンドラ（対象アラームハンドラ）を動作開始する．具体的な振舞いは以下
の通り．

対象アラームハンドラが動作していない状態であれば，対象アラームハンドラの割付けプロセッサが prcid で指定したプロセッサに変更された後，対象アラームハンドラは動作している状態となる．アラームハンドラを起動する時刻は，msta_alm を呼び出してから，almtim で指定した相対時間後に設定される．

対象アラームハンドラが動作している状態であれば，対象アラームハンドラの割付けプロセッサが prcid で指定したプロセッサに変更された後，アラームハンドラを起動する時刻の再設定が行われる．

対象アラームハンドラが実行中である場合には，割付けプロセッサを変更しても，実行中のアラームハンドラを実行するプロセッサは変更されない．対象アラームハンドラが変更後の割付けプロセッサで実行されるのは，次に起動される時からである．

対象アラームハンドラの属するクラスの割付け可能プロセッサが，prcid で指定したプロセッサを含んでいない場合には，E_PAR エラーとなる．

prcid に TPRC_INI（＝0）を指定すると，対象アラームハンドラの割付けプロセッサを，それが属するクラスの初期割付けプロセッサとする．

almtim は，TMAX_RELTIM 以下でなければならない．

グローバルタイマ方式を用いている場合，msta_alm／imsta_alm は E_NOSPT を返す．

【使用上の注意】

msta_alm／imsta_alm で実行中のアラームハンドラの割付けプロセッサを変更した場合，同じアラームハンドラが異なるプロセッサで同時に実行される可能性がある．特に，almtim に 0 を指定する場合に，注意が必要である．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様に定義されていないサービスコールである．

---

stp_alm　アラームハンドラの動作停止〔T〕
istp_alm　アラームハンドラの動作停止〔I〕

【C 言語 API】

```
ER ercd = stp_alm(ID almid)
ER ercd = istp_alm(ID almid)
```

【パラメータ】

ID	almid	対象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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し：stp_alm の場合，タスクコンテキストからの呼出し：istp_alm の場合，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（almid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象アラームハンドラが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象アラームハンドラに対する通常操作 2 が許可されていない：stp_alm の場合）

【機能】

almid で指定したアラームハンドラ（対象アラームハンドラ）を動作停止する．具体的な振舞いは以下の通り．

対象アラームハンドラが動作している状態であれば，動作していない状態となる．対象アラームハンドラが動作していない状態であれば，何も行われずに正常終了する．

ref_alm　　アラームハンドラの状態参照〔T〕

【C 言語 API】

```
ER ercd = ref_alm(ID almid, T_RALM *pk_ralm)
```

【パラメータ】

ID	almid	対象アラームハンドラの ID 番号
T_RALM *	pk_ralm	アラームハンドラの現在状態を入れる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

＊アラームハンドラの現在状態（パケットの内容）

STAT	almstat	アラームハンドラの動作状態
RELTIM	lefttim	アラームハンドラを起動する時刻までの相対時間
ID	prcid	アラームハンドラの割付けプロセッサの ID（マルチプロセッサ対応カーネルの場合）

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（almid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象アラームハンドラが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象アラームハンドラに対する参照操作が許可されていない
E_MACV〔P〕	メモリアクセス違反（pk_ralm が指すメモリ領域への書込みアクセスが許可されていない）

【機能】

almid で指定したアラームハンドラ（対象アラームハンドラ）の現在状態を参照する．参照した現在状態は，pk_ralm で指定したパケットに返される．

almstat には，対象アラームハンドラの現在の動作状態を表す次のいずれかの値が返される．

TALM_STP	0x01U	アラームハンドラが動作していない状態
TALM_STA	0x02U	アラームハンドラが動作している状態

対象アラームハンドラが動作している状態である場合には，lefttim に，アラームハンドラ起動する時刻までの相対時間が返される．対象アラームハンドラが動作していない状態である場合には，lefttim の値は保証されない．

マルチプロセッサ対応カーネルでは，prcid に，対象アラームハンドラの割付けプロセッサの ID 番号が返される．

【使用上の注意】
ref_alm はデバッグ時向けの機能であり，その他の目的に使用することは推奨しない．これは，ref_alm を呼び出し，対象アラームハンドラの現在状態を参照した直後に割込みが発生した場合，ref_alm から戻ってきた時には対象アラームハンドラの状態が変化している可能性があるためである．

【μITRON4.0 仕様との関係】
TALM_STP と TALM_STA を値を変更した．

4.6.4. オーバランハンドラ

オーバランハンドラは，タスクが使用したプロセッサ時間が，指定した時間を超え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は，システムで 1 つのみ登録することができる．

オーバランハンドラ機能に関連して，各タスクが持つ情報は次の通り．

- オーバランハンドラの動作状態
- 残りプロセッサ時間

オーバランハンドラの動作状態は，タスク毎に，動作している状態と動作していない状態のいずれかをとる．残りプロセッサ時間は，オーバランハンドラが動作している状態の時に，タスクが使用できる残りのプロセッサ時間を表す．

オーバランハンドラの動作状態は，タスクの起動時に，動作していない状態に初期化される．

残りプロセッサ時間は，オーバランハンドラが動作している状態でタスクが実行している間，タスクが使用したプロセッサ時間の分だけ減少する．残りプロセッサ時間が 0 になると（これをオーバランと呼ぶ），オーバランハンドラが起動される．

タスクが使用したプロセッサ時間には，そのタスク自身とタスク例外処理ルーチン，それらから呼び出したサービルコール（拡張サービスコールを含む）の
実行時間を含む．一方，タスクの実行中に起動されたカーネル管理の割込みハンドラ（割込みサービスルーチン，周期ハンドラ，アラームハンドラ，オーバランハンドラの実行時間を含む）とカーネル管理の CPU 例外ハンドラの実行時間は含まないが，割込みハンドラおよび CPU 例外ハンドラの呼出し／復帰にかかる時間と，それらの入口処理と出口処理の一部の実行時間は含んでしまう．また，タスクの実行中に起動されたカーネル管理外の割込みハンドラとカーネル管理外の CPU 例外ハンドラの実行時間も含む．

プロセッサ時間は，符号無しの整数型である OVRTIM 型で表し，単位はマイクロ秒とする．ただし，プロセッサ時間には，OVRTIM 型に格納できる任意の値を指定できるとは限らず，指定できる値にターゲット定義の上限がある場合がある．プロセッサ時間に指定できる最大値は，構成マクロ TMAX_OVRTIM に定義されている．また，タスクが使用したプロセッサ時間の計測精度はターゲットに依存する．

保護機能対応カーネルにおいて，オーバランハンドラは，カーネルドメインに属する．

ターゲット定義で，オーバランハンドラ機能がサポートされていない場合がある．オーバランハンドラ機能がサポートされている場合には，TOPPERS_SUPPORT_OVRHDR がマクロ定義される．サポートされていない場合にオーバランハンドラ機能のサービスコールを呼び出すと，E_NOSPT エラーが返るか，リンク時にエラーとなる．

オーバランハンドラ機能に用いるデータ型は次の通り．

OVRTIM	プロセッサ時間（符号無し整数，単位はマイクロ秒，ulong_t に定義）

オーバランハンドラ属性に指定できる属性はない．そのためオーバランハンドラ属性には，TA_NULL を指定しなければならない．

C 言語によるオーバランハンドラの記述形式は次の通り．

```
void overrun_handler(ID tskid, intptr_t exinf)
{
    オーバランハンドラ本体
}
```

tskid にはオーバランを起こしたタスクの ID 番号が，exinf にはそのタスクの拡張情報が，それぞれ渡

される．

オーバランハンドラ機能に関連するカーネル構成マクロは次の通り．

TMAX_OVRTIM	プロセッサ時間に指定できる最大値

TOPPERS_SUPPORT_OVRHDR	オーバランハンドラ機能がサポートされている

【使用上の注意】
マルチプロセッサ対応カーネルでは，オーバランハンドラが異なるプロセッサで同時に実行される可能性があるので，注意が必要である．

【TOPPERS/ASP カーネルにおける規定】
ASP カーネルでは，オーバランハンドラをサポートしない．ただし，オーバランハンドラ機能拡張パッケージを用いると，オーバランハンドラ機能を追加することができる．

【TOPPERS/FMP カーネルにおける規定】
FMP カーネルでは，オーバランハンドラをサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，オーバランハンドラをサポートする．

【μITRON4.0 仕様との関係】
OVRTIM の時間単位は，μITRON4.0 仕様では実装定義としていたが，この仕様ではマイクロ秒と規定した．

TMAX_OVRTIM は，μITRON4.0 仕様に規定されていないカーネル構成マクロである．

DEF_OVR　オーバランハンドラの定義〔S〕
def_ovr　オーバランハンドラの定義〔TD〕

【静的 API】

```
DEF_OVR({ ATR ovratr, OVRHDR ovrhdr })
```

【C 言語 API】

```
ER ercd = def_ovr(const T_DOVR *pk_dovr)
```

【パラメータ】

T_DOVR *	pk_dovr	オーバランハンドラの定義情報を入れたパケットへのポインタ（静的 API を除く）

＊オーバランハンドラの定義情報（パケットの内容）

ATR	ovratr	オーバランハンドラ属性
OVRHDR	ovrhdr	オーバランハンドラの先頭番地

【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_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_RSATR	予約属性（ovratr が不正または使用できない，属する保護ドメインかクラスが不正）
E_OACV〔sP〕	オブジェクトアクセス違反（システム状態に対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（pk_dovr が指すメモリ領域への読出しアクセスが許可されていない）
E_PAR	パラメータエラー（ovrhdr が不正）
E_OBJ	オブジェクト状態エラー（条件については機能の項を参照すること）

【機能】

各パラメータで指定したオーバランハンドラ定義情報に従って，オーバランハンドラを定義する．ただし，def_ovr において pk_dovr を NULL にした場合には，オーバランハンドラの定義を解除する．

静的 API においては，ovratr は整数定数式パラメータ，ovrhdr は一般定数式パラメータである．

オーバランハンドラを定義する場合（DEF_OVR の場合および def_ovr において pk_dovr を NULL 以外にした場合）で，すでにオーバランハンドラが定義されている場合には，E_OBJ エラーとなる．

保護機能対応カーネルにおいて，DEF_OVR は，カーネルドメインの囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．また，def_ovr でオーバランハンドラを定義する場合には，オーバランハンドラの属する保護ドメインを設定する必要はなく，オーバランハンドラ属性に TA_DOM(domid)を指定した場合には E_RSATR エラーとなる．ただし，TA_DOM(TDOM_SELF)を指定した場合には，指定が無視され，E_RSATR エラーは検出されない．

マルチプロセッサ対応カーネルでは，DEF_OVR は，クラスの囲みの外に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．また，def_ovr でオーバランハンドラを定義する場合には，オーバランハンドラの属するクラスを設定する必要はなく，オーバランハンドラ属性に TA_CLS(clsid)を指定した場合には E_RSATR エラーとなる．ただし，TA_CLS(TCLS_SELF)を指定した場合には，指定が無視され，E_RSATR エラーは検出されない．

オーバランハンドラの定義を解除する場合（def_ovr において pk_dovr を NULL にした場合）で，オーバランハンドラが定義されていない場合には，E_OBJ エラーとなる．

オーバランハンドラの定義を解除すると，オーバランハンドラの動作状態は，すべてのタスクに対して動作していない状態となる．

【使用上の注意】

def_ovr によりオーバランハンドラの定義を解除する場合，サービスコールの処理時間およびカーネル内での割込み禁止時間が，タスクの総数に比例して長くなる．特に，タスクの総数が多い場合，カーネル内での割込み禁止時間が長くなるため，注意が必要である．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルのオーバランハンドラ機能拡張パッケージでは，DEF_OVR のみをサポートする．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，DEF_OVR のみをサポートする．

【μITRON4.0 仕様との関係】

ovrhdr のデータ型を OVRHDR に変更した．

def_ovr によって定義済みのオーバランハンドラを再定義しようとした場合に，E_OBJ エラーとすることにした．オーバランハンドラの定義を変更するには，一度定義を解除してから，再度定義する必要がある．

---

sta_ovr　オーバランハンドラの動作開始〔T〕
ista_ovr　オーバランハンドラの動作開始〔I〕

【C 言語 API】

```
ER ercd = sta_ovr(ID tskid, OVRTIM ovrtim)
ER ercd = ista_ovr(ID tskid, OVRTIM ovrtim)
```

【パラメータ】

ID	tskid	対象タスクの ID 番号
OVRTIM	almtim	対象タスクの残りプロセッサ時間

【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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し：sta_ovr の場合，タスクコンテキストからの呼出し：ista_ovr の場合，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（tskid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象タスクが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象タスクに対する通常操作 2 が許可されていない：sta_ovr の場合）
E_PAR	パラメータエラー（ovrtim が不正）

【機能】

tskid で指定したタスク（対象タスク）に対して，オーバランハンドラの動作を開始する．具体的な振舞いは以下の通り．

対象タスクに対するオーバランハンドラの動作状態は，動作している状態となり，残りプロセッサ時間は，ovrtim に指定した時間に設定される．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状態であれば，残りプロセッサ時間の設定のみが行われる．

sta_ovr において tskid に TSK_SELF（＝0）を指定すると，自タスクが対象タスクとなる．

ovrtim は，0 より大きく，TMAX_OVRTIM 以下の値でなければならない．

【μITRON4.0 仕様との関係】

ista_ovr は，μITRON4.0 仕様に定義されていないサービスコールである．

stp_ovr　　オーバランハンドラの動作停止〔T〕
istp_ovr　　オーバランハンドラの動作停止〔I〕

【C 言語 API】

```
ER ercd = stp_ovr(ID tskid)
ER ercd = istp_ovr(ID tskid)
```

【パラメータ】

ID	tskid	対象タスクの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し：stp_ovr の場合，タスクコンテキストからの呼出し：istp_ovr の場合，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（tskid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象タスクが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象タスクに対する通常操作 2 が許可されていない：stp_ovr の場合）
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 で指定したタスク（対象タスク）に対して，オーバランハンドラの動作を停止する．具体的な振舞いは以下の通り．

対象タスクに対するオーバランハンドラの動作状態は，動作していない状態となる．対象タスクに対してオーバランハンドラが動作していない状態であれば，何も行われずに正常終了する．

stp_ovr において tskid に TSK_SELF（＝0）を指定すると，自タスクが対象タスクとなる．

【μITRON4.0 仕様との関係】

istp_ovr は，μITRON4.0 仕様に定義されていないサービスコールである．

ref_ovr　　オーバランハンドラの状態参照〔T〕

【C 言語 API】

```
ER ercd = ref_ovr(ID tskid, T_ROVR *pk_rovr)
```

【パラメータ】

ID	tskid	対象タスクの ID 番号
T_ROVR	pk_rovr	オーバランハンドラの現在状態を入れる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

＊タスクの現在状態（パケットの内容）

STAT	ovrstat	オーバランハンドラの動作状態
OVRTIM	leftotm	残りプロセッサ時間

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（tskid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象タスクが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象タスクに対する参照操作が許可されていない）
E_MACV〔P〕	メモリアクセス違反（pk_rovr が指すメモリ領域への書込みアクセスが許可されていない）
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 で指定したタスク（対象タスク）に対するオーバランハンドラの現在状態を参照する．参照した現在状態は，pk_rovr で指定したメモリ領域に返される．

ovrstat には，対象タスクに対するオーバランハンドラの動作状態を表す次のいずれかの値が返される．

TOVR_STP	0x01U	オーバランハンドラが動作していない状態
TOVR_STA	0x02U	オーバランハンドラが動作している状態

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状態の場合には，leftotm に，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が返される．オーバランハンドラが起動される直前には，leftotm に 0 が返される可能性がある．オーバランハンドラが動作していない状態の場合には，leftotm の値は保証されない．

tskid に TSK_SELF（＝0）を指定すると，自タスクが対象タスクとなる．

【使用上の注意】

ref_ovr はデバッグ時向けの機能であり，その他の目的に使用することは推奨しない．これは，ref_ovr を呼び出し，対象オーバランハンドラの現在状態を参照した直後に割込みが発生した場合，ref_ovr から戻ってきた時には対象オーバランハンドラの状態が変化している可能性があるためである．

【未決定事項】

マルチプロセッサ対応カーネルにおいて，対象タスクが，自タスクが割付けられたプロセッサと異なるプロセッサに割り付けられている場合に，leftotm を参照できるとするかどうかは，今後の課題である．

【μITRON4.0 仕様との関係】

TOVR_STP と TOVR_STA を値を変更した．

4.7. システム状態管理機能

システム状態管理機能は，特定のオブジェクトに関連しないシステムの状態を変更／参照するための機能である．

SAC_SYS　　システム状態のアクセス許可ベクタの設定〔SP〕
sac_sys　　システム状態のアクセス許可ベクタの設定〔TPD〕

【静的 API】

```
SAC_SYS({ACPTN acptn1, ACPTN acptn2,
                    ACPTN acptn3, ACPTN acptn4 })
```

【C 言語 API】

```
ER ercd = sac_sys(const ACVCT *p_acvct)
```

【パラメータ】

ACVCT *	p_acvct	アクセス許可ベクタを入れたパケットへのポインタ（静的 API を除く）

＊アクセス許可ベクタ（パケットの内容）

ACPTN	acptn1	通常操作 1 のアクセス許可パターン
ACPTN	acptn2	通常操作 2 のアクセス許可パターン
ACPTN	acptn3	管理操作のアクセス許可パターン
ACPTN	acptn4	参照操作のアクセス許可パターン

【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_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_RSATR	予約属性（属する保護ドメインかクラスが不正：SAC_SYS の場合）
E_OACV〔sP〕	オブジェクトアクセス違反（カーネルドメイン以外からの呼出し）
E_MACV〔sP〕	メモリアクセス違反（p_acvct が指すメモリ領域への読出しアクセスが許可されていない）
E_OBJ	オブジェクト状態エラー（システム状態のアクセス許可ベクタが設定済み：SAC_SYS の場合）

【機能】

システム状態のアクセス許可ベクタ（4 つのアクセス許可パターンの組）を，各パラメータで指定した値に設定する．

静的 API においては，acptn1～acptn4 は整数定数式パラメータである．

SAC_SYS は，カーネルドメインの囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，SAC_SYS，sac_sys をサポートしない．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，SAC_SYS，sac_sys をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，SAC_SYS のみをサポートする．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，SAC_SYS，sac_sys をサポートしない．

---

rot_rdq　タスクの優先順位の回転〔T〕
irot_rdq　タスクの優先順位の回転〔I〕

【C 言語 API】

```
ER ercd = rot_rdq(PRI tskpri)
ER ercd = irot_rdq(PRI tskpri)
```

【パラメータ】

PRI	tskpri	回転対象の優先度（対象優先度）

【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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し：rot_rdq の場合，タスクコンテキストからの呼出し：irot_rdq の場合，CPU ロック状態からの呼出し）
E_NOSPT	未サポート機能（対象優先度の最も優先順位が高いタスクが制約タスク）
E_PAR	パラメータエラー（tskpri が不正）
E_OACV〔P〕	オブジェクトアクセス違反（システム状態に対する通常操作 1 が許可されていない）

【機能】

tskpri で指定した優先度（対象優先度）を持つ実行できる状態のタスクの中で，最も優先順位が高いタスクを，同じ優先度のタスクの中で最も優先順位が低い状態にする．対象優先度を持つ実行できる状態のタスクが無いか 1 つのみの場合には，何も行われずに正常終了する．

rot_rdq において，tskpri に TPRI_SELF（＝0）を指定すると，自タスクのベース優先度が対象優先度となる．

対象優先度を持つ実行できる状態のタスクの中で，最も優先順位が高いタスクが制約タスクの場合には，E_NOSPT エラーとなる．

tskpri は，TPRI_SELF であるか（rot_rdq の場合のみ），TMIN_TPRI 以上，TMAX_TPRI 以下でなければならない．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，rot_rdq，irot_rdq をサポートしない．

---

mrot_rdq　プロセッサ指定でのタスクの優先順位の回転〔TM〕
imrot_rdq　プロセッサ指定でのタスクの優先順位の回転〔IM〕

【C 言語 API】

```
ER ercd = mrot_rdq(PRI tskpri, ID prcid)
ER ercd = imrot_rdq(PRI tskpri, ID prcid)
```

【パラメータ】

PRI	tskpri	回転対象の優先度（対象優先度）
ID	prcid	優先順位の回転対象とするプロセッサの 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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し：mrot_rdq の場合，タスクコンテキストからの呼出し：imrot_rdq の場合，CPU ロック状態からの呼出し）
E_NOSPT	未サポート機能（対象優先度の最も優先順位が高いタスクが制約タスク）
E_ID	不正 ID 番号（prcid が不正）
E_PAR	パラメータエラー（tskpri が不正）
E_OACV〔P〕	オブジェクトアクセス違反（システム状態に対する通常操作 1 が許可されていない）

【機能】

prcid で指定したプロセッサに割り付けられており，tskpri で指定した優先度（対象優先度）を持つ実行できる状態のタスクの中で，最も優先順位が高いタスクを，同じ優先度のタスクの中で最も優先順位が低い状態にする．対象優先度を持つ実行できる状態のタスクが無いか 1 つのみの場合には，何も行わ

れずに正常終了する.

mrot_rdq において，tskpri に TPRI_SELF（＝0）を指定すると，自タスクのベース優先度が対象優先度となる.

prcid で指定したプロセッサに割り付けられており，対象優先度を持つ実行できる状態のタスクの中で，最も優先順位が高いタスクが制約タスクの場合には，E_NOSPT エラーとなる.

tskpri は，TPRI_SELF であるか（mrot_rdq の場合のみ），TMIN_TPRI 以上，TMAX_TPRI 以下でなければならない.

【TOPPERS/ASP カーネルにおける規定】
ASP カーネルでは，mrot_rdq，imrot_rdq をサポートしない.

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，mrot_rdq，imrot_rdq をサポートしない.

【TOPPERS/SSP カーネルにおける規定】
SSP カーネルでは，mrot_rdq，imrot_rdq をサポートしない.

【μITRON4.0 仕様との関係】
μITRON4.0 仕様に定義されていないサービスコールである.

---

get_tid　実行状態のタスク ID の参照〔T〕
iget_tid　実行状態のタスク ID の参照〔I〕

【C 言語 API】

```
ER ercd = get_tid(ID *p_tskid)
ER ercd = iget_tid(ID *p_tskid)
```

【パラメータ】

ID *	p_tskid	タスク ID を入れるメモリ領域へのポインタ

【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
ID	tskid	タスク ID

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し：get_tid の場合，タスクコンテキストからの呼出し：iget_tid の場合，CPU ロック状態からの呼出し）
E_MACV〔P〕	メモリアクセス違反（p_tskid が指すメモリ領域への書込みアクセスが許可されていない）

【機能】

実行状態のタスク（get_tid の場合には自タスク）の ID 番号を参照する．参照したタスク ID は，p_tskid で指定したメモリ領域に返される．

iget_tid において，実行状態のタスクがない場合には，TSK_NONE（＝0）が返される．

マルチプロセッサ対応カーネルにおいては，サービスコールを呼び出した処理単位を実行しているプロセッサにおいて実行状態のタスクの ID 番号を参照する．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，get_tid をサポートしない．

get_did　　実行状態のタスクが属する保護ドメイン ID の参照〔TP〕

【C 言語 API】

```
ER ercd = get_did(ID *p_domid)
```

【パラメータ】

ID *	p_domid	保護ドメイン ID を入れるメモリ領域へのポインタ

【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
ID	domid	保護ドメイン ID

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_MACV	メモリアクセス違反（p_domid が指すメモリ領域への書込みアクセスが許可されていない）

【機能】

実行状態のタスク（自タスク）が属する保護ドメインの ID 番号を参照する．参照した保護ドメイン ID は，p_domid で指定したメモリ領域に返される．

マルチプロセッサ対応カーネルにおいては，サービスコールを呼び出した処理単位を実行しているプロセッサにおいて実行状態のタスクが属する保護ドメインの ID 番号を参照する．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，get_did をサポートしない．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，get_did をサポートしない．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，get_did をサポートしない．

---

get_pid　　割付けプロセッサの ID 番号の参照〔TM〕
iget_pid　　割付けプロセッサの ID 番号の参照〔IM〕

【C 言語 API】

```
ER ercd = get_pid(ID *p_prcid)
ER ercd = iget_pid(ID *p_prcid)
```

【パラメータ】

ID *	p_prcid	プロセッサ ID を入れるメモリ領域へのポインタ

【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
ID	prcid	プロセッサ ID

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し：get_pid の場合，タスクコンテキストからの呼出し：iget_pid の場合，CPU ロック状態からの呼出し）
E_MACV〔P〕	メモリアクセス違反（p_prcid が指すメモリ領域への書込みアクセスが許可されていない）

【機能】

サービスコールを呼び出した処理単位の割付けプロセッサの ID 番号を参照する．参照したプロセッサ ID は，p_prcid で指定したメモリ領域に返される．

【使用上の注意】

タスクは, get_pid を用いて, 自タスクの割付けプロセッサを正しく参照できるとは限らない. これは, get_pid を呼び出し，自タスクの割付けプロセッサの ID 番号を参照した直後に割込みが発生した場合, get_pid から戻ってきた時には割付けプロセッサが変化している可能性があるためである．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，get_pid，iget_pid をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，get_pid，iget_pid をサポートしない．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，get_pid，iget_pid をサポートしない．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様に定義されていないサービスコールである．

---

loc_cpu　CPU ロック状態への遷移〔T〕
iloc_cpu　CPU ロック状態への遷移〔I〕

【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】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し：loc_cpu の場合，タスクコンテキストからの呼出し：iloc_cpu の場合，CPU ロック状態からの呼出し）
E_OACV〔P〕	オブジェクトアクセス違反（システム状態に対する通常操作 2 が許可されていない：loc_cpu の場合）

【機能】

CPU ロックフラグをセットし, CPU ロック状態へ遷移する. CPU ロック状態で呼び出した場合には, 何も行われずに正常終了する.

---

unl_cpu　CPU ロック状態の解除〔T〕
iunl_cpu　CPU ロック状態の解除〔I〕

【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】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し：unl_cpu の場合，タスクコンテキストからの呼出し：iunl_cpu の場合，CPU ロック状態からの呼出し）
E_OACV〔P〕	オブジェクトアクセス違反（システム状態に対する通常操作 2 が許可されていない：unl cpu の場合）

【機能】

CPU ロックフラグをクリアし，CPU ロック解除状態へ遷移する．CPU ロック解除状態で呼び出した場合には，何も行われずに正常終了する．

マルチプロセッサ対応カーネルにおいて，unl_cpu／iunl_cpu を呼び出したプロセッサによって取得されている状態となっているスピンロックがある場合には，unl_cpu／iunl_cpu によって CPU ロック解除状態に遷移しない（何も行われずに正常終了する）．

【補足説明】

マルチプロセッサ対応カーネルでは，CPU ロック解除状態へ遷移した結果，ディスパッチ保留状態が解除され，ディスパッチが起こる可能性がある．また，保護機能対応カーネルとマルチプロセッサ対応カーネルでは，タスク例外処理ルーチンの実行が開始される可能性がある．

dis_dsp　　ディスパッチの禁止〔T〕

【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】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_OACV〔P〕	オブジェクトアクセス違反（システム状態に対する通常操作 1 が許可されていない）

【機能】

ディスパッチ禁止フラグをセットし，ディスパッチ禁止状態へ遷移する．ディスパッチ禁止状態で呼び出した場合には，何も行われずに正常終了する．

ena_dsp　　ディスパッチの許可〔T〕

【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】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_OACV〔P〕	オブジェクトアクセス違反（システム状態に対する通常操作 1 が許可されていない）

【機能】

ディスパッチ禁止フラグをクリアし，ディスパッチ許可状態へ遷移する．ディスパッチ許可状態で呼び出した場合には，何も行われずに正常終了する．

【補足説明】

ディスパッチ許可状態へ遷移した結果，ディスパッチ保留状態が解除され，ディスパッチが起こる可能性がある．

---

sns_ctx　　コンテキストの参照〔TI〕

【C 言語 API】

```
bool_t state = sns_ctx()
```

【パラメータ】

なし

【リターンパラメータ】

bool_t	state	コンテキスト

【機能】

実行中のコンテキストを参照する．具体的な振舞いは以下の通り．

sns_ctx を非タスクコンテキストから呼び出した場合には true, タスクコンテキストから呼び出した場合には false が返る．

sns_loc　　CPU ロック状態の参照〔TI〕

【C 言語 API】

```
bool_t state = sns_loc()
```

【パラメータ】

なし

【リターンパラメータ】

bool_t	state	CPU ロックフラグ

【機能】

CPU ロックフラグを参照する．具体的な振舞いは以下の通り．

sns_loc を CPU ロック状態で呼び出した場合には true，CPU ロック解除状態で呼び出した場合には false が返る．

sns_dsp　　ディスパッチ禁止状態の参照〔TI〕

【C 言語 API】

```
bool_t state = sns_dsp()
```

【パラメータ】

なし

【リターンパラメータ】

bool_t	state	ディスパッチ禁止フラグ

【機能】

ディスパッチ禁止フラグを参照する．具体的な振舞いは以下の通り．

sns_dsp をディスパッチ禁止状態で呼び出した場合には true，ディスパッチ許可状態で呼び出した場合には false が返る．

sns_dpn　　ディスパッチ保留状態の参照〔TI〕

【C 言語 API】

```
bool_t state = sns_dpn()
```

【パラメータ】

なし

【リターンパラメータ】

bool_t	state	ディスパッチ保留状態

【機能】

ディスパッチ保留状態であるか否かを参照する．具体的な振舞いは以下の通り．

sns_dpn をディスパッチ保留状態で呼び出した場合には true，ディスパッチ保留状態でない状態で呼び出した場合には false が返る．

sns_ker　　カーネル非動作状態の参照〔TI〕

【C 言語 API】

```
bool_t state = sns_ker()
```

【パラメータ】

なし

【リターンパラメータ】

bool_t	state	カーネル非動作状態

【機能】

カーネルが動作中であるか否かを参照する．具体的な振舞いは以下の通り．

sns_ker をカーネルの初期化完了前（初期化ルーチン実行中を含む）または終了処理開始後（終了処理ルーチン実行中を含む）に呼び出した場合には true，カーネルの動作中に呼び出した場合には false が返る．

【使用方法】

sns_ker は，カーネルが動作している時とそうでない時で，処理内容を変えたい場合に使用する．sns_ker が true を返した場合，他のサービスコールを呼び出すことはできない．sns_ker が true を返す時に他のサービスコールを呼び出した場合の動作は保証されない．

【使用上の注意】

どちらの条件で true が返るか間違いやすいので注意すること．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様に定義されていないサービスコールである．

---

ext_ker　　カーネルの終了〔TI〕

【C 言語 API】

```
ER ercd = ext_ker()
```

【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_SYS	システムエラー（カーネルの誤動作
E_OACV〔P〕	オブジェクトアクセス違反（カーネルドメイン以外からの呼出し）

【機能】

カーネルを終了する．具体的な振舞いについては，「2.9.2 システム終了手順」の節を参照すること．

ext_ker が正常に処理された場合，ext_ker からはリターンしない．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様に定義されていないサービスコールである．

ref_sys　　システムの状態参照〔T〕

【C 言語 API】

ER ercd = ref_sys(T_RSYS *pk_rsys)

☆未完成

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，ref_sys をサポートしない．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，ref_sys をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，ref_sys をサポートしない．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，ref_sys をサポートしない．

4.8. メモリオブジェクト管理機能

メモリオブジェクト管理機能は，保護機能対応カーネルでのみサポートされる機能である．保護機能対応でないカーネルでは，メモリオブジェクト管理機能をサポートしない．

〔メモリリージョン〕

メモリリージョンは，オブジェクトモジュールに含まれるセクションの配置対象となる同じ性質を持った連続したメモリ領域である．メモリリージョンは，メモリリージョン名によって識別する．

各メモリリージョンが持つ情報は次の通り．

- 先頭番地
- サイズ
- メモリリージョン属性

メモリリージョンの先頭番地とサイズには，ターゲット定義の制約が課せられる場合がある．

メモリリージョン属性には，次の属性を指定することができる．

TA_NOWRITE	0x01U	書込みアクセス禁止
TA_STDROM	0x02U	標準 ROM リージョン
TA_STDRAM	0x04U	標準 RAM リージョン

標準 ROM リージョン（TA_STDROM 属性が指定されたメモリリージョン）は，ATT_MOD／ATA_MOD において，書込みアクセスを行わない標準のセクションを配置するメモリリージョンである．また，標準 RAM リージョン（TA_STDRAM 属性が指定されたメモリリージョン）は，ATT_MOD／ATA_MOD において，書込みアクセスを行う標準のセクションを配置するメモリリージョンである．

標準 ROM リージョンは，書込みアクセス禁止と扱われる．すなわち，TA_STDROM 属性を指定すれば，TA_NOWRITE 属性も指定したことになる（TA_NOWRITE 属性を指定しても指定しなくても，同じ振舞いとなる）．

ターゲットによっては，ターゲット定義のメモリリージョン属性を指定できる場合がある．

〔メモリオブジェクト〕

メモリオブジェクトは，保護機能対応カーネルにおいてアクセス保護の対象とする連続したメモリ領域である．メモリオブジェクトは，その先頭番地によって識別する．

各メモリオブジェクトが持つ情報は次の通り．

- 先頭番地
- サイズ
- メモリオブジェクト属性
- アクセス許可ベクタ
- 属する保護ドメイン
- 属するクラス（マルチプロセッサ対応カーネルの場合）

メモリオブジェクトの先頭番地とサイズには，ターゲット定義の制約が課せられる．

メモリオブジェクト属性には，次の属性を指定することができる．

TA_NOWRITE	0x01U	書込みアクセス禁止
TA_NOREAD	0x02U	読出しアクセス禁止
TA_EXEC	0x04U	実行アクセス許可
TA_MEMINI	0x08U	メモリの初期化を行う
TA_MEMPRSV	0x10U	メモリの初期化を行わない
TA_SDATA	0x20U	ショートデータ領域に配置
TA_UNCACHE	0x40U	キャッシュ禁止
TA_IODEV	0x80U	周辺デバイスの領域

メモリオブジェクトに対して書込みアクセスできるのは，メモリオブジェクト属性に書込みアクセス禁止（TA_NOWRITE 属性）が指定されておらず，アクセス許可ベクタにより書込みアクセスが許可されている場合である．また，読出しアクセスできるのは，メモリオブジェクト属性に読出しアクセス禁止（TA_NOREAD 属性）が指定されておらず，アクセス許可ベクタにより読出し・実行アクセスが許可されている場合である．実行アクセスできるのは，メモリオブジェクト属性に実行アクセス許可（TA_EXEC 属性）が指定されており，アクセス許可ベクタにより読出し・実行アクセスが許可されている場合である．

ただし，ターゲットハードウェアの制約によってこれらの属性を実現できない場合には，次のように扱われる．書込みアクセス禁止が実現できない場合には，TA_NOWRITE を指定しても無視される．また，読出しアクセス禁止が実現できない場合には，TA_NOREAD を指定しても無視される．実行アクセス禁止が実現できない場合には，TA_EXEC を指定しなくても実行アクセス許可となり，TA_EXEC は無視される．どのような場合にどの属性の指定が無視されるかは，ターゲット定義である．

TA_MEMINI 属性は，システム初期化時に初期化するメモリオブジェクトであることを，TA_MEMPRSV 属性は，システム初期化時に初期化を行わないメモリオブジェクトであることを示す．いずれの属性も指定しない場合，そのメモリオブジェクトは，システム初期化時にクリア（言い換えると，0 に初期化）される．TA_MEMINI と TA_MEMPRSV の両方を指定した場合には，E_RSATR エラーとなる．

TA_NOWRITE が指定されている場合には，TA_MEMINI と TA_MEMPRSV は無視される（指定しても指定しなくても，同じ振舞いとなる）．

TA_MEMINI 属性を設定したメモリオブジェクトを初期化に用いる初期化データは，標準 ROM リージョンに配置され，メモリオブジェクトとしては登録されない．

TA_SDATA 属性は，メモリオブジェクトをショートデータ領域に配置することを示す．具体的な扱いはターゲット定義であるが，ショートデータ領域がサポートされていないターゲットでは，この属性は無視される．また，ターゲットによっては，TA_NOWRITE を指定した場合に，TA_SDATA が無視され

る場合がある．

TA_UNCACHE 属性は，メモリオブジェクトをキャッシュ禁止に設定することを，TA_IODEV 属性は，メモリオブジェクトを周辺デバイスの領域として扱うことを示す．これらの属性を指定しても意味がないターゲット（例えば，キャッシュを持たないターゲットプロセッサでの TA_UNCACHE）では，これらの属性は無視される．逆に，キャッシュ禁止にできないメモリオブジェクトに対して TA_UNCACHE を指定した場合や，周辺デバイスの領域として扱うことができないメモリオブジェクトに対して TA_IODEV を指定した場合には，E_RSATR エラーとなる．

ターゲットによっては，ターゲット定義のメモリオブジェクト属性を指定できる場合がある．ターゲット定義のメモリオブジェクト属性として，次の属性を予約している．

TA_WTHROUGH	－	ライトスルーキャッシュを用いる

〔カーネル構成マクロ〕

メモリオブジェクト管理機能に関連するカーネル構成マクロは次の通り．

TOPPERS_SUPPORT_ATT_MOD	ATT_MOD／ATA_MOD がサポートされている
TOPPERS_SUPPORT_ATT_PMA	ATT_PMA／ATA_PMA／att_pma がサポートされている

ただし，att_pma は，動的生成対応カーネルのみでサポートされる API であるため，サポートされているかを判定するには，TOPPERS_SUPPORT_DYNAMIC_CRE と TOPPERS_SUPPORT_ATT_PMA の両方が定義されていることをチェックする必要がある．

【補足説明】
メモリオブジェクトが属するクラスは，ATT_MOD／ATA_MOD において，標準のセクションが配置されるメモリリージョンを決定するためのみに使用される．

【TOPPERS/ASP カーネルにおける規定】
ASP カーネルでは，メモリオブジェクト管理機能をサポートしない．

【TOPPERS/FMP カーネルにおける規定】
FMP カーネルでは，メモリオブジェクト管理機能をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，メモリオブジェクト管理機能をサポートする．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，メモリオブジェクト管理機能をサポートしない．

【μITRON4.0/PX 仕様との関係】

値が 0 のメモリオブジェクト属性（TA_RW，TA_CACHE）は，デフォルトの扱いにして廃止した．TA_RO は TA_NOWRITE に改名し，TA_NOREAD，TA_EXEC，TA_MEMINI，TA_MEMPRSV，TA_IODEV を追加した．また，TA_UNCACHE の値を変更し，ターゲット定義のメモリオブジェクト属性として TA_WTHROUGH を予約した．

メモリリージョンは，μITRON4.0/PX 仕様にはない概念である．

【仕様決定の理由】

TA_IODEV 属性を導入したのは，ターゲットプロセッサによっては，周辺デバイスの領域として扱うためには，キャッシュ禁止に加えて，メモリのアクセス順序を変更しないことを指定しなければならないためである．メモリのアクセス順序を変更しないことを指定するメモリオブジェクト属性を，ターゲット定義で用意してもよいが，それを使うとアプリケーションのポータビリティが下がるため，TA_IODEV 属性を用意することにした．

ATT_REG　　メモリリージョンの登録〔SP〕

【静的 API】

```
ATT_REG("メモリリージョン名", {ATR regatr, void *base, SIZE size })
```

【パラメータ】

"メモリリージョン名"	－	登録するメモリリージョンを指定する文字列
ATR	regatr	メモリリージョン属性
void *	base	登録するメモリリージョンの先頭番地
SIZE	size	登録するメモリリージョンのサイズ（バイト数）

【エラーコード】

E_RSATR	予約属性（regatr が不正または使用できない，保護ドメインの囲みの中に記述）
E_PAR	パラメータエラー（メモリリージョン名，base，size が不正）
E_OBJ	オブジェクト状態エラー（登録済みのメモリリージョンの再登録，その他の条件については機能の項を参照すること）

【機能】

各パラメータで指定したメモリリージョン登録情報に従って，指定したメモリリージョンを登録する．具体的な振舞いは以下の通り．

base と size で指定したメモリ領域が，メモリリージョンとして登録される．登録されるメモリリージョンには，regatr で指定したメモリリージョン属性が設定される．

メモリリージョン名は文字列パラメータ，regatr，base，size は整数定数式パラメータである．

ATT_MOD／ATA_MOD がサポートされているターゲットでは，標準 ROM リージョンと標準 RAM リージョンを，それぞれ 1 つ登録しなければならない．標準 ROM リージョンと標準 RAM リージョンを登録していない場合や，複数登録した場合には，E_OBJ エラーとなる．

マルチプロセッサ対応カーネルでは，ATT_REG をクラスの囲みの中に記述するとそのクラス専用のメモリリージョンとなり，ATT_REG をクラスの囲みの外に記述すると共通のメモリリージョンとなる．ATT_MOD／ATA_MOD がサポートされているターゲットでは，共通の標準 ROM リージョンと共通の標準 RAM リージョンを，それぞれ 1 つ登録しなければならない．また，あるクラスの囲みの中で ATT_MOD／ATA_MOD を使用する場合には，そのクラス専用の標準 ROM リージョン／標準 RAM リージョンを 1 つ登録するか，共通の標準 ROM リージョン／標準 RAM リージョンを 1 つ登録しなければならない．これらを登録しない場合や，複数登録した場合には，E_OBJ エラーとなる．

mematr に，TA_STDROM と TA_STDRAM を同時に指定することはできない．指定した場合には，E_RSATR エラーとなる．

ATT_REG は，保護ドメインの囲みの外に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．

base や size に，ターゲット定義の制約に合致しない先頭番地やサイズを指定した時には，E_PAR エラーとなる．また，size が 0 の場合には，E_PAR エラーとなる．登録しようとしたメモリリージョンが，登録済みのメモリリージョンとメモリ領域が重なる場合には，E_OBJ エラーとなる．

【μITRON4.0/PX 仕様との関係】

μITRON4.0/PX 仕様に定義されていない静的 API である．

ATT_SEC　　セクションの登録〔SP〕
ATA_SEC　　セクションの登録（アクセス許可ベクタ付き）〔SP〕

【静的 API】

```
ATT_SEC("セクション名", {ATR mematr, "メモリリージョン名"})
```

```
ATA_SEC("セクション名", {ATR mematr, "メモリリージョン名"},
        {ACPTN acptn1, ACPTN acptn2, ACPTN acptn3, ACPTN acptn4})
```

【パラメータ】

"セクション名"	－	登録するセクションを指定する文字列
ATR	mematr	メモリオブジェクト属性
"メモリリージョン名"	－	セクションを配置するメモリリージョンを指定する文字列

＊アクセス許可ベクタ（パケットの内容）

ACPTN	acptn1	通常操作 1 のアクセス許可パターン
ACPTN	acptn2	通常操作 2 のアクセス許可パターン
ACPTN	acptn3	管理操作のアクセス許可パターン
ACPTN	acptn4	参照操作のアクセス許可パターン

【エラーコード】

E_RSATR	予約属性（mematr が不正または使用できない，属するクラスが不正）
E_PAR	パラメータエラー（セクション名，メモリリージョン名が不正）
E_OBJ	オブジェクト状態エラー（登録済みのセクションの再登録）

【機能】

各パラメータで指定した情報に従って，指定したセクションをカーネルに登録する．具体的な振舞いは以下の通り．

各オブジェクトモジュールに含まれるセクション名で指定したセクションが，メモリリージョン名で指定したメモリリージョンに配置され，メモリオブジェクトとして登録される．登録されるメモリオブジェクトには，mematr で指定したメモリオブジェクト属性が設定される．ATA_SEC の場合には，登録されるメモリオブジェクトのアクセス許可ベクタ（4 つのアクセス許可パターンの組）が，acptn1～acptn4 で指定した値に設定される．

指定したメモリリージョンが TA_NOWRITE 属性である場合には，メモリオブジェクト属性に TA_NOWRITE 属性を指定したことになる（TA_NOWRITE 属性を指定しても指定しなくても，同じ振舞いとなる）.

mematr に，TA_MEMINI と TA_MEMPRSV を同時に指定することはできない．指定した場合には，E_RSATR エラーとなる.

登録されるメモリオブジェクトと同じ保護ドメインに属し，メモリオブジェクト属性とアクセス許可ベクタがすべて一致するメモリオブジェクトがある場合には，1 つのメモリオブジェクトにまとめて登録される場合がある.

セクション名とメモリリージョン名は文字列パラメータ，mematr，acptn1～acptn4 は整数定数式パラメータである.

ターゲット定義で，ATA_SEC により登録できるセクションが属する保護ドメインや登録できる数に制限がある場合がある.

ATT_MOD／ATA_MOD がサポートされているターゲットでは，セクション名として，標準のセクションを指定することはできない．指定した場合には，E_PAR エラーとなる.

保護ドメイン毎の標準セクションは，コンフィギュレータによってカーネルに登録されるため，ATT_SEC／ATA_SEC で登録することはできない．セクション名として指定した場合には，E_PAR エラーとなる.

マルチプロセッサ対応カーネルにおいて，指定したメモリリージョンがあるクラス専用のメモリリージョンの場合で，ATT_SEC／ATA_SEC をクラスの囲みの外に記述するか，他のクラスの囲みの中に記述した場合には，E_RSATR エラーとなる.

【μITRON4.0/PX 仕様との関係】

μITRON4.0/PX 仕様に定義されていない静的 API である.

LNK_SEC　　セクションの配置〔SP〕

【静的 API】

```
LNK_SEC("セクション名", { "メモリリージョン名" })
```

【パラメータ】

"セクション名"	－	配置するセクションを指定する文字列
"メモリリージョン名"	－	セクションを配置するメモリリージョンを指定する文字列

【エラーコード】

E_RSATR	予約属性（属するクラスが不正）
E_PAR	パラメータエラー（セクション名，メモリリージョン名が不正
E_OBJ	オブジェクト状態エラー（登録済みのセクションの再登録）

【機能】

各オブジェクトモジュールに含まれるセクション名で指定したセクションを，メモリリージョン名で指定したメモリリージョンに配置する．

セクション名として，標準のセクションや保護ドメイン毎の標準セクションを指定することはできない．指定した場合には，E_PAR エラーとなる．

マルチプロセッサ対応カーネルにおいて，指定したメモリリージョンがあるクラス専用のメモリリージョンの場合で，LNK_SEC をクラスの囲みの外に記述するか，他のクラスの囲みの中に記述した場合には，E_RSATR エラーとなる．

【使用上の注意】

LNK_SEC により配置されたセクションは，メモリオブジェクトとしてカーネルに登録されず，メモリ保護が実現できる先頭番地とサイズになるとは限らない．

【μITRON4.0/PX 仕様との関係】

μITRON4.0/PX 仕様に定義されていない静的 API である．

ATT_MOD　オブジェクトモジュールの登録〔SP〕
ATA_MOD　オブジェクトモジュールの登録（アクセス許可ベクタ付き）〔SP〕

【静的 API】

```
ATT_MOD("オブジェクトモジュール名")
```

```
ATA_MOD("オブジェクトモジュール名",
        {ACPTN acptn1, ACPTN acptn2, ACPTN acptn3, ACPTN acptn4 })
```

【パラメータ】

"オブジェクトモジュール名"	－	登録するオブジェクトモジュールを指定する文字列

＊アクセス許可ベクタ（パケットの内容）

ACPTN	acptn1	通常操作 1 のアクセス許可パターン
ACPTN	acptn2	通常操作 2 のアクセス許可パターン
ACPTN	acptn3	管理操作のアクセス許可パターン
ACPTN	acptn4	参照操作のアクセス許可パターン

【エラーコード】

E_RSATR	予約属性（mematr が不正または使用できない）
E_NOSPT	未サポート機能（ATT_MOD／ATA_MOD がサポートされていない）
E_OBJ	オブジェクト状態エラー（登録済みのオブジェクトモジュールの再登録）

【機能】

各パラメータで指定した情報に従って，指定したオブジェクトモジュールをカーネルに登録する．具体的な振舞いは以下の通り．

オブジェクトモジュール名で指定したオブジェクトモジュールに含まれる標準のセクションの内，書込みアクセスを行わないセクションは標準 ROM リージョンに，書込みアクセスを行うセクションは標準 RAM リージョンに配置され，メモリオブジェクトとして登録される．登録されるメモリオブジェクトには，ターゲット定義でセクション毎に定まるメモリオブジェクト属性が設定される．ATA_MOD の場合には，登録されるメモリオブジェクトのアクセス許可ベクタ（4 つのアクセス許可パターンの組）が，acptn1～acptn4 で指定した値に設定される．

マルチプロセッサ対応カーネルでは，ATT_MOD／ATA_MOD を，クラスの囲みの外に記述することも，クラスの囲みの中に記述することもできる．ATT_MOD／ATA_MOD をクラスの囲みの外に記述した場合，標準のセクションが配置されるメモリリージョンは，共通の標準 ROM リージョン／標準 RAM リージョンとなる．クラスの囲みの中に記述した場合，そのクラス専用の標準 ROM リージョン／標準 RAM リージョンが登録されていればそのリージョン，登録されていなければ共通の標準 ROM リージョン／標準 RAM リージョンとなる．

登録されるメモリオブジェクトと同じ保護ドメインに属し，メモリオブジェクト属性とアクセス許可ベクタがすべて一致するメモリオブジェクトがある場合には，1 つのメモリオブジェクトにまとめて登録される場合がある．

オブジェクトモジュール名は文字列パラメータ，acptn1～acptn4 は整数定数式パラメータである．

ターゲット定義で，ATA_MOD により登録できるオブジェクトモジュールが属する保護ドメインや登録できる数に制限がある場合がある．

ターゲット定義で，ATT_MOD／ATA_MOD がサポートされていない場合がある．
ATT_MOD／ATA_MOD がサポートされている場合には，TOPPERS_SUPPORT_ATT_MOD がマクロ定義される．サポートされていない場合に ATT_MOD／ATA_MOD を使用すると，コンフィギュレータが E_NOSPT エラーを報告する．

【補足説明】
各セクションが配置されるメモリリージョンとそれに設定されるメモリオブジェクト属性は，例えば，プログラムコードを含むセクションであれば，フラッシュメモリのメモリリージョンに配置し，書込みアクセス禁止のメモリオブジェクト属性が設定されるというように，ターゲットシステム毎に定められる．

ATT_MOD／ATA_MOD では，標準のセクション以外は配置・登録されない．標準のセクション以外のセクションを配置・登録するためには，ATT_SEC/ATA_SEC を用いる必要がある．

【μITRON4.0/PX 仕様との関係】
オブジェクトモジュールに含まれるセクションの配置場所が，標準 ROM リージョンと標準 RAM リージョンであることを明確化した．

ATT_MEM　　メモリオブジェクトの登録〔SP〕
ATA_MEM　　メモリオブジェクトの登録（アクセス許可ベクタ付き）〔SP〕
att_mem　　メモリオブジェクトの登録〔TPD〕

【静的 API】

```
ATT_MEM({ ATR mematr, void *base, SIZE size })
```

```
ATA_MEM({ ATR mematr, void *base, SIZE size },
        { ACPTN acptn1, ACPTN acptn2, ACPTN acptn3, ACPTN acptn4 })
```

【C 言語 API】

```
ER ercd = att_mem(const T_AMEM *pk_amem)
```

【パラメータ】

T_AMEM *	pk_amem	メモリオブジェクトの登録情報を入れたパケットへのポインタ（静的 API を除く）

＊メモリオブジェクトの登録情報（パケットの内容）

ATR	mematr	メモリオブジェクト属性
void *	base	登録するメモリ領域の先頭番地
SIZE	size	登録するメモリ領域のサイズ（バイト数）

＊アクセス許可ベクタ（パケットの内容）

ACPTN	acptn1	通常操作 1 のアクセス許可パターン
ACPTN	acptn2	通常操作 2 のアクセス許可パターン
ACPTN	acptn3	管理操作のアクセス許可パターン
ACPTN	acptn4	参照操作のアクセス許可パターン

【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_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_RSATR	予約属性（mematr が不正または使用できない，属する保護ドメインが不正）
E_NOSPT	未サポート機能（条件はターゲット定義）
E_PAR	パラメータエラー（base，size が不正）
E_OACV〔sP〕	オブジェクトアクセス違反（システム状態に対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（pk_amem が指すメモリ領域への読出しアクセスが許可されていない）
E_OBJ	オブジェクト状態エラー（登録済みのメモリオブジェクトとメモリ領域が重なる）

【機能】

各パラメータで指定したメモリオブジェクト登録情報に従って，メモリオブジェクトを登録する．具体的な振舞いは以下の通り．

base と size で指定したメモリ領域が，メモリオブジェクトとして登録される．登録されるメモリオブジェクトには，mematr で指定したメモリオブジェクト属性が設定される．ATA_MEM の場合には，登

録されるメモリオブジェクトのアクセス許可ベクタ（4 つのアクセス許可パターンの組）が，acptn1～acptn4 で指定した値に設定される．

mematr には，TA_MEMPRSV を指定しなければならず，TA_MEMINI を指定することはできない．TA_MEMPRSV を指定しない場合や，TA_MEMINI を指定した場合には，E_RSATR エラーとなる．

静的 API においては，mematr，size，acptn1～acptn4 は整数定数式パラメータ，base は一般定数式パラメータである．

ターゲット定義で，ATT_MEM／ATA_MEM により登録できるメモリオブジェクトが属する保護ドメインや登録できる数に制限がある場合がある．

base や size に，ターゲット定義の制約に合致しない先頭番地やサイズを指定した時には，E_PAR エラーとなる．また，size が 0 の場合には，E_PAR エラーとなる．
登録しようとしたメモリオブジェクトが，登録済みのメモリオブジェクトとメモリ領域が重なる場合には，E_OBJ エラーとなる．

【使用上の注意】
ATT_MEM／ATA_MEM は，メモリ空間にマッピングされた I/O 領域にアクセスできるようにするために使用することを想定した静的 API である．メモリ領域に対しては，ATT_SEC／ATA_SEC か ATT_MOD／ATA_MOD を使用することを推奨する．

ATT_MEM／ATA_MEM で登録したメモリオブジェクトのメモリ領域が，ATT_REG で登録したメモリリージョンと重なっても，直ちにエラーとはならない．ただし，メモリリージョン内に配置されたメモリオブジェクトと，ATT_MEM／ATA_MEM で登録したメモリオブジェクトのメモリ領域が重なった場合には，E_OBJ エラーとなる．

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，ATT_MEM と ATA_MEM のみをサポートする．

【μITRON4.0/PX 仕様との関係】
アクセス許可ベクタを指定してメモリオブジェクトを登録するサービスコール（ata_mem）は廃止した．

base や size がターゲット定義の制約に合致しない場合，μITRON4.0/PX 仕様ではターゲット定義の制約に合致するようにメモリ領域を広げることとしていたが，この仕様では E_PAR エラーとなることとした．

ATT_PMA	物理メモリ領域の登録〔SP〕
ATA_PMA	物理メモリ領域の登録（アクセス許可ベクタ付き）〔SP〕
att_pma	物理メモリ領域の登録〔TPD〕

【静的 API】

```
ATT_PMA({ ATR mematr, void *base, SIZE size, void *paddr })
```

```
ATA_PMA({ ATR mematr, void *base, SIZE size, void *paddr   },
        { ACPTN acptn1, ACPTN acptn2, ACPTN acptn3, ACPTN acptn4 })
```

【C 言語 API】

```
ER ercd = att_pma(const T_APMA *pk_apma)
```

【パラメータ】

T_APMA *	pk_apma	物理メモリ領域の登録情報を入れたパケットへのポインタ（静的 API を除く）

＊物理メモリ領域の登録情報（パケットの内容）

ATR	mematr	メモリオブジェクト属性
void *	base	登録するメモリ領域の先頭番地
SIZE	size	登録するメモリ領域のサイズ（バイト数）
void *	paddr	登録するメモリ領域の物理アドレスの先頭番地

＊アクセス許可ベクタ（パケットの内容）

ACPTN	acptn1	通常操作 1 のアクセス許可パターン
ACPTN	acptn2	通常操作 2 のアクセス許可パターン
ACPTN	acptn3	管理操作のアクセス許可パターン
ACPTN	acptn4	参照操作のアクセス許可パターン

【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_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_RSATR	予約属性（mematr が不正または使用できない，属する保護ドメインが不正）
E_NOSPT	未サポート機能（条件はターゲット定義）
E_PAR	パラメータエラー（base，size，paddr が不正）
E_OACV〔sP〕	オブジェクトアクセス違反（システム状態に対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（pk_apma が指すメモリ領域への読出しアクセスが許可されていない）
E_OBJ	オブジェクト状態エラー（登録済みのメモリオブジェクトとメモリ領域が重なる）

【機能】

各パラメータで指定した物理メモリ領域の登録情報に従って，メモリオブジェクトを登録する．具体的な振舞いは以下の通り．

物理アドレス空間において先頭番地が paddr，サイズが size のメモリ領域が，論理アドレス空間において base で指定した番地からアクセスできるように，メモリオブジェクトとして登録される．登録されるメモリオブジェクトには，mematr で指定したメモリオブジェクト属性が設定される．ATA_PMA の場合には，登録されるメモリオブジェクトのアクセス許可ベクタ（4 つのアクセス許可パターンの組）が，acptn1～acptn4 で指定した値に設定される．

mematr には，TA_MEMPRSV を指定しなければならず，TA_MEMINI を指定することはできない．TA_MEMPRSV を指定しない場合や，TA_MEMINI を指定した場合には，E_RSATR エラーとなる．

静的 API においては，mematr，size，paddr，acptn1～acptn4 は整数定数式パラメータ，base は一般定数式パラメータである．

ターゲット定義で，ATT_PMA／ATA_PMA により登録できるメモリオブジェクトが属する保護ドメインや登録できる数に制限がある場合がある．

base, size, paddr に，ターゲット定義の制約に合致しない先頭番地やサイズを指定した時には，E_PAR エラーとなる．また，size が 0 の場合には，E_PAR エラーとなる．登録しようとしたメモリオブジェクトが，登録済みのメモリオブジェクトと論理アドレス空間においてメモリ領域が重なる場合には，E_OBJ エラーとなる．

ATT_PMA／ATA_PMA／att_pma は，MMU（Memory Management Unit）を持つターゲットシステムにおいて，ターゲット定義でサポートされる機能である．ATT_PMA／ATA_PMA／att_pma がサポートされている場合には，TOPPERS_SUPPORT_ATT_PMA がマクロ定義される．ATT_PMA／ATA_PMA がサポートされていない場合にこれらの静的 API を使用すると，コンフィギュレータが E_NOSPT エラーを報告する．また，att_pma がサポートされていない場合に att_pma を呼び出すと，E_NOSPT エラーが返るか，リンク時にエラーとなる．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，ターゲット定義で，ATT_PMA と ATA_PMA のみをサポートする．

【μITRON4.0/PX 仕様との関係】

μITRON4.0/PX 仕様に定義されていない静的 API およびサービスコールである．

sac_mem　　メモリオブジェクトのアクセス許可ベクタの設定〔TPD〕

【C 言語 API】

```
ER ercd = sac_mem(const void *base, const ACVCT *p_acvct)
```

【パラメータ】

void *	base	メモリオブジェクトの先頭番地
ACVCT *	p_acvct	アクセス許可ベクタを入れたパケットへのポインタ

＊アクセス許可ベクタ（パケットの内容）

ACPTN	acptn1	通常操作 1 のアクセス許可パターン
ACPTN	acptn2	通常操作 2 のアクセス許可パターン
ACPTN	acptn3	管理操作のアクセス許可パターン
ACPTN	acptn4	参照操作のアクセス許可パターン

【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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_PAR	パラメータエラー（base が不正）
E_NOEXS〔D〕	オブジェクト未登録（base で指定した番地を含むメモリオブジェクトが登録されていない）
E_OACV〔P〕	オブジェクトアクセス違反（対象メモリオブジェクトに対する管理操作が許可されていない）
E_MACV〔P〕	メモリアクセス違反（p_acvct が指すメモリ領域への読出しアクセスが許可されていない）
E_OBJ	オブジェクト状態エラー（対象メモリオブジェクトは静的 API で登録された）

【機能】

base で指定したメモリオブジェクト（対象メモリオブジェクト）のアクセス許可ベクタ（4 つのアクセス許可パターンの組）を，各パラメータで指定した値に設定する．

静的 API によって登録したメモリオブジェクトは，アクセス許可ベクタを設定することができない．アクセス許可ベクタを設定しようとした場合には，E_OBJ エラーとなる．

base に，メモリオブジェクトの先頭番地以外を指定した場合には，E_PAR エラーとなる．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，sac_mem をサポートしない．

【μITRON4.0/PX 仕様との関係】

静的 API によって登録したメモリオブジェクトは，アクセス許可ベクタを設定することができないこととした．

μITRON4.0/PX 仕様では，base はメモリオブジェクトに含まれる番地を指定するものとしていたが，この仕様では，メモリオブジェクトの先頭番地でなければならないものとした．

det_mem　　メモリオブジェクトの登録解除〔TPD〕

【C 言語 API】

```
ER ercd = det_mem(const void *base)
```

【パラメータ】

void *	base	メモリオブジェクトの先頭番地

【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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_PAR	パラメータエラー（base が不正）
E_NOEXS〔D〕	オブジェクト未登録（base で指定される番地を含むメモリオブジェクトが登録されていない）
E_OACV〔P〕	オブジェクトアクセス違反（対象メモリオブジェクトに対する管理操作が許可されていない）
E_OBJ	オブジェクト状態エラー（対象メモリオブジェクトは静的 API で登録された）

【機能】

base で指定したメモリオブジェクト（対象メモリオブジェクト）を登録解除する．

静的 API によって登録したメモリオブジェクトは，登録を解除することができない．登録を解除しようとした場合には，E_OBJ エラーとなる．

base に，メモリオブジェクトの先頭番地以外を指定した場合には，E_PAR エラーとなる．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，det_mem をサポートしない．

【μITRON4.0/PX 仕様との関係】

静的 API によって登録したメモリオブジェクトは，登録を解除することができないこととした．

μITRON4.0/PX 仕様では，base はメモリオブジェクトに含まれる番地を指定するものとしていたが，この仕様では，メモリオブジェクトの先頭番地でなければならないものとした．

prb_mem　　メモリ領域に対するアクセス権のチェック〔TP〕

【C 言語 API】

```
ER ercd = prb_mem(const void *base, SIZE size, ID tskid, MODE pmmode)
```

【パラメータ】

void *	base	メモリ領域の先頭番地
SIZE	size	メモリ領域のサイズ（バイト数）
ID	tskid	アクセス元のタスクの ID 番号
MODE	pmmode	アクセスモード

【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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（tskid が不正）
E_PAR	パラメータエラー（base，size，pmmode が不正）
E_NOEXS〔D〕	オブジェクト未登録（base で指定される番地を含むメモリオブジェクトが登録されていない）
E_OACV〔P〕	オブジェクトアクセス違反（対象メモリ領域を含むメモリオブジェクトに対する参照操作が許可されていない）
E_MACV〔P〕	メモリアクセス違反（対象メモリ領域へのアクセスが許可されていない）
E_OBJ	オブジェクト状態エラー（対象メモリ領域がメモリオブジェクトの境界を越えている）

【機能】

tskid で指定したタスクから，base と size で指定したメモリ領域（対象メモリ領域）に対して，pmmode で指定した種別のアクセスが許可されているかをチェックする．アクセスが許可されている場合に E_OK，そうでない場合に E_MACV が返る．tskid で指定したタスクがカーネルドメインに属する場合，E_MACV が返ることはない．

pmmode には，TPM_WRITE（＝0x01U），TPM_READ（＝0x02U），TPM_EXEC（＝0x04U）のいずれか，またはそれらの内のいくつかのビット毎論理和（C 言語の"|"）を指定することができる．TPM_WRITE，TPM_READ，TPM_EXEC を指定した場合には，それぞれ，読出しアクセス，書込みア

クセス，実行アクセスが許可されているかをチェックする．また，いくつかのビット毎論理和を指定した場合には，それらに対応した種別のアクセスがすべて許可されているかをチェックする．

tskid に TSK_SELF（＝0）を指定すると，自タスクから対象メモリ領域に対してアクセスが許可されているかをチェックする．

size が 0 の場合には，E_PAR エラーとなる．

【μITRON4.0/PX 仕様との関係】
アクセスする主体の指定方法を，保護ドメインによる指定（domid）から，タスクによる指定（tskid）に変更した．また，pmmode に指定できるアクセス種別に TPM_EXEC を追加し，TPM_WRITE と TPM_READ の値を入れ換えた．

---

ref_mem　　メモリオブジェクトの状態参照〔TP〕

【C 言語 API】

```
ER ercd = ref_mem(const void *base, T_RMEM *pk_rmem)
```

☆未完成

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，ref_mem をサポートしない．

4.9. 割込み管理機能

割込み処理のプログラムは，割込みサービスルーチン（ISR）として実現することを推奨する．割込みサービスルーチンをカーネルに登録する場合には，まず，割込みサービスルーチンの登録対象となる割込み要求ラインの属性を設定しておく必要がある．割込みサービスルーチンは，カーネル内の割込みハンドラを経由して呼び出される．

ただし，カーネルが用意する割込みハンドラで対応できないケースに対応するために，アプリケーションで割込みハンドラを用意することも可能である．この場合にも，割込みハンドラをカーネルに登録する前に，割込みハンドラの登録対象となる割込みハンドラ番号に対応する割込み要求ラインの属性を設定しておく必要がある．

割込み要求ラインの属性を設定する際に指定する割込み要求ライン属性には，次の属性を指定することができる．

TA_ENAINT	0x01U	割込み要求禁止フラグをクリア
TA_EDGE	0x02U	エッジトリガ

ターゲットによっては，ターゲット定義の割込み要求ライン属性を指定できる場合がある．ターゲット定義の割込み要求ライン属性として，次の属性を予約している．

TA_POSEDGE	ー	ポジティブエッジトリガ
TA_NEGEDGE	ー	ネガティブエッジトリガ
TA_BOTHEDGE	ー	両エッジトリガ
TA_LOWLEVEL	ー	ローレベルトリガ
TA_HIGHLEVEL	ー	ハイレベルトリガ
A_BROADCAST	ー	
TA_POSEDGE	ー	ポジティブエッジトリガ
TA_NEGEDGE	ー	ネガティブエッジトリガ
TA_BOTHEDGE	ー	すべてのプロセッサで割込みを処理（マルチプロセッサ対応カーネルの場合）

割込みサービスルーチンは，カーネルが実行を制御する処理単位である．割込みサービスルーチンは，割込みサービスルーチン ID と呼ぶ ID 番号によって識別する．

1 つの割込み要求ラインに対して複数の割込みサービスルーチンを登録した場合，それらの割込みサービスルーチンは，割込みサービスルーチン優先度の高い順にすべて呼び出される．割込みサービスルーチン優先度が同じ場合には，登録した順（静的 API により登録した場合には，割込みサービスルーチンを生成する API をコンフィギュレーションファイル中に記述した順）で呼び出される．

保護機能対応カーネルにおいて，割込みサービスルーチンが属することのできる保護ドメインは，カーネルドメインに限られる．

割込みサービスルーチン属性に指定できる属性はない．そのため割込みサービスルーチン属性には，TA_NULL を指定しなければならない．

C 言語による割込みサービスルーチンの記述形式は次の通り．

```
void interrupt_service_routine(intptr_t exinf)
{
    割込みサービスルーチン本体
}
```

exinf には，割込みサービスルーチンの拡張情報が渡される．

割込みハンドラは，カーネルが実行を制御する処理単位である．割込みハンドラは，割込みハンドラ番号と呼ぶオブジェクト番号によって識別する．

保護機能対応カーネルにおいて，割込みハンドラは，カーネルドメインに属する．

割込みハンドラを登録する際に指定する割込みハンドラ属性には，ターゲット定義で，次の属性を指定することができる．

TA_NONKERNEL	0x02U	カーネル管理外の割込み

TA_NONKERNEL を指定しない場合，カーネル管理の割込みとなる．また，ターゲットによっては，その他のターゲット定義の割込みハンドラ属性を指定できる場合がある．

C 言語による割込みハンドラの記述形式は次の通り．

```
void interrupt_handler(void)
{
    割込みハンドラ本体
}
```

割込み管理機能に関連するカーネル構成マクロは次の通り．

TMIN_INTPRI	―	割込み優先度の最小値（最高値）
TMAX_INTPRI	―	割込み優先度の最大値（最低値，＝-1）

TMIN_ISRPRI	―	割込みサービスルーチン優先度の最小値（＝1）
TMAX_ISRPRI	―	割込みサービスルーチン優先度の最大値

TOPPERS_SUPPORT_DIS_INT	―	dis_int がサポートされている
TOPPERS_SUPPORT_ENA_INT	―	ena_int がサポートされている

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，割込みサービスルーチン優先度の最大値（＝TMAX_ISRPRI）は 16 に固定されている．ただし，タスク優先度拡張パッケージでは，TMAX_ISRPRI を 256 に拡張する．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，割込みサービスルーチン優先度の最大値（＝TMAX_ISRPRI）は 16 に固定されている．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，割込みサービスルーチン優先度の最大値（＝TMAX_ISRPRI）は 16 に固定されている．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，割込みサービスルーチン優先度の最大値（＝TMAX_ISRPRI）は 16 に固定されている．

【μITRON4.0 仕様との関係】

割込み要求ラインの属性，割込み優先度，割込みサービスルーチン優先度は，μITRON4.0 仕様にない概念であり，TMIN_INTPRI，TMAX_INTPRI，TMIN_ISRPRI，TMAX_ISRPRI は，μITRON4.0 仕様に定義のないカーネル構成マクロである．また，TA_NONKERNEL は，μITRON4.0 仕様に定義のない割込みハンドラ属性である．

CFG_INT　　割込み要求ラインの属性の設定〔S〕
cfg_int　　割込み要求ラインの属性の設定〔TD〕

【静的 API】

```
CFG_INT(INTNO intno, {ATR intatr, PRI intpri})
```

【C 言語 API】

```
ER ercd = cfg_int(INTNO intno, const T_CINT *pk_cint)
```

【パラメータ】

INTNO	intno	割込み番号
T_CINT *	pk_cint	割込み要求ラインの属性の設定情報を入れたパケットへのポインタ（静的 API を除く）

＊割込み要求ラインの属性の設定情報（パケットの内容）

ATR	intatr	割込み要求ライン属性
PRI	intpri	割込み優先度

【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_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_RSATR	予約属性（intatr が不正または使用できない，属する保護ドメインかクラスが不正）
E_OACV〔sP〕	オブジェクトアクセス違反（システム状態に対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（pk_cint が指すメモリ領域への読出しアクセスが許可されていない）
E_PAR	パラメータエラー（intno，intpri が不正）
E_OBJ	オブジェクト状態エラー（対象割込み要求ラインに対してすでに属性が設定されている：CFG_INT の場合，カーネル管理の割込みか否かと intpri の値が整合していない）

【機能】

intno で指定した割込み要求ライン（対象割込み要求ライン）に対して，各パラメータで指定した属性を設定する．

対象割込み要求ラインの割込み要求禁止フラグは，intatr に TA_ENAINT を指定した場合にクリアされ，指定しない場合にセットされる．

静的 API においては，intno，intatr，intpri は整数定数式パラメータである．

cfg_int において，ターゲット定義で，複数の割込み要求ラインの割込み優先度が連動して設定される場合がある．

CFG_INT において, 対象割込み要求ラインに対してすでに属性が設定されている場合（言い換えると，同じ割込み番号に対する CFG_INT が複数ある場合）には，E_OBJ エラーとなる．

intpri に指定できる値は，基本的には，TMIN_INTPRI 以上，TMAX_INTPRI 以下の値である．ターゲット定義の拡張で，カーネル管理外の割込み要求ラインに対しても属性を設定できる場合には TMIN_INTPRI よりも小さい値を指定することができる．このように拡張されている場合，カーネル管理外の割込み要求ラインを対象として，intpri に TMIN_INTPRI 以上の値を指定した場合には，E_OBJ エラーとなる．逆に，カーネル管理の割込み要求ラインを対象として，intpri が TMIN_INTPRI よりも小さい値である場合にも，E_OBJ エラーとなる．

対象割込み要求ラインに対して，設定できない割込み要求ライン属性を intatr に指定した場合には E_RSATR エラー，設定できない割込み優先度を intpri に指定した場合には E_PAR エラーとなる．ここで，設定できない割込み要求ライン属性／割込み優先度には，ターゲット定義の制限によって設定できない値も含む．また，マルチプロセッサ対応カーネルにおいて，cfg_int を呼び出したタスクが割り付けられているプロセッサから，対象割込み要求ラインの属性を設定できない場合も，これに該当する．

保護機能対応カーネルにおいて，CFG_INT は，カーネルドメインの囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．また，cfg_int はカーネルオブジェクトを登録するサービスコールではないため, 割込み要求ライン属性に TA_DOM(domid)を指定した場合には E_RSATR エラーとなる．ただし，TA_DOM(TDOM_SELF)を指定した場合には，指定が無視され，E_RSATR エラーは検出されない．

マルチプロセッサ対応カーネルで，CFG_INT の記述が，対象割込み要求ラインに対して登録された割込みサービスルーチン（または対象割込み番号に対応する割込みハンドラ番号に対して登録された割込みハンドラ）と異なるクラスの囲み中にある場合には，E_RSATR エラーとなる．

【補足説明】

ターゲット定義の制限によって設定できない割込み要求ライン属性／割込み優先度は，主にターゲットハードウェアの制限から来るものである．例えば，対象割込み要求ラインに対して，トリガモードや割込み優先度が固定されていて，変更できないケースが考えられる．

cfg_int において，ターゲット定義で，複数の割込み要求ラインの割込み優先度が連動して設定されるのは，ターゲットハードウェアの制限により，異なる割込み要求ラインに対して，同一の割込み優先度しか設定できないケースに対応するための仕様である．この場合，CFG_INT においては，同一の割込み優先度しか設定できない割込み要求ラインに対して異なる割込み優先度を設定した場合には，E_PAR エラーとなる．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，CFG_INT のみをサポートする．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，CFG_INT のみをサポートする．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，CFG_INT のみをサポートする．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，CFG_INT のみをサポートする．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様に定義されていない静的 API およびサービスコールである．

CRE_ISR　　割込みサービスルーチンの生成〔S〕
ATT_ISR　　割込みサービスルーチンの追加〔S〕
acre_isr　　割込みサービスルーチンの生成〔TD〕

【静的 API】

```
CRE_ISR(ID isrid, {ATR isratr, intptr_t exinf,
                                NTNO intno, ISR isr, PRI isrpri })
```

```
ATT_ISR({ ATR isratr, intptr_t exinf, INTNO intno, ISR isr, PRI isrpri })
```

【C 言語 API】

```
ER_ID isrid = acre_isr(const T_CISR *pk_cisr)
```

【パラメータ】

ID	isrid	生成するイベントフラグの ID 番号（CRE_FLG の場合）
T_CFLG *	pk_cflg	対象割込みサービスルーチンの ID 番号(CRE_ISR の場合)

＊割込みサービスルーチンの生成情報（パケットの内容）

ATR	isratr	割込みサービスルーチン属性
intptr_t	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登録する割込み番号
ISR	isr	割込みサービスルーチンの先頭番地
PRI	isrpri	割込みサービスルーチン優先度

【リターンパラメータ】

ER_ID	isrid	生成され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_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_RSATR	予約属性（isratr が不正または使用できない，属する保護ドメインかクラスが不正）
E_PAR	パラメータエラー（intno，isr，isrpri が不正）
E_OACV〔sP〕	オブジェクトアクセス違反（システム状態に対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（pk_cisr が指すメモリ領域への読出しアクセスが許可されていない）
E_NOID〔sD〕	ID 番号不足（割り付けられる割込みサービスルーチン ID がない）
E_OBJ	オブジェクト状態エラー（isrid で指定した割込みサービスルーチンが登録済み：CRE_ISR の場合，その他の条件については機能の項を参照すること）

【機能】

各パラメータで指定した割込みサービスルーチン生成情報に従って，割込みサービスルーチンを生成する．

ATT_ISR によって生成された割込みサービスルーチンは，ID 番号を持たない．

intno で指定した割込み要求ラインの属性が設定されていない場合には，E_OBJ エラーとなる．また，intno で指定した割込み番号に対応する割込みハンドラ番号に対して，割込みハンドラを定義する機能（DEF_INH，def_inh）によって割込みハンドラが定義されている場合にも，E_OBJ エラーとなる．さらに，intno でカーネル管理外の割込みを指定した場合にも，E_OBJ エラーとなる．

静的 API においては，isrid はオブジェクト識別名，isratr，intno，isrpri は整数定数式パラメータ，

exinf と isr は一般定数式パラメータである．

保護機能対応カーネルにおいて，CRE_ISR および ATT_ISR は，カーネルドメインの囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．また，acre_isr で，生成する割込みサービスルーチンが属する保護ドメインとしてカーネルドメイン以外を指定した場合には，E_RSATR エラーとなる．

マルチプロセッサ対応カーネルで，生成する割込みサービスルーチンの属するクラスの割付け可能プロセッサが，intno で指定した割込み要求ラインが接続されたプロセッサの集合に含まれていない場合には，E_RSATR エラーとなる．また，intno で指定した割込み要求ラインに対して登録済みの割込みサービスルーチンがある場合に，生成する割込みサービスルーチンがそれと異なるクラスに属する場合にも，E_RSATR エラーとなる．さらに，ターゲット定義で，割込みサービスルーチンが属することができるクラスに制限がある場合がある．生成する割込みサービスルーチンの属するクラスが，ターゲット定義の制限に合致しない場合にも，E_RSATR エラーとなる．

isrpri は，TMIN_ISRPRI 以上，TMAX_ISRPRI 以下でなければならない．

静的 API において，isr が不正である場合に E_PAR エラーが検出されるか否かは，ターゲット定義である．

【TOPPERS/ASP カーネルにおける規定】
ASP カーネルでは，ATT_ISR のみをサポートする．ただし，動的生成機能拡張パッケージでは，acre_isr もサポートする．

【TOPPERS/FMP カーネルにおける規定】
FMP カーネルでは，ATT_ISR のみをサポートする．

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，ATT_ISR のみをサポートする．

【TOPPERS/SSP カーネルにおける規定】
SSP カーネルでは，ATT_ISR のみをサポートする．

【μITRON4.0 仕様との関係】
割込みサービスルーチンの生成情報に，isrpri（割込みサービスルーチンの割込み優先度）を追加した．CRE_ISR は，μITRON4.0 仕様に定義されていない静的 API である．

AID_ISR　　割付け可能な割込みサービスルーチン ID の数の指定〔SD〕

【静的 API】

```
AID_ISR(uint_t noisr)
```

【パラメータ】

uint_t	noisr	割付け可能な割込みサービスルーチン ID の数

【エラーコード】

E_RSATR	予約属性（属する保護ドメインまたはクラスが不正）

【機能】

noisr で指定した数の割込みサービスルーチン ID を，割込みサービスルーチンを生成するサービスコールによって割付け可能な割込みサービスルーチン ID として確保する．

noisr は整数定数式パラメータである．

SAC_ISR　　割込みサービスルーチンのアクセス許可ベクタの設定〔SP〕
sac_isr　　割込みサービスルーチンのアクセス許可ベクタの設定〔TPD〕

【静的 API】

```
SAC_ISR(ID isrid, {ACPTN acptn1, ACPTN acptn2,
                   ACPTN acptn3, ACPTN acptn4})
```

【C 言語 API】

```
ER ercd = sac_isr(ID isrid, const ACVCT *p_acvct)
```

【パラメータ】

ID	isrid	対象割込みサービスルーチンの ID 番号
ACVCT *	p_acvct	アクセス許可ベクタを入れたパケットへのポインタ（静的 API を除く）

＊アクセス許可ベクタ（パケットの内容）

ACPTN	acptn1	通常操作 1 のアクセス許可パターン
ACPTN	acptn2	通常操作 2 のアクセス許可パターン
ACPTN	acptn3	管理操作のアクセス許可パターン
ACPTN	acptn4	参照操作のアクセス許可パターン

【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_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（isrid が不正）
E_RSATR	予約属性（属する保護ドメインかクラスが不正：SAC_ISR の場合）
E_NOEXS〔D〕	オブジェクト未登録（対象割込みサービスルーチンが未登録）
E_OACV〔sP〕	オブジェクトアクセス違反（対象割込みサービスルーチンに対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（p_acvct が指すメモリ領域への読出しアクセスが許可されていない）
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は静的 API で生成された：sac_isr の場合，対象割込みサービスルーチンに対してアクセス許可ベクタが設定済み：SAC_ISR の場合）

【機能】

isrid で指定した割込みサービスルーチン（対象割込みサービスルーチン）のアクセス許可ベクタ（4 つのアクセス許可パターンの組）を，各パラメータで指定した値に設定する．

静的 API においては，isrid はオブジェクト識別名，acptn1～acptn4 は整数定数式パラメータである．

SAC_ISR は，対象割込みサービスルーチンが属する保護ドメイン（この仕様ではカーネルドメインに限られる）の囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，SAC_ISR，sac_isr をサポートしない．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，SAC_ISR，sac_isr をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，SAC_ISR，sac_isr をサポートしない．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，SAC_ISR，sac_isr をサポートしない．

【未決定事項】

割込みサービスルーチンのアクセス許可ベクタを設けず，システム状態のアクセス許可ベクタでアクセス保護する方法も考えられる．

del_isr　　割込みサービスルーチンの削除〔TD〕

【C 言語 API】

```
ER ercd = del_isr(ID isrid)
```

【パラメータ】

ID	isrid	対象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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_ID	不正 ID 番号（cycid が不正）
E_NOEXS〔D〕	オブジェクト未登録（対象割込みサービスルーチンが未登録）
E_OACV〔P〕	オブジェクトアクセス違反（対象割込みサービスルーチンに対する管理操作が許可されていない）
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は静的 API で生成された）

【機能】

isrid で指定した割込みサービスルーチン（対象割込みサービスルーチン）を削除する．具体的な振舞いは以下の通り．

対象割込みサービスルーチンの登録が解除され，その割込みサービスルーチン ID が未使用の状態に戻される．

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，del_isr をサポートしない．ただし，動的生成機能拡張パッケージでは，del_isr をサポートする．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，del_isr をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，del_isr をサポートしない．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，del_isr をサポートしない．

ref_isr　　割込みサービスルーチンの状態参照〔T〕

【C 言語 API】

```
ER ercd = ref_isr(ID isrid, T_RISR *pk_risr)
```

☆未完成

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，ref_isr をサポートしない．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，ref_isr をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，ref_isr をサポートしない．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，ref_isr をサポートしない．

DEF_INH　　割込みハンドラの定義〔S〕
def_inh　　割込みハンドラの定義〔TD〕

【静的 API】

```
DEF_INH(INHNO inhno, {ATR inhatr, INTHDR inthdr})
```

【C 言語 API】

```
ER ercd = def_inh(INHNO inhno, const T_DINH *pk_dinh)
```

【パラメータ】

INHNO	inhno	割込みハンドラ番号
T_DINH *	pk_dinh	割込みハンドラの定義情報を入れたパケットへのポインタ（静的 API を除く）

＊割込みハンドラの定義情報（パケットの内容）

ATR	inhatr	割込みハンドラ属性
INTHDR	inthdr	割込みハンドラの先頭番地

【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_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_RSATR	予約属性（inhatr が不正または使用できない，属する保護ドメインかクラスが不正）
E_OACV〔sP〕	オブジェクトアクセス違反（システム状態に対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（pk_dinh が指すメモリ領域への読出しアクセスが許可されていない）
E_PAR	パラメータエラー（inhno，inthdr が不正）
E_OBJ	オブジェクト状態エラー（条件については機能の項を参照すること）

【機能】

inhno で指定した割込みハンドラ番号（対象割込みハンドラ番号）に対して，各パラメータで指定した

割込みハンドラ定義情報に従って，割込みハンドラを定義する．ただし，def_inh において pk_dinh を NULL にした場合には，対象割込みハンドラ番号に対する割込みハンドラの定義を解除する．

静的 API においては，inhno と inhatr は整数定数式パラメータ，inthdr は一般定数式パラメータである．

割込みハンドラを定義する場合（DEF_INH の場合および def_inh において pk_dinh を NULL 以外にした場合）には，次のエラーが検出される．

対象割込みハンドラ番号に対応する割込み要求ラインの属性が設定されていない場合には，E_OBJ エラーとなる．また，対象割込みハンドラ番号に対してすでに割込みハンドラが定義されている場合と，対象割込みハンドラ番号に対応する割込み番号を対象に割込みサービスルーチンが登録されている場合にも，E_OBJ エラーとなる．

ターゲット定義の拡張で，カーネル管理外の割込みに対しても割込みハンドラを定義できる場合には，次のエラーが検出される．カーネル管理外の割込みハンドラを対象として，inhatr に TA_NONKERNEL を指定しない場合には，E_OBJ エラーとなる．逆に，カーネル管理の割込みハンドラを対象として，inhatr に TA_NONKERNEL を指定した場合にも，E_OBJ エラーとなる．また，ターゲット定義でカーネル管理外に固定されている割込みハンドラがある場合には，それを対象割込みハンドラに指定して，inhatr に TA_NONKERNEL を指定しない場合には，E_RSATR エラーとなる．逆に，ターゲット定義でカーネル管理に固定されている割込みハンドラがある場合には，それを対象割込みハンドラに指定して，inhatr に TA_NONKERNEL を指定した場合には，E_RSATR エラーとなる．

保護機能対応カーネルにおいて，DEF_INH は，カーネルドメインの囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．また，def_inh で割込みハンドラを定義する場合には，割込みハンドラの属する保護ドメインを設定する必要はなく，割込みハンドラ属性に TA_DOM(domid)を指定した場合には E_RSATR エラーとなる．ただし，TA_DOM(TDOM_SELF)を指定した場合には，指定が無視され，E_RSATR エラーは検出されない．

マルチプロセッサ対応カーネルで，登録する割込みハンドラの属するクラスの初期割付けプロセッサが，その割込みが要求されるプロセッサでない場合には，E_RSATR エラーとなる．また，ターゲット定義で，割込みハンドラが属することができるクラスに制限がある場合がある．登録する割込みハンドラの属するクラスが，ターゲット定義の制限に合致しない場合にも，E_RSATR エラーとなる．

割込みハンドラの定義を解除する場合（def_inh において pk_dinh を NULL にした場合）で，対象割込みハンドラ番号に対して割込みハンドラが定義されていない場合には，E_OBJ エラーとなる．また，対象割込みハンドラ番号に対して定義された割込みハンドラが，静的 API で定義されたものである場合には，ターゲット定義で E_OBJ エラーとなる場合がある．

ターゲット定義で，対象割込みハンドラを定義（または定義解除）できない場合には，E_PAR エラーとなる．具体的には，マルチプロセッサ対応カーネルにおいて，def_inh を呼び出したタスクが割り付けられているプロセッサから，対象割込みハンドラを定義（または定義解除）できない場合が，これに該当する．

静的 API において，inthdr が不正である場合に E_PAR エラーが検出されるか否かは，ターゲット定義である．

【TOPPERS/ASP カーネルにおける規定】
ASP カーネルでは，DEF_INH のみをサポートする．

【TOPPERS/FMP カーネルにおける規定】
FMP カーネルでは，DEF_INH のみをサポートする．

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，DEF_INH のみをサポートする．

【TOPPERS/SSP カーネルにおける規定】
SSP カーネルでは，DEF_INH のみをサポートする．

【μITRON4.0 仕様との関係】
inthdr のデータ型を INTHDR に変更した．

def_inh によって定義済みの割込みハンドラを再定義しようとした場合に，E_OBJ エラーとすることにした．割込みハンドラの定義を変更するには，一度定義を解除してから，再度定義する必要がある．

dis_int　　割込みの禁止〔T〕

【C 言語 API】

```
ER ercd = dis_int(INTNO intno)
```

【パラメータ】

INTNO	intno	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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し）
E_NOSPT	未サポートエラー（dis_int がサポートされていない）
E_PAR	パラメータエラー（intno が不正，その他の条件については機能の項を参照すること）
E_OACV〔P〕	オブジェクトアクセス違反（システム状態に対する通常操作 2 が許可されていない）
E_OBJ	オブジェクト状態エラー（intno で指定した割込み要求ラインに対して割込み要求ライン属性が設定されていない）

【機能】

intno で指定した割込み要求ライン（対象割込み要求ライン）の割込み要求禁止フラグをセットする．

ターゲット定義で，対象割込み要求ラインの割込み要求禁止フラグをセットできない場合には，E_PAR エラーとなる．具体的には，対象割込み要求ラインに対して割込み要求禁止フラグがサポートされていない場合や，マルチプロセッサ対応カーネルにおいて，dis_int を呼び出したタスクが割り付けられているプロセッサから，対象割込み要求ラインの割込み要求禁止フラグが操作できない場合が，これに該当する．

ターゲット定義で，割込み要求禁止フラグの振舞いが，この仕様の規定と異なる場合がある．特にマルチプロセッサ対応カーネルでは，あるプロセッサから dis_int を呼び出して割込み要求禁止フラグをセットしても，他のプロセッサに対しては割込みがマスクされない場合がある．

ターゲット定義で，dis_int がサポートされていない場合がある．dis_int がサポートされている場合には，TOPPERS_SUPPORT_DIS_INT がマクロ定義される．サポートされていない場合に dis_int を呼び出すと，E_NOSPT エラーが返るか，リンク時にエラーとなる．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様で実装定義としていた intno の意味を標準化した．

CPU ロック状態でも呼び出せるものとした．

ena_int　　割込みの許可〔T〕

【C 言語 API】

```
ER ercd = ena_int(INTNO intno)
```

【パラメータ】

INTNO	intno	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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し）
E_NOSPT	未サポートエラー（ena _int がサポートされていない）
E_PAR	パラメータエラー（intno が不正，その他の条件については機能の項を参照すること）
E_OACV〔P〕	オブジェクトアクセス違反（システム状態に対する通常操作 2 が許可されていない）
E_OBJ	オブジェクト状態エラー（intno で指定した割込み要求ラインに対して割込み要求ライン属性が設定されていない）

【機能】

intno で指定した割込み要求ライン（対象割込み要求ライン）の割込み要求禁止フラグをクリアする．

ターゲット定義で，対象割込み要求ラインの割込み要求禁止フラグをクリアできない場合には，E_PAR エラーとなる．具体的には，対象割込み要求ラインに対して割込み要求禁止フラグがサポートされていない場合や，マルチプロセッサ対応カーネルにおいて，ena_int を呼び出したタスクが割り付けられているプロセッサから，対象割込み要求ラインの割込み要求禁止フラグが操作できない場合が，これに該当する．

ターゲット定義で，割込み要求禁止フラグの振舞いが，この仕様の規定と異なる場合がある．特にマルチプロセッサ対応カーネルでは，あるプロセッサから ena_int を呼び出して割込み要求禁止フラグをクリアしても，他のプロセッサに対しては割込みがマスク解除されない場合がある．

ターゲット定義で，ena_int がサポートされていない場合がある．ena_int がサポートされている場合には，TOPPERS_SUPPORT_ENA_INT がマクロ定義される．サポートされていない場合に ena_int を

呼び出すと，E_NOSPT エラーが返るか，リンク時にエラーとなる．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様で実装定義としていた intno の意味を標準化した．

CPU ロック状態でも呼び出せるものとした．

ref_int　　割込み要求ラインの参照〔T〕

【C 言語 API】

```
ER ercd = ref_int(INTNO intno, T_RINT *pk_rint)
```

☆未完成

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，ref_int をサポートしない．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，ref_int をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，ref_int をサポートしない．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，ref_int をサポートしない．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様に定義されていないサービスコールである．

chg_ipm　　割込み優先度マスクの変更〔T〕

【C 言語 API】

```
ER ercd = chg_ipm(PRI intpri)
```

【パラメータ】

PRI	intpri	割込み優先度マスク

【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_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_PAR	パラメータエラー（intpri が不正）
E_OACV〔P〕	オブジェクトアクセス違反（システム状態に対する通常操作 2 が許可されていない）

【機能】

割込み優先度マスクを，intpri で指定した値に変更する．

intpri は，TMIN_INTPRI 以上，TIPM_ENAALL 以下でなければならない．ただし，ターゲット定義の拡張として，TMIN_INTPRI よりも小さい値を指定できる場合がある．

【補足説明】

割込み優先度マスクを TIPM_ENAALL に変更した場合，ディスパッチ保留状態が解除され，ディスパッチが起こる可能性がある．また，タスク例外処理ルーチンの実行が開始される可能性がある．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，chg_ipm をサポートしない．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様では，サービスコールの名称およびパラメータの名称が実装定義となっているサービスコールである．

get_ipm　　割込み優先度マスクの参照〔T〕

【C 言語 API】

```
ER ercd = get_ipm(PRI *p_intpri)
```

【パラメータ】

PRI *	p_intpri	割込み優先度マスクを入れるメモリ領域へのポインタ

【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
PRI	intpri	割込み優先度マスク

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_OACV〔P〕	オブジェクトアクセス違反（システム状態に対する参照操作が許可されていない）
E_MACV〔P〕	メモリアクセス違反（p_intpri が指すメモリ領域への書込みアクセスが許可されていない）

【機能】

割込み優先度マスクの現在値を参照する．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，get_ipm をサポートしない．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様では，サービスコールの名称およびパラメータの名称が実装定義となっているサービスコールである．

4.10. CPU 例外管理機能

CPU 例外ハンドラは，カーネルが実行を制御する処理単位である．CPU 例外ハンドラは，CPU 例外ハンドラ番号と呼ぶオブジェクト番号によって識別する．

保護機能対応カーネルにおいて，CPU 例外ハンドラは，カーネルドメインに属する．

CPU 例外ハンドラ属性に標準で指定できる属性はないが，ターゲットによってはターゲット定義の CPU 例外ハンドラ属性を指定できる場合がある．ターゲット定義の CPU 例外ハンドラ属性として，次の属性を予約している．

TA_DIRECT	－	CPU 例外ハンドラを直接呼び出す

C 言語による CPU 例外ハンドラの記述形式は次の通り．

```
void cpu_exception_handler(void *p_excinf)
{
      CPU 例外ハンドラ本体
}
```

p_excinf には，CPU 例外の情報を記憶しているメモリ領域の先頭番地が渡される．これは，CPU 例外ハンドラ内で，CPU 例外発生時の状態を参照する際に必要となる．

DEF_EXC　CPU 例外ハンドラの定義〔S〕
def_exc　CPU 例外ハンドラの定義〔TD〕

【静的 API】

```
DEF_EXC(EXCNO excno, {ATR excatr, EXCHDR exchdr})
```

【C 言語 API】

```
ER ercd = def_exc(EXCNO excno, const T_DEXC *pk_dexc)
```

【パラメータ】

EXCNO	excno	CPU 例外ハンドラ番号
T_DEXC *	pk_dexc	CPU 例外ハンドラの定義情報を入れたパケットへのポインタ（静的 API を除く）

＊CPU 例外ハンドラの定義情報（パケットの内容）

ATR	excatr	CPU 例外ハンドラ属性
EXCHDR	exchdr	CPU 例外ハンドラの先頭番地

【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_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_RSATR	予約属性（excatr が不正または使用できない，属する保護ドメインかクラスが不正）
E_PAR	パラメータエラー（excno，exchdr が不正）
E_OACV〔sP〕	オブジェクトアクセス違反（システム状態に対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（pk_dexc が指すメモリ領域への読出しアクセスが許可されていない）
E_OBJ	オブジェクト状態エラー（定義済みの CPU 例外ハンドラ番号に対する再定義，未定義の CPU 例外ハンドラ番号に対する定義解除）

【機能】

excno で指定した CPU 例外ハンドラ番号（対象 CPU 例外ハンドラ番号）に対して，各パラメータで指定した CPU 例外ハンドラ定義情報に従って，CPU 例外ハンドラを定義する．ただし，def_exc において pk_dexc を NULL にした場合には，対象 CPU 例外ハンドラ番号に対する CPU 例外ハンドラの定義を解除する．

静的 API においては，excno と excatr は整数定数式パラメータ，exchdr は一般定数式パラメータである．

CPU 例外ハンドラを定義する場合（DEF_EXC の場合および def_exc において pk_dexc を NULL 以外にした場合）で，対象 CPU 例外ハンドラ番号に対してすでに CPU 例外ハンドラが定義されている場合には，E_OBJ エラーとなる．

保護機能対応カーネルにおいて，DEF_EXC は，カーネルドメインの囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．また，def_exc で CPU 例外ハンドラを定義する場合には，CPU 例外ハンドラの属する保護ドメインを設定する必要はなく，CPU 例外ハンドラ属性に TA_DOM(domid)を指定した場合には E_RSATR エラーとなる．ただし，TA_DOM(TDOM_SELF)を指定した場合には，指定が無視され，E_RSATR エラーは検出されない．

マルチプロセッサ対応カーネルで，登録する CPU 例外ハンドラの属するクラスの初期割付けプロセッサが，その CPU 例外が発生するプロセッサでない場合には，E_RSATR エラーとなる．

CPU 例外ハンドラの定義を解除する場合（def_exc において pk_dexc を NULL にした場合）で，対象 CPU 例外ハンドラ番号に対して CPU 例外ハンドラが定義されていない場合には，E_OBJ エラーとなる．また，対象 CPU 例外ハンドラ番号に対して定義された CPU 例外ハンドラが，静的 API で定義されたも

のである場合には，ターゲット定義で E_OBJ エラーとなる場合がある．

静的 API において，exchdr が不正である場合に E_PAR エラーが検出されるか否かは，ターゲット定義である．

【TOPPERS/ASP カーネルにおける規定】
ASP カーネルでは，DEF_EXC のみをサポートする．

【TOPPERS/FMP カーネルにおける規定】
FMP カーネルでは，DEF_EXC のみをサポートする．

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，DEF_EXC のみをサポートする．

【TOPPERS/SSP カーネルにおける規定】
SSP カーネルでは，DEF_EXC のみをサポートする．

【μITRON4.0 仕様との関係】
def_exc によって，定義済みの CPU 例外ハンドラを再定義しようとした場合に，E_OBJ エラーとすることにした．

xsns_dpn　　CPU 例外発生時のディスパッチ保留状態の参照〔TI〕

【C 言語 API】

```
bool_t stat = xsns_dpn(void *p_excinf)
```

【パラメータ】

void *	p_excinf	PU 例外の情報を記憶しているメモリ領域の先頭番地

【リターンパラメータ】

bool_t	state	ディスパッチ保留状態

【機能】
CPU 例外発生時のディスパッチ保留状態を参照する．具体的な振舞いは以下の通り．

実行中の CPU 例外ハンドラの起動原因となった CPU 例外が，カーネル管理外の CPU 例外でなく，タスクコンテキストで発生し，そのタスクがディスパッチ保留状態でなかった場合に false，そうでない

場合に true が返る．

保護機能対応のカーネルにおいて，xsns_dpn をタスクコンテキストから呼び出した場合には，true が返る．

p_excinf には，CPU 例外ハンドラに渡される p_excinf パラメータをそのまま渡す．

【使用方法】

xsns_dpn は，CPU 例外ハンドラの中で，どのようなリカバリ処理が可能かを判別したい場合に使用する．xsns_dpn が false を返した場合（true を返した場合ではないので注意すること），非タスクコンテキスト用のサービスコールを用いて CPU 例外を起こしたタスクよりも優先度の高いタスクを起動または待ち解除し，そのタスクでリカバリ処理を行うことができる．ただし，CPU 例外を起こしたタスクが最高優先度の場合には，この方法でリカバリ処理を行うことはできない．

【使用上の注意】

xsns_dpn は，E_CTX エラーを返すことがないために〔TI〕となっているが，CPU 例外ハンドラから呼び出すためのものである．CPU 例外ハンドラ以外から呼び出した場合や，p_excinf に正しい値を渡さなかった場合，xsns_dpn が返す値は意味を持たない．

どちらの条件で true が返るか間違いやすいので注意すること．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，xsns_dpn をサポートしない．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様に定義されていないサービスコールである．

【仕様決定の理由】

保護機能対応のカーネルにおいては，xsns_dpn をユーザドメインから呼び出すことは禁止すべきである．ユーザドメインの実行中は，必ずタスクコンテキストであるため，xsns_dpn をタスクコンテキストから呼び出した場合に必ず true を返す仕様とすることで，xsns_dpn をユーザドメインから呼び出すことを実質的に禁止している．

xsns_xpn　　CPU 例外発生時のタスク例外処理保留状態の参照〔TI〕

【C 言語 API】

```
bool_t stat = xsns_xpn(void *p_excinf)
```

【パラメータ】

void *	p_excinf	例外の情報を記憶しているメモリ領域の先頭番地

【リターンパラメータ】

bool_t	state	タスク例外処理保留状態

【機能】

CPU 例外発生時にタスク例外処理ルーチンを実行開始できない状態であったかを参照する．具体的な振舞いは以下の通り．

実行中の CPU 例外ハンドラの起動原因となった CPU 例外が，カーネル管理外の CPU 例外でなく，タスクコンテキストで発生し，そのタスクがタスク例外処理ルーチンを実行開始できる状態であった場合に false，そうでない場合に true が返る．

保護機能対応カーネルにおいて，CPU 例外が発生したタスクがユーザタスクの場合には，ユーザスタック領域の残りが少なく，タスク例外処理ルーチンを実行開始できない（タスク例外処理ルーチンを実行開始しようとすると，タスク例外実行開始時スタック不正例外が発生する）場合にも，true を返す．

保護機能対応のカーネルにおいて，xsns_xpn をタスクコンテキストから呼び出した場合には，true が返る．

p_excinf には，CPU 例外ハンドラに渡される p_excinf パラメータをそのまま渡す．

【使用方法】

xsns_xpn は，CPU 例外ハンドラの中で，どのようなリカバリ処理が可能かを判別したい場合に使用する．xsns_xpn が false を返した場合（true を返した場合ではないので注意すること），非タスクコンテキスト用のサービスコールを用いて CPU 例外を起こしたタスクにタスク例外を要求し，タスク例外処理ルーチンでリカバリ処理を行うことができる．

【使用上の注意】

xsns_xpn は，E_CTX エラーを返すことがないために〔TI〕となっているが，CPU 例外ハンドラから呼び出すためのものである．CPU 例外ハンドラ以外から呼び出した場合や，p_excinf に正しい値を渡さなかった場合，xsns_xpn が返す値は意味を持たない．

どちらの条件で true が返るか間違いやすいので注意すること．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，xsns_xpn をサポートしない．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様に定義されていないサービスコールである．

【仕様決定の理由】

保護機能対応のカーネルにおいては，xsns_xpn をユーザドメインから呼び出すことは禁止すべきである．ユーザドメインの実行中は，必ずタスクコンテキストであるため，xsns_xpn をタスクコンテキストから呼び出した場合に必ず true を返す仕様とすることで，xsns_xpn をユーザドメインから呼び出すことを実質的に禁止している．

4.11. 拡張サービスコール管理機能

拡張サービスコールは，非特権モードで実行される処理単位から，特権モードで実行すべきルーチンを呼び出すための機能である．特権モードで実行するルーチンを，拡張サービスコールと呼ぶ．拡張サービスコールは，特権モードで実行される処理単位からも呼び出すことができる．

保護機能対応カーネルにおいて，拡張サービスコールは，カーネルドメインに属する．拡張サービスコールは，それを呼び出す処理単位とは別の処理単位であり，拡張サービスコールからカーネルオブジェクトをアクセスする場合には，拡張サービスコールがアクセスの主体となる．そのため，拡張サービスコールからは，すべてのカーネルオブジェクトに対して，すべての種別のアクセスを行うことが許可される．

保護機能対応でないカーネルでは，非特権モードと特権モードの区別がないため，拡張サービスコール管理機能をサポートしない．

C 言語による拡張サービスコールの記述形式は次の通り．

```
ER_UINT extended_svc(intptr_t par1, intptr_t par2, intptr_t par3,
                                    intptr_t par4, intptr_t par5, ID cdmid)
{
    拡張サービスコール本体
}
```

cdmid には，拡張サービスコールを呼び出した処理単位が属する保護ドメインの ID 番号が渡される．すなわち，拡張サービスコールから呼び出した場合には TDOM_KERNEL（＝-1）が，タスク本体（拡張サービスコールを除く）から呼び出した場合にはそのタスク（自タスク）の属する保護ドメイン ID が

渡される．

par1～par5 には，拡張サービスコールに対するパラメータが渡される．

拡張サービスコール管理機能に関連するカーネル構成マクロは次の通り．

TMAX_FNCD	拡張サービスコールの機能番号の最大値（動的生成対応カーネルでは，登録できる拡張サービスコールの数に一致）

【TOPPERS/ASP カーネルにおける規定】
ASP カーネルでは，拡張サービスコール管理機能をサポートしない．

【TOPPERS/FMP カーネルにおける規定】
FMP カーネルでは，拡張サービスコール管理機能をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，拡張サービスコール管理機能をサポートする．

【TOPPERS/SSP カーネルにおける規定】
SSP カーネルでは，拡張サービスコール管理機能をサポートしない．

【未決定事項】
動的生成対応カーネルにおいて TMAX_FNCD を設定する方法については，現時点では未決定である．

【μITRON4.0 仕様との関係】
この仕様では，拡張サービスコールに対するパラメータを，intptr_t 型のパラメータ 5 個に固定した．

拡張サービスコールに，それを呼び出した処理単位が属する保護ドメインの ID 番号を渡す機能を追加した．

TMAX_FNCD は，μITRON4.0 仕様に規定されていないカーネル構成マクロである．

DEF_SVC　　拡張サービスコールの定義〔SP〕
def_svc　　拡張サービスコールの定義〔TPD〕

【静的 API】

DEF_SVC(FN fncd, {ATR svcatr, EXTSVC extsvc, SIZE stksz})

【C 言語 API】

```
ER ercd = def_svc(FN fncd, const T_DSVC *pk_dsvc)
```

【パラメータ】

FN	fncd	拡張サービスコールの機能コード
T_DSVC *	pk_dsvc	拡張サービスコールの定義情報を入れたパケットへのポインタ（静的 API を除く）

＊拡張サービスコールの定義情報（パケットの内容）

ATR	svcatr	拡張サービスコール属性
EXTSVC	extsvc	拡張サービスコールの先頭番地
SIZE	stksz	拡張サービスコールで使用するスタックサイズ

【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_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_RSATR	予約属性（svcatr が不正または使用できない，属する保護ドメインかクラスが不正）
E_PAR	パラメータエラー（fncd，extsvc が不正）
E_OACV〔sP〕	オブジェクトアクセス違反（システム状態に対する管理操作が許可されていない）
E_MACV〔sP〕	メモリアクセス違反（pk_dsvc が指すメモリ領域への読出しアクセスが許可されていない）
E_OBJ	オブジェクト状態エラー（定義済みの機能コードに対する再定義，未定義の機能コードに対する定義解除）

【機能】

fncd で指定した機能コード（対象機能コード）に対して，各パラメータで指定した拡張サービスコール定義情報に従って，拡張サービスコールを定義する．

ただし，def_svc において pk_dsvc を NULL にした場合には，対象機能コードに対する拡張サービスコールの定義を解除する．

静的 API においては，fncd，svcatr，stksz は整数定数式パラメータ，svchdr は一般定数式パラメータである．

拡張サービスコールを定義する場合（DEF_SVC の場合および def_svc において pk_dsvc を NULL 以外にした場合）で，対象機能コードに対してすでに拡張サービスコールが定義されている場合には，E_OBJ エラーとなる．

DEF_SVC は，カーネルドメインの囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．また，def_svc で拡張サービスコールを定義する場合には，拡張サービスコールの属する保護ドメインを設定する必要はなく，拡張サービスコール属性に TA_DOM(domid)を指定した場合には E_RSATR エラーとなる．ただし，TA_DOM(TDOM_SELF)を指定した場合には，指定が無視され，E_RSATR エラーは検出されない．

マルチプロセッサ対応カーネルでは，DEF_SVC は，クラスの囲みの外に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．また，def_svc で拡張サービスコールを定義する場合には，拡張サービスコールの属するクラスを設定する必要はなく，拡張サービスコール属性に TA_CLS(clsid)を指定した場合には E_RSATR エラーとなる．ただし，TA_CLS(TCLS_SELF)を指定した場合には，指定が無視され，E_RSATR エラーは検出されない．

拡張サービスコールの定義を解除する場合（def_svc において pk_dsvc を NULL にした場合）で，対象機能コードに対して拡張サービスコールが定義されていない場合には，E_OBJ エラーとなる．

拡張サービスコールの機能コードには，正の値を用いる．fncd が 0 または負の値の場合には，E_PAR エラーとなる．また，fncd が TMAX_FNCD よりも大きい場合にも，E_PAR エラーとなる．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，DEF_SVC のみをサポートする．

【μITRON4.0 仕様との関係】

拡張サービスコールの定義情報に，stksz（拡張サービスコールで使用するスタックサイズ）を追加した．

extsvc のデータ型を，EXTSVC に変更した．

cal_svc　　拡張サービスコールの呼出し〔TIP〕

【C 言語 API】

```
ER_UINT ercd = cal_svc(FN fcnd, intptr_t par1, intptr_t par2,
                       intptr_t par3, intptr_t par4, intptr_t par5)
```

【パラメータ】

FN	fncd	呼び出す拡張サービスコールの機能コード
intptr_t	par1	拡張サービスコールへの第 1 パラメータ
intptr_t	par2	拡張サービスコールへの第 2 パラメータ
intptr_t	par3	拡張サービスコールへの第 3 パラメータ
intptr_t	par4	拡張サービスコールへの第 4 パラメータ
intptr_t	par5	拡張サービスコールへの第 5 パラメータ

【リターンパラメータ】

ER_UINT	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_SYS	システムエラー（拡張サービスコールのネストレベルが上限を超える）
E_RSFN	予約機能コード（fncd が不正．fncd に対して拡張サービスコールが定義されていない）
E_NOMEM	メモリ不足（スタックの残り領域が不足）

＊その他，拡張サービスコールが返すエラーコードがそのまま返る．

【機能】

fncd で指定した機能コードの拡張サービスコールを，par1，par2，…，par5 をパラメータとして呼び出し，拡張サービスコールの返値を返す．

fncd が不正な値である場合や，fncd で指定した機能コードに対して拡張サービスコールが定義されていない場合には，E_RSFN エラーとなる．

また，タスクコンテキストから呼び出した場合には，次のエラーが検出されるスタックの残り領域が，拡張サービスコールで使用するスタックサイズよりも小さい場合には，E_NOMEM エラーとなる．また，拡張サービスコールのネストレベルが上限（255）を超える場合には，E_SYS エラーが返る．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様では，cal_svc でカーネルのサービスコールを呼び出せるかどうかは実装定義としているが，この仕様では，カーネルのサービスコールを呼び出せないこととした．

拡張サービスコールが呼び出される時に，スタックの残り領域のサイズをチェックする機能を追加した．

拡張サービスコールに対するパラメータを，intptr_t 型のパラメータ 5 個に固定し，cal_svc から返るエラー（E_SYS，E_RSFN，E_NOMEM）について規定した．

【仕様決定の理由】
パラメータの型と数を固定したのは，型チェックを厳密にできるようにし，パラメータをコンパイラやコーリングコンベンションによらずに正しく渡せるようにするためである．

4.12. システム構成管理機能

システム構成管理機能には，非タスクコンテキスト用スタック領域を設定する機能，初期化ルーチンと終了処理ルーチンを登録する機能，カーネル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が含まれる．

非タスクコンテキスト用スタック領域は，非タスクコンテキストで実行される処理単位が用いるスタック領域である．

保護機能対応カーネルにおいて，非タスクコンテキスト用のスタック領域は，カーネルの用いるオブジェクト管理領域と同様に扱われる．

初期化ルーチンは，カーネルが実行を制御する処理単位で，カーネルの動作開始の直前に，カーネル非動作状態で実行される．

保護機能対応カーネルにおいて，初期化ルーチンは，カーネルドメインに属する．

初期化ルーチン属性に指定できる属性はない．そのため初期化ルーチン属性には，TA_NULL を指定しなければならない．

C 言語による初期化ルーチンの記述形式は次の通り．

```
void initialization_routine(intptr_t exinf)
{
    初期化ルーチン本体
}
```

exinf には，初期化ルーチンの拡張情報が渡される．

終了処理ルーチンは，カーネルが実行を制御する処理単位で，カーネルの動作終了の直後に，カーネ

ル非動作状態で実行される．

保護機能対応カーネルにおいて，終了処理ルーチンは，カーネルドメインに属する．

終了処理ルーチン属性に指定できる属性はない．そのため終了処理ルーチン属性には，TA_NULL を指定しなければならない．

C 言語による終了処理ルーチンの記述形式は次の通り．

```
void termination_routine(intptr_t exinf)
{
    終了処理ルーチン本体
}
```

exinf には，終了処理ルーチンの拡張情報が渡される．

【μITRON4.0 仕様との関係】
非タスクコンテキスト用スタック領域の設定と，終了処理ルーチンは，μITRON4.0 仕様に規定されていない機能である．

DEF_ICS　　非タスクコンテキスト用スタック領域の設定〔S〕

【静的 API】

```
DEF_ICS({ SIZE istksz, STK_T *istk })
```

【パラメータ】

＊非タスクコンテキスト用スタック領域の設定情報

SIZE	istksz	非タスクコンテキスト用スタック領域のサイズ（バイト数）
STK_T	istk	非タスクコンテキスト用スタック領域の先頭番地

【エラーコード】

E_RSATR	予約属性（属する保護ドメインかクラスが不正）
E_PAR	パラメータエラー（istksz，istk が不正）
E_NOMEM	メモリ不足（非タスクコンテキスト用スタック領域が確保できない）
E_OBJ	オブジェクト状態エラー（非タスクコンテキスト用スタック領域がすでに設定されている，その他の条件については機能の項を参照すること）

【機能】

各パラメータで指定した非タスクコンテキスト用スタック領域の設定情報に従って，非タスクコンテキスト用スタック領域を設定する．

istksz は整数定数式パラメータ，istk は一般定数式パラメータである．コンフィギュレータは，静的 API のメモリ不足（E_NOMEM）エラーを検出することができない．

istk を NULL とした場合，istksz で指定したサイズのスタック領域を，コンフィギュレータが確保する．istksz にターゲット定義の制約に合致しないサイズを指定した時には，ターゲット定義の制約に合致するようにサイズを大きい方に丸めて確保する．

istk に NULL 以外を指定した場合，istk と istksz で指定したスタック領域は，アプリケーションで確保しておく必要がある．スタック領域をアプリケーションで確保する方法については，「2.15.3 カーネル共通マクロ」の節を参照すること．その方法に従わず，istk や istksz にターゲット定義の制約に合致しない先頭番地やサイズを指定した時には，E_PAR エラーとなる．

保護機能対応カーネルでは，istk と istksz で指定した非タスクコンテキスト用のスタック領域がカーネル専用のメモリオブジェクトに含まれない場合，E_OBJ エラーとなる．

DEF_ICS により非タスクコンテキスト用スタック領域を設定しない場合，ターゲット定義のデフォルトのサイズのスタック領域を，コンフィギュレータが確保する．

マルチプロセッサ対応カーネルでは，非タスクコンテキスト用スタック領域はプロセッサ毎に確保する必要がある．DEF_ICS により設定する非タスクコンテキスト用スタック領域は，DEF_ICS の記述をその囲みの中に含むクラスの初期割付けプロセッサが使用する．そのプロセッサに対してすでに非タスクコンテキスト用スタック領域が設定されている場合には，E_OBJ エラーとなる．

保護機能対応カーネルにおいて，DEF_ICS は，カーネルドメインの囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，istk には NULL を指定しなくてはならず，その場合でも，コンフィギュレータは非タスクコンテキスト用のスタック領域を確保しない．これは，SSP カーネルでは，すべての処理単位が共有スタック領域を使用し，非タスクコンテキストのみが用いるスタック領域を持たないためである．そのため，DEF_ICS の役割は，非タスクコンテキストが用いるスタック領域のサイズを指定することのみとなる．itsk に NULL 以外を指定した場合には，E_PAR エラーとなる．

共有スタック領域の設定方法については，DEF_STK の項を参照すること．

【μITRON4.0 仕様との関係】

μITRON4.0 仕様に定義されていない静的 API である．

DEF_STK　　共有スタック領域の設定〔S〕

【静的 API】

```
DEF_STK({ SIZE stksz, STK_T *stk })
```

【パラメータ】

＊共有スタック領域の設定情報

SIZE	stksz	共有スタック領域のサイズ（バイト数）
STK_T	stk	共有スタック領域の先頭番地

【エラーコード】

E_RSATR	予約属性（属する保護ドメインかクラスが不正）
E_PAR	パラメータエラー（stksz，stk が不正）
E_NOMEM	メモリ不足（共有スタック領域が確保できない）
E_OBJ	オブジェクト状態エラー（共有スタック領域がすでに設定されている，その他の条件については機能の項を参照すること）

【サポートするカーネル】

DEF_STK は，TOPPERS/SSP カーネルのみがサポートする静的 API である．他のカーネルは，DEF_STK をサポートしない．

【機能】

各パラメータで指定した共有スタック領域の設定情報に従って，共有スタック領域を設定する．

stksz は整数定数式パラメータ，stk は一般定数式パラメータである．コンフィギュレータは，静的 API

のメモリ不足（E_NOMEM）エラーを検出することができない.

stk を NULL とした場合, stksz で指定したサイズのスタック領域を, コンフィギュレータが確保する. stksz にターゲット定義の制約に合致しないサイズを指定した時には, ターゲット定義の制約に合致するようにサイズを大きい方に丸めて確保する.

stk に NULL 以外を指定した場合，stk と stksz で指定したスタック領域は，アプリケーションで確保しておく必要がある．スタック領域をアプリケーションで確保する方法については,「2.15.3 カーネル共通マクロ」の節を参照すること.
その方法に従わず，stk や stksz にターゲット定義の制約に合致しない先頭番地やサイズを指定した時には，E_PAR エラーとなる.

コンフィギュレータは，各タスクのスタック領域のサイズと，非タスクコンテキスト用のスタック領域のサイズから，共有スタック領域に必要なサイズを計算する．DEF_STK により共有スタック領域を設定しない場合，必要なサイズの共有スタック領域を，コンフィギュレータが確保する.

stksz に指定したスタック領域のサイズが，共有スタック領域に必要なサイズよりも小さい場合，コンフィギュレータは警告メッセージを出力する.

【μITRON4.0 仕様との関係】
μITRON4.0 仕様に定義されていない静的 API である.

ATT_INI　　初期化ルーチンの追加〔S〕

【静的 API】

ATT_INI({ ATR iniatr, intptr_t exinf, INIRTN inirtn })

【パラメータ】
＊初期化ルーチンの追加情報

ATR	iniatr	初期化ルーチン属性
intptr_t	exinf	初期化ルーチンの拡張情報
INIRTN	inirtn	初期化ルーチンの先頭番地

【エラーコード】

E_RSATR	予約属性 (iniatr が不正または使用できない, 属する保護ドメインが不正)
E_PAR	パラメータエラー（inirtn が不正）

【機能】
　各パラメータで指定した初期化ルーチン追加情報に従って，初期化ルーチンを追加する．

　iniatr は整数定数式パラメータ，exinf と inirtn は一般定数式パラメータである．

　保護機能対応カーネルにおいて，ATT_INI は，カーネルドメインの囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．

inirtn が不正である場合に E_PAR エラーが検出されるか否かは，ターゲット定義である．

【補足説明】
　マルチプロセッサ対応カーネルでは，クラスに属さないグローバル初期化ルーチンはマスタプロセッサで実行され，クラスに属するローカル初期化ルーチンはそのクラスの初期割付けプロセッサにより実行される．

ATT_TER　　終了処理ルーチンの追加〔S〕

【静的 API】

```
ATT_TER({ ATR teratr, intptr_t exinf, TERRTN terrtn })
```

【パラメータ】

＊終了処理ルーチンの追加情報

ATR	teratr	終了処理ルーチン属性
intptr_t	exinf	終了処理ルーチンの拡張情報
TERRTN	terrtn	終了処理ルーチンの先頭番地

【エラーコード】

E_RSATR	予約属性（teratr が不正または使用できない，属する保護ドメインが不正）
E_PAR	パラメータエラー（terrtn が不正）

【機能】
　各パラメータで指定した終了処理ルーチン追加情報に従って，終了処理ルーチンを追加する．

　teratr は整数定数式パラメータ，exinf と terrtn は一般定数式パラメータである．

　保護機能対応カーネルにおいて，ATT_TER は，カーネルドメインの囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．

terrtn が不正である場合に E_PAR エラーが検出されるか否かは，ターゲット定義である．

【補足説明】
マルチプロセッサ対応カーネルでは，クラスに属さないグローバル終了処理ルー¥チンはマスタプロセッサで実行され，クラスに属するローカル終了処理ルーチンはそのクラスの初期割付けプロセッサにより実行される．

【μITRON4.0 仕様との関係】
μITRON4.0 仕様に定義されていない静的 API である．

ref_cfg　　コンフィギュレーション情報の参照〔T〕

【C 言語 API】

```
ER ercd = ref_cfg(T_RCFG *pk_rcfg)
```

☆未完成

【TOPPERS/ASP カーネルにおける規定】
ASP カーネルでは，ref_cfg をサポートしない．

【TOPPERS/FMP カーネルにおける規定】
FMP カーネルでは，ref_cfg をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】
HRP2 カーネルでは，ref_cfg をサポートしない．

【TOPPERS/SSP カーネルにおける規定】
SSP カーネルでは，ref_cfg をサポートしない．

ref_ver　　バージョン情報の参照〔T〕

【C 言語 API】

```
ER ercd = ref_ver(T_RVER *pk_rver)
```

☆未完成

【TOPPERS/ASP カーネルにおける規定】

ASP カーネルでは，ref_ver をサポートしない．

【TOPPERS/FMP カーネルにおける規定】

FMP カーネルでは，ref_ver をサポートしない．

【TOPPERS/HRP2 カーネルにおける規定】

HRP2 カーネルでは，ref_ver をサポートしない．

【TOPPERS/SSP カーネルにおける規定】

SSP カーネルでは，ref_ver をサポートしない．

5. リファレンス

5.1. サービスコール一覧

(1) タスク管理機能

ER_ID tskid = acre_tsk(const T_CTSK *pk_ctsk)	〔TD〕
ER ercd = sac_tsk(ID tskid, const ACVCT *p_acvct)	〔TPD〕
ER ercd = del_tsk(ID tskid)	〔TD〕
ER ercd = act_tsk(ID tskid)	〔T〕
ER ercd = iact_tsk(ID tskid)	〔I〕
ER ercd = mact_tsk(ID tskid, ID prcid)	〔TM〕
ER ercd = imact_tsk(ID tskid, ID prcid)	〔IM〕
ER_UINTactcnt = can_act(ID tskid)	〔T〕
ER ercd = mig_tsk(ID tskid, ID prcid)	〔TM〕
ER ercd = ext_tsk()	〔T〕
ER ercd = ter_tsk(ID tskid)	〔T〕
ER ercd = chg_pri(ID tskid, PRI tskpri)	〔T〕
ER ercd = get_pri(ID tskid, PRI *p_tskpri)	〔T〕
ER ercd = get_inf(intptr_t *p_exinf)	〔T〕
ER ercd = ref_tsk(ID tskid, T_RTSK *pk_rtsk)	〔T〕

(2) タスク付属同期機能

ER ercd = slp_tsk()	〔T〕
ER ercd = tslp_tsk(TMO tmout)	〔T〕
ER ercd = wup_tsk(ID tskid)	〔T〕
ER ercd = iwup_tsk(ID tskid)	〔I〕
ER_UINT wupcnt = can_wup(ID tskid)	〔T〕
ER ercd = rel_wai(ID tskid)	〔T〕
ER ercd = irel_wai(ID tskid)	〔I〕
ER ercd = sus_tsk(ID tskid)	〔T〕
ER ercd = rsm_tsk(ID tskid)	〔T〕
ER ercd = dis_wai(ID tskid)	〔TP〕
ER ercd = idis_wai(ID tskid)	〔IP〕
ER ercd = ena_wai(ID tskid)	〔TP〕
ER ercd = iena_wai(ID tskid)	〔IP〕
ER ercd = dly_tsk(RELTIM dlytim)	〔T〕

(3)　タスク例外処理機能

ER ercd = def_tex(ID tskid, const T_DTEX *pk_dtex)	〔TD〕
ER ercd = ras_tex(ID tskid, TEXPTN rasptn)	〔T〕
ER ercd = iras_tex(ID tskid, TEXPTN rasptn)	〔I〕
ER ercd = dis_tex()	〔T〕
ER ercd = ena_tex()	〔T〕
bool_t state = sns_tex()	〔TI〕
ER ercd = ref_tex(ID tskid, T_RTEX *pk_rtex)	〔T〕
ER ercd = def_tex(ID tskid, const T_DTEX *pk_dtex)	〔TD〕

(4)　同期・通信機能

- セマフォ

ER_ID semid = acre_sem(const T_CSEM *pk_csem)	〔TD〕
ER ercd = sac_sem(ID semid, const ACVCT *p_acvct)	〔TPD〕
ER ercd = del_sem(ID semid)	〔TD〕
ER ercd = sig_sem(ID semid)	〔T〕
ER ercd = isig_sem(ID semid)	〔I〕
ER ercd = wai_sem(ID semid)	〔T〕
ER ercd = pol_sem(ID semid)	〔T〕
ER ercd = twai_sem(ID semid, TMO tmout)	〔T〕
ER ercd = ini_sem(ID semid)	〔T〕
ER ercd = ref_sem(ID semid, T_RSEM *pk_rsem)	〔T〕

- イベントフラグ

ER_ID flgid = acre_flg(const T_CFLG *pk_cflg)	〔TD〕
ER ercd = sac_flg(ID flgid, const ACVCT *p_acvct)	〔TPD〕
ER ercd = del_flg(ID flgid)	〔TD〕
ER ercd = set_flg(ID flgid, FLGPTN setptn)	〔T〕
ER ercd = iset_flg(ID flgid, FLGPTN setptn)	〔I〕
ER ercd = clr_flg(ID flgid, FLGPTN clrptn)	〔T〕
ER ercd = wai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn)	〔T〕
ER ercd = pol_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn)	〔T〕
ER ercd = pol_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn)	〔T〕
ER ercd = twai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn, TMO tmout)	〔T〕

ER ercd = ini_flg(ID flgid)
ER ercd = ref_flg(ID flgid, T_RFLG *pk_rflg)

- データキュー

ER_ID dtqid = acre_dtq(const T_CDTQ *pk_cdtq)	〔TD〕
ER ercd = sac_dtq(ID dtqid, const ACVCT *p_acvct)	〔TPD〕
ER ercd = del_dtq(ID dtqid)	〔TD〕
ER ercd = snd_dtq(ID dtqid, intptr_t data)	〔T〕
ER ercd = psnd_dtq(ID dtqid, intptr_t data)	〔T〕
ER ercd = ipsnd_dtq(ID dtqid, intptr_t data)	〔I〕
ER ercd = tsnd_dtq(ID dtqid, intptr_t data, TMO tmout)	〔T〕
ER ercd = fsnd_dtq(ID dtqid, intptr_t data)	〔T〕
ER ercd = ifsnd_dtq(ID dtqid, intptr_t data)	〔I〕
ER ercd = rcv_dtq(ID dtqid, intptr_t *p_data)	〔T〕
ER ercd = prcv_dtq(ID dtqid, intptr_t *p_data)	〔T〕
ER ercd = trcv_dtq(ID dtqid, intptr_t *p_data, TMO tmout)	〔T〕
ER ercd = ini_dtq(ID dtqid)	〔T〕
ER ercd = ref_dtq(ID dtqid, T_RDTQ *pk_rdtq)	〔T〕

- 優先度データキュー

ER_ID pdqid = acre_pdq(const T_CPDQ *pk_cpdq)	〔TD〕
ER ercd = sac_pdq(ID pdqid, const ACVCT *p_acvct)	〔TPD〕
ER ercd = del_pdq(ID pdqid)	〔TD〕
ER ercd = snd_pdq(ID pdqid, intptr_t data, PRI datapri)	〔T〕
ER ercd = psnd_pdq(ID pdqid, intptr_t data, PRI datapri)	〔T〕
ER ercd = ipsnd_pdq(ID pdqid, intptr_t data, PRI datapri)	〔I〕
ER ercd = tsnd_pdq(ID pdqid, intptr_t data,PRI datapri, TMO tmout)	〔T〕
ER ercd = rcv_pdq(ID pdqid, intptr_t *p_data, PRI *p_datapri)	〔T〕
ER ercd = prcv_pdq(ID pdqid, intptr_t *p_data, PRI *p_datapri)	〔T〕
ER ercd = trcv_pdq(ID pdqid, intptr_t *p_data, PRI *p_datapri, TMO tmout)	〔T〕
ER ercd = ini_pdq(ID pdqid)	〔T〕
ER ercd = ref_pdq(ID pdqid, T_RPDQ *pk_rpdq)	〔T〕

- メールボックス

  ER_ID mbxid = acre_mbx(const T_CMBX *pk_cmbx) 〔TDp〕
  ER ercd = del_mbx(ID mbxid) 〔TDp〕
  ER ercd = snd_mbx(ID mbxid, T_MSG *pk_msg) 〔Tp〕
  ER ercd = rcv_mbx(ID mbxid, T_MSG **ppk_msg) 〔Tp〕
  ER ercd = prcv_mbx(ID mbxid, T_MSG **ppk_msg) 〔Tp〕
  ER ercd = trcv_mbx(ID mbxid, T_MSG **ppk_msg, TMO tmout) 〔Tp〕
  ER ercd = ini_mbx(ID mbxid) 〔Tp〕
  ER ercd = ref_mbx(ID mbxid, T_RMBX *pk_rmbx) 〔Tp〕

- ミューテックス

  ER_ID mtxid = acre_mtx(const T_CMTX *pk_cmtx) 〔TD〕
  ER ercd = sac_mtx(ID mtxid, const ACVCT *p_acvct) 〔TPD〕
  ER ercd = del_mtx(ID mtxid) 〔TD〕
  ER ercd = loc_mtx(ID mtxid) 〔T〕
  ER ercd = ploc_mtx(ID mtxid) 〔T〕
  ER ercd = tloc_mtx(ID mtxid, TMO tmout) 〔T〕
  ER ercd = unl_mtx(ID mtxid) 〔T〕
  ER ercd = ini_mtx(ID mtxid) 〔T〕
  ER ercd = ref_mtx(ID mtxid, T_RMTX *pk_rmtx) 〔T〕
  ER_ID mtxid = acre_mtx(const T_CMTX *pk_cmtx) 〔TD〕
  ER ercd = sac_mtx(ID mtxid, const ACVCT *p_acvct) 〔TPD〕

- メッセージバッファ

☆未完成

- スピンロック

  ER_ID spnid = acre_spn(const T_CSPN *pk_cspn) 〔TMD〕
  ER ercd = sac_spn(ID spnid, const ACVCT *p_acvct) 〔TPMD〕
  ER ercd = del_spn(ID spnid) 〔TMD〕
  ER ercd = loc_spn(ID spnid) 〔TM〕
  ER ercd = iloc_spn(ID spnid) 〔IM〕
  ER ercd = try_spn(ID spnid) 〔TM〕
  ER ercd = itry_spn(ID spnid) 〔IM〕
  ER ercd = unl_spn(ID spnid) 〔TM〕
  ER ercd = iunl_spn(ID spnid) 〔IM〕
  ER ercd = ref_spnID spnid, T_RSPN *pk_rspn) 〔TM〕

(5)　メモリプール管理機能

- 固定長メモリプール

ER_ID mpfid = acre_mpf(const T_CMPF *pk_cmpf)	〔TD〕
ER ercd = sac_mpf(ID mpfid, const ACVCT *p_acvct)	〔TPD〕
ER ercd = del_mpf(ID mpfid)	〔TD〕
ER ercd = get_mpf(ID mpfid, void **p_blk)	〔T〕
ER ercd = pget_mpf(ID mpfid, void **p_blk)	〔T〕
ER ercd = tget_mpf(ID mpfid, void **p_blk, TMO tmout)	〔T〕
ER ercd = rel_mpf(ID mpfid, void *blk)	〔T〕
ER ercd = ini_mpf(ID mpfid)	〔T〕
ER ercd = ref_mpf(ID mpfid, T_RMPF *pk_rmpf)	〔T〕

(6)　時間管理機能

- システム時刻管理

ER ercd = get_tim(SYSTIM *p_systim)	〔T〕
ER ercd = get_utm(SYSUTM *p_sysutm)	〔TI〕

- 周期ハンドラ

ER_ID cycid = acre_cyc(const T_CCYC *pk_ccyc)	〔TD〕
ER ercd = sac_cyc(ID cycid, const ACVCT *p_acvct)	〔TPD〕
ER ercd = del_cyc(ID cycid)	〔TD〕
ER ercd = sta_cyc(ID cycid)	〔T〕
ER ercd = msta_cyc(ID cycid, ID prcid)	〔TM〕
ER ercd = stp_cyc(ID cycid)	〔T〕
ER ercd = ref_cyc(ID cycid, T_RCYC *pk_rcyc)	〔T〕

- アラームハンドラ

ER_ID almid = acre_alm(const T_CALM *pk_calm)	〔TD〕
ER ercd = sac_alm(ID almid, const ACVCT *p_acvct)	〔TPD〕
ER ercd = del_alm(ID almid)	〔TD〕
ER ercd = sta_alm(ID almid, RELTIM almtim)	〔T〕
ER ercd = ista_alm(ID almid, RELTIM almtim)	〔I〕
ER ercd = msta_alm(ID almid, RELTIM almtim, ID prcid)	〔TM〕
ER ercd = imsta_alm(ID almid, RELTIM almtim, ID prcid)	〔IM〕
ER ercd = stp_alm(ID almid)	〔T〕
ER ercd = istp_alm(ID almid)	〔I〕
ER ercd = ref_alm(ID almid, T_RALM *pk_ralm)	〔T〕

- オーバランハンドラ

ER ercd = def_ovr(const T_DOVR *pk_dovr)	〔TD〕
ER ercd = sta_ovr(ID tskid, OVRTIM ovrtim)	〔T〕
ER ercd = ista_ovr(ID tskid, OVRTIM ovrtim)	〔I〕
ER ercd = stp_ovr(ID tskid)	〔T〕
ER ercd = istp_ovr(ID tskid)	〔I〕
ER ercd = ref_ovr(ID tskid, T_ROVR *pk_rovr)	〔T〕

(7) システム状態管理機能

ER ercd = sac_sys(const ACVCT *p_acvct)	〔TPD〕
ER ercd = rot_rdq(PRI tskpri)	〔T〕
ER ercd = irot_rdq(PRI tskpri)	〔I〕
ER ercd = mrot_rdq(PRI tskpri, ID prcid)	〔TM〕
ER ercd = imrot_rdq(PRI tskpri, ID prcid)	〔IM〕
ER ercd = get_tid(ID *p_tskid)	〔T〕
ER ercd = iget_tid(ID *p_tskid)	〔I〕
ER ercd = get_did(ID *p_domid)	〔TP〕
ER ercd = get_pid(ID *p_prcid)	〔TM〕
ER ercd = iget_pid(ID *p_prcid)	〔IM〕
ER ercd = loc_cpu()	〔T〕
ER ercd = iloc_cpu()	〔I〕
ER ercd = unl_cpu()	〔T〕
ER ercd = iunl_cpu()	〔I〕
ER ercd = dis_dsp()	〔T〕
ER ercd = ena_dsp()	〔T〕
bool_t state = sns_ctx()	〔TI〕
bool_t state = sns_loc()	〔TI〕
bool_t state = sns_dsp()	〔TI〕
bool_t state = sns_dpn()	〔TI〕
bool_t state = sns_ker()	〔TI〕
ER ercd = ext_ker()	〔TI〕
ER ercd = ref_sys(T_RSYS *pk_rsys)	〔T〕

(8)　メモリオブジェクト管理機能

ER ercd = att_mem(const T_AMEM *pk_amem)	〔TPD〕
ER ercd = att_pma(const T_AMEM *pk_apma)	〔TPD〕
ER ercd = sac_mem(const void *base, const ACVCT *p_acvct)	〔TPD〕
ER ercd = det_mem(const void *base)	〔TPD〕
ER ercd = prb_mem(const void *base, SIZE size, ID tskid, MODE pmmode)	〔TP〕
ER ercd = ref_mem(const void *base, T_RMEM *pk_rmem)	〔TP〕

(9)　割込み管理機能

ER ercd = cfg_int(INTNO intno, const T_CINT *pk_cint)	〔TD〕
ER_ID isrid = acre_isr(const T_CISR *pk_cisr)	〔TD〕
ER ercd = sac_isr(ID isrid, const ACVCT *p_acvct)	〔TPD〕
ER ercd = del_isr(ID isrid)	〔TD〕
ER ercd = ref_isr(ID isrid, T_RISR *pk_risr)	〔T〕
ER ercd = def_inh(INHNO inhno, const T_DINH *pk_dinh)	〔TD〕
ER ercd = dis_int(INTNO intno)	〔T〕
ER ercd = ena_int(INTNO intno)	〔T〕
ER ercd = ref_int(INTNO intno, T_RINT *pk_rint)	〔T〕
ER ercd = chg_ipm(PRI intpri)	〔T〕
ER ercd = get_ipm(PRI *p_intpri)	〔T〕

(10)　CPU 例外管理機能

ER ercd = def_exc(EXCNO excno, const T_DEXC *pk_dexc)	〔TD〕
bool_t stat = xsns_dpn(void *p_excinf)	〔TI〕
bool_t stat = xsns_xpn(void *p_excinf)	〔TI〕

(11)　拡張サービスコール管理機能

ER ercd = def_svc(FN fncd, const T_DSVC *pk_dsvc)	〔TPD〕
ER_UINT ercd = cal_svc(FN fcnd, intptr_t par1, intptr_t par2, intptr_t par3, intptr_t par4, intptr_t par5)	〔TIP〕

(12)　システム構成管理機能

ER ercd = ref_cfg(T_RCFG *pk_rcfg)	〔T〕
ER ercd = ref_ver(T_RVER *pk_rver)	〔T〕

5.2. 静的 API 一覧

(1)　タスク管理機能

＊　保護機能対応でないカーネルの場合

CRE_TSK(ID tskid, {ATR tskatr, intptr_t exinf, TASK task, PRI itskpri, SIZE stksz, STK_T *stk })	〔S〕

＊　保護機能対応カーネルの場合

CRE_TSK(ID tskid, {ATR tskatr, intptr_t exinf, TASK task, PRI itskpri, SIZE stksz, STK_T *stk, SIZE sstksz, STK_T *sstk }) ※ sstksz および sstk の記述は省略することができる.	〔S〕

AID_TSK(uint_t notsk)	〔SD〕

SAC_TSK(ID tskid, {ACPTN acptn1, ACPTN acptn2, ACPTN acptn3, ACPTN acptn4 })	〔SP〕

DEF_EPR(ID tskid, { PRI exepri })	〔S〕

(2)　タスク付属同期機能

なし

(3)　タスク例外処理機能

DEF_TEX(ID tskid, {ATR texatr, TEXRTN texrtn })	〔S〕

(4)　同期・通信機能

- セマフォ

CRE_SEM(ID semid, {ATR sematr, uint_t isemcnt, uint_t maxsem })	〔S〕

AID_SEM(uint_t nosem)	〔SD〕

SAC_SEM(ID semid, {ACPTN acptn1, ACPTN acptn2, ACPTN acptn3, ACPTN acptn4 })	〔SP〕

- イベントフラグ

CRE_FLG(ID flgid, {ATR flgatr, FLGPTN iflgptn})	〔S〕
AID_FLG(uint_t noflg)	〔SD〕
SAC_FLG(ID flgid, {ACPTN acptn1, ACPTN acptn2, ACPTN acptn3, ACPTN acptn4})	〔SP〕

- データキュー

CRE_DTQ(ID dtqid, {ATR dtqatr, uint_t dtqcnt, void *dtqmb})	〔S〕
AID_DTQ(uint_t nodtq)	〔SD〕
SAC_DTQ(ID dtqid, {ACPTN acptn1, ACPTN acptn2, ACPTN acptn3, ACPTN acptn4})	〔SP〕

- 優先度データキュー

CRE_PDQ(ID pdqid, {ATR pdqatr, uint_t pdqcnt, PRI maxdpri, void *pdqmb})	〔S〕
AID_PDQ(uint_t nopdq)	〔SD〕
SAC_PDQ(ID pdqid, {ACPTN acptn1, ACPTN acptn2, ACPTN acptn3, ACPTN acptn4})	〔SP〕

- メールボックス

CRE_MBX(ID mbxid, {ATR mbxatr, PRI maxmpri, void *mprihd})	〔Sp〕
AID_MBX(uint_t nombx)	〔SpD〕

- ミューテックス

CRE_MTX(ID mtxid, {ATR mtxatr, PRI ceilpri})	〔S〕
AID_MTX(uint_t nomtx)	〔SD〕
SAC_MTX(ID mtxid, {ACPTN acptn1, ACPTN acptn2, ACPTN acptn3, ACPTN acptn4})	〔SP〕

- メッセージバッファ

☆未完成

- スピンロック

CRE_SPN(ID spnid, {ATR spnatr})	〔S〕
AID_SPN(uint_t nospn)	〔SMD〕
SAC_SPN(ID spnid, {ACPTN acptn1, ACPTN acptn2, ACPTN acptn3, ACPTN acptn4})	〔SPM〕

(5) メモリプール管理機能

- 固定長メモリプール

CRE_MPF(ID mpfid, {ATR mpfatr, uint_t blkcnt, uint_t blksz, MPF_T *mpf, void *mpfmb})	〔S〕
AID_MPF(uint_t nompf)	〔SD〕
CRE_CYC(ID cycid, {ATR cycatr, intptr_t exinf, CYCHDR cychdr, ACPTN acptn3, ACPTN acptn4})	〔SP〕

(6) 時間管理機能

- 周期ハンドラ

CRE_MPF(ID mpfid, {ATR mpfatr, uint_t blkcnt, uint_t blksz, RELTIM cyctim, RELTIM cycphs})	〔S〕
AID_CYC(uint_t nocyc)	〔SD〕
SAC_CYC(ID cycid, {ACPTN acptn1, ACPTN acptn2, ACPTN acptn3, ACPTN acptn4})	〔SP〕

- アラームハンドラ

CRE_ALM(ID almid, {ATR almatr, intptr_t exinf, ALMHDR almhdr})	〔S〕
AID_ALM(uint_t noalm)	〔SD〕

SAC_ALM(ID almid, {ACPTN acptn1, ACPTN acptn2, ACPTN acptn3, ACPTN acptn4 })	〔SP〕

- オーバランハンドラ

DEF_OVR({ ATR ovratr, OVRHDR ovrhdr })	〔S〕

(7) システム状態管理機能

SAC_SYS({ ACPTN acptn1, ACPTN acptn2, ACPTN acptn3, ACPTN acptn4 })	〔SP〕

(8) メモリオブジェクト管理機能

ATT_REG("メモリリージョン名",{ ATR regatr, void *base, SIZE size })	〔SP〕
ATT_SEC("セクション名", { ATR mematr, "メモリリージョン名" })	〔SP〕
ATA_SEC("セクション名", { ATR mematr, "メモリリージョン名" }, { ACPTN acptn1, ACPTN acptn2, ACPTN acptn3, ACPTN acptn4 })	〔SP〕
LNK_SEC("セクション名", { "メモリリージョン名" })	〔SP〕
ATT_MOD("オブジェクトモジュール名")	〔SP〕
ATA_MOD("オブジェクトモジュール名", { ACPTN acptn1, ACPTN acptn2, ACPTN acptn3, ACPTN acptn4 })	〔SP〕
ATT_MEM({ ATR mematr, void *base, SIZE size })	〔SP〕
ATA_MEM({ ATR mematr, void *base, SIZE size }, { ACPTN acptn1, ACPTN acptn2, ACPTN acptn3, ACPTN acptn4 })	〔SP〕
ATT_PMA({ ATR mematr, void *base, SIZE size, void *paddr })	〔SP〕
ATA_PMA({ ATR mematr, void *base, SIZE size, void *paddr }, { ACPTN acptn1, ACPTN acptn2, ACPTN acptn3, ACPTN acptn4 })	〔SP〕

(9) 割込み管理機能

CFG_INT(INTNO intno, {ATR intatr, PRI intpri})	〔S〕
CRE_ISR(ID isrid, {ATR isratr, intptr_t exinf, INTNO intno, ISR isr, PRI isrpri})	〔SP〕
ATT_ISR({ATR isratr, intptr_t exinf,, INTNO intno, ISR isr, PRI isrpri})	〔S〕
AID_ISR(uint_t noisr)	〔SD〕
SAC_ISR(ID isrid, {ACPTN acptn1, ACPTN acptn2, ACPTN acptn3, ACPTN acptn4})	〔SP〕
DEF_INH(INHNO inhno, {ATR inhatr, INTHDR inthdr})	〔S〕

(10) CPU 例外管理機能

DEF_EXC(EXCNO excno, {ATR excatr, EXCHDR exchdr})	〔S〕

(11) (11) 拡張サービスコール管理機能

DEF_SVC(FN fncd, {ATR svcatr, EXTSVC svcrtn, SIZE stksz})	〔SP〕

(12) システム構成管理機能

DEF_ICS({SIZE istksz, STK_T *istk})	〔S〕
DEF_STK({SIZE stksz, STK_T *stk})	〔S〕
ATT_INI({ATR iniatr, intptr_t exinf, INIRTN inirtn})	〔S〕
ATT_TER({ATR teratr, intptr_t exinf, TERRTN terrtn})	〔S〕

5.3. データ型

5.3.1. TOPPERS 共通データ型

int8_t	符号付き 8 ビット整数（オプション，C99 準拠）
uint8_t	符号無し 8 ビット整数（オプション，C99 準拠）
int16_t	符号付き 16 ビット整数（C99 準拠）
uint16_t	符号無し 16 ビット整数（C99 準拠）
int32_t	符号付き 32 ビット整数（C99 準拠）
uint32_t	符号無し 32 ビット整数（C99 準拠）
int64_t	符号付き 64 ビット整数（オプション，C99 準拠）
uint64_t	符号無し 64 ビット整数（オプション，C99 準拠）
int128_t	符号付き 128 ビット整数（オプション，C99 準拠）
uint128_t	符号無し 128 ビット整数（オプション，C99 準拠）

int_least8_t	8 ビット以上の符号付き整数（C99 準拠）
uint_least8_t	int_least8_t 型と同じサイズの符号無し整数（C99 準拠）
float32_t	IEEE754 準拠の 32 ビット単精度浮動小数点数（オプション）
double64_t	IEEE754 準拠の 64 ビット倍精度浮動小数点数（オプション）

bool_t	真偽値（true または false）
int_t	16 ビット以上の符号付き整数
uint_t	int_t 型と同じサイズの符号無し整数
long_t	32 ビット以上かつ int_t 型以上のサイズの符号付き整数
ulong_t	long_t 型と同じサイズの符号無し整数

intptr_t	ポインタを格納できるサイズの符号付き整数（C99 準拠
uintptr_t	intptr_t 型と同じサイズの符号無し整数（C99 準拠）

FN	機能コード（符号付き整数，int_t に定義）
ER	正常終了（E_OK）またはエラーコード（符号付き整数，int_t に定義）
ID	オブジェクトの ID 番号（符号付き整数，int_t に定義）
ATR	オブジェクト属性（符号無し整数，uint_t に定義）
STAT	オブジェクトの状態（符号無し整数，uint_t に定義）
MODE	サービスコールの動作モード（符号無し整数，uint_t に定義）
PRI	優先度（符号付き整数，int_t に定義）
SIZE	メモリ領域のサイズ（符号無し整数，ポインタを格納できるサイズの符号無し整数型に定義）
TMO	タイムアウト指定（符号付き整数，単位はミリ秒，int_t に定義）
RELTIM	相対時間（符号無し整数，単位はミリ秒，uint_t に定義）
SYSTIM	システム時刻（符号無し整数，単位はミリ秒，ulong_t に定義）
SYSUTM	性能評価用システム時刻（符号無し整数，単位はマイクロ秒，
FP	プログラムの起動番地（型の定まらない関数ポインタ）
ER_BOOL	エラーコードまたは真偽値（符号付き整数，int_t に定義）
ER_ID	エラーコードまたは ID 番号（符号付き整数，int_t に定義，負の ID 番号は格納できない）
ER_UINT	エラーコードまたは符号無し整数（符号付き整数，int_t に定義，符号無し整数を格納する場合の有効ビット数は uint_t より 1 ビット短い
MB_T	オブジェクト管理領域を確保するためのデータ型
ACPTN	アクセス許可パターン（符号無し 32 ビット整数，uint32_t に定義）
ACVCT	アクセス許可ベクタ

```
typedef struct acvct {
    ACPTN   acptn1;     /* 通常操作 1 のアクセス許可パターン */
    ACPTN   acptn2;     /* 通常操作 2 のアクセス許可パターン */
    ACPTN   acptn3;     /* 管理操作のアクセス許可パターン */
    ACPTN   acptn4;     /* 参照操作のアクセス許可パターン */
} ACVCT;
```

5.3.2. カーネルの使用するデータ型

TEXPTN	タスク例外要因のビットパターン（符号無し整数，uint_t に定義）
FLGPTN	イベントフラグのビットパターン（符号無し整数，uint_t に定義）
OVRTIM	プロセッサ時間（符号無し整数，単位はマイクロ秒，ulong_t に定義）
INTNO	割込み番号（符号無し整数，uint_t に定義）
INHNO	割込みハンドラ番号（符号無し整数，uint_t に定義）
EXCNO	CPU 例外ハンドラ番号（符号無し整数，uint_t に定義）
TASK	タスクのメインルーチン（関数ポインタ）
TEXRTN	タスク例外処理ルーチン（関数ポインタ）
CYCHDR	周期ハンドラ（関数ポインタ）
ALMHDR	アラームハンドラ（関数ポインタ）
OVRHDR	オーバランハンドラ（関数ポインタ）
ISR	割込みサービスルーチン（関数ポインタ）
INTHDR	割込みハンドラ（関数ポインタ）
EXCHDR	CPU 例外ハンドラ（関数ポインタ）
EXTSVC	拡張サービスコール（関数ポインタ）
INIRTN	初期化ルーチン（関数ポインタ）
TERRTN	終了処理ルーチン（関数ポインタ）

STK_T	スタック領域を確保するためのデータ型
MPF_T	固定長メモリプール領域を確保するためのデータ型

```
typedef struct t_msg {          /* メールボックスのメッセージヘッダ */
    struct t_msg    *pk_next;
} T_MSG;
```

```
typedef struct t_msg_pri {      /* 優先度付きメッセージヘッダ */
    T_MSG       msgque;         /* メッセージヘッダ */
    PRI         msgpri;         /* メッセージ優先度 */
} T_MSG_PRI;
```

5.3.3. カーネルの使用するパケット形式

(1) タスク管理機能

- タスクの生成情報のパケット形式

```
typedef struct t_ctsk {
    ATR         tskatr;     /* タスク属性 */
    intptr_t    exinf;      /* タスクの拡張情報 */
    TASK        task;       /* タスクのメインルーチンの先頭番地 */
    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 */
    STK_T *     stk;        /* タスクのスタック領域の先頭番地 */
    /* 以下は，保護機能対応カーネルの場合 */
    SIZE        sstksz;     /* タスクのシステムスタック領域のサイズ */
    STK_T *     sstk;       /* タスクのシステムスタック領域の先頭番地 */
} T_CTSK;
```

- タスクの現在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 */
    TMO         lefttmo;    /* タイムアウトするまでの時間 */
    uint_t      actcnt;     /* 起動要求キューイング数 */
    uint_t      wupcnt;     /* 起床要求キューイング数 */
    /* 以下は，保護機能対応カーネルの場合 */
    bool_t      texmsk;     /* タスク例外マスク状態か否か */
    bool_t      waifbd;     /* 待ち禁止状態か否か */
    uint_t      svclevel;   /* 拡張サービスコールのネストレベル */
    /* 以下は，マルチプロセッサ対応カーネルの場合 */
    ID          prcid;      /* 割付けプロセッサの ID */
    ID          actprc      /* 次の起動時の割付けプロセッサの ID */
} T_RTSK;
```

(2) タスク付属同期機能

なし

(3) タスク例外処理機能

- タスク例外処理ルーチンの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 */
    TEXRTN      texrtn;     /* タスク例外処理ルーチンの先頭番地 */
} T_DTEX;
```

- タスク例外処理の現在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 */
} T_RTEX;
```

(4) 同期・通信機能

- セマフォの生成情報のパケット形式

```
typedef struct t_csem {
    ATR         sematr;     /* セマフォ属性 */
    uint_t      isemcnt;    /* セマフォの初期資源数 */
    uint_t      maxsem;     /* セマフォの最大資源数 */
} T_CSEM;
```

- セマフォの現在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_t      semcnt;     /* セマフォの資源数 */
} T_RSEM;
```

- イベントフラグの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;    /* イベントフラグの初期ビットパターン */
} T_CFLG;
```

- イベントフラグの現在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;     /* イベントフラグのビットパターン */
} T_RFLG;
```

- データキューの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_t      dtqcnt;     /* データキュー管理領域に格納できるデータ数 */
    void *      dtqmb;      /* データキュー管理領域の先頭番地 */
} T_CDTQ;
```

- データキューの現在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_t      sdtqcnt;  /* データキュー管理領域に格納されている データの数 */
} T_RDTQ;
```

- 優先度データキューの生成情報のパケット形式

```
typedef struct t_cpdq {
    ATR         pdqatr;     /* 優先度データキュー属性 */
    uint_t      pdqcnt;     /* 優先度データキュー管理領域に格納できるデータ数 */
    PRI         maxdpri;    /* 優先度データキューに送信できるデータ優先度の最大値 */
    void *      pdqmb;      /* 優先度データキュー管理領域の先頭番地 */
} T_CPDQ;
```

- 優先度データキューの現在状態のパケット形式

```
typedef struct t_rpdq {
    ID          stskid;     /* 優先度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;     /* 優先度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_t      spdqcnt;    /* 優先度データキュー管理領域に格納されているデータの数 */
} T_RPDQ;
```

- メールボックスの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;   /* 優先度メールボックスに送信できるメッセージ優先度の最大値 */
    void *      mprihd;     /* 優先度別のメッセージキューヘッダ領域の先頭番地 */
} T_CMBX;
```

- メールボックスの現在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につながれたメッセージの先頭番地 */
} T_RMBX;
```

- ミューテックスの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 */
} T_CMTX;
```

- ミューテックスの現在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 */
} T_RMTX;
```

- メッセージバッファの生成情報のパケット形式

☆未完成

- メッセージバッファの現在状態のパケット形式

☆未完成

- スピンロックの生成情報のパケット形式

```
typedef struct t_cspn {
    ATR         spnatr;     /* スピンロック属性 */
} T_CSPN;
```

- スピンロックの現在状態のパケット形式

```
typedef struct t_rspn {
    STAT        spnstat     /* スピンロックのロック状態 */
} T_RSPN;
```

(5)　メモリプール管理機能

- 固定長メモリプールの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_t      blkcnt;     /* 獲得できる固定長メモリブロックの数 */
    uint_t      blksz;      /* 固定長メモリブロックのサイズ */
    MPF_T *     mpf;        /* 固定長メモリプール領域の先頭番地 */
    void *      mpfmb;      /* 固定長メモリプール管理領域の先頭番地 */
} T_CMPF;
```

- 固定長メモリプールの現在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_t      fblkcnt;    /* 固定長メモリプール領域の空きメモリ領域に
                               割り付けることができる固定長メモリブロックの数 */
} T_RMPF;
```

(6)　時間管理機能

- 周期ハンドラの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 */
    intptr_t    exinf;      /* 周期ハンドラの拡張情報 */
    CYCHDR      cychdr;     /* 周期ハンドラの先頭番地 */
    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 */
} T_CCYC;
```

- 周期ハンドラの現在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;    /* 次に周期ハンドラを起動する時刻までの相対時間 */
    /* 以下は，マルチプロセッサ対応カーネルの場合 */
    ID          prcid;      /* 割付けプロセッサの ID */
} T_RCYC;
```

- アラームハンドラの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 */
    intptr_t    exinf;      /* アラームハンドラの拡張情報 */
    ALMHDR      almhdr;     /* アラームハンドラの先頭番地 */
} T_CALM;
```

- アラームハンドラの現在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;    /* アラームハンドラを起動する時刻までの相対時間 */
    /* 以下は，マルチプロセッサ対応カーネルの場合 */
    ID          prcid;      /* 割付けプロセッサの ID */
} T_RALM;
```

- オーバランハンドラの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 */
    OVRHDR      ovrhdr;     /* オーバランハンドラの先頭番地 */
} T_DOVR;
```

- オーバランハンドラの現在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;        /* 残りプロセッサ時間 */
} T_ROVR;
```

(7)　システム状態管理機能

- システムの現在状態のパケット形式

☆未完成

(8)　メモリオブジェクト管理機能

- メモリオブジェクトの登録情報のパケット形式

```
typedef struct t_amem {
    ATR         mematr      /* メモリオブジェクト属性 */
    void *      base        /* 登録するメモリ領域の先頭番地 */
    SIZE        size        /* 登録するメモリ領域のサイズ（バイト数）*/
} T_AMEM;
```

- 物理メモリ領域の登録情報のパケット形式

```
typedef struct t_apma {
    ATR         mematr      /* メモリオブジェクト属性 */
    void *      base        /* 登録するメモリ領域の先頭番地 */
    SIZE        size        /* 登録するメモリ領域のサイズ（バイト数）*/
    void *      paddr       /* 登録するメモリ領域の物理アドレスの先頭番地 */
} T_APMA;
```

- メモリオブジェクトの現在状態のパケット形式

☆未完成

(9)　割込み管理機能

- 割込み要求ラインの属性の設定情報のパケット形式

```
typedef struct t_cint {
    ATR         intatr;     /* 割込み要求ライン属性 */
    PRI         intpri;     /* 割込み優先度 */
} T_CINT;
```

- 割込みサービスルーチンの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 */
    intptr_t    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登録する割込み番号 */
    ISR         isr;        /* 割込みサービスルーチンの先頭番地 */
    PRI         isrpri;     /* 割込みサービスルーチン優先度 */
} T_CISR;
```

- 割込みサービスルーチンの現在状態のパケット形式

☆未完成

- 割込みハンドラの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 */
    INTHDR      inthdr;     /* 割込みハンドラの先頭番地 */
} T_DINH;
```

- 割込み要求ラインの現在状態のパケット形式

☆未完成

(10) CPU 例外管理機能

- CPU 例外ハンドラの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 */
    EXCHDR      exchdr;     /* CPU 例外ハンドラの先頭番地 */
} T_DEXC;
```

(11) 拡張サービスコール管理機能

- 拡張サービスコールの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 */
    EXTSVC      svcrtn      /* 拡張サービスコールの先頭番地 */
    SIZE        stksz       /* 拡張サービスコールで使用するスタックサイズ */
} T_DSVC;
```

(12) システム構成管理機能

- コンフィギュレーション情報のパケット形式

☆未完成

- バージョン情報のパケット形式

☆未完成

5.4. 定数とマクロ

5.4.1. TOPPERS 共通定数

(1) 一般定数

NULL	-	無効ポインタ
true	1	真
false	0	偽
E_OK	0	正常終了

(2) 整数型に格納できる最大値と最小値

INT8_MAX	int8_t に格納できる最大値（オプション，C99 準拠）
INT8_MIN	int8_t に格納できる最小値（オプション，C99 準拠）
UINT8_MAX	uint8_t に格納できる最大値（オプション，C99 準拠）
INT16_MAX	int16_t に格納できる最大値（C99 準拠）
INT16_MIN	int16_t に格納できる最小値（C99 準拠）
UINT16_MAX	uint16_t に格納できる最大値（C99 準拠）
INT32_MAX	int32_t に格納できる最大値（C99 準拠）
INT32_MIN	int32_t に格納できる最小値（C99 準拠）
UINT32_MAX	uint32_t に格納できる最大値（C99 準拠）
INT64_MAX	int64_t に格納できる最大値（オプション，C99 準拠）
INT64_MIN	int64_t に格納できる最小値（オプション，C99 準拠）
UINT64_MAX	uint64_t に格納できる最大値（オプション，C99 準拠）
INT128_MAX	int128_t に格納できる最大値（オプション，C99 準拠）
INT128_MIN	int128_t に格納できる最小値（オプション，C99 準拠）
UINT128_MAX	uint128_t に格納できる最大値（オプション，C99 準拠）

INT_LEAST8_MAX	int_least8_t に格納できる最大値（C99 準拠）
INT_LEAST8_MIN	int_least8_t に格納できる最小値（C99 準拠）
UINT_LEAST8_MAX	uint_least8_t に格納できる最大値（C99 準拠）
INT_MAX	int_t に格納できる最大値（C90 準拠）
INT_MIN	int_t に格納できる最小値（C90 準拠）
UINT_MAX	uint_t に格納できる最大値（C90 準拠）
LONG_MAX	long_t に格納できる最大値（C90 準拠）
LONG_MIN	long_t に格納できる最小値（C90 準拠）
ULONG_MAX	ulong_t に格納できる最大値（C90 準拠）

FLOAT32_MIN	float32_t に格納できる最小の正規化された正の浮動小数点数（オプション）
FLOAT32_MAX	float32_t に格納できる表現可能な最大の有限浮動小数点数（オプション）
DOUBLE64_MIN	double64_t に格納できる最小の正規化された正の浮動小数点数（オプション）
DOUBLE64_MAX	double64_t に格納できる表現可能な最大の有限浮動小数点数（オプション）

(3) 整数型のビット数

TA_NULL	0U	オブジェクト属性を指定しない

(4) オブジェクト属性

TA_NULL	0U	オブジェクト属性を指定しない

(5) 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

(6) アクセス許可パターン

TACP_KERNEL	0U	カーネルドメインのみにアクセスを許可
TACP_SHARED	~0U	べての保護ドメインにアクセスを許可

5.4.2. TOPPERS 共通マクロ

(1) 整数定数を作るマクロ

INT8_C(val)	int_least8_t 型の定数を作るマクロ（C99 準拠）
UINT8_C(val)	uint_least8_t 型の定数を作るマクロ（C99 準拠）
INT16_C(val)	int16_t 型の定数を作るマクロ（C99 準拠）
UINT16_C(val)	uint16_t 型の定数を作るマクロ（C99 準拠）
INT32_C(val)	int32_t 型の定数を作るマクロ（C99 準拠）
UINT32_C(val)	uint32_t 型の定数を作るマクロ（C99 準拠）
INT64_C(val)	int64_t 型の定数を作るマクロ（オプション，C99 準拠）
UINT64_C(val)	uint64_t 型の定数を作るマクロ（オプション，C99 準拠）
INT128_C(val)	int128_t 型の定数を作るマクロ（オプション，C99 準拠）
UINT128_C(val)	uint128_t 型の定数を作るマクロ（オプション，C99 準拠）

UINT_C(val)	uint_t 型の定数を作るマクロ
ULONG_C(val)	ulong_t 型の定数を作るマクロ

(2) 型に関する情報を取り出すためのマクロ

offsetof(structure, field)	構造体 structure 中のフィールド field のバイト位置を返すマクロ（C90 準拠）
alignof(type)	型 type のアラインメント単位を返すマクロ
ALIGN_TYPE(addr, type)	番地 addr が型 type に対してアラインしているかどうかを返すマクロ

(3) assert マクロ

assert(exp)	exp が成立しているかを検査するマクロ（C90 準拠）

(4) コンパイラの拡張機能のためのマクロ

inline	インライン関数
Inline	ファイルローカルなインライン関数
asm	インラインアセンブラ
Asm	インラインアセンブラ（最適化抑止）
throw()	例外を発生しない関数
NoReturn	リターンしない関数

(5) エラーコード生成・分解マクロ

ERCD(mercd, sercd)	メインエラーコード mercd とサブエラーコード sercd から，エラーコードを構成するためのマクロ
MERCD(ercd)	エラーコード ercd からメインエラーコードを抽出するためのマクロ
SERCD(ercd)	エラーコード ercd からサブエラーコードを抽出するためのマクロ

(6) アクセス許可パターン生成マクロ

TACP(domid)	domid で指定されるユーザドメインのみにアクセスを許可するアクセス許可パターンを構成するためのマクロ

5.4.3. カーネル共通定数

(1) オブジェクト属性

TA_TPRI	0x01U	タスクの待ち行列をタスクの優先度順に

(2)　保護ドメイン ID

TDOM_SELF	0	自タスクの属する保護ドメイン
TDOM_KERNEL	-1	カーネルドメイン
TDOM_NONE	-2	無所属（保護ドメインに属さない）

(3)　その他のカーネル共通定数

TCLS_SELF	0	自タスクの属するクラス

TPRC_NONE	0	割付けプロセッサの指定がない
TPRC_INI	0	初期割付けプロセッサ

TSK_SELF	0	自タスク指定
TSK_NONE	0	該当するタスクがない

TPRC_NONE	0	割付けプロセッサの指定がない
TPRC_INI	0	初期割付けプロセッサ

TIPM_ENAALL	0	割込み優先度マスク全解除

5.4.4.　カーネル共通マクロ

(1)　オブジェクト属性を作るマクロ

TA_DOM(domid)	domid で指定される保護ドメインに属する
TA_CLS(clsid)	svc で指定されるサービスコールを関数呼出しによって呼び出すための名称

(2)　サービスコールの呼出し方法を指定するマクロ

SVC_CALL(svc)	svc で指定されるサービスコールを関数呼出しによって呼び出すための名称

5.4.5.　カーネルの機能毎の定数

(1)　タスク管理機能

TA_ACT	0x02U	タスクの生成時にタスクを起動する
TA_RSTR	0x04U	生成するタスクを制約タスクとする
TA_FPU	－	FPU レジスタをコンテキストに含める

TTS_RUN	0x01U	実行状態
TTS_RDY	0x02U	実行可能状態
TTS_WAI	0x04U	待ち状態
TTS_SUS	0x08U	強制待ち状態
TTS_WAS	0x0cU	二重待ち状態
TTS_DMT	0x10U	休止状態
TTS_RUN	0x01U	実行状態

TTW_SLP	0x0001U	起床待ち
TTW_DLY	0x0002U	時間経過待ち
TTW_SEM	0x0004U	セマフォの資源獲得待ち
TTW_FLG	0x0008U	イベントフラグ待ち
TTW_SDTQ	0x0010U	データキューへの送信待ち
TTW_RDTQ	0x0020U	データキューからの受信待ち
TTW_SPDQ	0x0100U	優先度データキューへの送信待ち
TTW_RPDQ	0x0200U	優先度データキューからの受信待ち
TTW_MBX	0x0040U	メールボックスからの受信待ち
TTW_MTX	0x0080U	ミューテックスのロック待ち状態
TTW_MPF	0x2000U	固定長メモリブロックの獲得待ち

TA_FPU の値は，ターゲット定義とする．

(2) タスク例外処理機能

TTEX_ENA	0x01U	タスク例外処理許可状態
TTEX_DIS	0x02U	タスク例外処理禁止状態

(3) 同期・通信機能

- イベントフラグ

TA_WMUL	0x02U	複数のタスクが待つのを許す
TA_CLR	0x04U	タスクの待ち解除時にイベントフラグをクリアする

TWF_ORW	0x01U	イベントフラグの OR 待ちモード
TWF_ANDW	0x02U	イベントフラグの AND 待ちモード

- メールボックス

TA_MPRI	0x02U	メッセージキューをメッセージの優先度順にする

- スピンロック

TSPN_UNL	0x01U	取得されていない状態
TSPN_LOC	0x02U	取得されている状態

(4) 時間管理機能

- 周期ハンドラ

TA_STA	0x02U	待ち行列をタスクの優先度順にする
TA_PHS	0x04U	周期ハンドラを生成した時刻を基準時刻とする

TCYC_STP	0x01U	周期ハンドラが動作していない状態
TCYC_STA	0x02U	周期ハンドラが動作している状態

- アラームハンドラ

TALM_STP	0x01U	アラームハンドラが動作していない状態
TALM_STA	0x02U	アラームハンドラが動作している状態

- オーバランハンドラ

TOVR_STP	0x01U	オーバランハンドラが動作していない状態
TOVR_STA	0x02U	オーバランハンドラが動作している状態

(5) メモリオブジェクト管理機能

TA_NOWRITE	0x01U	書込みアクセス禁止
TA_NOREAD	0x02U	読出しアクセス禁止
TA_EXEC	0x04U	実行アクセス許可
TA_MEMINI	0x08U	メモリの初期化を行う
TA_MEMPRSV	0x10U	メモリの初期化を行わない
TA_SDATA	0x20U	ショートデータ領域に配置
TA_UNCACHE	0x40U	キャッシュ禁止
TA_IODEV	0x80U	周辺デバイスの領域
TA_WTHROUGH	－	ライトスルーキャッシュを用いる

TA_STDROM	0x02U	標準 ROM リージョン
TA_STDRAM	0x04U	標準 RAM リージョン

TPM_WRITE	0x01U	書込みアクセス権のチェック
TPM_READ	0x02U	読出しアクセス権のチェック
TPM_EXEC	0x04U	実行アクセス権のチェック

TA_WTHROUGH の値は，ターゲット定義とする．

(6) 割込み管理機能

TA_ENAINT	0x01U	割込み要求禁止フラグをクリア
TA_EDGE	0x02U	エッジトリガ
TA_POSEDGE	ー	ポジティブエッジトリガ
TA_NEGEDGE	ー	ネガティブエッジトリガ
TA_BOTHEDGE	ー	すべてのプロセッサで割込みを処理（マルチプロセッサ対応カーネルの場合）
TA_LOWLEVEL	ー	ローレベルトリガ
TA_HIGHLEVEL	ー	ハイレベルトリガ

TA_NONKERNEL	0x02U	カーネル管理外の割込み

TA_POSEDGE，TA_NEGEDGE，TA_BOTHEDGE，TA_LOWLEVEL，TA_HIGHLEVEL の値は，ターゲット定義とする．

(7) CPU 例外管理機能

TA_DIRECT	ー	CPU 例外ハンドラを直接呼び出す

TA_DIRECT の値は，ターゲット定義とする．

5.4.6. カーネルの機能毎のマクロ

(1) タスク管理機能

COUNT_STK_T(sz)	サイズ sz のスタック領域を確保するために必要な STK_T 型の配列の要素数
ROUND_STK_T(sz)	要素数 COUNT_STK_T(sz)の STK_T 型の配列のサイズ（sz を，STK_T 型のサイズの倍数になるように大きい方に丸めた値）

(2)　同期・通信機能

TSZ_DTQMB(dtqcnt)	dtqcnt で指定した数のデータを格納できるデータキュー管理領域のサイズ（バイト数）
TCNT_DTQMB(dtqcnt)	dtqcnt で指定した数のデータを格納できるデータキュー管理領域を確保するために必要な MB_T 型の配列の要素数

TSZ_PDQMB(pdqcnt)	pdqcnt で指定した数のデータを格納できる優先度データキュー管理領域のサイズ（バイト数）
TCNT_PDQMB(pdqcnt)	pdqcnt で指定した数のデータを格納できる優先度データキュー管理領域を確保するために必要な MB_T 型の配列の要素数

(3)　メモリプール管理機能

COUNT_MPF_T(blksz)	固定長メモリブロックのサイズが blksz の固定長メモリプール領域を確保するために，固定長メモリブロック 1 つあたりに必要な MPF_T 型の配列の要素数
ROUND_MPF_T(blksz)	要素数 COUNT_MPF_T(blksz)の MPF_T 型の配列のサイズ(blksz を，MPF_T 型のサイズの倍数になるように大きい方に丸めた値)

TSZ_MPFMB(blkcnt)	blkcnt で指定した数の固定長メモリブロックを管理することができる固定長メモリプール管理領域のサイズ（バイト数）
TCNT_MPFMB(blkcnt)	blkcnt で指定した数の固定長メモリブロックを管理することができる固定長メモリプール管理領域を確保するために必要な MB_T 型の配列の要素数

5.5. 構成マクロ

5.5.1. TOPPERS 共通構成マクロ

(1)　相対時間の範囲

TMAX_RELTIM	相対時間に指定できる最大値

5.5.2. カーネル共通構成マクロ

(1)　サポートする機能

TOPPERS_SUPPORT_PROTECT	保護機能対応のカーネル
TOPPERS_SUPPORT_MULTI_PRC	マルチプロセッサ対応のカーネル
TOPPERS_SUPPORT_DYNAMIC_CRE	動的生成対応のカーネル

(2)　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

(3)　プロセッサの数

TNUM_PRCID	プロセッサの数

(4)　特殊な役割を持ったプロセッサ

TOPPERS_MASTER_PRCID	マスタプロセッサの ID 番号
TOPPERS_SYSTIM_PRCID	システム時刻管理プロセッサの ID 番号（グローバルタイマ方式の場合のみ）

(5)　タイマ方式

TOPPERS_SYSTIM_LOCAL	ローカルタイマ方式の場合にマクロ定義
TOPPERS_SYSTIM_GLOBAL	グローバルタイマ方式の場合にマクロ定義

(6)　バージョン情報

TKERNEL_MAKER	カーネルのメーカコード（＝0x0118）
TKERNEL_PRID	カーネルの識別番号
TKERNEL_SPVER	カーネル仕様のバージョン番号
TKERNEL_PRVER	カーネルのバージョン番号

5.5.3.　カーネルの機能毎の構成マクロ

(1)　タスク管理機能

TMAX_ACTCNT	タスクの起動要求キューイング数の最大値
TNUM_TSKID	登録できるタスクの数（動的生成対応でないカーネルでは，静的 API によって登録されたタスクの数に一致）

(2)　タスク付属同期機能

TMAX_WUPCNT	タスクの起床要求キューイング数の最大値

(3)　タスク例外処理機能

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

(4)　同期・通信機能

- セマフォ

TMAX_ACTCNT	TMAX_MAXSEM
TNUM_SEMID	登録できるセマフォの数（動的生成対応でないカーネルでは，静的 API によって登録されたセマフォの数に一致）

- イベントフラグ

TBIT_FLGPTN	イベントフラグのビット数（FLGPTN の有効ビット数）
TNUM_FLGID	登録できるイベントフラグの数（動的生成対応でないカーネルでは，静的 API によって登録されたイベントフラグの数に一致）

- データキュー

TNUM_DTQID	登録できるデータキューの数（動的生成対応でないカーネルでは，静的 API によって登録されたデータキューの数に一致）

- 優先度データキュー

TMIN_DPRI	データ優先度の最小値（＝1）
TMAX_DPRI	データ優先度の最大値
TNUM_PDQID	登録できる優先度データキューの数（動的生成対応でないカーネルでは，静的 API によって登録された優先度データキューの数に一致）

- メールボックス

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
TNUM_MBXID	登録できるメールボックスの数（動的生成対応でないカーネルでは，静的 API によって登録されたメールボックスの数に一致）

- ミューテックス

TNUM_MTXID	登録できるミューテックスの数（動的生成対応でないカーネルでは，静的 API によって登録されたミューテックスの数に一致）

- スピンロック

TNUM_SPNID	登録できるスピンロックの数（動的生成対応でないカーネルでは，静的 API によって登録されたスピンロックの数に一致）

(5)　メモリプール管理機能

- 固定長メモリプール

TNUM_MPFID	登録できる固定長メモリプールの数（動的生成対応でないカーネルでは，静的 API によって登録された固定長メモリプールの数に一致）

(6)　時間管理機能

- システム時刻管理

TIC_NUME	タイムティックの周期（単位はミリ秒）の分子
TIC_DENO	タイムティックの周期（単位はミリ秒）の分母

TOPPERS_SUPPORT_GET_UTM	get_utm がサポートされている

- 周期ハンドラ

TNUM_CYCID	登録できる周期ハンドラの数（動的生成対応でないカーネルでは，静的 API によって登録された周期ハンドラの数に一致）

- アラームハンドラ

TNUM_ ALMID	登録できるアラームハンドラの数（動的生成対応でないカーネルでは，静的 API によって登録された周期ハンドラの数に一致）

- オーバランハンドラ

TMAX_OVRTIM	プロセッサ時間に指定できる最大値

TOPPERS_SUPPORT_OVRHDR	オーバランハンドラ機能がサポートされている

(7)　システム状態管理機能

なし

(8)　メモリオブジェクト管理機能

TOPPERS_SUPPORT_ATT_MOD	ATT_MOD／ATA_MOD がサポートされている
TOPPERS_SUPPORT_ATT_PMA	ATT_PMA／ATA_PMA／att_pma がサポートされている

(9)　割込み管理機能

TMIN_INTPRI	ー	割込み優先度の最小値（最高値）
TMAX_INTPRI	ー	割込み優先度の最大値（最低値，＝-1）

TMIN_ISRPRI	―	割込みサービスルーチン優先度の最小値（＝1）
TMAX_ISRPRI	―	割込みサービスルーチン優先度の最大値

TOPPERS_SUPPORT_DIS_INT	―	dis_int がサポートされている
TOPPERS_SUPPORT_ENA_INT	―	ena_int がサポートされている

(10)　(10) CPU 例外管理機能

なし

(11)　拡張サービスコール管理機能

TNUM_FNCD	登録できる拡張サービスコールの数（動的生成対応でないカーネルでは，静的 API によって登録された拡張サービスコールの数に一致）

(12)　システム構成管理機能

なし

5.6. エラーコード一覧

(1)　メインエラーコード

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_NORES	-35	資源不足
E_OBJ	-41	オブジェクト状態エラー
E_NOEXS	-42	オブジェクト未登録
E_QOVR	-43	キューイングオーバフロー
E_RLWAI	-49	待ち禁止状態または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または再初期化
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

5.7. 機能コード一覧

	0	-1	-2	-3		0	-1	-2	-3
-0x01	予約	予約	予約	予約	-0x91	xsns_dpn	xsns_xpn	予約	予約
-0x05	act_tsk	iact_tsk	can_act	ext_tsk	-0x95	ref_cfg	ref_ver	予約	予約
-0x09	ter_tsk	chg_pri	get_pri	get_inf	-0x99	予約	予約	予約	予約
-0x0d	slp_tsk	tslp_tsk	wup_tsk	iwup_tsk	-0x9d	予約	予約	予約	予約
-0x11	can_wup	rel_wai	irel_wai	予約	-0xa1	予約	ini_sem	ini_flg	ini_dtq
-0x15	dis_wai	idis_wai	ena_wai	iena_wai	-0xa5	ini_pdq	ini_mbx	ini_mtx	ini_mbf
-0x19	sus_tsk	rsm_tsk	dly_tsk	予約	-0xa9	ini_mpf	予約	予約	予約
-0x1d	ras_tex	iras_tex	dis_tex	ena_tex	-0xad	予約	予約	予約	予約
-0x21	sns_tex	ref_tex	予約	予約	-0xb1	ref_tsk	ref_sem	ref_flg	ref_dtq
-0x25	sig_sem	isig_sem	wai_sem	pol_sem	-0xb5	ref_pdq	ref_mbx	ref_mtx	ref_mbf
-0x29	twai_sem	予約	予約	予約	-0xb9	ref_mpf	ref_cyc	ref_alm	ref_isr
-0x2d	set_flg	iset_flg	clr_flg	wai_flg	-0xbd	ref_spn	予約	予約	予約
-0x31	pol_flg	twai_flg	予約	予約	-0xc1	acre_tsk	acre_sem	acre_flg	acre_dtq
-0x35	snd_dtq	psnd_dtq	ipsnd_dt	tsnd_dtq	-0xc5	acre_pdq	acre_mbx	acre_mtx	acre_mbf
-0x39	fsnd_dtq	ifsnd_dtq	rcv_dtq	prcv_dtq	-0xc9	acre_mpf	acre_cyc	acre_alm	acre_isr
-0x3d	trcv_dtq	予約	予約	予約	-0xcd	acre_spn	予約	予約	予約
-0x41	snd_pdq	psnd_pdq	ipsnd_pd	tsnd_pdq	-0xd1	del_tsk	del_sem	del_flg	del_dtq
-0x45	rcv_pdq	prcv_pdq	trcv_pdq	予約	-0xd5	del_pdq	del_mbx	del_mtx	del_mbf
-0x49	snd_mbx	rcv_mbx	prcv_mbx	trcv_mbx	-0xd9	del_mpf	del_cyc	del_alm	del_isr
-0x4d	loc_mtx	ploc_mtx	tloc_mtx	unl_mtx	-0xdd	del_spn	予約	予約	予約
-0x51	snd_mbf	psnd_mbf	tsnd_mbf	rcv_mbf	-0xe1	sac_tsk	sac_sem	sac_flg	sac_dtq
-0x55	prcv_mbf	trcv_mbf	予約	予約	-0xe5	sac_pdq	予約	sac_mtx	sac_mbf
-0x59	get_mpf	pget_mpf	tget_mpf	rel_mpf	-0xe9	sac_mpf	sac_cyc	sac_alm	sac_isr
-0x5d	get_tim	get_utm	予約	ref_ovr	-0xed	sac_spn	予約	予約	予約
-0x61	sta_cyc	stp_cyc	予約	予約	-0xf1	def_tex	def_ovr	def_inh	def_exc
-0x65	sta_alm	ista_alm	stp_alm	istp_alm	-0xf5	def_svc	予約	予約	予約
-0x69	sta_ovr	ista_ovr	stp_ovr	istp_ovr	-0xf9	予約	予約	予約	予約
-0x6d	sac_sys	ref_sys	rot_rdq	irot_rdq	-0xfd	予約	予約	予約	予約
-0x71	get_did	予約	get_tid	iget_tid	-0x101	mact_tsk	imact_tsk	mig_tsk	予約
-0x75	loc_cpu	iloc_cpu	unl_cpu	iunl_cpu	-0x105	msta_cyc	予約	msta_alm	imsta_alm
-0x79	dis_dsp	ena_dsp	sns_ctx	sns_loc	-0x109	mrot_rdq	imrot_rdq	get_pid	iget_pid
-0x7d	sns_dsp	sns_dpn	sns_ker	ext_ker	-0x10d	予約	予約	予約	予約
-0x81	att_mem	det_mem	sac_mem	prb_mem	-0x111	loc_spn	iloc_spn	try_spn	itry_spn
-0x85	ref_mem	予約	att_pma	予約	-0x115	unl_spn	iunl_spn	予約	予約

【μITRON4.0 仕様との関係】

サービスコールの機能コードを割り当てなおした.

5.8. カーネルオブジェクトに対するアクセスの種別

オブジェクトの種類	通常操作1	通常操作2	管理操作	参照操作
メモリオブジェクト	書込み	読出し	det_mem	ref_mem
		実行	sac_mem	prb_mem
タスク	act_tsk	ter_tsk	del_tsk	get_pri
	mact_tsk	chg_pri	sac_tsk	ref_tsk
	can_act	rel_wai	def_tex	ref_tex
	mig_tsk	sus_tsk		ref_ovr
	wup_tsk	rsm_tsk		
	can_wup	dis_wai		
		ena_wai		
		ras_tex		
		sta_ovr		
		stp_ovr		
セマフォ	sig_sem	wai_sem	del_sem	ref_sem
		pol_sem	ini_sem	
		twai_sem	sac_sem	
イベントフラグ	set_flg	wai_flg	del_flg	ref_flg
	clr_flg	pol_flg	ini_flg	
		twai_flg	sac_flg	
データキュー	snd_dtq	rcv_dtq	del_dtq	ref_dtq
	psnd_dtq	prcv_dtq	ini_dtq	
	tsnd_dtq	trcv_dtq	sac_dtq	
	fsnd_dtq			
優先度データキュー	snd_pdq	rcv_pdq	del_pdq	ref_pdq
	psnd_pdq	prcv_pdq	ini_pdq	
	tsnd_pdq	trcv_pdq	sac_pdq	
ミューテックス	loc_mtx	－	del_mtx	ref_mtx
	ploc_mtx		ini_mtx	
	tloc_mtx		sac_mtx	
スピンロック	loc_spn	－	del_spn	ref_spn
	try_spn		sac_spn	
	unl_spn			
固定長メモリプール	get_mpf	rel_mpf	del_mpf	ref_mpf
	pget_mpf		ini_mpf	
	tget_mpf		sac_mpf	
周期ハンドラ	sta_cyc	stp_cyc	del_cyc	ref_cyc
	msta_cyc		sac_cyc	
アラームハンドラ	sta_alm	stp_alm	del_alm	ref_alm
	msta_alm		sac_alm	
割込みサービスルーチン	－	－	del_isr	ref_isr
			sac_isr	
システム状態	rot_rdq	loc_cpu	acre_yyy	get_tim
	mrot_rdq	unl_cpu	att_mem	get_ipm
	dis_dsp	dis_int	att_pma	ref_sys
	ena_dsp	ena_int	cfg_int	ref_int
		chg_ipm	def_inh	ref_cfg
			def_exc	ref_ver
			def_svc	
			def_ovr	

すべての保護ドメインから呼び出すことができるサービスコール：

- 自タスクへの操作（ext_tsk，get_inf，slp_tsk，tslp_tsk，dly_tsk，dis_tex，ena_tex）
- タスク例外状態参照（sns_tex）
- 性能評価用システム時刻の参照（get_utm）
- システム状態参照（get_tid，get_did，get_pid，sns_ctx，sns_loc，sns_dsp，sns_dpn，sns_ker）
- CPU 例外発生時の状態参照（xsns_dpn，xsns_xpn）
- 拡張サービスコールの呼出し（cal_svc）

カーネルドメインのみから呼び出すことができるサービスコール：

- システム状態のアクセス許可ベクタの設定（sac_sys）
- カーネルの終了（ext_ker）
- 非タスクコンテキスト専用のサービスコール

アクセス許可ベクタによるアクセス保護を行わないサービスコール：

- ミューテックスのロック解除（unl_mtx）

【補足説明】

xsns_dpn と xsns_xpn は，エラーコードを返さないために，すべての保護ドメインから呼び出すことができるサービスコールとしているが，タスクコンテキストから呼び出した場合には必ず true が返ることとしており，実質的にはカーネルドメインのみから呼び出すことができる．

unl_mtx は，アクセス許可ベクタによるアクセス保護を行わないサービスコールとしているが，ミューテックスをロックしたタスク以外が呼び出すと E_ILUSE エラーとなるため，実質的には対象ミューテックスの通常操作 1 としてアクセス保護されているとみなすことができる（ミューテックスのロック中にアクセス許可ベクタを変更した場合の振舞いは異なる）．

【μITRON4.0/PX 仕様との関係】

get_pri は，μITRON4.0/PX 仕様ではタスクに対する通常操作 1 としていたのを，タスクに対する参照操作に変更した．また，get_ipm（μITRON4.0/PX 仕様では get_ixx）をシステム状態に対する通常操作 2 から参照操作に，sac_sys をシステム状態に対する管理操作からカーネルドメインのみから呼び出すことができるサービスコールに変更した．システム時刻に対するアクセス許可ベクタは廃止し，get_tim はシステム状態に対する参照操作とした．

5.9. 省略名の元になった英語

5.9.1. サービスコールと静的 API の名称の中の xxx の元になった英語

xxx	元になった英語	xxx	元になった英語
act	activate	pol	poll
aid	automatically assigned ID	prb	probe
ata	attach with access control vector	ras	raise
att	attach	rcv	receive
cal	call	ref	reference
can	cancel	rel	release
cfg	configure	rot	rotate
chg	change	rsm	resume
clr	clear	sac	set access control vector
cre	create	set	set
def	define	sig	signal
del	delete	slp	sleep
det	detach	snd	send
dis	disable	sns	sense
dly	delay	sta	start
ena	enable	stp	stop
epr	execution priority	sus	suspend
ext	exit	ter	terminate
get	get	try	try
ini	initialize	unl	unlock
lnk	link	wai	wait
loc	lock	wup	wake up

5.9.2. サービスコールと静的 API の名称の中の yyy の元になった英語

yyy	元になった英語	yyy	元になった英語
act	activation	mod	module
alm	alarm handler	mtx	mutex
cfg	configuration	ovr	overrun handler
cpu	CPU	pdq	priority data queue
ctx	context	pid	processor ID
cyc	cyclic handler	pma	physical memory area
did	domain ID	pri	priority
dpn	dispatch pending	rdq	ready queue
dsp	dispatch	reg	region
dtq	data queue	sec	section
exc	exception	sem	semaphore
flg	eventflag	spn	spin lock
ics	interrupt context stack	stk	stack
inf	information	sys	system
inh	interrupt handler	svc	service call
ini	initilization	ter	termination
int	interrupt	tex	task exception
ipm	interrupt priority mask	tid	task ID
isr	interrupt service routine	tim	time
ker	kernel	tsk	task
loc	lock	utm	time in micro second
mbf	message buffer	ver	version
mbx	mailbox	wai	wait
mpf	fixed-sized memory pool	wup	wake up
mem	memory	xpn	exception pending

5.9.3. サービスコールの名称の中の z の元になった英語

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

5.10. バージョン履歴

2008 年 11 月 19 日	Release 1.0.0	最初のリリース
2009 年 5 月 8 日	Release 1.1.0	FMP カーネルに関する記述が完成
2010 年 5 月 10 日	Release 1.2.0	－
2011 年 5 月 5 日	Release 1.3.0	HRP2 カーネルに関する記述が完成
2012 年 5 月 16 日	Release 1.4.0	SSP カーネルに関する記述が完成
2008 年 11 月 19 日	Release 1.0.0	最初のリリース

以上