μITRON4.0仕様 保護機能拡張
（μITRON4.0/PX仕様）
Ver. 1.00.00

(社)トロン協会 バージョンアップWG
高田 広章 編

μITRON4.0仕様 保護機能拡張（Ver. 1.00.00）

本仕様ならびに本仕様書に関する問い合わせ先は，下記の通りです．

(社)トロン協会 バージョンアップWG

〒108-0073 東京都港区三田1丁目3番39号 勝田ビル5階

TEL: 03-3454-3191　FAX: 03-3454-3224

---

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

§ ITRONは“Industrial TRON”の略称です．

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

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

まえがき

最近，組込みシステム分野においてもソフトウェアの大規模化・複雑化が著しい．それに加えてソフトウェアの開発期間短縮の要求も強い．その結果，ソフトウェアの品質や信頼性を確保することが難しくなっている．

トロン協会 バージョンアップWGでは，こういった問題を軽減もしくは解決するための１つのアプローチとして，μITRON 仕様に保護機能を追加するための検討を，2001年の始め頃より約1年半に渡って行ってきた．その検討の成果を，μITRON4.0仕様に対する拡張仕様の形でまとめたのが，本仕様ならびに仕様書である．本仕様が，組込みソフトウェア開発がかかえる問題の軽減・解決の一助となることを期待している．

また，仕様の検討と並行して，仕様案に準拠したカーネルの実装を，情報処理振興事業協会（IPA）による「情報技術開発支援事業」の採択テーマの1つとして実施した．ここで実装したカーネル（IIMP カーネル，IIMP は An Implementation of ITRON with Memory Protection の略）は，ITRON のウェブサイトからフリーソフトウェアとして配布する予定であるが，本仕様のリファレンス実装として役立つと同時に，そのまま実用化もすることも可能なものである．仕様の公開と同時にそのリファレンス実装を開発・公開することは，トロンプロジェクトとしては新しい試みであるが，仕様の速やかな普及に役立つものと確信している．

本仕様は，μITRON仕様に保護機能を導入するための最初のステップと位置付けられる．本仕様に盛り込んだ以外にも，状況によっては必要となる機能や有用な機能が考えられるが，最初のステップとの位置付けから，欲張りすぎないように留意した．今後，本仕様に準拠したカーネルの実装・応用を通してニーズをより正確に把握した上で，次のステップへ進めていきたいと考えている．この意味で，仕様に対するご意見・ご要望などを，トロン協会までお寄せいただければ幸いである．

最後に，本仕様の検討に直接的に貢献いただいたメモリ保護 SWG のメンバ各位を始めとして，トロンプロジェクトならびにIPAの関係者各位に感謝の意を表するとともに，引き続きのご支援・ご協力をお願いしたい．

2002年6月

高田 広章

トロン協会 バージョンアップWG 幹事／

豊橋技術科学大学 情報工学系

仕様書の構成

本仕様書は，μITRON4.0仕様（Ver. 4.01.00）に対して，メモリ領域を含むカーネルオブジェクトのアクセス保護機能を追加するための仕様を規定するものである．仕様のバージョン番号は，仕様書の表紙ならびに各ページの右上に表示されている．

本仕様書の構成は次の通りである．

第1章では，μITRON4.0仕様に保護機能を導入する目的と，この仕様の策定方針について述べる．この章の内容は，仕様の本体ではない．

第2章は，保護機能拡張の各種の概念と，複数の機能単位に共通する規定を示す．第3章では，μITRON4.0仕様の各機能単位毎に，保護機能拡張における変更内容を説明する．第4章では，保護機能拡張における新規機能について説明する．

第5章には，カーネルの仕様には直接反映されないが，本仕様に準拠したカーネルを実装・利用する上で参考となる情報を掲載する．第6章には，仕様を読む上で参考となる一覧表などを掲載する．第5章および第6章の内容は，仕様の本体ではない．

本仕様書の記述形式は，μITRON4.0仕様の記述形式を踏襲している．ただし，補足説明の項には，本来の補足説明に加えて，仕様決定の理由や従来の仕様との関連も含めて記述している．

目次

第1章 仕様策定の背景

1.1 保護機能導入の目的

μITRON 仕様のリアルタイムカーネルに対して保護機能を導入する目的として，大きく分けて次の3つが挙げられる．

(1) セキュリティ確保のための保護機能

悪意を持ったプログラムをダウンロード・実行した場合にも，システムの基幹部分を守るためには，カーネルが保護機能を持つことが必要である．

この目的のためには，保護を破る方法（セキュリティホール）があってはならないという意味で，完全な保護機能を提供することが要求される．また，共有メモリによるタスク間通信で完全な保護を実現するのは難しく，μITRON 仕様カーネル上での現状のアプリケーション構築モデルを見直さざるをえない部分がある．

(2) 信頼性向上のための保護機能

カーネルが保護機能を持つことにより，システムの一部分に問題があった場合にもシステム全体の動作を継続できるという意味で，システムの信頼性を向上させる効果がある．言い換えると，信頼性要件の異なるモジュールを，システム内で共存させることが可能になる．

この目的のためには，実行時のオーバヘッドを小さくすることが極めて重要である．どの程度の保護を実現すべきかは，各モジュールの信頼性要件がどの程度異なるかや，オーバヘッドとのトレードオフから決まることになる．そのため，オーバヘッド削減のためには，保護を破る方法があっても，偶然に起こる確率が十分に低ければ許容できる．また，現状のアプリケーション構築モデルの見直しが必要かどうかも，どの程度の保護が必要かに依存する．

(3) デバッグ支援のための保護機能

カーネルが保護機能を持つことにより，障害が生じた時に問題のある箇所の切り分けが容易になり，デバッグ作業を効率化する効果がある．また，潜在的な問題を早期に発見させるという効果もある．

この目的のためには，既存のアプリケーションを（できる限り）修正せずに，デバッグに必要な保護だけを導入したいという要求がある．

この目的で保護機能を用いる場合，製品出荷時には保護機能を取り外せると都合がよい（検証のやり直しになるので，取り外さない場合も多いだろう）．この観点からは，カーネルではなく，開発環境（例えばICE）で保護機能を実現する方が望ましいとも考えられる．

1.2 仕様策定の方針・目標

μITRON4.0仕様に対してメモリとカーネルオブジェクトに対するアクセス保護機能を追加する仕様を策定するにあたり，以下の方針・目標を設定した．

(1) 前述の3つの目的を，1つの仕様でカバーすることを目指す．

具体的には，悪意を持ったプログラムからも保護できるという意味で完全な保護機能を提供する方針で仕様を作成し，それをベースに実装毎に保護機能の一部を省略することを許す．そのためには，オーバヘッドの大きい保護機能を外しやすい仕様とすることが必要である．

また，保護機能を活用するには，既存のアプリケーション構築モデルを見直すことも重要であるが，一方で，既存のアプリケーションを（できる限り）修正せずに保護を導入したいという要求も強い．そこで，その両方の使い方が可能な仕様とすることを目指す．

(2) オーバヘッドの小さい実装が可能な仕様とすることを目指す．

このために，上記のようにオーバヘッドの大きい保護機能を外しやすくすることに加えて，次の2つの方針を設定する．

(A) アドレス変換を行う必要のない仕様とする．
(B) 静的な設定を活用して最適化する余地を大きくする．

ただし，これらの方針によりオーバヘッドがどの程度削減できるかは，メモリ保護のためのハードウェア（MMUなど）の機能に依存する．また，完全な保護機能を提供する場合には，ある程度のオーバヘッドは避けられない．

(3) なるべくシンプルな仕様とする．

ユーザがシステムの振舞いを理解しやすくするためには，仕様がシンプルであることが要求される．システムの振舞いを理解することは，誤ってセキュリティホールを作らないようにするためにも重要である．

また，シンプルな仕様とすることは，オーバヘッドを小さくすることにも貢献するが，実装毎に保護機能の一部を省略することを許すため，複雑な仕様であってもオーバヘッドの観点ではそれほど問題ないとも考えられる．

(4) 多重アドレス空間・多重ID空間はサポートしない．

オーバヘッド削減のために，アドレス変換が必要となる多重アドレス空間はサポートしない．論理アドレスと物理アドレスが一致するのが基本であるが，実装定義で論理アドレスと物理アドレスの間に何らかの変換を行ってもよい．また，それと整合させて，多重ID空間もサポートしない．ただし，将来のバージョンで多重アドレス空間・多重ID空間への拡張について検討する可能性はある．

第2章 仕様の概念と共通規定

2.1 主要な概念

(1) カーネルオブジェクト

カーネルが操作対象とする資源を，カーネルオブジェクト（kernel object）と呼ぶ（μITRON4.0仕様書 2.1.3節参照）.

この仕様におけるカーネルオブジェクトとは，μITRON4.0仕様に規定されているカーネルオブジェクトから，可変長メモリプールを除き，保護ドメイン，メモリオブジェクト，保護メモリプール，保護メールボックス，システム時刻，システム状態を加えたものである.

(2) 処理単位

カーネルが実行制御するプログラムの単位を，処理単位（processing unit）と呼ぶ（μITRON4.0仕様書 3.5.1節参照）．処理単位は，カーネルオブジェクトに含まれる.

この仕様における処理単位とは，μITRON4.0仕様に規定されている処理単位をいう．すなわち，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（周期ハンドラ，アラームハンドラ，オーバランハンドラ），CPU例外ハンドラ，拡張サービスコールルーチン，タスク，タスク例外処理ルーチンである.

(3) アクセス主体とアクセス対象

カーネルが提供するアクセス保護機能は，どのアクセス主体（access subject）がどのアクセス対象（access object）に対してどのような操作／アクセスを許可されているかを管理し，許可されていない操作／アクセスが行われないようにするための機能である.

この仕様では，アクセス主体は処理単位，アクセス対象はカーネルオブジェクトである．すなわち，カーネルは，どの処理単位がどのカーネルオブジェクトに対してどの種別の操作／アクセスを許可されているかを管理する.

【補足説明】

すべてのカーネルオブジェクトがアクセス保護の対象となるわけではない．アクセス保護の対象となるカーネルオブジェクトについては，2.3 節を参照すること.

拡張サービスコールルーチンは，それを呼び出した処理単位とは独立した処理単位である．そのため，タスクから呼び出された拡張サービスコールルーチンでカーネルオブジェクトを操作した場合，アクセス主体はタスクではなく拡張サービスコールルーチンとみなす．それに対して「自タスク」は，処理単位と

は独立の概念である．タスクから呼び出された拡張サービスコールルーチンでは，呼び出したタスクが「自タスク」となる．このことから，タスクから呼び出された拡張サービスコールルーチンにおいては，「実行中の処理単位の所属する保護ドメイン」と「自タスクの所属する保護ドメイン」は一致しない．CPU例外ハンドラがタスクコンテキストで実行される場合にも，これと同様である．

(4) 保護ドメイン

保護ドメイン（protection domain）は，アクセス保護機能を提供するための，カーネルオブジェクトの囲い（closure）である．

処理単位は，いずれか一つの保護ドメインに所属する．処理単位以外のカーネルオブジェクトは，いずれか一つの保護ドメインに所属するか，どの保護ドメインにも所属しない．

この仕様では，個々の処理単位に対してアクセス権を管理するのではなく，保護ドメインに対してアクセス権を管理する．すなわち，同じ保護ドメインに所属する処理単位は，同じアクセス権を持つ．そのため，この仕様が提供するアクセス保護機能に関する前述の記述を厳密化すると，「どの保護ドメインに所属する処理単位がどのカーネルオブジェクトに対してどの種別の操作／アクセスを許可されているかを管理する」となる．以下では，保護ドメインに所属する処理単位が操作／アクセスすることを，単に，保護ドメインが操作／アクセスするという．

保護ドメインに所属するカーネルオブジェクトは，デフォルトでは，同じ保護ドメインのみから操作／アクセスすることができる．デフォルトの設定を変更し，カーネルオブジェクト毎に操作／アクセスすることができる保護ドメインの集合を設定することができる．そのため，同じ保護ドメインに所属するカーネルオブジェクトであっても，同じように保護されるとは限らない．

なお保護ドメインは，カーネル仕様上はリンク単位とは独立であるが，コンフィギュレーション環境を構築する上では，リンク単位と関連づける必要がある．リンク単位と保護ドメインの関連については，5.1節を参照すること．

(5) アクセス許可パターンとアクセス許可ベクタ

アクセス許可パターンは（access permission pattern），あるカーネルオブジェクトに対して，ある種別の操作／アクセスを許可されている保護ドメインの集合を表す．

カーネルオブジェクトに対する操作／アクセスは，カーネルオブジェクトの種類毎に，4つの種別（通常操作1，通常操作2，管理操作，参照操作）に分類されている．アクセス許可ベクタ（access permission vector）は，操作／アクセスの種別毎の4つのアクセス許可パターンをひとまとめにしたものである．

ただし，μITRON4.0仕様のスタンダードプロファイルに対応する範囲のサービスコールは，通常操作1または通常操作2に分類する．そのため，スタンダー

ドプロファイルに対応する範囲の機能のみを実装する場合には，アクセス許可ベクタを，2つのアクセス許可パターンをひとまとめにしたものとすることができる．

また，実装定義で，カーネルオブジェクトに対する操作／アクセスを分類せず，1つのカーネルオブジェクトに対するすべての操作／アクセスの権限を，1つのアクセス許可パターンで表すことも許される．この場合には，アクセス許可ベクタは，1つのアクセス許可パターンのみで構成される．

(6) メモリオブジェクト

カーネルがメモリ保護の対象とする一連のメモリ領域を，メモリオブジェクト（memory object）と呼ぶ．メモリオブジェクトは，カーネルオブジェクトの一種であり，先頭番地によって識別する．また，メモリ領域に含まれる任意の番地により，メモリオブジェクトを指定する場合もある．

メモリオブジェクトの先頭番地とサイズは，ハードウェア的にメモリ保護が可能な境界・単位に制約される．また，メモリオブジェクトは，互いに重なりあうことはない．

この仕様では，保護メモリプールからのメモリブロックの獲得／返却時を除いては，メモリオブジェクトが動的に併合・分割されることはない．

(7) カーネルドメイン，システムドメイン，ユーザドメイン

カーネルと同じアクセス権，言い換えるとすべてのカーネルオブジェクトに対してすべての操作／アクセスができる保護ドメインを，カーネルドメイン（kernel domain）と呼ぶ．カーネルドメインに所属する処理単位は，プロセッサの特権モードで実行される．カーネルドメインは，システム内に唯一つ存在する．

そこに所属する処理単位は特権モードで実行されるが，カーネルオブジェクトに対する操作／アクセスには制限がある保護ドメインを，システムドメイン（system domain）と呼ぶ．特権モードで実行される処理単位からは，メモリ領域へのアクセス権を自由に変更することができるため，悪意を持ったプログラムからの保護にシステムドメインを使うことはできない．システムドメインは，システム内に複数存在することができる．

そこに所属する処理単位がプロセッサの非特権モードで実行され，カーネルオブジェクトに対する操作／アクセスに制限がある保護ドメインを，ユーザドメイン（user domain）と呼ぶ．ユーザドメインは，システム内に複数存在することができる．

(8) 無所属のオブジェクト

処理単位以外のカーネルオブジェクトは，どの保護ドメインにも所属しないものとすることができる．このようなカーネルオブジェクトを無所属のオブジェクト（independent object）と呼び，デフォルトでは，すべての保護ドメインか

ら操作／アクセスすることができる．保護ドメインに所属するカーネルオブジェクトと同様に，デフォルトの設定を変更し，カーネルオブジェクト毎に操作／アクセスすることができる保護ドメインの集合を設定することができる．

2.2 保護ドメイン管理と所属保護ドメインの設定

2.2.1 保護ドメインのID番号

保護ドメインは，ID番号で識別されるカーネルオブジェクトである．保護ドメインのID番号を保護ドメインIDと呼ぶ．

システムドメインとユーザドメインは，正のID番号を持つ（μITRON4.0仕様に規定しているシステムオブジェクトを負のID番号で区別する方法は採らない）．カーネルドメインは，特別なID番号TDOM_KERNEL（＝–1）を持つ．

また，TDOM_NONE（＝–2）によりどの保護ドメインにも所属しない（すなわち，無所属のオブジェクトである）ことを，TDOM_SELF（＝0）により自タスクの所属する保護ドメインを指定することができる．

2.2.2 保護ドメインの生成

この仕様では，保護ドメインは静的にのみ生成できるものとし，保護ドメインの動的な生成と削除はサポートしない．

保護ドメインを静的に生成するためには，コンフィギュレーションファイル中に次の記述を行う．

```
〈保護ドメイン種別〉〈保護ドメインID〉{
        保護ドメインに所属するカーネルオブジェクトの登録など
};
```

ここで，〈保護ドメイン種別〉にはsystem_domainとuser_domainのいずれかを，〈保護ドメインID〉には整数値または単一の識別子を記述する．単一の識別子を記述した場合には，コンフィギュレータがユニークな保護ドメインIDを割り付け，その識別子を割り付けたID番号にマクロ定義するプリプロセッサディレクティブ（#define）をkernel_id.h中に生成する．保護ドメインの囲み（ブロック）の中には，その保護ドメインに所属するカーネルオブジェクトを登録（生成・定義など）する静的APIを記述する．

保護ドメインに所属するカーネルオブジェクトがない場合や，前方参照のために保護ドメインの識別子のみを定義したい場合には，次の記法を用いる．

```
〈保護ドメイン種別〉〈保護ドメインID〉;
```

カーネルドメインは必ず存在し，生成する必要はない．カーネルドメインに所属するカーネルオブジェクトを登録する場合には，次の記法を用いる．

```
kernel_domain {
        カーネルドメインに所属するカーネルオブジェクトの登録など
```

};

同じ保護ドメインの囲みを，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．ただし，同じ〈保護ドメインID〉に対して，異なる〈保護ドメイン種別〉を指定した場合には，コンフィギュレータがエラーを報告する．

保護ドメインの囲みの外側にカーネルオブジェクトを登録するための静的APIを記述した場合，そのカーネルオブジェクトはどの保護ドメインにも所属しない無所属のオブジェクトとなる．

【補足説明】

保護ドメインの動的な生成と削除や，保護ドメインに対するその他の操作については，将来のバージョンで拡張を検討する．

保護ドメインに対する生成以外の操作は用意していないため，保護ドメインに対するアクセス許可ベクタを管理する必要はない．また，現状では保護ドメインを生成する静的APIは用意していないが，将来のバージョンで保護ドメインに対してアクセス許可ベクタを管理する必要が出てくれば，保護ドメインを生成する静的APIを設ける可能性もある．

2.2.3 カーネルオブジェクト生成時の所属保護ドメインの設定

生成対象となるカーネルオブジェクトが所属する保護ドメインは，生成時に設定することができる．この仕様では，タスク，セマフォ，イベントフラグ，データキュー，メールボックス，ミューテックス，メッセージバッファ，ランデブポート，固定長メモリプール，保護メモリプール，保護メールボックス，周期ハンドラ，アラームハンドラ，割込みサービスルーチンが，生成対象となるカーネルオブジェクトである．

カーネルオブジェクトを生成する静的APIにおいて，それが所属する保護ドメインを設定するには，前節で述べた通り，保護ドメインの囲みの内側に静的APIを記述すればよい．

カーネルオブジェクトをサービスコールによって動的に生成する場合には，オブジェクト属性により所属保護ドメインを設定する．具体的には，IDがdomidの保護ドメインに所属させる場合には，オブジェクト属性にTA_DOM (domid)を指定する．例えば，ID番号がDOM_Aの保護ドメインに所属するセマフォを生成するためには，セマフォ属性を次のように設定すればよい．

pk_csem.sematr = TA_DOM (DOM_A) |〈その他のセマフォ属性〉;

TA_DOMのパラメータ（domid）に，TDOM_KERNELまたはTDOM_SELFを指定して，それぞれカーネルドメインまたは自タスクと同じ保護ドメインに所属するカーネルオブジェクトを生成することができる．また，処理単位以外のカーネルオブジェクトを生成する場合には，domidにTDOM_NONEを指定して，無所属のカーネルオブジェクトを生成することができる．処理単位を生

成するサービスコールに対してdomidにTDOM_NONEを指定した場合には，サービスコールが E_RSATR エラーを返す．オブジェクト属性にTA_DOM (domid)を指定しない場合, domidにTDOM_SELFを指定した場合と同等となり, 生成されるカーネルオブジェクトは自タスクと同じ保護ドメインに所属する．

ただしこの仕様では，周期ハンドラ，アラームハンドラ，割込みサービスルーチンの3種類のカーネルオブジェクト（いずれも処理単位）は，カーネルドメインに所属するものと制限する. これらの3種類の処理単位を生成する静的API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．また，これらの3種類の処理単位を動的に生成するサービスコールでは，オブジェクト属性によりカーネルドメインに所属するように指定しなければならない. カーネルドメイン以外に所属するように指定された場合には，サービスコールがE_NOSPTエラーを返す. 実装独自に, これらの3種類の処理単位を他の保護ドメインに所属させることができるように拡張することは許される．

生成対象となる処理単位からこれらの3種類を除外すると, タスクのみが残る．タスクを生成する静的APIは，いずれかの保護ドメインの囲みの中に記述しなければならない．

2.2.4 カーネルオブジェクト定義時の所属保護ドメインの設定

この仕様では，割込みハンドラ，CPU例外ハンドラ，オーバランハンドラ，拡張サービスコールルーチン，タスク例外処理ルーチンが，定義対象となるカーネルオブジェクトである．これらのカーネルオブジェクトは，いずれも処理単位である．

これらの処理単位の中で，割込みハンドラ，CPU例外ハンドラ，オーバランハンドラ，拡張サービスコールルーチンは，カーネルドメインに所属する．これらの4種類の処理単位を定義する静的API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．また，この4種類の処理単位を動的に定義するサービスコールでは, オブジェクト属性により所属保護ドメインを設定することはできない．実装独自に，これらの4種類の処理単位を他の保護ドメインに所属させることができるように拡張することは許される．

タスク例外処理ルーチンは, タスクが所属するのと同じ保護ドメインに所属する．タスク例外処理ルーチンを定義する静的APIは，タスクと同じ保護ドメインの囲みの中に記述しなければならない．また，タスク例外処理ルーチンを動的に定義するサービスコールでは, オブジェクト属性により所属保護ドメインを設定することはできない．

2.2.5 メモリオブジェクトの所属保護ドメインの設定

メモリオブジェクトの所属する保護ドメインは, 保護メモリプールから獲得したメモリブロックを除いて，登録時に設定することができる．メモリオブジェ

クトの登録時の所属保護ドメインの設定方法については，4.1 節を参照すること．

2.2.6 システム時刻とシステム状態の所属保護ドメイン

システム時刻とシステム状態は，カーネルドメインに所属する．

2.2.7 所属保護ドメインの変更と参照

保護メモリプールから獲得したメモリブロックを除いて，カーネルオブジェクトが所属する保護ドメインを変更することはできない．また，カーネルオブジェクトの所属する保護ドメインを参照する方法は，標準では用意しない．

2.3 カーネルオブジェクトのアクセス保護

2.3.1 アクセス保護に関する共通事項

カーネルは，以下に列挙する種類のカーネルオブジェクトのそれぞれに対してアクセス許可ベクタを管理する．

- メモリオブジェクト
- タスク
- セマフォ，イベントフラグ，データキュー，メールボックス
- ミューテックス，メッセージバッファ，ランデブポート
- 固定長メモリプール
- 周期ハンドラ，アラームハンドラ
- 割込みサービスルーチン
- 保護メモリプール，保護メールボックス
- システム時刻
- システム状態

カーネルは，アクセス許可ベクタによって許可されていない操作／アクセスが行われようとした場合，それを検出し，許可されていない操作／アクセスが行われないようにする．

カーネルドメインからは，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操作／アクセスが許可される．この仕様で，ある保護ドメインのみが操作／アクセスできるといった場合でも，別に規定されている場合を除いて，カーネルドメインからは操作／アクセスできるものとする．

アクセス許可ベクタは，操作／アクセスの種別毎の4つのアクセス許可パターンをひとまとめにしたものである．アクセス許可パターンには，保護ドメインの任意の集合を指定できる．

アクセス保護に関して，実装定義で以下のサブセット化・制限を行うことが許される．これらのサブセット化・制限は，特定の種類のカーネルオブジェクト

に対してのみ行ってもよい.

(1) メモリオブジェクト以外のカーネルオブジェクトの保護を省略すること

メモリオブジェクト以外のカーネルオブジェクトをアクセス保護の対象から外し, それに対するアクセス許可ベクタの管理を省略することができる. アクセス保護の対象から外したカーネルオブジェクトに対しては, すべての保護ドメインからすべての操作／アクセスが許可される.

(2) 操作／アクセスの種別毎の保護を省略すること

操作／アクセスの種別毎にアクセス許可パターンを管理するのをやめ, すべての操作／アクセスが許可される保護ドメインの集合を表す1つのアクセス許可パターンのみを管理することができる. この場合には, アクセス許可ベクタは, 1つのアクセス許可パターンのみで構成される.

(3) アクセス許可パターンに指定できる保護ドメインの集合を制限すること

アクセス許可パターンに指定できる保護ドメインの集合を, そのカーネルオブジェクトが所属する保護ドメインのみ（プライベート）とすべての保護ドメイン（共有）のいずれかのみに制限することができる.

(2)と(3)の制限を組み合わせると, カーネルオブジェクトに対して設定できるアクセス許可ベクタを, 次の2種類にまで制限することができることになる.

(A) カーネルオブジェクトに対するすべての操作／アクセスを, それが所属する保護ドメインのみに許可する.

(B) カーネルオブジェクトに対するすべての操作／アクセスを, すべての保護ドメインに許可する.

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, アクセス許可ベクタによって許可されていない操作を行おうとした場合, 操作を行おうとしたサービスコールがE_OACVエラー（オブジェクトアクセス違反）を返す. メモリオブジェクトに対する操作／アクセス違反時の振舞いについては, 次節を参照すること.

【補足説明】

上で説明したE_OACVエラー（オブジェクトアクセス違反）は, 多くのサービスコールで発生する可能性があるため, サービスコールが返すメインエラーコードとしてサービスコール毎には記述しない（μITRON4.0仕様書 2.1.6節参照）.

2.3.2 メモリオブジェクトのアクセス保護

メモリオブジェクトのアクセス保護には, 以下に述べる点を除いて, 前節のアクセス保護に関する共通事項が適用される.

メモリオブジェクトに対する操作／アクセスは, 書込みアクセス, 読出しアクセス（実行アクセスを含む）, 管理操作, 参照操作の4つの種別に分類される.

この仕様に準拠したカーネルというためには, メモリオブジェクトのアクセス

保護機能は必須の機能であり，省略することはできない．ただし，実装定義で以下のサブセット化・制限を行うことが許される．

(1) 管理操作と参照操作の保護を省略すること

メモリオブジェクトに対する管理操作と参照操作を保護の対象から外し，それに対するアクセス許可パターンの管理を省略することができる．管理操作と参照操作を保護の対象から外した場合，すべての保護ドメインからこれらの操作が許可される．

(2) サービスコールを通じた書込みアクセスと読出しアクセスの保護を省略すること

サービスコールを通じて許可されていないメモリオブジェクトへの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（言い換えると，サービスコールに渡したポインタの指すメモリ領域がアクセスを許可されていないメモリオブジェクトに含まれる場合）にそれを検出する機能を省略することができる．

(3) 操作／アクセスの種別毎の保護を省略すること

操作／アクセスの種別毎にアクセス許可パターンを管理するのをやめ，すべての操作／アクセスが許可される保護ドメインの集合を表す1つのアクセス許可パターンのみを管理することができる．この場合には，アクセス許可ベクタは，1つのアクセス許可パターンのみで構成される．

(4) アクセス許可パターンに指定できる保護ドメインの集合を制限すること

アクセス許可パターンに指定できる保護ドメインの集合を，そのメモリオブジェクトが所属する保護ドメインのみ（プライベート）とすべての保護ドメイン（共有）のいずれかのみに制限することができる．

(3)と(4)の制限を組み合わせると，メモリオブジェクトに対して設定できるアクセス許可ベクタを，次の2種類にまで制限することができることになる．

(A) メモリオブジェクトに対するすべての操作／アクセスを，メモリオブジェクトが所属する保護ドメインにのみ許可する．このようなアクセス許可ベクタを，PRW（private read/write）と呼ぶ．

(B) メモリオブジェクトに対するすべての操作／アクセスを，すべての保護ドメインに許可する．このようなアクセス許可ベクタを，SRW（shared read/write）と呼ぶ．

これらに次いでサポートが望まれるアクセス許可ベクタの設定として，次の3種類を挙げることができる．

(C) メモリオブジェクトに対する書込みアクセス以外の操作／アクセスを，メモリ領域が所属する保護ドメインにのみ許可する．このようなアクセス許可ベクタを，PRO（private read only）と呼ぶ．

(D) メモリオブジェクトに対する書込みアクセス以外の操作／アクセスを，すべての保護ドメインに許可する．このようなアクセス許可ベクタを，SRO（shared read only）と呼ぶ．

(E) メモリオブジェクトに対する読出しアクセスをすべての保護ドメインに許可し，それ以外の操作／アクセスをメモリオブジェクトが所属する保護ドメインにのみ許可する．このようなアクセス許可ベクタを，SRPW（shared read private write）と呼ぶ.

メモリオブジェクトに対して, アクセス許可ベクタによって許可されていない操作／アクセスを行おうとした場合の振舞いは次の通りである.

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には，操作を行おうとしたサービスコールがE_OACVエラー（オブジェクトアクセス違反）を返す.

メモリオブジェクトに対し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で, アプリケーションプログラム内で許可されていないメモリアクセスを直接行おうとした場合には, 実装定義のCPU例外ハンドラを起動する.

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アクセスを行おうとした場合には，アクセスしようとしたサービスコールがE_MACVエラー（メモリアクセス違反）を返すか, いずれかのCPU例外ハンドラを起動する．どちらの方法を採るかは，実装定義である.

上の2つのケースにおいては，起動されたCPU例外ハンドラの中で，メモリ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，アクセスの方法，命令の番地など）を取り出せる方法を，実装定義で用意しなければならない.

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を指定して管理操作または参照操作を行おうとした場合には, 操作を行おうとしたサービスコールがE_NOEXSエラーを返す．また，カーネルドメイン以外の保護ドメインから, そのようなメモリ領域に対して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は, アクセスが許可されていない場合と同様に振る舞う.

【補足説明】

上で説明したE_OACVエラー（オブジェクトアクセス違反）とE_MACVエラー（メモリアクセス違反）は，多くのサービスコールで発生する可能性があるため, サービスコールが返すメインエラーコードとしてサービスコール毎には記述しない（μITRON4.0仕様書 2.1.6節参照）.

この仕様の将来のバージョンでは, メモリアクセス違反を検出した場合に起動するCPU例外ハンドラをメモリアクセス違反ハンドラとして規定し, 標準化の度合いを強くする可能性もある.

2.3.3 システム時刻とシステム状態のアクセス保護

この仕様では，通常のカーネルオブジェクトに加えて，システム時刻とシステム状態もカーネルオブジェクトの一種であるものとし, システム時刻とシステ

ム状態に対するアクセス許可ベクタを管理する. 通常のカーネルオブジェクトを操作対象としないサービスコールで，何らかの呼び出し制限が必要なものは, システム時刻ないしはシステム状態を操作対象とするサービスコールであるものと考え, それに対するアクセス許可ベクタで呼び出せるかどうかを決定する.

また，カーネルオブジェクトを生成または定義するサービスコールは，システム状態を操作対象とするサービスコールと考え, システム状態に対する管理操作のアクセス許可パターンにより呼び出せるかどうかを決定する.

2.3.4 カーネルオブジェクトに対する操作／アクセスの種別

カーネルオブジェクトに対する操作／アクセスを, カーネルオブジェクトの種類毎に，(1) 通常操作1，(2) 通常操作2，(3) 管理操作，(4) 参照操作の4つの種別に分類する.

以下に，カーネルオブジェクトの種類毎の具体的な分類を示す．なお，対応するタスクコンテキスト専用のサービスコールが存在する非タスクコンテキスト専用のサービスコールは, 対応するタスクコンテキスト専用のサービスコールと同じ種別に分類されるため，以下では列挙していない.

- メモリオブジェクト
  - (1) 書込みアクセス
  - (2) 読出しアクセス（実行アクセスを含む）
  - (3) 管理操作（det_mem，sac_mem）
  - (4) 参照操作（ref_mem，prb_mem）
- タスク
  - (1) 通常操作（act_tsk, can_act, sta_tsk, wup_tsk, can_wup, get_pri）
  - (2) 特殊操作（ter_tsk, chg_pri, rel_wai, sus_tsk, rsm_tsk, frsm_tsk, ras_tex，sta_ovr，stp_ovr）
  - (3) 管理操作（del_tsk，exd_tsk，sac_tsk，def_tex）
  - (4) 参照操作（ref_tsk，ref_tst，ref_tex，ref_ovr）
- セマフォ
  - (1) 返却操作（sig_sem）
  - (2) 獲得操作（wai_sem，pol_sem，twai_sem）
  - (3) 管理操作（del_sem，sac_sem）
  - (4) 参照操作（ref_sem）
- イベントフラグ
  - (1) セット／リセット操作（set_flg，clr_flg）
  - (2) 待ち操作（wai_flg，pol_flg，twai_flg）
  - (3) 管理操作（del_flg，sac_flg）
  - (4) 参照操作（ref_flg）

- データキュー
  - (1) 送信操作（snd_dtq，psnd_dtq，tsnd_dtq，fsnd_dtq）
  - (2) 受信操作（rcv_dtq，prcv_dtq，trcv_dtq）
  - (3) 管理操作（del_dtq，sac_dtq）
  - (4) 参照操作（ref_dtq）
- メールボックス
  - (1) 送信操作（snd_mbx）
  - (2) 受信操作（rcv_mbx，prcv_mbx，trcv_mbx）
  - (3) 管理操作（del_mbx，sac_mbx）
  - (4) 参照操作（ref_mbx）
- ミューテックス
  - (1) ロック操作（loc_mtx，ploc_mtx，tloc_mtx）
  - (2) 未使用
  - (3) 管理操作（del_mtx，sac_mtx）
  - (4) 参照操作（ref_mtx）
- メッセージバッファ
  - (1) 送信操作（snd_mbf，psnd_mbf，tsnd_mbf）
  - (2) 受信操作（rcv_mbf，prcv_mbf，trcv_mbf）
  - (3) 管理操作（del_mbf，sac_mbf）
  - (4) 参照操作（ref_mbf）
- ランデブポート
  - (1) 呼出し操作（cal_por，tcal_por，fwd_por）
  - (2) 受付操作（acp_por，pacp_por，tacp_por）
  - (3) 管理操作（del_por，sac_por）
  - (4) 参照操作（ref_por，ref_rdv）

    ※ ref_rdvは，操作対象のランデブが成立したランデブポートに対する参照操作が許可されている保護ドメインのみから呼び出すことができる．
- 固定長メモリプール
  - (1) 獲得操作（get_mpf，pget_mpf，tget_mpf）
  - (2) 返却操作（rel_mpf）
  - (3) 管理操作（del_mpf，sac_mpf）
  - (4) 参照操作（ref_mpf）
- システム時刻
  - (1) 設定操作（set_tim）
  - (2) 読出し操作（get_tim）
  - (3) 管理操作（sac_tim）
  - (4) 参照操作（ref_tim）

- 周期ハンドラ
  - (1) 開始操作（sta_cyc）
  - (2) 停止操作（stp_cyc）
  - (3) 管理操作（del_cyc，sac_cyc）
  - (4) 参照操作（ref_cyc）
- アラームハンドラ
  - (1) 開始操作（sta_alm）
  - (2) 停止操作（stp_alm）
  - (3) 管理操作（del_alm，sac_alm）
  - (4) 参照操作（ref_alm）
- 保護メモリプール
  - (1) 獲得操作（get_mpp，pget_mpp，tget_mpp）
  - (2) 未使用
  - (3) 管理操作（del_mpp，trf_mpp，sac_mpp）
  - (4) 参照操作（ref_mpp）
- 保護メールボックス
  - (1) 送信操作（snd_mbp）
  - (2) 受信操作（rcv_mbp，prcv_mbp，trcv_mbp）
  - (3) 管理操作（del_mbp，sac_mbp）
  - (4) 参照操作（ref_mbp）
- 割込みサービスルーチン
  - (1) 未使用
  - (2) 未使用
  - (3) 管理操作（del_isr，sac_isr）
  - (4) 参照操作（ref_isr）
- システム状態
  - (1) スケジューリング操作（rot_rdq，dis_dsp，ena_dsp）
  - (2) 割込み操作（loc_cpu, unl_cpu, ena_int, dis_int, chg_ixx, get_ixx）
  - (3) 管理操作（cre_*yyy*，cra_*yyy*，acre_*yyy*，acra_*yyy*，att_*yyy*，ata_*yyy*，def_inh，def_exc，def_svc，def_ovr，sac_sys）
  - (4) 参照操作（ref_sys，ref_cfg，ref_ver）

次のサービスコールは，カーネルドメインのみから呼び出すことができる．

- タイムティックの供給（isig_tim）

次のサービスコールは保護の対象外であり，すべての保護ドメインから呼び出すことができる．

- 自タスクへの操作（ext_tsk，slp_tsk，dly_tsk，dis_tex，ena_tex）
- システム状態参照（get_tid，get_did，sns_ctx，sns_loc，sns_dsp，sns_dpn）
- タスク例外状態参照（sns_tex）

次のサービスコールは, 呼び出せるタスクないしは保護ドメインがアクセス許可ベクタ以外の方法で限定されているため, アクセス許可ベクタによる保護を行わない.

- ミューテックスのロック解除（unl_mtx）
- ランデブの終了（rpl_rdv）
- 保護メモリブロックの返却（rel_mpp）

【補足説明】

カーネルオブジェクトに対する操作／アクセスの種別は, おおよそ次の原則に従って定めた．通常操作1は，先に呼ぶことが多い操作／データを送る側の操作, 通常操作2は, 後に呼ぶことが多い操作／データを受ける側の操作とした．また，スタンダードプロファイルに相当する機能は，通常操作1と通常操作2でカバーされるようにした.

ランデブの回送（fwd_por）には，回送するランデブに関する保護（アクセス許可ベクタを使わない）と回送先のランデブポートに関する保護（アクセス許可ベクタを使う）の両方が適用される.

2.4 アクセス許可ベクタの管理

2.4.1 アクセス許可ベクタのデータ型・定数・マクロ

アクセス許可パターンとアクセス許可ベクタのデータ型として, 次のデータ型を定義する.

ACPTN	アクセス許可パターン
ACVCT	アクセス許可ベクタ

ACPTNの定義やサイズは，実装定義である．ACVCTは，ACPTNを用いて次のように定義する.

```
typedef struct {
        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 ;
```

ただし，アクセス許可ベクタを2つのアクセス許可パターンで構成する場合には，acptn3とacptn4を除くものとする．また，アクセス許可ベクタを1つのアクセス許可パターンのみで構成する場合には，acptn2 〜 acptn4 を除き，acptn1のみで構成される構造体とする.

アクセス許可パターンを記述するために，次の定数およびマクロを定義する.

ACPTN acptn = TACP (ID domid)

domid で指定される保護ドメインのみが操作／アクセスできることを示すアクセス許可パターン

TACP_KERNEL	カーネルドメインのみが操作／アクセスできることを示すアクセス許可パターン
TACP_SHARED	すべての保護ドメインが操作／アクセスできることを示すアクセス許可パターン

ただし，TACPのパラメータ（domid）に，TDOM_KERNEL，TDOM_NONE，TDOM_SELFを指定することはできない．指定した場合の振舞いは未定義である．

また，アクセス許可パターンに対する演算子として，“|”演算子を用いることができる．acptn1とacptn2を任意のアクセス許可パターンとした場合に，(acptn1 | acptn2)は，acptn1とacptn2のいずれかまたは両方のアクセス許可パターンが操作／アクセスを許可する保護ドメインのみが，操作／アクセスできることを示すアクセス許可パターンを示す．

アクセス許可ベクタを記述するために，以下の定数およびマクロを定義する．

ACVCT acvct = TACT_PRIVATE (ID domid)

すべての操作／アクセスが，domidで指定される保護ドメインのみに許可されることを示すアクセス許可ベクタ

TACT_KERNEL	すべての操作／アクセスが，カーネルドメインのみに許可されることを示すアクセス許可ベクタ
TACT_SHARED	すべての操作／アクセスが，すべての保護ドメインに許可されることを示すアクセス許可ベクタ

また，メモリオブジェクトに対するアクセス許可ベクタを記述するために，次の定数およびマクロを定義する．

ACVCT acvct = TACT_PRW (ID domid)

メモリオブジェクトに対するすべての操作／アクセスが，domidで指定される保護ドメインにのみ許可されていることを示すアクセス許可ベクタ

ACVCT acvct = TACT_PRO (ID domid)

メモリオブジェクトに対する書込みアクセス以外の操作／アクセスが，domidで指定される保護ドメインにのみ許可されていることを示すアクセス許可ベクタ

TACT_SRW	メモリオブジェクトに対するすべての操作／アクセスが，すべての保護ドメインに許可されていることを示すアクセス許可ベクタ
TACT_SRO	メモリオブジェクトに対する書込みアクセス以外の操作／アクセスが，すべての保護ドメインに許可されていることを示すアクセス許可ベクタ

ACVCT acvct = TACT_SRPW (ID domid)

メモリオブジェクトに対する読出しアクセスがすべての保護ドメイン

に許可され, それ以外の操作／アクセスがdomidで指定される保護ドメインにのみ許可されていることを示すアクセス許可ベクタ

ただし，アクセス許可ベクタを記述するためのこれらの定数およびマクロは，静的APIのパラメータとC言語の初期代入文の右辺にのみ用いることができる．また，これらのマクロのパラメータ（domid）に，TDOM_KERNEL, TDOM_NONE，TDOM_SELFを指定することはできない．指定した場合の振舞いは未定義である．

静的APIのパラメータにアクセス許可ベクタを記述する場合には，上で定義したアクセス許可ベクタを記述するための定数またはマクロを用いるか, 4つ(実装によっては2つまたは1つ) のアクセス許可パターンを“,”で区切り, “{”と“}”で囲んだ形で記述する．

【補足説明】

アクセス許可パターンの“|”演算子を用いることで，例えば，DOM_AとDOM_Bの2つの保護ドメインのみが操作／アクセスできることを示すアクセス許可パターンを，(TACP (DOM_A) | TACP (DOM_B)) と記述することができる．

ただし, “|”演算子を用いる方法には, アクセス許可パターンの実装方法をビットパターンに限定してしまうという問題がある．また，保護ドメインの数が，C言語の処理系で扱える最長の整数型のビット数で制限される．それにもかかわらずこの方法を採ったのは，次のような考察の結果である．

(1) 他に良い方法がないこと

C言語のプリプロセッサには, 任意個の引数を取るマクロを定義する機能がない．例えば，DOM_AとDOM_Bの2つの保護ドメインのみからアクセスできる場合を, TACP_UNION (TACP (DOM_A), TACP (DOM_B))のように記述することはできるが，その場合には，3つの保護ドメインからアクセスできる場合をTACP_UNION (TACP (DOM_A), TACP (DOM_B), TACP (DOM_C)) とは記述できず，TACP_UNION (TACP (DOM_A), TACP_UNION (TACP (DOM_B), TACP (DOM_C))) と記述しなくてはならない．このような記述は，記述が長くなるだけでなく，ソースコードの読み易さを損なうという問題がある．

(2) “|”演算子のサポートは必須ではないこと

この仕様では，ある保護ドメインのみからアクセスできるアクセス許可パターンと，すべての保護ドメインがアクセスできるアクセス許可パターンのサポートは必須であるが，指定した複数の保護ドメインからアクセスできるアクセス許可パターンのサポートは必須ではない．メモリをより節約することを狙う場合には，“|”演算子はサポートしないと考えられる．

(3) C++を使えば“|”演算子をオーバロードできること

他に良い方法がないのは, C言語のプリプロセッサの限界によるところが大きいが，プリプロセッサを拡張する位であれば，C++を使って“|”演算子を

オーバロードする方法がより簡単である．保護ドメインの数が1ワードのビットパターンで表せる範囲で不足するような場合は，かなり大規模なアプリケーションを想定していると考えられ，C++を使って解決する方法が有力になる．ただし，C++を使っても，静的APIについては解決できない（オーバロードした“|”演算子を使うと定数と解釈できなくなるため）．

静的APIのパラメータにアクセス許可ベクタを記述する例を以下に示す．

```
CRA_TSK ( MAIN_TASK, { TA_HLNG, 0, main_task,
                       MAIN_TASK_PRIORITY, STACK_SIZE,
                       NULL, SSTACK_SIZE, NULL },
                       TACT_PRIVATE ( DOM_A) ) ;
CRA_SEM ( SEM_X, { TA_TFIFO, 1, 1 }, { TACP ( DOM_A ),
                       TACP(DOM_A), TACP_KERNEL,
                       TACP_KERNEL } ) ;
```

2.4.2 デフォルトのアクセス許可ベクタ

カーネルオブジェクトに対するデフォルトのアクセス許可ベクタは，次のように定義される．

(1) 保護ドメインに所属するカーネルオブジェクトに対しては，すべての種別の操作／アクセスをその保護ドメインのみに許可するアクセス許可ベクタ（カーネルドメインに所属する場合にはTACT_KERNEL，それ以外の保護ドメインに所属する場合にはTACT_PRIVATE (domid)．domidは所属する保護ドメインID）がデフォルトとなる．

(2) 無所属のカーネルオブジェクトに対しては，すべての種別のアクセス／操作をすべての保護ドメインに許可するアクセス許可ベクタ（TACT_SHARED）がデフォルトとなる．

2.4.3 アクセス許可ベクタの設定

アクセス許可ベクタを指定せずにカーネルオブジェクトを生成する機能を用いた場合，生成されるカーネルオブジェクトに対しては，デフォルトのアクセス許可ベクタが設定される．

アクセス許可ベクタを指定してカーネルオブジェクトを生成する機能として，カーネルオブジェクトを生成する静的API CRE_*YYY*に対応してCRA_*YYY*，サービスコールcre_*yyy*およびacre_*yyy*に対応してcra_*yyy*およびacra_*yyy*を新設する．

新設する静的APIおよびサービスコールは，元となる静的APIまたはサービスコールに，生成するカーネルオブジェクトのアクセス許可ベクタ（サービスコールの場合は，アクセス許可ベクタへのポインタ）を指定するパラメータを追加したものである．

メモリオブジェクトの登録時のアクセス許可ベクタの設定方法については，

4.1節を参照すること.

システム時刻とシステム状態のアクセス許可ベクタを設定する機能として, それぞれSAC_TIMとSAC_SYSを新設する. これらの静的API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. また, これらの静的APIのそれぞれは, 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してはならない. これらの静的APIによりアクセス許可ベクタを設定しない場合には, デフォルトのアクセス許可ベクタ（すなわち, すべての種別の操作／アクセスをカーネルドメインのみに許可するアクセス許可ベクタ）が設定される.

2.4.4 アクセス許可ベクタの変更

カーネルオブジェクトのアクセス許可ベクタを変更するための機能として, アクセス許可ベクタを持つカーネルオブジェクトの種類毎に, サービスコール sac_*yyy* を新設する.

待ち行列を持つカーネルオブジェクトのアクセス許可ベクタを変更するサービスコールは, 呼び出され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. 言い換えると, カーネルオブジェクトの待ち行列につながれているタスクが, カーネルオブジェクトに対するアクセスを許可されているかの再チェックは行わない.

【補足説明】

アクセス許可ベクタの変更により, 不都合な状況が起こら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, タスクがセマフォ資源を獲得した状態または獲得を待っている状態で, そのタスクが所属する保護ドメインからセマフォに対する返却操作を許可しないようにアクセス許可ベクタを変更すると, タスクは獲得したセマフォ資源を返却できなくなる（別のタスクに返却させることは可能である）.

2.4.5 アクセス許可ベクタの参照

カーネルオブジェクトのアクセス許可ベクタを参照するための機能として, アクセス許可ベクタを持つカーネルオブジェクトの状態を参照するサービスコールref_*yyy*で参照できる情報に, カーネルオブジェクトのアクセス許可ベクタを追加する. これに伴い, T_R*YYY*型の標準化されているフィールドの最後に, ACVCT型のフィールドacvctを追加する.

また, メモリオブジェクトとシステム時刻のアクセス許可ベクタを参照するための機能として, それぞれref_memとref_timを新設する.

2.5 カーネルオブジェクトに必要なメモリ領域

2.5.1 カーネルの用いる管理領域

μITRON4.0仕様においては，カーネルオブジェクトに必要なメモリ領域の中でサイズが一定しないものを，カーネルオブジェクト生成時にアプリケーション側で指定するか，カーネルが確保する仕様としている．それらのメモリ領域の中で，カーネル自身の動作に必要な情報を置くためのメモリ領域を，カーネルの用いる管理領域と呼ぶ．

アプリケーションプログラムからカーネルを保護するためには，カーネルの用いる管理領域を，カーネルドメイン以外の保護ドメインからはアクセスできないメモリ領域に置くことが必要である．この仕様においては，以下に挙げるメモリ領域が，カーネルの用いる管理領域に該当する．

- タスクのシステムスタック領域
- データキュー管理領域
- メールボックス管理領域
- メッセージバッファ管理領域
- 固定長メモリプール管理領域
- 保護メモリプール管理領域
- 保護メールボックス管理領域

各カーネルオブジェクトの制御ブロックやメモリオブジェクトの管理領域など，カーネルが使用するメモリ領域はこの他にもあるが，ここで言う「カーネルの用いる管理領域」には含めない．

カーネルの用いる管理領域を保護するためには，アプリケーションが用いるメモリ領域と分離することが必要である．具体的には，μITRON4.0仕様に対して，以下の管理領域の扱いを変更する．ただし，完全な保護機能が必要とされない場合には，これらを変更しない実装を行うことも許される．

(1) タスクのシステムスタック領域

ユーザドメインに所属するタスク自身は非特権モードで実行されるが，タスクから呼び出したサービスコール（拡張サービスコールを含む）の処理ルーチンは特権モードで実行される．ユーザドメインに所属するタスクが呼び出したサービスコールを実行するために使うスタックをシステムスタック，そのためのメモリ領域をシステムスタック領域と呼ぶ．

システムスタックには，特権モードで実行されるプログラムからの返り番地などが積まれており，それを書き換えると，カーネルを異常動作させることができる．そのため，システムスタック領域は，カーネルドメイン以外の保護ドメインからはアクセスできないメモリ領域に置く必要がある．それに対して，非特権モードで実行されるタスクのスタック領域は，そのタスクからアクセスできる必要がある．これらの理由により，ユーザドメインに所属するタスクに対

しては，タスクのスタック領域とシステムスタック領域を分離する必要がある．

(2) メールボックス管理領域

μITRON4.0仕様のメールボックス機能では，メッセージキューを作るために，メッセージの先頭に置かれたメッセージヘッダを用いている．そのため，メッセージキューの内容を書き換えるとカーネルを異常動作させることができ，カーネルの保護のためには適切でない．

そこでこの仕様では，メッセージキューを作るためにメッセージヘッダを用いる仕様を改め，そのためのメモリ領域を，カーネルドメインのみがアクセスできるメモリ領域の中に確保することとする．このメモリ領域のサイズを決定するために，メールボックス生成時のパラメータに，メールボックスに入れることができるメッセージの数を追加する．また，このメモリ領域は，μITRON4.0仕様でも必要であった優先度別のメッセージキューヘッダ領域と統合して，メールボックス管理領域と呼ぶ．それに伴って，メールボックス生成時のパラメータを，優先度別のメッセージキューヘッダ領域の先頭番地から，メールボックス管理領域の先頭番地に変更する．

(3) 固定長メモリプール管理領域

μITRON4.0仕様の固定長メモリプール機能では，返却されたメモリブロックを管理するための情報を，メモリブロック内に置くように実装することを想定している．そのため，返却されたメモリブロックの内容を書き換えるとカーネルを異常動作させることができ，カーネルの保護のためには適切でない．

そこでこの仕様では，返却されたメモリブロックを管理するための情報をメモリブロック内に置く想定を改め，そのためのメモリ領域を，カーネルドメインのみがアクセスできるメモリ領域の中に確保することとする．このメモリ領域を，固定長メモリプール管理領域と呼ぶ．それに伴って，固定長メモリプールの生成時のパラメータに，固定長メモリプール管理領域の先頭番地を追加する．

【補足説明】

カーネルの用いる管理領域であることを明確にするために，データキュー領域をデータキュー管理領域に，メッセージバッファ領域をメッセージバッファ管理領域に，それぞれ名称変更した．

2.5.2 管理領域の保護

前節で述べた通り，カーネルの用いる管理領域は，カーネルドメイン以外の保護ドメインからはアクセスできないメモリ領域に置くことが必要である．そこで，カーネルオブジェクトを生成する静的APIまたはサービスコールが，カーネルの用いる管理領域の先頭番地をパラメータとして取る場合には，次のよう

に扱う．

先頭番地を指定するパラメータにNULLが指定された場合には，必要なサイズの管理領域を，すべての操作／アクセスがカーネルドメインのみに許可されているメモリ領域の中に，カーネルが確保する．

NULL以外の値が指定された場合には，パラメータで指定された管理領域が適切であるかチェックし，適切でない場合にはエラーとする．具体的には，パラメータで指定された管理領域が，メモリオブジェクトの境界を越えている場合や，何らかの操作／アクセスがカーネルドメイン以外にも許可されているメモリオブジェクトに含まれる場合に，E_PARエラーとする．

2.5.3 カーネルオブジェクトに必要なその他のメモリ領域

カーネルオブジェクトに必要なメモリ領域の中でサイズが一定しないものの中には，カーネルの用いる管理領域には該当しないものがある．以下では，カーネルの用いる管理領域には該当しない各メモリ領域について，カーネルオブジェクトを生成する静的APIまたはサービスコールが先頭番地をパラメータとして取る場合の扱いについて述べる．

(1) タスクのスタック領域

スタック領域の先頭番地を指定するパラメータにNULLが指定された場合には，必要なサイズのメモリ領域を，タスクと同じ保護ドメインに所属し，すべての操作／アクセスがタスクの所属保護ドメインのみに許可されているメモリ領域の中に，カーネルが確保する．NULL以外の値が指定された場合には，指定されたメモリ領域をスタック領域として用いる．

(2) 固定長メモリプール領域

固定長メモリプール領域の先頭番地を指定するパラメータにNULLが指定された場合には，必要なサイズのメモリ領域を，固定長メモリプールと同じ保護ドメインに所属し，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領域の中に，カーネルが確保する．NULL以外の値が指定された場合には，指定されたメモリ領域を固定長メモリプール領域として用いる．

(3) 保護メモリプール領域

保護メモリプール領域は，カーネルの用いる管理領域には該当しないが，カーネルの用いる管理領域と同様に扱う．すなわち，保護メモリプール領域の先頭番地を指定するパラメータにNULLが指定された場合には，必要なサイズのメモリ領域を，すべての操作／アクセスがカーネルドメインのみに許可されているメモリ領域の中に，カーネルが確保する．NULL以外の値が指定された場合には，パラメータで指定されたメモリ領域が適切であるかチェックし，適切でない場合にはエラーとする．具体的なチェック条件については，cre_mppの機能説明を参照すること．

【補足説明】

これらのメモリ領域は，それぞれ単独のメモリオブジェクトとして登録する必要はなく，所属保護ドメイン，メモリオブジェクト属性，アクセス許可ベクタがすべて一致するメモリ領域を，1つのメモリオブジェクトにまとめて登録してもよい．また，これらがすべて一致すれば，プログラムモジュールのためのメモリオブジェクトにまとめて登録してもよい．

2.6 仕様準拠の条件

この仕様は，μITRON仕様の弱い標準化の方針を踏襲して策定されている．仕様準拠の条件に関する基本的な考え方は，μITRON4.0仕様と同様である（μITRON4.0仕様書 5.1.1節参照）．

ただし，この仕様に準拠しているというためには，最低限の保護機能は持っていなければならない．そこで，この仕様に準拠しているというために必要な最低限の機能を，μITRON4.0仕様に準拠しているというために必要な最低限の機能（μITRON4.0仕様書 5.1.2節参照）に，以下の機能を追加したものとする．

(A) メモリオブジェクトを登録できること．
(B) カーネルドメイン以外の保護ドメインを複数生成でき，タスクとメモリオブジェクトを，それらに所属させられること．
(C) メモリオブジェクトに対するアクセス保護機能を提供すること．

第3章 従来機能の変更

3.1 タスク管理機能

タスク管理機能には，アクセス許可ベクタを指定してタスクを生成する機能と，タスクのアクセス許可ベクタを変更する機能を追加する．また，タスク生成時に指定する情報と，参照できるタスク状態に変更がある．

タスク生成情報のパケット形式にタスクのシステムスタック領域のサイズと先頭番地を，タスク状態のパケット形式にアクセス許可ベクタを追加する．

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

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

タスク管理機能の新規サービスコールの機能コードは次の通りである．

TFN_CRA_TSK	–0x111	cra_tskの機能コード
TFN_ACRA_TSK	–0x121	acra_tskの機能コード
TFN_SAC_TSK	–0x131	sac_tskの機能コード

CRE_TSK	タスクの生成（静的API）
CRA_TSK	タスクの生成（静的API，アクセス許可指定）
cre_tsk	タスクの生成
cra_tsk	タスクの生成（アクセス許可指定）
acre_tsk	タスクの生成（ID番号自動割付け）
acra_tsk	タスクの生成（ID番号自動割付け，アクセス許可指定）

【静的API】

```
CRE_TSK ( ID tskid, { ATR tskatr, VP_INT exinf, FP task,
                      PRI itskpri, SIZE stksz, VP stk,
                      SIZE sstksz, VP sstk } ) ;
CRA_TSK ( ID tskid, { ATR tskatr, VP_INT exinf, FP task,
                      PRI itskpri, SIZE stksz, VP stk,
                      SIZE sstksz, VP sstk }, ACVCT acvct ) ;
```

【C言語API】

```
ER ercd = cre_tsk ( ID tskid, T_CTSK *pk_ctsk ) ;
ER ercd = cra_tsk ( ID tskid, T_CTSK *pk_ctsk,
                    ACVCT *p_acvct ) ;
ER_ID tskid = acre_tsk ( T_CTSK *pk_ctsk ) ;
ER_ID tskid = acra_tsk ( T_CTSK *pk_ctsk, ACVCT *p_acvct ) ;
```

【パラメータ】

ID	tskid	生成対象のタスクのID番号
T_CTSK *	pk_ctsk	タスク生成情報を入れたパケットへのポインタ（静的APIではパケットの内容を直接記述する）
ACVCT	acvct	タスクのアクセス許可ベクタ

pk_ctskの内容（T_CTSK型）

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

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cre_tskの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acre_tskの場合

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

【エラーコード】

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

【機能】

tskidで指定されるID番号を持つタスクを，pk_ctskで指定されるタスク生成情報に基づいて生成する．μITRON4.0仕様との違いは次の通り．

sstkszはタスクのシステムスタック領域のサイズ（バイト数），sstkはタスクのシステムスタック領域の先頭番地である．

stkで指定された番地からstkszバイトのメモリ領域を, タスクを実行するためのスタック領域として使用する．スタック領域として使用するメモリ領域が, メモリオブジェクトの境界を越えている場合や, タスクの所属する保護ドメインに対して書込みアクセスまたは読出しアクセスが許可されていないメモリオブジェクトに含まれる場合には，E_PARエラーとすることができる．

stkにNULL（＝0）が指定された場合には，stkszで指定された以上のサイズのメモリ領域を，タスクと同じ保護ドメインに所属し，すべての操作／アクセスがタスクの所属保護ドメインのみに許可されているメモリ領域の中にカーネルが確保する．

ユーザドメインに所属するタスクは，sstkで指定された番地からsstkszバイトのメモリ領域を，システムスタック領域として使用する．システムスタック領域として使用するメモリ領域が, メモリオブジェクトの境界を越えている場合や, 何らかの操作／アクセスがカーネルドメイン以外にも許可されているメモリオブジェクトに含まれる場合には，E_PARエラーを返す．

ユーザドメインに所属するタスクに対してsstkにNULL（＝0）が指定された場合には, sstkszで指定された以上のサイズのメモリ領域を, すべての操作／アクセスがカーネルドメインのみに許可されているメモリ領域の中に, カーネルが確保する．

ユーザドメイン以外の保護ドメインに所属するタスクや, ユーザドメインに所

属するタスクのシステムスタック領域を分離しない実装においては, sstkszとsstkは無効である. ただし実装定義で, stkにNULLが指定された場合にスタック領域として確保するメモリ領域のサイズの決定に, sstkszに指定された値を使うことができる.

静的APIにおいては, sstkszとsstkは省略することができる. 省略された場合には, 実装定義のデフォルト値を使う.

stkszまたはsstkszに, 実装定義の最小値よりも小さい値が指定された場合や, 実装定義の最大値よりも大きい値が指定された場合には, E_PAR エラーを返す.

【補足説明】

これらの静的 API を, 保護ドメインの囲みの外側に記述した場合には, コンフィギュレータがエラーを報告する. また, これらのサービスコールは, タスク属性にTA_DOM (TDOM_NONE)が指定された場合には, E_RSATRエラーを返す.

システムスタック領域のサイズをタスク毎に指定する必要があるのは, タスク毎に呼び出す拡張サービスコールが異なるためである. 拡張サービスコールをサポートしない場合や, すべてのタスクが同じ拡張サービスコールを呼び出すことを想定する場合には, システムスタック領域のサイズをシステム全体で共通にする方法も考えられる.

sac_tsk　タスクのアクセス許可ベクタの変更

【C言語API】

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

【パラメータ】

ID	tskid	変更対象のタスクのID番号
ACVCT	acvct	タスクのアクセス許可ベクタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

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

【機能】

tskidで指定されるタスクに対するアクセス許可ベクタを, acvctで指定されるアクセス許可ベクタに設定する.

ref_tsk　タスクの状態参照

【C言語API】

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

【パラメータ】

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

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rtskの内容（T_RTSK型）

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

（実装独自に他の情報を追加してもよい）

【エラーコード】

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

【機能】

tskidで指定されるタスクに関する状態を参照し, pk_rtskで指定されるパケットに返す．μITRON4.0仕様との違いは次の通り．

対象タスクが待ち状態（二重待ち状態を含む）の場合に，tskwaitに返す値に次の2つを追加する．

TTW_MPP	0x10000	保護メモリブロックの獲得待ち状態
TTW_MBP	0x20000	保護メールボックスからの受信待ち状態

acvctには，対象タスクのアクセス許可ベクタを返す．

【補足説明】

TTW_MPPとTTW_MBPは16ビットで表現することができない．そのため, 保護メモリプール機能と保護メールボックス機能を持ち，ref_tsk（またはref_tst）をサポートするカーネルでは，STATは32ビット以上の整数型に定義

する必要がある.

ref_tstはtskwaitにref_tskで返すのと同じ値を返すため, ref_tstがtskwaitに返す値にも，TTW_MPPとTTW_MBPを追加することになる.

3.2 タスク付属同期機能

タスクに付属する新たな状態として，待ち禁止状態を導入する．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をいう．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移行しようとした場合，サービスコールはE_RLWAIエラーを返す．

タスクは，実行状態，実行可能状態，強制待ち状態（二重待ち状態を除く）のいずれかの状態にある場合に，それらと重複して待ち禁止状態になることができる．タスクの待ち状態を強制解除するサービスコール（rel_wai，irel_wai）は，対象タスクを強制的に待ち解除することに加えて，対象タスクを待ち禁止状態にする．一方，次にそのタスク自身のコンテキスト（CPU例外ハンドラや拡張サービスコールルーチンのコンテキストを除く）を実行する時点で，タスクの待ち禁止状態を解除する．

タスクの起動時には，待ち禁止状態を解除した状態に初期化する．すなわち，待ち禁止状態の解除を，タスクの起動時に行うべき処理に追加する．

【補足説明】

待ち禁止状態を導入した理由については，3.3節の補足説明を参照すること．

rel_wai　　待ち状態の強制解除
irel_wai

【C言語API】

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

【パラメータ】

ID	tskid	待ち状態の強制解除対象のタスクのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

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

【機能】

tskidで指定されるタスクを，強制的に待ち解除し，待ち禁止状態に移行させる．強制的に待ち解除するとは，対象タスクが待ち状態の時は実行可能状態に，二重待ち状態の時は強制待ち状態に移行させることをいう．このサービスコールにより待ち解除されたタスクに対しては，待ち状態に入ったサービスコールの返値としてE_RLWAIエラーを返す．

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

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

【補足説明】

これらのサービスコールは，対象タスクが待ち状態でない場合でもE_OKを返す点で，µITRON4.0仕様との互換性がない．対象タスクが待ち状態でない場合にはE_OBJを返すとする仕様も考えられるが，たとえそのような仕様にしても，待ち禁止状態に移行させるとした時点ですでにµITRON4.0仕様との完全な互換性は失われており（E_OBJエラーを返した場合でも，後でタスクを待ち状態に入らせないという効果があるため），サービスコールがエラーを返した場合には副作用がないという原則を守る方が重要と考えたためである．

tskidに自タスクを指定する呼出しは，タスク自身のコンテキストで行っても何の効果もないため，拡張サービスコールルーチンやタスクコンテキストで実行されるCPU例外ハンドラの中でしか意味がない．

3.3 タスク例外処理機能

タスク例外処理ルーチンを起動する条件を,「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」「CPU ロック状態でない」「タスク自身のコンテキスト（CPU 例外ハンドラや拡張サービスコールルーチンのコンテキストを除く）が実行されている」の4つの条件が揃った場合に変更する.

この変更により, 拡張サービスコールルーチンの実行中はタスク例外処理ルーチンを起動しないこととなったため,「実装定義で，拡張サービスコールルーチンの起動によりタスク例外処理を禁止し, そこからのリターンにより起動前の状態に戻すことができる」という仕様は廃止する.

タスク例外処理ルーチンは, タスクが所属するのと同じ保護ドメインに所属する．タスク例外処理ルーチンを定義する静的APIを，タスクと同じ保護ドメインの囲みの内側以外に記述した場合には, コンフィギュレータがエラーを報告する．また，タスク例外処理ルーチンを定義するサービスコールでは，オブジェクト属性により所属保護ドメインを設定することはできない.

【補足説明】

μITRON4.0仕様におけるタスク例外処理ルーチンの4つの起動条件の内,「タスクが実行状態である」「非タスクコンテキストまたはCPU例外ハンドラが実行されていない」の 2 つの条件を,「タスク自身のコンテキスト（CPU 例外ハンドラや拡張サービスコールのコンテキストを除く）が実行されている」という条件に置き換えた．また,「CPUロック状態でない」という条件を追加した.

起動条件を置き換えたことにより, 拡張サービスコールルーチンの実行中はタスク例外処理ルーチンを起動しないこととなった．これは，拡張サービスコールルーチンの実行中にタスク例外処理ルーチンを起動すると, 特権モードでの処理を一時中断した状態で, タスク例外処理ルーチンを非特権モードで処理を実行することになり, タスク例外処理ルーチンからの大域脱出ができなくなるためである．ITRON2 仕様のように，拡張サービスコールルーチンに対する例外処理ルーチンを導入する方法も考えられるが, 仕様が複雑化するために採用しなかった.

タスクが待ち状態にある場合でも, すぐにタスク例外処理ルーチンを実行することを要求するために，ras_tex（またはiras_tex）に加えてrel_wai（またはirel_wai）も発行する方法がある．ところが，起動条件を置き換えたことにより, 拡張サービスコールルーチンの実行中のタスクに対してras_texとrel_waiを発行した後に，タスクが拡張サービスコールルーチン内で待ち状態に入ると，タスク例外処理ルーチンがすぐに起動されない結果となる．この問題を解決するために，3.2節で説明した待ち禁止状態を導入した.

また, 起動条件を置き換えたことにより，拡張サービスコールルーチンの中でCPUロック状態に移行し，そのままリターンした場合，CPUロック状態で他の3つの条件が揃うことになる．この時の振舞いを規定するために,「CPUロック状態でない」という条件を追加した.

タスク例外処理ルーチンの起動条件の変更は, 拡張サービスコール機能の明確化によるものであり, 拡張サービスコールをサポートしない場合の振舞いは変更されていない．すなわち，拡張サービスコールをサポートしない範囲では，μITRON4.0仕様と互換になっているということができる．ただし，前節で述べた通り，rel_wai と irel_wai についてはμITRON4.0仕様との互換性をあきらめることとした.

3.4 同期・通信機能

3.4.1 セマフォ

セマフォ機能には，アクセス許可ベクタを指定してセマフォを生成する機能と，セマフォのアクセス許可ベクタを変更する機能を追加する．また，参照できるセマフォ状態に追加がある．

セマフォ状態のパケット形式に，アクセス許可ベクタを追加する．

```
typedef struct t_rsem {
        ID          wtskid ;    /* セマフォの待ち行列の先頭のタスク
                                   のID番号 */
        UINT        semcnt ;    /* セマフォの現在の資源数 */
        ACVCT       acvct ;     /* セマフォのアクセス許可ベクタ */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RSEM ;
```

セマフォ機能の新規サービスコールの機能コードは次の通りである．

TFN_CRA_SEM	–0x112	cra_semの機能コード
TFN_ACRA_SEM	–0x122	acra_semの機能コード
TFN_SAC_SEM	–0x132	sac_semの機能コード

CRA_SEM セマフォの生成（静的API，アクセス許可指定）
cra_sem セマフォの生成（アクセス許可指定）
acra_sem セマフォの生成（ID番号自動割付け，アクセス許可指定）

【静的API】

```
CRA_SEM ( ID semid, { ATR sematr, UINT isemcnt,
                      UINT maxsem }, ACVCT acvct ) ;
```

【C言語API】

```
ER ercd = cra_sem ( ID semid, T_CSEM *pk_csem,
                    ACVCT *p_acvct ) ;
ER_ID semid = acra_sem ( T_CSEM *pk_csem,
                    ACVCT *p_acvct ) ;
```

【パラメータ】

ID	semid	生成対象のセマフォのID番号
T_CSEM *	pk_csem	セマフォ生成情報を入れたパケットへのポインタ（静的APIではパケットの内容を直接記述する）
ACVCT	acvct	セマフォのアクセス許可ベクタ

pk_csemの内容（T_CSEM型）

ATR	sematr	セマフォ属性
UINT	isemcnt	セマフォの資源数の初期値
UINT	maxsem	セマフォの最大資源数

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cra_semの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acra_semの場合

ER_ID	semid	生成したセマフォのID番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（semidが不正あるいは使用できない）
E_NOID	ID番号不足（割付け可能なセマフォIDがない）
E_RSATR	予約属性（sematrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_csem，isemcnt，maxsem，p_acvct，acvctが不正）
E_OBJ	オブジェクト状態エラー（対象セマフォが登録済み）

【機能】

semidで指定されるID番号を持つセマフォを, pk_csemで指定されるセマフォ生成情報に基づいて生成する．CRE_SEM ／ cre_sem ／ acre_sem との違いは，生成するセマフォに，acvct で指定されるアクセス許可ベクタを設定することのみである．

sac_sem　セマフォのアクセス許可ベクタの変更

【C言語API】

ER ercd = sac_sem (ID semid, ACVCT *p_acvct) ;

【パラメータ】

ID	semid	変更対象のセマフォのID番号
ACVCT	acvct	セマフォのアクセス許可ベクタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（semidが不正あるいは使用できない）
E_PAR	パラメータエラー（p_acvct，acvctが不正）
E_NOEXS	オブジェクト未生成（対象セマフォが未登録）

【機能】

semidで指定されるセマフォに対するアクセス許可ベクタを，acvctで指定されるアクセス許可ベクタに設定する．

【補足説明】

このサービスコールは，呼び出された時点ですでにセマフォの待ち行列につながれているタスクには影響しない．そのため，セマフォの待ち行列につながれているタスクが，新たに設定したアクセス許可ベクタではセマフォ資源を獲得することを許可されていない場合でも，そのタスクはセマフォ資源の獲得待ち状態のままとなり，セマフォ資源が返却されれば，セマフォ資源を獲得することができる．

ref_sem　セマフォの状態参照

【C言語API】

ER ercd = ref_sem (ID semid, T_RSEM *pk_rsem) ;

【パラメータ】

ID	semid	状態参照対象のセマフォのID番号
T_RSEM *	pk_rsem	セマフォ状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rsemの内容（T_RSEM型）

ID	wtskid	セマフォの待ち行列の先頭のタスクのID番号
UINT	semcnt	セマフォの現在の資源数
ACVCT	acvct	セマフォのアクセス許可ベクタ

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（semidが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_rsemが不正）
E_NOEXS	オブジェクト未生成（対象セマフォが未登録）

【機能】

semidで指定されるセマフォに関する状態を参照し, pk_rsemで指定されるパケットに返す．µITRON4.0仕様との違いは，acvctに対象セマフォのアクセス許可ベクタを返すことのみである．

3.4.2 イベントフラグ

イベントフラグ機能には, アクセス許可ベクタを指定してイベントフラグを生成する機能と, イベントフラグのアクセス許可ベクタを変更する機能を追加する．また，参照できるイベントフラグ状態に追加がある．

イベントフラグ状態のパケット形式に，アクセス許可ベクタを追加する．

```
typedef struct t_rflg {
        ID          wtskid ;    /* イベントフラグの待ち行列の先頭の
                                   タスクのID番号 */
        FLGPTN      flgptn ;    /* イベントフラグの現在のビットパ
                                   ターン */
        ACVCT       acvct ;     /* イベントフラグのアクセス許可ベク
                                   タ */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RFLG ;
```

イベントフラグ機能の新規サービスコールの機能コードは次の通りである．

TFN_CRA_FLG	–0x113	cra_flgの機能コード
TFN_ACRA_FLG	–0x123	acra_flgの機能コード
TFN_SAC_FLG	–0x133	sac_flgの機能コード

CRA_FLG イベントフラグの生成（静的API，アクセス許可指定）
cra_flg イベントフラグの生成（アクセス許可指定）
acra_flg イベントフラグの生成（ID番号自動割付け，アクセス許可指定）

【静的API】

```
CRA_FLG ( ID flgid, { ATR flgatr, FLGPTN iflgptn },
                                ACVCT acvct ) ;
```

【C言語API】

```
ER ercd = cra_flg ( ID flgid, T_CFLG *pk_cflg, ACVCT *p_acvct ) ;
ER_ID flgid = acra_flg ( T_CFLG *pk_cflg, ACVCT *p_acvct ) ;
```

【パラメータ】

ID	flgid	生成対象のイベントフラグのID番号
T_CFLG *	pk_cflg	イベントフラグ生成情報を入れたパケットへのポインタ（静的APIではパケットの内容を直接記述する）
ACVCT	acvct	イベントフラグのアクセス許可ベクタ

pk_cflgの内容（T_CFLG型）

ATR	flgatr	イベントフラグ属性
FLGPTN	iflgptn	イベントフラグのビットパターンの初期値

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cra_flgの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acra_flgの場合

ER_ID	flgid	生成したイベントフラグのID番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（flgidが不正あるいは使用できない）
E_NOID	ID番号不足（割付け可能なイベントフラグIDがない）
E_RSATR	予約属性（flgatrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_cflg，iflgptn，p_acvct，acvctが不正）
E_OBJ	オブジェクト状態エラー（対象イベントフラグが登録済み）

【機能】

flgidで指定されるID番号を持つイベントフラグを，pk_cflgで指定されるイベントフラグ生成情報に基づいて生成する．CRE_FLG／cre_flg／acre_flgとの

違いは，生成するイベントフラグに，acvctで指定されるアクセス許可ベクタを設定することのみである．

sac_flg　　イベントフラグのアクセス許可ベクタの変更

【C言語API】

```
ER ercd = sac_flg ( ID flgid, ACVCT *p_acvct ) ;
```

【パラメータ】

ID	flgid	変更対象のイベントフラグのID番号
ACVCT	acvct	イベントフラグのアクセス許可ベクタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（flgidが不正あるいは使用できない）
E_PAR	パラメータエラー（p_acvct，acvctが不正）
E_NOEXS	オブジェクト未生成（対象イベントフラグが未登録）

【機能】

flgidで指定されるイベントフラグに対するアクセス許可ベクタを，acvctで指定されるアクセス許可ベクタに設定する．

【補足説明】

このサービスコールは，呼び出された時点ですでにイベントフラグの待ち行列につながれているタスクには影響しない．そのため，イベントフラグの待ち行列につながれているタスクが，新たに設定したアクセス許可ベクタではイベントフラグを待つことを許可されていない場合でも，そのタスクはイベントフラグ待ち状態のままとなり，待っているイベントフラグがセットされれば，正常に待ち解除される．

ref_flg　　イベントフラグの状態参照

【C言語API】

ER ercd = ref_flg (ID flgid, T_RFLG *pk_rflg) ;

【パラメータ】

ID	flgid	状態参照対象のイベントフラグのID番号
T_RFLG *	pk_rflg	イベントフラグ状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rflgの内容（T_RFLG型）

ID	wtskid	イベントフラグの待ち行列の先頭のタスクのID番号
FLGPTN	flgptn	イベントフラグの現在のビットパターン
ACVCT	acvct	イベントフラグのアクセス許可ベクタ

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（flgidが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_rflgが不正）
E_NOEXS	オブジェクト未生成（対象イベントフラグが未登録）

【機能】

flgidで指定されるイベントフラグに関する状態を参照し，pk_rflgで指定されるパケットに返す．µITRON4.0仕様との違いは，acvctに対象イベントフラグのアクセス許可ベクタを返すことのみである．

3.4.3 データキュー

データキュー機能には, アクセス許可ベクタを指定してデータキューを生成する機能と，データキューのアクセス許可ベクタを変更する機能を追加する．また，参照できるデータキュー状態に追加がある．

カーネルの用いる管理領域であることを明確にするために, データキュー領域をデータキュー管理領域に改名する．それに伴い，データキュー生成情報のパケット形式中のフィールドと，カーネル構成マクロを改名する．

データキュー機能に関連するカーネル構成マクロTSZ_DTQは, TSZ_DTQMBと改名し，次のように定義する．

SIZE dtqmbsz = TSZ_DTQMB (UINT dtqcnt)

dtqcnt 個のデータを格納するのに必要なデータキュー管理領域のサイズ（バイト数）

データキュー生成情報のパケット形式は，次の通りに変更する．また，データキュー状態のパケット形式に，アクセス許可ベクタを追加する．

```
typedef struct t_cdtq {
        ATR             dtqatr ;        /* データキュー属性 */
        UINT            dtqcnt ;        /* データキューの容量（データの個
                                           数）*/
        VP              dtqmb ;         /* データキュー管理領域の先頭番
                                           地 */
        VP              dtq ;           /* データキュー領域の先頭番地 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_CDTQ ;

typedef struct t_rdtq {
        ID              stskid ;        /* データキューの送信待ち行列の先頭
                                           のタスクのID番号 */
        ID              rtskid ;        /* データキューの受信待ち行列の先頭
                                           のタスクのID番号 */
        UINT            sdtqcnt ;       /* データキューに入っているデータの
                                           数 */
        ACVCT           acvct ;         /* データキューのアクセス許可ベク
                                           タ */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RDTQ ;
```

データキュー機能の新規サービスコールの機能コードは次の通りである．

TFN_CRA_DTQ	–0x114	cra_dtqの機能コード
TFN_ACRA_DTQ	–0x124	acra_dtqの機能コード
TFN_SAC_DTQ	–0x134	sac_dtqの機能コード

【補足説明】

μITRON4.0仕様との互換性を重視する場合には，TSZ_DTQを，TSZ_DTQMBと同じに定義しておくとよい．

CRE_DTQ	データキューの生成（静的API）
CRA_DTQ	データキューの生成（静的API，アクセス許可指定）
cre_dtq	データキューの生成
cra_dtq	データキューの生成（アクセス許可指定）
acre_dtq	データキューの生成（ID番号自動割付け）
acra_dtq	データキューの生成（ID番号自動割付け，アクセス許可指定）

【静的API】

```
CRE_DTQ ( ID dtqid, { ATR dtqatr, UINT dtqcnt, VP dtqmb } ) ;
CRA_DTQ ( ID dtqid, { ATR dtqatr, UINT dtqcnt, VP dtqmb },
                        ACVCT acvct ) ;
```

【C言語API】

```
ER ercd = cre_dtq ( ID dtqid, T_CDTQ *pk_cdtq ) ;
ER ercd = cra_dtq ( ID dtqid, T_CDTQ *pk_cdtq,
                        ACVCT *p_acvct ) ;
ER_ID dtqid = acre_dtq ( T_CDTQ *pk_cdtq ) ;
ER_ID dtqid = acra_dtq ( T_CDTQ *pk_cdtq, ACVCT *p_acvct ) ;
```

【パラメータ】

ID	dtqid	生成対象のデータキューのID番号
T_CDTQ *	pk_cdtq	データキュー生成情報を入れたパケットへのポインタ（静的APIではパケットの内容を直接記述する）
ACVCT	acvct	データキューのアクセス許可ベクタ

pk_cdtqの内容（T_CDTQ型）

ATR	dtqatr	データキュー属性
UINT	dtqcnt	データキューの容量（データの個数）
VP	dtqmb	データキュー管理領域の先頭番地

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cre_dtq，cra_dtqの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acre_dtq，acra_dtqの場合

ER_ID	dtqid	生成したデータキューのID番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（dtqidが不正あるいは使用できない）
E_NOID	ID番号不足（割付け可能なデータキューIDがない）
E_NOMEM	メモリ不足（データキュー管理領域などが確保できな

	い）
E_RSATR	予約属性（dtqatrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_cdtq, dtqcnt, dtqmb, p_acvct, acvctが不正）
E_OBJ	オブジェクト状態エラー（対象データキューが登録済み）

【機能】

dtqidで指定されるID番号を持つデータキューを，pk_cdtqで指定されるデータキュー生成情報に基づいて生成する．μITRON4.0仕様との違いは次の通り．

dtqcntはデータキューに格納できるデータの個数，dtqmbはデータキュー管理領域の先頭番地である．

dtqmbで指定された番地から，dtqcnt個のデータを格納するのに必要なサイズのメモリ領域を，データキュー管理領域として使用する．アプリケーションプログラムは，TSZ_DTQMBを用いて，必要なデータキュー管理領域のサイズを知ることができる．データキュー管理領域として使用するメモリ領域が，メモリオブジェクトの境界を越えている場合や，何らかの操作／アクセスがカーネルドメイン以外にも許可されているメモリオブジェクトに含まれる場合には，E_PARエラーを返す．

dtqmbにNULL（＝0）が指定された場合には，必要なサイズのメモリ領域を，すべての操作／アクセスがカーネルドメインのみに許可されているメモリ領域の中に，カーネルが確保する．

sac_dtq　データキューのアクセス許可ベクタの変更

【C言語API】

```
ER ercd = sac_dtq ( ID dtqid, ACVCT *p_acvct ) ;
```

【パラメータ】

ID	dtqid	変更対象のデータキューのID番号
ACVCT	acvct	データキューのアクセス許可ベクタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（dtqidが不正あるいは使用できない）
E_PAR	パラメータエラー（p_acvct，acvctが不正）
E_NOEXS	オブジェクト未生成（対象データキューが未登録）

【機能】

dtqidで指定されるデータキューに対するアクセス許可ベクタを，acvctで指定されるアクセス許可ベクタに設定する．

【補足説明】

このサービスコールは，呼び出された時点ですでにデータキューの送信待ち行列につながれているタスクには影響しない．そのため，データキューの送信待ち行列につながれているタスクが，新たに設定したアクセス許可ベクタではデータキューへ送信することを許可されていない場合でも，そのタスクはデータキューへの送信待ち状態のままとなり，データキュー管理領域に空きができれば，データキューへ送信することができる．データキューの受信待ち行列につながれているタスクについても同様である．

ref_dtq　　データキューの状態参照

【C言語API】

ER ercd = ref_dtq (ID dtqid, T_RDTQ *pk_rdtq) ;

【パラメータ】

ID	dtqid	状態参照対象のデータキューのID番号
T_RDTQ *	pk_rdtq	データキュー状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rdtqの内容（T_RDTQ型）

ID	stskid	データキューの送信待ち行列の先頭のタスクのID番号
ID	rtskid	データキューの受信待ち行列の先頭のタスクのID番号
UINT	sdtqcnt	データキューに入っているデータの数
ACVCT	acvct	データキューのアクセス許可ベクタ

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（dtqidが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_rdtqが不正）
E_NOEXS	オブジェクト未生成（対象データキューが未登録）

【機能】

dtqidで指定されるデータキューに関する状態を参照し，pk_rdtqで指定されるパケットに返す．μITRON4.0仕様との違いは，acvctに対象データキューのアクセス許可ベクタを返すことのみである．

3.4.4 メールボックス

アプリケーションプログラムからカーネルの用いる管理領域を保護するために，メールボックス機能にはいくつかの変更がある．また，メールボックス機能には，アクセス許可ベクタを指定してメールボックスを生成する機能と，メールボックスのアクセス許可ベクタを変更する機能を追加する．参照できるメールボックス状態にも変更がある．

µITRON4.0 仕様では，メッセージの先頭に置かれたメッセージヘッダを用いて，メッセージキューを作る仕様としている．この仕様では，アプリケーションからカーネルを保護するためにこの仕様を改め，メッセージキューを作るためのメモリ領域を別に確保する．このメモリ領域は，µITRON4.0仕様でも必要であった優先度別のメッセージキューヘッダ領域と統合して，メールボックス管理領域と呼ぶ．メールボックス管理領域のサイズを決定するために，メールボックスに入れることができるメッセージの数を，メールボックスの生成時に指定することとする．

メッセージヘッダを用いない実装では，メッセージヘッダのデータ型である T_MSG は必要ないが，メッセージヘッダを用いる実装との互換性のために，何らかの定義を与えておくものとする．優先度付きメッセージヘッダのデータ型である T_MSG_PRI は，µITRON4.0仕様と同様に定義してもよいし，次のようにメッセージヘッダを除いたデータ型に定義してもよい．

```
typedef struct t_msg_pri {
        PRI             msgpri ;        /* メッセージ優先度 */
} T_MSG_PRI ;
```

メールボックス機能に関連して，次のカーネル構成マクロを定義する．

SIZE mbxmbsz = TSZ_MBXMB (UINT mbxcnt, PRI maxmpri)

mbxcnt 個のメッセージを入れることができ，送信されるメッセージの優先度の最大値が maxmpri であるメールボックスに必要なメールボックス管理領域のサイズ（バイト数）

µITRON4.0仕様で用意されていたカーネル構成マクロ TSZ_MPRIHD はサポートしない．

メールボックス生成情報のパケット形式は，次の通りに変更する．また，メールボックス状態のパケット形式に，アクセス許可ベクタを追加する．

```
typedef struct t_cmbx {
        ATR             mbxatr ;        /* メールボックス属性 */
        UINT            mbxcnt ;        /* 入れることができるメッセージの
                                           数 */
        PRI             maxmpri ;       /* 送信されるメッセージの優先度の最
                                           大値 */
        VP              mbxmb ;         /* メールボックス管理領域の先頭番
                                           地 */
        /* 実装独自に他のフィールドを追加してもよい */
```

```
} T_CMBX ;
typedef struct t_rmbx {
        ID          wtskid ;    /* メールボックスの待ち行列の先頭の
                                   タスクのID番号 */
        T_MSG *     pk_msg ;    /* メッセージキューの先頭のメッセー
                                   ジパケットの先頭番地 */
        ACVCT       acvct ;     /* メールボックスのアクセス許可ベク
                                   タ */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RMBX ;
```

メールボックス管理機能の新規サービスコールの機能コードは次の通りである．

TFN_CRA_MBX	–0x115	cra_mbxの機能コード
TFN_ACRA_MBX	–0x125	acra_mbxの機能コード
TFN_SAC_MBX	–0x135	sac_mbxの機能コード

【補足説明】

サービスコールのAPIへの変更を最小限にするために，μITRON4.0仕様と同様，メッセージ優先度はメッセージヘッダの中に置くこととする．

メールボックス機能は，メッセージの先頭番地のみを伝える機能である．メールボックスに送信するメッセージを，受信するタスクがアクセスできる（少なくとも，読み出せる）メモリ領域に置くことは，アプリケーションの責任である．アクセスできるメモリ領域に置かなかった場合には，先頭番地が伝わっても内容が読めないことになる．同様の議論は，データキューでメッセージの先頭番地を渡す場合にも適用される．

メールボックス生成時のパラメータが変更になったことにより，μITRON4.0仕様に合致したシステムコンフィギュレーションファイルそのままでは使えなくなるが，メールボックスの仕様が変更になったため，やむをえないと考えられる．むしろ，コンフィギュレータでエラーが検出されることで，メールボックスの仕様が変更になったという注意を促す効果が期待できる．

CRE_MBX	メールボックスの生成（静的API）
CRA_MBX	メールボックスの生成（静的API，アクセス許可指定）
cre_mbx	メールボックスの生成
cra_mbx	メールボックスの生成（アクセス許可指定）
acre_mbx	メールボックスの生成（ID番号自動割付け）
acra_mbx	メールボックスの生成（ID番号自動割付け，アクセス許可指定）

【静的API】

```
CRE_MBX ( ID mbxid, { ATR mbxatr, UINT mbxcnt,
                      PRI maxmpri, VP mbxmb } ) ;
CRA_MBX ( ID mbxid, { ATR mbxatr, UINT mbxcnt, PRI maxmpri,
                      VP mbxmb }, ACVCT acvct ) ;
```

【C言語API】

```
ER ercd = cre_mbx ( ID mbxid, T_CMBX *pk_cmbx ) ;
ER ercd = cra_mbx ( ID mbxid, T_CMBX *pk_cmbx,
                    ACVCT *p_acvct ) ;
ER_ID mbxid = acre_mbx ( T_CMBX *pk_cmbx ) ;
ER_ID mbxid = acra_mbx ( T_CMBX *pk_cmbx,
                    ACVCT *p_acvct ) ;
```

【パラメータ】

ID	mbxid	生成対象のメールボックスのID番号
T_CMBX *	pk_cmbx	メールボックス生成情報を入れたパケットへのポインタ（静的APIではパケットの内容を直接記述する）
ACVCT	acvct	メールボックスのアクセス許可ベクタ

pk_cmbxの内容（T_CMBX型）

ATR	mbxatr	メールボックス属性
UINT	mbxcnt	入れることができるメッセージの数
PRI	maxmpri	送信されるメッセージの優先度の最大値
VP	mbxmb	メールボックス管理領域の先頭番地

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cre_mbx，cra_mbxの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acre_mbx，acra_mbxの場合

ER_ID	mbxid	生成したメールボックスのID番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mbxidが不正あるいは使用できない）
E_NOID	ID番号不足（割付け可能なメールボックスIDがない）
E_NOMEM	メモリ不足（メールボックス管理領域などが確保できない）
E_RSATR	予約属性（mbxatrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_cmbx，mbxcnt，mbxmpri，mbxmb，p_acvct，acvctが不正）
E_OBJ	オブジェクト状態エラー（対象メールボックスが登録済み）

【機能】

mbxidで指定されるID番号を持つメールボックスを，pk_cmbxで指定されるメールボックス生成情報に基づいて生成する．mbxatrはメールボックスの属性，mbxcnt はメールボックスに入れることができるメッセージの数，maxmpri はメールボックスに送信されるメッセージの優先度の最大値，mbxmbはメールボックス管理領域の先頭番地である．maxmpriは，mbxatrにTA_MPRI（＝0x02）が指定された場合にのみ有効である．

静的APIにおいては，mbxidは自動割付け対応整数値パラメータ，mbxatrとmaxmpriはプリプロセッサ定数式パラメータである．

acre_mbx および acra_mbx は，生成するメールボックスの ID 番号をメールボックスが登録されていないID番号の中から割り付け，割り付けたID番号を返値として返す．

mbxatrには，（（TA_TFIFO ‖ TA_TPRI）｜（TA_MFIFO ‖ TA_MPRI））の指定ができる．メールボックスの待ち行列は，TA_TFIFO（＝0x00）が指定された場合にはFIFO順，TA_TPRI（＝ 0x01）が指定された場合にはタスクの優先度順となる．また，メールボックスのメッセージキューは，TA_MFIFO（＝0x00）が指定された場合にはFIFO順，TA_MPRI（＝ 0x02）が指定された場合にはメッセージの優先度順となる．

mbxmbで指定された番地から，mbxcnt個のメッセージを入れることができ，送信されるメッセージの優先度の最大値がmaxmpriの場合に必要なサイズのメモリ領域を，メールボックス管理領域として使用する．アプリケーションプログラムは，TSZ_MBXMB を用いて，必要なメールボックス管理領域のサイズを知ることができる．メールボックス管理領域として使用するメモリ領域が，メモリオブジェクトの境界を越えている場合や，何らかの操作／アクセスがカーネルドメイン以外にも許可されているメモリオブジェクトに含まれる場合には，E_PARエラーを返す．

mbxmbにNULL（＝0）が指定された場合には，必要なサイズのメモリ領域を，すべての操作／アクセスがカーネルドメインのみに許可されているメモリ領域の中に，カーネルが確保する．

mbxcntに0が指定された場合や，実装定義の最大値よりも大きい値が指定され

た場合には，E_PARエラーを返す．ただし，実装独自にmbxcntに0を指定できるように拡張することは許される．また，mbxatrにTA_MPRI（＝0x02）が指定された場合で，maxmpriに0が指定された場合や，メッセージ優先度の最大値（TMAX_MPRI）よりも大きい値が指定された場合にも，E_PAR エラーを返す．

sac_mbx　メールボックスのアクセス許可ベクタの変更

【C言語API】

ER ercd = sac_mbx (ID mbxid, ACVCT *p_acvct) ;

【パラメータ】

ID	mbxid	変更対象のメールボックスのID番号
ACVCT	acvct	メールボックスのアクセス許可ベクタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mbxidが不正あるいは使用できない）
E_PAR	パラメータエラー（p_acvct，acvctが不正）
E_NOEXS	オブジェクト未生成（対象メールボックスが未登録）

【機能】

mbxidで指定されるメールボックスに対するアクセス許可ベクタを，acvctで指定されるアクセス許可ベクタに設定する．

【補足説明】

このサービスコールは，呼び出された時点ですでにメールボックスの待ち行列につながれているタスクには影響しない．そのため，メールボックスの待ち行列につながれているタスクが，新たに設定したアクセス許可ベクタではメールボックスから受信することを許可されていない場合でも，そのタスクはメールボックスからの受信待ち状態のままとなり，メールボックスへメッセージが送信されれば，メッセージを受信することができる．

snd_mbx　メールボックスへの送信

【C言語API】

```
ER ercd = snd_mbx ( ID mbxid, T_MSG *pk_msg ) ;
```

【パラメータ】

ID	mbxid	送信対象のメールボックスのID番号
T_MSG *	pk_msg	メールボックスへ送信するメッセージパケットの先頭番地

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mbxidが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_msgが不正，メッセージパケット中のメッセージ優先度（msgpri）が不正）
E_MACV	メモリアクセス権違反（送信するメッセージのメッセージヘッダに対する読出しアクセスが許可されていない）
E_NOEXS	オブジェクト未生成（対象メールボックスが未登録）
E_QOVR	キューイングオーバフロー（メールボックスに入れることができるメッセージ数を越える送信）

【機能】

mbxidで指定されるメールボックスに，pk_msgを先頭番地とするメッセージを送信する．µITRON4.0仕様との違いは，メールボックスに入れることができるメッセージ数を越える送信の場合にE_QOVRエラーを返すことと，送信するメッセージのメッセージヘッダに対する読出しアクセスが許可されていない場合にE_MACVエラーを返すことである．

具体的には，メールボックスに入れることができる最大数のメッセージが入っており，それ以上のメッセージを入れることができない場合には，E_QOVRエラーを返す．また，サービスコールを呼び出した処理単位の所属する保護ドメインから，送信するメッセージの先頭番地に対する読出しアクセスが許可されていない場合に，E_MACVエラーを返す．それに加えて，メールボックス属性にTA_MPRIが指定されている場合には，メッセージパケット中のメッセージ優先度（msgpri）に対する読出しアクセスが許可されていない場合にも，E_MACVエラーを返す．

ref_mbx　メールボックスの状態参照

【C言語API】

ER ercd = ref_mbx (ID mbxid, T_RMBX *pk_rmbx) ;

【パラメータ】

ID	mbxid	状態参照対象のメールボックスのID番号
T_RMBX *	pk_rmbx	メールボックス状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rmbxの内容（T_RMBX型）

ID	wtskid	メールボックスの待ち行列の先頭のタスクのID番号
T_MSG *	pk_msg	メッセージキューの先頭のメッセージパケットの先頭番地
ACVCT	acvct	メールボックスのアクセス許可ベクタ

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（mbxidが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_rmbxが不正）
E_NOEXS	オブジェクト未生成（対象メールボックスが未登録）

【機能】

mbxidで指定されるメールボックスに関する状態を参照し，pk_rmbxで指定されるパケットに返す．µITRON4.0仕様との違いは，acvctに対象メールボックスのアクセス許可ベクタを返すことのみである．

3.5 拡張同期・通信機能

3.5.1 ミューテックス

ミューテックス機能には, アクセス許可ベクタを指定してミューテックスを生成する機能と, ミューテックスのアクセス許可ベクタを変更する機能を追加する．また，参照できるミューテックス状態に追加がある．

ミューテックスをロック解除するサービスコール（unl_mtx）は，ミューテックスをロックしたタスクのみが呼び出すことができる．そのため，アクセス許可ベクタによる保護を行わない．

ミューテックス状態のパケット形式に，アクセス許可ベクタを追加する．

```
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 */
        ACVCT       acvct ;     /* ミューテックスのアクセス許可ベクタ */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RMTX ;
```

ミューテックス機能の新規サービスコールの機能コードは次の通りである．

```
TFN_CRA_MTX     –0x116  cra_mtxの機能コード
TFN_ACRA_MTX    –0x126  acra_mtxの機能コード
TFN_SAC_MTX     –0x136  sac_mtxの機能コード
```

CRA_MTX ミューテックスの生成（静的API，アクセス許可指定）
cra_mtx ミューテックスの生成（アクセス許可指定）
acra_mtx ミューテックスの生成（ID番号自動割付け，アクセス許可指定）

【静的API】

```
CRA_MTX ( ID mtxid, { ATR mtxatr, PRI ceilpri }, ACVCT acvct ) ;
```

【C言語API】

```
ER ercd = cra_mtx ( ID mtxid, T_CMTX *pk_cmtx,
                    ACVCT *p_acvct ) ;
ER_ID mtxid = acra_mtx ( T_CMTX *pk_cmtx, ACVCT *p_acvct ) ;
```

【パラメータ】

ID	mtxid	生成対象のミューテックスのID番号
T_CMTX *	pk_cmtx	ミューテックス生成情報を入れたパケットへのポインタ（静的APIではパケットの内容を直接記述する）
ACVCT	acvct	ミューテックスのアクセス許可ベクタ

pk_cmtxの内容（T_CMTX型）

ATR	mtxatr	ミューテックス属性
PRI	ceilpri	ミューテックスの上限優先度

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cra_mtxの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acra_mtxの場合

ER_ID	mtxid	生成したミューテックスのID番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mtxidが不正あるいは使用できない）
E_NOID	ID番号不足（割付け可能なミューテックスIDがない）
E_RSATR	予約属性（mtxatrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_cmtx，ceilpri，p_acvct，acvctが不正）
E_OBJ	オブジェクト状態エラー（対象ミューテックスが登録済み）

【機能】

mtxidで指定されるID番号を持つミューテックスを，pk_cmtxで指定されるミューテックス生成情報に基づいて生成する．CRE_MTX／cre_mtx／

acre_mtxとの違いは，生成するミューテックスに，acvctで指定されるアクセス許可ベクタを設定することのみである．

sac_mtx　ミューテックスのアクセス許可ベクタの変更

【C言語API】

ER ercd = sac_mtx (ID mtxid, ACVCT *p_acvct) ;

【パラメータ】

ID	mtxid	変更対象のミューテックスのID番号
ACVCT	acvct	ミューテックスのアクセス許可ベクタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mtxidが不正あるいは使用できない）
E_PAR	パラメータエラー（p_acvct，acvctが不正）
E_NOEXS	オブジェクト未生成（対象ミューテックスが未登録）

【機能】

mtxidで指定されるミューテックスに対するアクセス許可ベクタを，acvctで指定されるアクセス許可ベクタに設定する．

【補足説明】

このサービスコールは，呼び出された時点ですでにミューテックスの待ち行列につながれているタスクには影響しない．そのため，ミューテックスの待ち行列につながれているタスクが，新たに設定したアクセス許可ベクタではミューテックスをロックすることを許可されていない場合でも，そのタスクはミューテックスのロック待ち状態のままとなり，ミューテックスがロック解除されれば，ミューテックスをロックすることができる．

ref_mtx　　ミューテックスの状態参照

【C言語API】

ER ercd = ref_mtx (ID mtxid, T_RMTX *pk_rmtx) ;

【パラメータ】

ID	mtxid	状態参照対象のミューテックスのID番号
T_RMTX *	pk_rmtx	ミューテックス状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rmtxの内容（T_RMTX型）

ID	htskid	ミューテックスをロックしているタスクの ID番号
ID	wtskid	ミューテックスの待ち行列の先頭のタスクのID番号
ACVCT	acvct	ミューテックスのアクセス許可ベクタ

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（mtxidが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_rmtxが不正）
E_NOEXS	オブジェクト未生成（対象ミューテックスが未登録）

【機能】

mtxidで指定されるミューテックスに関する状態を参照し, pk_rmtxで指定されるパケットに返す．µITRON4.0仕様との違いは，acvctに対象ミューテックスのアクセス許可ベクタを返すことのみである．

3.5.2 メッセージバッファ

メッセージバッファ機能には, アクセス許可ベクタを指定してメッセージバッファを生成する機能と, メッセージバッファのアクセス許可ベクタを変更する機能を追加する．また，参照できるメッセージバッファ状態に追加がある．

カーネルの用いる管理領域であることを明確にするために，メッセージバッファ領域をメッセージバッファ管理領域に改名する．それに伴い，メッセージバッファ生成情報のパケット形式中のフィールドと, カーネル構成マクロを改名する．

メッセージバッファ機能に関連するカーネル構成マクロTSZ_MBFは, TSZ_MBFMBと改名し，次のように定義する．

SIZE mbfmbsz = TSZ_MBFMB (UINT msgcnt, UINT msgsz)

サイズがmsgszバイトのメッセージをmsgcnt個格納するのに必要なメッセージバッファ管理領域のサイズ（目安のバイト数）

メッセージバッファ生成情報のパケット形式は，次の通りに変更する．また，メッセージバッファ状態のパケット形式に，アクセス許可ベクタを追加する．

```
typedef struct t_cmbf {
        ATR             mbfatr ;        /* メッセージバッファ属性 */
        UINT            maxmsz ;        /* メッセージの最大サイズ（バイト
                                           数）*/
        SIZE            mbfsz ;         /* メッセージバッファ管理領域のサイ
                                           ズ（バイト数）*/
        VP              mbfmb ;         /* メッセージバッファ管理領域の先頭
                                           番地 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_CMBF ;

typedef struct t_rmbf {
        ID              stskid ;        /* メッセージバッファの送信待ち行列
                                           の先頭のタスクのID番号 */
        ID              rtskid ;        /* メッセージバッファの受信待ち行列
                                           の先頭のタスクのID番号 */
        UINT            smsgcnt ;       /* メッセージバッファに入っている
                                           メッセージの数 */
        SIZE            fmbfsz ;        /* メッセージバッファ管理領域の空き
                                           領域のサイズ（バイト数，最低限の
                                           管理領域を除く）*/
        ACVCT           acvct ;         /* メッセージバッファのアクセス許可
                                           ベクタ */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RMBF ;
```

メッセージバッファ機能の新規サービスコールの機能コードは次の通りである．

TFN_CRA_MBF –0x117 cra_mbfの機能コード

TFN_ACRA_MBF	–0x127	acra_mbfの機能コード
TFN_SAC_MBF	–0x137	sac_mbfの機能コード

【補足説明】

μITRON4.0仕様との互換性を重視する場合には，TSZ_MBFを，TSZ_MBFMBと同じに定義しておくとよい．

CRE_MBF	メッセージバッファの生成（静的API）
CRA_MBF	メッセージバッファの生成（静的API，アクセス許可指定）
cre_mbf	メッセージバッファの生成
cra_mbf	メッセージバッファの生成（アクセス許可指定）
acre_mbf	メッセージバッファの生成（ID番号自動割付け）
acra_mbf	メッセージバッファの生成（ID番号自動割付け，アクセス許可指定）

【静的API】

```
CRE_MBF ( ID mbfid, { ATR mbfatr, UINT maxmsz, SIZE mbfsz,
                      VP mbfmb } ) ;
CRA_MBF ( ID mbfid, { ATR mbfatr, UINT maxmsz, SIZE mbfsz,
                      VP mbfmb }, ACVCT acvct ) ;
```

【C言語API】

```
ER ercd = cre_mbf ( ID mbfid, T_CMBF *pk_cmbf ) ;
ER ercd = cra_mbf ( ID mbfid, T_CMBF *pk_cmbf,
                    ACVCT *p_acvct ) ;
ER_ID mbfid = acre_mbf ( T_CMBF *pk_cmbf ) ;
ER_ID mbfid = acra_mbf ( T_CMBF *pk_cmbf, ACVCT *p_acvct ) ;
```

【パラメータ】

ID	mbfid	生成対象のメッセージバッファのID番号
T_CMBF *	pk_cmbf	メッセージバッファ生成情報を入れたパケットへのポインタ（静的APIではパケットの内容を直接記述する）
ACVCT	acvct	メッセージバッファのアクセス許可ベクタ

pk_cmbfの内容（T_CMBF型）

ATR	mbfatr	メッセージバッファ属性
UINT	maxmsz	メッセージの最大サイズ（バイト数）
SIZE	mbfsz	メッセージバッファ管理領域のサイズ（バイト数）
VP	mbfmb	メッセージバッファ管理領域の先頭番地

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cre_mbf，cra_mbfの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acre_mbf，acra_mbfの場合

ER_ID	mbfid	生成したメッセージバッファの ID 番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mbfidが不正あるいは使用できない）
E_NOID	ID番号不足（割付け可能なメッセージバッファIDがない）
E_NOMEM	メモリ不足（メッセージバッファ管理領域などが確保できない）
E_RSATR	予約属性（mbfatrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_cmbf，maxmsz，mbfsz，mbfmb，p_acvct，acvctが不正）
E_OBJ	オブジェクト状態エラー（対象メッセージバッファが登録済み）

【機能】

mbfidで指定されるID番号を持つメッセージバッファを，pk_cmbfで指定されるメッセージバッファ生成情報に基づいて生成する．µITRON4.0仕様との違いは次の通り．

mbfszはメッセージバッファ管理領域のサイズ（バイト数），mbfmbはメッセージバッファ管理領域の先頭番地である．

mbfmbで指定された番地からmbfszバイトのメモリ領域を，メッセージバッファ管理領域として使用する．メッセージバッファ管理領域内には，メッセージを管理するための情報も置くため，メッセージバッファ管理領域のすべてをメッセージを格納するために使えるわけではない．アプリケーションプログラムは，TSZ_MBFMBを用いて，必要なメッセージバッファ管理領域のサイズの目安を知ることができる．メッセージバッファ管理領域として使用するメモリ領域が，メモリオブジェクトの境界を越えている場合や，何らかの操作／アクセスがカーネルドメイン以外にも許可されているメモリオブジェクトに含まれる場合には，E_PARエラーを返す．

mbfmbにNULL（＝0）が指定された場合には，mbfszで指定されたサイズのメモリ領域を，すべての操作／アクセスがカーネルドメインのみに許可されているメモリ領域の中に，カーネルが確保する．

sac_mbf　メッセージバッファのアクセス許可ベクタの変更

【C言語API】

ER ercd = sac_mbf (ID mbfid, ACVCT *p_acvct) ;

【パラメータ】

ID	mbfid	変更対象のメッセージバッファのID番号
ACVCT	acvct	メッセージバッファのアクセス許可ベクタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mbfidが不正あるいは使用できない）
E_PAR	パラメータエラー（p_acvct，acvctが不正）
E_NOEXS	オブジェクト未生成（対象メッセージバッファが未登録）

【機能】

mbfidで指定されるメッセージバッファに対するアクセス許可ベクタを，acvctで指定されるアクセス許可ベクタに設定する．

【補足説明】

このサービスコールは，呼び出された時点ですでにメッセージバッファの送信待ち行列につながれているタスクには影響しない．そのため，メッセージバッファの送信待ち行列につながれているタスクが，新たに設定したアクセス許可ベクタではメッセージバッファへ送信することを許可されていない場合でも，そのタスクはメッセージバッファへの送信待ち状態のままとなり，メッセージバッファ管理領域に空きができれば，メッセージバッファへ送信することができる．メッセージバッファの受信待ち行列につながれているタスクについても同様である．

ref_mbf　メッセージバッファの状態参照

【C言語API】

ER ercd = ref_mbf (ID mbfid, T_RMBF *pk_rmbf) ;

【パラメータ】

ID	mbfid	状態参照対象のメッセージバッファのID番号
T_RMBF *	pk_rmbf	メッセージバッファ状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rmbfの内容（T_RMBF型）

ID	stskid	メッセージバッファの送信待ち行列の先頭のタスクのID番号
ID	rtskid	メッセージバッファの受信待ち行列の先頭のタスクのID番号
UINT	smsgcnt	メッセージバッファに入っているメッセージの数
SIZE	fmbfsz	メッセージバッファ管理領域の空き領域のサイズ（バイト数，最低限の管理領域を除く）
ACVCT	acvct	メッセージバッファのアクセス許可ベクタ

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（mbfidが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_rmbfが不正）
E_NOEXS	オブジェクト未生成（対象メッセージバッファが未登録）

【機能】

mbfidで指定されるメッセージバッファに関する状態を参照し，pk_rmbfで指定されるパケットに返す．µITRON4.0仕様との違いは，acvctに対象メッセージバッファのアクセス許可ベクタを返すことのみである．

3.5.3 ランデブ

ランデブ機能には, アクセス許可ベクタを指定してランデブポートを生成する機能と，ランデブポートのアクセス許可ベクタを変更する機能を追加する．また，参照できるランデブポート状態に追加がある．

ランデブを回送するサービスコール（fwd_por）とランデブを終了させるサービスコール（rpl_rdv）は，ランデブを受け付けたタスクの所属する保護ドメインのみから呼び出すことができる. 他の保護ドメインから呼び出された場合には，E_ILUSE エラーを返す．ランデブを受け付けたタスクがカーネルドメイン以外に所属する場合には，カーネルドメインから呼び出された場合にも，E_ILUSE エラーを返す．ランデブを回送するサービスコール（fwd_por）は，これに加えて, 回送先のランデブポートのアクセス許可ベクタによっても保護する．この場合のエラーコードは，E_OACVである．

ランデブ状態を参照するサービスコール（ref_rdv）は，ランデブが成立したランデブポートのアクセス許可ベクタにより, 参照操作が許可されている保護ドメインのみから呼び出せる（カーネルドメインからは常に呼び出せる）．他の保護ドメインから呼び出された場合には，E_OACVエラーを返す．

ランデブポート状態のパケット形式に，アクセス許可ベクタを追加する．

```
typedef struct t_rpor {
        ID          ctskid ;    /* ランデブポートの呼出し待ち行列の
                                   先頭のタスクのID番号 */
        ID          atskid ;    /* ランデブポートの受付待ち行列の先
                                   頭のタスクのID番号 */
        ACVCT       acvct ;     /* ランデブポートのアクセス許可ベク
                                   タ */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RPOR ;
```

ランデブ機能の新規サービスコールの機能コードは次の通りである．

```
TFN_CRA_POR     –0x118  cra_porの機能コード
TFN_ACRA_POR    –0x128  acra_porの機能コード
TFN_SAC_POR     –0x138  sac_porの機能コード
```

CRA_POR ランデブポートの生成（静的API，アクセス許可指定）
cra_por ランデブポートの生成（アクセス許可指定）
acra_por ランデブポートの生成（ID番号自動割付け，アクセス許可指定）

【静的API】

```
CRA_POR ( ID porid, { ATR poratr, UINT maxcmsz,
                      UINT maxrmsz }, ACVCT acvct ) ;
```

【C言語API】

```
ER ercd = cra_por ( ID porid, T_CPOR *pk_cpor,
                    ACVCT *p_acvct ) ;
ER_ID porid = acra_por ( T_CPOR *pk_cpor, ACVCT *p_acvct ) ;
```

【パラメータ】

ID	porid	生成対象のランデブポートのID番号
T_CPOR *	pk_cpor	ランデブポート生成情報を入れたパケットへのポインタ（静的APIではパケットの内容を直接記述する）
ACVCT	acvct	ランデブポートのアクセス許可ベクタ

pk_cporの内容（T_CPOR型）

ATR	poratr	ランデブポート属性
UINT	maxcmsz	呼出しメッセージの最大サイズ（バイト数）
UINT	maxrmsz	返答メッセージの最大サイズ（バイト数）

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cra_porの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acra_porの場合

ER_ID	porid	生成したランデブポートのID番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（poridが不正あるいは使用できない）
E_NOID	ID番号不足（割付け可能なランデブポートIDがない）
E_RSATR	予約属性（poratrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_cpor，maxcmsz，maxrmsz，p_acvct，acvctが不正）
E_OBJ	オブジェクト状態エラー（対象ランデブポートが登録済み）

【機能】

poridで指定されるID番号を持つランデブポートを，pk_cporで指定されるランデブポート生成情報に基づいて生成する．CRE_POR／cre_por／acre_porとの違いは，生成するランデブポートに，acvctで指定されるアクセス許可ベクタを設定することのみである．

sac_por　ランデブポートのアクセス許可ベクタの変更

【C言語API】

```
ER ercd = sac_por ( ID porid, ACVCT *p_acvct ) ;
```

【パラメータ】

ID	porid	変更対象のランデブポートのID番号
ACVCT	acvct	ランデブポートのアクセス許可ベクタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（poridが不正あるいは使用できない）
E_PAR	パラメータエラー（p_acvct，acvctが不正）
E_NOEXS	オブジェクト未生成（対象ランデブポートが未登録）

【機能】

poridで指定されるランデブポートに対するアクセス許可ベクタを，acvctで指定されるアクセス許可ベクタに設定する．

【補足説明】

このサービスコールは，呼び出された時点ですでにランデブポートの呼出し待ち行列につながれているタスクには影響しない．そのため，ランデブポートの呼出し待ち行列につながれているタスクが，新たに設定したアクセス許可ベクタではランデブを呼び出すことを許可されていない場合でも，そのタスクはランデブ呼出し待ち状態のままとなり，ランデブが成立すれば，ランデブ終了待ち状態となることができる．ランデブポートの受付待ち行列につながれているタスクについても同様である．

fwd_por　ランデブの回送

【C言語API】

```
ER ercd = fwd_por ( ID porid, RDVPTN calptn, RDVNO rdvno,
                    VP msg, UINT cmsgsz ) ;
```

【パラメータ】

ID	porid	回送先のランデブポートのID番号
RDVPTN	calptn	呼出し側のランデブ条件を示すビットパターン
RDVNO	rdvno	回送するランデブ番号
VP	msg	呼出しメッセージの先頭番地
UINT	cmsgsz	呼出しメッセージのサイズ（バイト数）

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（poridが不正あるいは使用できない）
E_PAR	パラメータエラー（calptn，msg，cmsgszが不正）
E_ILUSE	サービスコール不正使用（回送先のランデブポートの返答メッセージの最大サイズが大きすぎる，ランデブを受け付けたタスクの所属する保護ドメイン以外からの呼出し）
E_OBJ	オブジェクト状態エラー（rdvnoが不正）
E_NOEXS	オブジェクト未生成（対象ランデブポートが未登録）

【機能】

rdvnoで指定されるランデブ番号を付与されたランデブを，poridで指定されるランデブポートに対して，calptnで指定されるランデブ条件により回送する．μITRON4.0仕様との違いは，指定されたランデブを受け付けたタスクの所属する保護ドメイン以外（カーネルドメインも例外ではない）から呼び出された場合に，E_ILUSEエラーを返すことのみである．

rpl_rdv　　ランデブの終了

【C言語API】

```
ER ercd = rpl_rdv ( RDVNO rdvno, VP msg, UINT rmsgsz ) ;
```

【パラメータ】

RDVNO	rdvno	終了させるランデブ番号
VP	msg	返答メッセージの先頭番地
UINT	rmsgsz	返答メッセージのサイズ（バイト数）

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	パラメータエラー（msg，rmsgszが不正）
E_ILUSE	サービスコール不正使用（ランデブを受け付けたタスクの所属する保護ドメイン以外からの呼出し）
E_OBJ	オブジェクト状態エラー（rdvnoが不正）

【機能】

rdvnoで指定されるランデブ番号を付与されたランデブを終了させる．µITRON4.0仕様との違いは，指定されたランデブを受け付けたタスクの所属する保護ドメイン以外（カーネルドメインも例外ではない）から呼び出された場合に，E_ILUSEエラーを返すことのみである．

ref_por　　ランデブポートの状態参照

【C言語API】

ER ercd = ref_por (ID porid, T_RPOR *pk_rpor) ;

【パラメータ】

ID	porid	状態参照対象のランデブポートのID番号
T_RPOR *	pk_rpor	ランデブポート状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rporの内容（T_RPOR型）

ID	ctskid	ランデブポートの呼出し待ち行列の先頭のタスクのID番号
ID	atskid	ランデブポートの受付待ち行列の先頭のタスクのID番号
ACVCT	acvct	ランデブポートのアクセス許可ベクタ

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（poridが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_rporが不正）
E_NOEXS	オブジェクト未生成（対象ランデブポートが未登録）

【機能】

poridで指定されるランデブポートに関する状態を参照し，pk_rporで指定されるパケットに返す．μITRON4.0仕様との違いは，acvctに対象ランデブポートのアクセス許可ベクタを返すことのみである．

3.6 メモリプール管理機能

3.6.1 固定長メモリプール

アプリケーションプログラムからカーネルの用いる管理領域を保護するために，固定長メモリプール機能にはいくつかの変更がある．また，固定長メモリプール機能には，アクセス許可ベクタを指定して固定長メモリプールを生成する機能と，固定長メモリプールのアクセス許可ベクタを変更する機能を追加する．参照できる固定長メモリプール状態にも変更がある．

μITRON4.0仕様では，返却されたメモリブロックを管理するための情報を，メモリブロック内に置くように実装することを想定している．この仕様では，アプリケーションからカーネルを保護するためにこの想定を改め，返却されたメモリブロックを管理するための情報を置くためのメモリ領域を別に確保する．このメモリ領域を，固定長メモリプール管理領域と呼ぶ．

固定長メモリプール機能に関連して，次のカーネル構成マクロを追加する．

SIZE mpfmbsz = TSZ_MPFMB (UINT blkcnt, UINT blksz)

サイズがblkszバイトのメモリブロックをblkcnt個獲得できる固定長メモリプールの管理に必要な固定長メモリプール管理領域のサイズ（バイト数）

固定長メモリプール生成情報のパケット形式は，次の通りに変更する．また，固定長メモリプール状態のパケット形式に，アクセス許可ベクタを追加する．

```
typedef struct t_cmpf {
        ATR             mpfatr ;        /* 固定長メモリプール属性 */
        UINT            blkcnt ;        /* 獲得できるメモリブロック数（個
                                           数）*/
        UINT            blksz ;         /* メモリブロックのサイズ（バイト
                                           数）*/
        VP              mpf ;           /* 固定長メモリプール領域の先頭番
                                           地 */
        VP              mpfmb ;         /* 固定長メモリプール管理領域の先頭
                                           番地 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_CMPF ;

typedef struct t_rmpf {
        ID              wtskid ;        /* 固定長メモリプールの待ち行列の先
                                           頭のタスクのID番号 */
        UINT            fblkcnt ;       /* 固定長メモリプールの空きメモリブ
                                           ロック数（個数）*/
        ACVCT           acvct ;         /* 固定長メモリプールのアクセス許可
                                           ベクタ */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RMPF ;
```

固定長メモリプール機能の新規サービスコールの機能コードは次の通りであ

る．

TFN_CRA_MPF	–0x119	cra_mpfの機能コード
TFN_ACRA_MPF	–0x129	acra_mpfの機能コード
TFN_SAC_MPF	–0x139	sac_mpfの機能コード

【補足説明】

固定長メモリプール機能は，それぞれのメモリブロックを保護する機能は提供しない．カーネルは，それぞれのメモリブロックではなく，固定長メモリプール領域全体をメモリ保護の対象とする．すなわち，固定長メモリプール領域にアクセスできる保護ドメインは，すべてのメモリブロックにアクセスすることができる．また，固定長メモリブロックを返却するサービスコール（rel_mpf）は，返却するメモリブロックに関する保護を行わない．固定長メモリプールに対する返却操作を許可された保護ドメインは，他の保護ドメインに所属するタスクが獲得したメモリブロックであっても，返却することができる．

CRE_MPF	固定長メモリプールの生成（静的API）
CRA_MPF	固定長メモリプールの生成（静的API，アクセス許可指定）
cre_mpf	固定長メモリプールの生成
cra_mpf	固定長メモリプールの生成（アクセス許可指定）
acre_mpf	固定長メモリプールの生成（ID番号自動割付け）
acra_mpf	固定長メモリプールの生成（ID番号自動割付け，アクセス許可指定）

【静的API】

```
CRE_MPF ( ID mpfid, { ATR mpfatr, UINT blkcnt, UINT blksz,
                      VP mpf, VP mpfmb } ) ;
CRA_MPF ( ID mpfid, { ATR mpfatr, UINT blkcnt, UINT blksz,
                      VP mpf, VP mpfmb }, ACVCT acvct ) ;
```

【C言語API】

```
ER ercd = cre_mpf ( ID mpfid, T_CMPF *pk_cmpf ) ;
ER ercd = cra_mpf ( ID mpfid, T_CMPF *pk_cmpf,
                    ACVCT *p_acvct ) ;
ER_ID mpfid = acre_mpf ( T_CMPF *pk_cmpf ) ;
ER_ID mpfid = acra_mpf ( T_CMPF *pk_cmpf, ACVCT *p_acvct ) ;
```

【パラメータ】

ID	mpfid	生成対象の固定長メモリプールのID番号
T_CMPF *	pk_cmpf	固定長メモリプール生成情報を入れたパケットへのポインタ（静的APIではパケットの内容を直接記述する）
ACVCT	acvct	固定長メモリプールのアクセス許可ベクタ

pk_cmpfの内容（T_CMPF型）

ATR	mpfatr	固定長メモリプール属性
UINT	blkcnt	獲得できるメモリブロック数（個数）
UINT	blksz	メモリブロックのサイズ（バイト数）
VP	mpf	固定長メモリプール領域の先頭番地
VP	mpfmb	固定長メモリプール管理領域の先頭番地

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cre_mpf，cra_mpfの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acre_mpf，acra_mpfの場合

ER_ID	mpfid	生成した固定長メモリプールの ID 番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mpfidが不正あるいは使用できない）
E_NOID	ID番号不足（割付け可能な固定長メモリプールIDがない）
E_NOMEM	メモリ不足（固定長メモリプール領域，固定長メモリプール管理領域などが確保できない）
E_RSATR	予約属性（mpfatrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_cmpf，blkcnt，blksz，mpf，mpfmb，p_acvct，acvctが不正）
E_OBJ	オブジェクト状態エラー（対象固定長メモリプールが登録済み）

【機能】

mpfidで指定されるID番号を持つ固定長メモリプールを，pk_cmpfで指定される固定長メモリプール生成情報に基づいて生成する．μITRON4.0仕様との違いは次の通り．

mpfmbは，固定長メモリプール管理領域の先頭番地である．

mpfで指定された番地から，blkszバイトのメモリブロックをblkcnt個獲得できるのに必要なサイズのメモリ領域を，固定長メモリプール領域として使用する．アプリケーションプログラムは，TSZ_MPFを用いて，必要な固定長メモリプール領域のサイズを知ることができる．固定長メモリプール領域として使用するメモリ領域が，メモリオブジェクトの境界を越えている場合には，E_PARエラーを返す．

mpfにNULL（＝0）が指定された場合には，必要なサイズのメモリ領域を，固定長メモリプールと同じ保護ドメインに所属し，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領域の中に，カーネルが確保する．

mpfmbで指定された番地から，blkszバイトのメモリブロックをblkcnt個獲得できる固定長メモリプールの管理に必要なサイズのメモリ領域を，固定長メモリプール管理領域として使用する．アプリケーションプログラムは，TSZ_MPFMBを用いて，必要な固定長メモリプール管理領域のサイズを知ることができる．固定長メモリプール管理領域として使用するメモリ領域が，メモリオブジェクトの境界を越えている場合や，何らかの操作／アクセスがカーネルドメイン以外にも許可されているメモリオブジェクトに含まれる場合には，E_PARエラーを返す．

mpfmbにNULL（＝0）が指定された場合には，必要なサイズのメモリ領域を，すべての操作／アクセスがカーネルドメインのみに許可されているメモリ領域の中に，カーネルが確保する．

静的APIにおいては，mpfmbは省略することができる．省略された場合には，NULLが指定されたものと扱う．

【補足説明】

カーネルが固定長メモリプール領域を確保する場合, 固定長メモリプールに対するアクセス許可ベクタが, 固定長メモリプール領域のアクセス許可ベクタを兼ねる．すなわち，固定長メモリプール領域に対する書込みアクセス，読出しアクセス，管理操作，参照操作のアクセス許可パターンを，それぞれ固定長メモリプールに対する獲得操作，返却操作，管理操作，参照操作のアクセス許可パターンと同じに設定する．

固定長メモリプール領域を使ったメッセージ通信において, メッセージを中継するタスクがある場合, そのタスクから固定長メモリプールにアクセスする必要はないが, 固定長メモリプール領域にはアクセスしたいことになる．このような場合には, 広い方のアクセス許可ベクタを固定長メモリプールに設定する必要がある．

固定長メモリプール領域に対するアクセス許可ベクタを, 固定長メモリプールに対するアクセス許可ベクタと別に設定したい場合には, 固定長メモリプール領域をアプリケーションで確保すればよい．この場合カーネルは，固定長メモリプール領域の所属保護ドメインやアクセス許可ベクタが適切であるかのチェックを行わない．

sac_mpf　固定長メモリプールのアクセス許可ベクタの変更

【C言語API】

```
ER ercd = sac_mpf ( ID mpfid, ACVCT *p_acvct ) ;
```

【パラメータ】

ID	mpfid	変更対象の固定長メモリプールのID番号
ACVCT	acvct	固定長メモリプールのアクセス許可ベクタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mpfidが不正あるいは使用できない）
E_PAR	パラメータエラー（p_acvct，acvctが不正）
E_NOEXS	オブジェクト未生成（対象固定長メモリプールが未登録）

【機能】

mpfidで指定される固定長メモリプールに対するアクセス許可ベクタを，acvctで指定されるアクセス許可ベクタに設定する．

【補足説明】

このサービスコールは，呼び出された時点ですでに固定長メモリプールの待ち行列につながれているタスクには影響しない．そのため，固定長メモリプールの待ち行列につながれているタスクが，新たに設定したアクセス許可ベクタでは固定長メモリブロックを獲得することを許可されていない場合でも，そのタスクは固定長メモリブロックの獲得待ち状態のままとなり，メモリブロックが返却されれば，メモリブロックを獲得することができる．

ref_mpf　固定長メモリプールの状態参照

【C言語API】

ER ercd = ref_mpf (ID mpfid, T_RMPF *pk_rmpf) ;

【パラメータ】

ID	mpfid	状態参照対象の固定長メモリプールのID番号
T_RMPF *	pk_rmpf	固定長メモリプール状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rmpfの内容（T_RMPF型）

ID	wtskid	固定長メモリプールの待ち行列の先頭のタスクのID番号
UINT	fblkcnt	固定長メモリプールの空きメモリブロック数（個数）
ACVCT	acvct	固定長メモリプールのアクセス許可ベクタ

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（mpfidが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_rmpfが不正）
E_NOEXS	オブジェクト未生成（対象固定長メモリプールが未登録）

【機能】

mpfidで指定される固定長メモリプールに関する状態を参照し, pk_rmpfで指定されるパケットに返す．µITRON4.0仕様との違いは，acvctに対象固定長メモリプールのアクセス許可ベクタを返すことのみである.

3.6.2 可変長メモリプール

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．

【補足説明】
可変長メモリプール機能をサポートしない最大の理由は，小さいサイズの可変長メモリブロックの動的管理（獲得や解放など）が効率的に実現でき，メモリブロックを管理するためにカーネルが用いる管理領域を保護できる方法がないことである．大きいサイズ（ハードウェア的にメモリ保護が可能な単位以上）の可変長メモリブロックの動的管理には，保護メモリプールを用いることができる．アプリケーションが，小さいサイズの可変長メモリブロックの動的管理を必要とする場合には，そのための機能をライブラリとして実現するのが妥当であろう．

3.7 時間管理機能

3.7.1 システム時刻管理

システム時刻管理機能には, システム時刻のアクセス許可ベクタを設定／変更する機能と，システム時刻の状態を参照する機能を追加する.

システム時刻状態のパケット形式として，次のデータ型を定義する.

```
typedef struct t_rtim {
        ACVCT       acvct ;         /* システム時刻のアクセス許可ベク
                                       タ */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RTIM ;
```

システム時刻管理機能の新規サービスコールの機能コードは次の通りである.

TFN_SAC_TIM	–0xb6	sac_tim の機能コード
TFN_REF_TIM	–0xb5	ref_tim の機能コード

SAC_TIM システム時刻のアクセス許可ベクタの設定（静的API）
sac_tim システム時刻のアクセス許可ベクタの変更

【静的API】

SAC_TIM (ACVCT acvct) ;

【C言語API】

ER ercd = sac_tim (ACVCT *p_acvct) ;

【パラメータ】

ACVCT	acvct	システム時刻のアクセス許可ベクタ

【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_PAR	パラメータエラー（p_acvct，acvctが不正）

【機能】

システム時刻に対するアクセス許可ベクタを，acvctで指定されるアクセス許可ベクタに設定する．

【補足説明】

SAC_TIMを，カーネルドメインの囲みの内側以外に記述した場合や，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した場合には，コンフィギュレータがエラーを報告する．SAC_TIMが記述されなかった場合，システム時刻に対するアクセス許可ベクタは，デフォルトの値（すなわちすべての種別の操作／アクセスをカーネルドメインのみに許可するアクセス許可ベクタ）に初期化する．

ref_tim　　システム時刻状態の参照

【C言語API】

ER ercd = ref_tim (T_RTIM *pk_rtim) ;

【パラメータ】

T_RTIM *	pk_rtim	システム時刻状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rtimの内容（T_RTIM型）

ACVCT	acvct	システム時刻のアクセス許可ベクタ

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_PAR	パラメータエラー（pk_rtimが不正）

【機能】

システム時刻に関する状態を参照し，pk_rtimで指定されるパケットに返す．acvctには，システム時刻のアクセス許可ベクタを返す．

3.7.2 周期ハンドラ

周期ハンドラ機能には, アクセス許可ベクタを指定して周期ハンドラを生成する機能と, 周期ハンドラのアクセス許可ベクタを変更する機能を追加する. また, 参照できる周期ハンドラ状態に追加がある.

周期ハンドラは, この仕様ではカーネルドメインに所属するものと制限されており, 特権モードで実行される. 周期ハンドラを生成する静的APIを, カーネルドメインの囲みの内側以外に記述した場合には, コンフィギュレータがエラーを報告する. また, 周期ハンドラを生成するサービスコールで, オブジェクト属性によりカーネルドメイン以外に所属するように指定された場合には, サービスコールがE_NOSPTエラーを返す.

周期ハンドラ状態のパケット形式に, アクセス許可ベクタを追加する.

```
typedef struct t_rcyc {
        STAT        cycstat ;    /* 周期ハンドラの動作状態 */
        RELTIM      lefttim ;    /* 周期ハンドラを次に起動する時刻まで
                                    の時間 */
        ACVCT       acvct ;      /* 周期ハンドラのアクセス許可ベクタ */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RCYC ;
```

周期ハンドラ機能の新規サービスコールの機能コードは次の通りである.

TFN_CRA_CYC	–0x11b	cra_cycの機能コード
TFN_ACRA_CYC	–0x12b	acra_cycの機能コード
TFN_SAC_CYC	–0x13b	sac_cycの機能コード

CRA_CYC 周期ハンドラの生成（静的API，アクセス許可指定）
cra_cyc 周期ハンドラの生成（アクセス許可指定）
acra_cyc 周期ハンドラの生成（ID番号自動割付け，アクセス許可指定）

【静的API】

```
CRA_CYC ( ID cycid, { ATR cycatr, VP_INT exinf, FP cychdr,
                      RELTIM cyctim, RELTIM cycphs },
                      ACVCT acvct ) ;
```

【C言語API】

```
ER ercd = cra_cyc ( ID cycid, T_CCYC *pk_ccyc,
                      ACVCT *p_acvct ) ;
ER_ID cycid = acra_cyc ( T_CCYC *pk_ccyc, ACVCT *p_acvct ) ;
```

【パラメータ】

ID	cycid	生成対象の周期ハンドラのID番号
T_CCYC *	pk_ccyc	周期ハンドラ生成情報を入れたパケットへのポインタ（静的APIではパケットの内容を直接記述する）
ACVCT	acvct	周期ハンドラのアクセス許可ベクタ

pk_ccycの内容（T_CCYC型）

ATR	cycatr	周期ハンドラ属性
VP_INT	exinf	周期ハンドラの拡張情報
FP	cychdr	周期ハンドラの起動番地
RELTIM	cyctim	周期ハンドラの起動周期
RELTIM	cycphs	周期ハンドラの起動位相

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cra_cycの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acra_cycの場合

ER_ID	cycid	生成した周期ハンドラのID番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（cycidが不正あるいは使用できない）
E_NOID	ID番号不足（割付け可能な周期ハンドラIDがない）
E_RSATR	予約属性（cycatrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_ccyc, cychdr, cyctim, cycphs, p_acvct，acvctが不正）
E_OBJ	オブジェクト状態エラー（対象周期ハンドラが登録済

み）

【機能】

cycidで指定されるID番号を持つ周期ハンドラを，pk_ccycで指定される周期ハンドラ生成情報に基づいて生成する．CRE_CYC／cre_cyc／acre_cycとの違いは，生成する周期ハンドラに，acvctで指定されるアクセス許可ベクタを設定することのみである．

sac_cyc　周期ハンドラのアクセス許可ベクタの変更

【C言語API】

ER ercd = sac_cyc (ID cycid, ACVCT *p_acvct) ;

【パラメータ】

ID	cycid	変更対象の周期ハンドラのID番号
ACVCT	acvct	周期ハンドラのアクセス許可ベクタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（cycidが不正あるいは使用できない）
E_PAR	パラメータエラー（p_acvct，acvctが不正）
E_NOEXS	オブジェクト未生成（対象周期ハンドラが未登録）

【機能】

cycidで指定される周期ハンドラに対するアクセス許可ベクタを，acvctで指定されるアクセス許可ベクタに設定する．

ref_cyc　周期ハンドラの状態参照

【C言語API】

ER ercd = ref_cyc (ID cycid, T_RCYC *pk_rcyc) ;

【パラメータ】

ID	cycid	状態参照対象の周期ハンドラのID番号
T_RCYC *	pk_rcyc	周期ハンドラ状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rcycの内容（T_RCYC型）

STAT	cycstat	周期ハンドラの動作状態
RELTIM	lefttim	周期ハンドラを次に起動する時刻までの時間
ACVCT	acvct	周期ハンドラのアクセス許可ベクタ

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（cycidが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_rcycが不正）
E_NOEXS	オブジェクト未生成（対象周期ハンドラが未登録）

【機能】

cycidで指定される周期ハンドラに関する状態を参照し，pk_rcycで指定されるパケットに返す．μITRON4.0仕様との違いは，acvctに対象周期ハンドラのアクセス許可ベクタを返すことのみである．

3.7.3 アラームハンドラ

アラームハンドラ機能には, アクセス許可ベクタを指定してアラームハンドラを生成する機能と, アラームハンドラのアクセス許可ベクタを変更する機能を追加する．また，参照できるアラームハンドラ状態に追加がある．

アラームハンドラは, この仕様ではカーネルドメインに所属するものと制限されており, 特権モードで実行される．アラームハンドラを生成する静的APIを, カーネルドメインの囲みの内側以外に記述した場合には, コンフィギュレータがエラーを報告する．また，アラームハンドラを生成するサービスコールで, オブジェクト属性によりカーネルドメイン以外に所属するように指定された場合には，サービスコールがE_NOSPTエラーを返す．

アラームハンドラ状態のパケット形式に，アクセス許可ベクタを追加する．

```
typedef struct t_ralm {
        STAT        almstat ;    /* アラームハンドラの動作状態 */
        RELTIM      lefttim ;    /* アラームハンドラの起動時刻までの
                                    時間 */
        ACVCT       acvct ;      /* アラームハンドラのアクセス許可ベ
                                    クタ */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RALM ;
```

アラームハンドラ機能の新規サービスコールの機能コードは次の通りである．

TFN_CRA_ALM	–0x11c	cra_alm の機能コード
TFN_ACRA_ALM	–0x12c	acra_alm の機能コード
TFN_SAC_ALM	–0x13c	sac_alm の機能コード

CRA_ALM アラームハンドラの生成（静的API，アクセス許可指定）
cra_alm アラームハンドラの生成（アクセス許可指定）
acra_alm アラームハンドラの生成（ID番号自動割付け，アクセス許可指定）

【静的API】

```
CRA_ALM ( ID almid, { ATR almatr, VP_INT exinf, FP almhdr },
                        ACVCT acvct ) ;
```

【C言語API】

```
ER ercd = cra_alm ( ID almid, T_CALM *pk_calm,
                        ACVCT *p_acvct ) ;
ER_ID almid = acra_alm ( T_CALM *pk_calm, ACVCT *p_acvct ) ;
```

【パラメータ】

ID	almid	生成対象のアラームハンドラのID番号
T_CALM *	pk_calm	アラームハンドラ生成情報を入れたパケットへのポインタ（静的APIではパケットの内容を直接記述する）
ACVCT	acvct	アラームハンドラのアクセス許可ベクタ

pk_calmの内容（T_CALM型）

ATR	almatr	アラームハンドラ属性
VP_INT	exinf	アラームハンドラの拡張情報
FP	almhdr	アラームハンドラの起動番地

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cra_almの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acra_almの場合

ER_ID	almid	生成したアラームハンドラのID番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（almidが不正あるいは使用できない）
E_NOID	ID番号不足（割付け可能なアラームハンドラIDがない）
E_RSATR	予約属性（almatrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_calm, almhdr, p_acvct, acvctが不正）
E_OBJ	オブジェクト状態エラー（対象アラームハンドラが登録済み）

【機能】

almidで指定されるID番号を持つアラームハンドラを, pk_calmで指定されるアラームハンドラ生成情報に基づいて生成する．CRE_ALM ／ cre_alm ／ acre_alm との違いは, 生成するアラームハンドラに, acvctで指定されるアクセス許可ベクタを設定することのみである．

sac_alm　アラームハンドラのアクセス許可ベクタの変更

【C言語API】

ER ercd = sac_alm (ID almid, ACVCT *p_acvct) ;

【パラメータ】

ID	almid	変更対象のアラームハンドラのID番号
ACVCT	acvct	アラームハンドラのアクセス許可ベクタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（almidが不正あるいは使用できない）
E_PAR	パラメータエラー（p_acvct，acvctが不正）
E_NOEXS	オブジェクト未生成（対象アラームハンドラが未登録）

【機能】

almidで指定されるアラームハンドラに対するアクセス許可ベクタを，acvctで指定されるアクセス許可ベクタに設定する.

ref_alm　　アラームハンドラの状態参照

【C言語API】

ER ercd = ref_alm (ID almid, T_RALM *pk_ralm) ;

【パラメータ】

ID	almid	状態参照対象のアラームハンドラのID番号
T_RALM *	pk_ralm	アラームハンドラ状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_ralmの内容（T_RALM型）

STAT	almstat	アラームハンドラの動作状態
RELTIM	lefttim	アラームハンドラの起動時刻までの時間
ACVCT	acvct	アラームハンドラのアクセス許可ベクタ

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（almidが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_ralmが不正）
E_NOEXS	オブジェクト未生成（対象アラームハンドラが未登録）

【機能】

almidで指定されるアラームハンドラに関する状態を参照し，pk_ralmで指定されるパケットに返す．μITRON4.0仕様との違いは，acvctに対象アラームハンドラのアクセス許可ベクタを返すことのみである．

3.7.4 オーバランハンドラ

オーバランハンドラ機能には，特に変更がない．

オーバーランハンドラは，カーネルドメインに所属し，特権モードで実行される．オーバランハンドラを定義する静的APIを，カーネルドメインの囲みの内側以外に記述した場合には，コンフィギュレータがエラーを報告する．また，オーバランハンドラを定義するサービスコールでは，オブジェクト属性により所属保護ドメインを設定することはできない．

3.8 システム状態管理機能

システム状態管理機能には, システム状態のアクセス許可ベクタを設定／変更する機能と，実行状態のタスクの所属保護ドメインIDを参照する機能を追加する．また，参照できるシステム状態に追加がある．

システム状態のパケット形式に，アクセス許可ベクタを追加する．

```
typedef struct t_rsys {
        ACVCT       acvct ;         /* システム状態のアクセス許可ベク
                                       タ */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RSYS ;
```

システム状態管理機能の新規サービスコールの機能コードは次の通りである．

TFN_SAC_SYS	–0x62	sac_sysの機能コード
TFN_GET_DID	–0x57	get_didの機能コード

SAC_SYS システム状態のアクセス許可ベクタの設定（静的API）
sac_sys システム状態のアクセス許可ベクタの変更

【静的API】

SAC_SYS (ACVCT acvct) ;

【C言語API】

ER ercd = sac_sys (ACVCT *p_acvct) ;

【パラメータ】

ACVCT	acvct	システム状態のアクセス許可ベクタ

【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_PAR	パラメータエラー（p_acvct，acvctが不正）

【機能】

システム状態に対するアクセス許可ベクタを，acvctで指定されるアクセス許可ベクタに設定する.

get_did　実行状態のタスクの所属保護ドメインIDの参照

【C言語API】

ER ercd = get_did (ID *p_domid) ;

【パラメータ】

なし

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード
ID	domid	実行状態のタスクの所属する保護ドメインのID番号

【エラーコード】

E_PAR	パラメータエラー（p_domidが不正）

【機能】

実行状態のタスク（タスクコンテキストから呼び出された場合は，自タスクに一致する）の所属する保護ドメインのID番号を参照し，domidに返す.

【補足説明】

タスクから呼び出された拡張サービスコールルーチン内でget_did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の所属する保護ドメインのID番号が返る．また，タスクを実行中に起動されたCPU例外ハンドラ（タスクコンテキストで動作する場合）内でget_didを呼び出した場合には，CPU例外を発生させたタスクの所属する保護ドメインのID番号が返る.

呼び出した処理単位の所属する保護ドメイン（拡張サービスコール内で呼び出した場合にはカーネルドメイン）を参照する機能や，拡張サービスコールを呼び出した処理単位の所属する保護ドメインを参照する機能は，この仕様では用意していない（将来のバージョンで用意する可能性はある）.

非タスクコンテキスト用のiget_didについても，必要性が明らかでないため，この仕様では用意していない.

ref_sys　　システムの状態参照

【C言語API】

ER ercd = ref_sys (T_RSYS *pk_rsys) ;

【パラメータ】

T_RSYS *	pk_rsys	システム状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rsysの内容（T_RSYS型）

ACVCT	acvct	システム状態のアクセス許可ベクタ

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_PAR	パラメータエラー（pk_rsysが不正）

【機能】

システムの状態を参照し，pk_rsysで指定されるパケットに返す.

acvctには，システム状態のアクセス許可ベクタを返す.

3.9 割込み管理機能

割込み管理機能には, アクセス許可ベクタを指定して割込みサービスルーチンを生成する機能と, 割込みサービスルーチンのアクセス許可ベクタを変更する機能を追加する. また, 参照できる割込みサービスルーチン状態に追加がある.

割込みハンドラは，カーネルドメインに所属し，特権モードで実行される．割込みハンドラを定義する静的APIを，カーネルドメインの囲みの内側以外に記述した場合には，コンフィギュレータがエラーを報告する．また，割込みハンドラを定義するサービスコールでは, オブジェクト属性により所属保護ドメインを設定することはできない.

割込みサービスルーチンは, この仕様ではカーネルドメインに所属するものと制限されており，特権モードで実行される．割込みサービスルーチンを追加する静的APIを，カーネルドメインの囲みの内側以外に記述した場合には，コンフィギュレータがエラーを報告する．また，割込みサービスルーチンを生成するサービスコールで, オブジェクト属性によりカーネルドメイン以外に所属するように指定された場合には，サービスコールがE_NOSPTエラーを返す.

割込みサービスルーチン状態のパケット形式に, アクセス許可ベクタを追加する.

```
typedef struct t_risr {
        ACVCT       acvct ;         /* 割込みサービスルーチンのアクセス
                                       許可ベクタ */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RISR ;
```

割込みサービスルーチン機能の新規サービスコールの機能コードは次の通りである.

TFN_CRA_ISR	–0x11d	cra_isrの機能コード
TFN_ACRA_ISR	–0x12d	acra_isrの機能コード
TFN_SAC_ISR	–0x13d	sac_isrの機能コード

ATA_ISR 割込みサービスルーチンの追加（静的API，アクセス許可指定）
cra_isr 割込みサービスルーチンの生成（アクセス許可指定）
acra_isr 割込みサービスルーチンの生成（ID番号自動割付け，アクセス許可指定）

【静的API】

```
ATA_ISR ( { ATR isratr, VP_INT exinf, INTNO intno, FP isr },
                        ACVCT acvct ) ;
```

【C言語API】

```
ER ercd = cra_isr ( ID isrid, T_CISR *pk_cisr, ACVCT *p_acvct ) ;
ER_ID isrid = acra_isr ( T_CISR *pk_cisr, ACVCT *p_acvct ) ;
```

【パラメータ】

ID	isrid	生成対象の割込みサービスルーチンのID番号
T_CISR *	pk_cisr	割込みサービスルーチン生成情報を入れたパケットへのポインタ（静的APIではパケットの内容を直接記述する）
ACVCT	acvct	割込みサービスルーチンのアクセス許可ベクタ

pk_cisrの内容（T_CISR型）

ATR	isratr	割込みサービスルーチン属性
VP_INT	exinf	割込みサービスルーチンの拡張情報
INTNO	intno	割込みサービスルーチンを付加する割込み番号
FP	isr	割込みサービスルーチンの起動番地

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cra_isrの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acra_isrの場合

ER_ID	isrid	生成した割込みサービスルーチンのID番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（isridが不正あるいは使用できない）
E_NOID	ID 番号不足（割付け可能な割込みサービスルーチン ID がない）
E_RSATR	予約属性（isratrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_cisr, intno, isr, p_acvct, acvctが不正）
E_OBJ	オブジェクト状態エラー（対象割込みサービスルーチン

が登録済み）

【機能】

isridで指定されるID番号を持つ割込みサービスルーチンを，pk_cisrで指定される割込みサービスルーチン生成情報に基づいて生成する．ATT_ISR／cre_isr／acre_isrとの違いは，生成する割込みサービスルーチンに，acvctで指定されるアクセス許可ベクタを設定することのみである．

sac_isr　割込みサービスルーチンのアクセス許可ベクタの変更

【C言語API】

ER ercd = sac_isr (ID isrid, ACVCT *p_acvct) ;

【パラメータ】

ID	isrid	変更対象の割込みサービスルーチンのID番号
ACVCT	acvct	割込みサービスルーチンのアクセス許可ベクタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（isridが不正あるいは使用できない）
E_PAR	パラメータエラー（p_acvct，acvctが不正）
E_NOEXS	オブジェクト未生成（対象割込みサービスルーチンが未登録）

【機能】

isridで指定される割込みサービスルーチンに対するアクセス許可ベクタを，acvctで指定されるアクセス許可ベクタに設定する．

ref_isr 割込みサービスルーチンの状態参照

【C言語API】

ER ercd = ref_isr (ID isrid, T_RISR *pk_risr) ;

【パラメータ】

ID	isrid	状態参照対象の割込みサービスルーチンの ID番号
T_RISR *	pk_risr	割込みサービスルーチン状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_risrの内容（T_RISR型）

ACVCT	acvct	割込みサービスルーチンのアクセス許可ベクタ

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（isridが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_risrが不正）
E_NOEXS	オブジェクト未生成（対象割込みサービスルーチンが未登録）

【機能】

isridで指定される割込みサービスルーチンに関する状態を参照し，pk_risrで指定されるパケットに返す．µITRON4.0仕様との違いは，acvctに対象割込みサービスルーチンのアクセス許可ベクタを返すことのみである．

3.10 サービスコール管理機能

サービスコール管理機能は, 拡張サービスコールの定義と呼出しを行うための機能である．拡張サービスコールを呼び出すための機能は，標準のサービスコールを呼び出すために用いることもできる．

拡張サービスコールは，非特権モードで実行されるタスクから，特権モードで実行されるルーチンを呼び出すための機能である．また，システム全体を1つのリンク単位としない場合に, 他のリンク単位に含まれるルーチンを呼び出すために用いることもできる．

拡張サービスコールルーチンは，カーネルドメインに所属し，特権モードで実行される．拡張サービスコールを定義する静的APIを，カーネルドメインの囲みの内側以外に記述した場合には，コンフィギュレータがエラーを報告する．また，拡張サービスコールを定義するサービスコールでは，オブジェクト属性により所属保護ドメインを設定することはできない．

3.11 システム構成管理機能

システム構成管理機能に対する変更は, 参照できるバージョン情報に追加があることのみである.

CPU 例外ハンドラは，カーネルドメインに所属し，特権モードで実行される．CPU例外ハンドラを定義する静的APIを，カーネルドメインの囲みの内側以外に記述した場合には，コンフィギュレータがエラーを報告する．また，CPU例外ハンドラを定義するサービスコールでは, オブジェクト属性により所属保護ドメインを設定することはできない.

バージョン状態のパケット形式に, 保護機能拡張仕様のバージョン番号を追加する.

```
typedef struct t_rver {
        UH          maker ;     /* カーネルのメーカコード */
        UH          prid ;      /* カーネルの識別番号 */
        UH          spver ;     /* ITRON仕様のバージョン番号 */
        UH          prver ;     /* カーネルのバージョン番号 */
        UH          prno[4] ;   /* カーネル製品の管理情報 */
        UH          pxver ;     /* 保護機能拡張仕様のバージョン番
                                   号 */
} T_RVER ;
```

ref_ver　　バージョン情報の参照

【C言語API】

```
ER ercd = ref_ver ( T_RVER *pk_rver ) ;
```

【パラメータ】

T_RVER *	pk_rver	バージョン情報を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rverの内容（T_RVER型）

UH	maker	カーネルのメーカコード
UH	prid	カーネルの識別番号
UH	spver	ITRON仕様のバージョン番号
UH	prver	カーネルのバージョン番号
UH	prno[4]	カーネル製品の管理情報
UH	pxver	保護機能拡張仕様のバージョン番号

【エラーコード】

E_PAR	パラメータエラー（pk_rverが不正）

【機能】

使用しているカーネルのバージョン情報を参照し，pk_rverで指定されるパケットに返す．μITRON4.0仕様との違いは次の通り．

pxverでは，下位12ビットでカーネルが準拠する保護機能拡張仕様のバージョン番号をあらわす．上位4ビットは0とする．

3.12 その他の変更

3.12.1 サービスコールの機能コード

この仕様において追加されたサービスコールの機能コードとして，µITRON4.0仕様で将来のカーネルの機能拡張のために予約している範囲で割り付けた. 具体的には，(–0xe0)～(–0x05)の範囲でµITRON4.0仕様で割付けのなかった機能コードと，(–0x140)～(–0x101)の範囲の機能コードを割り付けている.

3.12.2 カーネル共通構成定数

バージョン情報を表すカーネル共通構成定数に, TKERNEL_PXVERを追加する.

TKERNEL_PXVER　保護機能拡張仕様のバージョン番号

TKERNEL_PXVERに定義する定数の意味については, ref_verの機能説明を参照すること.

第4章 新規機能

4.1 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, メモリオブジェクトの管理を行うための機能である. メモリオブジェクト管理機能には, メモリオブジェクトを登録／登録解除する機能, メモリオブジェクトのアクセス許可ベクタを変更する機能, メモリ領域に対するアクセス権をチェックする機能, メモリオブジェクトの状態を参照する機能が含まれる.

メモリオブジェクトは, 先頭番地とサイズを指定してカーネルに登録する方法と, プログラムモジュールを指定してカーネルに登録する方法がある. また, 保護メモリプールからのメモリブロックの獲得／返却によっても, メモリオブジェクトの登録／登録解除が行われる.

メモリオブジェクト属性として, 所属する保護ドメイン, リード／ライト可能かリードオンリーか, キャッシュ可能かキャッシュ不可かを指定することができる. ただし, 実装定義でこれらの指定を無効とすることができる. また, これらの指定の他に, プロセッサ依存のメモリオブジェクト属性を追加してもよい. 例えば, プロセッサによっては, ライトスルーキャッシュかライトバックキャッシュかを指定させることが考えられる.

メモリに対する書込みアクセスは, メモリオブジェクト属性でリード／ライト可能が指定されており, かつアクセス許可ベクタで書込みアクセスが許可されている場合にのみ可能である. そのため, メモリオブジェクト属性でリード／ライト可能に設定した場合でも, 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ない場合には, 書込みアクセスを行うことはできない.

メモリオブジェクト属性に用いる定数の名称と値は次の通りとする.

TA_RW	0x00	リード／ライト可能
TA_RO	0x01	リードオンリー
TA_CACHE	0x00	キャッシュ可能
TA_UNCACHE	0x02	キャッシュ不可

メモリオブジェクト登録情報およびメモリオブジェクト状態のパケット形式として, 次のデータ型を定義する.

```
typedef struct t_amem {
        ATR         mematr ;    /* メモリオブジェクト属性 */
        VP          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）*/
        /* 実装独自に他のフィールドを追加してもよい */
} T_AMEM ;

typedef struct t_rmem {
        ACVCT       acvct ;     /* メモリオブジェクトのアクセス許可
                                   ベクタ */
```

```
        /* 実装独自に他のフィールドを追加してもよい */
    } T_RMEM ;
```

メモリオブジェクト管理機能の各サービスコールの機能コードは次の通りである.

TFN_ATT_MEM	–0xb9	att_memの機能コード
TFN_ATA_MEM	–0xba	ata_memの機能コード
TFN_DET_MEM	–0xbb	det_memの機能コード
TFN_SAC_MEM	–0xbe	sac_memの機能コード
TFN_PRB_MEM	–0xbc	prb_memの機能コード
TFN_REF_MEM	–0xbd	ref_memの機能コード

【補足説明】

この仕様では，メモリオブジェクト属性を動的に参照・設定する機能は用意していない．そのため例えば，保護メモリプールから獲得したメモリブロックを，DMA転送のための領域として使うためにキャッシュ不可に設定するといったことはできない．実装独自の拡張として，これらの機能を用意することは許される．

ATT_MEM メモリオブジェクトの登録（静的API）
ATA_MEM メモリオブジェクトの登録（静的API，アクセス許可指定）
att_mem メモリオブジェクトの登録
ata_mem メモリオブジェクトの登録（アクセス許可指定）

【静的API】

ATT_MEM ({ ATR mematr, VP base, SIZE size }) ;
ATA_MEM ({ ATR mematr, VP base, SIZE size }, ACVCT acvct) ;

【C言語API】

ER ercd = att_mem (T_AMEM *pk_amem) ;
ER ercd = ata_mem (T_AMEM *pk_amem, ACVCT *p_acvct) ;

【パラメータ】

T_AMEM *	pk_amem	メモリオブジェクト登録情報を入れたパケットへのポインタ（静的APIではパケットの内容を直接記述する）
ACVCT	acvct	メモリオブジェクトのアクセス許可ベクタ

pk_amemの内容（T_AMEM型）

ATR	mematr	メモリオブジェクト属性
VP	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）

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_RSATR	予約属性（mematrが不正あるいは使用できない）
E_PAR	パラメータエラー（base，sizeが不正）
E_OBJ	オブジェクト状態エラー（登録済みのメモリオブジェクトと範囲が重なる）

【機能】

baseで指定される番地からsizeで指定されるサイズのメモリ領域を，メモリオブジェクトとして登録する．mematrは，登録するメモリオブジェクトのメモリオブジェクト属性である．

静的APIにおいては，mematrはプリプロセッサ定数式パラメータである．

登録するメモリオブジェクトの所属する保護ドメインは，静的APIにより登録される場合には，静的APIがどの保護ドメインの囲みの内側に記述されるかによって決定する．また，サービスコールにより登録される場合には，メモリオブジェクト属性（mematr）により決定する．メモリオブジェクト属性による所属保護ドメインの指定方法は，生成対象となるカーネルオブジェクトの所属保護ドメインの指定方法と同じである（2.2.3節参照）．

登録するメモリオブジェクトに対するアクセス許可ベクタには，アクセス許可ベクタを指定して登録する機能（ATA_MEM，ata_mem）を用いた場合には指定したアクセス許可ベクタ，指定せずに登録する機能（ATT_MEM，att_mem）を用いた場合にはデフォルトのアクセス許可ベクタを設定する．

mematrには，（(TA_RW ‖ TA_RO) | (TA_CACHE ‖ TA_UNCACHE) | [TA_DOM (domid)]）の指定ができる．また，プロセッサ依存のメモリオブジェクト属性が指定できるよう，実装独自に拡張してもよい．

指定された先頭番地とサイズが，ハードウェア的にメモリ保護が可能な境界・単位に一致しない場合には，メモリ保護が可能な境界・単位に一致するようにメモリ領域の範囲を広げ，メモリオブジェクトとして登録する．登録しようとしたメモリオブジェクトの範囲が，登録済みのメモリオブジェクトの範囲と重なる場合には，E_OBJエラーを返す．

【補足説明】

これらの静的APIまたはサービスコールのパラメータで指定されたメモリの範囲が互いに重なっていなくても，メモリ保護が可能な境界・単位に一致するように広げた後の範囲が重なっていれば，E_OBJ エラーとなる．そのためアプリケーションは，メモリ保護が可能な境界・単位を意識してメモリオブジェクトを登録することが必要な場合がある．例えば，デバイスレジスタのマップされたメモリ領域をメモリオブジェクトとして登録する場合，どのデバイスレジスタの番地が同じ単位に含まれるかを意識して登録しなければならない．

メモリオブジェクトの範囲を，指定した先頭番地より手前に広げて登録した場合には，広げた後の先頭番地を知りたい場合が考えられる．検討した結果，次の3つの選択肢の中から(A)を選んだ．

(A) 知る必要はない．
(B) 先頭番地は広げないこととし，先頭番地をアラインさせるマクロを用意する．
(C) 広げた後の先頭番地を返す．

ATT_MOD　プログラムモジュールの登録（静的API）
ATA_MOD　プログラムモジュールの登録（静的API，アクセス許可指定）

【静的API】

ATT_MOD (〈プログラムモジュール記述〉) ;
ATA_MOD (〈プログラムモジュール記述〉, ACVCT acvct) ;

【パラメータ】

〈プログラムモジュール記述〉プログラムモジュールを指定する文字列
ACVCT　acvct　メモリオブジェクトのアクセス許可ベクタ

【機能】

〈プログラムモジュール記述〉で指定されるプログラムモジュールに含まれる各セクション（テキストセクションやデータセクションなど）を，適当な番地に割り付け，それぞれメモリオブジェクトとして登録する．所属保護ドメイン，メモリオブジェクト属性，アクセス許可ベクタがすべて一致するセクションは，1つのメモリオブジェクトにまとめて登録してもよい．

〈プログラムモジュール記述〉には，オブジェクトファイルを指定する任意の文字列を記述することができ，その解釈方法は実装定義とする．例えば，最終オブジェクトファイル名（リンク単位全体をまとめて登録する場合），中間オブジェクトファイル名，ライブラリ名，それらを含むディレクトリ名，それらのリストを記述したファイルの名称，それらのリストなどを記述できるとする方法が考えられる．

登録するメモリオブジェクトの所属する保護ドメインは，静的APIがどの保護ドメインの囲みの内側に記述されるかによって決定する．

登録するメモリオブジェクトに対するメモリオブジェクト属性は，セクションの種類毎に実装定義のデフォルトの属性を設定する．

登録するメモリオブジェクトに対するアクセス許可ベクタには，アクセス許可ベクタを指定して登録する機能（ATA_MOD）を用いた場合には指定したアクセス許可ベクタ，指定せずに登録する機能（ATT_MOD）を用いた場合にはデフォルトのアクセス許可ベクタを設定する．ただし，いずれの場合も，プログラムコードや読出し専用データのセクションなど，読出し専用のセクションに対するデフォルトのアクセス許可ベクタでは，書込みアクセスは許可しないものとする．

【補足説明】

セクションの種類毎のデフォルトのメモリオブジェクト属性は，例えば，プログラムコードのセクションを含むメモリオブジェクトに対してはリードオンリー・キャッシュ可能，書込み可能なデータ領域のセクションを含むメモリオブジェクトに対してはリード／ライト可能・キャッシュ可能とするのが妥当であろう．

この静的APIで登録するプログラムモジュール内で，特殊なメモリオブジェクト属性を持ったメモリ領域を定義したい場合には, セクションで区別させる方法が考えられる．例えば，DMA 転送に使うためにキャッシュ不可に設定したいメモリ領域をプログラムモジュール内で定義したい場合, キャッシュ不可に設定するセクションを定めておく方法が考えられる．

det_mem　メモリオブジェクトの登録解除

【C言語API】

ER ercd = det_mem (VP base) ;

【パラメータ】

VP	base	メモリオブジェクトの番地

【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_PAR	パラメータエラー（baseが不正）
E_OBJ	オブジェクト状態エラー（登録解除できないメモリオブジェクト）
E_NOEXS	オブジェクト未生成（base で指定される番地を含むメモリオブジェクトがない）

【機能】

baseで指定される番地を含むメモリオブジェクトを登録解除する.

先頭番地とサイズ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（att_mem, ata_mem）によって登録されたメモリオブジェクト以外のメモリオブジェクトを登録解除できるかどうかは実装定義とする. 登録解除できない場合には，E_OBJエラーを返す.

sac_mem　メモリオブジェクトのアクセス許可ベクタの変更

【C言語API】

```
ER ercd = sac_mem ( VP base, ACVCT *p_acvct ) ;
```

【パラメータ】

VP	base	メモリオブジェクトの番地
ACVCT	acvct	メモリオブジェクトのアクセス許可ベクタ

【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_PAR	パラメータエラー（base，p_acvct，acvctが不正）
E_OBJ	オブジェクト状態エラー（アクセス許可ベクタを変更できないメモリオブジェクト）
E_NOEXS	オブジェクト未生成（base で指定される番地を含むメモリオブジェクトがない）

【機能】

baseで指定される番地を含むメモリオブジェクトに対するアクセス許可ベクタを，acvctで指定されるアクセス許可ベクタに設定する．

先頭番地とサイズ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（att_mem，ata_mem）によって登録されたメモリオブジェクト以外のメモリオブジェクトのアクセス許可ベクタを変更できるかどうかは実装定義とする．アクセス許可ベクタを変更できない場合には，E_OBJエラーを返す．

prb_mem　メモリ領域に対するアクセス権のチェック

【C言語API】

```
ER ercd = prb_mem ( VP base, SIZE size, ID domid,
                    MODE pmmode ) ;
```

【パラメータ】

VP	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	domid	アクセス元の保護ドメインの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_ID	不正ID番号（domidが不正あるいは使用できない）
E_PAR	パラメータエラー（base，size，pmmodeが不正）
E_MACV	メモリアクセス違反（メモリ領域へのアクセスが許可されていない）
E_OACV	オブジェクトアクセス違反（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_OBJ	オブジェクト状態エラー（メモリ領域がメモリオブジェクトの境界を越えている）
E_NOEXS	オブジェクト未生成（base で指定される番地を含むメモリオブジェクトがない）

【機能】

baseで指定される先頭番地からsizeで指定されるサイズのメモリ領域が，domidで指定される保護ドメインに対してアクセスが許可されているかをチェックし，アクセスが許可されている場合にE_OKを返す．

pmmodeはアクセスの種別を示し，（TPM_READ｜TPM_WRITE）の指定ができる．pmmodeにTPM_READ（＝0x01）が指定された場合には，読出しアクセスが許可されているかをチェックする．TPM_WRITE（＝0x02）が指定された場合には，書込みアクセスが許可されているかをチェックする．また，(TPM_WRITE | TPM_READ)（＝0x03）が指定された場合には，書込みアクセスと読出しアクセスの両方が許可されているかをチェックする．

domidにTDOM_SELF（＝0）が指定されると，指定されたメモリ領域が，自タスクの所属する保護ドメインに対してアクセスが許可されているかをチェックする．また，domidにTDOM_KERNEL（＝–1）が指定されると，カーネルドメインに対してアクセスが許可されているかをチェックする．domidにTDOM_NONE（＝–2）を指定することはできない．指定された場合

には，E_IDエラーを返す．

指定されたメモリ領域がメモリオブジェクトの境界を越えていることのチェックは必須ではない．実装定義で，指定されたメモリ領域がメモリオブジェクトの境界を越えている場合でも，E_OBJエラーを返さない場合があるものとしてよい．ただしこの場合でも，このサービスコールがE_OKを返すのは，指定されたメモリ領域全体に対してアクセスが許可されている場合のみである．

ref_mem　メモリオブジェクト状態の参照

【C言語API】

ER ercd = ref_mem (VP base, T_RMEM *pk_rmem) ;

【パラメータ】

VP	base	メモリオブジェクトの番地
T_RMEM *	pk_rmem	メモリオブジェクト状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rmemの内容（T_RMEM型）

ACVCT	acvct	メモリオブジェクトのアクセス許可ベクタ

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_PAR	パラメータエラー（base，pk_rmemが不正）
E_NOEXS	オブジェクト未生成（base で指定される番地を含むメモリオブジェクトがない）

【機能】

baseで指定される番地を含むメモリオブジェクトに関する状態を参照し，pk_rmemで指定されるパケットに返す.

acvctには，対象メモリオブジェクトのアクセス許可ベクタを返す.

4.2 保護メモリプール機能

保護メモリプールは, メモリ保護の対象となるメモリブロックを動的に管理するためのオブジェクトである. 保護メモリプール機能には, 保護メモリプールを生成／削除する機能, 保護メモリプールのアクセス許可ベクタを変更する機能, 保護メモリプールに対してメモリブロックを獲得／返却する機能, 保護メモリプールの状態を参照する機能が含まれる. 保護メモリプールはID番号で識別されるオブジェクトである. 保護メモリプールのID番号を保護メモリプールIDと呼ぶ.

保護メモリプールは, メモリブロックを割り付けるメモリ領域（これを保護メモリプール領域, または単にメモリプール領域と呼ぶ）と, それを管理するためのメモリ領域（これを保護メモリプール管理領域, または単にメモリプール管理領域と呼ぶ), メモリブロックの獲得を待つタスクの待ち行列を持つ.

保護メモリプールからメモリブロックを獲得するサービスコールは, 保護メモリプール領域の中にメモリブロックを割り付け, その先頭番地とサイズをリターンパラメータとして返す. 獲得されたメモリブロック（これを保護メモリブロック, または単にメモリブロックと呼ぶ）はメモリオブジェクトとして扱い, メモリ保護の対象とする. メモリ保護の対象とするために, 保護メモリブロックの先頭番地とサイズは, ハードウェア的にメモリ保護が可能な境界・単位に制約される.

保護メモリプールから獲得されたメモリブロックは, 獲得したタスクの所属する保護ドメインに所属させ, その保護ドメインのみからアクセスできるように設定する. 逆に, 保護メモリプールに返却されたメモリブロックは, カーネルドメインのみからアクセスできるように設定する. 保護メモリプール領域の中の未割当てのメモリ領域や返却されたメモリ領域を, メモリオブジェクトとして扱うかどうか, メモリオブジェクトとして扱う場合にはどの単位でメモリオブジェクトとするかは, 実装依存である.

保護メモリブロックを獲得するタスクは, メモリプール領域の空き領域が足りなくなった場合, 十分なサイズのメモリブロックが返却されるまで保護メモリブロックの獲得待ち状態となる. 保護メモリブロックの獲得待ち状態になったタスクは, その保護メモリプールの待ち行列につながれる.

保護メモリブロックを返却するサービスコール（rel_mpp）は, 保護メモリブロックを獲得したタスクの所属する保護ドメインのみから呼び出すことができる. そのため, アクセス許可ベクタによる保護を行わない.

保護メモリプール機能に関連して, 次のカーネル構成マクロを定義する.

> SIZE mppsz = TSZ_MPP (UINT blkcnt, UINT memsz)
>
> サイズがmemszバイト（ないしはそれ以上のサイズ）のメモリブロックをblkcnt個割り付けるのに必要な保護メモリプール領域のサイズ（目安のバイト数）

SIZE mppmbsz = TSZ_MPPMB (SIZE mppsz)

サイズが mppsz バイトの保護メモリプール領域を管理するのに必要な保護メモリプール管理領域のサイズ（バイト数）

TSZ_MPPは，あくまでもメモリプール領域のサイズを決める際の目安として使うためのものである．このマクロを使って，異なるサイズのメモリブロックを割り付けるのに必要なサイズを決定することはできない．また，フラグメンテーションが起こった場合などには，指定した数のメモリブロックが獲得できない場合もある．

保護メモリプール生成情報および保護メモリプール状態のパケット形式として，次のデータ型を定義する．

```
typedef struct t_cmpp {
        ATR             mppatr ;        /* 保護メモリプール属性 */
        SIZE            mppsz ;         /* 保護メモリプール領域のサイズ（バイト数）*/
        VP              mpp ;           /* 保護メモリプール領域の先頭番地 */
        VP              mppmb ;         /* 保護メモリプール管理領域の先頭番地 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_CMPP ;

typedef struct t_rmpp {
        ID              wtskid ;        /* 保護メモリプールの待ち行列の先頭のタスクのID番号 */
        SIZE            fmppsz ;        /* 保護メモリプールの空き領域の合計サイズ（バイト数）*/
        UINT            fblksz ;        /* すぐに獲得可能な最大メモリブロックサイズ（バイト数）*/
        ACVCT           acvct ;         /* 保護メモリプールのアクセス許可ベクタ */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RMPP ;
```

保護メモリプール機能の各サービスコールの機能コードは次の通りである．

TFN_CRE_MPP	–0x101	cre_mppの機能コード
TFN_CRA_MPP	–0x11e	cra_mppの機能コード
TFN_ACRE_MPP	–0xce	acre_mppの機能コード
TFN_ACRA_MPP	–0x12e	acra_mppの機能コード
TFN_DEL_MPP	–0x102	del_mppの機能コード
TFN_SAC_MPP	–0x13e	sac_mppの機能コード
TFN_GET_MPP	–0x105	get_mppの機能コード
TFN_PGET_MPP	–0x106	pget_mppの機能コード
TFN_TGET_MPP	–0x107	tget_mppの機能コード
TFN_REL_MPP	–0x103	rel_mppの機能コード
TFN_REF_MPP	–0x108	ref_mppの機能コード

【補足説明】

カーネルドメイン以外の保護ドメインに所属するタスクは, 保護メモリプールからメモリブロックを獲得することで, そのメモリブロックにアクセスできるようになる．逆に，保護メモリプールに返却したメモリブロックや，保護メールボックスに送信したメモリブロックは, タスクからアクセスすることができなくなる．

拡張サービスコールルーチン内で獲得した保護メモリブロックは, 拡張サービスコールルーチンが所属するカーネルドメインではなく, 拡張サービスコールルーチンを呼び出したタスクの所属する保護ドメインに所属する．

カーネルは, 割り付けた保護メモリブロックをメモリオブジェクトとして扱うため，保護メモリブロックのアクセス許可ベクタをsac_memにより変更することができる．

保護メモリプールでメモリブロックの獲得を待っているタスクは, 待ち行列につながれている順序でメモリブロックを獲得する．例えば，ある保護メモリプールに対して16KBのメモリブロックを獲得しようとしているタスクAと，8KBのメモリブロックを獲得しようとしているタスクBが，この順で待ち行列につながれている時に, 別のタスクからのメモリブロックの返却により8KBの連続した空き領域ができたとする．このような場合でも，タスクAがメモリブロックを獲得するまで，タスクBはメモリブロックを獲得できない．ただし，実装独自の拡張として，このような場合にタスクBに先にメモリブロックを獲得させる指定を，保護メモリプール属性に追加することは許される．

CRE_MPP	保護メモリプールの生成（静的API）
CRA_MPP	保護メモリプールの生成（静的API，アクセス許可指定）
cre_mpp	保護メモリプールの生成
cra_mpp	保護メモリプールの生成（アクセス許可指定）
acre_mpp	保護メモリプールの生成（ID番号自動割付け）
acra_mpp	保護メモリプールの生成（ID番号自動割付け，アクセス許可指定）

【静的API】

```
CRE_MPP ( ID mppid, { ATR mppatr, SIZE mppsz, VP mpp,
                      VP mppmb } ) ;
CRA_MPP ( ID mppid, { ATR mppatr, SIZE mppsz, VP mpp,
                      VP mppmb }, ACVCT acvct ) ;
```

【C言語API】

```
ER ercd = cre_mpp ( ID mppid, T_CMPP *pk_cmpp ) ;
ER ercd = cra_mpp ( ID mppid, T_CMPP *pk_cmpp,
                    ACVCT *p_acvct ) ;
ER_ID mppid = acre_mpp ( T_CMPP *pk_cmpp ) ;
ER_ID mppid = acra_mpp ( T_CMPP *pk_cmpp,
                    ACVCT *p_acvct ) ;
```

【パラメータ】

ID	mppid	生成対象の保護メモリプールのID番号
T_CMPP *	pk_cmpp	保護メモリプール生成情報を入れたパケットへのポインタ（静的APIではパケットの内容を直接記述する）
ACVCT	acvct	保護メモリプールのアクセス許可ベクタ

pk_cmppの内容（T_CMPP型）

ATR	mppatr	保護メモリプール属性
SIZE	mppsz	保護メモリプール領域のサイズ（バイト数）
VP	mpp	保護メモリプール領域の先頭番地
VP	mppmb	保護メモリプール管理領域の先頭番地

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cre_mpp，cra_mppの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acre_mpp，acra_mppの場合

ER_ID	mppid	生成した保護メモリプールのID番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mppidが不正あるいは使用できない）
E_NOID	ID番号不足（割付け可能な保護メモリプールIDがない）
E_NOMEM	メモリ不足（保護メモリプール領域，保護メモリプール管理領域などが確保できない）
E_RSATR	予約属性（mppatrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_cmpp，mppsz，mpp，mppmb，p_acvct，acvctが不正）
E_OBJ	オブジェクト状態エラー（対象保護メモリプールが登録済み）

【機能】

mppidで指定されるID番号を持つ保護メモリプールを，pk_cmppで指定される保護メモリプール生成情報に基づいて生成する．mppatrは保護メモリプールの属性，mppszは保護メモリプール領域のサイズ（バイト数），mppは保護メモリプール領域の先頭番地，mppmbは保護メモリプール管理領域の先頭番地である．

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

acre_mppおよびacra_mppは，生成する保護メモリプールのID番号を保護メモリプールが登録されていないID番号の中から割り付け，割り付けたID番号を返値として返す．

mppatrには，（TA_TFIFO ‖ TA_TPRI）の指定ができる．保護メモリプールの待ち行列は，TA_TFIFO（＝0x00）が指定された場合にはFIFO順，TA_TPRI（＝0x01）が指定された場合にはタスクの優先度順となる．

mppで指定された番地からmppszバイトのメモリ領域を，保護メモリプール領域として使用する．アプリケーションプログラムは，TSZ_MPPを用いて，mppszに指定すべきサイズの目安を知ることができる．mppに指定された番地が，ハードウェア的にメモリ保護が可能な境界にアラインしていない場合には，E_PARエラーを返す．また，保護メモリプール領域として使用するメモリ領域が，メモリオブジェクトの境界を越えている場合や，何らかの操作／アクセスがカーネルドメイン以外にも許可されているメモリオブジェクトに含まれる場合にも，E_PARエラーを返す．さらに，保護メモリプール領域として使用するメモリ領域が，他の保護メモリプールから獲得したメモリブロックに含まれている場合にも，E_PARエラーとすることができる．

mppにNULL（＝0）が指定された場合には，mppszで指定されたサイズのメモリ領域を，すべての操作／アクセスがカーネルドメインのみに許可されているメモリ領域の中に，カーネルが確保する．

また，mppmbで指定された番地から，mppszバイトの保護メモリプール領域を管理するのに必要なサイズのメモリ領域を，保護メモリプール管理領域として使用する．アプリケーションプログラムは，TSZ_MPPMBを用いて，必要

な保護メモリプール管理領域のサイズを知ることができる. 保護メモリプール管理領域として使用するメモリ領域が, メモリオブジェクトの境界を越えている場合や, 何らかの操作／アクセスがカーネルドメイン以外にも許可されているメモリオブジェクトに含まれる場合には，E_PAR エラーを返す.

mppmb に NULL（＝0）が指定された場合には, 必要なサイズのメモリ領域を, すべての操作／アクセスがカーネルドメインのみに許可されているメモリ領域の中に，カーネルが確保する.

mppsz に 0 が指定された場合や, 実装定義の最大値よりも大きい値が指定された場合には，E_PAR エラーを返す.

【補足説明】

mpp に NULL が指定された場合にカーネルが確保するメモリプール領域のサイズは, mppsz に指定されたサイズ以上であれば, それよりも大きくてもよい.

del_mpp　保護メモリプールの削除

【C言語API】

ER ercd = del_mpp (ID mppid) ;

【パラメータ】

ID	mppid	削除対象の保護メモリプールのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mppidが不正あるいは使用できない）
E_OBJ	オブジェクト状態エラー（未返却のメモリブロックがある）
E_NOEXS	オブジェクト未生成（対象保護メモリプールが未登録）

【機能】

mppidで指定される保護メモリプールを削除する. 保護メモリプール領域および保護メモリプール管理領域をカーネルが確保した場合には, それらの領域を解放する.

指定された保護メモリプールから獲得されたメモリブロックに未返却のものがある場合には，E_OBJエラーを返す.

【補足説明】

指定された保護メモリプールからメモリブロックの獲得を待っているタスクがある場合の扱いについては，μITRON4.0仕様書の3.8節を参照すること.

未返却のメモリブロックがある場合にE_OBJエラーを返す仕様は，メモリブロックの獲得を待っているタスクがある場合の扱いと一貫していない. 将来のバージョンでは，この仕様を変更する可能性がある.

sac_mpp　保護メモリプールのアクセス許可ベクタの変更

【C言語API】

```
ER ercd = sac_mpp ( ID mppid, ACVCT *p_acvct ) ;
```

【パラメータ】

ID	mppid	変更対象の保護メモリプールのID番号
ACVCT	acvct	保護メモリプールのアクセス許可ベクタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mppidが不正あるいは使用できない）
E_PAR	パラメータエラー（p_acvct，acvctが不正）
E_NOEXS	オブジェクト未生成（対象保護メモリプールが未登録）

【機能】

mppidで指定される保護メモリプールに対するアクセス許可ベクタを，acvctで指定されるアクセス許可ベクタに設定する．

【補足説明】

このサービスコールは，呼び出された時点ですでに保護メモリプールの待ち行列につながれているタスクには影響しない．そのため，保護メモリプールの待ち行列につながれているタスクが，新たに設定したアクセス許可ベクタでは保護メモリブロックを獲得することを許可されていない場合でも，そのタスクは保護メモリブロックの獲得待ち状態のままとなり，保護メモリプール領域に十分な空き領域ができれば，保護メモリブロックを獲得することができる．

get_mpp 保護メモリブロックの獲得
pget_mpp 保護メモリブロックの獲得（ポーリング）
tget_mpp 保護メモリブロックの獲得（タイムアウトあり）

【C言語API】

```
ER_UINT blksz = get_mpp ( ID mppid, UINT memsz, VP *p_blk ) ;
ER_UINT blksz = pget_mpp ( ID mppid, UINT memsz,
                           VP *p_blk ) ;
ER_UINT blksz = tget_mpp ( ID mppid, UINT memsz, VP *p_blk,
                           TMO tmout ) ;
```

【パラメータ】

ID	mppid	メモリブロック獲得対象の保護メモリプールのID番号
UINT	memsz	獲得するメモリブロックのサイズ（バイト数）
TMO	tmout	タイムアウト指定

【リターンパラメータ】

ER_UINT	blksz	獲得したメモリブロックのサイズ（正の値）またはエラーコード
VP	blk	獲得したメモリブロックの先頭番地

【エラーコード】

E_ID	不正ID番号（mppidが不正あるいは使用できない）
E_PAR	パラメータエラー（memsz，p_blk，tmoutが不正）
E_NOEXS	オブジェクト未生成（対象保護メモリプールが未登録）
E_RLWAI	待ち状態の強制解除（待ち状態の間にrel_waiを受付）
E_TMOUT	ポーリング失敗またはタイムアウト
E_DLT	待ちオブジェクトの削除（待ち状態の間に対象保護メモリプールが削除）

【機能】

mppidで指定される保護メモリプールから，memszで指定される以上のサイズのメモリブロックを割り付け，そのサイズをblkszに，先頭番地をblkに返す．割り付けたメモリブロックは，メモリオブジェクトとして登録し，自タスクの所属する保護ドメインに所属させる．また，メモリブロックに対するアクセス許可ベクタを，すべての操作／アクセスをその保護ドメインのみに許可するように設定する．

自タスクより優先してメモリブロックを獲得できるタスクが待っている場合や，保護メモリプール領域に十分な空き領域がなく，指定されたサイズのメモリブロックを割り付けられない場合には，自タスクを待ち状態とする．

具体的には，指定された保護メモリプールでメモリブロック獲得を待っている

タスクがない場合や，保護メモリプール属性にTA_TPRI（＝0x01）が指定されており，メモリブロック獲得を待っているタスクの優先度がいずれも自タスクの優先度よりも低い場合には，メモリプール領域からmemszバイト以上のサイズのメモリブロックを割り付ける．この条件を満たさない場合や，メモリブロックを割り付けるのに十分な空き領域がない場合には，自タスクを待ち行列につなぎ，保護メモリブロックの獲得待ち状態に移行させる．

他のタスクがすでに待ち行列につながっている場合，自タスクを待ち行列につなぐ処理は次のように行う．保護メモリプール属性にTA_TFIFO（＝0x00）が指定されている場合には，自タスクを待ち行列の末尾につなぐ．TA_TPRI（＝0x01）が指定されている場合には，自タスクを優先度順で待ち行列につなぐ．同じ優先度のタスクの中では，自タスクを最後につなぐ．

保護メモリブロックの獲得を待っているタスクが，rel_waiやter_tskにより待ち解除されたり，タイムアウトにより待ち解除された結果，待ち行列の先頭のタスクが変化する時には，新たに先頭になったタスクから順に可能ならメモリブロックを獲得させる処理を行う必要がある．具体的な処理内容は，rel_mppによってメモリブロックが返却された後の処理と同様であるため，rel_mppの機能説明を参照すること．また，保護メモリブロックの獲得を待っているタスクの優先度が，chg_priやミューテックスの操作によって変更された結果，保護メモリプールの待ち行列の先頭のタスクが変化する時にも，同様の処理が必要である．

pget_mppは，get_mppの処理をポーリングで行うサービスコール，tget_mppは，get_mppにタイムアウトの機能を付け加えたサービスコールである．tmoutには，正の値のタイムアウト時間に加えて，TMO_POL（＝0）とTMO_FEVR（＝–1）を指定することができる．

memszに0が指定された場合には，E_PARエラーを返す．また，保護メモリプールから取得可能な最大のメモリブロックサイズよりも大きい値をmemszに指定した場合，実装依存でE_PARエラーを返すことができる．

【補足説明】

これらのサービスコールは，割り付けたメモリブロックのクリアは行わないため，その内容は不定となる．

tget_mppは，tmoutにTMO_POLが指定された場合，E_CTXエラーにならない限りはpget_mppと全く同じ動作をする．また，tmoutにTMO_FEVRが指定された場合は，get_mppと全く同じ動作をする．

rel_mpp　保護メモリブロックの返却

【C言語API】

ER ercd = rel_mpp (ID mppid, VP blk) ;

【パラメータ】

ID	mppid	メモリブロック返却対象の保護メモリプールのID番号
VP	blk	返却するメモリブロックの先頭番地

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mppidが不正あるいは使用できない）
E_PAR	パラメータエラー（blkが不正）
E_ILUSE	サービスコール不正使用（未割付けのメモリブロックの返却，保護メールボックスに送信されたメモリブロックの返却，他の保護ドメインに所属するメモリブロックの返却）
E_NOEXS	オブジェクト未生成（対象保護メモリプールが未登録）

【機能】

mppidで指定される保護メモリプールに対して，blkを先頭番地とするメモリブロックを返却する．返却されたメモリブロックは，すべての操作／アクセスをカーネルドメインのみに許可するように設定する．

指定された保護メモリプールでメモリブロックの獲得を待っているタスクがある場合には，メモリブロックを返却した結果，待ち行列の先頭のタスクが獲得しようとしているサイズのメモリブロックを割り付けられるようになったかを調べる．割り付けられる場合には，そのタスクにメモリブロックを獲得させ，そのタスクを待ち解除する．この時，待ち解除されたタスクに対しては，待ち状態に入ったサービスコールの返値としてE_OKを返し，保護メモリプールから獲得したメモリブロックの先頭番地として割り付けたメモリブロックの先頭番地を返す．さらに，メモリブロックの獲得を待っているタスクが残っている場合には，新たに待ち行列の先頭になったタスクに対して同じ処理を繰り返す．

blkは，指定された保護メモリプール内のメモリブロックの先頭番地でなければならない．また，blkで指定される保護メモリブロックは，割り付けられているものであり（get_mpp，pget_mpp，tget_mppのいずれかのサービスコールで獲得されたもので，まだ返却されていないもの），保護メールボックスのメッセージキューに入っておらず，自タスクの所属する保護ドメインに所属するものでなければならない．blkにそれ以外の番地が指定された場合には，ほ

ぼ静的に検出できるエラーの場合にはE_PAR エラー，それ以外の場合にはE_ILUSEエラーを返す.

【補足説明】

カーネルドメインに所属するタスクであっても，このサービスコールを用いて, 他の保護ドメインに所属する保護メモリブロックを返却することはできない. 自タスク以外の保護ドメインに所属する保護メモリブロックを強制的に返却する機能を，実装独自に追加する場合には，別のサービスコールを用意すべきである（将来のバージョンで，そのようなサービスコールを用意する可能性もある）.

このサービスコールにより，複数のタスクが待ち解除される場合，保護メモリプールの待ち行列につながれていた順序で待ち解除する．そのため，実行可能状態に移行したタスクで同じ優先度を持つものの間では, 待ち行列の中で前につながれていたタスクの方が高い優先順位を持つことになる.

ref_mpp 保護メモリプールの状態参照

【C言語API】

```
ER ercd = ref_mpp ( ID mppid, T_RMPP *pk_rmpp ) ;
```

【パラメータ】

ID	mppid	状態参照対象の保護メモリプールのID番号
T_RMPP *	pk_rmpp	保護メモリプール状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rmppの内容（T_RMPP型）

ID	wtskid	保護メモリプールの待ち行列の先頭のタスクのID番号
SIZE	fmppsz	保護メモリプールの空き領域の合計サイズ（バイト数）
UINT	fblksz	すぐに獲得可能な最大メモリブロックサイズ（バイト数）
ACVCT	acvct	保護メモリプールのアクセス許可ベクタ

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（mppidが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_rmppが不正）
E_NOEXS	オブジェクト未生成（対象保護メモリプールが未登録）

【機能】

mppidで指定される保護メモリプールに関する状態を参照し，pk_rmppで指定されるパケットに返す.

wtskidには，対象保護メモリプールの待ち行列の先頭のタスクのID番号を返す．メモリブロックの獲得を待っているタスクがない場合には，TSK_NONE（＝0）を返す.

fmppszには，対象保護メモリプールの現在の空き領域の合計サイズ（バイト数）を返す.

fblkszには，対象保護メモリプールからすぐに獲得できる最大のメモリブロックサイズ（バイト数）を返す．すぐに獲得できる最大のメモリブロックサイズが，UINT型で表せる最大値よりも大きい場合には，UINT型で表せる最大値をfblkszに返す.

acvctには，対象保護メモリプールのアクセス許可ベクタを返す.

4.3 保護メールボックス機能

保護メールボックスは, メッセージを入れた保護メモリブロックを受け渡すことにより, 保護ドメイン間で同期と通信を行うためのオブジェクトである. 保護メールボックス機能には, 保護メールボックスを生成／削除する機能, 保護メールボックスのアクセス許可ベクタを変更する機能, 保護メールボックスに対してメッセージを送信／受信する機能, 保護メールボックスの状態を参照する機能が含まれる. 保護メールボックスはID番号で識別されるオブジェクトである. 保護メールボックスのID番号を保護メールボックスIDと呼ぶ.

保護メールボックスは, 送信されたメッセージを入れるためのメッセージキューと, メッセージの受信を待つタスクの待ち行列を持つ. メッセージを送信する側（イベントを知らせる側）では, 送信したいメッセージをメッセージキューに入れる. メッセージキューにメッセージを入れるための空きがない場合は, 送信するサービスコールがエラーを返す. 一方, メッセージを受信する側（イベントを待つ側）では, メッセージキューに入っているメッセージを一つ取り出す. メッセージキューにメッセージが入っていない場合は, 次にメッセージが送られてくるまで保護メールボックスからの受信待ち状態になる. 保護メールボックスからの受信待ち状態になったタスクは, その保護メールボックスの待ち行列につながれる.

保護メールボックスへメッセージを送信するサービスコールは, メッセージを格納した保護メモリブロックの先頭番地をパラメータとし, その保護メモリブロック全体をメッセージキューに入れる. そのため, 保護メモリブロックに格納されたメッセージを保護メールボックスに送信することを,「保護メモリブロックを送信する」ともいう. 送信された保護メモリブロックは, カーネルドメインに所属させ, カーネルドメインのみからアクセスできるように設定する.

保護メールボックスからメッセージを受信するサービスコールは, メッセージを格納した保護メモリブロック全体をメッセージキューから取り出し, その先頭番地とサイズをリターンパラメータとして返す. そのため, 保護メモリブロックに格納されたメッセージを保護メールボックスから受信することを,「保護メモリブロックを受信する」ともいう. 受信された保護メモリブロックは, 受信したタスクの所属する保護ドメインに所属させ, その保護ドメインのみからアクセスできるように設定する.

保護メールボックスによるメッセージの送受信は, 実際には, メッセージを格納した保護メモリブロックが所属する保護ドメインと, 保護メモリブロックのアクセス許可ベクタを変更するだけである. すなわち, 送受信されるメッセージの内容のコピーは行わない. そのため, 大きいサイズのデータを, 保護ドメイン間で効率的に受渡しすることができる.

保護メールボックス機能に関連して, 次のカーネル構成マクロを定義する.

SIZE mbpmbsz = TSZ_MBPMB (UINT mbpcnt, PRI maxmpri)

mbpcnt個のメッセージを入れることができ，送信されるメッセージの優先度の最大値が maxmpri である保護メールボックスに必要な保護メールボックス管理領域のサイズ（バイト数）

保護メールボックス生成情報および保護メールボックス状態のパケット形式として，次のデータ型を定義する．

```
typedef struct t_cmbp {
        ATR         mbpatr ;    /* 保護メールボックス属性 */
        UINT        mbpcnt ;    /* 入れることができるメッセージの
                                   数 */
        PRI         maxmpri ;   /* 送信されるメッセージの優先度の最
                                   大値 */
        VP          mbpmb ;     /* 保護メールボックス管理領域の先頭
                                   番地 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_CMBP ;

typedef struct t_rmbp {
        ID          wtskid ;    /* 保護メールボックスの待ち行列の先
                                   頭のタスクのID番号 */
        VP          blk ;       /* メッセージキューの先頭の保護メモ
                                   リブロックの先頭番地 */
        UINT        blksz ;     /* メッセージキューの先頭の保護メモ
                                   リブロックのサイズ */
        ACVCT       acvct ;     /* 保護メールボックスのアクセス許可
                                   ベクタ */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RMBP ;
```

保護メールボックス機能の各サービスコールの機能コードは次の通りである．

TFN_CRE_MBP	–0x109	cre_mbpの機能コード
TFN_CRA_MBP	–0x11f	cra_mbpの機能コード
TFN_ACRE_MBP	–0xcf	acre_mbpの機能コード
TFN_ACRA_MBP	–0x12f	acra_mbpの機能コード
TFN_DEL_MBP	–0x10a	del_mbpの機能コード
TFN_SAC_MBP	–0x13f	sac_mbpの機能コード
TFN_SND_MBP	–0x10b	snd_mbpの機能コード
TFN_RCV_MBP	–0x10d	rcv_mbpの機能コード
TFN_PRCV_MBP	–0x10e	prcv_mbpの機能コード
TFN_TRCV_MBP	–0x10f	trcv_mbpの機能コード
TFN_REF_MBP	–0x110	ref_mbpの機能コード

【補足説明】

カーネルドメイン以外の保護ドメインに所属するタスクは，保護メールボックスから保護メモリブロックを受信することで，その保護メモリブロックにアクセスできるようになる．逆に，保護メールボックスに送信した保護メモリブ

ロックや，保護メモリプールに返却した保護メモリブロックは，タスクからアクセスすることができなくなる．

保護メールボックスに送信するメッセージは，保護メモリプールから獲得したメモリブロックに格納しなければならない．一般的な使い方としては，送信側のタスクが保護メモリプールからメモリブロックを確保し，そこにメッセージを格納して送信し，受信側のタスクはメッセージの内容を取り出した後にそのメモリブロックを保護メモリプールに返却するという手順をとることが多い．

CRE_MBP	保護メールボックスの生成（静的API）
CRA_MBP	保護メールボックスの生成（静的API，アクセス許可指定）
cre_mbp	保護メールボックスの生成
cra_mbp	保護メールボックスの生成（アクセス許可指定）
acre_mbp	保護メールボックスの生成（ID番号自動割付け）
acra_mbp	保護メールボックスの生成（ID番号自動割付け，アクセス許可指定）

【静的API】

```
CRE_MBP ( ID mbpid, { ATR mbpatr, UINT mbpcnt,
                      PRI maxmpri, VP mbpmb } ) ;
CRA_MBP ( ID mbpid, { ATR mbpatr, UINT mbpcnt,
                      PRI maxmpri, VP mbpmb },
                      ACVCT acvct ) ;
```

【C言語API】

```
ER ercd = cre_mbp ( ID mbpid, T_CMBP *pk_cmbp ) ;
ER ercd = cra_mbp ( ID mbpid, T_CMBP *pk_cmbp,
                      ACVCT *p_acvct ) ;
ER_ID mbpid = acre_mbp ( T_CMBP *pk_cmbp ) ;
ER_ID mbpid = acra_mbp ( T_CMBP *pk_cmbp,
                      ACVCT *p_acvct ) ;
```

【パラメータ】

ID	mbpid	生成対象の保護メールボックスのID番号
T_CMBP *	pk_cmbp	保護メールボックス生成情報を入れたパケットへのポインタ（静的APIではパケットの内容を直接記述する）
ACVCT	acvct	保護メールボックスのアクセス許可ベクタ

pk_cmbpの内容（T_CMBP型）

ATR	mbpatr	保護メールボックス属性
UINT	mbpcnt	入れることができるメッセージの数
PRI	maxmpri	送信されるメッセージの優先度の最大値
VP	mbpmb	保護メールボックス管理領域の先頭番地

（実装独自に他の情報を追加してもよい）

【リターンパラメータ】

cre_mbp，cra_mbpの場合

ER	ercd	正常終了（E_OK）またはエラーコード

acre_mbp，acra_mbpの場合

ER_ID	mbpid	生成したメールボックスのID番号（正の値）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mbpidが不正あるいは使用できない）
E_NOID	ID番号不足（割付け可能な保護メールボックスIDがない）
E_NOMEM	メモリ不足（保護メールボックス管理領域などが確保できない）
E_RSATR	予約属性（mbpatrが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_cmbp，mbpcnt，maxmpri，mbpmb，p_acvct，acvctが不正）
E_OBJ	オブジェクト状態エラー（対象保護メールボックスが登録済み）

【機能】

mbpidで指定されるID番号を持つ保護メールボックスを，pk_cmbpで指定される保護メールボックス生成情報に基づいて生成する．mbpatrは保護メールボックスの属性，mbpcntは保護メールボックスに入れることができるメッセージの数，maxmpriは保護メールボックスに送信されるメッセージの優先度の最大値，mbpmbは保護メールボックス管理領域の先頭番地である．maxmpriは，mbpatrにTA_MPRI（＝0x02）が指定された場合にのみ有効である．

静的APIにおいては，mbpidは自動割付け対応整数値パラメータ，mbpatrとmaxmpriはプリプロセッサ定数式パラメータである．

acre_mbpおよびacra_mbpは，生成する保護メールボックスのID番号を保護メールボックスが登録されていないID番号の中から割り付け，割り付けたID番号を返値として返す．

mbpatrには，（（TA_TFIFO ‖ TA_TPRI）｜（TA_MFIFO ‖ TA_MPRI））の指定ができる．保護メールボックスの待ち行列は，TA_TFIFO（＝0x00）が指定された場合にはFIFO順，TA_TPRI（＝0x01）が指定された場合にはタスクの優先度順となる．また，保護メールボックスのメッセージキューは，TA_MFIFO（＝0x00）が指定された場合にはFIFO順，TA_MPRI（＝0x02）が指定された場合にはメッセージの優先度順となる．

mbpmbで指定された番地から，mbpcnt個のメッセージを入れることができ，送信されるメッセージの優先度の最大値がmaxmpriの場合に必要なメモリ領域を，保護メールボックス管理領域として使用する．アプリケーションプログラムは，TSZ_MBPMBを用いて，必要な保護メールボックス管理領域のサイズを知ることができる．保護メールボックス管理領域として使用するメモリ領域が，メモリオブジェクトの境界を越えている場合や，何らかの操作／アクセスがカーネルドメイン以外にも許可されているメモリオブジェクトに含まれる場合には，E_PARエラーを返す．

mbpmbにNULL（＝0）が指定された場合には，必要なサイズのメモリ領域を，すべての操作／アクセスがカーネルドメインのみに許可されているメモリ領

域の中から，カーネルが確保する．

mbpcntに0が指定された場合や，実装定義の最大値よりも大きい値が指定された場合には，E_PARエラーを返す．ただし，実装独自にmbpcntに0を指定できるように拡張することは許される．また，maxmpriに0が指定された場合や，メッセージ優先度の最大値（TMAX_MPRI）よりも大きい値が指定された場合にも，E_PARエラーを返す．

del_mbp　保護メールボックスの削除

【C言語API】

ER ercd = del_mbp (ID mbpid) ;

【パラメータ】

ID	mbpid	削除対象の保護メールボックスのID番号

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mbpidが不正あるいは使用できない）
E_OBJ	オブジェクト状態エラー（メッセージが入っている）
E_NOEXS	オブジェクト未生成（対象保護メールボックスが未登録）

【機能】

mbpidで指定される保護メールボックスを削除する. 保護メールボックス管理領域をカーネルが確保した場合には，その領域を解放する.

指定された保護メールボックスにメッセージが入っている場合には，E_OBJエラーを返す.

【補足説明】

指定された保護メールボックスでメッセージの受信を待っているタスクがある場合の扱いについては，µITRON4.0仕様書の3.8節を参照すること.

メッセージが入っている場合にE_OBJエラーを返す仕様は，メッセージの受信を待っているタスクがある場合の扱いと一貫していない. 将来のバージョンでは，この仕様を変更する可能性がある.

sac_mbp　保護メールボックスのアクセス許可ベクタの変更

【C言語API】

```
ER ercd = sac_mbp ( ID mbpid, ACVCT *p_acvct ) ;
```

【パラメータ】

ID	mbpid	変更対象の保護メールボックスのID番号
ACVCT	acvct	保護メールボックスのアクセス許可ベクタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mbpidが不正あるいは使用できない）
E_PAR	パラメータエラー（p_acvct，acvctが不正）
E_NOEXS	オブジェクト未生成（対象保護メールボックスが未登録）

【機能】

mbpidで指定される保護メールボックスに対するアクセス許可ベクタを，acvctで指定されるアクセス許可ベクタに設定する．

【補足説明】

このサービスコールは，呼び出された時点ですでに保護メールボックスの待ち行列につながれているタスクには影響しない．そのため，保護メールボックスの待ち行列につながれているタスクが，新たに設定したアクセス許可ベクタでは保護メールボックスからメッセージを受信することを許可されていない場合でも，そのタスクは保護メールボックスの受信待ち状態のままとなり，メッセージが送信されれば，メッセージを受信することができる．

snd_mbp　保護メールボックスへの送信

【C言語API】

```
ER ercd = snd_mbp ( ID mbpid, VP blk, PRI msgpri ) ;
```

【パラメータ】

ID	mbpid	送信対象の保護メールボックスのID番号
VP	blk	送信するメッセージを格納した保護メモリブロックの先頭番地
PRI	msgpri	送信するメッセージの優先度

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正ID番号（mbpidが不正あるいは使用できない）
E_PAR	パラメータエラー（blk，msgpriが不正）
E_ILUSE	サービスコール不正使用（未割付けの保護メモリブロックの送信，保護メールボックスに送信された保護メモリブロックの再送信，他の保護ドメインに所属する保護メモリブロックの送信）
E_NOEXS	オブジェクト未生成（対象保護メールボックスが未登録）
E_QOVR	キューイングオーバフロー（保護メールボックスに入れることができるメッセージ数を越える送信）

【機能】

mbpidで指定される保護メールボックスに，blkを先頭番地とする保護メモリブロックに格納されたメッセージを，msgpriで指定される優先度で送信する．保護メールボックスに送信された保護メモリブロックは，カーネルドメインに所属させ，すべての操作／アクセスをカーネルドメインのみに許可するようにアクセス許可ベクタを設定する．msgpriは，指定された保護メールボックス属性にTA_MPRI（＝0x02）が指定されている場合にのみ有効である．

具体的には，指定された保護メールボックスで受信を待っているタスクがある場合は，待ち行列の先頭のタスクにblkで指定された保護メモリブロックを渡し，そのタスクを待ち解除する．この時，待ち解除されたタスクに対しては，待ち状態に入ったサービスコールの返値としてblkで指定された保護メモリブロックのサイズを，保護メールボックスから受信した保護メモリブロックの先頭番地としてblkの値を返す．

受信を待っているタスクがない場合は，blkで指定された保護メモリブロックをメッセージキューに入れる．ここで，保護メールボックス属性にTA_MFIFO（＝0x00）が指定されている場合には，blkで指定された保護メモリブロック

をメッセージキューの末尾に入れる．TA_MPRI（＝ 0x02）が指定されている場合には，保護メモリブロックを，格納されたメッセージの優先度順でメッセージキューに入れる．同じ優先度のメッセージの中では，新たに送信されたメッセージを最後に入れる．メッセージキューに，入れることができる最大数のメッセージが入っており，それ以上のメッセージを入れることができない場合には，E_QOVR エラーを返す．

blk は，いずれかの保護メモリプールから獲得されたメモリブロックの先頭番地でなければならない．また，blk で指定される保護メモリブロックは，割り付けられているものであり（get_mpp，pget_mpp，tget_mpp のいずれかのサービスコールで獲得されたもので，まだ返却されていないもの），保護メールボックスのメッセージキューに入っておらず，自タスクの所属する保護ドメインに所属するものでなければならない．blk にそれ以外の番地が指定された場合には，ほぼ静的に検出できるエラーの場合には E_PAR エラー，それ以外の場合には E_ILUSE エラーを返す．

【補足説明】

カーネルドメインに所属するタスクであっても，このサービスコールを用いて，他の保護ドメインに所属する保護メモリブロックを送信することはできない．

rcv_mbp 保護メールボックスからの受信
prcv_mbp 保護メールボックスからの受信（ポーリング）
trcv_mbp 保護メールボックスからの受信（タイムアウトあり）

【C言語API】

```
ER_UINT blksz = rcv_mbp ( ID mbpid, VP *p_blk ) ;
ER_UINT blksz = prcv_mbp ( ID mbpid, VP *p_blk ) ;
ER_UINT blksz = trcv_mbp ( ID mbpid, VP *p_blk, TMO tmout ) ;
```

【パラメータ】

ID	mbpid	受信対象の保護メールボックスのID番号
TMO	tmout	タイムアウト指定

【リターンパラメータ】

ER_UINT	blksz	受信した保護メモリブロックのサイズ（正の値）またはエラーコード
VP	blk	保護メールボックスから受信した保護メモリブロックの先頭番地

【エラーコード】

E_ID	不正ID番号（mbpidが不正あるいは使用できない）
E_PAR	パラメータエラー（p_blk，tmoutが不正）
E_NOEXS	オブジェクト未生成（対象メールボックスが未登録）
E_RLWAI	待ち状態の強制解除（待ち状態の間にrel_waiを受付）
E_TMOUT	ポーリング失敗またはタイムアウト
E_DLT	待ちオブジェクトの削除（待ち状態の間に対象保護メールボックスが削除）

【機能】

mbpidで指定される保護メールボックスから,保護メモリブロックに格納されたメッセージを受信し，保護メモリブロックのサイズをblkszに，先頭番地をblkに返す．保護メールボックスから受信された保護メモリブロックは，自タスクの所属する保護ドメインに所属させ, すべての操作／アクセスをその保護ドメインのみに許可するようにアクセス許可ベクタを設定する．

具体的には, 指定された保護メールボックスのメッセージキューに保護メモリブロックが入っている場合には，先頭の保護メモリブロックをメッセージキューから取り出し，そのサイズをblkszに，先頭番地をblkに返す．メッセージキューに保護メールボックスが入っていない場合は, 自タスクを待ち行列につなぎ，保護メールボックスからの受信待ち状態に移行させる．

他のタスクがすでに待ち行列につながっている場合, 自タスクを待ち行列につなぐ処理は次のように行う．保護メールボックス属性にTA_TFIFO（＝0x00）が指定されている場合には，自タスクを待ち行列の末尾につなぐ．TA_TPRI

（＝0x01）が指定されている場合には，自タスクを優先度順で待ち行列につなぐ．同じ優先度のタスクの中では，自タスクを最後につなぐ．

prcv_mbpは，rcv_mbpの処理をポーリングで行うサービスコール，trcv_mbpは，rcv_mbpにタイムアウトの機能を付け加えたサービスコールである．tmoutには，正の値のタイムアウト時間に加えて，TMO_POL（＝0）とTMO_FEVR（＝–1）を指定することができる．

【補足説明】

blkszに返すのは，受信した保護メモリブロックのサイズであり，そこに格納されているメッセージのサイズと一致するとは限らない．

trcv_mbpは，tmoutにTMO_POLが指定された場合，E_CTXエラーにならない限りはprcv_mbpと全く同じ動作をする．また，tmoutにTMO_FEVRが指定された場合は，rcv_mbpと全く同じ動作をする．

ref_mbp 保護メールボックスの状態参照

【C言語API】

```
ER ercd = ref_mbp ( ID mbpid, T_RMBP *pk_rmbp ) ;
```

【パラメータ】

ID	mbpid	状態参照対象の保護メールボックスのID番号
T_RMBP *	pk_rmbp	保護メールボックス状態を返すパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

pk_rmbpの内容（T_RMBP型）

ID	wtskid	保護メールボックスの待ち行列の先頭のタスクのID番号
VP	blk	メッセージキューの先頭の保護メモリブロックの先頭番地
UINT	blksz	メッセージキューの先頭の保護メモリブロックのサイズ
ACVCT	acvct	保護メールボックスのアクセス許可ベクタ

（実装独自に他の情報を追加してもよい）

【エラーコード】

E_ID	不正ID番号（mbpidが不正あるいは使用できない）
E_PAR	パラメータエラー（pk_rmbpが不正）
E_NOEXS	オブジェクト未生成（対象メールボックスが未登録）

【機能】

mbpidで指定される保護メールボックスに関する状態を参照し，pk_rmbpで指定されるパケットに返す.

wtskidには，対象保護メールボックスの待ち行列の先頭のタスクのID番号を返す．受信を待っているタスクがない場合には，TSK_NONE（＝0）を返す.

blkとblkszには, それぞれ, 対象保護メールボックスのメッセージキューの先頭の保護メモリブロックの先頭番地とサイズを返す. メッセージキューに保護メモリブロックが入っていない場合には，blkにはNULL（＝0）を返す．また，その場合にblkszに返す値は実装依存である.

acvctには，対象保護メールボックスのアクセス許可ベクタを返す.

【補足説明】

wtskid ≠ TSK_NONEとblk ≠ NULLが同時に成立することはない.

4.4 アラインメントのチェックマクロ

与えられたポインタが, プロセッサがデータを書込み／読出しできるアラインメントになっているかをチェックするために，以下のマクロを定義する（サポートしていないデータ型に対応するチェックマクロは定義する必要がない）.

BOOL align = ALIGN_VB (VP addr)
BOOL align = ALIGN_VH (VP addr)
BOOL align = ALIGN_VW (VP addr)
BOOL align = ALIGN_VD (VP addr)
BOOL align = ALIGN_VP (VP addr)

addrで指定された番地に対して，それぞれVB型，VH型，VW型，VD型，VP型のデータを書込み／読出しできるアラインメントになっている場合にTRUE，そうでない場合にFALSE

実装定義で，これ以外のデータ型に対しても，同様のマクロを用意してもよい．例えば，データ型*TYPE*に対して，次のマクロを用意する.

BOOL align = ALIGN_*TYPE* (VP addr)

【補足説明】

ALIGN_VBは, addrの値にかかわらず常にTRUEとなるマクロになるが, 対称性のために定義することにした.

第5章 参考情報

5.1 リンク単位と保護ドメイン

リンク単位と保護ドメインは, カーネル仕様上は任意の対応関係にあってもよいが, コンフィギュレーション環境を構築する上では, 何らかの制約を設ける必要がある. ユーザがシステム構成を理解しやすくし, コンフィギュレーションを簡単にするためには, 次の2つのモデルが基本となる.

(1) 1リンクモデル

システムを構成するすべてのプログラムモジュールを1つにリンクし, 保護ドメインを中間オブジェクトファイルの集合に対応させるモデル.

(2) リンク単位と保護ドメインを一致させるモデル

保護ドメイン毎に, それに所属するプログラムモジュールを1つにリンクし, 保護ドメインを1つのリンク単位に対応させるモデル. すなわち, カーネルと一緒にリンクされた処理単位はカーネルドメインに所属し, カーネルと独立にリンクされた処理単位は, リンクされた単位毎に1つの保護ドメインを構成する.

これらはあくまでも基本となるモデルであり, それ以外のモデルを取ることもできる. 例えば, 複数の保護ドメインを含むリンク単位が複数あるモデル（実際に有用なのは, 複数の保護ドメインを含むリンク単位が1つと, 1つの保護ドメインのみを含むリンク単位が複数あるモデルであろう）や, 複数のリンク単位を1つの保護ドメインに所属させることができるモデルが考えられる.

5.2 複数の保護ドメインから呼び出されるモジュール

複数の保護ドメインから呼び出されるモジュールを実現する方法として, 共有ライブラリによる実現, 拡張サービスコールによる実現, 独立した保護ドメインでの実現の3つの方法がある.

5.2.1 共有ライブラリによる実現

複数の保護ドメインに所属する処理単位から呼び出され, 保護ドメインを切り換えずに実行されるルーチン（の集合）を, 共有ライブラリと呼ぶ. また共有ライブラリには, 複数の保護ドメインに所属する処理単位から参照される定数データも含まれる. 共有ライブラリのルーチンは, リエントラントに実装するのを基本とする.

5.2.2 拡張サービスコールによる実現

複数の保護ドメインに所属する処理単位から呼び出されるルーチンを，拡張サービスコールルーチンとして実現することができる．拡張サービスコールルーチンは，カーネルドメインに切り換えて実行される．

拡張サービスコールの呼出しを，特定の保護ドメインのみに許可するよう保護したい場合には，拡張サービスコールルーチンの先頭で，許可された保護ドメインからの呼び出しであるかをチェックし，許可されていない場合にはエラーを返す処理を行う．拡張サービスコールを呼び出した保護ドメインは，get_did を用いて取り出すことができる（拡張サービスコールがタスクコンテキストで実行されている場合）．ただし，get_did は，拡張サービスコールルーチンからさらに拡張サービスコールを呼び出した場合に，最初の拡張サービスコールを呼び出したタスクの所属する保護ドメインが返るので注意が必要である．どの保護ドメインからの呼び出しが許可されているかの管理は，拡張サービスコールルーチン側で行う必要がある．

また，拡張サービスコールルーチンで，パラメータによって渡された番地をアクセスする場合には，拡張サービスコールを呼び出した保護ドメインからその番地をアクセスすることを許可されているかをチェックし，許可されていない場合にはエラーを返す処理を行う必要がある．呼び出した保護ドメインから指定された番地へのアクセスが許可されているかどうかは，prb_mem を用いてチェックすることができる．

5.2.3 独立した保護ドメインでの実現

複数の保護ドメインに所属する処理単位から呼び出されるルーチンを，独立した保護ドメインに所属するタスクとして実現し，タスク間同期・通信機能を使って「呼び出す」方法が考えられる．

独立した保護ドメインで実現されたルーチンの呼出しを，特定の保護ドメインのみに許可するよう保護したい場合には，それを呼び出すためのタスク間同期・通信オブジェクトのアクセス許可ベクタにより，保護を実現することができる．例えば，メールボックスを用いてソフトウェア部品に対する処理要求を送る場合には，処理要求を出すことを許可された保護ドメインに対してのみ，メールボックスに対する送信操作を許可すればよい．

タスク間同期・通信オブジェクトのアクセス許可ベクタによる保護では不十分な場合には，独立した保護ドメインで実現されたルーチンを呼び出すためのグルールーチンを拡張サービスコールとして実現し，拡張サービスコールルーチンの中で許可されている呼び出しであるかどうかをチェックする方法をとる必要がある．例えば，メールボックスを用いてソフトウェア部品に対する処理要求を送る場合で，処理要求の内容によってそれを許可される保護ドメインが異なる場合がこれが該当する．これは，この仕様では，メールボックスに対してメッセージを送ったタスク（または，それが所属する保護ドメイン）を，

カーネルが管理していないことからくる限界である．

5.3 タスク優先度の保護

セキュリティ確保のための保護機能としては，使用できるタスク優先度を制限する機能も必要である．これを，タスク優先度の保護と呼ぶ．タスク優先度の保護を行わない場合，例えば，ダウンロードしたプログラムが自タスクを最高優先度に上げて実行することで，システム全体を停止させることができる．

この仕様では，タスク優先度の保護のための機能の標準化は行わないが，セキュリティ確保を目的に実装する場合には，タスク優先度の保護のための機能を設けることを強く推奨する．

タスク優先度の保護を導入する方法として，保護ドメイン毎に，それに所属する処理単位が他のタスクに対して設定できるタスク優先度の範囲（または優先度の最高値）を管理する方法が適当と考えられる．ただし，この仕様では保護ドメインを生成する静的APIを用意していないため，保護ドメイン毎に許されるタスク優先度の範囲を指定させる方法は，アドホックに設けざるをえない．

さらに，このような問題への対処方法として，単にタスク優先度を保護する機能では完全でなく，タスクが使ったプロセッサ時間自身を保護することが必要な場面もあると考えられる．これについては，今後の検討課題とする．

5.4 I/Oポートの保護

インテル系のプロセッサのように，メモリのアドレス空間とは別に，I/Oポートのアドレス空間を持つ場合には，I/Oポートの保護も必要になる．最も簡単な方法は，I/Oポートへのアクセスはカーネルドメインにのみ許可する方法であるが，ユーザドメインに所属するタスクからもI/Oポートにアクセスしたい状況が考えられる．このような場合のI/Oポートのアクセス保護の方法については，実装毎に定義するものとする．

5.5 コンフィギュレータの実現方法の例

この仕様に準拠したコンフィギュレータを実現する際に，プログラムモジュールを登録する静的API（ATT_MOD，ATA_MOD）の実現方法が課題となる．具体的には，プログラムモジュールを登録する際に，メモリ上での配置を制御する方法が課題となる．この節では，これを実現する方法の例を示す．

モジュール配置の制御の内容として，少なくとも，メモリオブジェクトの単位で，ハードウェア的にメモリ保護が可能な境界・単位に置くことが必要である．そのために，モジュールの配置順序やアラインメントを，コンフィギュレータがリンカに対して指定することが必要である．さらに，プロセッサの

アーキテクチャによっては，モジュールの配置を制御することによって，カーネルの実装を最適化できる場合もある．

コンフィギュレータが，メモリ保護に必要な情報を生成したり，モジュール配置の最適化を行うためには，各プログラムモジュールのサイズを知ることが必要になる．登録するモジュールが，最終オブジェクトファイルや中間オブジェクトファイルであれば，オブジェクトファイルを読むことでモジュールのサイズを知ることが可能であるが，ライブラリも登録できるようにする場合には，ライブラリの中でどのルーチンをリンクする必要があるかも知る必要がある．そのための最も簡単な方法は，実際にリンカを使って仮にリンクした結果から，どのルーチンをリンクしたか（ないしは，リンクした結果のモジュールのサイズ）を取り出す方法である．

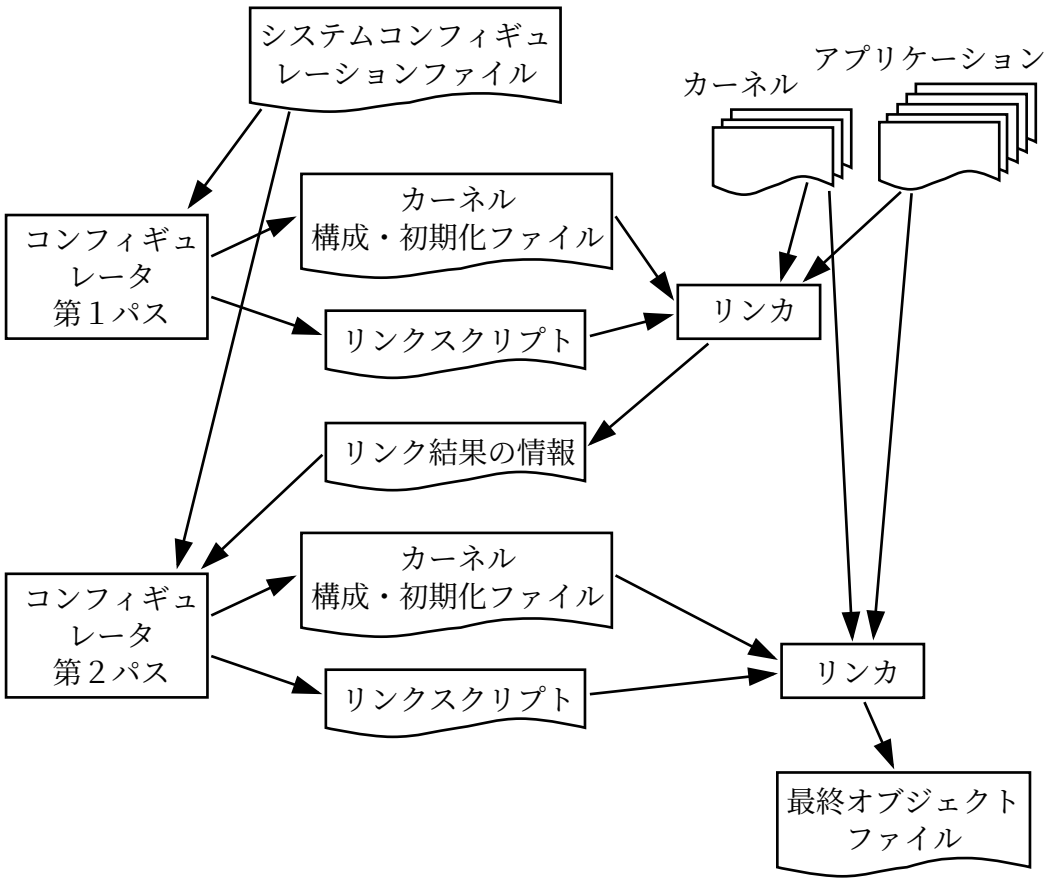

図5-1. コンフィギュレータの処理の流れの例

この方法によってコンフィギュレータを実現した場合の処理の流れを図5-1に示す．コンフィギュレータの第1パスでは，システムコンフィギュレーションファイルの情報を元に，仮リンクのためのカーネル構成・初期化ファイルとリンクスクリプト（リンカに対する指示ファイル）を生成する．生成したファイルを用いて，実際にリンカを使って仮リンクを行う．コンフィギュレータの第2パスでは，仮リンク結果の情報を使って，各プログラムモジュールのメモリ配置を決定する．決定したメモリ配置に従って，リンクのためのカーネル構

成・初期化ファイルとリンクスクリプトを再生成する．生成したファイルを用いて，リンカにより，最終オブジェクトファイルを生成する．

なお，この図では1リンクモデルを仮定しているが，他のモデルでも同様の実現方法が可能である．

5.6 ローダの実現方法の例

この仕様では，プログラムモジュールを動的にロードする機能を，カーネルの機能として用意するのではなく，カーネル上に実現できるようにしている．この節では，カーネル上でローダを実現する方法の例を示す．ローダは，カーネルドメインで実行させることを想定しているが，適切なアクセス許可ベクタを設定することで，他の保護ドメインで実行させることも可能である．

この仕様では，保護ドメインを動的に生成する機能はサポートしていないため，ロードするモジュールを実行するための保護ドメインは，あらかじめ生成しておく必要がある．ローダは，それらの保護ドメインの使用状況を管理し，新たなモジュールをロードする際に未使用の保護ドメインを割り当てる．

ロードするモジュールには，プログラムコードや初期化データに加えて，必要とする資源に関する情報を含めておく．具体的には，必要とするメモリのサイズや，モジュールの実行に必要なカーネルオブジェクトの生成情報がこれに該当する．ローダは，まず必要する資源に関する情報を読み込み，それらの資源が使用可能かどうかをチェックする．また，タスクの優先度の妥当性もチェックする必要がある．

これらのチェックの後に，ローダは，必要なメモリ領域を確保し，カーネルオブジェクトを生成する．必要なメモリ領域は，保護メモリプールから獲得する方法が考えられる．保護メモリプールから獲得したメモリブロックは，ロードするモジュールの保護ドメインからアクセスできるように，アクセス許可ベクタを設定変更する必要がある．この仕様の範囲では，保護メモリプールから獲得したメモリブロックを，ロードするモジュールの保護ドメインに所属させることはできないが，それによる問題はない．それに対して，カーネルオブジェクトを生成する際には，ロードするモジュールの保護ドメインに所属させる．ただし，タスク以外のカーネルオブジェクトについては，ローダの所属保護ドメインに所属させたままでも差し支えない．

以上が完了すると，ローダは，確保したメモリ領域に，プログラムコードと初期化データを転送する．プログラムコードは，位置独立なコードとしておく方法が簡単であるが，別の方法として，ロード時に動的にリロケートする方法も考えられる．プログラムモジュールをネットワークからダウンロードする場合には，このリロケートをサーバ側で行うことも可能である．

以上が完了すると，必要なタスクを起動して，ロードしたモジュールの実行を開始する．

ロードしたモジュールを削除する場合には，モジュールが使用していた資源の

解放もローダが行うべきである．上述の方法では，ロードしたモジュールに初期化時に割り付けた資源を，ローダが管理することができる．また，ロードしたモジュールが動的に資源を確保する場合には，それをローダに管理させる必要がある．そのためには，ロードしたモジュールが資源を動的に確保する場合に，ローダを経由して要求を出させる必要がある．

別の方法として，ロードするモジュールの実行に必要なカーネルオブジェクトを，ローダが生成せずに，モジュールの初期化ルーチンで生成する方法がある．この方法でも，初期化ルーチンで生成するカーネルオブジェクトをローダに管理させるために，生成要求をローダを経由して出させる必要がある．

5.7 仕様策定の経緯とメンバリスト

トロン協会 バージョンアップWGでは，μITRON仕様にメモリ保護機能を追加する仕様を検討するために，2001年1月にメモリ保護機能SWGを設置した．メモリ保護機能SWGは，バージョンアップWGのメンバ社を中心に，トロン協会の会員会社の希望者で構成した．

メモリ保護機能SWGは，毎月平均1回の頻度でミーティングを開き，μITRON4.0仕様の保護機能拡張について検討・策定作業を行った結果，2002年6月に仕様の正式版を公開することとなった．

2002年6月時点でのメモリ保護機能SWGのメンバリストは次の通りである（所属は参加時点のもの）．

メモリ保護機能SWG メンバリスト（あいうえおABC順）

荒木 彰一（松下システムテクノ(株)）
荒木 真司（松下電器産業(株)）
石井 秀弘（横河デジタルコンピュータ(株)）
石川 克己（ヤマハ(株)）
石田 克彦（(株)日立製作所）
稲光 一豊（富士通デバイス(株)）
上山 直美（日本電気(株)）
風見 晴雄（(株)日立製作所）
片山 吉章（三菱電機(株)）
金子 健（(株)エーアイコーポレーション）
亀井 達也（三菱電機セミコンダクタ・アプリケーション・エンジニアリング(株)）
木村 芳孝（富士通デバイス(株)）
金田一 勉（(株)エルミックシステム）
工藤 健治（富士通デバイス(株)）
小林 康浩（富士通(株)）
佐々木 満（NECマイクロシステム(株)）

佐藤 浩司（トヨタ自動車(株)）
澤田 勉（イーソル(株)）
高木 哲郎（(株)デンソークリエイト）
幹事: 高田 広章（豊橋技術科学大学）
事務局: 竹内 透（(社)トロン協会）
竹内 良輔（(株)リコー）
中本 幸二（日本電気(株)）
南角 茂樹（三菱電機(株)）
野村 琢家（松下電器産業(株)）
橋本 真一（(株)ACCESS）
原田 雅章（エーアイシーエンジニアリング(株)）
東原 修（日本電気(株)）
檜原 弘樹（NEC東芝スペースシステム(株)）
光来出 浩（(株)エーアイコーポレーション）
南 章一（横河デジタルコンピュータ(株)）
宮下 光明（(株)グレープシステム）
村木 宏行（三菱電機セミコンダクタ・アプリケーション・エンジニアリング(株)）
望月 望（東芝情報システム(株)）
屋代 基行（エーアイシーエンジニアリング(株)）
安田 吉幸（豊橋技術科学大学）
安武 剛一（松下電器産業(株)）
山田 真二郎（(株)日立製作所）
若林 隆行（豊橋技術科学大学）
脇坂 新路（(株)日立製作所）
渡辺 敏秋（(株)エーアイコーポレーション）
Eva A. Barcelon（豊橋技術科学大学）

5.8 バージョン履歴

2002年5月16日	Ver. 1.A0.00	（バージョンアップWGでの検討資料）
2002年5月28日	Ver. 1.B0.00	（バージョンアップWGでの検討資料）
2002年6月1日	Ver. 1.00.00	正式版を公開

第6章 リファレンス

6.1 サービスコール一覧

(1) タスク管理機能

```
ER ercd = cre_tsk ( ID tskid, T_CTSK *pk_ctsk ) ;
ER ercd = cra_tsk ( ID tskid, T_CTSK *pk_ctsk,
                                ACVCT *p_acvct ) ;
ER_ID tskid = acre_tsk ( T_CTSK *pk_ctsk ) ;
ER_ID tskid = acra_tsk ( T_CTSK *pk_ctsk, ACVCT *p_acvct ) ;
ER ercd = del_tsk ( ID tskid ) ;
ER ercd = sac_tsk ( ID tskid, ACVCT *p_acvct ) ;
ER ercd = act_tsk ( ID tskid ) ;
ER ercd = iact_tsk ( ID tskid ) ;
ER_UINT actcnt = can_act ( ID tskid ) ;
ER ercd = sta_tsk ( ID tskid, VP_INT stacd ) ;
void ext_tsk ( ) ;
void exd_tsk ( ) ;
ER ercd = ter_tsk ( ID tskid ) ;
ER ercd = chg_pri ( ID tskid, PRI tskpri ) ;
ER ercd = get_pri ( ID tskid, PRI *p_tskpri ) ;
ER ercd = ref_tsk ( ID tskid, T_RTSK *pk_rtsk ) ;
ER ercd = ref_tst ( ID tskid, T_RTST *pk_rtst ) ;
```

(2) タスク付属同期機能

```
ER ercd = slp_tsk ( ) ;
ER ercd = tslp_tsk ( TMO tmout ) ;
ER ercd = wup_tsk ( ID tskid ) ;
ER ercd = iwup_tsk ( ID tskid ) ;
ER_UINT wupcnt = can_wup ( ID tskid ) ;
ER ercd = rel_wai ( ID tskid ) ;
ER ercd = irel_wai ( ID tskid ) ;
ER ercd = sus_tsk ( ID tskid ) ;
ER ercd = rsm_tsk ( ID tskid ) ;
ER ercd = frsm_tsk ( ID tskid ) ;
ER ercd = dly_tsk ( RELTIM dlytim ) ;
```

(3) タスク例外処理機能

```
ER ercd = def_tex ( ID tskid, T_DTEX *pk_dtex ) ;
ER ercd = ras_tex ( ID tskid, TEXPTN rasptn ) ;
ER ercd = iras_tex ( ID tskid, TEXPTN rasptn ) ;
```

```
ER ercd = dis_tex ( ) ;
ER ercd = ena_tex ( ) ;
BOOL state = sns_tex ( ) ;
ER ercd = ref_tex ( ID tskid, T_RTEX *pk_rtex ) ;
```

(4) 同期・通信機能

セマフォ

```
ER ercd = cre_sem ( ID semid, T_CSEM *pk_csem ) ;
ER ercd = cra_sem ( ID semid, T_CSEM *pk_csem,
                    ACVCT *p_acvct ) ;
ER_ID semid = acre_sem ( T_CSEM *pk_csem ) ;
ER_ID semid = acra_sem ( T_CSEM *pk_csem,
                    ACVCT *p_acvct ) ;
ER ercd = del_sem ( ID semid ) ;
ER ercd = sac_sem ( ID semid, ACVCT *p_acvct ) ;
ER ercd = sig_sem ( ID semid ) ;
ER ercd = isig_sem ( ID semid ) ;
ER ercd = wai_sem ( ID semid ) ;
ER ercd = pol_sem ( ID semid ) ;
ER ercd = twai_sem ( ID semid, TMO tmout ) ;
ER ercd = ref_sem ( ID semid, T_RSEM *pk_rsem ) ;
```

イベントフラグ

```
ER ercd = cre_flg ( ID flgid, T_CFLG *pk_cflg ) ;
ER ercd = cra_flg ( ID flgid, T_CFLG *pk_cflg, ACVCT *p_acvct ) ;
ER_ID flgid = acre_flg ( T_CFLG *pk_cflg ) ;
ER_ID flgid = acra_flg ( T_CFLG *pk_cflg, ACVCT *p_acvct ) ;
ER ercd = del_flg ( ID flgid ) ;
ER ercd = sac_flg ( ID flgid, ACVCT *p_acvct ) ;
ER ercd = set_flg ( ID flgid, FLGPTN setptn ) ;
ER ercd = iset_flg ( ID flgid, FLGPTN setptn ) ;
ER ercd = clr_flg ( ID flgid, FLGPTN clrptn ) ;
ER ercd = wai_flg ( ID flgid, FLGPTN waiptn, MODE wfmode,
                    FLGPTN *p_flgptn ) ;
ER ercd = pol_flg ( ID flgid, FLGPTN waiptn, MODE wfmode,
                    FLGPTN *p_flgptn ) ;
ER ercd = twai_flg ( ID flgid, FLGPTN waiptn, MODE wfmode,
                    FLGPTN *p_flgptn, TMO tmout ) ;
ER ercd = ref_flg ( ID flgid, T_RFLG *pk_rflg ) ;
```

データキュー

```
ER ercd = cre_dtq ( ID dtqid, T_CDTQ *pk_cdtq ) ;
ER ercd = cra_dtq ( ID dtqid, T_CDTQ *pk_cdtq,
```

```
                                ACVCT *p_acvct ) ;
    ER_ID dtqid = acre_dtq ( T_CDTQ *pk_cdtq ) ;
    ER_ID dtqid = acra_dtq ( T_CDTQ *pk_cdtq, ACVCT *p_acvct ) ;
    ER ercd = del_dtq ( ID dtqid ) ;
    ER ercd = sac_dtq ( ID dtqid, ACVCT *p_acvct ) ;
    ER ercd = snd_dtq ( ID dtqid, VP_INT data ) ;
    ER ercd = psnd_dtq ( ID dtqid, VP_INT data ) ;
    ER ercd = ipsnd_dtq ( ID dtqid, VP_INT data ) ;
    ER ercd = tsnd_dtq ( ID dtqid, VP_INT data, TMO tmout ) ;
    ER ercd = fsnd_dtq ( ID dtqid, VP_INT data ) ;
    ER ercd = ifsnd_dtq ( ID dtqid, VP_INT data ) ;
    ER ercd = rcv_dtq ( ID dtqid, VP_INT *p_data ) ;
    ER ercd = prcv_dtq ( ID dtqid, VP_INT *p_data ) ;
    ER ercd = trcv_dtq ( ID dtqid, VP_INT *p_data, TMO tmout ) ;
    ER ercd = ref_dtq ( ID dtqid, T_RDTQ *pk_rdtq ) ;
```

メールボックス

```
    ER ercd = cre_mbx ( ID mbxid, T_CMBX *pk_cmbx ) ;
    ER ercd = cra_mbx ( ID mbxid, T_CMBX *pk_cmbx,
                                ACVCT *p_acvct ) ;
    ER_ID mbxid = acre_mbx ( T_CMBX *pk_cmbx ) ;
    ER_ID mbxid = acra_mbx ( T_CMBX *pk_cmbx,
                                ACVCT *p_acvct ) ;
    ER ercd = del_mbx ( ID mbxid ) ;
    ER ercd = sac_mbx ( ID mbxid, ACVCT *p_acvct ) ;
    ER ercd = snd_mbx ( ID mbxid, T_MSG *pk_msg ) ;
    ER ercd = rcv_mbx ( ID mbxid, T_MSG **ppk_msg ) ;
    ER ercd = prcv_mbx ( ID mbxid, T_MSG **ppk_msg ) ;
    ER ercd = trcv_mbx ( ID mbxid, T_MSG **ppk_msg,
                                TMO tmout ) ;
    ER ercd = ref_mbx ( ID mbxid, T_RMBX *pk_rmbx ) ;
```

(5) 拡張同期・通信機能

ミューテックス

```
    ER ercd = cre_mtx ( ID mtxid, T_CMTX *pk_cmtx ) ;
    ER ercd = cra_mtx ( ID mtxid, T_CMTX *pk_cmtx,
                                ACVCT *p_acvct ) ;
    ER_ID mtxid = acre_mtx ( T_CMTX *pk_cmtx ) ;
    ER_ID mtxid = acra_mtx ( T_CMTX *pk_cmtx, ACVCT *p_acvct ) ;
    ER ercd = del_mtx ( ID mtxid ) ;
    ER ercd = sac_mtx ( ID mtxid, ACVCT *p_acvct ) ;
    ER ercd = loc_mtx ( ID mtxid ) ;
    ER ercd = ploc_mtx ( ID mtxid ) ;
```

```
ER ercd = tloc_mtx ( ID mtxid, TMO tmout ) ;
ER ercd = unl_mtx ( ID mtxid ) ;
ER ercd = ref_mtx ( ID mtxid, T_RMTX *pk_rmtx ) ;
```

メッセージバッファ

```
ER ercd = cre_mbf ( ID mbfid, T_CMBF *pk_cmbf ) ;
ER ercd = cra_mbf ( ID mbfid, T_CMBF *pk_cmbf,
                    ACVCT *p_acvct ) ;
ER_ID mbfid = acre_mbf ( T_CMBF *pk_cmbf ) ;
ER_ID mbfid = acra_mbf ( T_CMBF *pk_cmbf, ACVCT *p_acvct ) ;
ER ercd = del_mbf ( ID mbfid ) ;
ER ercd = sac_mbf ( ID mbfid, ACVCT *p_acvct ) ;
ER ercd = snd_mbf ( ID mbfid, VP msg, UINT msgsz ) ;
ER ercd = psnd_mbf ( ID mbfid, VP msg, UINT msgsz ) ;
ER ercd = tsnd_mbf ( ID mbfid, VP msg, UINT msgsz,
                    TMO tmout ) ;
ER_UINT msgsz = rcv_mbf ( ID mbfid, VP msg ) ;
ER_UINT msgsz = prcv_mbf ( ID mbfid, VP msg ) ;
ER_UINT msgsz = trcv_mbf ( ID mbfid, VP msg, TMO tmout ) ;
ER ercd = ref_mbf ( ID mbfid, T_RMBF *pk_rmbf ) ;
```

ランデブ

```
ER ercd = cre_por ( ID porid, T_CPOR *pk_cpor ) ;
ER ercd = cra_por ( ID porid, T_CPOR *pk_cpor,
                    ACVCT *p_acvct ) ;
ER_ID porid = acre_por ( T_CPOR *pk_cpor ) ;
ER_ID porid = acra_por ( T_CPOR *pk_cpor, ACVCT *p_acvct ) ;
ER ercd = del_por ( ID porid ) ;
ER ercd = sac_por ( ID porid, ACVCT *p_acvct ) ;
ER_UINT rmsgsz = cal_por ( ID porid, RDVPTN calptn, VP msg,
                    UINT cmsgsz ) ;
ER_UINT rmsgsz = tcal_por ( ID porid, RDVPTN calptn, VP msg,
                    UINT cmsgsz, TMO tmout ) ;
ER_UINT cmsgsz = acp_por ( ID porid, RDVPTN acpptn,
                    RDVNO *p_rdvno, VP msg ) ;
ER_UINT cmsgsz = pacp_por ( ID porid, RDVPTN acpptn,
                    RDVNO *p_rdvno, VP msg ) ;
ER_UINT cmsgsz = tacp_por ( ID porid, RDVPTN acpptn,
                    RDVNO *p_rdvno, VP msg, TMO tmout ) ;
ER ercd = fwd_por ( ID porid, RDVPTN calptn, RDVNO rdvno,
                    VP msg, UINT cmsgsz ) ;
ER ercd = rpl_rdv ( RDVNO rdvno, VP msg, UINT rmsgsz ) ;
ER ercd = ref_por ( ID porid, T_RPOR *pk_rpor ) ;
```

```
ER ercd = ref_rdv ( RDVNO rdvno, T_RRDV *pk_rrdv ) ;
```

(6) メモリプール管理機能

固定長メモリプール

```
ER ercd = cre_mpf ( ID mpfid, T_CMPF *pk_cmpf ) ;
ER ercd = cra_mpf ( ID mpfid, T_CMPF *pk_cmpf,
                            ACVCT *p_acvct ) ;
ER_ID mpfid = acre_mpf ( T_CMPF *pk_cmpf ) ;
ER_ID mpfid = acra_mpf ( T_CMPF *pk_cmpf, ACVCT *p_acvct ) ;
ER ercd = del_mpf ( ID mpfid ) ;
ER ercd = sac_mpf ( ID mpfid, ACVCT *p_acvct ) ;
ER ercd = get_mpf ( ID mpfid, VP *p_blk ) ;
ER ercd = pget_mpf ( ID mpfid, VP *p_blk ) ;
ER ercd = tget_mpf ( ID mpfid, VP *p_blk, TMO tmout ) ;
ER ercd = rel_mpf ( ID mpfid, VP blk ) ;
ER ercd = ref_mpf ( ID mpfid, T_RMPF *pk_rmpf ) ;
```

(7) 時間管理機能

システム時刻管理

```
ER ercd = sac_tim ( ACVCT *p_acvct ) ;
ER ercd = set_tim ( SYSTIM *p_systim ) ;
ER ercd = get_tim ( SYSTIM *p_systim ) ;
ER ercd = isig_tim ( ) ;
ER ercd = ref_tim ( T_RTIM *pk_rtim ) ;
```

周期ハンドラ

```
ER ercd = cre_cyc ( ID cycid, T_CCYC *pk_ccyc ) ;
ER ercd = cra_cyc ( ID cycid, T_CCYC *pk_ccyc,
                            ACVCT *p_acvct ) ;
ER_ID cycid = acre_cyc ( T_CCYC *pk_ccyc ) ;
ER_ID cycid = acra_cyc ( T_CCYC *pk_ccyc, ACVCT *p_acvct ) ;
ER ercd = del_cyc ( ID cycid ) ;
ER ercd = sac_cyc ( ID cycid, ACVCT *p_acvct ) ;
ER ercd = sta_cyc ( ID cycid ) ;
ER ercd = stp_cyc ( ID cycid ) ;
ER ercd = ref_cyc ( ID cycid, T_RCYC *pk_rcyc ) ;
```

アラームハンドラ

```
ER ercd = cre_alm ( ID almid, T_CALM *pk_calm ) ;
ER ercd = cra_alm ( ID almid, T_CALM *pk_calm,
                            ACVCT *p_acvct ) ;
ER_ID almid = acre_alm ( T_CALM *pk_calm ) ;
ER_ID almid = acra_alm ( T_CALM *pk_calm, ACVCT *p_acvct ) ;
ER ercd = del_alm ( ID almid ) ;
```

```
ER ercd = sac_alm ( ID almid, ACVCT *p_acvct ) ;
ER ercd = sta_alm ( ID almid, RELTIM almtim ) ;
ER ercd = stp_alm ( ID almid ) ;
ER ercd = ref_alm ( ID almid, T_RALM *pk_ralm ) ;
```

オーバランハンドラ

```
ER ercd = def_ovr ( T_DOVR *pk_dovr ) ;
ER ercd = sta_ovr ( ID tskid, OVRTIM ovrtim ) ;
ER ercd = stp_ovr ( ID tskid ) ;
ER ercd = ref_ovr ( ID tskid, T_ROVR *pk_rovr ) ;
```

(8) システム状態管理機能

```
ER ercd = sac_sys ( ACVCT *p_acvct ) ;
ER ercd = rot_rdq ( PRI tskpri ) ;
ER ercd = irot_rdq ( PRI tskpri ) ;
ER ercd = get_tid ( ID *p_tskid ) ;
ER ercd = iget_tid ( ID *p_tskid ) ;
ER ercd = get_did ( ID *p_domid ) ;
ER ercd = loc_cpu ( ) ;
ER ercd = iloc_cpu ( ) ;
ER ercd = unl_cpu ( ) ;
ER ercd = iunl_cpu ( ) ;
ER ercd = dis_dsp ( ) ;
ER ercd = ena_dsp ( ) ;
BOOL state = sns_ctx ( ) ;
BOOL state = sns_loc ( ) ;
BOOL state = sns_dsp ( ) ;
BOOL state = sns_dpn ( ) ;
ER ercd = ref_sys ( T_RSYS *pk_rsys ) ;
```

(9) 割込み管理機能

```
ER ercd = def_inh ( INHNO inhno, T_DINH *pk_dinh ) ;
ER ercd = cre_isr ( ID isrid, T_CISR *pk_cisr ) ;
ER ercd = cra_isr ( ID isrid, T_CISR *pk_cisr, ACVCT *p_acvct ) ;
ER_ID isrid = acre_isr ( T_CISR *pk_cisr ) ;
ER_ID isrid = acra_isr ( T_CISR *pk_cisr, ACVCT *p_acvct ) ;
ER ercd = del_isr ( ID isrid ) ;
ER ercd = sac_isr ( ID isrid, ACVCT *p_acvct ) ;
ER ercd = ref_isr ( ID isrid, T_RISR *pk_risr ) ;
ER ercd = dis_int ( INTNO intno ) ;
ER ercd = ena_int ( INTNO intno ) ;
ER ercd = chg_ixx ( IXXXX ixxxx ) ;
ER ercd = get_ixx ( IXXXX *p_ixxxx ) ;
```

(10) サービスコール管理機能

```
ER ercd = def_svc ( FN fncd, T_DSVC *pk_dsvc ) ;
ER_UINT ercd = cal_svc ( FN fncd, VP_INT par1, VP_INT par2,
                         ...) ;
```

(11) システム構成管理機能

```
ER ercd = def_exc ( EXCNO excno, T_DEXC *pk_dexc ) ;
ER ercd = ref_cfg ( T_RCFG *pk_rcfg ) ;
ER ercd = ref_ver ( T_RVER *pk_rver ) ;
```

(12) メモリオブジェクト管理機能

```
ER ercd = att_mem ( T_AMEM *pk_amem ) ;
ER ercd = ata_mem ( T_AMEM *pk_amem, ACVCT *p_acvct ) ;
ER ercd = det_mem ( VP base ) ;
ER ercd = sac_mem ( VP base, ACVCT *p_acvct ) ;
ER ercd = prb_mem ( VP base, SIZE size, ID domid,
                    MODE pmmode ) ;
ER ercd = ref_mem ( VP base, T_RMEM *pk_rmem ) ;
```

(13) 保護メモリプール機能

```
ER ercd = cre_mpp ( ID mppid, T_CMPP *pk_cmpp ) ;
ER ercd = cra_mpp ( ID mppid, T_CMPP *pk_cmpp,
                    ACVCT *p_acvct ) ;
ER_ID mppid = acre_mpp ( T_CMPP *pk_cmpp ) ;
ER_ID mppid = acra_mpp ( T_CMPP *pk_cmpp,
                    ACVCT *p_acvct ) ;
ER ercd = del_mpp ( ID mppid ) ;
ER ercd = sac_mpp ( ID mppid, ACVCT *p_acvct ) ;
ER_UINT blksz = get_mpp ( ID mppid, UINT memsz, VP *p_blk ) ;
ER_UINT blksz = pget_mpp ( ID mppid, UINT memsz,
                    VP *p_blk ) ;
ER_UINT blksz = tget_mpp ( ID mppid, UINT memsz, VP *p_blk,
                    TMO tmout ) ;
ER ercd = rel_mpp ( ID mppid, VP blk ) ;
ER ercd = ref_mpp ( ID mppid, T_RMPP *pk_rmpp ) ;
```

(14) 保護メールボックス機能

```
ER ercd = cre_mbp ( ID mbpid, T_CMBP *pk_cmbp ) ;
ER ercd = cra_mbp ( ID mbpid, T_CMBP *pk_cmbp,
                    ACVCT *p_acvct ) ;
ER_ID mbpid = acre_mbp ( T_CMBP *pk_cmbp ) ;
ER_ID mbpid = acra_mbp ( T_CMBP *pk_cmbp,
                    ACVCT *p_acvct ) ;
ER ercd = del_mbp ( ID mbpid ) ;
```

```
ER ercd = sac_mbp ( ID mbpid, ACVCT *p_acvct ) ;
ER ercd = snd_mbp ( ID mbpid, VP blk, PRI msgpri ) ;
ER_UINT blksz = rcv_mbp ( ID mbpid, VP *p_blk ) ;
ER_UINT blksz = prcv_mbp ( ID mbpid, VP *p_blk ) ;
ER_UINT blksz = trcv_mbp ( ID mbpid, VP *p_blk, TMO tmout ) ;
ER ercd = ref_mbp ( ID mbpid, T_RMBP *pk_rmbp ) ;
```

6.2 静的API一覧

(1) タスク管理機能

```
CRE_TSK ( ID tskid, { ATR tskatr, VP_INT exinf, FP task,
                      PRI itskpri, SIZE stksz, VP stk,
                      SIZE sstksz, VP sstk } ) ;
CRA_TSK ( ID tskid, { ATR tskatr, VP_INT exinf, FP task,
                      PRI itskpri, SIZE stksz, VP stk,
                      SIZE sstksz, VP sstk }, ACVCT acvct ) ;
```

(3) タスク例外処理機能

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

(4) 同期・通信機能

```
CRE_SEM ( ID semid, { ATR sematr, UINT isemcnt,
                      UINT maxsem } ) ;
CRA_SEM ( ID semid, { ATR sematr, UINT isemcnt,
                      UINT maxsem }, ACVCT acvct ) ;
CRE_FLG ( ID flgid, { ATR flgatr, FLGPTN iflgptn } ) ;
CRA_FLG ( ID flgid, { ATR flgatr, FLGPTN iflgptn },
                      ACVCT acvct ) ;
CRE_DTQ ( ID dtqid, { ATR dtqatr, UINT dtqcnt, VP dtqmb } ) ;
CRA_DTQ ( ID dtqid, { ATR dtqatr, UINT dtqcnt, VP dtqmb },
                      ACVCT acvct ) ;
CRE_MBX ( ID mbxid, { ATR mbxatr, UINT mbxcnt,
                      PRI maxmpri, VP mbxmb } ) ;
CRA_MBX ( ID mbxid, { ATR mbxatr, UINT mbxcnt, PRI maxmpri,
                      VP mbxmb }, ACVCT acvct ) ;
```

(5) 拡張同期・通信機能

```
CRE_MTX ( ID mtxid, { ATR mtxatr, PRI ceilpri } ) ;
CRA_MTX ( ID mtxid, { ATR mtxatr, PRI ceilpri }, ACVCT acvct ) ;
CRE_MBF ( ID mbfid, { ATR mbfatr, UINT maxmsz, SIZE mbfsz,
                      VP mbfmb } ) ;
CRA_MBF ( ID mbfid, { ATR mbfatr, UINT maxmsz, SIZE mbfsz,
                      VP mbfmb }, ACVCT acvct ) ;
```

```
CRE_POR ( ID porid, { ATR poratr, UINT maxcmsz,
                      UINT maxrmsz } ) ;
CRA_POR ( ID porid, { ATR poratr, UINT maxcmsz,
                      UINT maxrmsz }, ACVCT acvct ) ;
```

(6) メモリプール管理機能

```
CRE_MPF ( ID mpfid, { ATR mpfatr, UINT blkcnt, UINT blksz,
                      VP mpf, VP mpfmb } ) ;
CRA_MPF ( ID mpfid, { ATR mpfatr, UINT blkcnt, UINT blksz,
                      VP mpf, VP mpfmb }, ACVCT acvct ) ;
```

(7) 時間管理機能

```
SAC_TIM ( ACVCT acvct ) ;
CRE_CYC ( ID cycid, { ATR cycatr, VP_INT exinf, FP cychdr,
                      RELTIM cyctim, RELTIM cycphs } ) ;
CRA_CYC ( ID cycid, { ATR cycatr, VP_INT exinf, FP cychdr,
                      RELTIM cyctim, RELTIM cycphs },
                      ACVCT acvct ) ;
CRE_ALM ( ID almid, { ATR almatr, VP_INT exinf, FP almhdr } ) ;
CRA_ALM ( ID almid, { ATR almatr, VP_INT exinf, FP almhdr },
                      ACVCT acvct ) ;
DEF_OVR ( { ATR ovratr, FP ovrhdr } ) ;
```

(8) システム状態管理機能

```
SAC_SYS ( ACVCT acvct ) ;
```

(9) 割込み管理機能

```
DEF_INH ( INHNO inhno, { ATR inhatr, FP inthdr } ) ;
ATT_ISR ( { ATR isratr, VP_INT exinf, INTNO intno, FP isr } ) ;
ATA_ISR ( { ATR isratr, VP_INT exinf, INTNO intno, FP isr },
                      ACVCT acvct ) ;
```

(10) サービスコール管理機能

```
DEF_SVC ( FN fncd, { ATR svcatr, FP svcrtn } ) ;
```

(11) システム構成管理機能

```
DEF_EXC ( EXCNO excno, { ATR excatr, FP exchdr } ) ;
ATT_INI ( { ATR iniatr, VP_INT exinf, FP inirtn } ) ;
```

(12) メモリオブジェクト管理機能

```
ATT_MEM ( { ATR ematr, VP base, SIZE size } ) ;
ATA_MEM ( { ATR ematr, VP base, SIZE size }, ACVCT acvct ) ;
ATT_MOD (〈プログラムモジュール記述〉) ;
ATA_MOD (〈プログラムモジュール記述〉, ACVCT acvct ) ;
```

(13) 保護メモリプール機能

```
CRE_MPP ( ID mppid, { ATR mppatr, SIZE mppsz, VP mpp,
                      VP mppmb } ) ;
CRA_MPP ( ID mppid, { ATR mppatr, SIZE mppsz, VP mpp,
                      VP mppmb }, ACVCT acvct ) ;
```

(14) 保護メールボックス機能

```
CRE_MBP ( ID mbpid, { ATR mbpatr, UINT mbpcnt,
                      PRI maxmpri, VP mbpmb } ) ;
CRA_MBP ( ID mbpid, { ATR mbpatr, UINT mbpcnt,
                      PRI maxmpri, VP mbpmb },
                      ACVCT acvct ) ;
```

ITRON仕様共通静的API

```
INCLUDE ( 文字列 ) ;
```

6.3 追加／変更されたデータ型

この仕様で追加されたデータ型は次の通り（パケットのためのデータ型を除く）.

ACPTN	アクセス許可パターン
ACVCT	アクセス許可ベクタ

この中で，ACVCTは次のいずれかに定義する.

```
typedef struct {
        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 ;
```

```
typedef struct {
        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 */
} ACVCT ;
```

```
typedef struct {
        ACPTN     acptn1 ;   /* アクセス許可パターン */
} ACVCT ;
```

また，T_MSG_PRIは，次のいずれかに定義するものと変更された.

```
typedef struct t_msg_pri {
        T_MSG     msgque ;   /* メッセージヘッダ */
        PRI       msgpri ;   /* メッセージ優先度 */
} T_MSG_PRI ;
```

```
typedef struct t_msg_pri {
        PRI       msgpri ;   /* メッセージ優先度 */
```

```
} T_MSG_PRI ;
```

6.4 追加／変更されたパケット形式

この仕様で追加または変更されたパケット形式は次の通り.

(1) タスク管理機能

タスク生成情報のパケット形式

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

タスク状態のパケット形式

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

(4) 同期・通信機能

セマフォ状態のパケット形式

```
typedef struct t_rsem {
        ID          wtskid ;    /* セマフォの待ち行列の先頭のタスク
                                   のID番号 */
        UINT        semcnt ;    /* セマフォの現在の資源数 */
        ACVCT       acvct ;     /* セマフォのアクセス許可ベクタ */
        /* 実装独自に他のフィールドを追加してもよい */
```

```
} T_RSEM ;
```

イベントフラグ状態のパケット形式

```
typedef struct t_rflg {
        ID          wtskid ;    /* イベントフラグの待ち行列の先頭の
                                   タスクのID番号 */
        FLGPTN      flgptn ;    /* イベントフラグの現在のビットパ
                                   ターン */
        ACVCT       acvct ;     /* イベントフラグのアクセス許可ベク
                                   タ */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RFLG ;
```

データキュー生成情報のパケット形式

```
typedef struct t_cdtq {
        ATR         dtqatr ;    /* データキュー属性 */
        UINT        dtqcnt ;    /* データキューの容量（データの個
                                   数）*/
        VP          dtqmb ;     /* データキュー管理領域の先頭番
                                   地 */
        VP          dtq ;       /* データキュー領域の先頭番地 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_CDTQ ;
```

データキュー状態のパケット形式

```
typedef struct t_rdtq {
        ID          stskid ;    /* データキューの送信待ち行列の先頭
                                   のタスクのID番号 */
        ID          rtskid ;    /* データキューの受信待ち行列の先頭
                                   のタスクのID番号 */
        UINT        sdtqcnt ;   /* データキューに入っているデータの
                                   数 */
        ACVCT       acvct ;     /* データキューのアクセス許可ベク
                                   タ */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RDTQ ;
```

メールボックス生成情報のパケット形式

```
typedef struct t_cmbx {
        ATR         mbxatr ;    /* メールボックス属性 */
        UINT        mbxcnt ;    /* 入れることができるメッセージの
                                   数 */
        PRI         maxmpri ;   /* 送信されるメッセージの優先度の最
                                   大値 */
        VP          mbxmb ;     /* メールボックス管理領域の先頭番
                                   地 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_CMBX ;
```

メールボックス状態のパケット形式

```
typedef struct t_rmbx {
```

```
        ID          wtskid ;    /* メールボックスの待ち行列の先頭の
                                   タスクのID番号 */
        T_MSG *     pk_msg ;    /* メッセージキューの先頭のメッセー
                                   ジパケットの先頭番地 */
        ACVCT       acvct ;     /* メールボックスのアクセス許可ベク
                                   タ */
        /* 実装独自に他のフィールドを追加してもよい */
    } T_RMBX ;
```

(5) 拡張同期・通信機能

ミューテックス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 */
        ACVCT       acvct ;     /* ミューテックスのアクセス許可ベク
                                   タ */
        /* 実装独自に他のフィールドを追加してもよい */
    } T_RMTX ;
```

メッセージバッファ生成情報のパケット形式

```
    typedef struct t_cmbf {
        ATR         mbfatr ;    /* メッセージバッファ属性 */
        UINT        maxmsz ;    /* メッセージの最大サイズ（バイト
                                   数）*/
        SIZE        mbfsz ;     /* メッセージバッファ管理領域のサイ
                                   ズ（バイト数）*/
        VP          mbfmb ;     /* メッセージバッファ管理領域の先頭
                                   番地 */
        /* 実装独自に他のフィールドを追加してもよい */
    } T_CMBF ;
```

メッセージバッファ状態のパケット形式

```
    typedef struct t_rmbf {
        ID          stskid ;    /* メッセージバッファの送信待ち行列
                                   の先頭のタスクのID番号 */
        ID          rtskid ;    /* メッセージバッファの受信待ち行列
                                   の先頭のタスクのID番号 */
        UINT        smsgcnt ;   /* メッセージバッファに入っている
                                   メッセージの数 */
        SIZE        fmbfsz ;    /* メッセージバッファ管理領域の空き
                                   領域のサイズ（バイト数，最低限の
                                   管理領域を除く）*/
        ACVCT       acvct ;     /* メッセージバッファのアクセス許可
                                   ベクタ */
        /* 実装独自に他のフィールドを追加してもよい */
    } T_RMBF ;
```

ランデブポート状態のパケット形式

```
typedef struct t_rpor {
        ID          ctskid ;    /* ランデブポートの呼出し待ち行列の
                                   先頭のタスクのID番号 */
        ID          atskid ;    /* ランデブポートの受付待ち行列の先
                                   頭のタスクのID番号 */
        ACVCT       acvct ;     /* ランデブポートのアクセス許可ベク
                                   タ */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RPOR ;
```

(6) メモリプール管理機能

固定長メモリプール生成情報のパケット形式

```
typedef struct t_cmpf {
        ATR         mpfatr ;    /* 固定長メモリプール属性 */
        UINT        blkcnt ;    /* 獲得できるメモリブロック数（個
                                   数）*/
        UINT        blksz ;     /* メモリブロックのサイズ（バイト
                                   数）*/
        VP          mpf ;       /* 固定長メモリプール領域の先頭番
                                   地 */
        VP          mpfmb ;     /* 固定長メモリプール管理領域の先頭
                                   番地 */
        /* 実装独自に他のフィールドを追加してもよい */
} T_CMPF ;
```

固定長メモリプール状態のパケット形式

```
typedef struct t_rmpf {
        ID          wtskid ;    /* 固定長メモリプールの待ち行列の先
                                   頭のタスクのID番号 */
        UINT        fblkcnt ;   /* 固定長メモリプールの空きメモリブ
                                   ロック数（個数）*/
        ACVCT       acvct ;     /* 固定長メモリプールのアクセス許可
                                   ベクタ */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RMPF ;
```

(7) 時間管理機能

システム時刻状態のパケット形式

```
typedef struct t_rtim {
        ACVCT       acvct ;     /* システム時刻のアクセス許可ベク
                                   タ */
        /* 実装独自に他のフィールドを追加してもよい */
} T_RTIM ;
```

周期ハンドラ状態のパケット形式

```
typedef struct t_rcyc {
        STAT        cycstat ;   /* 周期ハンドラの動作状態 */
```

```
        RELTIM    lefttim ;    /* 周期ハンドラを次に起動する時刻まで
                                  の時間 */
        ACVCT     acvct ;      /* 周期ハンドラのアクセス許可ベクタ */
        /* 実装独自に他のフィールドを追加してもよい */
    } T_RCYC ;
```

アラームハンドラ状態のパケット形式

```
    typedef struct t_ralm {
        STAT      almstat ;    /* アラームハンドラの動作状態 */
        RELTIM    lefttim ;    /* アラームハンドラの起動時刻までの
                                  時間 */
        ACVCT     acvct ;      /* アラームハンドラのアクセス許可ベ
                                  クタ */
        /* 実装独自に他のフィールドを追加してもよい */
    } T_RALM ;
```

(8) システム状態管理機能

システム状態のパケット形式

```
    typedef struct t_rsys {
        ACVCT     acvct ;      /* システム状態のアクセス許可ベクタ */
        /* 実装独自に他のフィールドを追加してもよい */
    } T_RSYS ;
```

(9) 割込み管理機能

割込みサービスルーチン状態のパケット形式

```
    typedef struct t_risr {
        ACVCT     acvct ;      /* 割込みサービスルーチンのアクセス
                                  許可ベクタ */
        /* 実装独自に他のフィールドを追加してもよい */
    } T_RISR ;
```

(11) システム構成管理機能

バージョン情報のパケット形式

```
    typedef struct t_rver {
        UH        maker ;      /* カーネルのメーカコード */
        UH        prid ;       /* カーネルの識別番号 */
        UH        spver ;      /* ITRON仕様のバージョン番号 */
        UH        prver ;      /* カーネルのバージョン番号 */
        UH        prno[4] ;    /* カーネル製品の管理情報 */
        UH        pxver ;      /* 保護機能拡張仕様のバージョン番
                                  号 */
    } T_RVER ;
```

(12) メモリオブジェクト管理機能

メモリオブジェクト登録情報のパケット形式

```
    typedef struct t_amem {
```

```
        ATR         matr ;    /* メモリオブジェクト属性 */
        VP          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）*/
        /* 実装独自に他のフィールドを追加してもよい */
    } T_AMEM ;
```

メモリオブジェクト状態のパケット形式

```
    typedef struct t_rmem {
        ACVCT       acvct ;   /* メモリオブジェクトのアクセス許可
                                 ベクタ */
        /* 実装独自に他のフィールドを追加してもよい */
    } T_RMEM ;
```

(13) 保護メモリプール機能

保護メモリプール生成情報のパケット形式

```
    typedef struct t_cmpp {
        ATR         mppatr ;  /* 保護メモリプール属性 */
        SIZE        mppsz ;   /* 保護メモリプール領域のサイズ（バ
                                 イト数）*/
        VP          mpp ;     /* 保護メモリプール領域の先頭番
                                 地 */
        VP          mppmb ;   /* 保護メモリプール管理領域の先頭番
                                 地 */
        /* 実装独自に他のフィールドを追加してもよい */
    } T_CMPP ;
```

保護メモリプール状態のパケット形式

```
    typedef struct t_rmpp {
        ID          wtskid ;  /* 保護メモリプールの待ち行列の先頭
                                 のタスクのID番号 */
        SIZE        fmppsz ;  /* 保護メモリプールの空き領域の合計
                                 サイズ（バイト数）*/
        UINT        fblksz ;  /* すぐに獲得可能な最大メモリブロッ
                                 クサイズ（バイト数）*/
        ACVCT       acvct ;   /* 保護メモリプールのアクセス許可ベ
                                 クタ */
        /* 実装独自に他のフィールドを追加してもよい */
    } T_RMPP ;
```

(14) 保護メールボックス機能

保護メールボックス生成情報のパケット形式

```
    typedef struct t_cmbp {
        ATR         mbpatr ;  /* 保護メールボックス属性 */
        UINT        mbpcnt ;  /* 入れることができるメッセージの
                                 数 */
        PRI         maxmpri ; /* 送信されるメッセージの優先度の最
                                 大値 */
        VP          mbpmb ;   /* 保護メールボックス管理領域の先頭
                                 番地 */
```

```
        /* 実装独自に他のフィールドを追加してもよい */
    } T_CMBP ;
```

保護メールボックス状態のパケット形式

```
    typedef struct t_rmbp {
        ID          wtskid ;    /* 保護メールボックスの待ち行列の先
                                   頭のタスクのID番号 */
        VP          blk ;       /* メッセージキューの先頭の保護メモ
                                   リブロックの先頭番地 */
        UINT        blksz ;     /* メッセージキューの先頭の保護メモ
                                   リブロックのサイズ */
        ACVCT       acvct ;     /* 保護メールボックスのアクセス許可
                                   ベクタ */
        /* 実装独自に他のフィールドを追加してもよい */
    } T_RMBP ;
```

6.5 追加／変更された定数とマクロ

この仕様で追加された定数とマクロは次の通り（機能コード，構成定数，構成マクロを除く）.

(1) 保護ドメインID

TDOM_SELF	0	自タスクの所属する保護ドメイン
TDOM_KERNEL	–1	カーネルドメイン
TDOM_NONE	–2	保護ドメインに所属しない

(2) アクセス許可パターン

ACPTN acptn = TACP (ID domid)

domid で指定される保護ドメインのみが操作／アクセスできることを示すアクセス許可パターン

TACP_KERNEL	カーネルドメインのみが操作／アクセスできることを示すアクセス許可パターン
TACP_SHARED	すべての保護ドメインが操作／アクセスできることを示すアクセス許可パターン

(3) アクセス許可ベクタ

ACVCT acvct = TACT_PRIVATE (ID domid)

すべての操作／アクセスが，domidで指定される保護ドメインのみに許可されることを示すアクセス許可ベクタ

TACT_KERNEL	すべての操作／アクセスが，カーネルドメインのみに許可されることを示すアクセス許可ベクタ

TACT_SHARED　すべての操作／アクセスが，すべての保護ドメインに許可されることを示すアクセス許可ベクタ

ACVCT acvct = TACT_PRW (ID domid)

メモリオブジェクトに対するすべての操作／アクセスが，domidで指定される保護ドメインにのみ許可されていることを示すアクセス許可ベクタ

ACVCT acvct = TACT_PRO (ID domid)

メモリオブジェクトに対する書込みアクセス以外の操作／アクセスが，domidで指定される保護ドメインにのみ許可されていることを示すアクセス許可ベクタ

TACT_SRW　メモリオブジェクトに対するすべての操作／アクセスが，すべての保護ドメインに許可されていることを示すアクセス許可ベクタ

TACT_SRO　メモリオブジェクトに対する書込みアクセス以外の操作／アクセスが，すべての保護ドメインに許可されていることを示すアクセス許可ベクタ

ACVCT acvct = TACT_SRPW (ID domid)

メモリオブジェクトに対する読出しアクセスがすべての保護ドメインに許可され，それ以外の操作／アクセスがdomidで指定される保護ドメインにのみ許可されていることを示すアクセス許可ベクタ

(4) アラインメントのチェックマクロ

BOOL align = ALIGN_VB (VP addr)

addrで指定された番地に対して，VB型のデータを書込み／読出しできるアラインメントになっている場合にTRUE，そうでない場合にFALSE

BOOL align = ALIGN_VH (VP addr)

addrで指定された番地に対して，VH型のデータを書込み／読出しできるアラインメントになっている場合にTRUE，そうでない場合にFALSE

BOOL align = ALIGN_VW (VP addr)

addrで指定された番地に対して，VW型のデータを書込み／読出しできるアラインメントになっている場合にTRUE，そうでない場合にFALSE

BOOL align = ALIGN_VD (VP addr)

addrで指定された番地に対して，VD型のデータを書込み／読出しできるアラインメントになっている場合にTRUE，そうでない場合にFALSE

BOOL align = ALIGN_VP (VP addr)

addrで指定された番地に対して，VP型のデータを書込み／読出しできるアラインメントになっている場合にTRUE，そうでない場合にFALSE

BOOL align = ALIGN_*TYPE* (VP addr)

addrで指定された番地に対して，*TYPE*型のデータを書込み／読出しできるアラインメントになっている場合にTRUE，そうでない場合にFALSE

(5) メモリオブジェクト属性

TA_RW	0x00	リード／ライト可能
TA_RO	0x01	リードオンリー
TA_CACHE	0x00	キャッシュ可能
TA_UNCACHE	0x02	キャッシュ不可

(6) オブジェクトの状態

TTW_MPP	0x10000	保護メモリブロックの獲得待ち状態
TTW_MBP	0x20000	保護メールボックスからの受信待ち状態

6.6 追加／変更された構成定数と構成マクロ

この仕様で追加された構成定数と構成マクロは次の通り.

(1) バージョン情報

TKERNEL_PXVER　保護機能拡張仕様のバージョン番号

(2) 必要なメモリ領域のサイズ

SIZE dtqmbsz = TSZ_DTQMB (UINT dtqcnt)

dtqcnt 個のデータを格納するのに必要なデータキュー管理領域のサイズ（バイト数）

SIZE mbxmbsz = TSZ_MBXMB (UINT mbxcnt, PRI maxmpri)

mbxcnt個のメッセージを入れることができ，送信されるメッセージの優先度の最大値がmaxmpriであるメールボックスに必要なメールボックス管理領域のサイズ（バイト数）

SIZE mbfmbsz = TSZ_MBFMB (UINT msgcnt, UINT msgsz)

サイズが msgsz バイトのメッセージを msgcnt 個格納するのに必要なメッセージバッファ管理領域のサイズ（目安のバイト数）

SIZE mpfmbsz = TSZ_MPFMB (UINT blkcnt, UINT blksz)

サイズがblkszバイトのメモリブロックをblkcnt個獲得できる固定長メモリプールの管理に必要な固定長メモリプール管理領域のサイズ（バイト数）

SIZE mppsz = TSZ_MPP (UINT blkcnt, UINT memsz)

サイズが memsz バイト（ないしはそれ以上のサイズ）のメモリブロックをblkcnt個割り付けるのに必要な保護メモリプール領域のサイズ(目安のバイト数)

SIZE mppmbsz = TSZ_MPPMB (SIZE mppsz)

サイズが mppsz バイトの保護メモリプール領域を管理するのに必要な保護メモリプール管理領域のサイズ（バイト数）

SIZE mbpmbsz = TSZ_MBPMB (UINT mbpcnt, PRI maxmpri)

mbpcnt個のメッセージを入れることができ，送信されるメッセージの優先度の最大値が maxmpri である保護メールボックスに必要な保護メールボックス管理領域のサイズ（バイト数）

6.7 機能コード一覧

	–0	–1	–2	–3
–0x01	予約	予約	予約	予約
–0x05	cre_tsk	del_tsk	act_tsk	can_act
–0x09	sta_tsk	ext_tsk	exd_tsk	ter_tsk
–0x0d	chg_pri	get_pri	ref_tsk	ref_tst
–0x11	slp_tsk	tslp_tsk	wup_tsk	can_wup
–0x15	rel_wai	sus_tsk	rsm_tsk	frsm_tsk
–0x19	dly_tsk	予約	def_tex	ras_tex
–0x1d	dis_tex	ena_tex	sns_tex	ref_tex
–0x21	cre_sem	del_sem	sig_sem	予約
–0x25	wai_sem	pol_sem	twai_sem	ref_sem
–0x29	cre_flg	del_flg	set_flg	clr_flg
–0x2d	wai_flg	pol_flg	twai_flg	ref_flg
–0x31	cre_dtq	del_dtq	予約	予約
–0x35	snd_dtq	psnd_dtq	tsnd_dtq	fsnd_dtq
–0x39	rcv_dtq	prcv_dtq	trcv_dtq	ref_dtq
–0x3d	cre_mbx	del_mbx	snd_mbx	予約
–0x41	rcv_mbx	prcv_mbx	trcv_mbx	ref_mbx
–0x45	cre_mpf	del_mpf	rel_mpf	予約
–0x49	get_mpf	pget_mpf	tget_mpf	ref_mpf
–0x4d	set_tim	get_tim	cre_cyc	del_cyc
–0x51	sta_cyc	stp_cyc	ref_cyc	予約
–0x55	rot_rdq	get_tid	get_did	予約
–0x59	loc_cpu	unl_cpu	dis_dsp	ena_dsp

–0x5d	sns_ctx	sns_loc	sns_dsp	sns_dpn
–0x61	ref_sys	sac_sys	予約	予約
–0x65	def_inh	cre_isr	del_isr	ref_isr
–0x69	dis_int	ena_int	chg_i*xx*	get_i*xx*
–0x6d	def_svc	def_exc	ref_cfg	ref_ver
–0x71	iact_tsk	iwup_tsk	irel_wai	iras_tex
–0x75	isig_sem	iset_flg	ipsnd_dtq	ifsnd_dtq
–0x79	irot_rdq	iget_tid	iloc_cpu	iunl_cpu
–0x7d	isig_tim	予約	予約	予約
–0x81	cre_mtx	del_mtx	unl_mtx	予約
–0x85	loc_mtx	ploc_mtx	tloc_mtx	ref_mtx
–0x89	cre_mbf	del_mbf	予約	予約
–0x8d	snd_mbf	psnd_mbf	tsnd_mbf	予約
–0x91	rcv_mbf	prcv_mbf	trcv_mbf	ref_mbf
–0x95	cre_por	del_por	cal_por	tcal_por
–0x99	acp_por	pacp_por	tacp_por	fwd_por
–0x9d	rpl_rdv	ref_por	ref_rdv	予約
–0xa1	cre_mpl	del_mpl	rel_mpl	予約
–0xa5	get_mpl	pget_mpl	tget_mpl	ref_mpl
–0xa9	cre_alm	del_alm	sta_alm	stp_alm
–0xad	ref_alm	予約	予約	予約
–0xb1	def_ovr	sta_ovr	stp_ovr	ref_ovr
–0xb5	ref_tim	sac_tim	予約	予約
–0xb9	att_mem	ata_mem	det_mem	prb_mem
–0xbd	ref_mem	sac_mem	予約	予約
–0xc1	acre_tsk	acre_sem	acre_flg	acre_dtq
–0xc5	acre_mbx	acre_mtx	acre_mbf	acre_por
–0xc9	acre_mpf	acre_mpl	acre_cyc	acre_alm
–0xcd	acre_isr	acre_mpp	acre_mbp	予約
–0xd1	予約	予約	予約	予約
–0xd5	予約	予約	予約	予約
–0xd9	予約	予約	予約	予約
–0xdd	予約	予約	予約	予約
–0xe1	実装独自のサービスコール			
–0xe5	実装独自のサービスコール			
–0xe9	実装独自のサービスコール			
–0xed	実装独自のサービスコール			
–0xf1	実装独自のサービスコール			
–0xf5	実装独自のサービスコール			

–0xf9	実装独自のサービスコール			
–0xfd	実装独自のサービスコール			
–0x101	cre_mpp	del_mpp	rel_mpp	予約
–0x105	get_mpp	pget_mpp	tget_mpp	ref_mpp
–0x109	cre_mbp	del_mbp	snd_mbp	予約
–0x10d	rcv_mbp	prcv_mbp	trcv_mbp	ref_mbp
–0x111	cra_tsk	cra_sem	cra_flg	cra_dtq
–0x115	cra_mbx	cra_mtx	cra_mbf	cra_por
–0x119	cra_mpf	cra_mpl	cra_cyc	cra_alm
–0x11d	cra_isr	cra_mpp	cra_mbp	予約
–0x121	acra_tsk	acra_sem	acra_flg	acra_dtq
–0x125	acra_mbx	acra_mtx	acra_mbf	acra_por
–0x129	acra_mpf	acra_mpl	acra_cyc	acra_alm
–0x12d	acra_isr	acra_mpp	acra_mbp	予約
–0x131	sac_tsk	sac_sem	sac_flg	sac_dtq
–0x135	sac_mbx	sac_mtx	sac_mbf	sac_por
–0x139	sac_mpf	sac_mpl	sac_cyc	sac_alm
–0x13d	sac_isr	sac_mpp	sac_mbp	予約