1. 主要な概念と共通定義

1.1. 仕様の位置付け

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

1.1.1. カーネルの機能セット

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

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

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

【TOPPERS/ASPカーネルにおける規定】

ASPカーネルは，保護機能対応カーネル，マルチプロセッサ対応カーネル，動的生成対応カーネルのいずれでもない【ASPS0001】．ただし，動的生成機能拡張パッケージを用いると，動的生成対応カーネルの機能の一部がサポートされる 【ASPS0002】．

【TOPPERS/FMPカーネルにおける規定】

FMPカーネルは，マルチプロセッサ対応カーネルであり，保護機能対応カーネル，動的生成対応カーネルではない【FMPS0001】．

【TOPPERS/HRP2カーネルにおける規定】

HRP2カーネルは，保護機能対応カーネルであり，マルチプロセッサ対応カーネル，動的生成対応カーネルではない【HRPS0001】．ただし，動的生成機能拡張パッケージを用いると，動的生成対応カーネルの機能の一部がサポートされる 【HRPS0009】．

【TOPPERS/SSPカーネルにおける規定】

SSPカーネルは，保護機能対応カーネル，マルチプロセッサ対応カーネル，動的生成対応カーネルのいずれでもない【SSPS0001】．

【μITRON4.0仕様，μITRON4.0/PX仕様との関係】

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

1.1.2. ターゲット非依存の規定とターゲット定義の規定

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

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

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

1.1.3. 想定するソフトウェア構成

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

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

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

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

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

Programming Interface）と呼ぶ．

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

【μITRON4.0仕様との関係】

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

1.1.4. 想定するハードウェア構成

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

a. メモリ番地は，常に同一のメモリを指すこと（オーバレイのように，異なるメモリを同一のメモリ番地でアクセスすることがないこと）【NGKI0001】．マルチプロセッサ対応カーネルにおいては，同一のメモリに対しては，各プロセッサから同一の番地でアクセスできること【NGKI0002】．

b. マルチプロセッサ対応カーネルにおいては，各プロセッサが同一の機械語命令を実行できること【NGKI0003】．

1.1.5. 想定するプログラミング言語

この仕様におけるAPI仕様は，ISO/IEC　9899:1990（以下，C90と呼ぶ）または　ISO/IEC　9899:1999（以下，C99と呼ぶ）に準拠したC言語を，フリースタンディング環境で用いることを想定して規定している【NGKI0004】．

ただし，C90の規定に加えて，以下のことを仮定している．

- 16ビットおよび32ビットの整数型があること【NGKI0005】
- ポインタが格納できるサイズの整数型があること【NGKI0006】

1.2. APIの構成要素とコンベンション

1.2.1. APIの構成要素

1. サービスコール

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

2. コールバック

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

3. 静的API

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

4. 構成マクロ

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

1.2.2. パラメータとリターンパラメータ

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

オブジェクトを生成するサービスコールなど，パラメータの数が多い場合やターゲット定義のパラメータを追加する可能性がある場合には，複数のパラメータを1つの構造体に入れ，その領域へのポインタをパラメータとして渡す【NGKI0007】．また，パラメータのサイズが大きい場合にも，パラメータを入れた領域へのポインタをパラメータとして渡す場合がある【NGKI0008】.

C言語APIでは，リターンパラメータは，関数の返値とするか，リターンパラメータを入れる領域へのポインタをパラメータとして渡すことで実現する【NGKI0009】．オブジェクトの状態を参照するサービスコールなど，リターンパラメータの数が多い場合やターゲット定義のリターンパラメータを追加する可能性がある場合には，複数のリターンパラメータを1つの構造体に入れて返すこととし，その領域へのポインタをパラメータとして渡す【NGKI0010】.

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

サービスコールやコールバックに，パケットを置く領域へのポインタやリターンパラメータを入れる領域へのポインタを渡す場合，別に規定がない限りは，サービスコールやコールバックの処理が完了した後は，それらの領域が参照されることはなく，別の目的に使用できる【NGKI0011】.

1.2.3. 返値とエラーコード

一部の例外を除いて，サービスコールおよびコールバックの返値は，処理が正常終了したかを表す符号付き整数とする．処理が正常終了した場合には，E_OK（＝0）または正の値が返るものとし，値の意味はサービスコールまたはコールバック毎に定める【NGKI0012】．処理が正常終了しなかった場合には，その原因を表す負の値が返る【

NGKI0013】．処理が正常終了しなかった原因を表す値 を，エラーコード（error code）と呼ぶ．

エラーコードは，いずれも負の値のメインエラーコードとサブエラーコードで 構成される【NGKI0014】．メインエラーコードとサブエラーコードからエラー コードを構成するマクロ（ERCD）と，エラーコードからメインエラーコードを 取り出すマクロ（MERCD），サブエラーコードを取り出すマクロ（SERCD）が用 意されている【NGKI0015】．

メインエラーコードの名称・意味・値は，カーネルとシステムサービスで共通 に定める（「TOPPERS共通エラーコード」の節を参照）【NGKI0016】．
サービスコールおよびコールバックの機能説明中の「E_XXXXXエラーとなる」ま たは「E_XXXXXエラーが返る」という記述は，メインエラーコードとして E_XXXXXが返ることを意味する．

サブエラーコードは，エラーの原因をより詳細に表すために用いる．カーネルはサブエラーコードを使用せず，サブエラーコードとして常に-1が返る 【NGKI0017】．サブエラーコードの名称・意味・値は，サブエラーコードを使 用するシステムサービスのAPI仕様において規定する【NGKI0018】．

サービスコールが負の値のエラーコード（警告を表すものを除く）を返した場合には，サービスコールによる副作用がないのが原則である【NGKI0019】．ただし，そのような実装ができない場合にはこの原則の例外とし，サービスコールの機能説明にその旨を記述する【NGKI0020】．

サービスコールが複数のエラーを検出するべき状況では，その内のいずれか1つのエラーを示すエラーコードが返る【NGKI0021】．

コールバックが複数のエラーを検出するべき状況では，その内のいずれか1つのエラーを示すエラーコードを返せばよい【NGKI0022】．

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

1.2.4. 機能コード

ソフトウェア割込みによりサービスコールを呼び出す場合などに用いるためのサービスコールを識別するための番号を，機能コード（function code）と呼ぶ．
機能コードは符号付きの整数値とし，カーネルのサービスコールには負の値を割り付け，拡張サービスコールには正の値を用いる【NGKI0023】．

1.2.5. ヘッダファイル

カーネルやシステムサービスを用いるために必要な定義を含むファイル．

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

レクティブが付加されている【NGKI0024】.

1.3. 主な概念

1.3.1. オブジェクトと処理単位

1. オブジェクト

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

   オブジェクトは，種類毎に，番号によって識別する【NGKI0025】．カーネルまたはシステムサービスで，オブジェクトに対して任意に識別番号を付与できる場合には，1から連続する正の整数値でオブジェクトを識別するのを原則とする【NGKI0026】．この場合に，オブジェクトの識別番号を，オブジェクトのID番号（ID number）と呼ぶ．そうでない場合，すなわちカーネルまたはシステムサービスの内部または外部からの条件によって識別番号が決まる場合には，オブジェクトの識別番号を，オブジェクト番号（object number）と呼ぶ．識別する必要のないオブジェクトには，識別番号を付与しない場合がある【NGKI0027】.

   オブジェクト属性（object attribute）は，オブジェクトの動作モードや初期状態を定めるもので，オブジェクトの登録時に指定する【NGKI0028】．オブジェクト属性にTA_XXXXが指定されている場合，そのオブジェクトを，TA_XXXX属性のオブジェクトと呼ぶ．複数の属性を指定する場合には，オブジェクト属性を渡すパラメータに，指定する属性値のビット毎論理和（C言語の"|"）を渡す【NGKI0029】．また，指定すべきオブジェクト属性がない場合には，TA_NULLを指定する【NGKI0030】.

2. 処理単位

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

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

   拡張情報（extended information）は，処理単位が呼び出される時にパラメータとして渡される情報で，処理単位の登録時に指定する【NGKI0031】．拡張情報は，カーネルやシステムサービスの動作には影響しない【NGKI0032】.

3. タスク

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

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

カーネルには，静的APIにより，少なくとも1つのタスクを登録しなければなら
ない．タスクが登録されていない場合には，コンフィギュレータがエラーを報 告する【NGKI0033】．

【補足説明】

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

4. ディスパッチとスケジューリング

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

ディスパッチが起こるべき状態（すなわち，スケジューリングによって，現在
実行しているタスクとは異なるタスクが，実行すべきタスクに決定されている
状態）となっても，何らかの理由でディスパッチを行わないことを，ディスパッチの保留（pend dispatching）という．ディスパッチを行わない理由が解除さ
れた時点で，ディスパッチが起こる【NGKI0034】．

5. 割込みとCPU例外

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

周辺デバイスからの割込み要求をプロセッサに伝える経路を遮断し，割込み要
求が受け付けられるのを抑止することを，割込みのマスク（mask interrupt）
または割込みの禁止（disable interrupt）という．マスクが解除された時点で，
まだ割込み要求が保持されていれば，その時点で割込み要求を受け付ける 【NGKI0035】．

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

【μITRON4.0仕様との関係】

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

6. タイムイベントとタイムイベントハンドラ

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

タイムイベントハンドラ（time event handler）と呼ぶ.

1.3.2. サービスコールとパラメータ

1. 優先順位と優先度

   優先順位（precedence）とは，処理単位の実行順序を説明するための仕様上の概念である．複数の処理単位が実行できる場合には，その中で最も優先順位の高い処理単位が実行される【NGKI0036】.

   優先度（priority）は，タスクなどの処理単位の優先順位や，メッセージなどの配送順序を決定するために，アプリケーションが処理単位やメッセージなどに与える値である．優先度は，符号付きの整数型であるPRI型で表し，1から連続した正の値を用いるのを原則とする【NGKI0037】．優先度は，値が小さいほど優先度が高い（すなわち，先に実行または配送される）ものとする 【NGKI0038】.

2. システム時刻と相対時間

   カーネルが管理する時刻を，システム時刻（system time）と呼ぶ．システム時刻は，符号無しの整数型であるSYSTIM型で表し，単位はミリ秒とする【NGKI0039】．システム時刻は，タイムティック（time tick）を通知するためのタイマ割込みが発生する毎に更新される【NGKI0040】.

   イベントを発生させる時刻を指定する場合には，基準時刻（base time）からの 相対時間（relative time）によって指定する【NGKI0041】．基準時刻は，別に規定がない限りは，相対時間を指定するサービスコールを呼び出した時刻とな る【NGKI0042】.

   相対時間は，符号無しの整数型であるRELTIM型で表し，単位はシステム時刻と同一，すなわちミリ秒とする【NGKI0043】．相対時間には，少なくとも，16ビットの符号無しの整数型（uint16_t型）に格納できる任意の値を指定することができるが，RELTIM型（uint_t型に定義される）に格納できる任意の値を指定できるとは限らない【NGKI0044】．相対時間に指定できる最大値は，構成マクロ TMAX_RELTIM に定義されている【NGKI0045】.

   イベントを発生させる時刻を相対時間で指定した場合，イベントの処理が行われるのは，基準時刻から相対時間によって指定した以上の時間が経過した後となる【NGKI0046】．ただし，基準時刻を定めるサービスコールを呼び出した時に，タイムティックを通知するためのタイマ割込みがマスクされている場合（タイマ割込みより優先して実行される割込み処理が実行されている場合を含む）は，相対時間によって指定した以上の時間が経過した後となることは保証 されない【NGKI0047】.

   イベントが発生する時刻を参照する場合には，基準時刻からの相対時間として返される【NGKI0048】．基準時刻は，相対時間を返すサービスコールを呼び出 した時刻となる【NGKI0049】.

   イベントが発生する時刻が相対時間で返された場合，イベントの処理が行われるのは，基準時刻から相対時間として返された以上の時間が経過した後となる 【

NGKI0050】．ただし，相対時間を返すサービスコールを呼び出した時に，タイムティックを通知するためのタイマ割込みがマスクされている場合（タイマ割込みより優先して実行される割込み処理が実行されている場合を含む）は，相対時間として返された以上の時間が経過した後となることは保証されない 【NGKI0051】．

【補足説明】

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

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

【μITRON4.0仕様との関係】

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

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

3. タイムアウトとポーリング

サービスコールの中で待ち状態が指定した時間以上継続した場合に，サービスコールの処理を取りやめて，サービスコールからリターンすることを，タイムアウト（timeout）という．タイムアウトしたサービスコールからは，E_TMOUT エラーが返る【NGKI0052】．

タイムアウトを起こすまでの時間（タイムアウト時間）は，符号付きの整数型である TMO型で表し，単位はシステム時刻と同一，すなわちミリ秒とする【NGKI0053】．タイムアウト時間に正の値を指定した場合には，タイムアウトを起こすまでの相対時間を表す【NGKI0054】．すなわち，タイムアウトの処理が行われるのは，サービスコールを呼び出してから指定した以上の時間が経過 した後となる．

ポーリング（polling）を行うサービスコールとは，サービスコールの中で待ち状態に遷移すべき状況になった場合に，サービスコールの処理を取りやめてリターンするサービスコールのことをいう．ここで，サービスコールの処理を取りやめてリターンすることを，ポーリングに失敗したという．ポーリングに失敗したサービスコールからは，E_TMOUTエラーが返る【NGKI0055】．

ポーリングを行うサービスコールでは，待ち状態に遷移することはないのが原則である【NGKI0056】．そのため，ポーリングを行うサービスコールは，ディスパッチ保留状態であっても呼び出せる【NGKI0057】．ただし，サービスコールの中で待ち状態に遷移する状況が複数ある場合，ある状況でポーリング動作をしても，他の状況では待ち状態に遷移する場合がある．このような場合の振

舞いは，該当するサービスコール毎に規定する【NGKI0058】．

タイムアウト付きのサービスコールは，別に規定がない限りは，タイムアウト　　時間にTMO_POL（＝0）を指定した場合にはポーリングを行い，TMO_FEVR（＝-1）
を指定した場合にはタイムアウトを起こさないものとする【NGKI0059】．

【補足説明】

［NGKI0019］の原則より，サービスコールがタイムアウトした場合やポーリングに失敗した場合には，サービスコールによる副作用がないのが原則である．ただし，そのような実装ができない場合にはこの原則の例外とし，どのような副作用があるかをサービスコール毎に規定する．

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

【μITRON4.0仕様との関係】

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

【仕様決定の理由】

ディスパッチ保留状態において，ポーリングを行うサービスコールを呼び出せる場合があるのに対して，タイムアウト付きのサービスコールをタイムアウト　　時間をTMO_POLとして呼び出すとエラーになるのは，割込み優先度マスクが全解除でない状態やディスパッチ禁止状態では，自タスクを広義の待ち状態に遷移させる可能性のあるサービスコール（タイムアウト付きのサービスコールはこれに該当）を呼び出すことはできないという原則［NGKI0175］と［NGKI0179］があるためである．

4. ノンブロッキング

サービスコールの中で待ち状態に遷移すべき状況になった時，サービスコールの処理を継続したままサービスコールからリターンする場合，そのサービスコールをノンブロッキング（non-blocking）という．処理を継続したままリターンする場合，サービスコールからはE_WBLKエラーが返る【NGKI0060】．E_WBLKは警告を表すエラーコードであり，サービスコールによる副作用がないという原則は適用されない【NGKI0061】．

サービスコールからE_WBLKエラーが返った場合には，サービスコールの処理は継続しているため，サービスコールに渡したパラメータまたはリターンパラメータを入れる領域はまだ参照される可能性があり，別の目的に使用することはで　　きない【NGKI0062】．継続している処理が完了した場合や，何らかの理由で処理が取りやめられた場合には，コールバックを呼び出すなどの方法で，サービスコールを呼び出したソフトウェアに通知するものとする【NGKI0063】．

ノンブロッキングの指定は，タイムアウト時間にTMO_NBLK（＝-2）を指定することによって行う【NGKI0064】．ノンブロッキングの指定を行えるサービスコールは，指定した場合の振舞いをサービスコール毎に規定する【NGKI0065】．

【補足説明】

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

1.3.3. 保護機能

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

1. アクセス保護

   保護機能対応カーネルは，処理単位が，許可されたカーネルオブジェクトに対して，許可された種別のアクセスを行うことのみを許し，それ以外のアクセスを防ぐアクセス保護機能を提供する【NGKI0066】．

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

2. メモリオブジェクト

   保護機能対応カーネルにおいては，メモリ領域をカーネルオブジェクトとして扱い，アクセス保護の対象とする【NGKI0067】．カーネルがアクセス保護の対象とする連続したメモリ領域を，メモリオブジェクト（memory object）と呼ぶ．メモリオブジェクトは，互いに重なりあうことはない【NGKI0068】．

   メモリオブジェクトは，その先頭番地によって識別する【NGKI0069】．言い換えると，先頭番地がオブジェクト番号となる．

   メモリオブジェクトの先頭番地とサイズには，ターゲットハードウェアでメモリ保護が実現できるように，ターゲット定義の制約が課せられる【NGKI0070】．

3. 保護ドメイン

   保護機能を提供するために用いるカーネルオブジェクトの集合を，保護ドメイン（protection domain）と呼ぶ．保護ドメインは，保護ドメインIDと呼ぶID番 号によって識別する【NGKI0071】．

   カーネルオブジェクトは，たかだか1つの保護ドメインに属する．処理単位は，いずれか1つの保護ドメインに属さなければならないのに対して，それ以外のカーネルオブジェクトは，いずれの保護ドメインにも属さないことができる【NGKI0072】．いずれの保護ドメインにも属さないカーネルオブジェクトを，無所属のカーネルオブジェクト（independent kernel object）と呼ぶ．

   処理単位がカーネルオブジェクトにアクセスできるかどうかは，処理単位が属

する保護ドメインにより決まるのが原則である【NGKI0073】．すなわち，カーネルオブジェクトに対するアクセス権は，処理単位ではなく，保護ドメイン単位で管理される．このことから，ある保護ドメインに属する処理単位がアクセスできることを，単に，その保護ドメインからアクセスできるという．

ただし，タスクのユーザスタック領域は，ターゲット定義での変更がない限りは，そのタスク（とカーネルドメインに属する処理単位）のみがアクセスできる（「2.11.6 ユーザタスクのユーザスタック領域」の節を参照）【NGKI0074】．これは，［NGKI0073］の原則の例外となっている．

デフォルトでは，保護ドメインに属するカーネルオブジェクトは，同じ保護ドメイン（とカーネルドメイン）のみからアクセスできる【NGKI0075】．また，無所属のカーネルオブジェクトは，すべての保護ドメインからアクセスできる 【NGKI0076】．

4. カーネルドメインとユーザドメイン

システムには，カーネルドメイン（kernel domain）と呼ばれる保護ドメインが1つ存在する【NGKI0077】．カーネルドメインに属する処理単位は，プロセッサの特権モードで実行される【NGKI0078】．また，すべてのカーネルオブジェクトに対して，すべての種別のアクセスを行うことが許可される【NGKI0079】．この仕様で，「ある保護ドメイン（またはタスク）のみからアクセスできる」といった場合でも，カーネルドメインドメインからはアクセスすることができ る．

カーネルドメイン以外の保護ドメインを，ユーザドメイン（user domain）と呼ぶ．ユーザドメインに属する処理単位は，プロセッサの非特権モードで実行される【NGKI0080】．また，どのカーネルオブジェクトに対してどの種別のアクセスを行えるかを制限することができる【NGKI0081】．

ユーザドメインには，1から連続する正の整数値の保護ドメインIDが付与される【NGKI0082】．カーネルドメインの保護ドメインIDは，TDOM_KERNEL（=-1）で ある【NGKI0083】．

この仕様では，システムに登録できるユーザドメインの数は，32個以下に制限する【NGKI0084】．これを超える数のユーザドメインを登録した場合には，コンフィギュレータがエラーを報告する【NGKI0085】．

【補足説明】

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

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

【μITRON4.0/PX仕様との関係】

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

できる保護ドメインである．

5. システムタスクとユーザタスク

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

   【補足説明】

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

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

6. アクセス許可パターン

   あるカーネルオブジェクトに対するある種別のアクセスが，どの保護ドメインに属する処理単位に許可されているかを表現するビットパターンを，アクセス許可パターン（access permission pattern）と呼ぶ．アクセス許可パターンの各ビットは，1つのユーザドメインに対応する【NGKI0086】．カーネルドメインには，すべてのアクセスが許可されているため，カーネルドメインに対応するビットは用意されていない．

   アクセス許可パターンは，符号無し32ビット整数に定義されるデータ型（ACPTN）で保持し，値が1のビットに対応するユーザドメインにアクセスが許可されていることを表す【NGKI0087】．そのため，2つのアクセス許可パターンのビット毎論理和（C言語の"|"）を求めることで，アクセスを許可されているユーザドメインの和集合（union）を得ることができる．また，2つのアクセス許可パターンのビット毎論理積（C言語の"&"）を求めることで，アクセスを許可されているユーザドメインの積集合（intersection）を得ることができる．

   アクセス許可パターンの指定に用いるために，指定したユーザドメインのみにアクセスを許可することを示すアクセス許可パターンを構成するマクロ（TACP）が用意されている【NGKI0088】．また，カーネルドメインのみにアクセスを許可することを示すアクセス許可パターンを表す定数（TACP_KERNEL）と，すべての保護ドメインにアクセスを許可することを示すアクセス許可パターンを表す定数（TACP_SHARED）が用意されている【NGKI0089】．

7. アクセス許可ベクタ

   カーネルオブジェクトに対するアクセスは，カーネルオブジェクトの種類毎に，通常操作1，通常操作2，管理操作，参照操作の4つの種別に分類されている【NGKI0090】．あるカーネルオブジェクトに対する4つの種別のアクセスに関するアクセス許可パターンをひとまとめにしたものを，アクセス許可ベクタ（access permission vector）と呼び，次のように定義されるデータ型（ACVCT）で保持する【NGKI0091】．

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

【補足説明】

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

【μITRON4.0/PX仕様との関係】

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

8. サービスコールの呼出し方法

保護機能対応カーネルでは，サービスコールは，ソフトウェア割込みによって呼び出すのが基本である．サービスコール呼出しを通常の方法で記述した場合，ソフトウェア割込みによって呼び出すコードが生成される【NGKI0092】．

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

カーネルドメインに属する処理単位が実行する関数のみを含んだソースファイルでは，カーネルヘッダファイル（kernel.h）をインクルードする前に，TOPPERS_SVC_CALLをマクロ定義することで，サービスコール呼出しを通常の方法で記述した場合に，関数呼出しによって呼び出すコードが生成される 【NGKI0093】．

また，カーネルドメインに属する処理単位が実行する関数と，ユーザドメインに属する処理単位が実行する関数の両方を含んだソースファイルでは，関数呼出しによってサービスコールを呼び出すための名称を作るマクロ（SVC_CALL）を用いることで，関数呼出しによって呼び出すコードが生成される 【NGKI0094】．例えば，act_tskを関数呼出しによって呼び出す場合には，次の ように記述すればよい．

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

【補足説明】

拡張サービスコールを，関数呼出しによって呼び出す方法は用意されていない．カーネルドメインに属する処理単位が，関数呼出しによって，拡張サービスコー

ルとして登録した関数を呼び出すことはできるが，その場合には，処理単位が呼び出した通常の関数であるとみなされ，拡張サービスコールであるとは扱わ れない．

9. ユーザドメインから行える処理に対する制限

   ユーザドメインに属する処理単位が，システムの重要な処理に悪影響を及ぼすのを防ぐために，ユーザドメインから行える処理に対して制限を設ける機能が用意されている．具体的には，ユーザドメインに属する処理単位が，タスクのベース優先度を変更する際に，指定できるタスク優先度を制限することができ る．

   この機能を実現するために，各ユーザドメインは次の情報を持つ【NGKI0531】．

   - 指定できる最高のタスク優先度

     なお，カーネルドメインに対しては，制限を設ける機能を用意していない．すなわち，カーネルドメインに属する処理単位は，すべてのタスク優先度を使うことができる【NGKI0532】．

1.3.4. マルチプロセッサ対応

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

1. クラス

   マルチプロセッサに対応するために用いるカーネルオブジェクトの集合を，クラス（class）と呼ぶ．クラスは，クラスIDと呼ぶID番号によって識別する 【NGKI0095】．

   カーネルオブジェクトは，いずれか1つのクラスに属するのが原則である【NGKI0096】．カーネルオブジェクトが属するクラスは，オブジェクトの登録時に決定し，登録後に変更することはできない【NGKI0097】．

   【補足説明】

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

   ［NGKI0096］の原則に関わらず，以下のオブジェクトはいずれのクラスにも属 さない．

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

     マルチプロセッサ対応でないカーネルは，カーネルによって規定された1つのク

ラスのみをサポートしているとみなすこともできる.

2. プロセッサ

たかだか1つの処理単位のみを同時に実行できるハードウェアの単位を，プロセッサ（processor）と呼ぶ．プロセッサは，プロセッサIDと呼ぶID番号によって識 別する【NGKI0098】.

+ 複数のプロセッサを持つシステム構成をマルチプロセッサ（multiprocessor）と呼び，同時に複数の処理単位を実行することができる【NGKI0099】.

+ システムの初期化時と終了時に特別な役割を果たすプロセッサを，マスタプロセッサ（master processor）と呼び，システムに1つ存在する【NGKI0100】．どのプロセッサをマスタプロセッサとするかは，ターゲット定義である【NGKI0101】．マスタプロセッサ以外のプロセッサを，スレーブプロセッサ（slave processor）と呼ぶ．なお，カーネル動作状態では，マスタプロセッサとスレーブプロセッサの振舞いに違いはない【NGKI0102】.

1. 処理単位の割付けとマイグレーション

   処理単位は，後述のマイグレーションが発生しない限りは，いずれか1つのプロセッサに割り付けられて実行される【NGKI0103】．処理単位を実行するプロセッサを，割付けプロセッサと呼ぶ．また，処理単位が登録時に割り付けられるプロセッサを，初期割付けプロセッサと呼ぶ.

   処理単位によっては，処理単位の登録後に，割付けプロセッサを変更することが可能である【NGKI0104】．処理単位の登録後に割付けプロセッサを変更することを，処理単位のマイグレーション（migration）と呼ぶ.

   割付けプロセッサを変更できる処理単位に対しては，処理単位を割り付けることができるプロセッサ（これを，割付け可能プロセッサと呼ぶ）を制限することができる【NGKI0105】.

2. クラスの持つ属性とカーネルオブジェクト

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

   各クラスが持ち，それに属するカーネルオブジェクトに適用される属性は，次の通りである【NGKI0106】.

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

- その他の管理情報（ロック単位など）

使用できるクラスのID番号とその属性は，ターゲット定義である【NGKI0107】.

【仕様決定の理由】

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

3. ローカルタイマ方式とグローバルタイマ方式

システム時刻の管理方式として，プロセッサ毎にシステム時刻を持つローカルタイマ方式と，システム全体で1つのシステム時刻を持つグローバルタイマ方式 の2つの方式がある．どちらの方式を用いることができるかは，ターゲット定義 である【NGKI0108】.

ローカルタイマ方式では，プロセッサ毎のシステム時刻は，それぞれのプロセッサが更新する【NGKI0109】．異なるプロセッサのシステム時刻を同期させる機能は，カーネルでは用意しない.

グローバルタイマ方式では，システム中の1つのプロセッサがシステム時刻を更 新する【NGKI0110】．これを，システム時刻管理プロセッサと呼ぶ．どのプロセッサをシステム時刻管理プロセッサとするかは，ターゲット定義である 【NGKI0111】.

【補足説明】

システム時刻管理プロセッサが，マスタプロセッサと一致している必要はない.

【未決定事項】

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

1.3.5. その他

1. オブジェクトモジュール

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

2. メモリリージョン

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

メモリリージョンは，文字列によって識別する【NGKI0112】．メモリリージョンを識別する文字列を，メモリリージョン名と呼ぶ.

【補足説明】

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

3. 標準のセクション

コンパイラに特別な指定をしない場合に出力するセクションを，標準のセクション（standard sections）と呼ぶ．コンパイラが出力しないセクションの中で，ターゲット定義のものを，標準のセクションと扱う場合もある【NGKI0113】．

4. 保護ドメイン毎の標準セクション

保護機能対応カーネルにおいては，保護ドメイン毎に，標準のセクションを配置するためのセクションが登録される【NGKI0114】．また，無所属の標準のセクションを配置するためのセクションが登録される【NGKI0115】．これらのセクションを，保護ドメイン毎の標準セクションと呼ぶ（standards for each protection domain）．保護ドメイン毎の標準セクションのセクション名は，ターゲット定義で別に規定がない限りは，標準のセクション名と保護ドメイン名（カーネルドメインの場合は"kernel"，無所属の場合は"shared"）を"_"でつないだものとする【NGKI0116】．例えば，カーネルドメインの".text" セクションのセクション名は，".text_kernel" とする．

1.4. 処理単位の種類と実行順序

1.4.1. 処理単位の種類

カーネルが実行を制御する処理単位の種類は次の通りである【NGKI0117】．

```
(a) タスク
    (a.1) タスク例外処理ルーチン
(b) 割込みハンドラ
    (b.1) 割込みサービスルーチン
    (b.2) タイムイベントハンドラ
(c) CPU例外ハンドラ
(d) 拡張サービスコール
(e) 初期化ルーチン
(f) 終了処理ルーチン
```

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

【TOPPERS/ASPカーネルにおける規定】

ASPカーネルでは，オーバランハンドラと拡張サービスコールをサポートしていない【ASPS0003】．ただし，オーバランハンドラ機能拡張パッケージを用いる

と，オーバランハンドラ機能を追加することができる【ASPS0004】．

【TOPPERS/FMPカーネルにおける規定】

FMPカーネルでは，オーバランハンドラと拡張サービスコールをサポートしてい ない【FMPS0002】．

【TOPPERS/SSPカーネルにおける規定】

SSPカーネルでは，タスク例外処理ルーチン，タイムイベントハンドラ，拡張サービスコールをサポートしていない【SSPS0002】．

1.4.2. 処理単位の実行順序

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

タスクの優先順位は，ディスパッチャの優先順位よりも低い【NGKI0118】．タスク間では，高い優先度を持つ方が優先順位が高く，同じ優先度を持つタスク間では，先に実行できる状態となった方が優先順位が高い【NGKI0119】．詳し くは，「2.6.3 タスクのスケジューリング規則」の節を参照すること．

タスク例外処理ルーチンの優先順位は，例外が要求されたタスクと同じであるが，タスクよりも先に実行される【NGKI0120】．

割込みハンドラの優先順位は，ディスパッチャの優先順位よりも高い 【NGKI0121】．割込みハンドラ間では，高い割込み優先度を持つ方が優先順位が高く，同じ割込み優先度を持つ割込みハンドラ間では，先に実行開始された方が優先順位が高い【NGKI0122】．同じ割込み優先度を持つ割込みハンドラ間での実行開始順序は，この仕様では規定しない．詳しくは，「2.7.2 割込み優 先度」の節を参照すること．

割込みサービスルーチンとタイムイベントハンドラの優先順位は，それを呼び出す割込みハンドラと同じである【NGKI0123】．

CPU例外ハンドラの優先順位は，CPU例外がタスクまたはタスク例外処理ルーチンで発生した場合には，ディスパッチャの優先順位と同じであるが，ディスパッチャよりも先に実行される【NGKI0124】．CPU例外がその他の処理単位で発生し た場合には，CPU例外ハンドラの優先順位は，その処理単位の優先順位と同じであるが，その処理単位よりも先に実行される【NGKI0125】．

拡張サービスコールの優先順位は，それを呼び出した処理単位と同じであるが，それを呼び出した処理単位よりも先に実行される【NGKI0126】．

初期化ルーチンは，カーネルの動作開始前に，システムコンフィギュレーションファイル中に初期化ルーチンを登録する静的APIを記述したのと同じ順序で実 行される【NGKI0127】．終了処理ルーチンは，カーネルの動作終了後に，終了 処理ルーチンを登録する静的APIを記述したのと逆の順序で実行される 【NGKI0128】．

マルチプロセッサ対応カーネルでは，初期化ルーチンには，クラスに属さないグローバル初期化ルーチンと，クラスに属するローカル初期化ルーチンがある【NGKI0129】．グローバル初期化ルーチンがマスタプロセッサで実行された後に，各プロセッサでローカル初期化ルーチンが実行される【NGKI0130】．また，終了処理ルーチンには，クラスに属さないグローバル終了処理ルーチンと，クラスに属するローカル終了処理ルーチンがある【NGKI0131】．ローカル終了処理ルーチンが各プロセッサで実行された後に，マスタプロセッサでグローバル終了処理ルーチンが実行される【NGKI0132】．

【仕様決定の理由】

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

1.4.3. カーネル処理の不可分性

カーネルのサービスコール処理やディスパッチャ，割込みハンドラとCPU例外ハンドラの入口処理と出口処理などのカーネル処理は不可分に実行されるのが基本である．実際には，カーネル処理の途中でアプリケーションが実行される場合はあるが，アプリケーションがサービスコールを用いて観測できる範囲で，カーネル処理が不可分に実行された場合と同様に振る舞うのが原則である【NGKI0133】．これを，カーネル処理の不可分性という．

ただし，マルチプロセッサ対応カーネルにおいては，カーネル処理が実行されているプロセッサ以外のプロセッサから，カーネル処理の途中の状態が観測できる場合がある．具体的には，1つのサービスコールにより複数のオブジェクトの状態が変化する場合に，一部のオブジェクトの状態のみが変化し，残りのオブジェクトの状態が変化していない過渡的な状態が観測できる場合がある 【NGKI0134】．

【補足説明】

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

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

1.4.4. 処理単位を実行するプロセッサ

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

タスク，周期ハンドラ，アラームハンドラは，登録時に，属するクラスの初期割付けプロセッサに割り付けられる【NGKI0135】．また，割付けプロセッサを変更するサービスコール（mact_tsk／imact_tsk，mig_tsk，msta_cyc，msta_alm／imsta_alm）によって，割付けプロセッサを，クラスの割付け可能プロセッサのいずれかに変更することができる【NGKI0136】．

割込みハンドラ，CPU例外ハンドラ，ローカル初期化ルーチン，ローカル終了処理ルーチンは，属するクラスの初期割付けプロセッサで実行される【NGKI0137】．クラスの割付け可能プロセッサの情報は用いられない．

割込みサービスルーチンは，属するクラスの割付け可能プロセッサのいずれか（オプション設定によりすべて）で実行される【NGKI0138】．クラスの初期割付けプロセッサの情報は用いられない．

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

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

オーバランハンドラ，拡張サービスコール，グローバル初期化ルーチン，グローバル終了処理ルーチンは，いずれのクラスにも属さない【NGKI0139】．オーバランハンドラは，オーバランを起こしたタスクの割付けプロセッサによって実行される【NGKI0140】．拡張サービスコールは，それを呼び出した処理単位の割付けプロセッサによって実行される【NGKI0141】．グローバル初期化ルーチンとグローバル終了処理ルーチンは，マスタプロセッサによって実行される 【NGKI0142】．

1.5. システム状態とコンテキスト

1.5.1. カーネル動作状態と非動作状態

カーネルの初期化が完了した後，カーネルの終了処理が開始されるまでの間を，カーネル動作状態と呼ぶ．それ以外の状態，すなわちカーネルの初期化完了前（初期化ルーチンの実行中を含む）と終了処理開始後（終了処理ルーチンの実行中を含む）を，カーネル非動作状態と呼ぶ．プロセッサは，カーネル動作状態かカーネル非動作状態のいずれかの状態を取る【NGKI0143】．

カーネル非動作状態では，原則として，NMIを除くすべての割込みがマスクされ る【NGKI0144】．

カーネル非動作状態では，システムインタフェースレイヤのAPIとカーネル非動作状態を参照するサービスコール（sns_ker）のみを呼び出すことができる【NGKI0145】．カーネル非動作状態で，その他のサービスコールを呼び出した場合の動作は，保証されない【NGKI0146】．

マルチプロセッサ対応カーネルでは，プロセッサ毎に，カーネル動作状態かカーネル非動作状態のいずれかの状態を取る【NGKI0147】．

1.5.2. タスクコンテキストと非タスクコンテキスト

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

カーネル動作状態において，処理単位が実行されるコンテキストは，タスクコンテキストと非タスクコンテキストに分類される【NGKI0148】．

タスク（タスク例外処理ルーチンを含む）が実行されるコンテキストは，タスクコンテキストに分類される【NGKI0149】．また，タスクコンテキストから呼び出した拡張サービスコールが実行されるコンテキストは，タスクコンテキストに分類される【NGKI0150】．

割込みハンドラ（割込みサービスルーチンおよびタイムイベントハンドラを含む）とCPU例外ハンドラが実行されるコンテキストは，非タスクコンテキストに分類される【NGKI0151】．また，非タスクコンテキストから呼び出した拡張サービスコールが実行されるコンテキストは，非タスクコンテキストに分類される 【NGKI0152】．

タスクコンテキストで実行される処理単位は，別に規定がない限り，タスクのスタック領域を用いて実行される【NGKI0153】．非タスクコンテキストで実行される処理単位は，別に規定がない限り，非タスクコンテキスト用スタック領域を用いて実行される【NGKI0154】．

タスクコンテキストからは，非タスクコンテキスト専用のサービスコールを呼び出すことはできない【NGKI0155】．逆に，非タスクコンテキストからは，タスクコンテキスト専用のサービスコールを呼び出すことはできない【NGKI0156】．いずれも，呼び出した場合にはE_CTXエラーとなる【NGKI0157】．

1.5.3. カーネルの振舞いに影響を与える状態

カーネル動作状態において，プロセッサは，カーネルの振舞いに影響を与える状態として，次の状態を持つ【NGKI0158】.

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

これらの状態は，それぞれ独立な状態である．すなわち，プロセッサは上記の状態の任意の組合せを取ることができ，それぞれの状態を独立に変化させるこ とができる【NGKI0159】.

1.5.4. 全割込みロック状態と全割込みロック解除状態

プロセッサは，NMIを除くすべての割込みをマスクするための全割込みロックフラグを持つ【NGKI0160】．全割込みロックフラグがセットされた状態を全割込みロック状態，クリアされた状態を全割込みロック解除状態と呼ぶ．すなわち，全割込みロック状態では，NMIを除くすべての割込みがマスクされる.

全割込みロック状態では，システムインタフェースレイヤのAPIとカーネル非動作状態を参照するサービスコール（sns_ker），カーネルを終了するサービスコール（ext_ker）のみを呼び出すことができる【NGKI0161】．全割込みロック状態で，その他のサービスコール（拡張サービスコールを含む）を呼び出した場合の動作は，保証されない【NGKI0162】．また，全割込みロック状態で，実行中の処理単位からリターンしてはならない．リターンした場合の動作は保証され ない【NGKI0164】.

マルチプロセッサ対応カーネルでは，プロセッサ毎に，全割込みロックフラグを持つ【NGKI0165】．すなわち，プロセッサ毎に，全割込みロック状態か全割込みロック解除状態のいずれかの状態を取る.

1.5.5. CPUロック状態とCPUロック解除状態

プロセッサは，カーネル管理の割込み（「2.7.7 カーネル管理外の割込み」の節を参照）をすべてマスクするためのCPUロックフラグを持つ【NGKI0166】.
CPUロックフラグがセットされた状態をCPUロック状態，クリアされた状態をCPUロック解除状態と呼ぶ．CPUロック状態では，すべてのカーネル管理の割込みがマスクされ，ディスパッチが保留される【NGKI0167】.

CPUロック状態で呼び出すことができるサービスコールは次の通り【NGKI0168】.

- システムインタフェースレイヤのAPI
- loc_cpu／iloc_cpu，unl_cpu／iunl_cpu
- unl_spn／iunl_spn（マルチプロセッサ対応カーネルのみ）
- dis_int，ena_int

- sns_yyy
- xsns_yyy（CPU例外ハンドラからのみ）
- get_utm
- ext_tsk，ext_ker
- prb_mem（保護機能対応カーネルのみ）
- cal_svc（保護機能対応カーネルのみ）

CPUロック状態で，その他のサービスコールを呼び出した場合には，E_CTXエラー となる【NGKI0169】.

マルチプロセッサ対応カーネルでは，プロセッサ毎に，CPUロックフラグを持つ 【NGKI0170】．すなわち，プロセッサ毎に，CPUロック状態かCPUロック解除状 態のいずれかの状態を取る.

【補足説明】

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

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

1.5.6. 割込み優先度マスク

プロセッサは，割込み優先度を基準に割込みをマスクするための割込み優先度 マスクを持つ【NGKI0171】．割込み優先度マスクがTIPM_ENAALL（＝0）の時は，いずれの割込み要求もマスクされない【NGKI0172】．この状態を割込み優先度マスク全解除状態と呼ぶ．割込み優先度マスクがTIPM_ENAALL（＝0）以外の時は，割込み優先度マスクと同じかそれより低い割込み優先度を持つ割込みはマスクされ，ディスパッチは保留される【NGKI0173】．この状態を割込み優先度マスクが全解除でない状態と呼ぶ.

割込み優先度マスクが全解除でない状態では，別に規定がない限りは，自タスクを広義の待ち状態に遷移させる可能性のあるサービスコールを呼び出すことはできない．呼び出した場合には，E_CTXエラーとなる【NGKI0175】.

マルチプロセッサ対応カーネルでは，プロセッサ毎に，割込み優先度マスクを 持つ【NGKI0176】.

1.5.7. ディスパッチ禁止状態とディスパッチ許可状態

プロセッサは，ディスパッチを保留するためのディスパッチ禁止フラグを持つ 【NGKI0177】．ディスパッチ禁止フラグがセットされた状態をディスパッチ禁止状態，クリアされた状態をディスパッチ許可状態と呼ぶ．すなわち，ディスパッチ禁止状態では，ディスパッチは保留される.

ディスパッチ禁止状態では，別に規定がない限りは，自タスクを広義の待ち状態に遷移させる可能性のあるサービスコールを呼び出すことはできない．呼び出した場合には，E_CTXエラーとなる【NGKI0179】．

マルチプロセッサ対応カーネルでは，プロセッサ毎に，ディスパッチ禁止フラグを持つ【NGKI0180】．すなわち，プロセッサ毎に，ディスパッチ禁止状態かディスパッチ許可状態のいずれかの状態を取る．

【補足説明】

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

1.5.8. ディスパッチ保留状態

非タスクコンテキストの実行中，CPUロック状態，割込み優先度マスクが全解除でない状態，ディスパッチ禁止状態では，ディスパッチが保留される【NGKI0181】．これらの状態を総称して，ディスパッチ保留状態と呼ぶ．

マルチプロセッサ対応カーネルでは，プロセッサ毎に，ディスパッチ保留状態かそうでない状態のいずれかの状態を取る【NGKI0182】．

【補足説明】

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

1.5.9. カーネル管理外の状態

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

カーネル管理外の状態では，システムインタフェースレイヤのAPIとsns_ker，ext_kerのみ（カーネル管理外のCPU例外ハンドラからは，それに加えてxsns_dpnとxsns_xpn）を呼び出すことができ，その他のサービスコールを呼び出すことはできない【NGKI0543】．カーネル管理外の状態から，その他のサービスコールを呼び出した場合の動作は，保証されない【NGKI0544】．

カーネル管理外の状態では，少なくとも，カーネル管理の割込みはマスクされている【NGKI0545】．カーネル管理外の割込み（の一部）もマスクされている場合もある【NGKI0546】．保護機能対応カーネルでは，カーネル管理外の状態になるのは，特権モードで実行している間に限られる【NGKI0547】．

1.5.10. 処理単位の開始・終了とシステム状態

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

	CPUロックフラグ	割込み優先度マスク	ディスパッチ禁止フラグ
【タスク】【NGKI0183】			
実行開始条件	解除	全解除	許可
実行開始時処理	そのまま	そのまま	そのまま
終了前	原則解除(*1)	原則全解除(*1)	原則許可(*1)
終了時処理	解除する	全解除する	許可する
【タスク例外処理ルーチン】【NGKI0184】			
実行開始条件	解除	全解除	任意
実行開始時処理	そのまま	そのまま	そのまま
リターン前	原則解除(*1)	原則全解除(*1)	元に戻す
リターン時処理	解除する	全解除する	元に戻す(*4)
【カーネル管理の割込みハンドラ】【NGKI0185】			
【割込みサービスルーチン】【NGKI0186】			
【タイムイベントハンドラ】【NGKI0187】			
実行開始条件	解除	自優先度より低い	任意
実行開始時処理	そのまま	自優先度に(*2)	そのまま
リターン前	原則解除(*1)	変更不可(*3)	変更不可(*3)
リターン時処理	解除する	元に戻す(*5)	そのまま
【CPU例外ハンドラ】【NGKI0188】			
実行開始条件	任意	任意	任意
実行開始時処理	そのまま(*6)	そのまま	そのまま
リターン前	原則元に(*1)	変更不可(*3)	変更不可(*3)
リターン時処理	元に戻す	元に戻す(*5)	そのまま
【拡張サービスコール】【NGKI0189】			
実行開始条件	任意	任意	任意

	CPUロックフラグ	割込み優先度マスク	ディスパッチ禁止フラグ
実行開始時処理	そのまま	そのまま	そのまま
リターン前	任意	任意	任意
リターン時処理	そのまま	そのまま	そのまま

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

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

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

保護機能対応カーネルでは，タスク例外処理ルーチンからのリターン時にディスパッチ禁止フラグを元に戻す処理(*4)は，タスクにディスパッチ禁止フラグの変更を許可している場合にのみ行われる【NGKI0529】．カーネルは，ディスパッチ禁止フラグの元の状態をユーザスタック上に保存する【NGKI0530】．アプリケーションがユーザスタック上に保存されたディスパッチ禁止フラグの状態を書き換えた場合，タスク例外処理ルーチンからのリターン時には，書き換えた後のディスパッチ禁止フラグの状態に変更される（すなわち，元に戻されるとは限らない）【NGKI0190】．

また，保護機能対応カーネルでは，タスクにディスパッチ禁止フラグの変更を許可していない場合で，タスク例外処理ルーチン中で拡張サービスコールを用いてディスパッチ禁止フラグを変更した場合，カーネルは元の状態に戻さない【NGKI0191】．このことから，タスク例外処理ルーチンからの終了前に，ディスパッチ禁止フラグを元の状態に戻すのは，アプリケーションの責任とする 【NGKI0192】．

【補足説明】

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

【仕様決定の理由】

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

割込みハンドラやCPU例外ハンドラで，その処理単位中で割込み優先度マスクを変更する

APIが用意されていないにもかかわらず，処理単位からのリターン時に　　元の状態に戻す(*5)のは，プロセッサによっては，割込み優先度マスクがステー　　タスレジスタ等に含まれており，APIを用いずに変更できてしまう場合があるた めである．

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

1.6. タスクの状態遷移とスケジューリング規則

1.6.1. 基本的なタスク状態

カーネルに登録したタスクは，実行できる状態，休止状態，広義の待ち状態のいずれかの状態を取る【NGKI0193】．また，実行できる状態と広義の待ち状態を総称して，起動された状態と呼ぶ．さらに，タスクをカーネルに登録していない仮想的な状態を，未登録状態と呼ぶ．

1. 実行できる状態（runnable）

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

   a. 実行状態（running）

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

   b. 実行可能状態（ready）

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

2. 休止状態（dormant）

   タスクが実行すべき処理がない状態．タスクの実行を終了した後，次に起動するまでの間は，タスクは休止状態となっている．タスクが休止状態にある時には，タスクの実行を再開するための情報（実行再開番地やレジスタの内容など）は保存されていない【NGKI0194】．

3. 広義の待ち状態（blocked）

   タスクが，処理の途中で実行を止められている状態．タスクが広義の待ち状態

にある時には，タスクの実行を再開するための情報（実行再開番地やレジスタの内容など）は保存されており，タスクが実行を再開する時には，広義の待ち状態に遷移する前の状態に戻される【NGKI0195】．広義の待ち状態は，さらに，（狭義の）待ち状態，強制待ち状態，二重待ち状態に分類される．

a. （狭義の）待ち状態（waiting）

タスクが何らかの条件が揃うのを待つために，自ら実行を止めている状態．

b. 強制待ち状態（suspended）

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

c. 二重待ち状態（waiting-suspended）+

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

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

4. 未登録状態（non-existent）

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

カーネルによっては，これらのタスク状態以外に，過渡的な状態が存在する場合がある【NGKI0196】．過渡的な状態については，「2.6.6 ディスパッチ保留状態で実行中のタスクに対する強制待ち」の節を参照すること．

【TOPPERS/ASPカーネルにおける規定】

ASPカーネルでは，タスクが未登録状態になることはない【ASPS0005】．また，上記のタスク状態以外の過渡的な状態になることもない【ASPS0006】．ただし，動的生成機能拡張パッケージでは，タスクが未登録状態になる【ASPS0007】．

【TOPPERS/FMPカーネルにおける規定】

FMPカーネルでは，タスクが未登録状態になることはない【FMPS0003】．上記のタスク状態以外の過渡的な状態として，タスクが強制待ち状態［実行継続中］になることがある【FMPS0004】．詳しくは，「2.6.6 ディスパッチ保留状態で実行中のタスクに対する強制待ち」の節を参照すること．

【TOPPERS/HRP2カーネルにおける規定】

HRP2カーネルでは，タスクが未登録状態になることはない【HRPS0002】．また，
上記のタスク状態以外の過渡的な状態になることもない【HRPS0003】．ただし，
動的生成機能拡張パッケージでは，タスクが未登録状態になる【HRPS0010】．

【TOPPERS/SSPカーネルにおける規定】

SSPカーネルでは，タスクが広義の待ち状態と未登録状態になることはない【
SSPS0003】．また，上記のタスク状態以外の過渡的な状態になることもない【SSPS0004】．

1.6.2. タスクの状態遷移

タスクの状態遷移を図2-2に示す【NGKI0197】．

未登録状態のタスクをカーネルに登録することを，タスクを生成する（create）
という．生成されたタスクは，休止状態に遷移する【NGKI0198】．また，タス
ク生成時の属性指定により，生成と同時にタスクを起動し，実行できる状態に
することもできる【
NGKI0199】．逆に，登録されたタスクを未登録状態に遷移
させることを，タスクを削除する（
delete）という．

休止状態のタスクを，実行できる状態にすることを，タスクを起動する
（
activate）という．起動されたタスクは，実行できる状態になる
【
NGKI0200】．逆に，起動された状態のタスクを，休止状態（または未登録状
態）に遷移させることを，タスクを終了する（terminate）という．

実行できる状態になったタスクは，まずは実行可能状態に遷移するが，そのタ
スクの優先順位が実行状態のタスクよりも高い場合には，ディスパッチ保留状
態でない限りはただちにディスパッチが起こり，実行状態へ遷移する
【
NGKI0201】．この時，それまで実行状態であったタスクは実行可能状態に遷
移する【
NGKI0202】．この時，実行状態に遷移したタスクは，実行可能状態に
遷移したタスクをプリエンプトしたという．逆に，実行可能状態に遷移したタ
スクは，プリエンプトされたという．

タスクを待ち解除するとは，タスクが待ち状態（二重待ち状態を除く）であれ
ば実行できる状態に，二重待ち状態であれば強制待ち状態に遷移させることを
いう．また，タスクを強制待ちから再開するとは，タスクが強制待ち状態（二
重待ち状態を除く）であれば実行できる状態に，二重待ち状態であれば待ち状 態に遷移させることをいう．

【補足説明】

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

1.6.3. タスクのスケジューリング規則

実行できるタスクは，優先順位の高いものから順に実行される【NGKI0203】．
すなわち，ディスパッチ保留状態でない限りは，実行できるタスクの中で最も
高い優先順位を持つタスクが実行状態となり，他は実行可能状態となる．

タスクの優先順位は，タスクの優先度とタスクが実行できる状態になった順序から，次のように定まる．優先度の異なるタスクの間では，優先度の高いタスクが高い優先順位を持つ【NGKI0204】．優先度が同一のタスクの間では，先に実行できる状態になったタスクが高い優先順位を持つ【NGKI0205】．すなわち，同じ優先度を持つタスクは，FCFS（First Come First Served）方式でスケジューリングされる．ただし，サービスコールの呼出しにより，同じ優先度を持つタスク間の優先順位を変更することも可能である【NGKI0206】．

最も高い優先順位を持つタスクが変化した場合には，ディスパッチ保留状態でない限りはただちにディスパッチが起こり，最も高い優先順位を持つタスクが実行状態となる【NGKI0207】．ディスパッチ保留状態においては，実行状態のタスクは切り換わらず，最も高い優先順位を持つタスクは実行可能状態にとど まる【NGKI0208】．

マルチプロセッサ対応カーネルでは，プロセッサ毎に，上記のスケジューリング規則を適用して，タスクスケジューリングを行う【NGKI0209】．すなわち，プロセッサがディスパッチ保留状態でない限りは，そのプロセッサに割り付けられた実行できるタスクの中で最も高い優先順位を持つタスクが実行状態となり，他は実行可能状態となる．そのため，実行状態のタスクは，プロセッサ毎 に存在する．

1.6.4. 待ち行列と待ち解除の順序

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

待ち行列にタスクをつなぐ順序には，FIFO順とタスクの優先度順がある．どちらの順序でつなぐかは，待ち行列毎に規定される【NGKI0210】．多くの待ち行列において，どちらの順序でつなぐかを，オブジェクト属性により指定できる 【NGKI0211】．

FIFO順の待ち行列においては，新たに待ち状態に遷移したタスクは待ち行列の最後につながれる【NGKI0212】．それに対してタスクの優先度順の待ち行列においては，新たに待ち状態に遷移したタスクは，優先度の高い順に待ち行列につながれる【NGKI0213】．同じ優先度のタスクが待ち行列につながれている場合には，新たに待ち状態に遷移したタスクが，同じ優先度のタスクの中で最後 につながれる【NGKI0214】．

待ち解除の条件がタスクによって異なる場合には，待ち行列の先頭のタスクは待ち解除の条件を満たさないが，後方のタスクが待ち解除の条件を満たす場合がある．このような場合の振舞いとして，次の2つのケースがある．どちらの振舞いをするかは，待ち行列毎に規定される【NGKI0215】．

(a) 待ち解除の条件を満たしたタスクの中で，待ち行列の前方につながれたものから順に待ち解除される【NGKI0216】．すなわち，待ち行列の前方に待ち解除の条件を満たさないタスクがあっても，後方のタスクが待ち解除の条件を満たしていれば，先に待ち解除される．

(b) タスクの待ち解除は，待ち行列につながれている順序で行われる【NGKI0217】．すなわち，待ち行列の前方に待ち解除の条件を満たさないタス

クがあると，後方のタスクが待ち解除の条件を満たしても，待ち解除されない．

ここで，(b)の振舞いをする待ち行列においては，待ち行列につながれたタスクの強制終了，タスク優先度の変更（待ち行列がタスクの優先度順の場合のみ），待ち状態の強制解除が行われた場合に，タスクの待ち解除が起こることがある．具体的には，これらの操作により新たに待ち行列の先頭になったタスクが，待ち解除の条件を満たしていれば，ただちに待ち解除される【NGKI0218】．さらに，この待ち解除により新たに待ち行列の先頭になったタスクに対しても，同じ処理が繰り返される【NGKI0219】．

1.6.5. タスク例外処理マスク状態と待ち禁止状態

保護機能対応カーネルにおいて，ユーザタスクについては特権モードで実行している間（特権モードを実行している間に，実行可能状態や広義の待ち状態になっている場合を含む．また，サービスコールを呼び出して，実行可能状態や広義の待ち状態になっている場合も含む．タスクの実行開始前は含まない），システムタスクについては拡張サービスコールを実行している間（拡張サービスコールを実行している間に，実行可能状態や広義の待ち状態になっている場合を含む）は，タスク例外処理ルーチンの実行は開始されない【NGKI0220】．これらの状態を，タスク例外処理マスク状態と呼ぶ．

タスクは，タスク例外処理マスク状態である時に，基本的なタスク状態と重複して，待ち禁止状態になることができる【NGKI0221】．待ち禁止状態とは，タスクが待ち状態に入ることが一時的に禁止された状態である．待ち禁止状態にあるタスクが，サービスコールを呼び出して待ち状態に遷移しようとした場合，サービスコールはE_RLWAIエラーとなる【NGKI0222】．

タスクを待ち禁止状態に遷移させるサービスコールは，対象タスクがタスク例外処理マスク状態である場合に，対象タスクを待ち禁止状態に遷移させる【NGKI0223】．その後，タスクがタスク例外処理マスク状態でなくなる時点（ユーザタスクについては特権モードから戻る時点，システムタスクについて拡張サービスコールからリターンする時点）で，待ち禁止状態が解除される【NGKI0224】．また，タスクの待ち禁止状態を解除するサービスコールによっても，待ち禁止状態を解除することができる【NGKI0225】．

【仕様決定の理由】

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

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

ムスタック上に不要な情報が残ってしまう．

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

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

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

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

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

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

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

【μITRON4.0仕様，μITRON4.0/PX仕様との関係】

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

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

1.6.6. ディスパッチ保留状態で実行中のタスクに対する強制待ち

ディスパッチ保留状態において，実行状態のタスクを強制待ち状態へ遷移させるサービスコールを呼び出した場合，実行状態のタスクの切換えは，ディスパッチ保留状態が解除されるまで保留される【NGKI0226】．

この間，それまで実行状態であったタスクは，実行状態と強制待ち状態の間の過渡的な状態にあると考える【NGKI0227】．この状態を，強制待ち状態［実行継続中］と呼ぶ．一方，ディスパッチ保留状態が解除された後に実行すべきタスクは，実行可能状態にとどまる【NGKI0228】．

タスクが強制待ち状態［実行継続中］にある時に，ディスパッチ保留状態が解除されると，ただちにディスパッチが起こり，タスクは強制待ち状態に遷移す る【NGKI0229】．

過渡的な状態も含めたタスクの状態遷移を図2-3に示す【NGKI0230】．

タスクが強制待ち状態［実行継続中］である時の扱いは次の通りである．

1. プロセッサを占有して実行を継続する．

   強制待ち状態［実行継続中］のタスクは，プロセッサを占有して，そのまま継続して実行される【NGKI0231】．

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

   実行状態のタスクに関する情報を参照するサービスコール（get_tid／iget_tid，get_did，sns_tex）では，強制待ち状態［実行継続中］のタスクが，
   それを実行するプロセッサにおいて実行状態のタスクであるものと扱う．具体的には，強制待ち状態［実行継続中］のタスクが実行されている時にget_tid／iget_tidを発行すると，そのタスクのID番号を参照する【NGKI0232】．また，get_didを発行するとそのタスクが属する保護ドメインのID番号を，sns_texを発行するとそのタスクのタスク例外処理禁止フラグを参照する【NGKI0233】．

3. その他のサービスコールでは，強制待ち状態であるものと扱う．

   その他のサービスコールでは，強制待ち状態［実行継続中］のタスクは，強制待ち状態であるものと扱う【NGKI0234】．

   【TOPPERS/ASPカーネルにおける規定】

   ASPカーネルでは，ディスパッチ保留状態において実行状態のタスクを強制待ち状態へ遷移させるサービスコールはサポートしていないため，タスクが強制待ち状態［実行継続中］になることはない【ASPS0008】．

【TOPPERS/FMPカーネルにおける規定】

FMPカーネルでは，ディスパッチ保留状態において実行状態のタスクを強制待ち状態へ遷移させるサービスコールを，他のプロセッサから呼び出すことができるため，タスクが強制待ち状態［実行継続中］になる場合がある【FMPS0005】.

【TOPPERS/HRP2カーネルにおける規定】

HRP2カーネルでは，ディスパッチ保留状態において実行状態のタスクを強制待ち状態へ遷移させるサービスコールはサポートしていないため，タスクが強制待ち状態［実行継続中］になることはない【HRPS0004】.

【TOPPERS/SSPカーネルにおける規定】

SSPカーネルでは，タスクが広義の待ち状態になることはないため，タスクが強制待ち状態［実行継続中］になることもない【SSPS0005】.

【補足説明】

この仕様では，ディスパッチ保留状態において，実行状態のタスクを強制終了させるサービスコールはサポートしていない．そのため，実行状態と休止状態の間の過渡的な状態は存在しない.

1.6.7. 制約タスク

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

a. 広義の待ち状態に入ることができない【NGKI0235】.
b. サービスコールによりベース優先度を変更することができない【NGKI0236】.
c. 対象優先度の中の先頭のタスクが制約タスクである場合には，タスクの優先順位の回転（rot_rdq／irot_rdq）を行うことができない【NGKI0237】.
d. マルチプロセッサ対応カーネルでは，割付けプロセッサを変更することが できない【NGKI0238】.

制約タスクに対して，機能制限により使用できなくなったサービスコールを呼び出した場合には，E_NOSPTエラーとなる【NGKI0239】．E_NOSPTエラーが返ることに依存している場合を除いては，制約タスクを通常のタスクに置き換える ことができる【NGKI0240】.

【未決定事項】

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

【TOPPERS/ASPカーネルにおける規定】

ASPカーネルでは，制約タスクをサポートしていない【ASPS0009】．ただし，制約タスク拡張パッケージを用いると，制約タスクの機能を追加することができ る【ASPS0010】．

【TOPPERS/FMPカーネルにおける規定】

FMPカーネルでは，制約タスクをサポートしていない【FMPS0006】．

【TOPPERS/HRP2カーネルにおける規定】

HRP2カーネルでは，制約タスクをサポートしていない【HRPS0005】．

【TOPPERS/SSPカーネルにおける規定】

SSPカーネルでは，制約タスクのみをサポートする【SSPS0006】．そのため，すべてのタスクと非タスクコンテキストがスタック領域を共有することができ，すべての処理単位で同一のスタック領域を使用している【SSPS0007】．このスタック領域を，共有スタック領域と呼ぶ．

【μITRON4.0仕様との関係】

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

1.7. 割込み処理モデル

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

TOPPERS標準割込み処理モデルの概念図を図2-4に示す【NGKI0241】．この図は，割込み処理モデルの持つすべての機能が，ハードウェア（プロセッサおよび割込みコントローラ）で実現されているとして描いた概念図である．実際のハードウェアで不足している機能については，カーネル内の割込み処理のソフトウェ アで実現される．

【μITRON4.0仕様との関係】

割込み処理モデルは，μITRON4.0仕様から大幅に拡張している．

1.7.1. 割込み処理の流れ

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

プロセッサは，デバイスからの割込み要求を受け付ける条件が満たされた場合，割込み要求を受け付ける【NGKI0242】．受け付けた割込み要求が，カーネル管理の割込みである場合には，カーネル内の割込みハンドラの入口処理（割込み

入口処理）を経由して，カーネル内の割込みハンドラを実行する【NGKI0243】．

カーネル内の割込みハンドラは，アプリケーションが割込み要求ラインに対して登録した割込みサービスルーチン（ISR）を呼び出す【NGKI0244】．割込みサービスルーチンは，プロセッサの割込みアーキテクチャや割込みコントローラに依存せず，割込みを要求したデバイスのみに依存して記述するのが原則である【NGKI0245】．1つの割込み要求ラインに対して複数のデバイスが接続されるこ とから，1つの割込み要求ラインに対して複数の割込みサービスルーチンを登録 することができる【NGKI0246】．

ただし，カーネルが標準的に用意している割込みハンドラで対応できない特殊なケースも考えられる．このような場合に対応するために，アプリケーションが用意した割込みハンドラをカーネルに登録することもできる【NGKI0247】．

カーネルが用いるタイマデバイスからの割込み要求の場合，カーネル内の割込みハンドラにより，タイムイベントの処理が行われる．具体的には，タイムアウト処理等が行われることに加えて，アプリケーションが登録したタイムイベントハンドラが呼び出される【NGKI0248】．

なお，受け付けた割込み要求に対して，割込みサービスルーチンも割込みハンドラも登録していない場合の振舞いは，ターゲット定義である【NGKI0249】．

1.7.2. 割込み優先度

割込み要求は，割込み処理の優先順位を指定するための割込み優先度を持つ【NGKI0250】．プロセッサは，割込み優先度マスクの現在値よりも高い割込み優先度を持つ割込み要求のみを受け付ける【NGKI0251】．逆に言うと，割込み優先度マスクの現在値と同じか，それより低い割込み優先度を持つ割込みは， マスクされる．

プロセッサは，割込み要求を受け付けると，割込み優先度マスクを，受け付けた割込み要求の割込み優先度に設定する（ただし，受け付けた割込みがNMIである場合には例外とする）【NGKI0252】．また，割込み処理からのリターンにより，割込み優先度マスクを，割込み要求を受け付ける前の値に戻す 【NGKI0253】．

これらのことから，他の方法で割込みをマスクしていない限り，ある割込み要求の処理中は，それと同じかそれより低い割込み優先度を持つ割込み要求は受け付けられず，それより高い割込み優先度を持つ割込み要求は受け付けられることになる．つまり，割込み優先度は，多重割込みを制御するためのものと位置付けることができる．それに対して，同時に発生している割込み要求の中で，割込み優先度の高い割込み要求が先に受け付けられるとは限らない 【NGKI0254】．

割込み優先度は，PRI型で表現し，値が小さいほど優先度が高いものとするが，［NGKI0037］の原則には従わず，-1から連続した負の値を用いる【NGKI0255】．

割込み優先度の段階数は，ターゲット定義である【NGKI0256】．プロセッサが割込み優先度マスクを実現するための機能を持たないか，実現するために大きいオーバヘッドを生じる場合には，ターゲット定義で，割込み優先度の段階数を1にする（すなわち，多重割込みを許さない）場合がある．

【仕様決定の理由】

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

1.7.3. 割込み要求ラインの属性

各割込み要求ラインは，以下の属性を持つ．なお，1つの割込み要求ラインに複数のデバイスからの割込み要求が接続されている場合，それらの割込み要求は同一の属性を持つ【NGKI0257】．それらの割込み要求に別々の属性を設定する ことはできない．

1. 割込み要求禁止フラグ

   割込み要求ライン毎に，割込みをマスクするための割込み要求禁止フラグを持つ【NGKI0258】．割込み要求禁止フラグをセットすると，その割込み要求ラインによって伝えられる割込み要求はマスクされる【NGKI0259】．

   プロセッサが割込み要求禁止フラグを実現するための機能を持たないか，実現するために大きいオーバヘッドを生じる場合には，ターゲット定義で，割込み要求禁止フラグをサポートしない場合がある【NGKI0260】．また，プロセッサの持つ割込み要求禁止フラグの機能がこの仕様に合致しない場合には，ターゲット定義で，割込み要求禁止フラグをサポートしないか，振舞いが異なるものとする場合がある【NGKI0261】．

2. 割込み優先度

   割込み要求ライン毎に，割込み優先度を設定することができる【NGKI0262】．割込み要求の割込み優先度とは，その割込み要求を伝える割込み要求ラインに対して設定された割込み優先度のことである【NGKI0263】．

3. トリガモード

   割込み要求ラインに対する割込み要求が，レベルトリガであるかエッジトリガであるかを設定することができる【NGKI0264】．エッジトリガの場合には，さらに，ターゲット定義で，ポジティブエッジトリガかネガティブエッジトリガか両エッジトリガかを設定できる場合もある【NGKI0265】．また，レベルトリガの場合には，ターゲット定義で，ローレベルトリガかハイレベルトリガかを設定できる場合もある【NGKI0266】．

   プロセッサがトリガモードを設定するための機能を持たないか，設定するために大きいオーバヘッドを生じる場合には，ターゲット定義で，トリガモードの設定をサポートしない場合がある【NGKI0267】．

   属性が設定されていない割込み要求ラインに対しては，割込み要求禁止フラグがセットされ，割込み要求はマスクされる【NGKI0268】．また，割込み要求禁止フラグをクリアすることもできない【NGKI0269】．

【使用上の注意】

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

複数のデバイスからの割込み要求が接続されている割込み要求ラインを，エッジトリガに設定することは推奨されない．これは，次のような状況において，割込み要求を取りこぼす可能性があるためである．ある割込み要求ラインに，デバイスAとデバイスBからの割込み要求が接続されており，デバイスAの割込み処理を先に行う場合を考える．この時，デバイスBからの割込み要求によって割込みハンドラが実行され，デバイスAの割込み処理を行った後，デバイスBの割込み処理を行う前に，デバイスAからの割込み要求が発生した場合に，デバイスAからの割込み要求を取りこぼしてしまう．

1.7.4. 割込みを受け付ける条件

NMI以外の割込み要求は，次の4つの条件が揃った場合に受け付けられる 【NGKI0270】．

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

これらの条件が揃った割込み要求が複数ある場合に，どの割込み要求が最初に受け付けられるかは，この仕様では規定しない【NGKI0271】．すなわち，割込み優先度の高い割込み要求が先に受け付けられるとは限らない．

1.7.5. 割込み番号と割込みハンドラ番号

割込み要求ラインを識別するための番号を，割込み番号と呼ぶ．割込み番号は，符号無しの整数型であるINTNO型で表し，ターゲットハードウェアの仕様から決まる自然な番号付けを基本として，ターゲット定義で付与される【NGKI0272】．そのため，1から連続した正の値であるとは限らない．

それに対して，アプリケーションが用意した割込みハンドラをカーネルに登録する場合に，割込みハンドラの登録対象となる割込みを識別するための番号を，割込みハンドラ番号と呼ぶ．割込みハンドラ番号は，符号無しの整数型であるINHNO型で表し，ターゲットハードウェアの仕様から決まる自然な番号付けを基本として，ターゲット定義で付与される【NGKI0273】．そのため，1から連続した正の値であるとは限らない．

割込みハンドラ番号は，割込み番号と1対1に対応するのが基本である（両者が一致する場合が多い）【NGKI0274】．

ただし，割込みを要求したデバイスが割込みベクタを生成してプロセッサに渡すアーキテクチャなどでは，割込み番号と割込みハンドラ番号の対応を，カーネルが管理していない場合がある【NGKI0275】．そこで，ターゲット定義で，割込み番号に対応しない割込みハンドラ番号や，割込みハンドラ番号に対応しない割込み番号を設ける場合もある【NGKI0276】．ただし，割込みサービスルーチンの登録対象にできる割込み番号は，割込みハンドラ番号との1対1の対応関係をカーネルが管理しているもののみである【NGKI0277】．

1.7.6. マルチプロセッサにおける割込み処理

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

マルチプロセッサ対応カーネルでは，TOPPERS標準割込み処理モデルの構成要素の中で，図2-4の破線に囲まれた部分はプロセッサ毎に持ち，それ以外の部分はシステム全体で1つのみ持つ【NGKI0278】．すなわち，全割込みロックフラグ，CPUロックフラグ，割込み優先度マスクはプロセッサ毎に持つのに対して，割込み要求ラインおよびその属性（割込み要求禁止フラグ，割込み優先度，トリガモード）はシステム全体で共通に持つ．

割込み番号は，割込み要求ラインを識別するための番号であることから，割込み要求ラインが複数のプロセッサに接続されている場合でも，1つの割込み要求ラインには1つの割込み番号を付与する【NGKI0279】．逆に，複数のプロセッサが同じ種類のデバイスを持っている場合でも，別のデバイスからの割込み要求ラインには異なる割込み番号を付与する（図2-5）【NGKI0280】．図2-5において，ローカルIRCは個々のプロセッサに対する割込みを制御するための回路であり，グローバルIRCはデバイスからの割込みをプロセッサに分配するための回路である．グローバルIRCは，必ず備わっているとは限らない．

割込み要求禁止フラグは，この仕様上はシステム全体で共通に持つこととしているが，実際のターゲットハードウェア（特に，グローバルIRCを備えていないもの）では，プロセッサ毎に持っている場合がある．そのため，ターゲット定義で，あるプロセッサで割込み要求禁止フラグを動的にセット／クリアしても，他のプロセッサに対しては割込みがマスク／マスク解除されない場合があるも のとする【NGKI0281】．

複数のプロセッサに接続された割込み要求ラインに対して登録された割込みサービスルーチンは，それらのプロセッサのいずれによっても実行することができる【NGKI0282】．ただし，その内のどのプロセッサで割込みサービスルーチンを実行するかは，割込みサービスルーチンが属するクラスの割付け可能プロセッサにより決定される（「2.4.4 処理単位を実行するプロセッサ」の節を参照）．

割込みサービスルーチンが属するクラスの割付け可能プロセッサは，登録対象の割込み要求ラインが接続されたプロセッサの集合に含まれていなければならない【NGKI0283】．また，同一の割込み要求ラインに対して登録する割込みサー

ビスルーチンは，同一のクラスに属していなければならない【NGKI0284】．

それに対して，割込みハンドラはプロセッサ毎に登録する．そのため，同じ割込み要求に対応する割込みハンドラであっても，プロセッサ毎に異なる割込みハンドラ番号を付与する（図2-5）【NGKI0285】．割込みハンドラが属するクラスの初期割付けプロセッサは，割込みが要求されるプロセッサと一致していなければならない【NGKI0286】．

【補足説明】

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

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

【使用上の注意】

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

1.7.7. カーネル管理外の割込み

高い割込み応答性を求められるアプリケーションでは，カーネル内で割込みをマスクすることにより，割込み応答性の要求を満たせなくなる場合がある．このような要求に対応するために，カーネル内では，ある割込み優先度（これを，TMIN_INTPRIと書く）よりも高い割込み優先度を持つ割込みをマスクしないこととしている【NGKI0287】．TMIN_INTPRIを固定するか設定できるようにするか，設定できるようにする場合の設定方法は，ターゲット定義である【NGKI0288】．

TMIN_INTPRIよりも高い割込み優先度を持ち，カーネル内でマスクしない割込みを，カーネル管理外の割込みと呼ぶ．また，カーネル管理外の割込みによって起動される割込みハンドラを，カーネル管理外の割込みハンドラと呼ぶ．NMIは，カーネル管理外の割込みとして扱う．NMI以外にカーネル管理外の割込みを設けるか（設けられるようにするか）どうかは，ターゲット定義である【NGKI0289】．

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

カーネル管理外の割込みハンドラは，カーネル内の割込み入口処理を経由せずに実行するのが基本である【NGKI0290】．ただし，すべての割込みで同じ番地

に分岐するプロセッサでは，カーネル内の割込み入口処理を全く経由せずにカーネル管理外の割込みハンドラを実行することができず，入口処理の一部分を経由してカーネル管理外の割込みハンドラが実行されることになる【NGKI0291】.

カーネル管理外の割込みハンドラが実行開始される時のシステム状態とコンテキスト，割込みハンドラの終了時に行われる処理，割込みハンドラの記述方法は，ターゲット定義である【NGKI0292】．カーネル管理外の割込みハンドラからは，システムインタフェースレイヤのAPIとsns_ker，ext_kerのみを呼び出すことができ，その他のサービスコールを呼び出すことはできない【NGKI0293】．カーネル管理外の割込みハンドラから，その他のサービスコールを呼び出した場合の動作は，保証されない【NGKI0294】.

1.7.8. カーネル管理外の割込みの設定方法

カーネル管理外の割込みの設定方法は，ターゲット定義で，次の3つの方法のいずれかが採用される【NGKI0295】.

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

   これら場合には，カーネル管理外とする割込みはカーネル構築時（ターゲット依存部の実装時やカーネルのコンパイル時）に決まるため，カーネル管理外とする割込みをアプリケーション側で設定する必要はない【NGKI0296】．ここで，カーネル管理外とされた割込みに対して，カーネルのAPIにより割込みハンドラを登録できるかと，割込み要求ラインの属性を設定できるかは，ターゲット定義である【NGKI0297】．割込みハンドラを登録できる場合には，それを定義するAPIにおいて，カーネル管理外であることを示す割込みハンドラ属性（TA_NONKERNEL）を指定する【NGKI0298】．また，割込み要求ラインの属性を設定できる場合には，設定する割込み優先度をTMIN_INTPRIよりも高い値とする【NGKI0299】.

b. カーネル管理外とする割込みをアプリケーションで設定できるようにする

   この場合には，カーネル管理外とする割込みの設定は，次の方法で行う．まず，カーネル管理外とする割込みハンドラを定義するAPIにおいて，カーネル管理外であることを示す割込みハンドラ属性（TA_NONKERNEL）を指定する【NGKI0300】．また，カーネル管理外とする割込みの割込み要求ラインに対して設定する割込み優先度を，TMIN_INTPRIよりも高い値とする【NGKI0301】.

いずれの場合にも，カーネル管理の割込みの割込み要求ラインに対して設定する割込み優先度は，TMIN_INTPRIより高い値であってはならない【NGKI0302】．また，カーネル管理外の割込みに対して，割込みサービスルーチンを登録することはできない【NGKI0303】.

1.8. CPU例外処理モデル

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

1.8.1. CPU例外処理の流れ

アプリケーションは，プロセッサが検出するCPU例外の種類毎に，CPU例外ハンドラを登録することができる【NGKI0304】．プロセッサがCPU例外の発生を検出すると，カーネル内のCPU例外ハンドラの入口処理（CPU例外入口処理）を経由して，発生したCPU例外に対して登録したCPU例外ハンドラが呼び出される 【NGKI0305】．

CPU例外ハンドラの登録対象となるCPU例外を識別するための番号を，CPU例外ハンドラ番号と呼ぶ．CPU例外ハンドラ番号は，符号無しの整数型であるEXCNO型で表し，ターゲットハードウェアの仕様から決まる自然な番号付けを基本として，ターゲット定義で付与される【NGKI0306】．そのため，1から連続した正の 値であるとは限らない．

マルチプロセッサ対応カーネルでは，異なるプロセッサで発生するCPU例外は，異なるCPU例外であると扱う【NGKI0307】．すなわち，同じ種類のCPU例外であっても，異なるプロセッサのCPU例外には異なるCPU例外ハンドラ番号を付与し，プロセッサ毎にCPU例外ハンドラを登録する．CPU例外ハンドラが属するクラスの初期割付けプロセッサは，CPU例外が発生するプロセッサと一致していなけれ ばならない【NGKI0308】．

CPU例外ハンドラにおいては，CPU例外が発生した状態からのリカバリ処理を行う【NGKI0309】．どのようなリカバリ処理を行うかは，一般にはCPU例外の種類やそれが発生したコンテキストおよび状態に依存するが，大きく次の4つの方法が考えられる【NGKI0310】．

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

(b) CPU例外を起こしたタスクよりも優先度の高いタスクを起動または待ち解除し，そのタスクでリカバリ処理を行う（例えば，CPU例外を起こしたタスクを強制終了し，再度起動する）．ただし，CPU例外を起こしたタスクが最高優先度の場合には，この方法でリカバリ処理を行うことはできない（リカバリ処理を行うタスクを最高優先度とし，タスクの起動または待ち解除後に優先順位を回転させることで，リカバリ処理を行える可能性があるが，CPU例外を起こしたタスクが制約タスクの場合には適用できないなど，推奨できる方法ではない）【NGKI0311】．

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

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

この中で(a)と(d)の方法は，カーネルの機能を必要としないため，CPU例外が発生したコンテキストおよび状態に依存せずに常に行える【NGKI0312】．それに対して(b)と(c)の方法は，CPU例外ハンドラからそのためのサービスコールを呼び出せることが必要であり，それが行えるかどうかは，CPU例外が発生したコンテキストおよび状態に依存する【NGKI0313】．

なお，発生したCPU例外に対して，CPU例外ハンドラを登録していない場合の振舞いは，ターゲット定義である【NGKI0314】．

【使用上の注意】

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

【補足説明】

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

【μITRON4.0仕様との関係】

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

1.8.2. CPU例外ハンドラから呼び出せるサービスコール

CPU例外ハンドラからは，CPU例外発生時のディスパッチ保留状態を参照するサービスコール（xsns_dpn）と，CPU例外発生時にタスク例外処理ルーチンを実行開始できない状態であったかを参照するサービスコール（xsns_xpn）を呼び出すことができる【NGKI0315】．

xsns_dpnは，CPU例外がタスクコンテキストで発生し，そのタスクがディスパッチできる状態であった場合にfalseを返す【NGKI0316】．xsns_dpnがfalseを返した場合，そのCPU例外ハンドラから，非タスクコンテキストから呼び出せるすべてのサービスコールを呼び出すことができ，(b)の方法によるリカバリ処理が可能である【NGKI0317】．ただし，CPU例外を起こしたタスクが最高優先度の場合には，この方法でリカバリ処理を行うことはできない【NGKI0318】．

xsns_xpnは，CPU例外がタスクコンテキストで発生し，そのタスクがタスク例外処理ルーチンを実行できる状態であった場合にfalseを返す【NGKI0319】．xsns_xpnがfalseを返した場合，そのCPU例外ハンドラから，非タスクコンテキストから呼び出せるすべてのサービスコールを呼び出すことができ，(c)の方法によるリカバリ処理が可能である【NGKI0320】．

xsns_dpnとxsns_xpnのいずれのサービスコールもtrueを返した場合，そのCPU例外ハンドラからは，xsns_dpnとxsns_xpnに加えて，システムインタフェースレイヤのAPIとsns_ker，ext_kerのみを呼び出すことができ，その他のサービスコールを呼び出すことはできない【NGKI0321】．いずれのサービスコールもtrueを返したにもかかわらず，その他のサービスコールを呼び出した場合の動作は，保証されない【NGKI0322】．この場合には，(b)と(c)の方法によるリカバリ処理は行うことはできず，(a)または(d)の方法によるリカバリ処理を行うしかな いことになる．

【μITRON4.0仕様との関係】

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

1.8.3. エミュレートされたCPU例外ハンドラ

エラーコードによってアプリケーションに通知できないエラーをカーネルが検出した場合に，アプリケーションが登録したエラー処理を，カーネルが呼び出す場合がある【NGKI0323】．この場合に，カーネルが検出するエラーをCPU例外と同等に扱うものとし，エミュレートされたCPU例外と呼ぶ【NGKI0324】．また，エラー処理のためのプログラムをCPU例外ハンドラと同等に扱うものとし，エミュレートされたCPU例外ハンドラと呼ぶ【NGKI0325】．

具体的には，エミュレートされたCPU例外ハンドラに対してもCPU例外ハンドラ番号が付与され，CPU例外ハンドラと同じ方法で登録できる【NGKI0326】．また，エミュレートされたCPU例外ハンドラからも，CPU例外ハンドラから呼び出せるサービスコールを呼び出すことができ，CPU例外ハンドラと同様のリカバリ処理 を行うことができる【NGKI0327】．

【μITRON4.0仕様との関係】

エミュレートされたCPU例外およびCPU例外ハンドラは，μITRON4.0仕様に定義 されていない概念である．

1.8.4. カーネル管理外のCPU例外

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

それに対して，カーネル管理外のCPU例外以外のCPU例外をカーネル管理のCPU例外，カーネル管理のCPU例外によって起動されるCPU例外ハンドラをカーネル管 理のCPU例外ハンドラと呼ぶ．

カーネル管理外のCPU例外ハンドラにおいては，xsns_dpnとxsns_xpnのいずれのサービスコールもtrueを返す【NGKI0330】．そのため，「2.8.2 CPU例外ハンドラから呼び出せるサービスコール」の節で述べた制限［NGKI0321］［NGKI0322］ が課される．

【補足説明】

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

1.9. システムの初期化と終了

1.9.1. システム初期化手順

システムのリセット後，最初に実行するプログラムを，スタートアップモジュールと呼ぶ．スタートアップモジュールはカーネルの管理外であり，アプリケーションで用意するのが基本であるが，スタートアップモジュールで行うべき処理を明確にするために，カーネルの配布パッケージの中に，標準のスタートアップモジュールが用意されている【NGKI0331】．

標準のスタートアップモジュールは，プロセッサのモードとスタックポインタ　　等の初期化，NMIを除くすべての割込みのマスク（全割込みロック状態と同等の状態にする），ターゲットシステム依存の初期化フックの呼出し，非初期化データセクション（bssセクション）のクリア，初期化データセクション（dataセクション）の初期化，ソフトウェア環境（ライブラリなど）依存の初期化フックの呼出しを行った後，カーネルの初期化処理へ分岐する【NGKI0332】．ここで呼び出すターゲットシステム依存の初期化フックでは，リセット後に速やかに行うべき初期化処理を行うことが想定されている．

マルチプロセッサ対応カーネルでは，すべてのプロセッサがスタートアップモジュールを実行し，カーネルの初期化処理へ分岐する【NGKI0333】．ただし，共有リソースの初期化処理（非初期化データセクションのクリア，初期化データセクションの初期化，ソフトウェア環境依存の初期化フックの呼出しなど）は，マスタプロセッサのみで実行する【NGKI0334】．各プロセッサがカーネルの初期化処理へ分岐するのは，共有リソースの初期化処理が完了した後でなければならないため，スレーブプロセッサは，カーネルの初期化処理へ分岐する前に，マスタプロセッサによる共有リソースの初期化処理の完了を待ち合わせ る必要がある【NGKI0335】．

カーネルの初期化処理においては，まず，カーネル自身の初期化処理（カーネル内のデータ構造の初期化，カーネルが用いるデバイスの初期化など）と静的APIの処理（オブジェクトの登録など）が行われる【NGKI0336】．静的APIのパラメータに関するエラーは，コンフィギュレータによって検出されるのが原則であるが，コンフィギュレータで検出できないエラーが，この処理中に検出さ　　れる場合もある【NGKI0337】．

静的APIの処理順序によりシステムの規定された振舞いが変化する場合には，システムコンフィギュレーションファイルにおける静的APIの記述順と同じ順序で　　静的APIが処理された場合と，同じ振舞いとなる【NGKI0338】．例えば，静的APIによって同じ優先度のタスクを複数生成・起動した場合，静的APIの記述順が先のタスクが高い優先順位を持つ．それに対して，周期ハンドラの動作開始順序は，同じタイムティックで行うべき処理が複数ある場合の処理順序が規定　　されないことから（「4.6.1 システム時刻管理」の節を参照），静的APIの記述 順となるとは限らない．

次に，静的API（ATT_INI）により登録した初期化ルーチンが，システムコンフィギュレーションファイルにおける静的APIの記述順と同じ順序で実行される【NGKI0339】．

マルチプロセッサ対応カーネルでは，すべてのプロセッサがカーネル自身の初期化処理と静的APIの処理を完了した後に，マスタプロセッサがグローバル初期化ルーチンを実行する【NGKI0340】．グローバル初期化ルーチンの実行が完了した後に，各プロセッサは，自プロセッサに割り付けられたローカル初期化ルーチンを実行する【NGKI0341】．すなわち，ローカル初期化ルーチンは，初期割 付けプロセッサにより実行される．

以上が終了すると，カーネル非動作状態から動作状態に遷移し（「2.5.1 カーネル動作状態と非動作状態」の節を参照），カーネルの動作が開始される【NGKI0342】．具体的には，システム状態が，全割込みロック解除状態・CPUロック解除状態・割込み優先度マスク全解除状態・ディスパッチ許可状態に設定され（すなわち，割込みがマスク解除され），タスクの実行が開始される．

マルチプロセッサ対応カーネルでは，すべてのプロセッサがローカル初期化ルーチンの実行を完了した後に，カーネル非動作状態から動作状態に遷移し，カーネルの動作が開始される【NGKI0343】．マルチプロセッサ対応カーネルにおけるシステム初期化の流れと，各プロセッサが同期を取るタイミングを，図2-6に 示す【NGKI0344】．

【μITRON4.0仕様との関係】

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

1.9.2. システム終了手順

カーネルを終了させるサービスコール（ext_ker）を呼び出すと，カーネル動作状態から非動作状態に遷移する（「2.5.1 カーネル動作状態と非動作状態」の 節を参照）【NGKI0345】．具体的には，NMIを除くすべての割込みがマスクされ， タスクの実行が停止される．

マルチプロセッサ対応カーネルでは，カーネルを終了させるサービスコール（ext_ker）は，どのプロセッサからでも呼び出すことができる【NGKI0346】．1つのプロセッサでカーネルを終了させるサービスコールを呼び出すと，そのプロセッサがカーネル動作状態から非動作状態に遷移した後，他のプロセッサに対してカーネル終了処理の開始を要求する【NGKI0347】．複数のプロセッサから，カーネルを終了させるサービスコール（ext_ker）を呼び出してもよい【NGKI0348】．

次に，静的API（ATT_TER）により登録した終了処理ルーチンが，システムコンフィギュレーションファイルにおける静的APIの記述順と逆の順序で実行される【NGKI0349】．

マルチプロセッサ対応カーネルでは，すべてのプロセッサがカーネル非動作状態に遷移した後に，各プロセッサが，自プロセッサに割り付けられたローカル終了処理ルーチンを実行する【NGKI0350】．すなわち，ローカル終了処理ルーチンは，初期割付けプロセッサにより実行される．すべてのプロセッサでローカル終了処理ルーチンの実行が完了した後に，マスタプロセッサがグローバル

終了処理ルーチンを実行する【NGKI0351】.

以上が終了すると，ターゲットシステム依存の終了処理が呼び出される【NGKI0352】．ターゲットシステム依存の終了処理は，カーネルの管理外であり，アプリケーションで用意するのが基本であるが，カーネルの配布パッケージの中に，ターゲットシステム毎に標準的なルーチンが用意されている【NGKI0353】．標準のターゲットシステム依存の終了処理では，ソフトウェア環境（ライブラリなど）依存の終了処理フックを呼び出す【NGKI0354】.

マルチプロセッサ対応カーネルでは，すべてのプロセッサで，ターゲットシステム依存の終了処理が呼び出される【NGKI0355】．マルチプロセッサ対応カーネルにおけるシステム終了処理の流れと，各プロセッサが同期を取るタイミングを，図2-7に示す【NGKI0356】.

【使用上の注意】

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

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

【μITRON4.0仕様との関係】

μITRON4.0仕様には，システム終了に関する規定はない.

1.10. オブジェクトの登録とその解除

1.10.1. ID番号で識別するオブジェクト

ID番号で識別するオブジェクトは，オブジェクトを生成する静的API（CRE_YYY），サービスコール（acre_yyy），またはオブジェクトを追加する静的API（ATT_YYY，ATA_YYY）によってカーネルに登録する【NGKI0357】．オブジェクトを追加する静的APIによって登録されたオブジェクトはID番号を持たないため，ID番号を指定して操作することができない【NGKI0358】.

オブジェクトを生成する静的API（CRE_YYY）は，生成するオブジェクトにID番号を割り付け，ID番号を指定するパラメータとして記述した識別名を，割り付けたID番号にマクロ定義する【NGKI0359】．同じ識別名のオブジェクトが生成 済みの場合には，E_OBJエラーとなる【NGKI0360】.

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

1つ選び，生成するオブジェクトに割り付ける【NGKI0361】．割り付けたID番号は，サービスコールの返値としてアプリケーションに通知する【NGKI0362】．使用されていないID番号が残っていない場合 には，E_NOIDエラーとなる【NGKI0363】．

割付け可能なID番号の数を指定する静的API（AID_YYY）は，システムコンフィギュレーションファイル中に複数記述することができる【NGKI0364】．その場合，各静的APIで指定した数の合計の数のID番号が確保される【NGKI0365】．

オブジェクトを生成するサービスコール（acre_yyy）によって登録したオブジェクトは，オブジェクトを削除するサービスコール（del_yyy）によって登録を解除することができる【NGKI0366】．登録解除したオブジェクトのID番号は，未使用の状態に戻され，そのID番号を用いて新しいオブジェクトを登録することができる【NGKI0367】．この場合に，登録解除前のオブジェクトに対して行うつもりの操作が，新たに登録したオブジェクトに対して行われないように，注 意が必要である．

オブジェクトを生成または追加する静的APIによって登録したオブジェクトは，登録を解除することができない．登録を解除しようとした場合には，E_OBJエラー となる【NGKI0369】．

タスク以外の処理単位は，その処理単位が実行されている間でも，登録解除することができる【NGKI0370】．この場合，登録解除された処理単位に実行が強制的に終了させられることはなく，処理単位が自ら実行を終了するまで，処理単位の実行は継続される【NGKI0371】．

同期・通信オブジェクトを削除した時に，そのオブジェクトを待っているタスクがあった場合，それらのタスクは待ち解除され，待ち状態に遷移させたサービスコールはE_DLTエラーとなる【NGKI0372】．複数のタスクが待ち解除される場合には，待ち行列につながれていた順序で待ち解除される【NGKI0373】．削除した同期・通信オブジェクトが複数の待ち行列を持つ場合には，別の待ち行列で待っていたタスクの間の待ち解除の順序は，該当するサービスコール毎に 規定する【NGKI0374】．

オブジェクトを再初期化するサービスコール（ini_yyy）は，指定したオブジェクトを削除した後に，同じパラメータで再度生成したのと等価の振舞いをする【NGKI0375】．ただし，オブジェクトを生成または追加する静的APIによって登録したオブジェクトも，再初期化することができる【NGKI0376】．

なお，動的生成対応カーネル以外では，オブジェクトを生成するサービスコール（acre_yyy），割付け可能なID番号の数を指定する静的API（AID_YYY），オブジェクトのアクセス許可ベクタを設定するサービスコール（sac_yyy），オブジェクトを削除するサービスコール（del_yyy）は，サポートされない 【NGKI0377】．

【μITRON4.0仕様との関係】

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

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

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

【μITRON4.0/PX仕様との関係】

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

【仕様決定の理由】

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

1.10.2. オブジェクト番号で識別するオブジェクト

オブジェクト番号で識別するオブジェクトは，オブジェクトを定義する静的 API（
DEF_YYY）またはサービスコール（def_yyy）によってカーネルに登録する 【NGKI0378】．

オブジェクトを定義するサービスコール（def_yyy）によって登録したオブジェ
クトは，同じサービスコールを，オブジェクトの定義情報を入れたパケットへ のポインタを
NULLとして呼び出すことによって，登録を解除することができる 【
NGKI0379】．登録解除したオブジェクト番号は，オブジェクト登録前の状態
に戻され，同じオブジェクト番号に対して新たにオブジェクトを定義すること ができる【
NGKI0380】．登録解除されていないオブジェクト番号に対して再度
オブジェクトを登録しようとした場合には，E_OBJエラーとなる【NGKI0381】．

オブジェクトを定義する静的APIによって登録したオブジェクトは，登録を解除
することができない【NGKI0382】．登録を解除しようとした場合には，E_OBJエ ラーとなる【
NGKI0383】．

なお，動的生成対応カーネル以外では，オブジェクトを定義するサービスコー ル（
def_yyy）はサポートされない【NGKI0384】．

【μITRON4.0仕様との関係】

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

1.10.3. 識別番号を持たないオブジェクト

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

1.10.4. オブジェクト生成に必要なメモリ領域

カーネルオブジェクトを生成する際に，サイズが一定でないメモリ領域を必要とする場合には，カーネルオブジェクトを生成する静的APIおよびサービスコールに，使用するメモリ領域の先頭番地を渡すパラメータを設けている【NGKI0385】．このパラメータをNULLとした場合，必要なメモリ領域は，コンフィギュレータまたはカーネルにより確保される【NGKI0386】．

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

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

【補足説明】

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

1.10.5. オブジェクトが属する保護ドメインの設定

保護機能対応カーネルにおいて，カーネルオブジェクトが属する保護ドメインは，オブジェクトの登録時に決定し，登録後に変更することはできない【NGKI0387】．

カーネルオブジェクトを静的APIによって登録する場合には，オブジェクトを登録する静的APIを，そのオブジェクトを属させる保護ドメインの囲みの中に記述する【NGKI0388】．無所属のオブジェクトを登録する静的APIは，保護ドメインの囲みの外に記述する（「2.12.3 保護ドメインの指定」の節を参照）【NGKI0389】．

カーネルオブジェクトをサービスコールによって登録する場合には，オブジェクト属性にTA_DOM(domid)を指定することにより，オブジェクトを属させる保護ドメインを設定する【NGKI0390】．ここでdomidは，そのオブジェクトを属させる保護ドメインのID番号であり，TDOM_KERNEL（=-1）を指定することでカーネルドメインに属させることができる．また，domidにTDOM_SELF（=0）を指定するか，オブジェクト属性にTA_DOM(domid)を指定しないことで，自タスクが属する保護ドメインに属させることができる．さらに，無所属のオブジェクトを登録する場合には，domidにTDOM_NONE（=-2）を指定する．

ただし，特定の保護ドメインのみに属することができるカーネルオブジェクトを登録するサービスコールの中には，オブジェクトを属させる保護ドメインをオブジェクト属性で設定する必要がないものもある【NGKI0391】．

割付け可能なID番号の数を指定する静的API（AID_YYY）で確保したID番号は，どの保護ドメインに属するオブジェクトにも（また，無所属のオブジェクトにも）割り付けられる【NGKI0392】．これらの静的APIは，保護ドメインの囲みの外に記述しなければならない．保護ドメインの囲みの中に記述した場合には，E_RSATRエラーとなる【NGKI0394】．

【補足説明】

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

【仕様決定の理由】

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

1.10.6. オブジェクトが属するクラスの設定

マルチプロセッサ対応カーネルにおいて，カーネルオブジェクトが属するクラスは，オブジェクトの登録時に決定し，登録後に変更することはできない 【NGKI0395】．

カーネルオブジェクトを静的APIによって登録する場合には，オブジェクトを登録する静的APIを，そのオブジェクトを属させるクラスの囲みの中に記述する【NGKI0396】．クラスに属さないオブジェクトを登録する静的APIは，クラスの囲みの外に記述する（「2.12.4 クラスの指定」の節を参照）【NGKI0397】．

カーネルオブジェクトをサービスコールによって登録する場合には，オブジェクト属性にTA_CLS(clsid)を指定することにより，オブジェクトを属させるクラスを設定する【NGKI0398】．ここでclsidは，そのオブジェクトを属させるクラスのID番号であり，clsidにTCLS_SELF（＝0）を指定するか，オブジェクト属性にTA_CLS(clsid)を指定しないことで，自タスクが属するクラスに属させること ができる．

割付け可能なID番号の数を指定する静的API（AID_YYY）で確保したID番号は，静的APIを囲むクラスに属するオブジェクトにのみ割り付けられる【NGKI0399】．これらの静的APIは，確保したID番号を割り付けるオブジェクトの属すべきクラスの囲みの中に記述しなければならない．クラスの囲みの外に記述した場合には，E_RSATRエラーとなる【NGKI0401】．

【補足説明】

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

【仕様決定の理由】

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

1.10.7. オブジェクトの状態参照

ID番号で識別するオブジェクトのすべてと，オブジェクト番号で識別するオブジェクトの一部に対して，オブジェクトの状態を参照するサービスコール（ref_yyy，get_yyy）を用意する【NGKI0402】．

オブジェクトの状態を参照するサービスコールでは，オブジェクトの登録時に指定し，その後に変化しない情報（例えば，タスクのタスク属性や初期優先度）を参照するための機能は用意しないことを原則とする【NGKI0403】．自タスクの拡張情報の参照するサービスコール（get_inf）は，この原則に対する例外で ある【NGKI0404】．

1.11. オブジェクトのアクセス保護

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

1.11.1. オブジェクトのアクセス保護とアクセス違反の通知

カーネルオブジェクトに対するアクセスは，そのオブジェクトに対して設定されたアクセス許可ベクタによって保護される【NGKI0405】．ただし，アクセス許可ベクタを持たないオブジェクトに対するアクセスは，システム状態に対するアクセス許可ベクタによって保護される【NGKI0406】．また，オブジェクトを登録するサービスコールと，特定のオブジェクトに関連しないシステムの状態に対するアクセスについては，システム状態のアクセス許可ベクタによって 保護される【NGKI0407】．

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

サービスコールにより，メモリオブジェクト以外のカーネルオブジェクトに対して，許可されていないアクセスを行おうとした場合，サービスコールからE_OACVエラーが返る【NGKI0408】．また，メモリオブジェクトに対して，許可されていない管理操作または参照操作を行おうとした場合も，サービスコールからE_OACVエラーが返る【NGKI0409】．

メモリオブジェクトに対して，通常のメモリアクセスにより，許可されていない書込みアクセスまたは読出しアクセス（実行アクセスを含む）を行おうとした場合，CPU例外ハンドラが起動される【NGKI0410】．どのCPU例外ハンドラが起動されるかは，ターゲット定義である【NGKI0411】．ターゲットによっては，エミュレートされたCPU例外ハンドラの場合もある．また，ターゲット定義で，アクセス違反の状況に応じて異なるCPU例外ハンドラが起動される場合もある．この（これらの）CPU例外ハンドラを，メモリアクセス違反ハンドラと呼ぶ．

メモリオブジェクトに対して，サービスコールを通じて，許可されていない書込みアクセスまたは読出しアクセスを行おうとした場合，サービスコールからE_MACVエラーが返るか，メモリアクセス違反ハンドラが起動される【NGKI0412】．E_MACVエラーが返るかメモリアクセス違反ハンドラが起動されるかは，ターゲット定義である【

NGKI0413】.

メモリアクセス違反ハンドラでは，アクセス違反を発生させたアクセスに関する情報（アクセスした番地，アクセスの種別，アクセスした命令の番地など）を参照する方法を，ターゲット定義で用意する【NGKI0414】.

メモリオブジェクトとしてカーネルに登録されていないメモリ領域に対して，ユーザドメインから書込みアクセスまたは読出しアクセス（実行アクセスを含む）を行おうとした場合には，メモリオブジェクトに対するアクセスが許可されていない場合と同様に扱われる【NGKI0415】．カーネルドメインから同様のアクセスを行おうとした場合の動作は保証されない【NGKI0416】.

【未決定事項】

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

【μITRON4.0/PX仕様との関係】

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

【仕様決定の理由】

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

1.11.2. メモリオブジェクトに対するアクセス許可ベクタの制限

メモリオブジェクトの書込みアクセスと読出しアクセス（実行アクセスを含む）に対して設定できるアクセス許可パターンは，ターゲット定義で制限される場 合がある【NGKI0417】.

ただし，少なくとも，次の5つの組み合わせの設定は，行うことができる.

a. メモリオブジェクトが属する保護ドメインのみに，読出しアクセス（実行アクセスを含む）のみを許可する【NGKI0418】．これを，専有リードオンリー（private read only）と呼ぶ.
b. メモリオブジェクトが属する保護ドメインのみに，書込みアクセスと読出しアクセス（実行アクセスを含む）を許可する【NGKI0419】．これを，専有リードライト（private read/write）と呼ぶ.
c. すべての保護ドメインに，読出しアクセス（実行アクセスを含む）のみを許可する【NGKI0420】．これを，共有リードオンリー（shared read only）と呼ぶ.
d. すべての保護ドメインに，書込みアクセスと読出しアクセス（実行アクセスを含む）を許可する【NGKI0421】．これを，共有リードライト（shared read/write）と呼ぶ.
e. メモリオブジェクトが属する保護ドメインに，書込みアクセスと読出しア

クセス（実行アクセスを含む）を許可し，他の保護ドメインには，読出しアクセス（実行アクセスを含む）のみを許可する【NGKI0422】．これを，共有リード専有ライト（shared read private write）と呼ぶ．

また，ターゲット定義で，1つの保護ドメインに登録できるメモリオブジェクトの数が制限される場合がある【NGKI0423】．

1.11.3. 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を登録した直後は，次に規定されるデフォルトのアクセ ス許可ベクタが設定される．

保護ドメインに属するカーネルオブジェクトに対しては，4つの種別のアクセスがいずれも，その保護ドメインのみに許可される【NGKI0424】．すなわち，カーネルドメインに属するオブジェクトに対しては，4つのアクセス許可パターンがいずれもTACP_KERNELに，ユーザドメインに属するオブジェクトに対しては，4つのアクセス許可パターンがいずれもTACP(domid)（domidはオブジェクトが属する保護ドメインのID番号）に設定される．ただし，カーネルオブジェクトをサービスコールにより登録した場合には，管理操作に対するアクセスは，サービスコールを呼び出した処理単位が属する保護ドメインにも許可される 【NGKI3427】．

無所属のカーネルオブジェクトに対しては，4つの種別のアクセスがいずれも，すべての保護ドメインに許可される【NGKI0425】．すなわち，4つのアクセス許可パターンがいずれも，TACP_SHAREDに設定される．

システム状態のアクセス許可ベクタは，4つの種別のアクセスがいずれも，カーネルドメインのみに許可される【NGKI0426】．すなわち，4つのアクセス許可パターンがいずれも，TACP_KERNELに設定される．

1.11.4. アクセス許可ベクタの設定

アクセス許可ベクタをデフォルト以外の値に設定するために，カーネルオブジェクトのアクセス許可ベクタを設定する静的API（SAC_YYY）と，システム状態のアクセス許可ベクタを設定する静的API（SAC_SYS）が用意されている 【NGKI0427】．

また，動的生成対応カーネルにおいては，カーネルオブジェクトのアクセス許可ベクタを設定するサービスコール（sac_yyy）と，システム状態のアクセス許可ベクタを設定するサービスコール（sac_sys）が用意されている【NGKI0428】．ただし，静的APIによって登録したオブジェクトは，サービスコール（sac_yyy）によってアクセス許可ベクタを設定することができない．アクセス許可ベクタを設定しようとした場合には，E_OBJエラーとなる【NGKI0430】．

メモリオブジェクトに対しては，アクセス許可ベクタを設定する静的APIは用意されておらず，オブジェクトの登録と同時にアクセス許可ベクタを設定する静的API（ATA_YYY）が用意されている【NGKI0431】．

オブジェクトに対するアクセスが許可されているかは，そのオブジェクトにアクセスするサービスコールを呼び出した時点でチェックされる【NGKI0432】．

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

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

【使用上の注意】

カーネルオブジェクトのアクセス許可ベクタをデフォルト以外の値に設定する際に，オブジェクトに対して同じ保護ドメインに属する処理単位からアクセスできるようにするには，その保護ドメインからアクセスできることを明示的に 指定する必要がある．

【μITRON4.0/PX仕様との関係】

アクセス許可ベクタを指定してオブジェクトを生成する静的API（CRA_YYY）は廃止し，オブジェクトの登録後にアクセス許可ベクタを設定する静的 API（SAC_YYY）をサポートすることとした．

静的APIによって登録したオブジェクトは，サービスコール（sac_yyy）によってアクセス許可ベクタを設定することができないこととした．

オブジェクトの状態参照するサービスコール（ref_yyy）により，オブジェクトに設定されたアクセス許可ベクタを参照する機能サポートしないこととした． これは，［NGKI0403］の原則に合わせるための修正である．

1.11.5. カーネルの管理領域のアクセス保護

カーネルが動作するために，カーネルの内部で用いるメモリ領域を，カーネルの管理領域と呼ぶ．ユーザタスクからカーネルを保護するためには，カーネルの管理領域にアクセスできるのは，カーネルドメインのみでなければならない．そのため，カーネルの管理領域は，書込みアクセスおよび読出しアクセスが可 能で，4つの種別のアクセスがカーネルドメインのみに許可されたメモリオブジェクト（これを，カーネル専用のメモリオブジェクトと呼ぶ）の中に置かれる 【NGKI0433】．

カーネルの用いるオブジェクト管理領域（カーネルの管理領域に該当する． 「2.10.4 オブジェクト生成に必要なメモリ領域」の節を参照）として，カーネル専用のメモリオブジェクトに含まれないメモリ領域を指定した場合，E_OBJエ ラーとなる【NGKI0434】．また，カーネルの用いるオブジェクト管理領域の先 頭番地にNULLを指定した場合，必要なメモリ領域が，カーネル専用のメモリオ ブジェクトの中に確保される【NGKI0435】．

システムタスクのスタック領域，ユーザタスクのシステムスタック領域，非タスクコンテキスト用のスタック領域は，カーネルの用いるオブジェクト管理領域には該当しないが，カーネルドメインの実行中にのみアクセスされるため，カーネルの用いるオブジェクト管理領域と同様の扱いとなる【NGKI0436】．一

方，ユーザタスクのユーザスタック領域と固定長メモリプール領域は，ユーザドメインの実行中にもアクセスされるため，カーネルの用いるオブジェクト管理領域とは異なる扱いとなる．

1.11.6. ユーザタスクのユーザスタック領域

ユーザタスクが非特権モードで実行する間に用いるスタック領域を，システムスタック領域（「4.1タスク管理機能」の節を参照）と対比させて，ユーザスタック領域と呼ぶ．ユーザスタック領域は，そのタスクと同じ保護ドメインに属する1つのメモリオブジェクトとしてカーネルに登録される【NGKI0437】．ただし，他のメモリオブジェクトとは異なり，次のように扱われる．

タスクのユーザスタック領域に対しては，そのタスクのみが書込みアクセスおよび読出しアクセスを行うことができる【NGKI0438】．そのため，書込みアクセスと読出しアクセス（実行アクセスを含む）に対するアクセス許可パターンは意味を持たない【NGKI0439】．ユーザスタック領域に対して実行アクセスを行えるかどうかは，ターゲット定義である【NGKI0440】．

ただし，上記の仕様を実現するために大きいオーバヘッドを生じる場合には，ターゲット定義で，タスクのユーザスタック領域を，そのタスクが属する保護ドメイン全体からアクセスできるものとする場合がある【NGKI0441】．

【μITRON4.0/PX仕様との関係】

この仕様では，タスクのユーザスタック領域は，そのタスクのみがアクセスで きるものとした．

1.12. システムコンフィギュレーション手順

1.12.1. システムコンフィギュレーションファイル

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

システムコンフィギュレーションファイルには，カーネルの静的API，システムサービスの静的API，保護ドメインの囲み，クラスの囲み，コンフィギュレータに対するINCLUDEディレクティブ，C言語プリプロセッサのインクルードディレクティブ（#include）と条件ディレクティブ（#if, #ifdefなど）のみを記述す ることができる【NGKI0442】．

コンフィギュレータに対するINCLUDEディレクティブは，システムコンフィギュレーションファイルを複数のファイルに分割して記述するために用いるもので，その文法は次のいずれかである（両者の違いは，指定されたファイルを探すディレクトリの違いのみ）【NGKI0443】．

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

コンフィギュレータは，INCLUDEディレクティブによって指定されたファイル中の記述を，システムコンフィギュレーションファイルの一部分として解釈する【NGKI0444】．すなわち，INCLUDEディレクティブによって指定されたファイル中には，カーネルの静的API，システムサービスの静的API，コンフィギュレータに対するINCLUDEディレクティブ，C言語プリプロセッサのインクルードディ レクティブと条件ディレクティブのみを記述することができる．

C言語プリプロセッサのインクルードディレクティブは，静的APIのパラメータを解釈するために必要なC言語のヘッダファイルを指定するために用いる【NGKI0445】．また，条件ディレクティブは，有効とする静的APIを選択するために用いることができる【NGKI0446】．ただし，インクルードディレクティブは，コンフィギュレータが生成するファイルでは先頭に集められる【NGKI0447】．そのため，条件ディレクティブの中にインクルードディレクティブを記述しても，インクルードディレクティブは常に有効となる．また，1つの静的APIの記述の途中に，条件ディレクティブを記述することはできない【NGKI0448】．

コンフィギュレータは，システムコンフィギュレーションファイル中の静的APIを，その記述順に解釈する【NGKI0449】．そのため例えば，タスクを生成する静的APIの前に，そのタスクにタスク例外処理ルーチンを定義する静的APIが記述されていた場合，タスク例外処理ルーチンを定義する静的APIがE_NOEXSエ ラーとなる．

【μITRON4.0仕様との関係】

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

1.12.2. 静的APIの文法とパラメータ

静的APIは，次に述べる例外を除いては，C言語の関数呼出しと同様の文法で記述する【NGKI0450】．すなわち，静的APIの名称に続けて，静的APIの各パラメータを","で区切って列挙したものを"("と")"で囲んで記述し，最後に";"を記述する．ただし，静的APIのパラメータに構造体（または構造体へのポインタ）を記述する場合には，構造体の各フィールドを","で区切って列挙したものを"{" と"}"で囲んだ形で記述する【NGKI0451】．

サービスコールに対応する静的APIの場合，静的APIのパラメータは，対応するサービスコールのパラメータと同一とすることを原則とする【NGKI0452】．

静的APIのパラメータは，次の4種類に分類される．

a. オブジェクト識別名

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

コンフィギュレータは，オブジェクト生成のための静的API（CRE_YYY）を処理する際に，オブジェクトにID番号を割り付け，構成・初期化ヘッダファイルに，指定された識別名を割り付けたID番号にマクロ定義するC言語プリプロセッサのディレクティブ（#define）を生成する【NGKI0453】．

オブジェクト生成以外の静的APIが，オブジェクトのID番号をパラメータに取る場合（カーネルの静的APIでは，SAC_TSKやDEF_TEXのtskidパラメータ等がこれに該当する）には，パラメータとして記述する識別名は，生成済みのオブジェクトの名称を表す識別名でなければならない．そうでない場合には，コンフィギュレータがエラーを報告する【NGKI0455】．

静的APIの整数定数式パラメータの記述に，オブジェクト識別名を使用すること はできない【NGKI0456】．

b. 整数定数式パラメータ

オブジェクト番号や機能コード，オブジェクト属性，サイズや数，優先度など，整数値を指定するパラメータ．プログラムが配置される番地に依存せずに値の決まる整数定数式を記述することができる．

整数定数式の解釈に必要な定義や宣言等は，システムコンフィギュレーション ファイルからC言語プリプロセッサのインクルードディレクティブによってインクルードするファイルに含まれていなければならない【NGKI0457】．

c. 一般定数式パラメータ

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

定数式の解釈に必要な定義や宣言等は，システムコンフィギュレーションファ イルからC言語プリプロセッサのインクルードディレクティブによってインクルードするファイルに含まれていなければならない【NGKI0458】．

d. 文字列パラメータ

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

【μITRON4.0仕様との関係】

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

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

D. 一般定数式パラメータ

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

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

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

1.12.3. 保護ドメインの指定

保護機能対応カーネルでは，オブジェクトを登録する静的API等を，そのオブジェクトが属する保護ドメインの囲みの中に記述する【NGKI0459】．無所属のオブジェクトを登録する静的APIは，保護ドメインの囲みの外に記述する【NGKI0460】．保護ドメインに属すべきオブジェクトを登録する静的API等を，保護ドメインの囲みの外に記述した場合には，コンフィギュレータがE_RSATRエラーを報告する【NGKI0461】．

ユーザドメインの囲みの文法は次の通り【NGKI0462】．

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

保護ドメイン名には，ユーザドメインの名称を表す単一の識別名のみを記述することができる【NGKI0463】．

コンフィギュレータは，ユーザドメインの囲みを処理する際に，ユーザドメインに保護ドメインIDを割り付け，構成・初期化ヘッダファイルに，指定された保護ドメイン名を割り付けた保護ドメインIDにマクロ定義するC言語プリプロセッサのディレクティブ（#define）を生成する【NGKI0464】．また，ユーザドメインの囲みの中およびそれ以降に記述する静的APIの整数定数式パラメータの記述に保護ドメイン名を記述すると，割り付けた保護ドメインIDの値に評価される 【NGKI0465】．

ユーザドメインの囲みの中を空にすることで，ユーザドメインへの保護ドメインIDの割付けのみを行うことができる【NGKI0466】．

カーネルドメインの囲みの文法は次の通り【NGKI0467】．

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

同じ保護ドメイン名を指定したユーザドメインの囲みや，カーネルドメインの囲みを，複数回記述してもよい【NGKI0468】．保護機能対応でないカーネルで保護ドメインの囲みを記述した場合や，保護ドメインの囲みの中に保護ドメインの囲みを記述した場合には，コンフィギュレータがエラーを報告する 【NGKI0469】．

【μITRON4.0/PX仕様との関係】

保護ドメインの囲みの文法を変更した．

【仕様決定の理由】

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

1.12.4. クラスの指定

マルチプロセッサ対応カーネルでは，オブジェクトを登録する静的API等を，そのオブジェクトが属するクラスの囲みの中に記述する【NGKI0470】．クラスに属すべきオブジェクトを登録する静的API等を，クラスの囲みの外に記述した場合には，コンフィギュレータがE_RSATRエラーを報告する【NGKI0471】．

クラスの囲みの文法は次の通り【NGKI0472】．

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

クラスIDには，静的APIの整数定数式パラメータと同等の定数式を記述することができる【NGKI0473】．使用できないクラスIDを指定した場合には，コンフィギュレータがE_IDエラーを報告する【NGKI0474】．

同じクラスIDを指定したクラスの囲みを複数回記述してもよい【NGKI0475】．マルチプロセッサ対応でないカーネルでクラスの囲みを記述した場合や，クラスの囲みの中にクラスの囲みを記述した場合には，コンフィギュレータがエラー を報告する【NGKI0476】．

なお，保護機能とマルチプロセッサの両方に対応するカーネルでは，保護ドメインの囲みとクラスの囲みはどちらが外側になっていてもよい【NGKI0477】．

【仕様決定の理由】

クラスに属すべきオブジェクトを登録する静的API等をクラスの囲みの外に記述

した場合のエラーコードをE_RSATRとしたのは，オブジェクトを動的に登録するAPIにおいては，オブジェクトの属するクラスを，オブジェクト属性によって指 定するためである．

1.12.5. コンフィギュレータの処理モデル

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

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

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

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

パス2で生成されたファイルを，他のソースファイルとあわせてコンパイルし，アプリケーションのロードモジュールを生成する．また，そのSレコードファイル（system.srec）とシンボルファイル（system.syms）を生成する．静的APIの一般定数式パラメータに解釈できない式が記述された場合には，この段階でエ ラーが検出される．

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

保護機能対応カーネルにおいては，メモリ配置を決定し，メモリ保護のための設定情報を生成するために，さらに以下の処理を行う（図2-9）．

コンフィギュレータは，決定したメモリ配置に従ってロードモジュールを生成するために，リンクスクリプト（ldscript.ld）を生成する．また，メモリ保護

のための設定情報を，メモリ構成・初期化ファイル（kernel_mem.c）に生成する．これらのファイルを生成するためには，パス3以降で初めて得られる情報が必要となるため，これらのファイルはパス3以降でしか生成できず，最終的なロードモジュールも，パス3以降で生成する．

そのため，パス2で生成されたロードモジュールは，仮のロードモジュールという位置付けになる．ここで，パス3以降で必要な情報を取り出し，最終的なロードモジュールのサイズを割り出せるように，パス3以降でメモリ構成・初期化ファイルに生成するのと同様のデータ構造を，パス2において仮のメモリ構成・初期化ファイル（kernel_mem2.c）に生成する．また，これをリンクするための仮のリンクスクリプト（cfg2_out.ld）を生成し，これらを用いて仮のロードモジュールを生成する．さらに，仮のロードモジュールのSレコードファイル（cfg2_out.srec）とシンボルファイル（cfg2_out.syms）も，最終的なものと混同しないように，異なるファイル名で生成する．

パス3は，ターゲット依存で用いるパスで，メモリ配置やメモリ保護のための設定情報のサイズを最適化するための処理を行う．パス2で生成された仮のロードモジュールのSレコードファイルとシンボルファイルから必要な情報を取り出し，再度，仮のメモリ構成・初期化ファイル（kernel_mem3.c）と仮のリンクスクリプト（cfg3_out.ld）を生成する．また，これらのファイルを他のソースファイルとあわせてコンパイルして仮のロードモジュールを生成し，そのSレコードファイル（cfg3_out.srec）とシンボルファイル（cfg3_out.syms）を生成する．この段階で，メモリオブジェクトに重なりがあるなどのエラーが検出される場合 もある．

パス4では，パス3（パス3を用いない場合はパス2）で生成された仮のロードモジュールのSレコードファイルとシンボルファイルから必要な情報を取り出し，最終的なメモリ構成・初期化ファイル（kernel_mem.c）とリンクスクリプト（ldscript.ld）を生成する．またパス4では，保護機能対応でないカーネルにおいてパス3で行っていた静的APIパラメータの値などの妥当性のチェックも行う．そのため，静的APIの一般定数式パラメータに不正がある場合には，この段 階でエラーが検出される．

パス4で生成されたファイルを，他のソースファイルとあわせてコンパイルし，アプリケーションの最終的なロードモジュールを生成する．また，そのSレコードファイル（system.srec，必要な場合のみ）とシンボルファイル（system.syms）を生成する．

最後に，最終的なロードモジュールが，パス3（パス3を用いない場合はパス2）で生成された仮のロードモジュールと同じメモリ配置であることをチェックする．両者のメモリ配置が異なっていた場合には，ロードモジュールが正しく生成されていない可能性があるが，これは，コンフィギュレーション処理の不具 合を示すものである．

【μITRON4.0仕様との関係】

コンフィギュレータの処理モデルは全面的に変更した．

1.12.6. 静的APIのパラメータに関するエラー検出

静的APIのパラメータに関するエラー検出は，同じものがサービスコールとして

呼ばれた場合と同等とすることを原則とする【NGKI0478】．言い換えると，サービスコールによっても検出できないエラーは，静的APIにおいても検出しない．静的APIの機能説明中の「E_XXXXXエラーとなる」または「E_XXXXXエラーが返る」という記述は，コンフィギュレータがそのエラーを検出することを意味する．

ただし，エラーの種類によっては，サービスコールと同等のエラー検出を行うことが難しいため，そのようなものについては例外とする【NGKI0479】．例えば，メモリ不足をコンフィギュレータによって検出するのは容易ではない．

逆に，オブジェクト属性については，サービスコールより強力なエラーチェックを行える可能性がある．例えば，タスク属性にTA_STAと記述されている場合，サービスコールではエラーを検出できないが，コンフィギュレータでは検出できる可能性がある．ただし，このようなエラー検出を完全に行おうとするとコンフィギュレータが複雑になるため，このようなエラーを検出することは必須とせず，検出できた場合には警告として報告する【NGKI0480】．

【μITRON4.0仕様との関係】

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

1.12.7. オブジェクトのID番号の指定

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

コンフィギュレータのオプション指定により，オブジェクト識別名とID番号の対応表を含むファイルを渡すと，コンフィギュレータはそれに従ってオブジェクトにID番号を割り付ける【NGKI0481】．それに従ったID番号割付けができない場合（ID番号に抜けができる場合など）には，コンフィギュレータはエラー を報告する【NGKI0482】．

またコンフィギュレータは，オプション指定により，オブジェクト識別名とコンフィギュレータが割り付けたID番号の対応表を含むファイルを，コンフィギュレータに渡すファイルと同じフォーマットで生成する【NGKI0483】．

【μITRON4.0仕様との関係】

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

1.13. TOPPERSネーミングコンベンション

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

1.13.1. モジュール識別名

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

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

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

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

1.13.2. データ型名

各サイズの整数型など，データの意味を定めない基本データ型の名称は，英小 文字，数字，"_"で構成する．データ型であることを明示するために，末尾が "_t"である名称とする．

複合データ型やデータの意味を定めるデータ型の名称は，英大文字，数字，" "で構成する．データ型であることを明示するために，先頭が"*T*"または末尾 が"_T"である名称とする場合もある．

データ型の種類毎に，次のネーミングコンベンションを定める．

A. パケットのデータ型

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

1.14. 関数名

関数の名称は，英小文字，数字，"_"で構成する．

関数の種類毎に，次のネーミングコンベンションを定める．

A. サービスコール

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

またはwww_zzxxx_yyyの名称となる．

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

【補足説明】

サービスコールの名称を構成する省略名（xxx，yyy，z）の元になった英語につ　　いては，「5.10 省略名の元になった英語」の節を参照すること．

B. コールバック

コールバックの名称は，サービスコールのネーミングコンベンションに従う．

1.14.1. 変数名

変数（const修飾子のついたものを含む）の名称は，英小文字，数字，"_"で構成する．データ型が異なる変数には，異なる名称を付けることを原則とする．

変数の名称に関して，次のガイドラインを設ける．

```
〜id     〜ID（オブジェクトのID番号，ID型）
〜no     〜番号（オブジェクト番号）
〜atr        〜属性（オブジェクト属性，ATR型）
〜stat       〜状態（オブジェクト状態，STAT型）
〜mode       〜モード（サービスコールの動作モード，MODE型）
〜pri        〜優先度（優先度，PRI型）
〜sz     〜サイズ（単位はバイト数，SIZE型またはuint_t型）
〜cnt        〜の個数（単位は個数，uint_t型）
〜ptn        〜パターン
〜tim        〜時刻，〜時間
〜cd     〜コード
i〜           〜の初期値
max〜        〜の最大値
min〜        〜の最小値
left〜       〜の残り
```

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

```
p_〜     ポインタ
pp_〜        ポインタを入れる領域へのポインタ
pk_〜        パケットへのポインタ
ppk_〜       パケットへのポインタを入れる領域へのポインタ
```

変数の種類毎に，次のネーミングコンベンションを定める．

(A) パケットへのポインタ

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

1.14.2. 定数名

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

定数の種類毎に，次のネーミングコンベンションを定める．

(A) メインエラーコード

メインエラーコードは，先頭が"E_"である名称とする．

(B) 機能コード

```
TFN_XXX_YYY         xxx_yyyの機能コード
TFN_WWW_XXX_YYY     www_xxx_yyyの機能コード
```

(C) その他の定数

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

また，定数の名称に関して，次のガイドラインを設ける．

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

1.14.3. マクロ名

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

マクロの種類毎に，次のネーミングコンベンションを定める．

(A) 構成マクロ

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

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

1.14.4. 静的API名

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

1.14.5. ファイル名

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

ファイルの種類毎に，次のネーミングコンベンションを定める．

(A) ヘッダファイル

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

1.14.6. モジュール内部の名称の衝突回避

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

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

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

1.15. TOPPERS共通定義

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

1.15.1. TOPPERS共通ヘッダファイル

TOPPERS共通定義（共通データ型，共通定数，共通マクロ）は，TOPPERS共通ヘッダファイル（t_stddef.h）およびそこからインクルードされるファイルに含まれている【NGKI0484】．TOPPERS共通定義を用いる場合には，TOPPERS共通ヘッダファイルをインクルードする【NGKI0485】．

TOPPERS共通ヘッダファイルは，カーネルヘッダファイル（kernel.h）やシステムインタフェースレイヤヘッダファイル（sil.h）からインクルードされるため，これらのファイルをインクルードする場合には，TOPPERS共通ヘッダファイルを直接インクルードする必要はない【NGKI0486】．

1.15.2. TOPPERS共通データ型

C90に規定されているデータ型以外で，TOPPERSソフトウェアで共通に用いるデータ型は次の通りである【NGKI0487】．

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

```
int_least8_t    8ビット以上の符号付き整数（C99準拠）
uint_least8_t   int_least8_t型と同じサイズの符号無し整数（C99準拠）
```

```
float32_t   IEEE754準拠の32ビット単精度浮動小数点数（オプション）
double64_t  IEEE754準拠の64ビット倍精度浮動小数点数（オプション）
```

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

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

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

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

FP	プログラムの起動番地（型の定まらない関数ポインタ）

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

MB_T	オブジェクト管理領域を確保するためのデータ型

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

ここで，データ型が「AまたはB」とは，AかBのいずれかの値を取ることを示す．例えばER_BOOLは，エラーコードまたは真偽値のいずれかの値を取る．

int8_t，uint8_t，int64_t，uint64_t，int128_t，uint128_t，float32_t，double64_tが使用できるかどうかは，ターゲット定義である【NGKI0488】．これらが使用できるかどうかは，それぞれ，INT8_MAX，UINT8_MAX，INT64_MAX，UINT64_MAX，INT128_MAX，UINT128_MAX，FLOAT32_MAX，DOUBLE64_MAXがマクロ

定義されているかどうかで判別することができる【NGKI0489】．IEEE754準拠の浮動小数点数がサポートされていない場合には，ターゲット定義で，float32_tとdouble64_tは使用できないものとする【NGKI0490】．

【μITRON4.0仕様との関係】

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

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

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

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

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

1.15.3. TOPPERS共通定数

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

(1) 一般定数【NGKI0491】

```
NULL                    無効ポインタ
```

```
true            1       真
false           0       偽
```

```
E_OK            0       正常終了
```

【μITRON4.0仕様との関係】

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

(2) 整数型に格納できる最大値と最小値【NGKI0492】

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

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

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

(3) 整数型のビット数【NGKI0493】

```
CHAR_BIT            char型のビット数（C90準拠）
```

(4) オブジェクト属性【NGKI0494】

```
TA_NULL     0U      オブジェクト属性を指定しない
```

(5) タイムアウト指定【NGKI0495】

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

(6) アクセス許可パターン【NGKI0496】

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

1.15.4. TOPPERS共通エラーコード

TOPPERSソフトウェアで共通に用いるメインエラーコードは次の通りである 【NGKI0497】.

(A) 内部エラークラス（EC_SYS，-5〜-8）

```
E_SYS       -5      システムエラー
```

(B) 未サポートエラークラス（EC_NOSPT，-9〜-16）

```
E_NOSPT     -9      未サポート機能
E_RSFN      -10     予約機能コード
E_RSATR     -11     予約属性
```

(C) パラメータエラークラス（EC_PAR，-17〜-24）

```
E_PAR       -17     パラメータエラー
E_ID        -18     不正ID番号
```

(D) 呼出しコンテキストエラークラス（EC_CTX，-25〜-32）

```
E_CTX       -25     コンテキストエラー
E_MACV      -26     メモリアクセス違反
E_OACV      -27     オブジェクトアクセス違反
E_ILUSE     -28     サービスコール不正使用
```

(E) 資源不足エラークラス（EC_NOMEM，-33～-40）

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

(F) オブジェクト状態エラークラス（EC_OBJ，-41～-48）

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

(G) 待ち解除エラークラス（EC_RLWAI，-49～-56）

```
E_RLWAI     -49     待ち禁止状態または待ち状態の強制解除
E_TMOUT     -50     ポーリング失敗またはタイムアウト
E_DLT       -51     待ちオブジェクトの削除または再初期化
E_CLS       -52     待ちオブジェクトの状態変化
```

(H) 警告クラス（EC_WARN，-57～-64）

```
E_WBLK      -57     ノンブロッキング受付け
E_BOVR      -58     バッファオーバフロー
```

このエラークラスに属するエラーコードは，警告を表すエラーコードであり，［NGKI0019］の原則では例外としている．

【μITRON4.0仕様との関係】

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

1.15.5. TOPPERS共通マクロ

(1) 整数定数を作るマクロ【NGKI0498】

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

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

【仕様決定の理由】

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

(2) 型に関する情報を取り出すためのマクロ【NGKI0499】

```
offsetof(structure, field)  構造体structure中のフィールドfieldの
                            バイト位置を返すマクロ（C90準拠）
```

```
alignof(type)               型typeのアラインメント単位を返すマクロ
```

```
ALIGN_TYPE(addr, type)      番地addrが型typeに対してアラインしてい
                            るかどうかを返すマクロ
```

(3) assertマクロ【NGKI0500】

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

(4) コンパイラの拡張機能のためのマクロ【NGKI0501】

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

(5) エラーコード構成・分解マクロ【NGKI0502】

```
ERCD(mercd, sercd)  メインエラーコードmercdとサブエラーコードsercdか
                    ら，エラーコードを構成するためのマクロ
```

```
MERCD(ercd)         エラーコードercdからメインエラーコードを抽出する
                    ためのマクロ
SERCD(ercd)         エラーコードercdからサブエラーコードを抽出するた
                    めのマクロ
```

(6) アクセス許可パターン構成マクロ【NGKI0503】

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

ここで，TACPのパラメータ（domid）には，ユーザドメインのID番号のみを指定することができる【NGKI0504】．TDOM_SELF，TDOM_KERNEL，TDOM_NONEを指定した場合，どのようなアクセス許可パターンが構成されるかは保証されない 【NGKI0505】．

1.15.6. TOPPERS共通構成マクロ

(1) 相対時間の範囲【NGKI0506】

```
TMAX_RELTIM     相対時間に指定できる最大値
```

2.15 カーネル共通定義

カーネルの複数の機能で共通に用いる定義を，カーネル共通定義と呼ぶ.

2.15.1 カーネルヘッダファイル

カーネルを用いるために必要な定義は，カーネルヘッダファイル（kernel.h）およびそこからインクルードされるファイルに含まれている【NGKI0507】．カーネルを用いる場合には，カーネルヘッダファイルをインクルードする 【NGKI0508】．

ただし，カーネルを用いるために必要な定義の中で，コンフィギュレータによって生成されるものは，カーネル構成・初期化ヘッダファイル（kernel_cfg.h）に含まれる【NGKI0509】．具体的には，登録できるオブジェクトの数（TNUM_YYY）やオブジェクトのID番号などの定義が，これに該当する．これらの定義を用いる場合には，カーネル構成・初期化ヘッダファイルをインクルードする【NGKI0510】．

μITRON4.0仕様で規定されており，この仕様で廃止されたデータ型および定数を用いる場合には，ITRON仕様互換ヘッダファイル（itron.h）をインクルードする【NGKI0511】．

【μITRON4.0仕様との関係】

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

1.15.7. カーネル共通定数

(1) オブジェクト属性【NGKI0512】

```
TA_TPRI         0x01U   タスクの待ち行列をタスクの優先度順に
```

【μITRON4.0仕様との関係】

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

(2) 保護ドメインID【NGKI0513】

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

(3) その他のカーネル共通定数【NGKI0514】

```
TCLS_SELF       0       自タスクの属するクラス
```

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

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

```
TPRI_SELF       0       自タスクのベース優先度の指定
TPRI_INI        0       タスクの起動時優先度の指定
```

```
TIPM_ENAALL     0       割込み優先度マスク全解除
```

(4) カーネルで用いるメインエラーコード

「TOPPERS共通エラーコード」の節で定義したメインエラーコードの中　で，E_CLS，E_WBLK，E_BOVRの3つは，カーネルでは使用しない【NGKI0515】．

【TOPPERS/ASPカーネルにおける規定】

ASPカーネルでは，サービスコールから，E_RSFN，E_RSATR，E_MACV，E_OACV，E_NOMEM，E_NOID，E_NORES，E_NOEXSが返る状況は起こらない【ASPS0011】．E_RSATRは，コンフィギュレータによって検出される【ASPS0012】．ただし，動的生成機能拡張パッケージでは，サービスコールから，E_RSATR，E_NOMEM，E_NOID，E_NOEXSが返る状況が起こる【ASPS0013】．

【TOPPERS/FMPカーネルにおける規定】

FMPカーネルでは，サービスコールから，E_RSFN，E_RSATR，E_MACV，E_OACV，E_NOMEM，E_NOID，E_NORES，E_NOEXSが返る状況は起こらない【FMPS0007】．E_RSATRとE_NORESは，コンフィギュレータによって検出される【FMPS0008】．

【TOPPERS/HRP2カーネルにおける規定】

HRP2カーネルでは，サービスコールから，E_RSATR，E_NOMEM，E_NOID，E_NORES，E_NOEXSが返る状況は起こらない【HRPS0006】．E_RSATRは，コンフィギュレータによって検出される【HRPS0007】．ただし，動的生成機能拡張パッケージでは，サービスコールから，E_RSATR，E_NOMEM，E_NOID，E_NOEXSが返る状況が起こる【HRPS0011】．

【TOPPERS/SSPカーネルにおける規定】

SSPカーネルでは，サービスコールから，E_RSFN，E_RSATR，E_MACV，E_OACV，E_ILUSE，E_NOMEM，E_NOID，E_NORES，E_NOEXS，E_RLWAI，E_TMOUT，E_DLTが返る状況は起こらない【SSPS0008】．E_RSATRは，コンフィギュレータによって検 出される【SSPS0009】．

1.15.8. カーネル共通マクロ

(1) スタック領域をアプリケーションで確保するためのデータ型とマクロ

スタック領域をアプリケーションで確保するために，次のデータ型とマクロを 用意している【NGKI0516】.

STK_T	スタック領域を確保するためのデータ型

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

これらを用いてスタック領域を確保する方法は次の通り【NGKI0517】.

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

この方法で確保したスタック領域を，サービスコールまたは静的APIに渡す場合には，スタック領域の先頭番地に<スタック領域の変数名>を，スタック領域の サイズにROUND_STK_T(<スタック領域のサイズ>)を指定する【NGKI0518】.

ただし，保護機能対応カーネルにおいては，上の方法によりタスクのユーザスタック領域を確保することはできない【NGKI0519】. 詳しくは，「4.1 タスク 管理機能」の節のCRE_TSKの機能の項を参照すること.

(2) オブジェクト属性を作るマクロ

保護機能対応カーネルでは，オブジェクトが属する保護ドメインを指定するためのオブジェクト属性を作るマクロとして，次のマクロを用意している 【NGKI0520】.

TA_DOM(domid)	domidで指定される保護ドメインに属する

マルチプロセッサ対応カーネルでは，オブジェクトが属するクラスを指定するためのオブジェクト属性を作るマクロとして，次のマクロを用意している 【NGKI0521】.

TA_CLS(clsid)	clsidで指定されるクラスに属する

(3) サービスコールの呼出し方法を指定するマクロ

保護機能対応カーネルでは，サービスコールの呼出し方法を指定するためのマクロとして，次のマクロを用意している【NGKI0522】.

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

1.15.9. カーネル共通構成マクロ

(1) サポートする機能【NGKI0523】

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

【未決定事項】

マクロ名は，今後変更する可能性がある．

(2) 優先度の範囲【NGKI0524】

```
TMIN_TPRI        タスク優先度の最小値（＝1）
TMAX_TPRI        タスク優先度の最大値
```

【TOPPERS/ASPカーネルにおける規定】

ASPカーネルでは，タスク優先度の最大値（TMAX_TPRI）は16に固定されている 【ASPS0014】．ただし，タスク優先度拡張パッケージを用いると，TMAX_TPRIを256に拡張することができる【ASPS0015】．

【TOPPERS/FMPカーネルにおける規定】

FMPカーネルでは，タスク優先度の最大値（TMAX_TPRI）は16に固定されている 【FMPS0009】．

【TOPPERS/HRP2カーネルにおける規定】

HRP2カーネルでは，タスク優先度の最大値（TMAX_TPRI）は16に固定されている 【HRPS0008】．

【TOPPERS/SSPカーネルにおける規定】

SSPカーネルでは，タスク優先度の最大値（TMAX_TPRI）は16に固定されている 【SSPS0010】．

【μITRON4.0仕様との関係】

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

(3) プロセッサの数

マルチプロセッサ対応カーネルでは，プロセッサの数を知るためのマクロとして，次の構成マクロを用意している【NGKI0525】．

```
TNUM_PRCID       プロセッサの数
```

(4) 特殊な役割を持ったプロセッサ

マルチプロセッサ対応カーネルでは，特殊な役割を持ったプロセッサを知るためのマクロとして，次の構成マクロを用意している【NGKI0526】.

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

(5) タイマ方式

マルチプロセッサ対応カーネルでは，システム時刻の方式を知るためのマクロとして，次の構成マクロを用意している【NGKI0527】.

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

(6) バージョン情報【NGKI0528】

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

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

カーネルの識別番号（TKERNEL_PRID）は，TOPPERSカーネルの種類を表す.

```
0x0001      TOPPERS/JSPカーネル
0x0002      予約（IIMPカーネル）
0x0003      予約（IDLカーネル）
0x0004      TOPPERS/FI4カーネル
0x0005      TOPPERS/FDMPカーネル
0x0006      TOPPERS/HRPカーネル
0x0007      TOPPERS/ASPカーネル
0x0008      TOPPERS/FMPカーネル
0x0009      TOPPERS/SSPカーネル
0x000a      TOPPERS/ASP Safetyカーネル
```

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

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