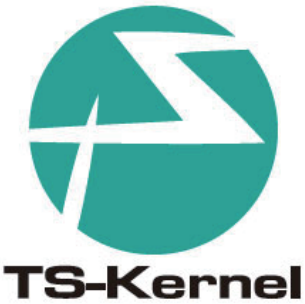

TRON Safe Kernel WG報告書

2017年12月

トロンフォーラム

http://www.tron.org/

TRON Safe Kernel WG報告書 (Ver.1.00.00)

トロンフォーラム事務局
〒141-0031
東京都品川区西五反田2-12-3 第一誠実ビル 9F
YRPユビキタス・ネットワーキング研究所内
TEL: 03-5437-0572
FAX: 03-5437-2399
E-mail: office@tron.org

― 目次 ―

本仕様書の記述形式

本仕様書の記述形式を記す。

■補足事項

仕様書の本文中に、仕様に対する補足事項を記す。補足事項は【】に囲まれた項目名で示す。

以下の項目がある。

【補足説明】の項は、仕様の記述に対して、仕様を理解するための補足的な説明を記す。

【ハードウェア制約】の項は、ハードウェアの機能制約により生じる TRON Safe Kernel の機能制約を記す。

■API 説明の記述形式

仕様書中の API 説明は、以下の項目により API 仕様の説明を行っている。なお、全 API にすべての項目が記載されるわけではない。

API 説明

API の名称および説明を示す。

C 言語インタフェース

【安全 API】

安全 API の C 言語インタフェースを示す。

【通常 API】

通常 API の C 言語インタフェースを示す。

【初期登録 API】

初期登録 API の C 言語インタフェースを示す。

パラメータ

API を発行するときに TRON Safe Kernel に渡す情報に関する説明を行う。

リターンパラメータ

API の実行が終ったときに TRON Safe Kernel から返される情報に関する説明を行う。

なお、リターンパラメータのうち、API の関数値として戻されるものを「戻り値」と呼ぶ。

リターンパラメータには、戻り値のほかに、パラメータとして渡されたポインタの参照先に情報を返すものがある。

エラーコード

API で発生する可能性のあるエラーに関して説明を行う。

利用可能なコンテキスト

API を発行することができるコンテキストを示す。コンテキストは以下に分ける。

タスク部: タスク部実行中、または準タスク部実行中

タイムイベントハンドラ:　　タイムイベントハンドラ（周期ハンドラまたはアラームハンドラ）実行中

タスク独立部:　　タスク部、準タスク部、タイムイベントハンドラ以外の実行中

発行可否を表す記号の意味は以下の通りである。

○　　そのコンテキストから実行可能である。

×　　そのコンテキストから実行するとエラーE_CTX となる。
（API のリターンパラメータで E_CTX を返せない場合は、各 API の説明に発生時動作を記載）

解説

API の機能の解説を行う。

本項では、API 毎について説明する。パラメータパケットやエラーコードなど、API に共通の仕様については本項では解説しない。共通の仕様に関しては「2.4 API」を参照すること。

安全 API(ts_xxx_yyy)と通常 API(tn_xxx_yyy)を一括で示すときは、ts_xxx_yyy/tn_xxx_yyy と表示する。

いくつかの値を選択して設定するようなパラメータの場合には、以下のような記述方法によって仕様説明を行っている。

(x || y || z)
　　x, y, z のいずれか一つを選択して指定する。

x | y
　　x と y を同時に指定可能である。（同時に指定する場合は x と y の論理和をとる）

[x]
　　x は指定しても指定しなくても良い。

パラメータの記述例
　　wfmode := (TWF_ANDW || TWF_ORW) | [TWF_CLR]

の場合、wfmode の指定は次の 4 種のいずれかになる。
　　TWF_ANDW
　　TWF_ORW
　　TWF_ANDW | TWF_CLR
　　TWF_ORW | TWF_CLR

【補足事項】

解説に対する補足事項を述べる。

1. TRON Safe Kernel の概要

1.1 TRON Safe Kernel の位置付け

トロンプロジェクトでは、2011 年に公開された T-Kernel2.0 をはじめ、μTRON、T-Kernel などの組込システム向けのリアルタイム OS の仕様策定を行ってきた。

今後、組込ソフトウェアの分野では機能安全が重要となってくると考えられる。そこで、トロンの OS を発展させる形で機能安全に対応した新たな OS である TRON Safe Kernel の仕様を策定することとした。

1.2 TRON Safe Kernel の基本方針

TRON Safe Kernel の基本方針を以下に示す。

(1) TRON Safe Kernel は、高水準の機能安全ソフトウェアの OS として使用されることを前提に、OS 自身としての機能安全の実現、および OS 上で動作するソフトウェアに対する安全機能の提供を行う。

安全水準は、IEC61508 規格の SIL3 を想定する。

(2) TRON Safe Kernel は、目的の機能安全水準を満たしたユーザプログラムを実行できる。また機能安全水準を満たしたユーザプログラムの実装が容易となる機能を提供する。

(3) TRON Safe Kernel は、機能安全水準を満たさないユーザプログラムを実行できる。これらのユーザプログラムに対して、(1)、(2)の実現に支障のない範囲において、T-Kernel2.0 と同等の機能を提供する。

(4) TRON Safe Kernel は、機能安全水準の異なるユーザプログラムをお互いの影響なく実行できる。これらのユーザプログラムは、(2)の機能安全水準を満たしたユーザプログラムと、(3)の機能安全水準を満たさないユーザプログラムを想定する。

1.3 TRON Safe Kernel の安全要求

TRON Safe Kernel が実現する機能安全の要求仕様を以下に示す。

(1) OS のソフトウェア自体の機能安全性

TRON Safe Kernel のソフトウェア自体が、想定する IEC61508 規格の SIL3 にて開発されること。よって、この規格を満たす OS の仕様でなくてはならない。

(2) 基本的な安全機能の実現

基本的な安全機能として、ハードウェアやソフトウェアの異常動作を検出し、OS として安全な状態に移行する機能を有する。

また、ハードウェアやソフトウェアの異常を検出するために、自己診断を実行する機能を有する。

これらの機能は、異常例外機能と故障診断機能として実現される。

(3) 機能安全アプリケーション実現のための安全機能

非安全を含む安全水準の異なった複数のアプリケーションを安全に実行できる機能を提供する。本機能はドメイン管理機能として実現される。

1.4 想定するハードウェア

TRON Safe Kernel は、以下に規定する標準的なハードウェアにおいて全機能を実現できる。一部のハードウェア機能は、それが欠如した場合、TRON Safe Kernel の機能に制約が生じる。具体的な制約は【ハードウェア制約】に記す。

(1) プロセッサ

プログラムを実行するハードウェアである。プロセッサの内部で単一のプログラムを実行する単位を「プロセッサコア」または単に「コア」と呼ぶ。複数のコアから構成されるプロセッサを「マルチコアプロセッサ」と呼ぶ。

TRON Safe Kernel では、単一のコアを持つプロセッサのみを対象とし、マルチコアプロセッサは対象としない。

プロセッサは特権的なモード、非特権的なモードといった二段階、またはそれ以上の実行モードを有する。非特権的モードではプログラムから周辺デバイスへのアクセスが禁止できる。

(2) メモリ

プロセッサのアドレス空間に物理的に割り当てられたメモリである。

プロセッサは、MMU(Memory Management Unit: メモリ管理ユニット)や MPU(Memory Protection Unit:メモリ保護ユニット)などの、ソフトウェアから特定のメモリ領域を保護するハードウェア機能を持つ。

また、前述のプロセッサはプログラムの実行モードにおいて、特権的なモードからのみアクセス可能なメモリ領域が実現できる。

(3) システムタイマ

TRON Safe Kernel が時間・時刻を計測するために使用するハードウェアのタイマである。定めた時間間隔で周期的に、または指定した時刻にタイマ割込みを発生できる。

(4)最高優先度タイマ

システムソフトウェアが使用する他の割込みよりも優先度の高い割り込みを発生できるタイマである。もしくは、システムソフトウェアが使用する他の割込みがマスクされている場合であっても、割込みを発生できるタイマである。

最高優先度タイマは、システムソフトウェアが正常に実行できない状態になった際に、それを検出するウォッチドックタイマとして使用する。

(5) 周辺デバイス

プログラムから操作可能なハードウェアである。プロセッサに内蔵または外部に接続されるものがある。TRON Safe Kernel から直接的に周辺デバイスを制御することはない。デバイスドライバなどのユーザ定義のソフトウェアを介して制御を行う。

2. TRON Safe Kernel の概念

2.1 基本的な用語

本仕様書で使用する基本的な用語を以下のように定める。

(1) 実装依存と実装定義

TRON Safe Kernel が動作するハードウェア、またはシステムの動作条件によって振舞いが変わる事項を「実装依存」と呼ぶ。実装依存の仕様は、実装毎に振舞いを規定しなければならない。これを「実装定義」と呼ぶ。実装定義は、具体的な実装内容を実装仕様書に明記しなければならない。

(2) コンフィギュレーションとコンフィギュレーション定義ファイル

TRON Safe Kernel のシステムに関する各種設定や、初期状態で組み込まれるソフトウェアの定義を行うことを「コンフィギュレーション」と呼ぶ。コンフィギュレーションに関する情報は「コンフィギュレーション定義ファイル」に記述される。コンフィギュレーションに関する情報は、プログラムの実行により変更することはできない。

【補足説明】

前項の実装定義との相違点は、実装定義はプログラムコードの変更を必要とするのに対し、コンフィギュレーションは、プログラムコードは変更なく、設定値、初期値などのデータ、初期状態で組み込まれるシステムソフトウェアを指定するだけである。

(3) 割込みと異常例外

プロセッサが検出するイベントにより、実行中のプログラムとは別にプログラムを起動する処理を、一般的に割込みや例外と呼ぶ。TRON Safe Kernel では、アドレス違反や未定義命令実行などのプログラムの実行が困難となるイベントにより起動される処理を「異常例外」と呼び、その他を「割込み」と呼ぶ。具体的な異常例外や割込みのイベントは実装定義とする。

【補足説明】

「割込み」と「例外」の用語の使い分けは、プロセッサの仕様においても異なる。TRON Safe Kernel では混乱を避けるため、仕様上では「例外」という名称は「異常例外」でのみ使用する。それ以外の、プロセッサの仕様などで規定される一般的な割込みや例外はまとめて「割込み」と呼ぶ。

(4) コンテキスト

プログラムの実行される環境を「コンテキスト」と呼ぶ。本仕様において、コンテキストが同じというためには、少なくとも用いているアドレス空間が同一であり、スタック領域が共通でなければならない。

2.2 ソフトウェアとドメイン

2.2.1 安全ソフトウェアと通常ソフトウェア

TRON Safe Kernel では、機能安全の観点から、ソフトウェアを安全ソフトウェアと通常ソフトウェアの二種類に分ける。安全ソフトウェアは、機能安全の観点から高い信頼性と安全性が求められるソフトウェアである。通常ソフトウェアは、それ以外のソフトウェアである。

安全ソフトウェアの安全水準は実装定義で定められる。安全ソフトウェアは、実装定義された安全水準を満たさなければならない。

2.2.2 アプリケーションとシステムソフトウェア

TRON Safe Kernel では、プログラムはアプリケーションとシステムソフトウェアの二種類に大別される。

アプリケーションは、TRON Safe Kernel 上で実行されるユーザが作成するソフトウェアである。

システムソフトウェアは、TRON Safe Kernel 自体と、ユーザ定義システムソフトウェアから構成される。ユーザ定義システムソフトウェアは、ユーザが対象となるハードウェアやアプリケーションに応じて作成し、システムソフトウェアに登録したプログラムである。

システムソフトウェアは必ず安全ソフトウェアである。

アプリケーションは安全ソフトウェアと通常ソフトウェアに分かれる。安全ソフトウェアのみのアプリケーションも許される。

システムソフトウェアは、アプリケーションよりも特権的に実行される。つまり、プロセッサが提供するプログラムの実行モードのうち、システムソフトウェアは特権的なモードで実行され、アプリケーションは非特権的なモードで実行される。

【ハードウェア制約】

プロセッサが特権、非特権といった複数のプログラムの実行モードを持たない場合、プログラムはシステムソフトウェアのみとし、アプリケーションは存在できない。ユーザが作成できるプログラムは、ユーザ定義システムソフトウェアのみとなる。

2.2.3 ドメイン

TRON Safe Kernel では、ソフトウェアとそのソフトウェアが使用する資源を、ドメインという単位で管理する。ドメインは複数存在し、ソフトウェアはいずれかのドメインに所属する。

あるドメインに所属するソフトウェアの安全水準は同一でなくてはならない。

システムソフトウェアが所属するドメインをシステムドメインと呼ぶ。

安全ソフトウェアが所属するドメインは、広義の安全ドメインという。広義の安全ドメインは、システムドメインと、アプリケーションの安全ソフトウェアが所属するドメインである。後者は、狭義の安全ドメインまたは単に安全ドメインと呼ぶ。

通常ソフトウェアが所属するドメインは、通常ドメインという。通常ドメインに所属するソフトウェアはすべてアプリケーションである。

以上より TRON Safe Kernel のドメインは以下の種類に分類される。

(1) システムドメイン

システムソフトウェアが所属するドメインである。アプリケーションはシステムドメインに所属できない。

システムソフトウェアは安全ソフトウェアであり、よってシステムドメインは広義の安全ドメインである。

システムドメインはただ一つ存在する。

(2) 安全ドメイン

アプリケーションの安全ソフトウェアが所属するドメインである。

安全ドメインは 1 つ、またはそれ以上が存在する。安全ドメインの最大数は実装定義である。

(3) 通常ドメイン

アプリケーションの通常ソフトウェアが所属するドメインである。

通常ドメインは通常は 0 以上の数が存在する。通常ドメインが存在しない場合もありうる。この場合は通常ソフトウェアも存在しない。通常ドメインの最大数は実装定義である。

安全ドメインと通常ドメインを総称して、アプリケーションドメインと呼ぶ。

ドメインとソフトウェアの関係を「図 2-1 ドメインとソフトウェアの関係」に示す。

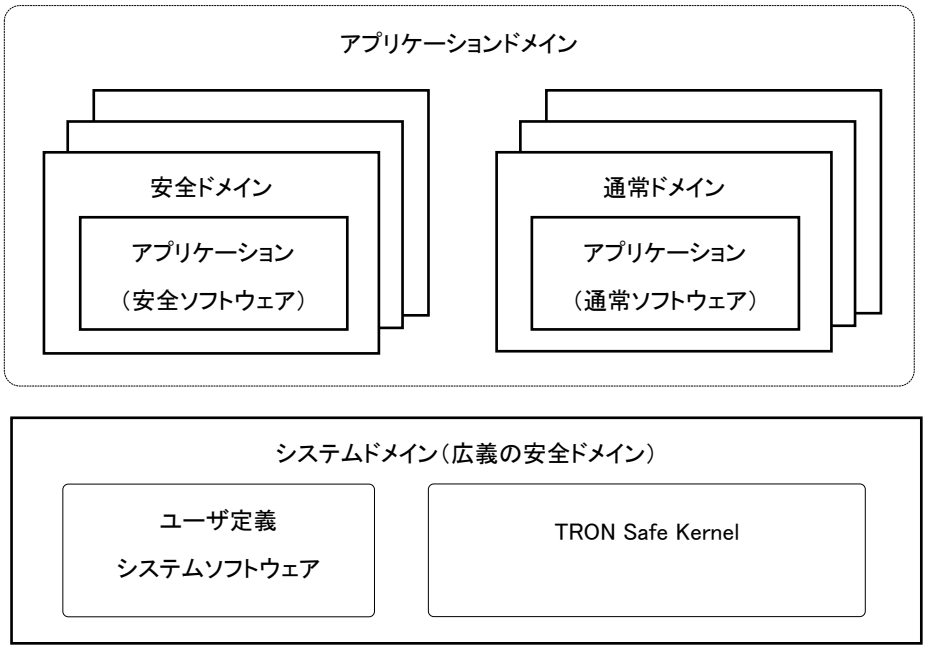

図 2-1 ドメインとソフトウェアの関係

【ハードウェア制約】

プロセッサが特権、非特権といった複数のプログラムの実行モードを持たない場合、プログラムはシステムソフトウェアのみとなるため、ドメインはシステムドメインのみとなる。

プロセッサが特定のメモリ領域を保護する機能を持たない場合、アプリケーションドメインは一つの安全ドメインのみとなる。

2.2.4　ドメイン間の保護

ドメインのソフトウェアは、他のドメインのソフトウェアから、空間的および時間的に保護される。

ドメインの空間的保護とは、ドメイン間のメモリ保護である（詳細は「3.6.9 ドメインの空間的保護」を参照）。

ドメインの時間的保護とは、ソフトウェアに割り当てられる実行時間の保護である（詳細は「3.6.10 ドメインの時間的保護」を参照）。

また、あるドメインのソフトウェアは、他のドメインのソフトウェアからの操作についても保護される（詳細は「2.6.4 カーネルオブジェクトのドメイン間の保護」を参照）。

ドメイン間の保護により、システムソフトウェアはアプリケーションから保護される。また、通常ソフトウェアから、安全ソフトウェアは保護される。

2.3 メモリと保護レベル

2.3.1 メモリ

TRON Safe Kernel が操作対象とするメモリは、プロセッサのアドレス空間に物理的に割り当てられたメモリである。二次記憶装置などを用いた仮想記憶管理には対応しない。

アドレス空間はシステムとして一つである。多重メモリアドレス空間には対応しない。また、メモリのアドレスは静的に決定され、動的なアドレス変更は行わない。

メモリは複数のメモリ領域に分割される。メモリ領域のアドレスおよびサイズは静的に決定される。メモリ領域毎にキャッシュやメモリアクセスの属性が指定できる。メモリ領域が取りうるアドレス、サイズ、属性はハードウェアに依存するため、実装定義とする。

メモリ領域は特定のドメインに属する。これは静的に決定される。また、一つのドメインに複数のメモリ領域が属することが可能である。各ドメインのメモリ領域は、ドメインの空間的保護により他ドメインのソフトウェアによるアクセスから保護される。

2.3.2 保護レベル

保護レベルは、プロセッサによるソフトウェアの実行モードと、それに伴うメモリのアクセス保護を、一般的に表現したものである。保護レベルは 0～3 の値を取り、値が小さいほど特権的である。

保護レベルには、プロセッサのソフトウェアの実行モードが割り当てられる。保護レベルとプロセッサの実行モードの対応は実装定義である。プロセッサは必ずしも 4 段階の実行モードを持つわけではないため、異なる保護レベルに同一の動作モードを割り当てることは許される。ただし、MMU などのメモリ保護機能や周辺デバイスへのアクセスは、保護レベル 0 と 1 でのみ許され、保護レベル 2 および 3 では禁止されなくてはならない。

保護レベルは、ドメイン毎に定められ、ソフトウェアおよびメモリは所属するドメインに応じた保護レベルを持つ。システムドメインは保護レベル 0、安全ドメインは保護レベル 2、通常ドメインは保護レベル 3 である。保護レベル 1 は将来の拡張のための予約とする。

【補足説明】

T-Kernel の仕様では、ソフトウェアは自身の保護レベルと同じか低い保護レベルのメモリにのみアクセスできると規定している。TRON Safe Kernel では、T-Kernel 仕様と矛盾することがないように、ドメインの種別毎に保護レベルを定めた。ただし、TRON Safe Kernel ではドメイン間のメモリアクセスを禁じているので、保護レベルが同一であってもドメインが異なればメモリアクセスはできない。TRON Safe Kernel における保護レベルとは、プロセッサ（ハードウェア）の実行モードに伴うメモリ保護を一般化したものにすぎない。

T-Kernel の仕様では、システムコールを呼び出すことができる保護レベルを設定・変更が可能であるが、TRON Safe Kernel では呼び出せるシステムコールはドメイン毎に決められ変更はできない。

【ハードウェア制約】

多くのプロセッサは特権モード/ユーザモードのような 2 段階の実行モードを持つ。この場合は、保護レベル 0、1 は特権モード、保護レベル 2、3 はユーザモードとする。

2 段階の実行モードしか持たないプロセッサの保護レベルとドメイン、実行モードの関係を「表 2-1 保護レベルとドメイン」に示す。

プロセッサが複数の実行モードを持たない場合は、全ての保護レベルは同一の実行モードとなる。この場合は、TRON Safe

Kernel はシステムソフトウェアのみとなる。つまり、保護レベル 0 のプログラムしか存在しない。

表 2-1 保護レベルとドメイン

保護レベル	ドメイン	プロセッサの実行モード
0	システムドメイン	特権モード
1	予約	特権モード
2	安全ドメイン	ユーザ（非特権）モード
3	通常ドメイン	ユーザ（非特権）モード

2.4 API

2.4.1 API の概要

システムソフトウェアの機能をアプリケーションから使うためのインタフェースを API（Application Programming Interface）と呼ぶ。

TRON Safe Kernel の API には、安全ドメインから使用できる安全 API と、通常ドメインから使用できる通常 API がある。

安全ドメインのプログラムが通常ドメインのカーネルオブジェクトを操作する場合は、安全 API を使う。

特別な安全 API として初期登録 API がある。初期登録 API は、TRON Safe Kernel の初期状態で組み込まれるソフトウェアの登録を行うために使用される。

通常ドメインでは T-Kernel2.0 互換 API を有する T-Kernel2.0 ライブラリを使用することができる。

「図 2-2 API とドメインの関係」に各 API とドメインの関係を示す。

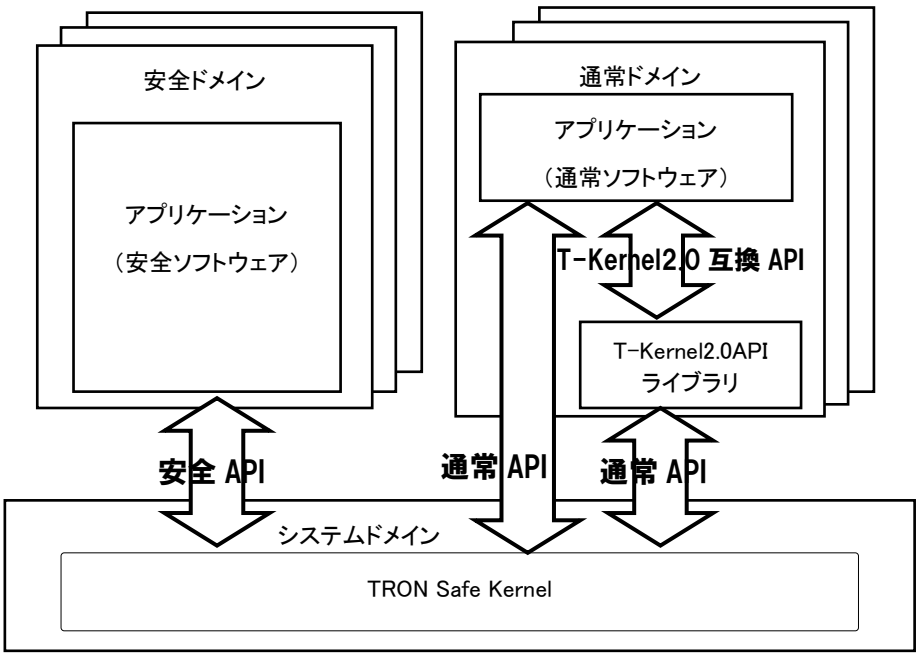

図 2-2 API とドメインの関係

2.4.2 安全 API

安全 API は安全ソフトウェア（システムドメインおよび安全ドメインに属するソフトウェア）から使用できる TRON Safe Kernel の API である。

安全 API は、名称が"ts_"で始まる C 言語の関数として定義される。実行されると速やかに TRON Safe Kernel を呼び出す。

安全 API の処理は TRON Safe Kernel で実行される。

安全 API では、「タスク管理機能」、「システム状態管理機能」、「同期・通信機能」、「時間管理機能」、「システム構成情報管理機能」、「デバイス管理機能」、「サブシステム管理機能」に加え、通常 API で使用不可である、「ドメイン管理機能」（一部）、「割込み管理機能」、「故障診断機能」、「異常例外機能」がサポートされる。詳細は、「8 API 仕様」参照。

2.4.3 通常 API

通常 API は通常ドメインのソフトウェアからのみ使用できる TRON Safe Kernel の API である。

通常 API は、名称が"tn_"で始まる C 言語の関数として定義される。実行されると速やかに TRON Safe Kernel を呼び出す。

通常 API の処理は TRON Safe Kernel で実行される。

通常 API は機能の観点からは、安全 API に対して通常ドメインのソフトウェアで使用するための制約をかけたものである。つまり、通常 API は安全 API の機能的なサブセット（一部処理を除く。詳細は「8.API 仕様」参照）である。

通常 API では、「タスク管理機能」、「システム状態管理機能」、「同期・通信機能」、「時間管理機能」、「システム構成情報管理機能」、「デバイス管理機能」、「サブシステム管理機能」、「ドメイン管理機能」（一部）がサポートされる。詳細は、「8 API 仕様」参照。

2.4.4 初期登録 API

初期登録 API は TRON Safe Kernel の初期化処理にて使用される TRON Safe Kernel の API である。

初期登録 API は、名称が"TS_"で始まる C 言語の関数として定義される。初期登録 API はコンフィギュレーション定義ファイルの定められた個所にのみ記述可能であり、TRON Safe Kernel の初期化処理でのみ使用される。初期登録 API の処理は TRON Safe Kernel で実行される。ユーザが記述したプログラム中から使用することはできない。(詳細は「7 TRON Safe Kernel のコンフィギュレーション」参照)。

2.4.5 T-Kernel2.0 互換 API

T-Kernel2.0 互換 API は、従来の T-Kernel2.0 のアプリケーションを TRON Safe Kernel に容易に移植できるようにすることを目的として提供する関数ライブラリである。

T-Kernel2.0 互換 API は、T-Kernel2.0 仕様の API 仕様に準拠する。ただし、機能安全の観点から提供が困難な機能については除外される。

T-Kernel2.0 互換 API は、T-Kernel2.0API ライブラリとして実装される。T-Kernel2.0API ライブラリの各ライブラリ関数は、C 言語の関数であり、通常ソフトウェアのプログラムの一部として実行される。よって、T-Kernel2.0API ライブラリは TRON Safe Kernel 自体のソフトウェアには含まれず、通常ソフトウェアの位置づけである。ただし、ライブラリ関数の処理の中から通常 API が呼ばれる場合はある。

【補足説明】

TRON Safe Kernel 仕様では、T-Kernel2.0API ライブラリおよび T-Kernel2.0 互換 API については、その実装有無及び実現方法を実装定義とする。

2.5 ソフトウェアの実行

2.5.1 実行単位の概要

TRON Safe Kernel が実行を制御するプログラムの単位を実行単位と呼ぶ。一つの実行単位の中のプログラムは逐次的に実行される。

アプリケーションを構成するプログラムの実行単位は、タスクとタイムイベントハンドラである。

ユーザ定義システムソフトウェアを構成するプログラムの実行単位は、タスク、タイムイベントハンドラ、割込みハンドラ、故障診断ハンドラ、異常例外ハンドラ、拡張 SVC ハンドラである。

各実行単位の説明を示す。

(1) タスク

タスクは、定常的に実行されるプログラムの単位である。TRON Safe Kernel は複数のタスクを並行して実行する。

ただし、並行して実行されるというのは、ソフトウェアから見た概念的な動作であり、実際には TRON Safe Kernel のタスクのスケジューリングの制御に従って実行される。

タスクはカーネルオブジェクトであり、システムドメイン、安全ドメイン、通常ドメインのすべてのドメインで生成、実行できる。

(2) タイムイベントハンドラ

タイムイベントハンドラは時間の経過に従って起動されるプログラムの単位である。

タイムイベントハンドラは周期ハンドラとアラームハンドラの二種類がある。周期ハンドラは、時間の経過に従って一定周期で起動されるタイムイベントハンドラである。アラームハンドラは、特定の時刻に一回のみ起動されるタイムイベントハンドラである。

タイムイベントハンドラはカーネルオブジェクトであり、システムドメイン、安全ドメイン、通常ドメインのすべてのドメインで生成、実行できる。

(3) 割込みハンドラ

割込みハンドラは、割込みが発生した際に実行される実行単位である。

割込みハンドラは、ユーザ定義システムソフトウェアとして、システムドメインで実行される。

(4) 故障診断ハンドラ

故障診断ハンドラは、ハードウェアの故障検出のために定期的または定められたときに実行される実行単位である。

故障診断ハンドラは、ユーザ定義システムソフトウェアとして、システムドメインで実行される。

(5) 異常例外ハンドラ

異常例外ハンドラは、異常例外が発生した際に実行される実行単位である。

デフォルトの異常例外ハンドラは、TRON Safe Kernel の一部として静的に登録される。

異常例外ハンドラの実行は、カーネルが割込み禁止中あるいはディスパッチ禁止中であっても抑止されない。

(6) 拡張 SVC ハンドラ

拡張 SVC は、ユーザ定義システムソフトウェアの構成要素の一つであるサブシステムの機能を呼び出す。

拡張 SVC ハンドラは、サブシステムの拡張 SVC が他のプログラムから呼び出された際に実行される実行単位である。拡張 SVC ハンドラは、サブシステムを構成するプログラムの一つであり、一つのサブシステムに一つの拡張 SVC ハンドラが存在する。

2.5.2 実行優先度

ソフトウェアの実行単位には、それぞれ実行優先度が割り当てられる。

TRON Safe Kernel は、実行優先度に従って各実行単位の実行順番を決定し、ソフトウェアを実行する。これをスケジューリングと呼ぶ。

実行優先度は、タスク優先度とシステム優先度に大別され、タスク優先度よりシステム優先度の方が優先される。

(1) タスク優先度

タスク優先度は、タスク、タイムイベントハンドラ、故障診断ハンドラに割り当てられる優先度である。

タスク優先度は 1 から始まる整数である。値が小さい方が高い優先度となる。つまり優先度 1 のタスクは優先度 2 のタスクより優先度が高い。

タスク優先度の最大値(優先度低)はコンフィギュレーション(TS_LST_PRI)にて 16 から 250 の間の値が指定できる。また、ドメイン毎に、そのドメインで指定可能なタスク優先度の範囲が定められる。

タイムイベントハンドラの優先度は、所属するドメインにおいて最も高い優先度となる。

(2) システム優先度

システム優先度は、TRON Safe Kernel 自体および、割込みハンドラ、異常例外ハンドラに割り当てられる優先度である。

システム優先度は-1 以下の負の値である。値が小さい方が高い優先度となる。

システム優先度の最小値(優先度高)はコンフィギュレーション(TS_HST_SPRI)にて-250 から-1 の間の値が指定できる。

システム優先度の割り当ては実装定義であるが、以下の優先度順とする。

異常例外ハンドラ（優先度高） ＞ TRON Safe Kernel ＞ 割込みハンドラ（優先度低）
ただし、割込みハンドラは TRON Safe Kernel より高い優先度とすることはできる。その場合、割込みハンドラから使用できる API に制約が生じる場合がある。制約は実装定義である。

2.5.3 連続実行時間

ソフトウェアの実行単位には、それぞれ連続実行時間が割り当てられる。TRON Safe Kernel は、実行単位が定められた連続実行時間を超えて実行を続けた場合、異常例外として処理を行う。
各実行単位について連続実行時間は以下のように定められる。

(1) タスク

タスクの連続実行時間は、タスクが実行状態となってから、それ以外の状態になるまでの時間である。他のより優先度の高いタスクに実行が切り替わった場合、連続実行時間の計測は終了する。
タスクが連続実行時間を超えた場合、その時点で TRON Safe Kernel は異常例外を発生する。よって、対象のタスクの実行は中断され、異常例外ハンドラが実行される。
タスクの実行中に、タスク以外の実行単位の処理が割り込んで実行された場合は、その処理の実行時間は除外される。割込みハンドラやタイムイベントハンドラ、故障診断ハンドラの実行がこれに該当する。この場合は、連続実行時間は保持されるので、元のタスクに実行が戻ると、継続して連続実行時間が計測される。

(2) タイムイベントハンドラ

タイムイベントハンドラの連続実行時間は、タイムイベントハンドラが実行されてから終了するまでの時間である。
タイムイベントハンドラが連続実行時間を超えた場合、その時点で TRON Safe Kernel は異常例外を発生する。よって、対象のタイムイベントハンドラの実行は中断され、異常例外ハンドラが実行される。
タイムイベントハンドラの実行中に、他の実行単位の処理が割り込んで実行された場合は、その処理の実行時間は除外される。割込みハンドラや、他のドメインのタイムイベントハンドラ、タスクなどの実行がこれに該当する。この場合は、連続実行時間は保持されるので、元のタイムイベントハンドラに実行が戻ると、継続して連続実行時間が計測される。

(3) 拡張 SVC ハンドラ

拡張 SVC ハンドラは、タスクまたはタイムイベントハンドラから呼び出され、そのコンテキスト上で動作する。よって、拡張 SVC ハンドラ自体は固有の連続実行時間をもたず、拡張 SVC ハンドラの実行時間は、呼び出したタスクまたはタイムイベントハンドラの実行時間に加算される。

(4) 割込みハンドラ

割込みハンドラの連続実行時間は、割込みハンドラが実行されてから終了するまでの時間である。
割込みハンドラが連続実行時間を超えた場合、TRON Safe Kernel は異常例外を発生する。時間の検証は、その割込みハンドラの実行が終了した時点で行われる。よって、割込みハンドラは連続実行時間を超えても、タスクのようにその時点で実行が中断されない。割込みハンドラの実行が何らかの不具合により長時間終了しない異常は、故障診断機能で検出する。
多重割込みにより、割込みハンドラ実行中に他の割込みハンドラが実行された場合、その処理時間は除外される。

(5) 故障診断ハンドラ

故障診断ハンドラが連続実行時間を超えた場合、その時点で TRON Safe Kernel は異常例外を発生する。よって、故障診断ハンドラの実行は中断され、異常例外ハンドラが実行される。

故障診断ハンドラの実行中に、他の実行単位の処理が割り込んで実行された場合は、その処理の実行時間は除外される。割込みハンドラや他の故障診断ハンドラなどの実行がこれに該当する。この場合は、連続実行時間は保持されるので、元の故障診断ハンドラに実行が戻ると、継続して連続実行時間が計測される。

故障診断ハンドラは実行中に割込みをマスクすることがある。割込みがマスクされている期間は、連続実行時間を超えてもその時点で検出できない場合がある。その場合は割込みマスクが解除された時点で異常例外が発生する。また、割込みマスク時間の異常については故障診断機能で検出する。

全ての実行単位に該当することとして、システムコールの実行中に異常例外が発生することはない。システムコールの実行中に連続実行時間を超えた場合は、システムコールの処理が終了した時点で異常例外が発生する。

拡張 SVC の実行中に連続実行時間の上限を超えた場合は、異常例外が発生する。

2.6 カーネルオブジェクト

2.6.1 カーネルオブジェクトの概要

TRON Safe Kernel 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と呼ぶ。

具体的にはカーネルオブジェクトは、タスクやタイムイベントハンドラなどのプログラムの実行単位や、それらの間の同期・通信を行うためのミューテックスやメッセージバッファなどである。

「表 2-2 カーネルオブジェクト一覧」にカーネルオブジェクトの一覧を示す。

表 2-2 カーネルオブジェクト一覧

分類	種類	備考	参照先
タスク	タスク	実行単位（プログラム）	3.1 章
タイムイベントハンドラ	アラームハンドラ	実行単位（プログラム）	3.4 章
	周期ハンドラ	実行単位（プログラム）	
同期・通信	イベントフラグ	—	3.3 章
	セマフォ	—	
	ミューテックス	—	
	メッセージバッファ	—	

2.6.2 カーネルオブジェクトの生成と削除

カーネルオブジェクトは、API を呼び出すことにより動的に生成、削除される。カーネルオブジェクトは、API を呼び出したソフトウェアが属しているドメインに属する。

例外として、各ドメインに一つ存在する初期タスクがある。初期タスクは各ドメインにおいて最初に実行される特別なタスクであり、システムの構築時に属するドメインが決定される。初期タスクの生成、実行は、カーネルにて実行されるので明示的に API を呼ぶ必要はない。

カーネルオブジェクトが必要とするメモリなどの資源はシステムの構築時にすべて静的に確保される。

よって、カーネルオブジェクトの生成時の処理はデータの初期化のみである。

【補足説明】

あるドメインに属するカーネルオブジェクトは、そのドメインの初期タスクから直接的または間接的に生成されたものとなる。

2.6.3 オブジェクト ID とオブジェクト名

カーネルオブジェクトには生成時に固有の ID 番号が割り当てられる。ID 番号は TRON Safe Kernel においてカーネルオブジェクトを識別するための情報であり、そのカーネルオブジェクトに対する API の操作は、ID 番号を指定して行う。
タスクの ID 番号はタスク ID と呼ぶ。同様にセマフォの ID 番号はセマフォ ID、と言うように、カーネルオブジェクトの種別に合わせて呼ぶ。また ID 番号を総称してオブジェクト ID と呼ぶ。
オブジェクト ID は、オブジェクトの種類毎に、全ドメインを通して固有の値(ユニークな値)を持つ。すなわち、同種のカーネルオブジェクトが複数存在する場合、それらのカーネルオブジェクトの所属ドメインが同じか異なるかに関わらず、それらのオブジェクト ID はすべて異なった値を持つ。
削除されたカーネルオブジェクトの ID 番号は、その後に生成されるカーネルオブジェクトに割り当てられる可能性がある。

オブジェクト名は、カーネルオブジェクトの生成時に指定できる固有の名称である。ただし、オブジェクト名は生成時に指定しなくてもよい。その場合、そのカーネルオブジェクトはオブジェクト名が無いことになる。
オブジェクト名は最大 8 文字の 1 バイトコードであり、任意の文字列が指定できる。ただし、同一ドメインで同一の種類のカーネルオブジェクトは、異なったオブジェクト名でなくてはならない。例えば、同一ドメインのタスクは全て異なったオブジェクト名でなくてはならない。

他のドメインのカーネルオブジェクトの ID 番号を参照する場合は、安全 API を用いて行う。この場合、オブジェクト名を生成時に指定しておく必要がある。

2.6.4 カーネルオブジェクトのドメイン間の保護

カーネルオブジェクトは特定のドメインに属している。カーネルオブジェクトの生成、削除ができるのは同一のドメインに属するソフトウェアからのみである。
また、他のドメインのカーネルオブジェクトを操作する場合、以下の制約がある。

(1) 通常ソフトウェアは他のドメインのカーネルオブジェクトを操作できない。つまり、通常 API は自ドメインのカーネルオブジェクトのみを操作できる。
(2) 安全ソフトウェアは他のアプリケーションドメインのカーネルオブジェクトを操作できる。ただし、他のドメインのカーネルオブジェクトに対し、待ち状態となる操作はできない。他のドメインのカーネルオブジェクトに対してはポーリングのみが可能である。
(3) アプリケーションからシステムドメインのカーネルオブジェクトを操作できない。

2.6.5 コントロールブロック

TRON Safe Kernel は、それぞれのカーネルオブジェクトに対して、それを管理するためのデータを持つ。このデータをそのカーネルオブジェクトのコントロールブロックと呼ぶ。カーネルオブジェクトの種類によっては、コントロールブロック以外に実行プログラムやメモリ領域を持つものも存在する。
コントロールブロックは TRON Safe Kernel の内部データであり、外部から参照や操作はできない。

【補足説明】
コントロールブロックは、TRON Safe Kernel の内部データであるため、ユーザがその存在を意識する必要はない。ただし、機能安全における TRON Safe Kernel の故障診断などにおいて、コントロールブロックの説明が必要であるため、ここに記した。

2.7 システムソフトウェア

2.7.1　システムソフトウェアの構成

システムソフトウェアは、TRON Safe Kernel 自体と、ユーザ定義システムソフトウェアから構成される。

TRON Safe Kernel は、さらに TRON Safe Kernel/OS、TRON Safe Kernel/SM 、TRON Safe Kernel/Core に分かれる。

(1) TRON Safe Kernel/OS

TRON Safe Kernel/OS は、主にカーネルオブジェクトの管理を行う。

TRON Safe Kernel/OS の主な機能は、タスク管理、システム状態管理、同期通信管理、時間管理、システム構成情報管理、ドメイン管理である。

アプリケーションは API を用いて TRON Safe Kernel/OS の機能を使用することができる。

(2)TRON Safe Kernel/SM

TRON Safe Kernel/SM は、主にユーザ定義システムソフトウェアの管理を行う。

TRON Safe Kernel/SM の主な機能は、デバイス管理、割込み管理、サブシステム管理、故障検出管理、異常例外処理である。

アプリケーションは API を用いて TRON Safe Kernel/SM を介し、ユーザ定義システムソフトウェアの機能を使用することができる。

(3) TRON Safe Kernel/Core

TRON Safe Kernel/Core は、TRON Safe Kernel の中でハードウェアに近い低水準の処理を行う。

TRON Safe Kernel/Core の主な機能は、起動処理、低水準メモリ管理、低水準割込み管理、低水準時間管理、コンテキスト管理である。

TRON Safe Kernel/Core は TRON Safe Kernel の中でハードウェアに依存する処理を集約している。TRON Safe Kernel を特定のプロセッサに実装する場合、主に TRON Safe Kernel/Core を対象とするプロセッサ向けに記述する。

アプリケーションから TRON Safe Kernel/Core の機能を直接使用することはできない。

「表 2-3 TRON Safe Kernel の構成と機能」に TRON Safe Kernel の構成と機能を示す。

表 2-3 TRON Safe Kernel の構成と機能

構成	機能	説明
TRON Safe Kernel /OS	タスク管理	タスクの管理
	システム状態管理	タスクのスケジューリングなどシステム状態管理
	同期通信管理	同期通信機能の管理
	時間管理	システム時刻、タイムイベントハンドラの管理
	システム構成情報管理	システム構成情報の管理
	ドメイン管理	ドメインの管理
TRON Safe Kernel /SM	デバイス管理	デバイスドライバの管理
	割込み管理	割込みハンドラの管理
	サブシステム管理	サブシステムの管理
	故障診断管理	故障診断ハンドラの管理
	異常例外処理	異常例外の処理
TRON Safe Kernel /Core	起動処理	システムソフトウェアの起動、アプリケーションの実行
	低水準メモリ管理	メモリ、キャッシュの設定などの基本的な制御
	低水準割込み管理	ハードウェアからの割込みの受付、制御
	低水準時間管理	システムタイマを用いたスケジューリングの処理
	コンテキスト管理	ディスパッチ処理（タスク、ハンドラの切り替え処理）

2.7.2 ユーザ定義システムソフトウェア

ユーザ定義システムソフトウェアは、以下のソフトウェアから構成される。

(1) サブシステム

サブシステムは、システムソフトウェアの機能を拡張するためのプログラムである。サブシステムの機能を呼び出すインタフェースを拡張 SVC という。

サブシステムは、任意の数のタスク、タイムイベントハンドラと一つの拡張 SVC ハンドラから構成される。また、TRON Safe Kernel とのインタフェースとしてサブシステム I/F 関数を持つ。

(2) デバイスドライバ

デバイスドライバは、周辺デバイスを制御するためのプログラムである。

デバイスドライバは、任意の数のタスク、タイムイベントハンドラから構成される。また TRON Safe Kernel とのインタフェースとしてデバイス I/F 関数を持つ。

(3) 割込みハンドラ

割込みハンドラは、割込みが発生した際に実行されるプログラムの実行単位である。

割込みハンドラは通常、特定のサブシステムやデバイスドライバと密な関係を持ち、機能的にはそのサブシステムやデバイスドライバの一部とみなすことができる。

(4) 故障診断ハンドラ

故障診断ハンドラは、ハードウェアの故障検出のために定期的または定められたときに実行されるプログラムの実行単位である。

(5) 異常例外ハンドラ

異常例外ハンドラは、異常例外が発生した際に実行されるプログラムの実行単位である。

デフォルトの異常例外ハンドラは、TRON Safe Kernel の一部として静的に登録される。

異常例外ハンドラの実行は、カーネルが割込み禁止中あるいはディスパッチ禁止中であっても抑止されない。

【補足説明】

TRON Safe Kernel としては、標準的なサブシステム、デバイスドライバ、割込みハンドラなどは規定しない。ただし、特定のシステムにおいてサブシステム、デバイスドライバ、割込みハンドラなどを規定することはありうる。これは実装定義とする。

2.7.3 TRON Safe Kernel の状態

TRON Safe Kernel の状態は以下に大別される。

(1) カーネル起動状態(カーネル停止状態)

TRON Safe Kernel が電源投入やシステムリセットなどにより起動された状態である。この状態では TRON Safe Kernel の起動処理プログラムによって規定の起動処理を行い、処理終了後にカーネル初期化状態に遷移する。

本状態では、TRON Safe Kernel は動作しておらず、一切の機能が使用できない。よってカーネル停止状態とも呼ぶ。

(2) カーネル初期化状態

TRON Safe Kernel が起動され、コンフィギュレーション情報に基づいて初期化処理を行っている状態である。

初期化処理の終了後、システムドメインおよび特定のドメインの実行を開始し、カーネル実行状態に遷移する。実行を開始する安全ドメインは静的に定められる。

本状態では、異常例外処理のみが機能する。異常例外が検出されると異常例外状態に遷移する。

(3) カーネル実行状態

カーネルが実行されている通常の状態である。TRON Safe Kernel の全ての機能が使用可能となる。

(4) カーネルエラー状態

異常が検出され異常例外が発生し、それに対応する処理が実行されている状態である。

「図 2-3 TRON Safe Kernel の状態遷移(概略)」に上記の TRON Safe Kernel の状態遷移の概略を示す。

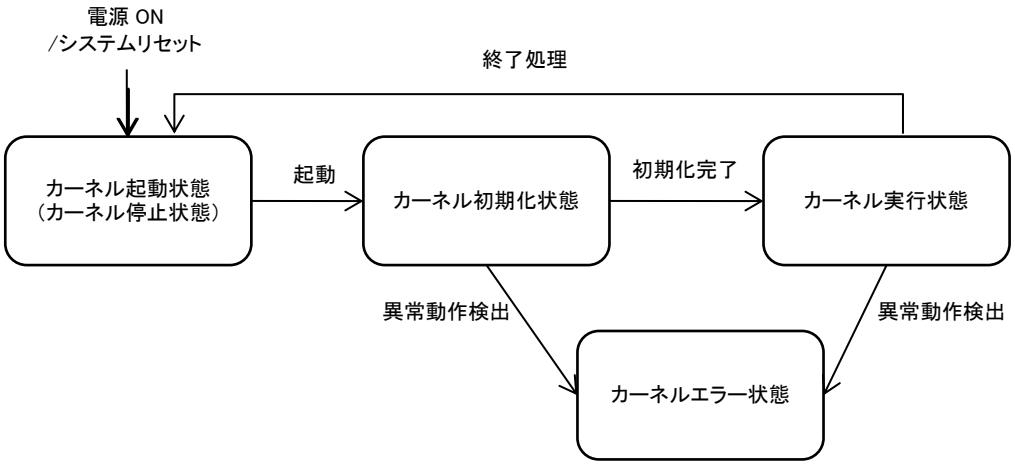

図 2-3 TRON Safe Kernel の状態遷移(概略)

(3)のカーネル実行状態と、(4)のカーネルエラー状態は、サブ状態を持つ。

(3)のカーネル実行状態は、そのとき実行されている実行単位によって、タスク部実行状態、タイムイベントハンドラ実行状態、準タスク部実行状態、タスク独立部実行状態、カーネル部実行状態のシステム状態に分かれる。

タスク独立部実行状態は、さらに割込み実行状態、故障診断実行状態に分かれる。

「表 2-4 システム状態と実行単位」にシステム状態と実行中の実行単位を示す。

表 2-4 システム状態と実行単位

システム状態分類	システム状態	実行している実行単位
タスク部実行状態	タスク部実行状態	タスク
準タスク部実行状態	準タスク部実行状態	拡張 SVC ハンドラ
タイムイベントハンドラ実行状態	タイムイベントハンドラ実行状態	タイムイベントハンドラ
タスク独立部実行状態	割込み実行状態	割込みハンドラ
	故障診断実行状態	故障診断ハンドラ
カーネル部実行状態	カーネル部実行状態	カーネル(システムコールなど)

(3-1) タスク部実行状態

タスクを実行している状態である。実行中のタスクが「自タスク」として認識される。タスクの待ち状態を伴う API を使用できる。ディスパッチを禁止していないかぎり、常にディスパッチは可能である。

(3-2) 準タスク部実行状態

タスクから呼び出された拡張 SVC ハンドラを実行している状態である。拡張 SVC ハンドラは、呼び出した実行単位のコンテキストで実行される。よって、タスクが拡張 SVC を呼び出した場合は、そのタスクが「自タスク」として認識され、基本的にそのタスクの実行状態と同等の動作を行う。よって、ディスパッチも発生し、また待ちを伴う API も使用できる。

(3-3) タイムイベントハンドラ実行状態

タイムイベントハンドラを実行している状態である。実行中のタスクは存在しないので、API で自タスクを指定することはできない。また、タスクの待ち状態を伴う API を使用できない。常にディスパッチは可能である。

(3-4) タスク独立部実行状態

割込みハンドラ、故障診断ハンドラのいずれかを実行している状態である。実行中のタスクは存在しないので、API で自タスクを指定することはできない。また、タスクの待ち状態を伴う API を使用できない。

基本的にこの状態ではタスクのディスパッチが発生しない。ディスパッチは本状態が終了しディスパッチ可能な状態に遷移するまで保留される。これを遅延ディスパッチという。

(3-5) カーネル部実行状態

TRON Safe Kernel のカーネル本体を実行している状態である。システムコールの実行や、低水準の割込み処理などがこれに該当する。他のプログラムから見て、カーネル部実行状態は不可分に実行され、この状態を認識することはできない。

(4)のカーネルエラー状態は、異常例外状態と安全状態に分かれる。

(4-1) 異常例外状態

異常例外が発生し異常例外ハンドラが実行されている状態である。本状態では基本的に限られた API のみが使用できる。

本状態は、カーネル初期化状態およびカーネル実行状態において、異常例外が発生した場合に遷移する。異常例外ハンドラにおいて必要な異常例外に対応する処理を実行したのち、安全状態へと遷移する。ただし、異常例外ハンドラの処理において、システムの実行上問題がないと判断した場合は、元の状態に遷移することができる。

(4-2) 安全状態

TRON Safe Kernel 本体を含む全てのプログラムが停止した状態である。すべての割込みも禁止され、ソフトウェアとしては何もできない状態である。異常例外状態からのみ遷移される。

「図 2-4 TRON Safe Kernel のカーネル状態遷移」に上記のサブ状態を含めた TRON Safe Kernel のカーネル状態の状態遷移を示す。

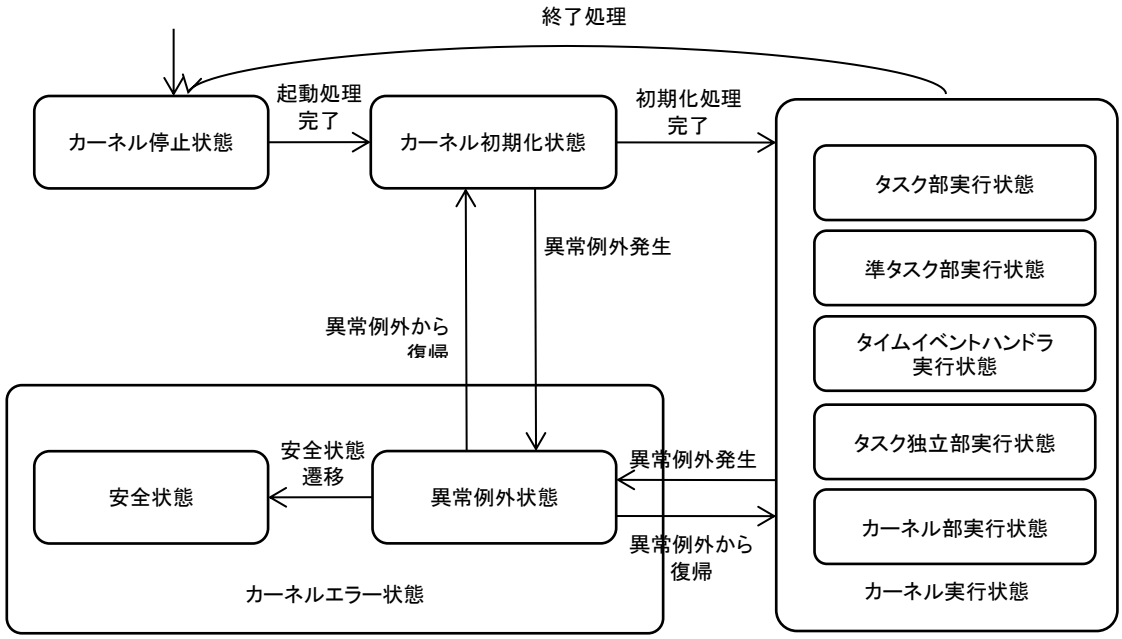

図 2-4 TRON Safe Kernel のカーネル状態遷移

3. TRON Safe Kernel/OS の機能

3.1 タスク管理機能

3.1.1 タスク管理機能の概要

タスクは、定常的に実行されるプログラムの実行単位であり、カーネルオブジェクトである。

タスク管理機能は、タスクに対する各種の操作と、タスク付属同期の制御を行う。タスク付属同期は、タスクの状態を直接的に操作することによって同期を行う。

各タスクは「表 3-1 タスク管理情報」に示す情報を持つ。各情報は、API によって参照できる。また、一部の情報は API により変更できる。

表 3-1 タスク管理情報

名称	意味	動的変更※1
タスク ID	タスクを識別する ID 番号	不可
所属ドメイン	タスクが所属するドメイン	不可
保護レベル	タスクの保護レベル	不可
タスク属性	タスクの属性	不可
タスク状態	タスクの状態	可※2
タスク起動アドレス	タスクの実行プログラムの起動アドレス	不可
最大連続実行時間	タスクの連続実行できる時間の上限	不可
ユーザスタック情報	タスクが通常使用するスタックの情報	不可
システムスタック情報	タスク実行中にシステムが使用するスタックの情報	不可
起動時優先度	起動時に設定されるタスクの優先度	不可
ベース優先度	タスクの現在優先度を決定するために使用する優先度	可
現在優先度	タスクの現在の優先度	可※2
起床要求数	タスク起床要求のキューイング数	可※2
強制待ち要求数	強制待ちの要求ネスト数	可※2
待ち禁止要因	待ちを禁止されている要因	可※2
オブジェクト名	オブジェクトを識別するための名称(文字列)	不可

※1 生成後に API により動的な変更ができるか否か

※2 API の結果として変更されるが、直接 API から値を指定することはできない

タスク ID は、タスク生成時に TRON Safe Kernel により自動的に割り当てられる。

所属ドメインは、タスクが所属するドメインであり、そのタスクを生成したプログラムが所属するドメインとなる。

保護レベルは、タスクの所属するドメインで定められた保護レベルが設定される。

タスク属性は、タスク生成時に指定されたタスク属性値である。詳細は「3.1.2 タスクの属性」を参照。

タスク状態は現在のタスクの状態である。詳細は「3.1.3 タスクの状態」を参照。

タスク起動アドレスは、タスクの起動時に実行されるプログラムの起動アドレスである。詳細は「3.1.4 タスクのメイン関数」を参照。

最大連続実行時間は、タスクが連続実行可能な時間の上限を設定する。

ユーザスタック情報は、タスクのユーザスタックの情報であり、システムスタック情報は、タスクのシステムスタックの情報であ

る。詳細は「3.1.5 タスクのスタック」を参照。

起動時優先度、ベース優先度、現在優先度はタスクのスケジューリングにおける優先度である。詳細は「3.1.6 タスクの優先度」を参照。

起床要求数は、タスクが現在キューイングしている起床要求の数である。起床要求については「8.2.1.11 ts_wup_tsk/tn_wup_tsk – 他タスクの起床」で述べる。

強制待ち要求数は、タスクに対する強制待ちの要求のネスト数である。強制待ちについては「8.2.1.14 ts_sus_tsk/tn_sus_tsk – 他タスクを強制待ち状態へ移行」で述べる。

待ち禁止要因は、タスクが待ちを禁止されている要因である。待ち禁止については「8.2.1.18 ts_dis_wai/tn_dis_wai – タスク待ち状態の禁止」で述べる。

オブジェクト名は、タスクの生成時に指定されたタスクの名称である。

3.1.2 タスクの属性

タスクの生成時にタスク属性が指定される。「表 3-2 タスク属性」にタスク属性を示す。

表 3-2 タスク属性

名称	属性	意味
TA_ONAME	オブジェクト名称指定	オブジェクト名称を指定
TA_COP*	コプロセッサ使用指定	タスクでコプロセッサを使用 *はコプロセッサを識別する ID 番号(実装依存)
TA_FPU	FPU 使用指定	タスクで浮動小数点演算用コプロセッサを使用

3.1.3 タスクの状態

タスクの状態は以下に分類される。

(1) 実行状態(RUNNING)

タスクが実行されている状態である。または、タスク以外のより優先度の高いプログラムを実行している間は、それ以前に実行していたタスクが実行状態であるものとする。

(2) 実行可能状態(READY)

タスクを実行できるが、そのタスクよりも優先順位の高いタスクが実行中であるために、そのタスクを実行できない状態である。

(3) 広義の待ち状態

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

(3-1) 待ち状態(WAITING)

自タスクによって、何らかの条件が整うまでタスクの実行が中断された状態である。

(3-2) 強制待ち状態(SUSPENDED)

他のタスクによって、強制的に実行を中断させられた状態である。

(3-3) 二重待ち状態(WAITING-SUSPENDED)

待ち状態と強制待ち状態が重なった状態である。待ち状態にあるタスクに対して、強制待ち状態への移行が要求されると、二重待ち状態となる。

(4) 休止状態(DORMANT)

タスクがまだ起動されていないか、実行を終了した後の状態である。

タスクが休止状態にある間は、プログラムカウンタやレジスタなどのプログラムの実行を再開するための情報は保存されていない。タスクを休止状態から起動する時には、タスクの起動番地から実行を開始する。

タスクの起動とは、休止状態のタスクを実行可能状態に移行させることをいう。このことから、休止状態以外の状態を総称して、起動された状態と呼ぶ。

また、実行状態と実行可能状態を総称して、実行できる状態と呼ぶ。

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

タスクの終了とは、起動された状態のタスクを休止状態に移行させることをいう。

タスクの状態遷移を「図 3-1 タスクの状態遷移」に示す。

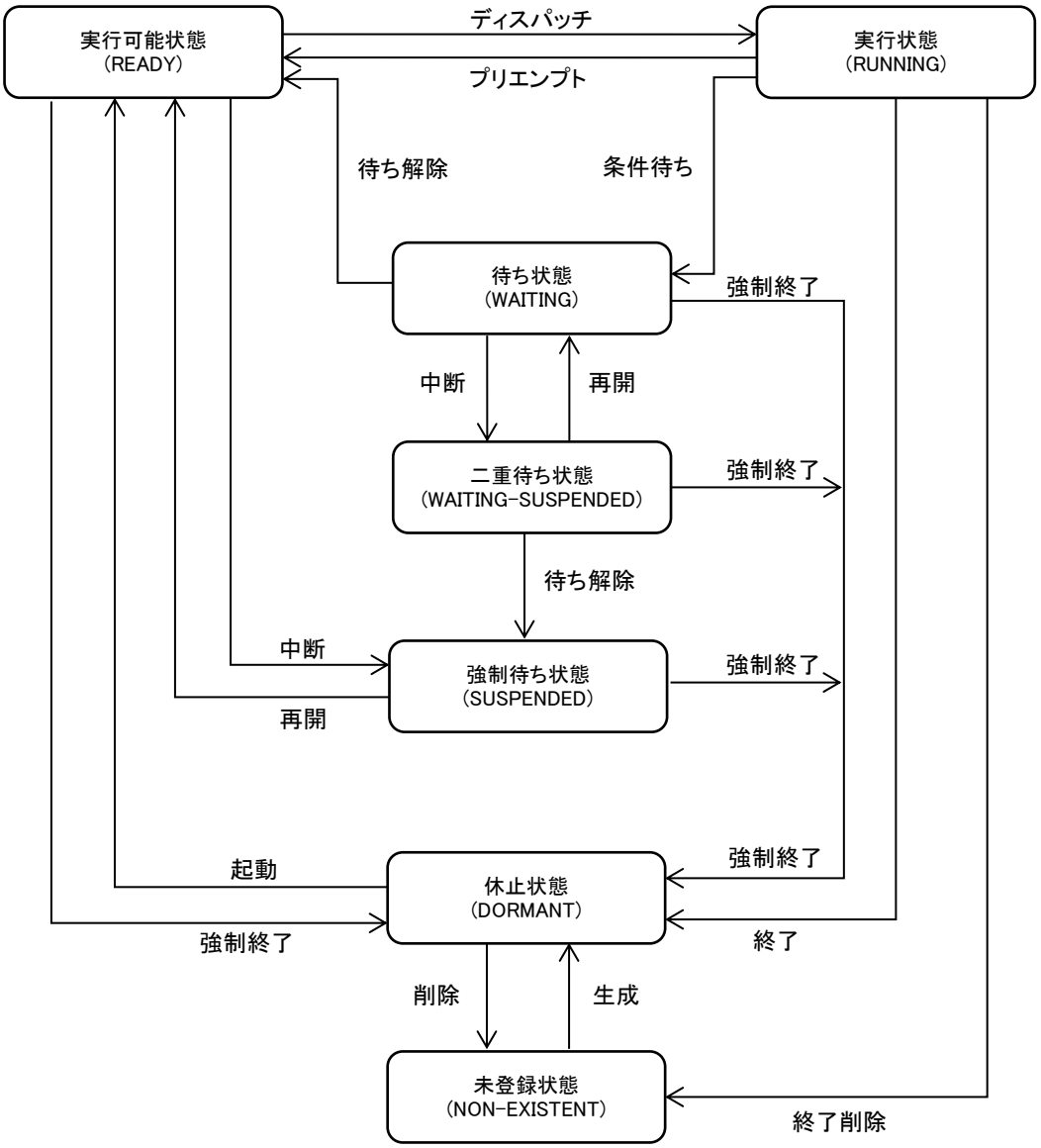

図 3-1 タスクの状態遷移

3.1.4 タスクのメイン関数

タスクの起動時に実行されるプログラムを、そのタスクのメイン関数と呼ぶ。

タスクのメイン関数は以下の形式の C 言語の関数とする。関数名は任意である。

```
void    task_main( INT stacd)
```

関数のパラメータとして、起動時にユーザが指定した起動コード stacd が渡される。

ただし、ドメインの初期タスクには必ず 0 が渡される。

3.1.5 タスクのスタック

それぞれのタスクは、ユーザスタックとシステムスタックの二種類のスタックを一つずつ持つ。

ユーザスタックは、タスクのプログラム実行中に使用されるスタックである。

システムスタックは、タスクのプログラム実行中にAPIを発行した場合などに、その処理の中で使用されるスタックである。システムスタックが使用される条件は実装定義である。

スタックのサイズは静的に決定され、スタックが使用するメモリ領域も静的に確保される。

3.1.6 タスクの優先度

タスクは以下の優先度を持つ。

(1) 起動時優先度

起動時優先度は、タスクの生成の際に指定する。タスクの起動時には、すべての優先度の値は、この起動時優先度の値が設定される。また、タスクが終了したあと、再度起動された場合も、すべての優先度の値は、この起動時優先度の値が設定される。

(2) ベース優先度

タスクの基本となる優先度である。タスクの生成時に、起動時優先度に初期化される。その後、タスクが終了するまで、ベース優先度は API により変更可能である。

ただし、ts_chg_pri/tn_chg_pri 使用時の動作については、「8.2.1.7 ts_chg_pri/tn_chg_pri － タスク優先度変更」を参照。

(3) 現在優先度

タスクの現在の優先度であり、この値をもとにタスクのスケジューリングが行われる。単に「タスクの優先度」と言った場合、この現在優先度を指す。

現在優先度は、通常はベース優先度と一致する。ただし、ミューテックスをロックした場合は、現在優先度はベース優先度と異なる場合がある。ミューテックスをロックした場合の現在優先度については「3.3.3 ミューテックス」を参照。

3.1.7 タスクのスケジューリング規則

TRON Safe Kernel のタスクスケジューリングには、タスクに与えられた優先度に基づくプリエンプティブな優先度ベーススケジューリング方式を採用している。以下に規則に従い、実行可能なタスクの優先順位を定め、最も優先順位の高いタスクを実行する。

- タスクの現在優先度の高いものほど、高い優先順位とする。
- 優先度が同一のタスクは、ドメインの種別により以下の順で優先順位を定める。
  システムドメイン（優先度高） ＞ 安全ドメイン ＞ 通常ドメイン（優先度低）
- 優先度が同一かつドメインの種別も同一のタスクは、ドメインのID番号が小さいものほど優先順位が高くなる（ドメインのID 番号は静的に設定されている）。
- 同一ドメイン内の同一優先度のタスクでは、先に実行できる状態(実行状態または実行可能状態)になったタスクの方が高い優先順位とする。ただし、API の呼び出しによりドメイン内の優先順位は変更される場合がある。

3.1.8 タスク管理機能の API

TRON Safe Kernel のタスク管理機能は、「表 3-3 タスク管理機能 API 一覧」に示す API を提供する（各 API の詳細は、「8.2.1 タスク管理機能の API」参照）。

表 3-3 タスク管理機能 API 一覧

分類	安全 API 名称	通常 API 名称	説明
タスクの生成・削除	ts_cre_tsk	tn_cre_tsk	タスクの生成
	ts_del_tsk	tn_del_tsk	タスクの削除
タスクの実行・終了	ts_sta_tsk	tn_sta_tsk	タスクの起動
	ts_ext_tsk	tn_ext_tsk	自タスク終了
	ts_exd_tsk	tn_exd_tsk	自タスクの終了と削除
	ts_ter_tsk	tn_ter_tsk	他タスクの強制終了
タスクの情報参照・変更	ts_chg_pri	tn_chg_pri	タスク優先度変更
	ts_inf_tsk	tn_inf_tsk	タスク統計情報参照
	ts_ref_tsk	tn_ref_tsk	タスク状態参照
タスク付属同期	ts_slp_tsk	tn_slp_tsk	自タスクを起床待ち状態へ移行
	ts_wup_tsk	tn_wup_tsk	他タスクを起床
	ts_can_wup	tn_can_wup	タスクの起床要求を無効化
	ts_rel_wai	tn_rel_wai	他タスクの待ち状態を解除
	ts_sus_tsk	tn_sus_tsk	他タスクを強制待ち状態へ移行
	ts_rsm_tsk	tn_rsm_tsk	強制待ち状態のタスクを再開
	ts_frsm_tsk	tn_frsm_tsk	強制待ち状態のタスクを強制再開
	ts_dly_tsk	tn_dly_tsk	タスク遅延
	ts_dis_wai	tn_dis_wai	タスク待ち状態の禁止
	ts_ena_wai	tn_ena_wai	タスク待ち禁止の解除

3.2 システム状態管理機能

3.2.1 タスクのスケジューリング操作

タスクは「3.1.7 タスクのスケジューリング規則」で規定したスケジューリング規則に従って実行される。

TRON Safe Kernel において、このスケジューリング規則に従いタスクを実行することを、タスクのスケジューリングと呼ぶ。

タスクのスケジューリングは、以下の API 操作により可能である。なお、API による操作は、API を発行したタスクが所属するドメインに対してのみ有効であり、他のドメインのタスクのスケジューリングには影響しない。

(1) タスクの優先順位の回転

各ドメインの各優先度について、タスクの実行待ち行列(タスクのレディキュー)が存在していると考えることができる。待ち行列の一番前のタスクが、そのドメインのその優先度においてもっとも優先順位が高い。

タスクは実行できる状態(実行状態または実行可能状態)になると、待ち行列の一番後ろ(もっとも優先順位が低い)に入る。よって、待ち行列は基本的には、タスクが実行できる状態になった順番に並んでいる。

APIを用いて自ドメインを指定した優先度のタスクの待ち行列を回転させることができる。具体的には、この実行待ち行列の中で最も高い優先順位のタスクを、最低の優先順位にする。よって、二番目に優先順位の高いタスクが、最も高い優先順位のタスクとなる。

(2) ディスパッチの禁止

自ドメインのタスクのディスパッチを禁止することができる。

API の発行時に、そのドメインにおいて実行状態のタスクが、ディスパッチ禁止が解除されるまで実行状態であり続ける。

(3) 実行状態タスクのタスク ID 参照

自ドメインの中で実行状態にあるタスクの ID 番号を取得することができる。

具体的にはドメインに所属する実行可能なタスクの中で、最も高い優先順位のタスクが実行状態のタスクとみなされる。ただし、ドメインがディスパッチ禁止状態の場合は、優先順位に変化があっても、ディスパッチ禁止になったときに実行状態であったタスクが、そのまま実行状態となる。

3.2.2 システムの情報取得

TRON Safe Kernel のシステム状態、バージョン番号などのシステムの情報を API により取得できる。(「8.2.2.5 ts_ref_sys/tn_ref_sys – システム状態参照」、「8.2.2.6 ts_ref_ver/tn_ref_ver – バージョン参照」を参照。)

なおバージョン番号は、ts_ref_ver/tn_ref_ver の T_RVER 構造体を介して、取得することができる。

```
typedef struct st_rver {
        UH      maker;          /* カーネルのメーカーコード */
        UH      prid;           /* カーネルの識別番号 */
        UH      spver;          /* 仕様書バージョン番号 */
        UH      prver;          /* カーネルのバージョン番号 */
        UH      prno[4];        /* カーネル製品の管理情報 */
} T_RVER;
```

バージョン番号情報は、以下のように定められる

maker は、TRON Safe Kernel のカーネルを実装したメーカーコードである。

prid は、カーネルの種類を識別するための番号である。

prid の具体的な値の割付けは、カーネルを実装した TRON Safe Kernel の提供者に任される。ただし、製品の区別はあくまでもこの番号のみで行うので、提供者において番号の付け方を十分に検討した上、体系づけて使用するようにしなければならない。

これらから、maker と prid の組でカーネルの種類を一意に識別することができる。

TRON Safe Kernel のオリジナルは、トロンフォーラムから提供され、その maker と prid は次のようになる。

maker = 0x0000

prid = 0x0000

spver では、上位 4 ビットで OS 仕様の種類と、下位 12 ビットでカーネルが準拠する仕様のバージョン番号を表す。

例えば TRON Safe Kernel の Ver 1.00.xx の仕様書に対応する spver は次のようになる。

MAGIC = 0x3 (TRON Safe Kernel)

SpecVer = 0x100 (Ver 1.00)

spver = 0x3100

また、TRON Safe Kernel 仕様書のドラフト版である Ver 1.B0.xx の仕様書に対応する spver は次のようになる。

MAGIC = 0x3 (TRON Safe Kernel)

SpecVer = 0x1B0 (Ver 1.B0)

spver = 0x31B0

MAGIC:

OS 仕様の種類

- 0x0 TRON 共通 (TAD 等)
- 0x1 reserved
- 0x2 reserved
- 0x3 TRON Safe Kernel
- 0x4 AMP T-Kernel
- 0x5 SMP T-Kernel
- 0x6 μT-Kernel
- 0x7 T-Kernel

SpecVer は、カーネルが準拠する仕様のバージョン番号である。

3 桁のパックド形式 BCD コードで入れる。ドラフトの仕様書の場合は、上から 2 桁目が A, B, C となる場合もある。この場合は、対応する 16 進数の A, B, C を入れる。

prver は、カーネル実装上のバージョン番号を表す。prver の具体的な値の割付けは、カーネルを実装した TRON Safe Kernel 提供者に任される。

prno は、カーネル製品の管理情報や製品番号などを入れるために使用するためのリターンパラメータである。prno の具体的な値の意味は、カーネルを実装した TRON Safe Kernel 提供者に任される。

3.2.3 システム管理機能の API

TRON Safe Kernel のシステム管理機能は、「表 3-4 システム管理機能 API 一覧」に示す API を提供する（各 API の詳細は「8.2.2 システム状態管理機能の API」参照）。

表 3-4 システム管理機能 API 一覧

分類	安全 API 名称	通常 API 名称	説明
タスクスケジューリング	ts_rot_rdq	tn_rot_rdq	タスクの優先順位の回転
	ts_get_tid	tn_get_tid	実行状態タスクのタスク ID 参照
ディスパッチ禁止	ts_dis_dsp	tn_dis_dsp	ディスパッチ禁止
	ts_ena_dsp	tn_ena_dsp	ディスパッチ許可
状態参照	ts_ref_sys	tn_ref_sys	システム状態参照
	ts_ref_ver	tn_ref_ver	バージョン参照

3.3 同期・通信機能

3.3.1 セマフォ

セマフォは、使用されていない資源の有無や数量を数値で表現することにより、その資源を使用する際の排他制御や同期を行うためのカーネルオブジェクトである。

セマフォは、対応する資源の有無や数量を表現する資源数と、資源の獲得を待つタスクの待ち行列を持つ。

資源を返却する側では、セマフォのカウント値(セマフォカウント)を指定された数だけ増やす。

資源を獲得する側では、セマフォカウントを指定された数だけ減らす。セマフォカウントが足りなくなった場合(具体的には、資源を獲得する側で指定した資源数を減らすとセマフォカウントが負になる場合)、資源を獲得しようとしたタスクは、次に資源が返却されるまでセマフォ資源の獲得待ち状態となる。セマフォ資源の獲得待ち状態になったタスクは、そのセマフォの待ち行列につながれる。

また、セマフォに対して資源が返却され過ぎるのを防ぐために、セマフォ毎にセマフォカウントの最大値を設定することができる。最大値を超える資源がセマフォに返却されようとした場合(具体的には、資源を返却する側で指定した資源数を増やし、セマフォカウントが最大値を超える場合)には、エラーを報告する。

セマフォの API を「表 3-5 同期・通信機能(セマフォ)API 一覧」に示す(各 API の詳細は「8.2.3 同期・通信機能(セマフォ)の API」参照)。

表 3-5 同期・通信機能(セマフォ)API 一覧

分類	安全 API 名称	通常 API 名称	説明
セマフォの生成・削除	ts_cre_sem	tn_cre_sem	セマフォ生成
	ts_del_sem	tn_del_sem	セマフォ削除
セマフォの資源操作	ts_sig_sem	tn_sig_sem	セマフォ資源返却
	ts_wai_sem	tn_wai_sem	セマフォ資源獲得待ち
セマフォの状態参照	ts_ref_sem	tn_ref_sem	セマフォ状態参照

3.3.2 イベントフラグ

イベントフラグは、イベントの有無をビット毎のフラグで表現することにより、同期を行うためのカーネルオブジェクトである。
イベントフラグは、対応するイベントの有無をビット毎に表現するビットパターンと、そのイベントフラグで待つタスクの待ち行列を持つ。

イベントを知らせる側では、イベントフラグのビットパターンの指定したビットをセットすることでイベントを通知する。
イベントを待つ側では、イベントフラグのビットパターンの指定したビットのすべてまたはいずれかがセットされるまで、タスクをイベントフラグ待ち状態にすることができる。イベントフラグ待ち状態になったタスクは、そのイベントフラグの待ち行列につながれる。

イベントフラグのビットパターンは、UINT 型で指定する。イベントフラグのビットパターンの操作は UINT 型のビット数を単位として処理する。

イベントフラグの API を「表 3-6 同期・通信機能(イベントフラグ)API 一覧」に示す（各 API の詳細は「8.2.4 同期・通信機能（イベントフラグ）の API」参照）。

表 3-6 同期・通信機能(イベントフラグ)API 一覧

操作	安全 API 名称	通常 API 名称	API の説明
生成・削除	ts_cre_flg	tn_cre_flg	イベントフラグ生成
	ts_del_flg	tn_del_flg	イベントフラグ削除
資源操作	ts_set_flg	tn_set_flg	イベントフラグのセット
	ts_clr_flg	tn_clr_flg	イベントフラグのクリア
	ts_wai_flg	tn_wai_flg	イベントフラグ待ち
状態参照	ts_ref_flg	tn_ref_flg	イベントフラグ状態参照

3.3.3 ミューテックス

ミューテックスは、共有資源を使用する際にタスク間で排他制御を行うためのカーネルオブジェクトである。
ミューテックスは、ロックされているかどうかの状態と、ロックを待つタスクの待ち行列を持つ。タスクは、資源を使用する前に、ミューテックスをロックする。ミューテックスが他のタスクにロックされていた場合には、ミューテックスがロック解除されるまで、ミューテックスのロック待ち状態となる。ミューテックスのロック待ち状態になったタスクは、そのミューテックスの待ち行列につながれる。タスクは、資源の使用を終えると、ミューテックスのロックを解除する。

ミューテックスは、排他制御に伴う上限のない優先度逆転を防ぐための機構として、優先度継承プロトコル(priority inheritance protocol)と優先度上限プロトコル(priority ceiling protocol)をサポートする。
上限のない優先度逆転とは、低い優先度のタスクが、高い優先度のタスクとの間で共有するリソースをロックしているときに、中程度の優先度のタスクによりプリエンプティブされることにより、高い優先度のタスクが中程度の優先度のタスクにより「時間的に際限なく」待ち状態にさせられる現象をいう。

(1) 優先度継承プロトコル
ミューテックス属性に TA_INHERIT を指定することにより優先度継承プロトコルを選択できる。
優先度継承プロトコルは、ミューテックスをロックしているタスクの現在優先度をロック待ちのタスクの優先度の変化に合わせて制御する方式である。
ミューテックスをロックしたタスクの現在優先度には、そのミューテックスを待っているタスクの中の最高の現在優先度が設定される。

(2) 優先度上限プロトコル
ミューテックス属性に、TA_CEILING を指定することにより優先度上限プロトコルを選択できる。
優先度上限プロトコルは、ミューテックスに上限優先度を設定してタスクの優先度を制御する方式である。
タスクの現在優先度にはロックしたミューテックスの中の最高の上限優先度が設定される(タスクのベース優先度（優先度高）＞上限優先度（優先度低）の場合は除く)。
TA_CEILING 属性のミューテックスを、その上限優先度よりも高いベース優先度を持つタスクがロックしようとした場合、E_ILUSE エラーとなる。また、TA_CEILING 属性のミューテックスをロックしているかロックを待っているタスクのベース優先度を、そのミューテックスの上限優先度よりも高く設定しようとした場合、ts_chg_pri/tn_chg_pri が E_ILUSE エラーを返す。

これらのプロトコルを用いた場合、上限のない優先度逆転を防ぐために、ミューテックスの操作に伴ってタスクの現在優先度が自動的に変更される。優先度継承プロトコルと優先度上限プロトコルに厳密に従うため、タスクの現在優先度を、次に挙げる優先度の最高値に常に一致するよう変更する。

- タスクのベース優先度
- タスクが TA_INHERIT 属性のミューテックスをロックしている場合、それらのミューテックスのロックを待っているタスクの中で、最も高い現在優先度を持つタスクの現在優先度
- タスクが TA_CEILING 属性のミューテックスをロックしている場合、それらのミューテックス中で、最も高い上限優先度を持つミューテックスの上限優先度

ここで、TA_INHERIT 属性のミューテックスを待っているタスクの現在優先度が、ミューテックス操作か ts_chg_pri/tn_chg_pri によ

るベース優先度の変更に伴って変更された場合、そのミューテックスをロックしているタスクの現在優先度の変更が必要になる場合がある。これを推移的な優先度継承と呼ぶ。さらにそのタスクが、別の TA_INHERIT 属性のミューテックスを待っていた場合には、そのミューテックスをロックしているタスクに対して推移的な優先度継承の処理が必要になる場合がある。

ミューテックスの操作に伴ってタスクの現在優先度を変更した場合には、次の処理を行う。
優先度を変更されたタスクが実行できる状態である場合、タスクの優先順位を、変更後の優先度に従って変化させる。変更後の優先度と同じ優先度を持つタスクの間での優先順位は最低になる。優先度を変更されたタスクが何らかのタスク優先度順の待ち行列につながれている場合も、その待ち行列の中での順序を、変更後の優先度に従って変化させる。変更後の優先度と同じ優先度を持つタスクの間での順序は最低になる。

タスクが終了する時に、そのタスクがロックしているミューテックスが残っている場合には、それらのミューテックスをすべてロック解除する。

ミューテックスの APIを「表 3-7 同期・通信機能(ミューテックス)API 一覧」に示す（各 API の詳細は「8.2.5 同期・通信機能（ミューテックス）の API」参照）。

表 3-7 同期・通信機能(ミューテックス)API 一覧

操作	安全 API 名称	通常 API 名称	API の説明
生成・削除	ts_cre_mtx	tn_cre_mtx	ミューテックス生成
	ts_del_mtx	tn_del_mtx	ミューテックス削除
資源操作	ts_loc_mtx	tn_loc_mtx	ミューテックスのロック
	ts_unl_mtx	tn_unl_mtx	ミューテックスのロック解除
状態参照	ts_ref_mtx	tn_ref_mtx	ミューテックス状態参照

3.3.4 メッセージバッファ

メッセージバッファは、可変長のメッセージを受渡しすることにより、同期と通信を行うためのカーネルオブジェクトである。

メッセージバッファは、送信されたメッセージを格納するためのメッセージバッファ領域を持つ。また、メッセージの送信を待つタスクの待ち行列(送信待ち行列)とメッセージの受信を待つタスクの待ち行列(受信待ち行列)を持つ。
メッセージを送信する側では、送信したいメッセージをメッセージバッファにコピーする。メッセージバッファ領域の空き領域が足りなくなった場合、メッセージバッファ領域に十分な空きができるまでメッセージバッファへの送信待ち状態になる。メッセージバッファへの送信待ち状態になったタスクは、対象メッセージバッファの送信待ち行列につながれる。
メッセージバッファへの送信を待っているタスクは、待ち行列につながれている順序でメッセージを送信する。
例えば、あるメッセージバッファに対して 40 バイトのメッセージを送信しようとしているタスク A と、10 バイトのメッセージを送信しようとしているタスクＢが、この順で待ち行列につながれている時に、別のタスクによるメッセージの受信により 20 バイトの空き領域ができたとする。このような場合でも、タスク A がメッセージを送信するまで、タスク B はメッセージを送信できない。
メッセージを受信する側では、メッセージバッファに入っているメッセージを一つ取り出す。メッセージバッファにメッセージが入っていない場合は、次にメッセージが送られてくるまでメッセージバッファからの受信待ち状態になる。メッセージバッファからの受信待ち状態になったタスクは、対象メッセージバッファの受信待ち行列につながれる。
メッセージバッファからの受信を待っているタスクは、待ち行列につながれている順序でメッセージを受信する。

メッセージバッファからの受信待ち状態になっているタスクがある時に、メッセージを送信した場合、メッセージはメッセージバッファ領域にコピーされることなく、送信側のバッファから受信側のバッファにコピーされる。
メッセージバッファ領域のサイズを 0 にすることで、同期メッセージ機能を実現することができる。
すなわち、送信側のタスクと受信側のタスクが、それぞれ相手のタスクがシステムコールを呼び出すのを待ち合わせ、両者がシステムコールを呼び出した時点で、メッセージの受渡しが行われる。

メッセージバッファの API を「表 3-8 同期・通信機能(メッセージバッファ)API 一覧」に示す（各 API の詳細は「8.2.6 同期・通信機能（メッセージバッファ）の API」参照）。

表 3-8 同期・通信機能(メッセージバッファ)API 一覧

操作	安全 API 名称	通常 API 名称	API の説明
生成・削除	ts_cre_mbf	tn_cre_mbf	メッセージバッファ生成
	ts_del_mbf	tn_del_mbf	メッセージバッファ削除
送受信	ts_snd_mbf	tn_snd_mbf	メッセージバッファへ送信
	ts_rcv_mbf	tn_rcv_mbf	メッセージバッファから受信
状態参照	ts_ref_mbf	tn_ref_mbf	メッセージバッファ状態参照

3.4 時間管理機能

3.4.1　時間管理機能の概要

時間管理機能は、時間に依存した処理を行うための機能である。システム時刻とタイムイベントハンドラの管理機能を持つ。タイムイベントハンドラは、周期ハンドラとアラームハンドラに分けられる。

(1) システム時刻管理

TRON Safe Kernel が管理する時刻を、システム時刻という。システム時刻は、TRON Safe Kernel でただ一つ存在する。TRON Safe Kernel はシステム時刻をもとに、タスクやハンドラなどの実行単位の時間に関する管理、制御を行う。

システム時刻の単位はミリ秒とする。ただし、実際のシステム時刻の精度、時間分解能はハードウェアに依存するため、実装定義である。

システム時刻の値は、1985 年 1 月 1 日 0 時(GMT)を基準として通算のミリ秒数を標準とする。ただし、ハードウェアやアプリケーションの要求仕様により、実装定義を許す。TRON Safe Kernel 自体は、システム時刻の経過時間のみを利用するので、時刻の基準の影響は受けない。

(2)タイムイベントハンドラ管理

タイムイベントハンドラは、時間の経過に従って起動されるプログラムの実行単位であり、カーネルオブジェクトである。タイムイベントは特定のドメインに所属する。

タイムイベントハンドラには周期ハンドラとアラームハンドラの二種類がある。周期ハンドラは、時間の経過に従って一定周期で起動されるタイムイベントハンドラである。アラームハンドラは、特定の時刻に一回のみ起動されるタイムイベントハンドラである。

タイムイベントハンドラには以下の特徴がある。

- タイムイベントハンドラは、所属するドメインの最高実行優先度で実行される。優先度の変更はできない。
- タイムイベントハンドラの優先順位は、同一ドメインのタスクより優先順位が高い。
- タイムイベントハンドラは、待ち状態になることはできない。よって、待ち状態を伴う API は使用できない。

3.4.2 タイムイベントハンドラの概要

各タイムイベントハンドラは「表 3-9 タイムイベントハンドラ管理情報」に示す情報を持つ。各情報は、API によって参照できる。また、一部の情報は API により変更できる。

表 3-9 タイムイベントハンドラ管理情報

名称	意味	API による変更※1
ハンドラ ID	タイムイベントハンドラを識別する ID 番号	不可
所属ドメイン ID	タイムイベントハンドラが所属するドメインの ID 番号	不可
保護レベル	タイムイベントハンドラの保護レベル	不可
ハンドラ属性	タイムイベントハンドラの属性	不可
ハンドラ状態	タイムイベントハンドラの状態	可※2
起動時刻	アラームハンドラを起動する時刻(アラームハンドラのみ)	可
起動時間間隔	周期ハンドラを起動する間隔(周期ハンドラのみ)	不可
周期起動位相	周期ハンドラを起動する位相時間(周期ハンドラのみ)	不可
ハンドラ起動コード	タイムイベントハンドラの起動時に渡されるデータ	不可
ハンドラ起動アドレス	タイムイベントハンドラの実行プログラムの起動アドレス	不可
最大連続実行時間	タイムイベントハンドラが連続実行できる時間の上限	不可
ユーザスタック情報	タイムイベントハンドラが通常使用するスタックの情報	不可
システムスタック情報	タイムイベントハンドラ実行中にシステムが使用するスタックの情報	不可
オブジェクト名	オブジェクトを識別するための名称(文字列)	不可

※1 生成後に API により動的な変更ができるか否か

※2 API の結果として変更されるが、直接 API から値を指定することはできない

ハンドラ ID は、タイムイベントハンドラ生成時に TRON Safe Kernel により自動的に割り当てられる。

所属ドメインは、タイムイベントハンドラが所属するドメインであり、そのタイムイベントハンドラを生成したプログラムが所属するドメインとなる。

保護レベルは、タイムイベントハンドラの所属するドメインで定められた保護レベルが設定される。

ハンドラ属性は、タイムイベントハンドラ生成時に指定されたハンドラ属性値である。詳細は「3.4.5 タイムイベントハンドラのメイン関数」を参照。

ハンドラ状態は現在のタイムイベントハンドラの状態である。詳細は「3.4.4 タイムイベントハンドラの状態」を参照。

ハンドラ起動アドレスは、タイムイベントハンドラの起動時に実行されるプログラムの起動アドレスである。詳細は「3.4.5 タイムイベントハンドラのメイン関数」を参照。

最大連続実行時間は、タイムイベントハンドラが連続実行可能な時間の上限を設定する。

ユーザスタック情報は、タイムイベントハンドラのユーザスタックの情報であり、システムスタック情報は、タイムイベントハンドラのシステムスタックの情報である。

オブジェクト名は、タイムイベントハンドラの生成時に指定されたタイムイベントハンドラの名称である。

3.4.3 タイムイベントハンドラの属性

タイムイベントハンドラの生成時にハンドラ属性が指令される。「表 3-10 タイムイベントハンドラ属性」にハンドラ属性を示す。

表 3-10 タイムイベントハンドラ属性

名称	属性	意味	ハンドラ種別
TA_STA	ハンドラ起動	ハンドラ生成後、動作状態とする	周期ハンドラのみ
TA_PHS	起動位相を保存	停止中も周期ハンドラの起動位相を保存する	周期ハンドラのみ
TA_ONAME	オブジェクト名指定	オブジェクト名称を指定する	全て

3.4.4 タイムイベントハンドラの状態

タイムイベントハンドラの状態は以下に分類される。

(1) 実行状態

タイムイベントハンドラが実行されている状態である。

(2) 動作状態

タイムイベントハンドラが起動する時刻を待っている状態である。

(3) 停止状態

タイムイベントハンドラがまだ起動されていないか、実行を終了した後の状態である。

タイムイベントハンドラの起動とは、停止状態のタイムイベントハンドラを動作状態に移行させることをいう。

動作状態のタイムイベントハンドラは、起動時刻に達し、かつより優先順位の高いプログラムの実行単位が実行されていなければ、実行状態に遷移する。

実行状態のタイムイベントハンドラは処理を終えたら、自らを終了する。周期ハンドラは終了すると、動作状態に遷移し、次の起動時刻を待つ。アラームハンドラは終了すると、停止状態に遷移する。

周期ハンドラの状態遷移を「図 3-2 周期ハンドラの状態遷移」に示す。

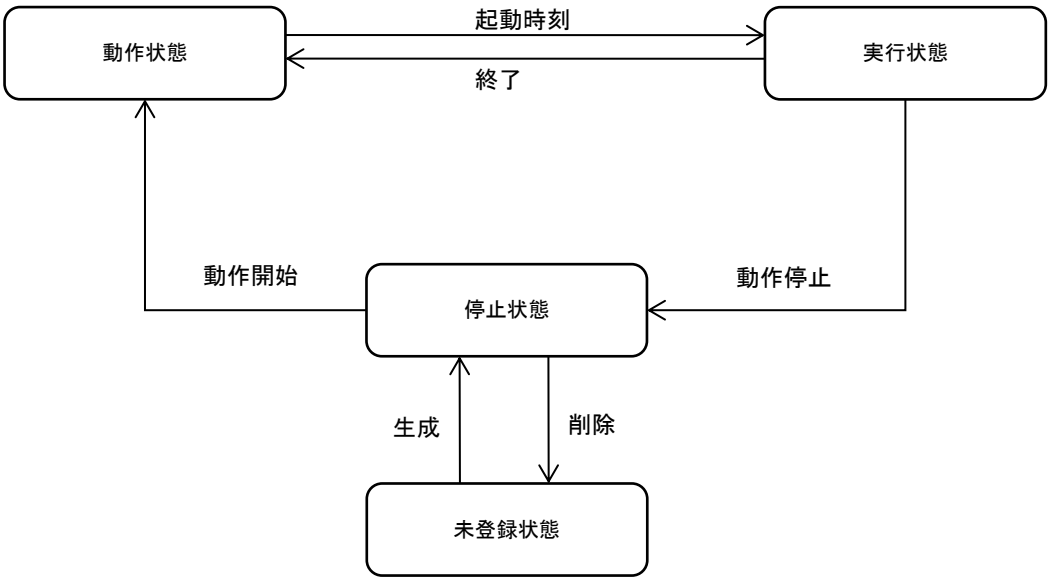

図 3-2 周期ハンドラの状態遷移

アラームハンドラの状態遷移を「図 3-3 アラームハンドラの状態遷移」に示す。

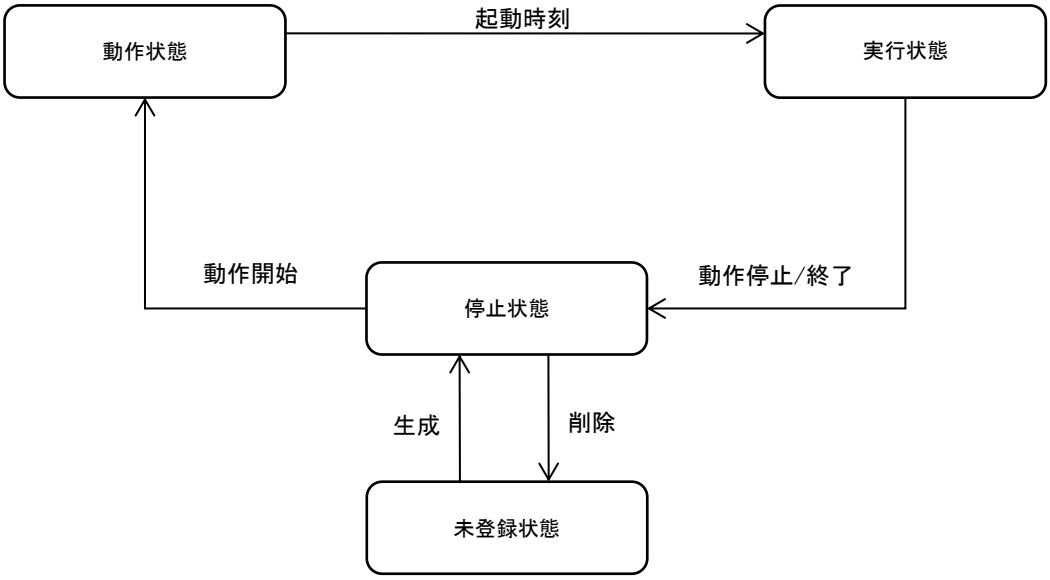

図 3-3 アラームハンドラの状態遷移

3.4.5 タイムイベントハンドラのメイン関数

タイムイベントハンドラの起動時に実行されるプログラムを、そのタイムイベントハンドラのメイン関数と呼ぶ。

タイムイベントハンドラのメイン関数は以下の形式の C 言語の関数とする。関数名は任意である。

```
void hdr_main( INT stacd)
```

関数のパラメータとして、生成時にユーザが指定した起動コード stacd が渡される。

3.4.6 タイムイベントハンドラのスケジューリング規則

タイムイベントハンドラは、以下の規則に従い、実行される。

- タイムイベントハンドラの優先度は、所属するドメインの最高実行優先度である。
- タイムイベントハンドラは、自ドメインの中で同一優先度のタスクより高い優先順位を持つ。
- 同一ドメインでタイムイベントハンドラの起動時間が同一だった場合、先に動作状態となったタイムイベントハンドラを高い優先順位とする。
- ドメインの種別が異なる場合、タイムイベントハンドラやタスクの優先順位は、以下の順で決まる。
  システムドメイン（優先度高） ＞ 安全ドメイン ＞ 通常ドメイン（優先度低）
- 同じ種別の異なったドメインで、タイムイベントハンドラやタスクの優先順位は、ドメインの ID 番号が小さいものほど優先順位が高くなる（ドメインの ID 番号は静的に設定されている）。

上記のスケジューリング規則により、実行状態のタイムイベントハンドラは自ドメインのタイムイベントハンドラまたはタスクによってプリエンプトされることはない。一方、他のドメインのタイムイベントハンドラまたはタスクからプリエンプトされることは起こりうる。

3.4.7 周期ハンドラの動作

周期ハンドラの生成時に、周期ハンドラの起動周期と起動位相を、周期ハンドラ毎に設定する。
TRON Safe Kernel は、設定された起動周期と起動位相から、周期ハンドラを次に起動すべき時刻を決定する。最初に起動する時刻は、周期ハンドラを生成した時刻に起動位相を加えた時刻である。
周期ハンドラを起動すべき時刻になると、その周期ハンドラが動作状態であれば、起動コード(stacd)をパラメータとして、周期ハンドラを起動する。またこの時、周期ハンドラの起動すべき時刻に起動周期を加えた時刻を、次に起動すべき時刻とする。
周期ハンドラが停止状態の時には、周期ハンドラを起動すべき時刻となっても周期ハンドラを起動せず、次に起動すべき時刻の決定のみを行う。
周期ハンドラの動作を開始するシステムコール(ts_sta_cyc/tn_sta_cyc)が呼び出されると、周期ハンドラを動作状態に移行させ、ハンドラ属性により必要なら周期ハンドラを次に起動すべき時刻を決定しなおす。
周期ハンドラの動作を停止するシステムコール(ts_stp_cyc/tn_stp_syc)が呼び出されると、周期ハンドラを停止状態に移行させる。
周期ハンドラの実行状態は、生成時に指定した連続実行時間を超えてはならない。時間を超えた場合は異常例外が発生する。
また、周期ハンドラの連続実行時間は、周期ハンドラの周期時間より短くなければならない。

3.4.8 アラームハンドラ

アラームハンドラを起動する時刻は、アラームハンドラ毎に設定する。アラームハンドラの起動時刻になると、そのアラームハンドラの起動コード(stacd)をパラメータとして、アラームハンドラを起動する。

アラームハンドラの生成直後は、アラームハンドラの起動時刻が設定されておらず、アラームハンドラは停止状態である。
アラームハンドラの動作を開始する API(ts_sta_alm/tn_sta_alm)が呼び出されると、アラームハンドラの起動時刻を、API が呼び出された時刻から指定された相対時間後に設定し、動作状態とする。

アラームハンドラの動作を停止するシステムコール(ts_stp_alm/tn_stp_alm)が呼び出されると、アラームハンドラの起動時刻の設定を解除し、停止状態とする。

アラームハンドラの実行状態は、生成時に指定した連続実行時間を超えてはならない。時間を超えた場合は異常例外が発生する。

3.4.9 時間管理機能の API

TRON Safe Kernel の時間管理機能は、「表 3-11 時間管理機能 API 一覧」に示す API を提供する(各 API の詳細は「8.2.7 時間管理機能の API」参照)。

表 3-11 時間管理機能 API 一覧

分類	安全 API 名称	通常 API 名称	説明
システム時刻	ts_set_tim	tn_set_tim	システム時刻設定
	ts_get_tim	tn_get_tim	システム時刻参照
	ts_get_otm	tn_get_otm	システム稼働時間参照
周期ハンドラ	ts_cre_cyc	tn_cre_cyc	周期ハンドラの生成
	ts_del_cyc	tn_del_cyc	周期ハンドラの削除
	ts_sta_cyc	tn_sta_cyc	周期ハンドラの動作開始
	ts_stp_cyc	tn_stp_cyc	周期ハンドラの動作停止
	ts_ref_cyc	tn_ref_cyc	周期ハンドラ状態参照
アラームハンドラ	ts_cre_alm	tn_cre_alm	アラームハンドラの生成
	ts_del_alm	tn_del_alm	アラームハンドラの削除
	ts_sta_alm	tn_sta_alm	アラームハンドラの動作開始
	ts_stp_alm	tn_stp_alm	アラームハンドラの動作停止
	ts_ref_alm	tn_ref_alm	アラームハンドラ状態参照

3.5 システム構成情報管理機能

3.5.1 システム構成情報管理機能の概要

システム構成情報管理機能は、システム構成に関する各種の情報を保持・管理するための機能である。

システム構成情報の一部は、標準システム構成情報として定めてあり、その中で最大タスク数やタイマ割込み間隔などの情報を定義する。またそれ以外にもアプリケーションやサブシステム、デバイスドライバで任意に定義した情報を追加して利用できる。

システム構成情報はコンフィギュレーションにて静的に決定され、TRON Safe Kernel の実行中に変更することはできない。

3.5.2 システム構成情報の形式

システム構成情報の形式は、名称と定義データの組である。

(1) 名称

名称は最大 16 文字の文字列で表される。使用可能な文字は以下の英数字である。

使用可能な文字(UB 型) 'a'～'z' 'A'～'Z' '0'～'9' '_'

(2) 定義データ

定義データは、数値の並び、または文字列である。

数値の並びは、INT 型の整数の配列として取得できる。

文字列は、最大 128 文字の文字列であり、使用可能な文字は以下とする。

使用可能な文字(UB 型) 0x00～0x1F, 0x7F, 0xFF(文字コード)を除く文字

3.5.3 標準システム構成情報

標準システム構成情報は、TRON Safe Kernel が標準で必ず持つシステム構成情報である。標準システム構成情報の名称は T で始まる。

「表 3-12 標準システム構成情報一覧」に標準システム構成情報の一覧を示す。

表 3-12 標準システム構成情報一覧

名称	型	要素数	説明
TMaxTskId	N	1	最大タスク数
TMaxSemId	N	1	最大セマフォ数
TMaxFlgId	N	1	最大イベントフラグ数
TMaxMtxId	N	1	最大ミューテックス数
TMaxMbfId	N	1	最大メッセージバッファ数
TMaxCycId	N	1	最大周期ハンドラ数
TMaxAlmId	N	1	最大アラームハンドラ数

※ 上記情報は、ドメイン ID 別に定義される。

※ 「表 3-12 標準システム構成情報一覧」以外のシステム構成情報については、その拡張有無・実現方法を含め実装定義とする。

3.5.4 システム構成情報管理機能の API

TRON Safe Kernel のシステム構成情報管理機能は、「表 3-13 システム構成情報管理機能API一覧」に示す API を提供する（各 API の詳細は「8.2.8 システム構成情報管理機能の API」参照）。

表 3-13 システム構成情報管理機能API一覧

分類	安全 API 名称	通常 API 名称	説明
システム構成情報管理	ts_get_cfn	tn_get_cfn	システム構成情報（数値列）の取得
	ts_get_cfs	tn_get_cfs	システム構成情報（文字列）の取得

3.6 ドメイン管理機能

3.6.1　ドメイン管理機能の概要

ドメインは、TRON Safe Kernel にて各プログラムが使用する資源を管理する単位である。

ドメイン管理機能は、ドメインに対する各種の操作を行う。

各ドメインは、「表 3-14 ドメイン管理情報」に示す情報を持つ。各情報はドメインの生成時に静的に決定される。各情報は安全 API にて参照することができる。

表 3-14 ドメイン管理情報

名称	意味	動的変更※1
ドメイン ID	ドメインを識別する ID 番号	不可
保護レベル	ドメインに属するプログラム、メモリの保護レベル	不可
ドメイン属性	ドメインの属性	不可
ドメイン状態	ドメインの状態	可※2
初期タスク	ドメインの実行時に最初に実行されるタスク	不可
最高実行優先度	ドメインに属するタスクおよびタイムイベントハンドラが取り得る最高の優先度	不可
タスク 最大連続実行時間	ドメインに属する個々のタスクが連続して実行できる時間の上限	不可
タイムイベントハンドラ 最大連続実行時間	ドメインに属する個々のタイムイベントハンドラが連続して実行できる時間の上限	不可
ドメイン 最大連続実行時間	ドメインに属するタスクおよびタイムイベントハンドラが連続して実行できる時間の上限	不可

※1 生成後に API により動的な変更ができるか否か

※2 API の結果として変更されるが、直接 API から値を指定することはできない

ドメイン ID は、静的に定められたドメインを識別する ID 番号である。詳細は「3.6.3 ドメイン ID と優先順位」を参照。

保護レベルは、そのドメインに所属するプログラムやメモリの保護レベルであり、ドメインの種別により決定される。詳細は「2.3.2 保護レベル」を参照。

ドメイン状態は、ドメインの状態を示す。詳細は「3.6.7 ドメインの状態」を参照。

初期タスクは、ドメインの生成時に実行されるタスクである。詳細は「3.6.4 ドメインの生成と実行」を参照。

最高実行優先度は、ドメインに属するタスクおよびタイムイベントハンドラが取り得る最高の優先度である。

タスク最大連続実行時間、タイムイベントハンドラ最大連続実行時間、ドメイン最大連続実行時間は、ドメインに所属するプログラムの実行単位の連続実行時間の上限である。詳細は「3.6.10 ドメインの時間的保護」を参照。

3.6.2 ドメイン属性

ドメイン登録時にドメイン属性が指定される。「表 3-15 ドメイン属性」にドメイン属性を示す。

表 3-15 ドメイン属性

名称	属性	意味
TA_START	起動時実行属性	TRON Safe Kernel 起動時にドメイン実行する（安全ドメインのみ有効）

3.6.3 ドメイン ID と優先順位

各ドメインにはドメインを識別するための ID 番号が静的に割り当てられる。

システムドメインの ID 番号は必ず 1 である。安全ドメインの ID 番号は 2 から始まる連続した数値である。通常ドメインの ID 番号は、安全ドメインの最大の ID 番号の次の値から始まる連続した数値である。よって、ドメイン ID 番号はドメインの種別により以下の関係となる。

システムドメイン ID(1),
安全ドメイン ID(2), ・・・ 安全ドメイン ID(N+1),
通常ドメイン ID(N+2), ・・・ 通常ドメイン ID(N+M+1)

※ N は安全ドメインの最大数、M は通常ドメインの最大数

ドメイン ID 番号は値が小さいほど、以下のように優先的である。

- タスクやタイムイベントハンドラの実行は、基本的にそれぞれの実行優先度に従ってスケジューリングされるが、優先度が同じ場合はドメインの優先順位が高いものほど優先される。
- TRON Safe Kernel の起動時に、起動時実行属性のドメインはドメイン ID 番号の小さいものから順番に実行される。

3.6.4 ドメインの生成と実行

すべてのドメインは静的に生成される。プログラムの実行中に動的にドメインが生成や削除が行われることはない。

TRON Safe Kernel が起動（カーネル実行状態に遷移）すると、まずシステムドメインが実行される。続いて、安全ドメインの中で、起動時実行属性が設定されているものが実行される。該当する安全ドメインが複数存在する場合はドメイン ID 番号の小さいものから順番に実行される。通常ドメインは TRON Safe Kernel の起動時には実行することはできない。

TRON Safe Kernel の起動時に実行されなかったドメインは安全 API により実行される。

ドメインの実行とは、そのドメインの内部情報の初期化と、初期タスクの実行である。初期タスクはドメイン毎に一つ存在する特別なタスクであり、初期タスクの生成、実行は、カーネルにて実行されるので明示的に API を呼ぶ必要はない。

初期タスク以外のタスク、タイムイベントハンドラ、その他のカーネルオブジェクトは、API により生成する必要がある。

3.6.5 ドメインの終了

ドメインの終了は以下の場合に行われる。

- そのドメインの初期タスクの終了
- 安全 API または通常 API による自ドメイン終了
- 安全 API による他ドメインの強制終了

ドメインの終了時には以下の終了処理が行われる。
- ドメインに属する実行中の全てのタスクの強制終了
- ドメインに属する全てのカーネルオブジェクトの削除
- ドメインに属するタスクからオープンされている全てのデバイスのクローズ

ドメインの終了処理は、そのドメインの初期タスクのコンテキストにて実行される。ただし、本 API の処理中、対象ドメインはディスパッチ禁止状態となり、また対象ドメインに属するタイムイベントハンドラの実行は抑止される。よって、ドメインに所属するタスクやタイムイベントハンドラが実行されることはない。
ただし、終了処理は初期タスクの現在優先度で実行されているので、他のドメインのタスクやタイムイベントハンドラは実行される。終了処理中のドメインに属するカーネルオブジェクトへのアクセスは、エラーE_NOEXS となる。

システムドメインが終了すると、TRON Safe Kernel が終了する。ただし、終了時の処理は実装定義とする。
アプリケーションドメインは、終了後に他のドメインがドメインの実行開始(ts_sta_dmn)を発行することで、再び起動できる。

【補足説明】
システムドメインに対して、他ドメインから強制終了することはできない。また、システムドメインの初期タスクは終了しない。よって、システムドメインが終了するのは、システムドメイン内のタスクから、自ドメインの終了 API が実行される場合のみである。

3.6.6 ドメインの強制停止

ドメインは、他のドメインの実行単位より安全 API によって強制停止することができる。
システムドメインは強制停止することはできない。また、安全ドメインは強制停止可能か否か、生成時の設定にて静的に決められる。
強制停止状態のドメインに所属するタスクおよびタイムイベントハンドラは、TRON Safe Kernel のプログラム実行のスケジューリングから外され、実行されることはない。タスクおよびタイムイベントハンドラの状態は、強制停止状態になった時点の状態が保持される。
強制停止状態は安全 API により解除され、ドメインの実行が再開される。

【補足説明】
ドメインの強制停止は、ドメイン内のソフトウェアにおいて何らかの異常（エラー）が発生し、ドメイン内のプログラムの実行が許されない状態（ドメインエラー状態）においての使用を想定している。エラー以外の要因で、単にドメイン内のプログラムの実行を停止する場合は、通常のドメインの終了を行う。

3.6.7 ドメインの状態

ドメインは以下の状態をとる。

(1) ドメイン停止状態

ドメインが起動される前の所属するタスクおよびタイムイベントハンドラが停止している状態である。具体的にはタスクは全て休止状態（DORMANT）であり、タイムイベントハンドラは停止状態である。

ドメイン生成後はこの状態となり、安全 API により起動され実行状態に遷移する。ただし、システムドメインのみは TRON Safe Kernel の起動時に起動されているため実行状態からの開始となる。

(2) ドメイン実行状態

ドメインが実行中の状態である。ドメインに所属するタスクおよびタイムイベントハンドラが動作している。少なくともドメインの初期タスクは休止状態(DORMANT)以外である。

ドメインは初期タスクの終了または、API によるドメインの終了により、終了処理状態を経て停止状態になる。

(3) ドメイン終了処理状態

ドメインの終了処理を実行中の状態である。この状態は、実行状態から停止状態に遷移する途中の過渡的な状態である。ドメインの終了処理は、ドメインの初期タスクのコンテキストにて実行されている。本状態ではドメインはディスパッチ禁止状態となり、またタイムイベントハンドラの実行は抑止されるので、ドメインに所属する他のタスクやタイムイベントハンドラは動作しない。

(4) ドメイン強制停止状態

ドメインが強制停止されている状態である。ドメインに所属するすべてのタスクおよびタイムイベントハンドラの動作が停止している。

本状態は実行状態から安全 API により遷移する。また、安全 API により実行状態に復帰することができる。

「図 3-4 ドメインの状態遷移」にドメインの状態遷移図を示す。

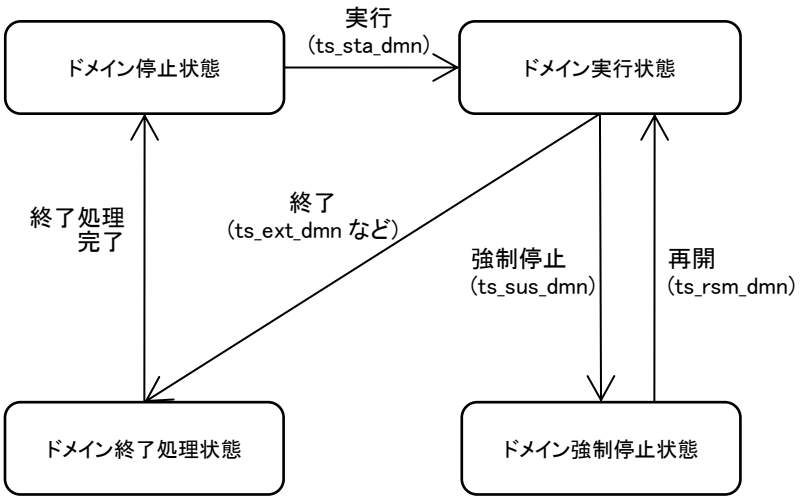

図 3-4 ドメインの状態遷移

3.6.8 ディスパッチ禁止

ディスパッチ禁止は、ドメインに所属するタスクのディスパッチが禁止される特別な状態である。
ディスパッチ禁止は API により行われる。また、ディスパッチ禁止の解除も API により行われる。API を発行できるのは、そのドメインに所属するタスクのみである。つまり、API を発行したタスクがディスパッチ禁止の間、そのドメインで実行され続ける。
ディスパッチ禁止中のドメインの動作は以下のようになる。

- 実行中のタスク(ディスパッチ禁止の API を発行したタスク)により、優先度の高いタスクが実行可能となっても、そのタスクにはディスパッチされない。優先度の高いタスクへのディスパッチは、ディスパッチ禁止状態が終了するまで遅延される。
- 実行中のタスクは実行状態以外の状態に遷移できない。状態を変更する可能性のある API を発行した場合は、エラー E_CTX となる。
- タイムイベントハンドラは実行される。ディスパッチ禁止はドメイン内のタスクのみに有効である。
- 原則として、実行中のタスクはタイムアウト時間内にディスパッチ禁止を解除しなくてはならない。API 発行時に指定したタイムアウト時間を超えてディスパッチ禁止が続いた場合は、ディスパッチ禁止は解除され、異常例外が発生する。

【補足説明】
ディスパッチ禁止はあくまで API が発行されたドメインにおいてのみ有効である。よって、他のドメインのタスクやタイムイベントハンドラは優先度に応じて実行される。
機能安全の観点から、無制限のディスパッチ禁止は許されない。ディスパッチ禁止にできるタスクの優先度に上限を設定できること、および、設定される上限はディスパッチ禁止を実行するタスクの所属しているドメインに設定されている上限を超えないことが要求される。よって、TRON Safe Kernel では、ディスパッチ禁止の範囲を、API を発行したタスクが所属しているドメインに限定した。これにより、ディスパッチ禁止されるタスクの優先度の上限は、ドメインに設定されている優先度上限となる。また、他のドメインへの影響を排除することができる。

3.6.9 ドメインの空間的保護

TRON Safe Kernel では、ソフトウェアが使用するメモリはドメインの単位で保護される。これをドメインの空間的保護という。
各ドメインには独立のメモリ領域が静的に割り当てられる。ドメイン間でメモリ領域を共有することはできない。

ドメインに割り当てられたメモリ領域はプロセッサのメモリ保護機能(MMU または MPU など)により保護され、他のドメインのソフトウェアからアクセスすることは禁止される。メモリのアクセス違反が発生した場合は異常例外が発生する。

アプリケーションは、プロセッサのメモリ保護機能を変更できない実行モードで実行される。つまり、アプリケーションは他のドメインのメモリ領域をアクセスすることはできない。
システムソフトウェアは特権的な実行モードで実行されるため、プロセッサのメモリ保護機能の変更は可能である。ただし、システムソフトウェアも、他のドメインのメモリ領域を直接操作することはない(API で直接指定された場合を除く)。

3.6.10 ドメインの時間的保護

TRON Safe Kernel では、ソフトウェアには実行優先度が与えられ、その優先度に従って実行される。
優先度の高いソフトウェアの実行が、優先度の低いソフトウェアの実行に阻害されないように、ドメイン単位で実行優先度と実行時間の制限を設けられる。

これをドメインの時間的保護と呼ぶ。
ドメインの時間的保護には以下がある。

(1) 優先度下限・上限

ドメインの最低/最高実行優先度は、そのドメインに属するタスクおよびタイムイベントハンドラの実行優先度の下限/上限である。ドメインの最低/最高実行優先度は、ドメイン毎に設定される。
タスクは、自身が属するドメインの最低/最高実行優先度以内の優先度しかとりえない。タイムイベントハンドラは常に自身が属するドメインの最高実行優先度の優先度で実行される。

システムドメインは最高優先度の制限がない。
安全ドメインの最高優先度は、システムドメインの最高優先度より低い値(大きい値)しか設定できない。よって、安全ドメインで設定可能な最高優先度は優先度 2 以下(2 及びそれより大きい値)である。
通常ドメインの最高優先度は、安全ドメインの最高優先度より低い値(大きい値)しか設定できない。

よって、各ドメインでとりうる最高優先度は以下の関係になる。

システムドメインの最高優先度(優先度高) ＞ 安全ドメインの最高優先度 ＞ 通常ドメインの最高優先度(優先度低)

(2) タスク最大連続実行時間

タスク最大連続実行時間は、そのドメインに属するタスクの連続実行時間の上限である。タスクは、自身が属するドメインの最大連続実行時間以下の連続実行時間しかとりえない。

(3) タイムイベントハンドラ最大連続実行時間

タイムイベントハンドラ最大連続実行時間は、そのドメインに属するタイムイベントハンドラの連続実行時間の上限である。タイムイベントハンドラは、自身が属するドメインの最大連続実行時間以下の連続実行時間しかとりえない。

(4) ドメイン連続実行時間

ドメイン連続実行時間は、そのドメインに属するタスクおよびタイムイベントハンドラが連続して実行できる時間である。同一のドメインであれば、タスクやタイムイベントハンドラが切り替わっても時間の計測は継続される。ドメイン連続実行時間を超えて、タスクやタイムイベントハンドラが実行されると異常例外が発生する。

3.6.11 ドメイン管理機能の API

TRON Safe Kernel のドメイン管理機能は、「表 3-16 ドメイン管理機能 API 一覧」に示す API を提供する（各 API の詳細は「8.2.9 ドメイン管理機能の API」参照）。

表 3-16 ドメイン管理機能 API 一覧

分類	安全 API 名称	通常 API 名称	説明
ドメインの実行管理	ts_sta_dmn	–	ドメインの実行開始
	ts_ext_dmn	tn_ext_dmn	自ドメインの実行終了
	ts_ter_dmn	–	他ドメインの強制終了
	ts_sus_dmn	–	他ドメインの強制停止
	ts_rsm_dmn	–	他ドメインの再開
カーネルオブジェクトの管理	ts_get_oid	tn_get_oid	カーネルオブジェクトの ID 取得
ドメインの情報	ts_ref_dmn	tn_ref_dmn	ドメイン状態取得
	ts_inf_dmn	tn_inf_dmn	ドメイン統計情報参照

4. TRON Safe Kernel/SM の機能

4.1 デバイス管理機能

4.1.1 デバイス管理機能の概要

デバイス管理機能は、TRON Safe Kernel のシステムドメインで動作するデバイスドライバを管理するための機能である。

デバイスドライバとは、主としてハードウェアのデバイスの操作や入出力を行うために、独立して実装されるプログラムである。

デバイス管理機能は、TRON Safe Kernel にデバイスドライバを登録するための機能と、登録済みのデバイスドライバを他のソフトウェアから利用するための機能が含まれる。

各デバイスドライバは「表 4-1 デバイスドライバ管理情報」に示す情報を持つ。各情報は、API によって参照できる。

表 4-1 デバイスドライバ管理情報

名称	意味	動的変更※1
デバイス ID	デバイスドライバを識別する ID 番号	不可
所属ドメイン	デバイスドライバが所属するドメイン（システムドメインに固定）	不可
保護レベル	デバイスドライバの保護レベル（保護レベル0に固定）	不可
デバイスドライバ属性	デバイスドライバの属性	不可
サブユニット数	デバイスドライバのサブユニットの数	不可
デバイスドライバ・インタフェース	デバイスドライバに各種処理を要求するためのインタフェース（ドライバインタフェース関数から構成される）	不可
事象通知用 ID	事象通知を送信するメッセージバッファの ID	可
オブジェクト名	オブジェクトを識別するための名称(文字列)	不可

※1 生成後に API により動的な変更ができるか否か

デバイス ID は、デバイスドライバ登録時に TRON Safe Kernel により自動的に割り当てられる。

所属ドメインはデバイスドライバが所属するドメインの ID 番号である。デバイスドライバは必ずシステムドメインに所属するので、システムドメインの ID 番号に固定される。

保護レベルは、デバイスドライバの保護レベルである。デバイスドライバは必ずシステムドメインに所属するので、システムドメインの保護レベルに固定される。

デバイスドライバ属性は、デバイスドライバの属性である。詳細は「4.1.2 デバイスドライバ属性」を参照。

サブユニット数は、デバイスドライバが持つサブユニットの数である。詳細は「4.1.3 ユニットとサブユニット」を参照。

デバイスドライバ・インタフェースは、TRON Safe Kernel のデバイス管理機能とデバイスドライバのインタフェースである。詳細は「4.1.5 デバイスドライバ・インタフェース」を参照。

事象通知用 ID は、事象通知を送信するメッセージバッファの ID である。詳細は「4.1.7 デバイス事象通知」を参照。

オブジェクト名は、デバイスドライバの登録時に指定されたデバイスドライバの名称である。この名称をもとにデバイス名が決められる。詳細は「4.1.7 デバイス事象通知」を参照。

4.1.2 デバイスドライバ属性

デバイスドライバの登録時にデバイスドライバ属性が指定される。「表 4-2 デバイスドライバ属性」にデバイスドライバ属性を示す。

表 4-2 デバイスドライバ属性

名称	属性	意味
TDA_SDEV	安全デバイス属性	安全 API により操作可能なデバイスドライバ （安全ドメインおよびシステムドメインのソフトウェアからのみ使用可能）
TDA_NDEV	通常デバイス属性	通常 API により操作可能なデバイスドライバ （通常ドメインのソフトウェアからのみ使用可能）
TDA_OPENREQ	多重オープン要求属性	多重オープンの際にオープン処理を毎回行う

デバイスドライバ属性の組合せを以下に示す。

デバイスドライバ属性 := (TDA_SDEV || TDA_NDEV) | TDA_OPENREQ

TRON Safe Kernel のデバイスドライバは、機能安全の観点より、デバイスドライバ属性により、安全 API により操作可能なものと、通常 API により操作可能なものに分かれる。
安全 API により操作可能なデバイスドライバは、安全ドメインおよびシステムドメインのソフトウェアからのみ使用可能である。
通常 API により操作可能なデバイスドライバは、通常ドメインのソフトウェアからのみ使用可能である。
通常ドメインと、安全ドメインまたはシステムドメインの両方から使用できるデバイスドライバは存在しない。

4.1.3 ユニットとサブユニット

ハードウェアのデバイスは、その中に複数の構造を持つものがある。その場合、デバイス全体を物理デバイス、その中の構造の一つを論理デバイスと呼ぶ。
同種類の物理デバイスは「ユニット」により区別され、1 つの物理デバイス内の論理デバイスは「サブユニット」により区別される。
デバイスドライバは、ユニット、つまり物理ドライバに一対一で対応する。物理デバイスがサブユニットを持つ場合、一つのデバイスドライバがその物理デバイスの持つすべてのサブユニットに対応する。

【補足事項】
ハードディスクを例にユニットとサブユニットを説明する。
ハードディスクの中に区画(パーティション)を作った場合は、ハードディスク全体が物理デバイスであり、その中の 1 つの区画が論理デバイスとなる。
ハードディスクが 2 台ある場合、1 台目のハードディスクと 2 台目のハードディスクを区別する情報が「ユニット」であり、1 台目のハードディスク内の 1 つ目の区画と 2 つ目の区画を区別する情報が「サブユニット」である。

4.1.4 デバイス名

デバイスには、固有のデバイス名が与えられる。

デバイス名は、最大 8 文字の文字列であり、物理デバイス名とサブユニット番号から構成される。

物理デバイス名は、英文字(a～z A～Z)で構成される。

サブユニット番号は最大 3 文字の数字であり、0 から順に最大 254 までとする。

物理デバイスの場合は、デバイス名は物理デバイス名のみである。

論理デバイスの場合は、デバイス名は物理デバイス名のあとにサブユニット番号が続く。文字数の総計は 8 文字以内でなければならない。これを論理デバイス名という。

サブユニットが存在しない場合は、物理デバイス名と論理デバイス名は同じになる。

物理デバイス名は、デバイスドライバの登録時に指定したデバイス名が使用される。

4.1.5 デバイスドライバ・インタフェース

TRON Safe Kernel のデバイス管理機能と、登録されているデバイスドライバとの間のソフトウェアのインタフェースを、デバイスドライバ・インタフェースと呼ぶ。

デバイスドライバ・インタフェースは、各デバイスドライバが提供する複数の C 言語の関数からなる。これらの関数はドライバインタフェース関数とも呼ばれる。ドライバインタフェース関数は、「**表 4-3 デバイスドライバ・インタフェース一覧**」に示す種類が存在し、そのすべてをデバイスドライバの登録時に設定する。つまり、デバイスドライバは、すべてのドライバ処理関数を用意しなくてはならない。

表 4-3 デバイスドライバ・インタフェース一覧

関数の形式	機能
ER initfn(T_INIT_DEV init_dev);	初期化関数（デバイスドライバの初期化を行う）
ER openfn(ID devid, UINT omode);	オープン関数（デバイスのオープン処理を行う）
ER closefn(ID devid, INT openno, UINT option);	クローズ関数(デバイスのクローズ処理を行う)
ER execfn(TS_DEVREQ *devreq, TMO tmout);	処理開始関数（デバイスに対しデータの入出力処理の要求を行う）
INT waitfn(INT reqno, INT openno, TS_DEVRTN *devrtn, TMO tmout);	完了待ち関数（デバイスに対するデータの入出力処理の要求の完了を待つ）
ER cancelfn (INT reqno);	要求無効関数（デバイスに対する入出力処理の要求を無効とする）
ER abortfn(INT openno);	中止処理関数（デバイスに対するすべての入出力処理を中止する）
INT eventfn(INT evttyp, void *evtinf, TMO tmout);	イベント関数（デバイスに対するイベントの処理を行う）

これらドライバ処理関数は、安全ドメインからのみ呼び出し可能なデバイスドライバ登録 I/F によって登録される。

デバイスドライバ登録 I/F は、登録するデバイスドライバのハードウェアアクセス I/F の呼び出し元ドメインを設定する I/F を備える。

4.1.6　入出力データ

デバイスドライバ管理機能の API を用いて、各デバイスのデータの入出力ができる。

データはデータ番号で識別され、固有データと属性データに大別される。

(1) 固有データ

デバイス固有のデータである。データ番号は 0 以上で、デバイス毎に定義される。

(2) 属性データ

デバイスドライバやデバイス自体の状態の各種情報や設定、特殊機能などを指定する。

データ番号は負の値である。データ番号のいくつかは共通で定義されているが、デバイス独自にも定義される。

4.1.7　デバイス事象通知

デバイスドライバは、各デバイスで発生した事象を、デバイス事象通知のメッセージとして特定のメッセージバッファ(事象通知用メッセージバッファ)へ送信する。事象通知用メッセージバッファの ID は、各デバイスに対する TDN_EVENT の属性データとして参照あるいは設定される。

デバイス登録の直後には、システムデフォルトの事象通知用メッセージバッファを使用する。なお、システムデフォルトの事象通知用メッセージバッファは、TRON Safe Kernel の起動時に生成され、そのサイズとメッセージ最大長は、システム管理情報の TS_DEVT_MBFSZ_S／TS_DEVT_MBFSZ_N と TS_DEVT_MBFMAX_S／TS_DEVT_MBFMAX_N により定義される。

デバイス事象通知は事象タイプで分類される。

事象タイプの大分類を以下に示す。

a. 基本事象通知（事象タイプ 0x0001～0x002F）

　　デバイスからの基本的な事象の通知

b. システム事象通知（事象タイプ 0x0030～0x007F）

　　電源制御などシステム全体に関係する事象の通知

c. 拡張情報付き事象通知（事象タイプ 0x0080～0x00FF）

　　拡張情報を持つデバイスからの事象の通知

d. ユーザ定義事象通知（事象タイプ 0x0100～0xFFFF）

　　ユーザが内容を任意に定義可能な事象の通知

デバイス事象通知で使用するメッセージの形式は以下のようになる。事象通知のメッセージの内容やサイズは、事象タイプ毎に決められる。

(1) デバイス事象通知の基本形式

```
typedef struct st_devevt {
        TDEvtTyp  evttyp;               /* 事象タイプ */
```

```
        /* 以下に事象タイプ別の情報が付加される */
} T_DEVEVT;
```

(2) デバイス ID 付きのデバイス事象通知の形式

```
typedef struct st_devevt_id {
        TDEvtTyp  evttyp;           /* 事象タイプ */
        ID        devid;            /* デバイス ID */
        /* 以下に事象タイプ別の情報が付加される */
} T_DEVEVT_ID;
```

(3) 拡張情報付きのデバイス事象通知の形式

```
typedef struct st_devevt_ex {
        TDEvtTyp  evttyp;           /* 事象タイプ */
        ID        devid;            /* デバイス ID */
        UB        exdat[16];        /* 拡張情報 */
        /* 以下に事象タイプ別の情報が付加される */
} T_DEVEVT_EX;
```

事象通知用メッセージバッファが一杯で事象通知を送信できない場合は、その事象が通知されないことで事象通知の受信側の動作に悪影響が出ないようにしなければならない。メッセージバッファが空くまで待ってから事象通知を行ってもよいが、その場合も原則として事象通知以外のデバイスドライバの処理が滞ってはならない。具体的な処理は、デバイスドライバ毎に実装定義とする。

なお、事象通知の受信側は、できる限りメッセージバッファが溢れることがないように処理しなければならない。

4.1.8 デバイスドライバ管理機能の API

デバイスドライバ管理機能の API を「表 4-4 デバイスドライバ管理機能 API 一覧」に示す（各 API の詳細は「8.3.2 デバイス管理機能の API」参照）。

表 4-4 デバイスドライバ管理機能 API 一覧

分類	安全 API 名称	通常 API 名称	初期登録 API 名称	説明
デバイス登録	—	—	TS_DEF_DEV	デバイスの登録
オープン・クローズ	ts_opn_dev	tn_opn_dev	—	デバイスのオープン
	ts_cls_dev	tn_cls_dev	—	デバイスのクローズ
データの非同期読込みおよび書込み	ts_rea_dev	tn_rea_dev	—	デバイスの読込み要求
	ts_wri_dev	tn_wri_dev	—	デバイスの書込み要求
	ts_wai_dev	tn_wai_dev	—	デバイスの要求完了待ち
	ts_can_dev	tn_can_dev	—	デバイスの要求の無効化
データの同期読込みおよび書込み	ts_srea_dev	tn_srea_dev	—	デバイスの同期読込み
	ts_swri_dev	tn_swri_dev	—	デバイスの同期書込み
デバイス要求イベント	ts_evt_dev	tn_evt_dev	—	デバイス要求イベントの送信
情報取得	ts_get_dev	tn_get_dev	—	デバイスのデバイス名取得
	ts_ref_dev	tn_ref_dev	—	デバイスのデバイス情報取得
	ts_oref_dev	tn_oref_dev	—	デバイスディスクリプタからの デバイス情報取得

4.2 割込み管理機能

4.2.1 割込み管理機能の概要

割込み管理機能は、割込みハンドラの登録などの操作を行う機能である。

割込みハンドラは、システムドメインに所属するプログラムの実行単位である。ハードウェアの割込みの発生時に起動される。

割込み管理機能は、システムドメインのソフトウェアからのみ使用できる。

そして、安全・通常ドメインのアプリケーションから割込みを禁止することはできない。

TRON Safe Kernel の割込み管理機能が対応する割込みは、プロセッサに対して周辺デバイスからのイベントの通知、またはそれに準じるものを想定している。

アドレス違反や未定義命令実行などプログラムの実行に関わるイベントや、なんらかの異常検出に関わるものは異常例外として処理する。

具体的な割込みの種別、要因は、ハードウェアに依存するので実装定義とする。

4.2.2 割込みハンドラの概要

各割込みハンドラは「表 4-5 割込みハンドラ管理情報」に示す情報を持つ。各情報は、API によって参照できる。

表 4-5 割込みハンドラ管理情報

名称	意味	動的変更※1
割込みハンドラ番号	割込みハンドラを識別する番号	不可
所属ドメイン	割込みハンドラが所属するドメインの ID 番号(システムドメインに固定)	不可
保護レベル	割込みハンドラの保護レベル(保護レベル 0 に固定)	不可
ハンドラ属性	割込みハンドラの属性	不可
実行優先度	割込みハンドラの実行優先度	不可
ハンドラ起動アドレス	割込みハンドラの実行プログラムの起動アドレス	不可
最大連続実行時間	割込みハンドラが連続実行できる時間の上限	不可

※1 生成後に API により動的な変更ができるか否か

割込みハンドラ番号は、割込みを識別するために静的に割り当てられた番号である。割込みハンドラ番号は、ハードウェアにおいて発生する割込みの要因に対し、一対一に対応して割り当てられる。よって、割込みハンドラ番号はハードウェアに依存するので、実装定義とする。

所属ドメインは割込みハンドラが所属するドメインの ID 番号である。割込みハンドラは必ずシステムドメインに所属するので、システムドメインの ID 番号に固定される。

保護レベルは、割込みハンドラの保護レベルである。割込みハンドラは必ずシステムドメインに所属するので、システムドメインの保護レベルに固定される。

ハンドラ属性は、割込みハンドラの属性である。詳細は「4.2.3 割込みハンドラの属性」を参照。

実行優先度は、割込みハンドラが実行されるシステム優先度である。詳細は「4.2.4 割込みハンドラの実行優先度」を参照。

ハンドラ起動アドレスは、割込みハンドラの起動時に実行されるプログラムの起動アドレスである。詳細は「4.2.5 割込みハンドラの実行プログラム」を参照。

最大連続実行時間は、割込みハンドラが連続実行可能な時間の上限を設定する。詳細は「4.2.7 割込みハンドラの最大連続実行時間」を参照。

4.2.3 割込みハンドラの属性

割込みハンドラの登録時に割込みハンドラ属性が指定される。「表 4-6 割込みハンドラ属性」に割込みハンドラ属性を示す。

表 4-6 割込みハンドラ属性

名称	属性	意味
TIA_MLTINT	多重割込み	多重割込みを許可する。 割込みハンドラ実行中に優先度の高い割込みを受け付ける。

4.2.4 割込みハンドラの実行優先度

割込みハンドラの実行優先度は、割込み要因に応じて特定のシステム優先度が割り当てられる。システム優先度の割り当ては、ハードウェアの仕様に依存し、静的に決められる。TRON Safe Kernel の機能としては、割込みハンドラの実行優先度の変更はできない。システム優先度については「2.5.2 実行優先度」を参照。

複数の割込みが同時に発生した場合、実行優先度の高い割込みハンドラが先に実行される。

割込みハンドラの実行中に、より優先度の高い割込みが発生した場合、多重割込み属性(TIA_MLTINT)が指定されていれば、より実行優先度の高い割込みの割込みハンドラが実行される。多重割込み属性が指定されていない場合、および優先度が同じか低い割込みの場合は、実行中の割込みハンドラが終了したあとに次の割込みが受け付けられる。

ただし、ハードウェアの仕様によっては、マスクが不可能な割込みなども存在する。よって多重割込み属性を指定してなくても、割込みハンドラの実行中に他の割込みハンドラが実行されることはありうる。このようなハードウェアに依存する仕様は実装定義とする。

4.2.5 割込みハンドラの実行プログラム

TRON Safe Kernel は、割込みを受け付けると、登録されている割込みハンドラの実行プログラムを呼び出す。

割込みハンドラの実行プログラムは、以下の形式の C 言語の関数とする。関数名は任意である。

```
void        inthdr( UINT dintno )
```

関数のパラメータとして、発生した割込みの割込みハンドラ番号が渡される。この割込みハンドラ番号は、実行される割込みハンドラ自身の割込みハンドラ番号と同一である。

割込み発生から割込みハンドラが呼び出されるまでの間は、原則としてすべての割込みが禁止される。

割込みハンドラが多重割込み受付属性の場合は、割込みハンドラを呼び出す際に、実行される割込みハンドラより優先度の高い割込みは許可状態となる。

割込みハンドラが終了したあとは、割込みが発生する前の状態に戻される。

4.2.6 割込みハンドラのスタック

割込みハンドラは、割込みハンドラ専用のスタックを使用する。割込みハンドラ専用のスタックのサイズは静的に決定され、スタックが使用するメモリ領域も静的に確保される。具体的な仕様は実装定義とする。

4.2.7 割込みハンドラの最大連続実行時間

各割込みハンドラには、割込みハンドラが連続して実行可能な時間の上限を設定する。

割込みハンドラの連続実行時間とは、TRON Safe Kernel より割込みハンドラが呼び出され、割込みハンドラが実行を終了するまでの時間である。ただし、多重割込みを許可している場合、割り込まれたより高い優先度の割込みハンドラの実行時間は除外される。同様に、その割込みハンドラより実行優先度の高いプログラムに割り込まれた時間は除外される。

割込みの連続実行時間が、設定されている最大連続実行時間を超えていないかの判定は、割込みハンドラの終了時に行われる。よって、割込みハンドラが最大連続実行時間を超えて実行されることは起こりうる。

割込みハンドラの終了時の判定で、最大連続実行時間を超えていた場合は、異常例外を発生する。

なお、割込みハンドラが実行中に規定の連続実行時間を超えた場合の処理は、最高優先度タイマを使用し、故障診断機能で検出・対応する。

4.2.8 割込み管理機能の API

割込み管理機能の API はシステムドメインのタスクのみから使用できる。

割込み管理機能の API を「表 4-7 割込み管理機能 API 一覧」に示す（各 API の詳細は「8.3.3 割込み管理機能の API」参照）。

表 4-7 割込み管理機能 API 一覧

分類	安全 API 名称	通常 API 名称	初期登録 API 名称	説明
ハンドラ登録	ts_def_int	—	TS_DEF_INT	割込みハンドラの登録

4.3 サブシステム管理機能

4.3.1 サブシステム管理機能の概要

サブシステム管理機能は、TRON Safe Kernel のシステムドメインで動作するサブシステムを管理するための機能である。

サブシステムとは、システムソフトウェアの機能を追加、拡張するために、独立した機能モジュールとして実装されるプログラムである。サブシステムは、他のソフトウェアへのインタフェースとして拡張 SVC を提供する。拡張 SVC は、TRON Safe Kernel の API を用いて呼び出すことができる。

サブシステム管理機能は、TRON Safe Kernel にサブシステムを登録するための機能と、登録済みのサブシステムを他のソフトウェアから利用するための機能が含まれる。

【補足説明】

アプリケーションから見ると、サブシステムは TRON Safe Kernel と同様にシステムソフトウェアの一部である。TRON Safe Kernel の機能を API の呼び出しで使用できるのと同様に、拡張 SVC を呼び出すことにより、サブシステムの機能を使用することができる。

4.3.2 サブシステムの構成

サブシステムは、以下の要素から構成される。

(1) 拡張 SVC ハンドラ

サブシステムへの操作は拡張 SVC の呼び出しで行われる。拡張 SVC ハンドラは、呼び出された拡張 SVC の処理を行う。

(2) サブシステム I/F 関数

TRON Safe Kernel のサブシステム管理機能と、サブシステムとのインタフェースとなる関数である。サブシステム管理機能から呼び出される。

(3) タスクおよび割込みハンドラ

サブシステムは、任意の数のタスクおよび割込みハンドラを作成し使用することができる。タスクおよび割込みハンドラは必須ではなく、必要に応じて作成、登録し使用する。

サブシステムの構成図を「図 4-1 サブシステムの構成図」に示す。

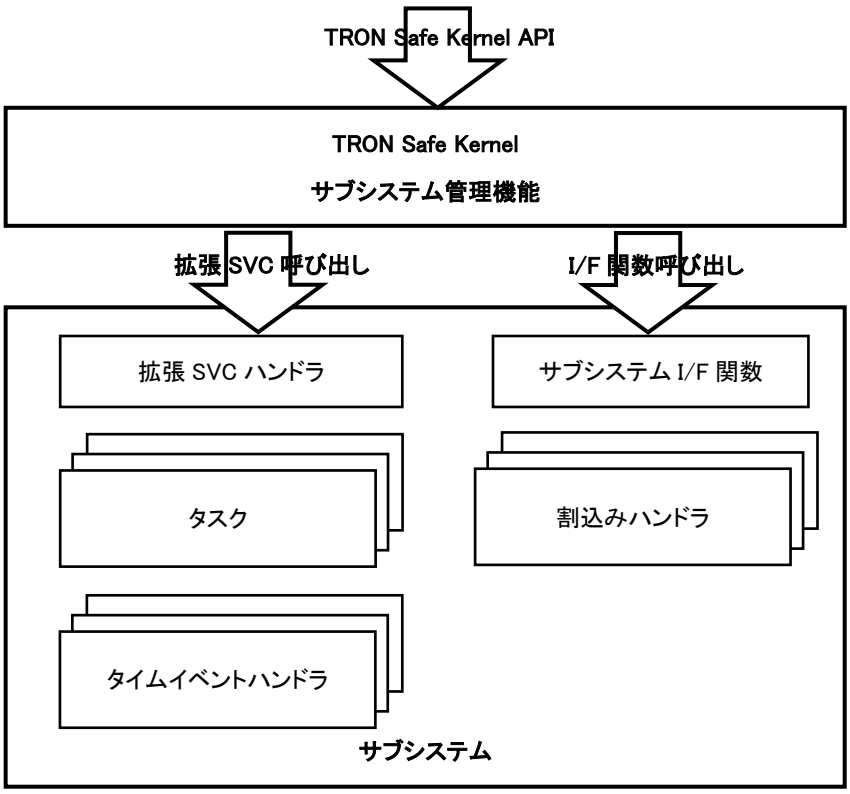

図 4-1 サブシステムの構成図

4.3.3 サブシステムの情報

各サブシステムは「表 4-8 サブシステム管理情報」に示す情報を持つ。各情報は、API によって参照できる。各情報はサブシステムの登録時に決められ変更することはできない。一部の情報は固定値であり、必ずしも実際のデータとして実装されるわけではない。

表 4-8 サブシステム管理情報

名称	意味	動的変更※1
サブシステム ID	サブシステムを識別する ID 番号	不可
所属ドメイン	サブシステムが所属するドメイン（システムドメインに固定）	不可
保護レベル	サブシステムの保護レベル（保護レベル0に固定）	不可
サブシステム属性	サブシステムの属性	不可
サブシステム優先度	サブシステムの I/F 関数を実行する優先度	不可
機能コード数	サブシステムが提供する機能コードの数	不可
安全 API 拡張 SVC 起動アドレス	安全 API 用の拡張 SVC ハンドラの実行プログラムの起動アドレス	不可
通常 API 拡張 SVC 起動アドレス	通常 API 用の拡張 SVC ハンドラの実行プログラムの起動アドレス	不可
サブシステム・ インタフェース	サブシステムに各種処理を要求するためのインタフェース （サブシステム I/F 関数から構成される）	不可

※1 生成後に API により動的な変更ができるか否か

サブシステムIDは、サブシステムを識別するために、サブシステムの登録時に静的に指定された番号である。サブシステムIDは1～255の値である。ただし、サブシステムIDは1～9がシステム用に予約されている。10～255がユーザ定義ソフトウェアのサブシステムで使用できる番号となる。ただし、使用可能なサブシステムIDの最大値は実装定義であり、255より小さい場合がある。実際に指定可能な最大のサブシステムIDはTS_MAX_SSYである。TS_MAX_SSYは、コンフィギュレーションによって静的に決定する。

所属ドメインはサブシステムが所属するドメインの ID 番号である。サブシステムは必ずシステムドメインに所属するので、システムドメインの ID 番号に固定される。

保護レベルは、サブシステムの保護レベルである。サブシステムは必ずシステムドメインに所属するので、システムドメインの保護レベルに固定される。

サブシステム属性は、サブシステムの属性である。詳細は「4.3.4 サブシステムの属性」を参照。

サブシステム優先度は、サブシステム I/F 関数を実行する際の優先度を示す。サブシステム優先度としては、1～TS_LST_SSYPRI の値を指定することができ、数の小さい方が高い優先度となる。TS_LST_SSYPRI は、16～255 の範囲の値であり、コンフィギュレーションによって静的に決定する。

機能コード数は、サブシステムが提供する機能コードの数を示す。機能コードは拡張 SVC ハンドラに渡される引数となる。詳細は「4.3.5 拡張 SVC ハンドラ」を参照。

拡張 SVC 起動アドレスは、拡張 SVC の呼び出し時に実行されるプログラムの起動アドレスである。詳細は「4.3.5 拡張 SVC ハンドラ」を参照。

サブシステム・インタフェースは、TRON Safe Kernel のサブシステム管理機能とサブシステムのインタフェースである。詳細は「4.3.6 サブシステム・インタフェース」を参照。

4.3.4 サブシステムの属性

サブシステムの登録時にサブシステム属性が指定される。「表 4-9 サブシステム属性」にサブシステム属性を示す。

表 4-9 サブシステム属性

名称	属性	意味
TSA_SSSY	安全サブシステム属性	安全 API を提供するサブシステム （安全ドメインおよびシステムドメインのソフトウェアから使用可能）
TSA_NSSY	通常サブシステム属性	通常 API を提供するサブシステム （通常ドメインのソフトウェアから使用可能）

サブシステム属性の組合せを以下に示す。

```
サブシステム属性 := TSA_SSSY || TSA_NSSY
```

4.3.5 拡張 SVC ハンドラ

TRON Safe Kernel は、API により拡張 SVC の呼び出しを受け付けると、該当するサブシステムの拡張 SVC ハンドラの実行プログラムを呼び出す。

拡張 SVC ハンドラの実行プログラムは、安全 API 用と通常 API 用の二つが存在する。

拡張 SVC ハンドラの実行プログラムは、以下の形式の C 言語の関数とする。関数名は任意である。

```
ER      svchdr( ID dmnid, FN fncd, T_SVCPARA *pk_svcpara)
```

関数の引数 dmnid は、ts_cal_svc/tn_cal_svc を呼び出したドメインの ID である。

関数の引数 fncd は機能コードである。機能コードは拡張 SVC が実行すべき機能を表す。機能コードは連続した正の値であり、最大値はサブシステム毎に決められる。

関数の引数 pk_svcpara は、拡張 SVC に渡されるパラメータである。以下の型に定義される。

```
typedef struct st_svcpara {
        UW      par[TS_MAX_ SVCPARA];
} T_SVCPARA;
```

TS_MAX_ SVCPARA は、4 以上の実装定義の数である。T_SVCPARA の各メンバーの内容は、サブシステム毎に決められる。

拡張 SVC ハンドラの実行プログラムの戻り値は、拡張 SVC の戻り値となる。

拡張 SVC ハンドラは、サブシステムの一部であるので、システムドメインに属し保護レベル 0 で実行される。

拡張 SVC ハンドラは、拡張 SVC を呼び出したプログラムのコンテキストを継承する。

タスクから呼び出された拡張 SVC ハンドラを実行中のシステム状態は準タスク部実行状態となる。準タスク部実行状態では、拡張 SVC を呼び出したタスクが「自タスク」として認識され、またスタックはタスクのシステムスタックを使用する。

タスク独立部から呼び出された拡張 SVC ハンドラを実行中のシステム状態はタスク独立部となる。

4.3.6 サブシステム・インタフェース

TRON Safe Kernel のサブシステム管理機能と、登録されているサブシステムとの間のソフトウェアのインタフェースを、サブシステム・インタフェースと呼ぶ。

サブシステム・インタフェースは、各サブシステムが提供する複数の C 言語の関数からなる。これらの関数はサブシステム I/F 関数と呼ばれる。

サブシステム I/F 関数は、TRON Safe Kernel のサブシステム管理機能から呼び出される。その際に呼び出される順番はサブシステム優先度で決められる。同一の優先度の場合は、サブシステム ID が小さいほど優先順位は高くなる。

サブシステム I/F 関数は「表 4-10 サブシステム・インタフェース一覧」に示す種類が存在し、そのすべてをサブシステムの登録時に設定する。つまり、サブシステムはすべてのサブシステム I/F 関数を用意しなくてはならない。

表 4-10 サブシステム・インタフェース一覧

関数の形式	機能
ER initfn(T_INIT_SSY init_ssy);	初期化関数（サブシステムの初期化を行う）
void startupfn(ID dmnid);	スタートアップ関数（サブシステムの内部資源の初期化を行う）
void cleanupfn(ID dmnid);	クリーンアップ関数（サブシステムの内部資源の解放を行う）
void breakfn(ID dmnid);	ブレーク関数（サブシステムの処理を中断する）
ER eventfn(ID dmnid, INT evttyp, INT info);	イベント処理関数（サブシステムに対するシステムからの要求の処理を行う）

4.3.7 サブシステム管理機能の API

サブシステム管理機能の API を「表 4-11 サブシステム管理機能 API 一覧」に示す（各 API の詳細は「8.3.1 サブシステム管理機能の API」参照）。

表 4-11 サブシステム管理機能 API 一覧

分類	安全 API 名称	通常 API 名称	初期登録 API 名称	説明
サブシステム登録	—	—	TS_DEF_SSY	サブシステムの登録
拡張 SVC	ts_cal_svc	tn_cal_svc	—	拡張 SVC の呼び出し

4.4 故障診断管理機能

4.4.1 故障診断管理機能の概要

故障診断管理機能は、TRON Safe Kernel のシステムドメインで動作する故障診断ハンドラを管理するための機能である。
故障診断ハンドラは、ハードウェアの故障検出のために定期的または定められたときに実行されるプログラムの実行単位である。故障診断ハンドラはシステムドメインに所属する。
故障診断管理機能は、TRON Safe Kernel に故障診断ハンドラを登録するための機能と、登録済みの故障診断ハンドラを他のソフトウェアから利用するための機能が含まれる。

故障診断を実行するにあたり、タスクや割込みハンドラが必要な場合は、それらはサブシステムとして実装する。サブシステムと故障診断ハンドラは、お互いに呼び合い連携して動作を行う。

4.4.2 故障診断ハンドラの概要

各故障診断ハンドラは「表 4-12 故障診断ハンドラ管理情報」に示す情報を持つ。各情報は、ハンドラの登録時に決められ変更することはできない。一部の情報は固定値であり、必ずしも実際のデータとして実装されるわけではない。

表 4-12 故障診断ハンドラ管理情報

名称	意味	API による変更※1
ハンドラ ID	故障診断ハンドラを識別する ID 番号	不可
所属ドメイン ID	故障診断ハンドラが所属するドメインの ID 番号(システムドメインに固定)	不可
保護レベル	故障診断ハンドラの保護レベル(保護レベル 0 に固定)	不可
ハンドラ属性	故障診断ハンドラの属性	不可
ハンドラ実行優先度	故障診断ハンドラの実行優先度	不可
ハンドラ状態	故障診断ハンドラの状態	不可
起動時間間隔	故障診断ハンドラを起動する間隔	不可
ハンドラ起動アドレス	故障診断ハンドラの実行プログラムの起動アドレス	不可
最大連続実行時間	故障診断ハンドラが連続実行できる時間の上限	不可

※1 生成後に API により動的な変更ができるか否か

ハンドラ ID は、故障診断ハンドラを識別するために、故障診断ハンドラの登録時に静的に指定された番号である。
所属ドメインは故障診断ハンドラが所属するドメインの ID 番号である。故障診断ハンドラは必ずシステムドメインに所属するので、システムドメインの ID 番号に固定される。
保護レベルは、故障診断ハンドラの保護レベルである。故障診断ハンドラは必ずシステムドメインに所属するので、システムドメインの保護レベルに固定される。
ハンドラ属性は、故障診断ハンドラの属性である。詳細は「4.4.3 故障診断ハンドラの属性」を参照。
ハンドラ実行優先度は、故障診断ハンドラの実行優先度である。実行優先度は、タスク優先度の値を設定できる。ハンドラ状態は、故障診断ハンドラの実行状態である。詳細は「4.4.4 故障診断ハンドラの状態」を参照。
起動時間間隔は、故障診断ハンドラが起動される時間の間隔である。ハンドラ起動アドレスは、故障診断ハンドラの実行プログラムの起動アドレスである。詳細は「4.4.5 故障診断ハンドラの起動」を参照。

4.4.3 故障診断ハンドラの属性

故障診断ハンドラの登録時にハンドラ属性が指定される。「表 4-13 故障診断ハンドラ属性」にハンドラ属性を示す。

表 4-13 故障診断ハンドラ属性

名称	属性	意味
TFA_START	システム起動時実行	システムの起動時に故障診断ハンドラを実行する
TFA_CYC	周期実行	故障診断ハンドラを周期実行する
TFA_API	API 操作	安全 API による起動、停止の操作が可能

故障診断ハンドラ属性は次の組合せが指定できる。

故障診断ハンドラ属性 := [TFA_START] | [TFA_CYC] | [TFA_API]

4.4.4 故障診断ハンドラの状態

故障診断ハンドラの状態は以下に分類される。

(1) 実行状態

故障診断ハンドラのプログラムが実行されている状態である。

(2) 起動待ち状態

故障診断ハンドラが動作しており、起動する時刻を待っている状態である。

(3) 停止状態

故障診断ハンドラが動作を停止している状態である。

故障診断ハンドラは、TRON Safe Kernel の初期化処理で登録され、起動待ち状態となる。
起動待ち状態の故障診断ハンドラは、起動時刻(タイミング)に達し、かつより優先順位の高いプログラムの実行単位が実行されていなければ、実行状態に遷移し自らのプログラムを実行する。
実行状態の故障診断ハンドラは処理を終えたら起動待ち状態に遷移し、次の起動時刻を待つ。
TFA_API 属性の故障診断ハンドラは、安全 API を用いて停止状態、また再び起動待ち状態に遷移させることができる。
故障診断ハンドラの状態遷移を「図 4-2 故障診断ハンドラの状態遷移」に示す。

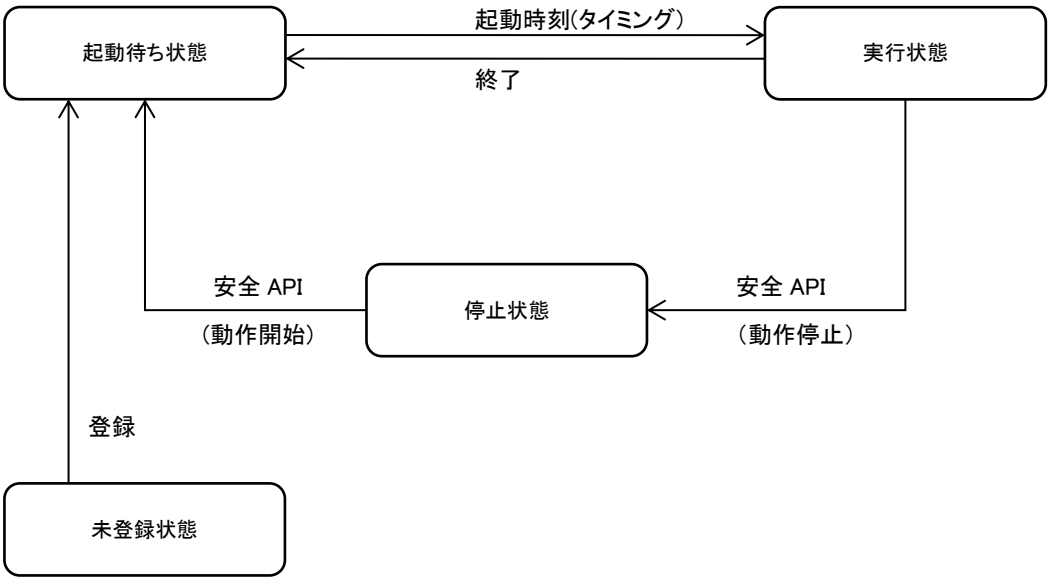

図 4-2 故障診断ハンドラの状態遷移

4.4.5 故障診断ハンドラの起動

故障診断ハンドラは属性に応じて以下のタイミングで起動される。

- TFA_START 属性が指定されている場合は、TRON Safe Kernel の初期化処理が終了し、システムドメインが実行を開始した際に起動される。
- TFA_CYC 属性が指定され、かつ起動待ち状態の場合は、TRON Safe Kernel がカーネル実行状態の間、設定されている起動時間間隔に従い周期的に起動される。
- TFA_API 属性が指定されている場合は、安全 API により起動できる。

起動された故障診断ハンドラは、設定されている実行優先度に従って実行される。
実行優先度が同じプログラムが存在した場合は以下の優先順位となる。

故障診断ハンドラ（優先度高） ＞ タイムイベントハンドラ ＞ タスク（優先度低）

同一優先度の故障診断ハンドラでは、先に起動された方が優先される。

【補足説明】
故障診断ハンドラはシステムドメインにおいて最も高い実行優先度で実行することができる。このとき、故障診断ハンドラの実行中にタスクやタイムイベントハンドラにディスパッチにすることはない。ただし、故障診断の内容によっては、タスクやタイムイベントハンドラより低い優先度で実行したい場合もありうる。その場合は、故障診断ハンドラの登録時に、実行優先度をタスクやタイムイベントハンドラより低く設定することにより、それを実現できる。

4.4.6 故障診断ハンドラの実行プログラム

故障診断ハンドラの実行プログラムは、以下の形式の C 言語の関数とする。関数名は任意である。

```
void        fdhdr( void)
```

4.4.7 故障診断ハンドラの時間保護

故障診断ハンドラの実行状態の時間が連続実行時間を超えた場合、その時点で TRON Safe Kernel は異常例外を発生する。よって、故障診断ハンドラの実行は中断され、異常例外ハンドラが実行される。

故障診断ハンドラの実行中に、他の実行単位の処理が割り込んで実行された場合は、その処理の実行時間は除外される。割込みハンドラや他の故障診断ハンドラなどの実行がこれに該当する。この場合は、連続実行時間は保持されるので、元の故障診断ハンドラに実行が戻ると、継続して連続実行時間が計測される。

故障診断ハンドラは実行中に割込みをマスクすることがある。割込みがマスクされている期間は、連続実行時間を超えてもその時点で検出できない場合がある。その場合は、割込みマスクが解除された時点で異常例外が発生する。

4.4.8 故障診断ハンドラから使用可能な API

故障診断ハンドラから使用可能な API は限定される（タイムイベントハンドラと同一）。また、API が使用可能なのは、カーネル実行状態から故障診断ハンドラが実行された場合のみである。その他の状態から故障診断ハンドラが実行された場合、API を発行するとエラーE_CTX を返す。

4.4.9 故障診断ハンドラのスタック

故障診断ハンドラは、故障診断ハンドラ専用のスタックを使用する。故障診断ハンドラ専用のスタックのサイズは静的に決定され、スタックが使用するメモリ領域も静的に確保される。具体的な仕様は実装定義とする。

4.4.10 故障診断管理機能の API

故障診断管理機能の API を以下に示す（各 API の詳細は「8.4.4 故障診断管理機能の API」参照）。

表 4-14 故障診断管理機能 API 一覧

分類	安全 API 名称	通常 API 名称	初期登録 API 名称	説明
ハンドラ登録	—	—	TS_DEF_FDH	故障診断ハンドラの登録
ハンドラ操作	ts_sta_fdh	—	—	故障診断ハンドラの動作開始
	ts_stp_fdh	—	—	故障診断ハンドラの動作停止
	ts_act_fdh	—	—	故障診断ハンドラの起動

5. TRON Safe Kernel 共通規定

5.1 TRON Safe Kernel 共通データ型

5.1.1 汎用的なデータ型

TRON Safe Kernel では汎用的なデータの型を以下のように定義する。

```
typedef signed char         B;      /* 符号付き  8 ビット整数 */
typedef signed short        H;      /* 符号付き 16 ビット整数 */
typedef signed long         W;      /* 符号付き 32 ビット整数 */
typedef signed long long    D;      /* 符号付き 64 ビット整数 */
typedef unsigned char       UB;     /* 符号無し  8 ビット整数 */
typedef unsigned short      UH;     /* 符号無し 16 ビット整数 */
typedef unsigned long       UW;     /* 符号無し 32 ビット整数 */
typedef unsigned long long  UD;     /* 符号無し 64 ビット整数 */

typedef char                VB;     /* 型が一定しない  8 ビットのデータ */
typedef short               VH;     /* 型が一定しない 16 ビットのデータ */
typedef long                VW;     /* 型が一定しない 32 ビットのデータ */
typedef long long           VD;     /* 型が一定しない 64 ビットのデータ */
typedef void                *VP;    /* 型が一定しないデータへのポインタ */

typedef volatile B          _B;     /* volatile 宣言付 */
typedef volatile H          _H;
typedef volatile W          _W;
typedef volatile D          _D;
typedef volatile UB         _UB;
typedef volatile UH         _UH;
typedef volatile UW         _UW;
typedef volatile UD         _UD;

typedef signed int          INT;    /* プロセッサのビット幅の符号付き整数、32 ビット以上 */
typedef unsigned int        UINT;   /* プロセッサのビット幅の符号無し整数、32 ビット以上 */

typedef INT                 ID;     /* ID 一般 */
typedef W                   MSEC;   /* 時間一般(ミリ秒) */

typedef void                (*FP)();            /* 関数アドレス一般 */
typedef INT                 (*FUNCP)();         /* 関数アドレス一般 */

typedef UINT                BOOL;               /* 真偽値(ブール値) */
```

VB, VH, VW, VD と B, H, W, D との違いは、前者はビット数のみが分かっており、データ型の中身が分からないものを表すのに対して、後者は整数を表すことがはっきりしているという点である。
プロセッサのビット幅は 32 ビット以上に限定する。従って、INT, UINT は必ず 32 ビット以上の幅がある。
BOOL は、TRUE を 1 と定義するが、0 以外はすべて真である。従って、ブール値 == TRUE の様な判定を行ってはいけない。ブール値 != FALSE の様に判定する。

API のパラメータは、明らかに負の数にならないパラメータも、原則として符号付き整数(INT)のデータ型となっている。これは、整数はできるだけ符号付きの数として扱うという TRON 全般のルールに基づいたものである。また、タイムアウト(TMO tmout)のパラメータでは、これが符号付きの整数であることを利用し、TMO_FEVR(= -1)を特殊な意味に使っている。符号無しのデータ型を持つパラメータは、ビットパターンとして扱われるもの(オブジェクト属性やイベントフラグなど)である。

5.1.2 意味が定義されているデータ型

TRON Safe Kernelでは API のパラメータの意味を明確にするため、出現頻度の高いデータ型や特殊な意味を持つデータ型を以下のように定義する。

```
typedef INT         FN;             /* 機能コード */
typedef UW          ATR;            /* 属性 */
typedef INT         ER;             /* エラーコード */
typedef INT         PRI;            /* 優先度 */
typedef W           TMO;            /* ミリ秒単位のタイムアウト指定 */
typedef UW          RELTIM;         /* ミリ秒単位の相対時間 */

typedef struct st_systim {                  /* ミリ秒単位のシステム時刻 */
        W       hi;                         /* 上位 32 ビット */
        UW      lo;                         /* 下位 32 ビット */
} SYSTIM;
```

5.2 TRON Safe Kernel 共通定数

5.2.1 一般的な定数

TRON Safe Kernel では一般的な定数を以下のように定義する。

```
#define TRUE        (1)         /* 真(ブール値) */
#define FALSE       (0)         /* 偽(ブール値) */

#define NULL        (0)         /* 無効ポインタ */
#define TA_NULL     (0)         /* 特別な属性を指定しない */
#define TMO_POL     (0)         /* ポーリング */
#define TMO_FEVR    (-1)        /* 永久待ち */
```

5.2.2 シンボル定義

TRON Safe Kernel では以下の一般的なシンボルが定義されている。

```
#define LOCAL               static          /* ローカルシンボル定義 */
#define EXPORT                              /* グローバルシンボル定義 */
#define IMPORT              extern          /* グローバルシンボル参照 */
```

5.3 エラーコード

5.3.1 エラーコードの概要

エラーコードは、TRON Safe Kernel の処理において発生したエラーを表現する値であり、主に API の戻り値に使用される。

エラーコードは、メインエラーコードとサブエラーコードで構成される。エラーコードの下位 16 ビットがサブエラーコード、残りの上位ビットがメインエラーコードとなる。メインエラーコードは、検出の必要性や発生状況などにより、エラークラスに分類される。

TRON Safe Kernel ではサブエラーコードは使用せず、常に 0 となる。

5.3.2 エラーコードの一覧

TRON Safe Kernel のエラーコードとメインエラーコードを以下に示す。

(1) 正常終了のエラークラス

エラーコード	メインエラーコード(値)	意味
E_OK	0	正常終了

(2) 内部エラークラス

エラーコード	メインエラーコード(値)	意味
E_SYS	-5	システムエラー システムソフトウェアの内部の動作に起因するエラー。
E_NOCOP	-6	コプロセッサ使用不可エラー 現在動作中のハードウェアに指定のコプロセッサが搭載されていない。または、コプロセッサの動作異常を検出した。

(3) 未サポートエラークラス

エラーコード	メインエラーコード(値)	意味
E_NOSPT	-9	未サポートエラー 指定された機能がサポートされていない。
E_RSFN	-11	予約機能コード番号エラー 指定した機能コードが未定義である。未定義の拡張 SVC ハンドラを実行しようとした場合にも、このエラーが発生する。
E_RSATR	-12	予約属性エラー 対象カーネルオブジェクトの属性として指定することができない属性が指定された。または、相反する属性が指定された。

(4) パラメータエラークラス

エラーコード	メインエラーコード(値)	意味
E_PAR	-17	パラメータエラー API のパラメータの値が不正。具体的には以下の場合に発生する。 ・パラメータとして指定可能な範囲外の値が指定されていた場合 ・ポインタの値が不正の場合
E_ID	-18	不正 ID 番号エラー 指定された ID 番号が不正あるいは利用できない。

(5) 呼び出しコンテキストエラークラス

エラーコード	メインエラーコード(値)	意味
E_CTX	-25	コンテキストエラー 対象 API を発行できるコンテキストではない。 例えば、タスク独立部やタイムイベントハンドラの実行中、またはディスパッチ禁止状態において、自タスクを待ち状態にするシステムコールを発行した場合に発生する。
E_MACV	-26	メモリアクセス違反エラー 対象のメモリ領域に対するアクセス権限がない。または対象のメモリ領域が存在しない。
E_OACV	-27	オブジェクトアクセス違反エラー 対象のカーネルオブジェクトに対してアクセスする権限がない。
E_ILUSE	-28	システムコール不正使用 ミューテックスの上限優先度違反、または多重ロック

(6) 資源不足エラークラス

エラーコード	メインエラーコード(値)	意味
E_NOMEM	-33	メモリ不足エラー 必要なメモリが確保できなかった。
E_LIMIT	-34	システム上限超過エラー カーネルオブジェクト数の上限を超えてオブジェクトを生成しようとした。

(7) オブジェクト状態エラー

エラーコード	メインエラーコード(値)	意味
E_OBJ	-41	オブジェクト状態エラー 対象カーネルオブジェクトの状態が API の要求を受けられる状態でない。
E_NOEXS	-42	オブジェクト未登録エラー 指定されたカーネルオブジェクトが存在しない(生成されていない)。 ドメイン管理機能の API に共通事項として、全てのドメインはシステム起動時に静的に生成され、未登録状態も無いため、dmnid のドメインが存在しないことによる E_NOEXS は発生しない。(8.2.9 ドメイン管理機能の API 参照)
E_QOVR	-43	キューイングオーバフローエラー キューイング数やネスト回数などの上限を超えて要求がなされた。

(8) 待ち解除エラークラス

エラーコード	メインエラーコード(値)	意味
E_RLWAI	-49	待ち状態解除エラー 自タスクの待ち状態が強制的に解除された。
E_TMOUT	-50	タイムアウトエラー API の処理が指定した時間内に完了しなかった。
E_DLT	-51	待ちオブジェクトの削除エラー 待ち状態の対象となっていたカーネルオブジェクトが削除された。 例えば、タスクがセマフォの資源獲得待ち状態になっていて、対象セマフォが削除された場合、そのタスクは資源獲得待ち状態は解除されて本エラーが返される。
E_DISWAI	-52	待ち状態禁止エラー 自タスクが指定した待ち要因に対して待ち禁止状態である。

(9) デバイスエラークラス

エラーコード	メインエラーコード(値)	意味
E_IO	-57	デバイスエラー デバイスドライバの処理内でエラーが発生した。デバイスドライバの操作を行う API の戻り値に使用される。エラーの内容はサブコードに返される。
E_BUSY	-65	ビジー状態エラー デバイスドライバが使用中で操作できない。排他オープンされているデバイスドライバをオープンしようとした場合にも発生する。
E_ABORT	-66	中断エラー デバイスの処理完了待ちが途中で中断された。

(10) ドメインエラークラス

エラーコード	メインエラーコード(値)	意味
E_DOMAIN	-70	ドメインエラー API で指定した操作が、対象カーネルオブジェクトの属するドメインでは禁止されている。
E_ONAME	-71	オブジェクト名エラー 指定されたオブジェクト名が既にドメイン内で使用されている。
E_DACV	-72	オブジェクトアクセスエラー 他ドメインのカーネルオブジェクトに対して要求した操作が、ドメインによる保護で禁止されている。

5.4 TRON Safe Kernel API 仕様

5.4.1 API のインタフェース形式

TRON Safe Kernel では、C 言語を標準的なプログラミング言語として使用することを前提とし、C 言語から API を実行する場合のインタフェースを標準化する。

API インタフェースは、次の共通原則を設ける。

- API はすべて C 言語の関数として定義される。
- 関数としての戻り値は、0 または正の値が正常終了、負の値がエラーコードとなる。

API の実装は、実装定義とする。

5.4.2 API のパラメータ

API のパラメータは、API を定義した C 言語の関数の引数となる。
パラメータのいくつかは、パケット形式になっている。パケットは、C 言語の構造体として定義される。
パケットやポインタを使ったパラメータは、ポインタ参照先のパラメータを書き換えないことを明示するために、CONST 修飾子を付ける。CONST は、C 言語の const 修飾子を意図したものである。
なお、API に渡されるパラメータは全て、API 呼び出し時に、許可された値の範囲内であるかがチェックされる。

5.4.3 API のタイムアウト

TRON Safe Kernel では、待ち状態に入る可能性のある API には、タイムアウトの機能を持たせる。
タイムアウトは、指定された時間が経過しても処理が完了しない場合に、処理をキャンセルして API からリターンする機能である。この時、API はタイムアウトエラー(E_TMOUT)を返す。
API がタイムアウトした場合には、API を呼び出したことで、システムの状態は変化していないのが原則である。ただし、API の機能上、処理のキャンセル時に元の状態に戻せない場合は、システムコールの説明でその旨を明示する。
タイムアウト時間(TMO 型)としては、正の値のみを指定することができる。タイムアウト時間の基準時間(時間の単位)はシステム時刻の基準時間(= 1 ミリ秒)と同じである。タイムアウト時間が指定された場合、タイムアウトの処理は、API が呼び出されてから、指定された以上の時間が経過した後に行われることを保証する。
タイムアウト時間として TMO_POL(= 0)を指定した場合は、API の中で待ち状態に入るべき状況になっても、待ち状態には入らない。タイムアウト時間を TMO_POL とした API 呼び出しを、ポーリングと呼ぶ。ポーリングを行う API では、待ち状態に入る可能性がない。
タイムアウト時間として TMO_FEVR (= −1)を指定した場合は、タイムアウト値として無限大の時間を指定したことを示し、タイムアウトせずに待ち解除条件が成立するまで待ち続ける。ただし、安全 API では永久待ちを指定することは許されない。
各 API の説明で、「待ち状態に入る」または「待ち状態に移行させる」と記述されている場合でも、タイムアウト時間を指定した場合には、指定時間経過後に待ち状態が解除され、メインエラーコードを E_TMOUT として API から戻る。また、ポーリングの場合には、待ち状態に入らずにメインエラーコードを E_TMOUT として API から戻る。

5.4.4 絶対時間と相対時間

API はパラメータとして、絶対時間または相対時間を指定する場合がある。

絶対時間は、TRON Safe Kernel が管理するシステム時刻である。

絶対時間は、1 ミリ秒単位の 64 ビットの符号付整数であり、SYSTIM 型として定義される。

SYSTIM 型（システム時刻）

```
typedef    struct st_systim {
           W       hi;          /* 上位 32 ビット */
           UW      lo;          /* 下位 32 ビット */
} SYSTIM;
```

相対時間は、API を呼び出した時刻などからの相対値である。

相対時間は、1 ミリ秒単位の 32 ビットの符号なし整数であり、RELTIM 型として定義される。

RELTIM 型(相対時間)

```
typedef    UW      RELTIM;
```

API のタイムアウト時間は、相対時間の一種であるが、TMO 型として独立して定義される。

TMO 型（タイムアウト時間）

```
typedef    W       TMO;
```

相対時間およびタイムアウト時間は、指定された時間以上が必ず経過することを保証する。

TRON Safe Kernel には、現在のシステム時刻を変更する機能が用意されているが、この機能を用いてシステム時刻を変更した場合でも、相対時間を用いて指定された処理を実行する実世界の時刻(これを実時刻と呼ぶ)は変化しない。同様に、システム時刻を変更しても、タイムアウトが発生する実時刻は変化しない。

5.4.5 安全 API の共通仕様

安全 API の共通仕様を以下に示す。

- 安全 API は安全ソフトウェア（システムドメインおよび安全ドメインに属するソフトウェア）からのみ呼び出しが可能である。
- 安全 API の名称は、プリフィックスを"ts_"とする。
- 通常ドメインのカーネルオブジェクトを操作対象とする場合は、タイムアウト時間はポーリングのみ指定可能である。ポーリング以外を指定した場合は、ドメインエラー(E_DOMAIN)が返る。
- 通常ドメインから安全 APIを呼び出した場合は、異常例外を発生する。また、異常例外から復帰する場合は、呼び出した安全 API からはドメインエラー(E_DOMAIN)が返る。
- タイムアウト時間に永久待ちを指定した場合は、ドメインエラー(E_DOMAIN)が返る。

5.4.6 通常 API の共通仕様

通常 API の共通仕様を以下に示す。

- 通常 API は通常ソフトウェア（通常ドメインに属するソフトウェア）からのみ呼び出しが可能である。
- 通常 API の名称は、プリフィックスを"tn_"とする。
- 通常 API は他のドメインのカーネルオブジェクトを操作できない。
- 安全・システムドメインから通常 API を呼び出した場合は、異常例外を発生する。また、異常例外から復帰する場合は、呼び出した通常 API からはドメインエラー(E_DOMAIN)が返る。

5.4.7 API の個別仕様

TRON Safe Kernel の個々の API の仕様は、「第 8 章 API 仕様」にて規定する。

6. 安全機能

6.1 安全機能の概要

TRON Safe Kernel は、機能安全規格である IEC61508 の SIL3 の要求を満たすために以下の安全機能を実現する。

- 異常動作の検出
  安全ソフトウェアのアプリケーションおよびシステムソフトウェアの異常動作の防止のため、異常例外機能(「6.2 異常例外機能」参照)および故障診断機能(「6.3 故障診断機能」参照)をもつ。
- ドメインの分割
  安全ソフトウェアのアプリケーションおよびシステムソフトウェアの実行が、通常ソフトウェアのアプリケーションの実行により阻害されることを防止するため、各ソフトウェアをドメインにて分割する。それぞれのドメインは、空間的および時間的に分離される(「6.4 ドメインの分割」参照)。
- システムソフトウェアの設定の保護
  システムソフトウェアの設定は、安全ソフトウェアからのみ許される(「6.5 システムソフトウェアの設定の保護」参照)。
- 安全状態への移行
  異常動作を検出した際にシステムを安全状態に移行される機能を提供する(「6.6 安全状態への移行」参照)。
- 応答性の保証
  システムの割込み応答性を保証するために機能を提供する(「6.7 応答性の保証」参照)。

6.2 異常例外機能

6.2.1 異常例外機能の概要

異常例外機能は、異常を検出しそれを防止するための機能である。検出された異常を異常例外と呼ぶ。異常例外に対応する処理は異常例外ハンドラと呼ぶ。

異常例外機能は、異常例外を検出し、それに対応する異常例外ハンドラを実行する機能である。

異常例外は、ハードウェア故障やソフトウェアの実行が困難となるイベントであり、異常例外番号にて識別される。

異常例外は以下の要因により発生する。

(1) ハードウェアが検出する異常例外

ハードウェアの機能により検出され発生する異常例外である。アドレス違反や未定義命令実行、MMU や MPU によるメモリ保護違反などがある。

(2) ソフトウェアが検出する異常例外

ソフトウェアの機能により検出され発生する異常例外である。以下の種類がある。

(2-1) TRON Safe Kernel による異常例外

TRON Safe Kernel の動作中に検出された異常例外である。連続実行時間の超過、システムコールの実行時のエラーなどがある。

(2-2) 故障診断機能による異常例外

故障診断機能により検出された異常例外である(故障診断機能については「6.3 故障診断機能」を参照)。

(2-3) ユーザ定義の異常例外

ユーザ定義システムソフトウェア、安全ドメインのアプリケーションにより検出された異常例外である。安全 API を用いて発生

することができる。

なお、異常例外は、ハードウェア故障やソフトウェアの実行が困難となる異常のみを対象としている。それより軽度な異常が検出された場合は、異常例外を発生せずに API の実行を終了し、API の戻り値としてエラーコードを通知する。軽度な異常には以下が含まれる。

- 許可されていない API の呼び出し
- 許可された引数範囲外の API の呼び出し

6.2.2 異常例外の種類

異常例外は、異常例外番号にて識別される。異常例外番号は以下のように分類される。

(1) 標準異常例外（異常例外番号 0x00000000～0x00007FFF）
TRON Safe Kernel で一般的に起こりうる異常例外。

(2) 故障診断異常例外(異常例外番号 0x00008000～0x0000FFFF)
故障診断機能にて発生する異常例外。実装定義とする。

(3) ユーザ定義異常例外(異常例外番号 0x00010000～0x0001FFFF)
ユーザ定義システムソフトウェア、安全ドメインのアプリケーションから安全 API を用いて発生する異常例外。ユーザが任意に定めることができる。TRON Safe Kernel の仕様では規定しない。

異常例外の種別毎に最大の異常例外番号が定められる。標準異常例外の最大番号は、本章で規定されている最大の標準異常例外の番号である。故障診断異常例外とユーザ定義異常例外の最大の番号は、コンフィギュレーションで規定される。

TRON Safe Kernel で定義された標準異常例外を「表 6-1 標準異常例外一覧」に示す。

表 6-1 標準異常例外一覧

分類	種別	名称	値	意味
未定義	未定義異常例外	AEXP_UND_AE	0x00000000	異常例外ハンドラが定義されていない異常例外が発生した
命令実行	未定義命令	AEXP_UND_CD	0x00000001	未定義の命令が実行された
	特権命令	AEXP_PRI_INS	0x00000002	実行できない動作モードで特権命令が実行された
	ゼロ除算	AEXP_DIVZ	0x00000003	ゼロ除算が実行された
	実行領域違反	AEXP_NOEXE	0x00000004	実行不可のメモリ領域を実行しようとした
	実行アライメント違反	AEXP_CALI	0x00000005	実行コードのアライメントが不正
FPU	浮動小数点例外	AEXP_FPU	0x00000006	FPU に係わる例外が発生した
メモリ保護	メモリ保護違反	AEXP_MEM	0x00000007	保護されたメモリ領域をアクセスした
	データアライメント違反	AEXP_DALI	0x00000008	データアクセスのアライメントが不正
割込み	未定義割込み	AEXP_UND_INT	0x00000009	割込みハンドラが定義されていない割込みが発生した
	ノンマスカブル割込み	AEXP_NMI	0x0000000A	ノンマスカブル割込みが発生した
時間保護	タスク連続実行違反	AEXP_TOUT_TSK	0x00000010	タスクが連続実行時間を超えた
	タイムイベントハンドラ連続実行違反	AEXP_TOUT_TEH	0x00000011	タイムイベントハンドラが連続実行時間を超えた
	割込みハンドラ連続実行違反	AEXP_TOUT_INH	0x00000012	割込みハンドラが連続実行時間を超えた
	故障診断ハンドラ連続実行違反	AEXP_TOUT_FDH	0x00000013	故障診断ハンドラが連続実行時間を超えた
	ドメイン連続実行違反	AEXP_TOUT_DMN	0x00000014	ドメインが連続実行時間を超えた
	ディスパッチ禁止連続時間違反	AEXP_TOUT_DSP	0x00000015	ディスパッチ禁止時間がタイムアウト時間を超えた
OS機能	コンフィギュレーション異常	AEXP_CFG	0x00000020	コンフィギュレーション情報の値が不正
	コントロールブロック異常	AEXP_CB	0x00000021	コントロールブロックのデータが不正
	不正 API 呼び出し	AEXP_ILAPI	0x00000022	機能番号やパラメータ数が不正な API の呼び出し
	タスク不正終了	AEXP_ILEXT	0x00000023	タスクが終了 API を呼ばずに終了した
	ドメイン／タスク終了エラー	AEXP_EEXIT	0x00000024	ドメイン／タスクの終了 API 実行中にエラーが発生
	メッセ―ジバッファ伝送エラー	AEXP_ERR_MBF	0x00000025	メッセージバッファの通信データが不正
	デバイスドライバ異常	AEXP_DEV	0x00000026	デバイスドライバで異常が発生
	サブシステム異常	AEXP_SSY	0x00000027	サブシステムで異常が発生

	初期登録 API のエラー	AEXP_INIDEF_API	0x00000028	初期登録 API でエラーが発生
	ドメインの開始エラー	AEXP_DMN	0x00000029	ドメインの開始でエラーが発生

なお異常例外発生時は、異常例外ハンドラ(「6.2.6 異常例外ハンドラの実行プログラム」)の引数パラメータとして、異常例外番号(axepno)、異常例外が発生したときのカーネル状態(tskstat)、異常例外毎に定められた情報(aexppar)の 3 パラメータが渡される。異常例外毎に定められた情報(aexppar)は、異常例外番号(axepno)別に、以下に示す構造体と定義値が使用される(一部内容については実装定義となる)。

(1) 未定義異常例外(AEXP_UND_AE) 情報

```
typedef struct st_aexp_und_ae {
    UINT    aexpno;                 /* 異常例外番号 */
} T_AEXP_UND_AE;
```

(2) 未定義命令(AEXP_UND_CD) 情報

```
typedef struct st_aexp_und_cd {
    /* 実装定義 */
} T_AEXP_UND_CD;
```

(3) 特権命令(AEXP_PRI_INS) 情報

```
typedef struct st_aexp_pri_ins {
    /* 実装定義 */
} T_AEXP_PRI_INS;
```

(4) ゼロ除算(AEXP_DIVZ) 情報

```
typedef struct st_aexp_divz {
    /* 実装定義 */
} T_AEXP_DIVZ;
```

(5) 実行領域違反(AEXP_NOEXE) 情報

```
typedef struct st_aexp_noexe {
    /* 実装定義 */
```

```
} T_AEXP_NOEXE;
```

(6) 実行アライメント違反(AEXP_CALI) 情報

```
typedef struct st_aexp_noexe {
    /* 実装定義 */
} T_AEXP_CALI;
```

(7) 浮動小数点例外(AEXP_FPU) 情報

```
typedef struct st_aexp_fpu {
    /* 実装定義 */
} T_AEXP_FPU;
```

(8) メモリ保護違反(AEXP_MEM) 情報

```
typedef struct st_aexp_mem {
    /*実装定義*/
} T_AEXP_MEM;
```

(9) データアライメント違反(AEXP_DALI) 情報

```
typedef struct st_aexp_dali {
    /*実装定義*/
} T_AEXP_DALI;
```

(10) 未定義割込み(AEXP_UND_INT) 情報

```
typedef struct st_aexp_und_int {
    UINT    intno;                  /* 割込み番号 */
} T_AEXP_UND_INT;
```

(11) ノンマスカブル割込み(AEXP_NMI) 情報

```
typedef struct st_aexp_nmi {
```

```
    /* 実装定義 */
} T_AEXP_NMI;
```

(12) タスク連続実行違反(AEXP_TOUT_TSK) 情報

```
typedef struct st_aexp_tout_tsk {
    ID    dmnid;                    /* 異常例外発生ドメイン ID */
    ID    tskid;                    /* 異常例外発生タスク ID */
} T_AEXP_TOUT_TSK;
```

(13) タイムイベントハンドラ連続実行違反(AEXP_TOUT_TEH) 情報

```
typedef struct st_aexp_tout_teh {
    ID    dmnid;                    /* 異常例外発生ドメイン ID */
    ID    tehid;                    /* 異常例外発生タイムイベントハンドラ ID */
} T_AEXP_TOUT_TEH;
```

(14) 割込みハンドラ連続実行違反(AEXP_TOUT_INH) 情報

```
typedef struct st_aexp_tout_inh {
    ID        dmnid;                /* 異常例外発生ドメイン ID(システムドメイン ID に固定) */
    ID        inhid;                /* 異常例外発生割込みハンドラ ID */
} T_AEXP_TOUT_INH;
```

(15) 故障診断ハンドラ連続実行違反(AEXP_TOUT_FDH) 情報

```
typedef struct st_aexp_tout_fdh
{
    ID        dmnid;                /* 異常例外発生ドメイン ID(システムドメイン ID に固定) */
    ID        fdhid;                /* 異常例外発生故障診断ハンドラ ID */
} T_AEXP_TOUT_FDH;
```

(16) ドメイン連続実行違反(AEXP_TOUT_DMN) 情報

```
typedef struct st_aexp_tout_dmn
```

```
{
    ID      dmnid;                  /* 異常例外発生ドメイン ID */
} T_AEXP_TOUT_DMN;
```

(17) ディスパッチ禁止連続実行違反(AEXP_TOUT_DSP) 情報

```
typedef struct st_aexp_tout_dsp {
    ID      dmnid;                  /* 異常例外発生ドメイン ID */
} T_AEXP_TOUT_DSP;
```

(18) コンフィギュレーション異常(AEXP_CFG) 情報

```
typedef struct st_aexp_cfg
{
    T_AEXP_CFG_FACT     factor;     /* 詳細要因 */
    T_AEXP_CFG_CLASS    class;      /* クラス */
    INT                 item;       /* 項目 */
    ER                  ercd;       /* エラーコード */
} T_AEXP_CFG;

typedef enum
{
    AEXP_CFG_OUT_RANGE = 1,         /* 範囲外 */
    AEXP_CFG_DUPLICATTION,          /* メモリ領域重複 */
    AEXP_CFG_UNDEFINED,             /* 未定義情報 */
    AEXP_CFG_OVERFLOW,              /* 構成メモリ領域オーバーフロー */
    AEXP_CFG_INITFN,                /* 初期化機能エラー */
    AEXP_CFG_CREATE_ERR             /* オブジェクト生成エラー */
} T_AEXP_CFG_FACT;                  /* 詳細要因(コンフィギュレーション異常) */

typedef enum
{
    AEXP_CFG_CLASS_SYSTEM = 1,      /* システム管理情報 */
    AEXP_CFG_CLASS_MEMORY,          /* メモリ管理情報 */
    AEXP_CFG_CLASS_DOMAIN,          /* ドメイン管理情報 */
    AEXP_CFG_CLASS_USERSYS          /* ユーザ定義システムソフトウェア */
} T_AEXP_CFG_CLASS;                 /* 異常クラス(コンフィギュレーション異常) */
```

```
typedef enum {
    AEXP_CFG_ITEM_SYSINF = 1,          /* システム情報 */
    AEXP_CFG_ITEM_OBJINF,              /* オブジェクト情報(共通) */
    AEXP_CFG_ITEM_OBJINF_SYS,          /* オブジェクト情報(システム) */
    AEXP_CFG_ITEM_OBJINF_S,            /* オブジェクト情報(安全) */
    AEXP_CFG_ITEM_OBJINF_N,            /* オブジェクト情報(通常) */
    AEXP_CFG_ITEM_SSY_MNG,             /* サブシステム管理 */
    AEXP_CFG_ITEM_SDEV_MNG,            /* デバイス管理(安全) */
    AEXP_CFG_ITEM_NDEV_MNG,            /* デバイス管理(通常) */
    AEXP_CFG_ITEM_INT_MNG,             /* 割込み管理 */
    AEXP_CFG_ITEM_FD,                  /* 故障診断 */
    AEXP_CFG_ITEM_AH                   /* 異常例外 */
} T_AEXP_CFG_SYSTEM_ITEM;              /* 項目(システム管理情報) */

typedef enum {
    AEXP_CFG_ITEM_MEMMAP = 1,          /* メモリマップ */
    AEXP_CFG_ITEM_STACK,               /* スタック */
    AEXP_CFG_ITEM_MBF                  /* メッセージバッファ */
} T_AEXP_CFG_MEMORY_ITEM;              /* 項目(メモリ管理情報) */

typedef enum {
    AEXP_CFG_ITEM_SYSDMN = 1,          /* システムドメイン */
    AEXP_CFG_ITEM_SAFDMN,              /* 安全ドメイン */
    AEXP_CFG_ITEM_NORDMN               /* 通常ドメイン */
} T_AEXP_CFG_DOMAIN_ITEM;              /* 項目(ドメイン管理情報項目) */

typedef enum {
    AEXP_CFG_ITEM_DDEV = 1,            /* デバイスドライバ */
    AEXP_CFG_ITEM_INH,                 /* 割込みハンドラ */
    AEXP_CFG_ITEM_SSY,                 /* サブシステム */
    AEXP_CFG_ITEM_FDH,                 /* 故障診断ハンドラ */
    AEXP_CFG_ITEM_AEH                  /* 異常例外ハンドラ */
} T_AEXP_CFG_USER_ITEM;                /* 項目(ユーザ定義システムソフトウェア情報) */
```

(19) コントロールブロック異常(AEXP_CB) 情報

```
typedef struct st_aexp_cb
{
    T_AEXP_CB_FACT    factor;          /* 詳細要因 */
```

```
    VP          error_addr;         /* エラー発生アドレス */
    INT         size;               /* リードサイズ */
    UD          read_value;         /* リード値 */
    UD          clone_value;        /* リード値 ( 複製コントロールブロック側 ) */
} T_AEXP_CB;

typedef enum
{
    AEXP_CB_DATA_ERROR = 1,         /* データエラー */
    /* 以降に付加する場合は、実装定義 */
} T_AEXP_CB_FACT;
```

(20) 不正 API 呼び出し(AEXP_ILAPI) 情報

```
typedef struct st_aexp_ilapi
{
    T_AEXP_ILAPI_FACT   factor;     /* 詳細要因 */
    FN                  fncd;       /* 機能コード */
    /* 以降に付加する場合は、実装定義 */
} T_AEXP_ILAPI;

typedef enum
{
    AEXP_IAPI_FORBIDDEN_STATE = 1,  /* 呼び出し不可カーネル状態 */
    AEXP_IAPI_UNDEFINED,            /* 未定義 API */
    AEXP_IAPI_PARAM_MISS,           /* パラメータ数ミスマッチ */
    AEXP_IAPI_ILLEGAL_CALL_SAPI,    /* 安全 API 不正呼び出し */
    AEXP_IAPI_ILLEGAL_CALL_NAPI,    /* 通常 API 不正呼び出し */
} T_AEXP_ILAPI_FACT;
```

(21) タスク不正終了(AEXP_TSKE) 情報

```
typedef struct st_aexp_tske
{
    T_AEXP_TSKE_FACT    factor;     /* 詳細要因 */
    ID                  dmnid;      /* ドメイン ID */
    ID                  tskid;      /* タスク ID */
} T_AEXP_TSKE;
```

```
typedef enum
{
    AEXP_TSKE_ILLEGAL_TERM = 1,        /* 不正終了 */
    /* 以降に付加する場合は、実装定義 */
} T_AEXP_TSKE_FACT;                    /* 詳細要因(ドメイン/タスク終了エラー) */
```

(22) ドメイン/タスク終了エラー(AEXP_EEXIT_FACT) 情報

```
typedef struct st_aexp_eexit
{
    T_AEXP_EEXIT_FACT    factor;       /* 詳細要因 */
    ID                   id;           /* ドメイン/タスク ID */
} T_AEXP_EEXIT;                        /* 異常例外(ドメイン/タスク終了エラー) */

typedef enum
{
    AEXP_EEXIT_DMN_INDEPENDENT = 1, /* タスク独立部からのドメイン終了 */
    AEXP_EEXIT_TSK_INDEPENDENT,     /* タスク独立部からのタスク終了 */
    AEXP_EEXIT_TSK_DISDSP,          /* ディスパッチ禁止状態からの終了 */
    AEXP_EEXIT_DMN_SYSDMN,          /* システムドメインの終了 */
    AEXP_EEXIT_DMN_FEXT             /* ドメイン異常終了処理中のエラー発生 */
} T_AEXP_EEXIT_FACT;                /* 詳細要因(ドメイン/タスク終了エラー) */
```

(23) メッセージバッファ伝送エラー(AEXP_ERR_MBF) 情報

```
typedef struct st_aexp_err_mbf
{
    ID          mbfid;              /* メッセージバッファ ID */
    VP          data_ptr;           /* 受信データポインタ */
} T_AEXP_ERR_MBF;
```

(24) デバイスドライバ異常(AEXP_DEV) 情報

```
typedef struct st_aexp_dev
{
    T_AEXP_DEV_FACT factor;         /* 詳細要因 */
```

```
    ID              id;             /* デバイス ID / デバイスディスクリプタ / リクエスト ID */
} T_AEXP_DEV;                       /* 異常例外(デバイスドライバ異常) */

typedef enum
{
    AEXP_DEV_CLS = 1,               /* ts_cls_dev/tn_cls_dev にて発生 */
    AEXP_DEV_CAN,                   /* ts_can_dev/tn_can_dev にて発生 */
    AEXP_DEV_EVT,                   /* ts_evt_dev/tn_evt_dev にて発生 */
} T_AEXP_DEV_FACT;                  /* 詳細要因(デバイスドライバ異常) */
```

(25) サブシステム異常(AEXP_SSY) 情報

```
typedef struct st_aexp_ssy
{
    ID  ssid;                       /* サブシステム ID */
    /* 以降に付加する場合は、実装定義 */
} T_AEXP_SSY;                       /* 異常例外(サブシステム異常) */
```

(26) 初期登録 API エラー(AEXP_INIDEF_API) 情報

```
typedef struct st_aexp_inidef_api
{
    T_AEXP_INIDEF_API_FACT  factor; /* 詳細要因 */
    UW                      class;  /* クラス */
    UW                      item;   /* 項目 */
    ID                      id;     /* ID 番号 */
} T_AEXP_INIDEF_API;                /* 異常例外(初期登録 API) */

typedef enum
{
    AEXP_INIDEF_API_PAR = 1,        /* パラメータエラー */
    AEXP_INIDEF_API_ID,             /* 不正 ID */
    AEXP_INIDEF_API_LIMIT,          /* 値制限オーバ */
} T_AEXP_INIDEF_API_FACT;           /* 詳細要因(初期登録 API エラー) */
```

(27) ドメインの開始エラー(AEXP_DMN) 情報

```
typedef struct st_aexp_dmn
{
        T_AEXP_DMN_FACT   factor;       /* 詳細要因 */
        ID                dmnid;        /* ドメイン ID */
} T_AEXP_DMN;

typedef enum
{
        AEXP_DMN_START= 1,              /* ドメイン起動処理中にエラー発生 */
       /* 以降に付加する場合は、実装定義 */
} T_AEXP_DMN_FACT;
```

6.2.3 異常例外ハンドラの概要

異常例外ハンドラは、異常例外の発生時に実行されるプログラムの実行単位である。

各異常例外ハンドラは「表 6-2 異常例外ハンドラ管理情報」に示す情報を持つ。各情報は、ハンドラの登録時に決められ変更することはできない。一部の情報は固定値であり、必ずしも実際のデータとして実装されるわけではない。

表 6-2 異常例外ハンドラ管理情報

名称	意味
異常例外番号	異常例外を識別する番号
所属ドメイン	異常例外ハンドラが所属するドメインの ID 番号(システムドメインに固定)
保護レベル	異常例外ハンドラの保護レベル(保護レベル 0 に固定)
ハンドラ属性	異常例外ハンドラの属性
ハンドラ起動アドレス	異常例外ハンドラの実行プログラムの開始アドレス

異常例外番号は、異常例外を識別するために静的に割り当てられた番号である。異常例外ハンドラは、異常例外に一対一で対応する。よって、異常例外ハンドラは異常例外番号で識別することができる。

所属ドメインは、異常例外ハンドラが所属するドメインの ID 番号である。異常例外ハンドラは必ずシステムドメインに所属するので、システムドメインの ID 番号に固定される。

保護レベルは、異常例外ハンドラの保護レベルである。異常例外ハンドラは必ずシステムドメインに所属するので、システムドメインの保護レベルに固定される。

ハンドラ属性は、異常例外ハンドラの属性である。詳細は「6.2.4 異常例外ハンドラの属性」を参照。

ハンドラ起動アドレスは、異常例外ハンドラの起動時に実行されるプログラムの起動アドレスである。詳細は「6.2.6 異常例外ハンドラの実行プログラム」を参照。

6.2.4 異常例外ハンドラの属性

異常例外ハンドラの登録時にハンドラ属性が静的に指定される。「表 6-3 異常例外ハンドラ属性」にハンドラ属性を示す。

表 6-3 異常例外ハンドラ属性

名称	属性	意味
TAA_UPDATE	ハンドラ変更可能	安全 API による異常例外ハンドラの変更を許可する。
TAA_RETURN	復帰可能	異常例外ハンドラの終了後に、元の状態に復帰可能である。

6.2.5　異常例外ハンドラの登録

異常例外ハンドラは、TRON Safe Kernel に静的に組み込まれており、カーネル起動処理にて有効となる。よって、カーネル初期化状態においても、異常例外を受け付け処理することができる。

TAA_UPDATE 属性(ハンドラ変更可能)の異常例外ハンドラのみ、安全 API により他の異常例外ハンドラへ変更が可能である。TAA_UPDATE 属性ではない異常例外ハンドラは、API による変更を許さない。よって、TRON Safe Kernel に最初に組み込まれたハンドラが有効であり続ける。

6.2.6　異常例外ハンドラの実行プログラム

TRON Safe Kernel は、異常例外を受け付けると、登録されている異常例外ハンドラの実行プログラムを呼び出す。
異常例外ハンドラの実行プログラムは、以下の形式の C 言語の関数とする。関数名は任意である。

```
INT     aehdr( UINT aexpno,   INT tskstat,   VP aexppar )
```

関数のパラメータとして、発生した異常例外の異常例外番号が axepno に渡される。tskstat には異常例外が発生したときのカーネル状態が渡される。aexppar は異常例外毎に定められた情報である。

異常例外が発生すると TRON Safe Kernel は最優先で異常例外ハンドラを実行する。その際、すべての割込みが禁止される。異常例外ハンドラは実行中に割込みを許可してはならない。

TAA_RETURN 属性の異常例外ハンドラは、戻り値として AE_RETURN(=1)を返すことにより、ハンドラを終了し、元の状態に復帰することができる。戻り値として AE_NORETURN(=0、または AE_RETURN 以外の値)を返した場合は、TRON Safe Kernel は安全状態に遷移し、すべてのプログラムの実行を停止する。
TAA_RETURN 属性ではない異常例外ハンドラが終了した場合、TRON Safe Kernel は安全状態に遷移し、すべてのプログラムの実行を停止する。

6.2.7 異常例外ハンドラから使用可能な API

異常例外ハンドラから使用可能な API は限定される。また、API が使用可能なのは、カーネル実行状態から異常例外ハンドラが実行された場合のみである。その他の状態から異常例外ハンドラが実行された場合、API を発行するとエラーE_CTX を返す。

「表 6-4 異常例外ハンドラから使用可能な API 一覧」に異常例外ハンドラから使用可能な API を示す。

表 6-4 異常例外ハンドラから使用可能な API 一覧

分類	名称	機能
同期制御	ts_wup_tsk	タスクの起床
	ts_sig_sem	セマフォの資源返却
	ts_set_flg	イベントフラグのセット
情報取得	ts_ref_sys	システム状態の参照
	ts_ref_dmn	ドメイン状態の参照
	ts_get_tim	システム時刻の参照
	ts_get_otm	システム稼働時間の参照

6.2.8 異常例外ハンドラのスタック

異常例外ハンドラは、異常例外ハンドラ専用のスタックを使用する。異常例外ハンドラ専用のスタックのサイズは静的に決定され、スタックが使用するメモリ領域も静的に確保される。具体的な仕様は実装定義とする。

6.2.9 異常例外機能の API

異常例外機能の API を「表 6-5 異常例外機能 API 一覧」に示す（各 API の詳細は「8.3.5 異常例外機能の API」参照）。

表 6-5 異常例外機能 API 一覧

分類	安全 API 名称	通常 API 名称	初期登録 API 名称	説明
ハンドラ登録	ts_def_aeh	—	TS_DEF_AEH	異常例外ハンドラの登録
ハンドラ操作	ts_ras_aexp	—	—	異常例外の発生

6.3 故障診断機能

TRON Safe Kernel は、故障を検出するために、ハードウェアの故障診断処理を実行する機能と、カーネルを自己診断する機能を持つ。故障が検出された場合は異常例外として、異常例外機能で処理を行う。

故障診断機能の一覧を、「表 6-6 故障診断機能一覧」に示す。

表 6-6 故障診断機能一覧

故障診断名	診断対象	参照先
カーネル自己診断	コントロールブロックの診断	6.3.1 章
	システムタイマ割込みハンドラの診断	
	メッセージバッファ伝送エラー検出	
ハードウェア故障診断	特定ハードウェアに依存する故障診断	6.3.2 章

6.3.1 カーネル自己診断

TRON Safe Kernel 自体の故障を検出するための機能である。カーネル自己診断には以下の項目がある。

(1)コントロールブロックの診断

コントロールブロックは、TRON Safe Kernel の制御に関わるデータを格納しているメモリ領域である。コントロールブロックはデータの複製を持ち、複製には複製元のデータをビット反転して保存する。読み出し時に複製元と複製を比較することによりエラーを検出する。ここでエラー検出した場合は、異常例外を発生する。

(2)システムタイマ割込みハンドラの診断

TRON Safe Kernel は、自身が使用する割り込みハンドラの自己診断を定期的に実行することにより、割込みハンドラが正しく呼び出せることを確認する。

具体的には、TRON Safe Kernel が使用する割込みハンドラは、システムタイマの割込みハンドラである。システムタイマの割込みハンドラは定められた間隔で実行され、その処理の中で最高優先度タイマのリセットを実施する。本処理が正常に機能していなければ、最高優先度タイマがリセットされず、最高優先度タイマ割込みが発生する。

(3)メッセージバッファ伝送エラー検出

メッセージバッファ「3.3.4 メッセージバッファ」を用いた通信において、伝送内容の誤り検出を行う。伝送エラーの検出には、CRC(Cyclic Redundancy Check)による伝送内容の誤り検出を実施する。

CRC により誤り検出した場合は、異常例外を発生する。

カーネル自己診断機能は項目ごとに有効/無効は静的に設定できる。ユーザはシステム全体の機能安全の設計に基づき、有効/無効の設定を行う。

【補足説明】

カーネルの自己診断機能の無効の設定は、システム全体の機能安全の設計上、その診断が不要である場合に処理を軽減させるために使用する。たとえば、ハードウェアで故障検出が保障され、カーネルとして自己診断が不要である場合などが考えられる。

6.3.2 ハードウェア故障診断

TRON Safe Kernel のカーネル自己診断の対象は、特定のハードウェアに依存しない事項のみである(「6.3.1 カーネル自己診断」参照)。

よって、カーネル自己診断の対象外となるハードウェアの故障診断の処理は、ユーザ定義システムソフトウェア(デバイスドライバ、サブシステム、割込みハンドラ)として実現する。

ハードウェア故障診断の処理を必要とするタイミングで実行するには、TRON Safe Kernel の故障診断管理機能の故障診断ハンドラを使用することができる。故障診断ハンドラは指定したタイミングで実行され、そこからユーザ定義システムソフトウェアで実現したハードウェア故障診断の処理を実行する(詳細は「4.4 故障診断管理機能」を参照)。

診断の内容はハードウェアに依存し、実装定義である。

【補足説明】

ハードウェアの故障診断処理が簡単なものである場合は、故障診断ハンドラのみで故障診断を行うことも可能である。

6.4 ドメインの分割

6.4.1 ドメインの空間的分離

ドメインの空間的分離は、ドメインごとに独立した資源を割り当て、その資源を他のドメインから保護することにより実現する。

ドメインに割り当てられる資源は、ドメインに所属するプログラムが使用するメモリ領域と、カーネルオブジェクトである。

ドメインの空間的分離は以下のように実現される。

(1) 資源の割り当て

メモリ領域およびカーネルオブジェクトは、各ドメインに独立に割り当てられ共有することはない。

安全ドメイン、システムドメインのメモリ領域の割り当ては、システム構築時にコンフィギュレーションにより静的に行われる。カーネルオブジェクトが使用する資源(メモリ)はシステム構築時にコンフィギュレーションにより静的に決定される。

通常ドメインの資源割り当ては基本的には安全ドメインに準じるが、タスクのスタックなどの一部のメモリ領域は動的に割り当てられる。ただし、割り当てる元となるメモリ領域は、システム構築時にコンフィギュレーションにより静的に定められる。

(2) 資源の保護

各アプリケーションドメインに割り当てられたリソースは、ハードウェアのメモリ保護機能の機構により他ドメインからのアクセスに対して保護される(詳細は「3.6.9 ドメインの空間的保護」を参照)。

カーネルオブジェクトは、安全ドメインと通常ドメインで独立して管理される。具体的には、カーネルオブジェクトのコントロールブロックは安全ドメインと通常ドメインで異なったメモリ領域に確保されており、また、コントロールブロックに対する操作は TRON Safe Kernel 内部のコントロールブロック I/F を介して可能とする。よって、通常アプリケーションからの、安全ドメインに割り当てられたカーネルオブジェクトに対するアクセスや、許可されていないカーネルオブジェクトに対する操作は防止される。

(3) 故障伝搬の防止

通常ドメインのプログラムの故障が、安全ドメインのプログラムへ伝播することは、以下のように防止される。

安全ドメインのプログラムから、通常ドメインのカーネルオブジェクトへの操作は、TRON Safe Kernel の API でのみ行うことができる。API はパラメータのチェックを行うことにより、許可されていない値の伝播を防ぐ（防衛的プログラミング）。
また、安全ドメインのプログラムから、通常ドメインのカーネルオブジェクトを操作する API は待ち状態を伴わない。よって、通常ドメインの故障により、安全ドメインのプログラムの待ちが解除されなくなることはない（詳細は「2.6.4 カーネルオブジェクトのドメイン間の保護」を参照）。

【補足説明】
安全ドメインから通常ドメインの操作に限らず、カーネルオブジェクトに対する操作は API を介してのみ可能である。よって、全てのカーネルオブジェクトの操作において、防衛的プログラムは有効である。

6.4.2 ドメインの時間的分離

ドメインの時間的分離とは、ドメインおよびドメインに所属する実行単位に対し、そのプログラムの実行優先度と連続実行可能な時間を設定することにより、時間資源（CPU 使用時間）をドメイン毎に割り当てることである。
もし設定された連続実行時間を超えた場合は、異常例外として検出することにより、あるドメインのプログラムの実行が他のドメインのプログラムの実行により阻害されることを検出する。
ドメインの時間的分離は以下のように実現される。

(1) 実行優先度の制御
すべてのプログラムの実行単位に実行優先度が割り当てられる。TRON Safe Kernel はその実行優先度に基づき、プリエンプティブな優先度ベーススケジューリング方式を行う。具体的には、TRON Safe Kernel の内部のスケジューラおよびディスパッチャにより、実行可能なタスクおよびタイムイベントハンドラのなかから最高優先度を持つものを選択し実行する。スケジュールの実行は、タスク起動に関連した API 実行およびシステムタイマの周期において行われる。

割込みハンドラおよび異常例外ハンドラは、実行中のプログラムの実行単位よりも優先度が高い場合に、実行中のプログラムの実行単位に割り込んで実行される。

各ドメインにはドメイン内の全プログラム実行単位に共通となる優先度の上限と下限を設定することができる。システムドメインには上限がない。安全ドメインには指定可能な上限と下限が設定される。各プログラムの実行単位の優先度は、所属しているドメインに設定されている優先度の上限と下限を超えて設定することはできない。

(2) 最大実行時間の制御
特定のプログラムの実行単位が CPU を占有し続けることにより、高優先度あるいは同優先度の他のプログラムの実行単位の実行を阻害しないことは、以下の機能により実現される。

- プログラムの実行単位およびドメインには連続して実行可能な最大連続実行時間が設定される。
- TRON Safe Kernel の時間管理において、実行中のプログラムの実行単位の連続実行時間が、最大連続実行時間を超過していないか確認される。もし超えた場合は異常例外を発生する。また、タスクおよびドメインの全体実行時間と連続実行時間は API により取得可能である。
- ディスパッチ禁止区間には、最大有効時間が設定される。TRON Safe Kernel の時間管理において、ディスパッチ禁止区間の時間が、最大有効時間を超過していないか確認される。もし超えた場合は異常例外を発生する。

- ディスパッチ禁止は、ディスパッチ禁止の APIを実行したタスクが所属するドメインにおいてのみ有効となる。よって、ディスパッチ禁止区間で、ディスパッチ禁止にできるタスクの優先度の上限は、ディスパッチ禁止を実行したタスクの所属しているドメインに設定されている上限となる。
- タイムイベントハンドラは、TRON Safe Kernel の時間管理において指定された起動時間後に、優先度の低いプログラムの実行単位に割り込んで実行される
- 割込みハンドラは、各割込み終了時に、最大連続割込み実行時間を超過していないか確認される。もし超えた場合は異常例外を発生する。

6.5 システムソフトウェアの設定の保護

ユーザ定義システムソフトウェア（サブシステム、デバイスドライバ、割込みハンドラ、など）はアプリケーションからの登録などの操作は禁止する。

ユーザ定義ソフトウェアの登録は、システムドメインにおいて、コンフィギュレーションに記述された初期登録 API によってのみ可能である。

また、サブシステム、デバイスドライバは、登録時に呼び出し可能なドメイン（安全ドメイン、通常ドメイン）を設定する。

6.6 安全状態への移行

TRON Safe Kernel は、異常が検出された際に、それに対応する異常例外ハンドラを実行する。異常例外ハンドラの処理はユーザにより実装される。異常例外ハンドラは終了時に元の状態への復帰、または TRON Safe Kernel の安全状態への移行を指定できる。TRON Safe Kernel の安全状態とは、すべてのタスク、タイムイベントハンドラ、割込みハンドラを含む実行単位の実行を停止することである。

異常例外ハンドラ自体の異常は、最高優先度タイマにより検出する。最高優先度タイマの発火時の動作は、ユーザにより規定するものとする。TRON Safe Kernel 自体は、最高優先度タイマの発火後は全ての動作は停止する。

【補足説明】

最高優先度タイマは、システムソフトウェアの実行自体の故障を検出するためのウォッチドックタイマである。最高優先度タイマの動作は TRON Safe Kernel では規定しない。

6.7 応答性の保証

TRON Safe Kernel の割込み応答性能は、以下の機構により保証される。

- ハードウェアによって規定される割込みの最大遅延時間ならびに本ソフトの割込み処理機構の実行命令の処理時間を計算することにより、割込み発生から割込みハンドラの処理開始までの最大時間が予測できる.
- TRON Safe Kernel は、割込みハンドラ終了時に既定の実行時間を超過していないか確認する。超過していた場合には、異常例外処理を実行する。
- アプリケーションは割込みを禁止することができない。

6.8 デバッグ機能

デバッグ機能とは、システムソフトウェアに対する故障注入と、メモリ、I/O に対する故障注入の模擬ができる機能である。

デバッグ機能は、TRON Safe Kernel の機能としては対応しない。必要に応じて、ユーザ定義システムソフトウェア（デバイスドライバ、サブシステム、割込みハンドラ、故障診断ハンドラ）として実現する。よって、TRON Safe Kernel 自体にはデバッグ機能および、それを実現するソフトウェアは含まれない。

7. TRON Safe Kernel のコンフィギュレーション

7.1 コンフィギュレーションの概要

TRON Safe Kernel のシステムに関する各種設定や、初期状態で組み込まれるソフトウェアの定義を行うことをコンフィギュレーションと呼ぶ。

コンフィギュレーションは、コンフィギュレーション定義ファイルに記述される。コンフィギュレーションは、TRON Safe Kernel の構築時に指定され、実行中に変更することはできない。

コンフィギュレーションの記述は、コンフィギュレーション情報と初期登録 API に分けられる。

7.1.1 コンフィギュレーション情報

コンフィギュレーション情報は、システムに関する各種の静的なデータである。

コンフィギュレーション情報は、定数である。具体的には、C 言語のマクロ定義(#define)で定義される数値データ、または定数データ(const 型)である。

コンフィギュレーション情報には以下がある。

- システム管理情報(7.2 章)
  TRON Safe Kernel のシステムの資源数や制限値などを指定する情報である。
  例えば、ドメインの定義、タスクの最大数などがこれに相当する。
- メモリ管理情報(7.3 章)
  メモリ資源に関する管理情報である。
  例えば、メモリマップ、スタック領域の情報などがこれに相当する。
- ドメイン管理情報(7.4 章)
  ドメインに関する管理情報である。ドメイン毎に規定される資源数や制限値などを指定する情報である。
  例えば、ドメインで指定可能な最高優先度などがこれに相当する。

7.1.2 初期登録 API

初期登録 API は、TRON Safe Kernel の初期状態で組み込まれるソフトウェアの登録を行う。

対象は、以下のユーザ定義システムソフトウェアである。

- デバイスドライバ
- 割込みハンドラ
- サブシステム
- 故障診断ハンドラ
- 異常例外ハンドラ

初期登録 API は、TRON Safe Kernel の他の API と同様に、C 言語の関数の形式で定義する。ただし、初期登録 API はコンフィギュレーション定義ファイルの定められた個所にのみ記述可能であり、プログラム中から使用することはできない。

初期登録 API は以下の規則で記述する。

(1) 名称

初期登録 API の名称は、英大文字と"_"で構成され、"TS_"から始まる。初期登録 API の名称は"TS_XXX_YYY"の形式である。"XXX"は機能(操作)を表し、"YYY"は操作対象を表す。

(2) パラメータ

初期登録 API のパラメータは、整数定数または文字列のいずれかである。詳細は個々の初期登録 API にて定義する。

7.1.3 コンフィギュレーションの実行

コンフィギュレーションは、コンフィギュレーション定義ファイルにて記述する。

コンフィギュレーション定義ファイルは、コンフィギュレーションの種別毎に存在する。

「表 7-1 コンフィギュレーション定義ファイル一覧」にコンフィギュレーション定義ファイルの一覧を示す。

表 7-1 コンフィギュレーション定義ファイル一覧

分類	ファイル名	内容
システム管理情報	SYSCONF.CNF	システム管理情報
メモリ管理情報	MEMCONF.CNF	メモリマップ管理情報
	STKCONF.CNF	スタック管理情報
	MBFCONF.CNF	メッセージバッファ管理情報
ドメイン管理情報	DMNCNF.CNF	ドメイン管理情報
ユーザ定義ソフトウェア登録情報	DEVCONF.CNF	デバイスドライバ登録情報
	INTCONF.CNF	割込みハンドラ登録情報
	SSYCONF.CNF	サブシステム登録情報
	FDHCONF.CNF	故障診断ハンドラ登録情報
	AEHCONF.CNF	異常例外ハンドラ登録情報

7.2 システム管理情報

システム管理情報は、TRON Safe Kernel の各種設定を指定する数値情報である。

システム管理情報で指定される情報の一覧を「表 7-2 システム管理情報一覧」に示す。

表 7-2 システム管理情報一覧

分類	名称	意味	値の範囲
システム情報	TS_LST_PRI	指定可能な最低実行優先度 (タスク優先度)	16 ～ 250
	TS_HST_SPRI	指定可能な最高実行優先度(システム優先度)	-250 ～ -1
	TS_MAX_TSKTIM	タスク最大連続実行時間	1 ～ 実装定義
	TS_MAX_TEHTIM	タイムイベントハンドラ最大連続実行時間	1 ～ 実装定義
	TS_MAX_DMNTIM	ドメイン最大連続実行時間	1 ～ 実装定義(※1)

	TS_MAX_DSPTIM	ディスパッチ禁止最大連続実行時間	1 ～ 実装定義
	TS_MAX_INTTIM	割込みハンドラ最大連続実行時間	1 ～ 実装定義
	TS_MIN_SCHTIM	スケジュール最小間隔時間	1 ～ 実装定義(※7)
	TS_MAX_SCHTIM	スケジュール最大間隔時間	1 ～ 実装定義(※7)
	TS_MAX_SDMN	安全ドメイン数	1 ～ 実装定義(※8)
	TS_MAX_NDMN	通常ドメイン数	0 ～ 実装定義(※8)
	TS_CONF_DLYCHK	コンフィギュレーション情報の遅延チェック指定	1: 遅延チェック 0: 起動時チェック(※9)
	TS_SYSTIM_MET	システムタイマ方式	1: タイマ可変間隔時間方式 0: タイマ固定間隔時間方式 (※10)
	TS_STK_SIZE_INH	割込み専用スタックサイズ	1 ～ UINT_32MAX(※3) (※11)
	TS_STK_SIZE_AEH	異常例外専用スタックサイズ	1 ～ UINT_32MAX(※3) (※11)
	TS_STK_SIZE_FDH	故障診断専用スタックサイズ	1 ～ UINT_32MAX(※3) (※11)
	TS_TOTAL_MBFSZ	バッファ用メモリの総メモリサイズ	1 ～ 実装定義
	TS_MAX_MRGN	メモリマップ管理情報中のメモリ領域の最大数	1 ～ INT_16MAX(※2)
	TS_MAX_STACK	ドメイン毎のスタック数の最大値	1 ～ INT_16MAX(※2)
	TS_MAX_TMOUT	最大タイムアウト時間	1 ～ 実装定義
オブジェクト情報 (共通)	TS_MAX_SEMCNT	最大セマフォ資源数	1 ～ INT_16MAX(※2)
	TS_MAX_WUPCNT	最大タスク起床要求数	1 ～ INT_16MAX(※2)
	TS_MAX_SUSCNT	最大タスク強制待ち要求数	1 ～ INT_16MAX(※2)
	TS_MAX_CYCTIM	最大周期ハンドラ起動時間間隔	1 ～ UINT_32MAX(※3)
	TS_MAX_CYCPHS	最大周期ハンドラ起動位相時間	1 ～ UINT_32MAX(※3)
	TS_MAX_ALMTIM	最大アラームハンドラ起動時間	1 ～ UINT_32MAX(※3)
オブジェクト情報 (システムドメイン)	TS_MAX_TSK_SY	最大タスク数(システムドメイン)	1 ～ INT_16MAX(※2)
	TS_MAX_SEM_SY	最大セマフォ数(システムドメイン)	0 ～ INT_16MAX(※2)
	TS_MAX_FLG_SY	最大イベントフラグ数(システムドメイン)	0 ～ INT_16MAX(※2)
	TS_MAX_MTX_SY	最大ミューテックス数(システムドメイン)	0 ～ INT_16MAX(※2)
	TS_MAX_MBF_SY	最大メッセージバッファ数(システムドメイン)	0 ～ INT_16MAX(※2)
	TS_MAX_CYC_SY	最大周期ハンドラ数(システムドメイン)	0 ～ INT_16MAX(※2)
	TS_MAX_ALM_SY	最大アラームハンドラ数(システムドメイン)	0 ～ INT_16MAX(※2)
	TS_MAX_MSZ_SY	最大メッセージサイズ(システムドメイン)	1 ～ INT_16MAX(※2)
オブジェクト情報	TS_MAX_TSK_SA	最大タスク数(安全ドメイン)	1 ～ INT_16MAX(※2)
	TS_MAX_SEM_SA	最大セマフォ数(安全ドメイン)	0 ～ INT_16MAX(※2)

(安全ドメイン)	TS_MAX_FLG_SA	最大イベントフラグ数(安全ドメイン)	0 ～ INT_16MAX(※2)
	TS_MAX_MTX_SA	最大ミューテックス数(安全ドメイン)	0 ～ INT_16MAX(※2)
	TS_MAX_MBF_SA	最大メッセージバッファ数(安全ドメイン)	0 ～ INT_16MAX(※2)
	TS_MAX_CYC_SA	最大周期ハンドラ数(安全ドメイン)	0 ～ INT_16MAX(※2)
	TS_MAX_ALM_SA	最大アラームハンドラ数(安全ドメイン)	0 ～ INT_16MAX(※2)
	TS_LST_DPRI_SA	指定可能な最低タスク優先度(安全ドメイン)	16 ～ TS_LST_PRI(※4)
	TS_HST_DPRI_SA	指定可能な最高タスク優先度(安全ドメイン)	16 ～ TS_LST_PRI(※4)
	TS_MAX_MSZ_SA	最大メッセージサイズ(安全ドメイン)	1 ～ INT_16MAX(※2)
オブジェクト情報 (通常ドメイン)	TS_MAX_TSK_NA	最大タスク数(通常ドメイン)	1 ～ INT_16MAX(※2)
	TS_MAX_SEM_NA	最大セマフォ数(通常ドメイン)	0 ～ INT_16MAX(※2)
	TS_MAX_FLG_NA	最大イベントフラグ数(通常ドメイン)	0 ～ INT_16MAX(※2)
	TS_MAX_MTX_NA	最大ミューテックス数(通常ドメイン)	0 ～ INT_16MAX(※2)
	TS_MAX_MBF_NA	最大メッセージバッファ数(通常ドメイン)	0 ～ INT_16MAX(※2)
	TS_MAX_CYC_NA	最大周期ハンドラ数(通常ドメイン)	0 ～ INT_16MAX(※2)
	TS_MAX_ALM_NA	最大アラームハンドラ数(通常ドメイン)	0 ～ INT_16MAX(※2)
	TS_LST_DPRI_NA	指定可能な最低タスク優先度(通常ドメイン)	16 ～ TS_LST_PRI(※5)
	TS_HST_DPRI_NA	指定可能な最高タスク優先度(通常ドメイン)	16 ～ TS_LST_PRI(※5)
	TS_MAX_MSZ_NA	最大メッセージサイズ(通常ドメイン)	1 ～ INT_16MAX(※2)
サブシステム管理	TS_MAX_SSY	最大サブシステム登録数	0 ～ 255
	TS_LST_SSYPRI	指定可能な最低サブシステム優先度	16 ～ 255
	TS_MAX_SSYFNO	最大サブシステム機能コード数	1 ～ 4095
	TS_MAX_SVCPARA	最大拡張 SVC パラメータ数	4 ～ 255
デバイス管理 (安全 API)	TS_MAX_DEV_S	最大デバイス登録数(安全 API)	0 ～ INT_16MAX(※2)
	TS_MAX_OPDEV_S	最大デバイスオープン数(安全 API)	1 ～ INT_16MAX(※2)
	TS_MAX_RQDEV_S	最大デバイスリクエスト数(安全 API)	1 ～ INT_16MAX(※2)
	TS_DEVT_MBFSZ_S	事象通知用メッセージバッファのサイズ(安全 API)	0 ～ INT_16MAX(※2、※6)
	TS_DEVT_MBFMAX_S	事象通知用メッセージバッファのメッセージ最大長(安全 API)	0 ～ INT_16MAX(※2)
デバイス情報 (通常 API)	TS_MAX_DEV_N	最大デバイス登録数(通常 API)	0 ～ INT_16MAX(※2)
	TS_MAX_OPDEV_N	最大デバイスオープン数(通常 API)	1 ～ INT_16MAX(※2)
	TS_MAX_RQDEV_N	最大デバイスリクエスト数(通常 API)	1 ～ INT_16MAX(※2)
	TS_DEVT_MBFSZ_N	事象通知用メッセージバッファのサイズ(通常 API)	0 ～ INT_16MAX(※2、※6)
	TS_DEVT_MBFMAX	事象通知用メッセージバッファのメッセージ	0 ～ INT_16MAX(※2)

	_N	最大長(通常 API)	
割込み管理	TS_MAX_INTNO	最大割込みハンドラ番号	1 ～ INT_16MAX(※2)
故障診断	TS_CHK_CB	コントロールブロック診断	1: 有効 0: 無効
	TS_CHK_SYSTIM	システムタイマ割込みハンドラ診断	1: 有効 0: 無効
	TS_CHK_MBF	メッセージバッファ伝送エラー診断	1: 有効 0: 無効
	TS_MAX_FDH	最大故障診断ハンドラ数	0 ～ INT_16MAX(※2)
	TS_MAX_FDHCYC	最大故障診断ハンドラ起動時間間隔	1 ～ UINT_32MAX(※3)
異常例外	TS_MAX_AEX_FD	最大故障診断異常例外番号	0x00008000 ～ 0x0000FFFF
	TS_MAX_AEX_UD	最大ユーザ定義異常例外番号	0x00010000 ～ 0x0001FFFF

※1 TS_MAX_DMNTIM は、TS_MAX_TSKTIM および TS_MAX_TEHTIM より大きくなければならない。

※2 “INT16_MAX”と記載したものは、16 ビット幅の符号付き整数の表現できる最大値(32767)を示す。ただし、INT16_MAX は機能仕様上において許される最大値であるが、実装仕様においてそれ以下の値に定めることができる。

※3 “UINT32_MAX”と記載したものは、32 ビット幅の符号なし整数の表現できる最大値(4294967295)を示す。ただし、UINT32_MAX は機能仕様上において許される最大値であるが、実装仕様においてそれ以下の値に定めることができる。

※4 TS_HST_DPRI_SA は、TS_LST_DPRI_SA と同じ、またはそれより高い優先度（小さい値）でなければならない。

※5 TS_HST_DPRI_NA は、TS_LST_DPRI_NA と同じ、またはそれより高い優先度（小さい値）でなければならない。

※6 0 の場合は、事象通知用メッセージバッファは生成されない。

※7 TS_MIN_SCHTIM および TS_MAX_SCHTIM は、TS_MIN_SCHTIM≦TS_MAX_SCHTIM の関係でなくてはならない。

※8 TS_MAX_SDMN(安全ドメイン数) + TS_MAX_NDMN(通常ドメイン数) + 1 ≦ 127 でなければならない。

※9 遅延チェックを指定することにより、システム起動時に実行されるコンフィギュレーションのチェックがドメイン起動時などに実行される。

※10 指定可能な方式は実装定義である。なおタイマ可変間隔時間方式は、システムタイマ割込みの間隔時間が可変である。一方のタイマ固定間隔時間方式は、システムタイマ割込みの間隔時間が固定である。

※11 割込みハンドラ、異常例外ハンドラ、故障診断ハンドラで使用するスタックは、「7.3.3 スタック管理情報」と異なる領域に専用スタックが確保される。

システム管理情報は、C 言語のマクロ定義(#define)にて各値が定義される。

以下に記述例を示す。

```
/* システム管理情報 */
/* オブジェクト管理(システムドメイン) */
#define   TSCF_MAX_TSK_SY           (100)     /* 最大タスク数(システムドメイン) */
#define   TSCF_MAX_SEM_SY           (50)      /* 最大セマフォ数(システムドメイン) */
#define   TSCF_MAX_FLG_SY           (50)      /* 最大イベントフラグ数(システムドメイン) */
#define   TSCF_MAX_MTX_SY           (50)      /* 最大ミューテックス数(システムドメイン) */
#define   TSCF_MAX_MBF_SY           (50)      /* 最大メッセージバッファ数(システムドメイン) */
#define   TSCF_MAX_CYC_SY           (50)      /* 最大周期ハンドラ数(システムドメイン) */
#define   TSCF_MAX_ALM_SY           (50)      /* 最大アラームハンドラ数(システムドメイン) */
```

7.3 メモリ管理情報

メモリ管理情報はメモリ領域に関するコンフィギュレーション情報である。

メモリ管理情報には以下がある。

- メモリマップ管理情報
- スタック管理情報
- メッセージバッファ管理情報

7.3.1 メモリ領域の概要

TRON Safe Kernel の操作対象とするメモリは、複数のメモリ領域に分割され管理される。

各メモリ領域は、特定のメモリの領域を示すアドレスとその属性を表す情報を持つ。メモリ領域の情報の内容を「表 7-3 メモリ領域情報一覧」に示す。

表 7-3 メモリ領域情報一覧

分類	名称	意味	値の範囲
メモリ領域	mem_atr	メモリ領域の属性	—
	mem_top	メモリ領域の先頭アドレス	実装定義
	mem_bottom	メモリ領域の終端アドレス	実装定義

mem_top と mem_bottom によりメモリ領域のアドレスが指定される。mem_top < mem_bottom とする。

メモリ領域は属性により、コード領域、データ領域、スタック領域に分類される。メモリ領域の属性を「表 7-4 メモリ領域属性一覧」に示す。

TSMA_CDIS 属性は他の属性と組み合わせて使用する。TMSA_CDIS が組合せ可能な領域は実装定義である。

TSMA_NONE 属性は、そのメモリ領域情報は無効（使用しないデータ）であることを示す。

表 7-4 メモリ領域属性一覧

名称	意味	用途
TSMA_CODE	コード領域	プログラムのコードを格納する領域
TSMA_DATA	データ領域	主に静的に確保されたデータを格納する領域
TSMA_STACK	スタック領域	タスクまたはタイムイベントハンドラの実行中にスタックに使用される領域
TSMA_NONE	未使用データ	該当のメモリ領域情報は無効であることを示す
TSMA_CDIS	キャッシュ禁止	メモリアクセスにキャッシュを使用しない領域

7.3.2 メモリマップ管理情報

メモリマップは複数のメモリ領域から構成される。メモリマップ管理情報は、メモリ領域情報の集合である。
ドメイン毎に一つのメモリマップ管理情報が設定される。つまり、(TS_MAX_SDMN + TS_MAX_NDMN + 1)個のメモリマップ管理情報が存在する。
一つのメモリマップ管理情報に含まれるメモリ領域の最大数は実装定義である。ただし、一つ以上のコード領域と一つ以上のデータ領域、および一つのスタック領域を含まなくてはならない。つまり、一つのメモリマップ管理情報には3つ以上のメモリ領域は含まれる。

メモリマップ管理情報は、C 言語のマクロ定義(#define)と、定数型データ(const 型)にて各値が定義される。
メモリマップ管理情報の記述形式を以下に示す。

```
#define TS_MAX_MRGN             メモリマップ管理情報中のメモリ領域の最大数

TS_CONF_MMAP = {   /* メモリマップ管理情報の並び */
        /* 一番目のメモリマップ管理情報 */
        {
                1番目のメモリ領域 ,
                2番目のメモリ領域 ,
                /* 繰り返し */
                TS_MAX_MRGN 番目のメモリ領域
        } ,
        /* 二番目のメモリマップ管理情報*/
        {
                /* 省略 */
        },

        /* 繰り返し(省略) */

        /* (TS_MAX_SDMN + TS_MAX_NDMN + 1)番目のメモリマップ管理情報*/
        {
                /* 省略 */
        }
};
```

TS_CONF_MMAP に記述されたメモリマップ管理情報は、記述された順番にメモリマップ番号が振られる。メモリマップ番号は 1～(TS_MAX_SDMN + TS_MAX_NDMN + 1)までの値となる。
後述のドメイン管理情報にて、ドメインが使用するメモリマップ番号が指定され、ドメインとメモリマップ管理情報は関連付けられる。

一つのメモリマップ管理情報には、TS_MAX_MRGN 個のメモリ領域情報が記述される。使用しないメモリ領域情報は

TSMA_NONE 属性を指定する。

以下にメモリマップ管理情報の記述例を示す。この例では、一つのドメインに3つのメモリ領域（コード領域、データ領域、スタック領域が一つずつ）が存在する場合を示す。

```
/* メモリマップ管理情報 */
#define TS_MAX_MRGN             3           /* メモリマップ管理情報中のメモリ領域の最大数*/

TS_CONF_MMAP  =   { /* メモリマップ管理情報の並び */
        /* 一番目のメモリマップ管理情報 */
        {
                {       /* コード領域 */
                        TSMA_CODE,          /* メモリ属性 */
                        0x01000000, 0x01FFFFFF          /* メモリ領域のアドレス */
                } ,
                {       /* データ領域 */
                        TSMA_DATA,          /* メモリ属性 */
                        0x02000000, 0x02FFFFFF          /* メモリ領域のアドレス */
                } ,
                {       /* スタック領域 */
                        TSMA_STACK,         /* メモリ属性 */
                        0x03000000, 0x03FFFFFF          /* メモリ領域のアドレス */
                }
        } ;
        /* 二番目のメモリマップ管理情報 */
        {
                /* 省略 */
        } ,

        /* 繰り返し(省略) */

        /* (TS_MAX_SDMN + TS_MAX_NDMN + 1)番目のメモリマップ管理情報 */
        {
                /* 省略 */
        }
};
```

7.3.3 スタック管理情報

スタック管理情報は、スタック領域に関するコンフィギュレーション情報である。

前項に記したように、スタック領域は各ドメインに一つ存在し、タスクおよびタイムイベントハンドラの実行時のスタックとして使用される。
各ドメインのスタック領域の用途を「表 7-5 スタック領域の用途」に示す。

表 7-5 スタック領域の用途

ドメイン種別	スタック領域の用途
安全ドメイン	安全ドメインに属するタスクおよびタイムイベントハンドラのユーザスタック
通常ドメイン	通常ドメインに属するタスクおよびタイムイベントハンドラのユーザスタック
システムドメイン	すべてのタスクおよびタイムイベントハンドラのシステムスタック

通常ドメインのユーザスタックは、タスクおよびタイムイベントハンドラの生成時に、スタック領域から個々のスタックにメモリが割り当てられる。よって、通常ドメインにはスタック管理情報は存在しない。

通常ドメインのユーザスタック以外は、スタック領域から個々のスタックが静的に確保される。タスクおよびタイムイベントハンドラの生成時に、個々のスタックが関係づけられる。スタック管理情報は、この静的なスタックの設定を記述する。
通常ドメイン以外のドメイン毎に一つのスタック管理情報が設定される。つまり、(TS_MAX_SDMN + 1)個のスタック管理情報が存在する。

スタック管理情報は、対応するドメインにおいて使用されるスタックのサイズを記述したものである。
一つのドメインにおいて使用されるスタックの最大数は、TS_MAX_STACK として定義する。よって、一つのスタック管理情報は、TS_MAX_STACK 個のスタックサイズの並びである。

スタック管理情報は、C 言語のマクロ定義(#define)と、定数型データ(const 型)にて各値が定義される。
スタック管理情報の記述形式を以下に示す。

```
#define TS_MAX_STACK            ドメイン毎のスタック数の最大値

TS_CONF_STACK = { /* スタック管理情報の並び */
        /* 一番目のスタック管理情報 */
        { スタックサイズの並び },
        /* 二番目のスタック管理情報 */
        { スタックサイズの並び },

        /* 繰り返し(省略) */

        /* (TS_MAX_SDMN + 1)番目のスタック管理情報 */
        { スタックサイズの並び }
}
```

スタック管理情報は、記述された順番にスタック管理番号が振られる。スタック管理番号は 1～(TS_MAX_SDMN + 1)までの値となる。
後述のドメイン管理情報にて、ドメインが使用するスタック管理番号が指定され、ドメインとスタック管理情報は関連付けられる。

スタック管理情報の中で記述されるスタックサイズは、記述された順番にスタック資源番号が振られる。スタック資源番号は 1～TS_MAX_STACK までの値となる。
実際に使用するスタックが TS_MAX_STACK に満たない場合は、使用しないスタックのスタックサイズの値を 0 とする。使用しないスタック(スタックサイズ=0)は、スタックサイズの並びの後方におく。
タスクまたはタイムイベントハンドラの生成時に、使用するスタック資源番号が指定され、タスクまたはタイムイベントハンドラとスタックが関係づけられる。

以下にスタック管理情報の記述例を示す。

```
/* スタック管理情報 */
#define TS_MAX_STACK            10          /* ドメイン毎のスタックの最大値*/

TS_CONF_STACK = { /* スタック管理情報の並び */
            /* 一番目のスタック管理情報 */
            { 1024, 1024, 1024, 1024, 512, 512, 256, 256, 0, 0 },
            /* 二番目のスタック管理情報 */
            { 512, 512, 512, 512, 512, 512, 256, 256, 256, 256 },

            /* 繰り返し(省略) */

            /* (TS_MAX_SDMN + 1)番目のスタック管理情報 */
            { 512, 512, 512, 512, 512, 512, 0, 0, 0, 0 },
}
```

7.3.4　メッセージバッファ管理情報

メッセージバッファは、通信のためのバッファ用メモリを使用する。バッファ用メモリは、システムドメインのデータ領域上に静的に確保される。メッセージバッファ管理情報は、このメッセージバッファのバッファ用メモリを確保するための情報を記述する。

メッセージバッファ管理情報は、バッファ用メモリの総メモリサイズと、メッセージバッファ毎のバッファ用メモリのサイズからなる。
メッセージバッファ毎のバッファ用メモリのサイズは、メッセージバッファの総数である(TS_MAX_MBF_SY + TS_MAX_MBF_SA + TS_MAX_MBF_NA) 個が存在する。

メッセージバッファ管理情報は、C 言語のマクロ定義(#define)と、定数型データ(const 型)にて各値が定義される。
メッセージバッファ管理情報の記述形式を以下に示す。

```
#define TS_TOTAL_MBFSZ        バッファ用メモリの総メモリサイズ

TS_CONF_MBFSZ = {             /* バッファ用メモリのサイズの並び */
        一番目のバッファ用メモリのサイズ（システムデフォルトの事象通知用メッセージバッファ用（安全 API）），
        二番目のバッファ用メモリのサイズ（システムデフォルトの事象通知用メッセージバッファ用（通常 API）），
        三番目のバッファ用メモリのサイズ ，

        /* 繰り返し(省略) */

        (TS_MAX_MBF_SY + TS_MAX_MBF_SA + TS_MAX_MBF_NA)番目のバッファ用メモリのサイズ ，
        }
```

バッファ用メモリのサイズは、記述された順番にメッセージバッファ資源番号が振られる。メッセージバッファ資源番号は 1～(TS_MAX_MBF_SY + TS_MAX_MBF_SA + TS_MAX_MBF_NA)までの値となる。
実際に使用するバッファ用メモリが(TS_MAX_MBF_SY + TS_MAX_MBF_SA + TS_MAX_MBF_NA)に満たない場合は、使用しないバッファ用メモリのサイズの値を 0 とする。使用しないバッファ用メモリ(サイズ=0)は、バッファ用メモリのサイズの並びの後方におく。

メッセージバッファの生成時に、メッセージバッファ資源番号が指定され、メッセージバッファとメッセージバッファ管理情報は関連付けられる。

一番目と二番目のバッファ用メモリのサイズには、デバイス管理機能のシステムデフォルトの事象通知用メッセージバッファのメモリのサイズを指定する。サイズには、TS_DEVT_MBFSZ_S と TS_DEVT_MBFSZ_N を指定する。事象通知用メッセージバッファの詳細は「4.1.7 デバイス事象通知」を参照。

以下にメッセージバッファ管理情報の記述例を示す。例では、メッセージバッファの総数(TS_MAX_MBF_SY + TS_MAX_MBF_SA + TS_MAX_MBF_NA) は6個とする。

```
/* メッセージバッファ管理情報 */
#define TS_TOTAL_MBFSZ

TS_CONF_MBFSZ = {               /* バッファ用メモリのサイズの並び */
        TS_DEVT_MBFSZ_S,  /* 一番目のバッファ用メモリのサイズ */
        TS_DEVT_MBFSZ_N,  /* 二番目のバッファ用メモリのサイズ */
        2048,       /* 三番目のバッファ用メモリのサイズ */
        512,        /* 四番目のバッファ用メモリのサイズ */
        512,        /* 五番目のバッファ用メモリのサイズ */
        0           /* 六番目のバッファ用メモリのサイズ */,
}
```

7.4 ドメイン管理情報

ドメイン管理情報は各ドメインの各種設定を指定する数値情報である。

ドメイン管理情報には以下がある。

- システムドメイン管理情報
- 安全ドメイン管理情報
- 通常ドメイン管理情報

7.4.1 ドメイン管理情報の概要

ドメインは、個々に「表 7-6 ドメイン管理情報一覧」に示すドメイン管理情報を持つ。

表 7-6 ドメイン管理情報一覧

名称	意味	値の範囲
TDmnAtr	ドメイン属性	※7
TMemMap	メモリマップ番号	1 ～ メモリマップ番号の最大値(※1)
TStackRes	スタック資源番号	1 ～ スタック資源番号の最大値(※2)
TMaxTskId	最大タスク数	1 ～ 最大設定可能数(※3)
TMaxSemId	最大セマフォ数	0 ～ 最大設定可能数(※3)
TMaxFlgId	最大イベントフラグ数	0 ～ 最大設定可能数(※3)
TMaxMtxId	最大ミューテックス数	0 ～ 最大設定可能数(※3)
TMaxMbfId	最大メッセージバッファ数	0 ～ 最大設定可能数(※3)
TMaxCycId	最大周期ハンドラ数	0 ～ 最大設定可能数(※3)
TMaxAlmId	最大アラームハンドラ数	0 ～ 最大設定可能数(※3)
TLstTskPri	指定可能な最低タスク優先度	ドメインにおける設定可能範囲(※4)
THstTskPri	指定可能な最高タスク優先度	ドメインにおける設定可能範囲(※4)
TMaxTskTim	タスク最大連続実行時間	1 ～ TS_MAX_TSKTIM
TMaxTehTim	タイムイベントハンドラ最大連続実行時間	1 ～ TS_MAX_TEHTIM
TMaxDmnTim	ドメイン最大連続実行時間	1 ～ TS_MAX_DMNTIM(※5)
TMaxDspTim	ディスパッチ禁止最大連続実行時間	1 ～ TS_MAX_DSPTIM
TIniTsk	初期タスク生成情報	※6

※1 ドメインで使用するメモリマップを指定する。メモリマップの詳細は「7.3.2 メモリマップ管理情報」を参照。

※2 ドメインで使用するスタック管理情報を指定する。ただし、通常ドメインにはスタック管理情報は存在しないので無視される。スタック管理情報の詳細は「7.3.3 スタック管理情報」を参照。

※3 最大設定可能数は、同一種別のドメインにおける合計値が、システム管理情報のオブジェクト情報の最大値以下でならなくてはならない。例えば、すべての安全ドメインの TMaxTskId の合計値は、システム管理情報の TS_MAX_TSK_SA の値以下でなければならない。

※4 アプリケーションのドメイン(安全ドメイン、通常ドメイン)の THstTskPri、TLstTskPri は、システム管理情報で規定した優先度の範囲でなければならない。システムドメインは、1～TS_LST_PRI の範囲が指定可能である。
THstTskPri (最高優先度)は、TLstTskPri (最低優先度)と同じ、または高い優先度(小さい値)でなければならない。

※5 TMaxDmnTim は、TMaxTskTim および TMaxTehTim より大きな値でなければならない。

※6 タスク生成 API のパラメータ(T_CTSK)に準ずる。ただし、システムドメインにはコンフィギュレーションで定義すべき初期タスクは存在しないので無視される。T_CTSK の詳細は「8.2.1.1ts_cre_tsk/tn_cre_tsk － タスク生成」を参照。

※7 ドメインの属性を指定する。指定可能な属性は、起動時実行属性(TA_START)のみである。また、起動時実行属性は、安全ドメインでのみ有効であり、システムドメインと通常ドメインでは無視する。

7.4.2 システムドメイン管理情報

システムドメインはただ一つ存在する。よって、システムドメイン管理情報の内容は、ドメイン情報と同一である。

システムドメイン管理情報は、C 言語の定数型データ(const 型)として定義される。

システムドメイン管理情報の記述形式を以下に示す。

```
SYS_DMN_CONF    = { /* システムドメイン管理情報 */
        .TDmnAtr        = ドメイン属性
        .TMemMap        = メモリマップ番号
        .TStackRes      = スタック資源番号
        .TMaxTskId      = 最大タスク数,
        .TMaxSemId      = 最大セマフォ数,
        .TMaxFlgId      = 最大イベントフラグ数,
        .TMaxMtxId      = 最大ミューテックス数,
        .TMaxMbfId      = 最大メッセージバッファ数,
        .TMaxCycId      = 最大周期ハンドラ数,
        .TMaxAlmId      = 最大アラームハンドラ数,
        .TLstTskPri     = 指定可能な最低タスク優先度 ,
        .THstTskPri     = 指定可能な最高タスク優先度,
        .TMaxTskTim     = タスク最大連続実行時間,
        .TMaxTehTim     = タイムイベントハンドラ最大連続実行時間,
        .TMaxDmnTim     = ドメイン最大連続実行時間
        .TMaxDspTim     = ディスパッチ禁止最大連続実行時間
        .TIniTsk        = 初期タスク生成情報(システムドメインは無効)
};
```

システムドメインにはコンフィギュレーションで定義すべき初期タスクが存在しないので初期タスク生成情報は無視される。

システムドメインでは、システム起動時に必ず実行されるので、ドメイン属性の起動時実行属性は無視される。

以下にシステムドメイン管理情報の記述例を示す。

```
/* システムドメイン管理情報 */
SYS_DMN_CONF = {
        .TDmnAtr        = (ATR)0, /* ドメイン属性 */
        .TMemMap        = 1,      /* メモリマップ番号 */
        .TStackRes      = 1,      /* スタック資源番号 */
        .TMaxTskId      = 20,     /* 最大タスク数*/
        .TMaxSemId      = 10,     /* 最大セマフォ数 */
        .TMaxFlgId      = 5,      /* 最大イベントフラグ数 */
        .TMaxMtxId      = 5,      /* 最大ミューテックス数 */
        .TMaxMbfId      = 5,      /* 最大メッセージバッファ数 */
        .TMaxCycId      = 5,      /* 最大周期ハンドラ数 */
        .TMaxAlmId      = 5,      /* 最大アラームハンドラ数 */
        .TLstTskPri     = 250,    /* 指定可能な最低タスク優先度 */
        .THstTskPri     = 1,      /* 指定可能な最高タスク優先度 */
        .TMaxTskTim     = 500,    /* タスク最大連続実行時間 */
        .TMaxTehTim     = 500,    /* タイムイベントハンドラ最大連続実行時間 */
        .TMaxDmnTim     = 1000,   /* ドメイン最大連続実行時間 */
        .TMaxDspTim     = 1000,   /* ディスパッチ禁止最大連続実行時間 */
        .TIniTsk        =         /* 初期タスク生成情報（システムドメインでは無効） */
                        { (ATR)0, (FP)0, (PRI)0, (RELTIM)0, {0,0,0,0,0,0,0,0}}
};
```

7.4.3　安全ドメイン管理情報

安全ドメインはシステム管理情報で定義した TS_MAX_SDMN の数だけ存在する。よって、安全ドメイン管理情報は TS_MAX_SDMN 個のドメイン情報の並びとなる。

安全ドメイン管理情報は、C 言語の定数型データ(const 型)として定義される。
安全ドメインの管理情報の記述形式を以下に示す。

```
SAFE_DMN_CONF = { /* 安全ドメイン管理情報 */
        /* 一番目の安全ドメインのドメイン情報 */
        {
                .TDmnAtr        = ドメイン属性
                .TMemMap        = メモリマップ番号
                .TStackRes      = スタック資源番号
                .TMaxTskId      = 最大タスク数,
                .TMaxSemId      = 最大セマフォ数,
                .TMaxFlgId      = 最大イベントフラグ数,
                .TMaxMtxId      = 最大ミューテックス数,
                .TMaxMbfId      = 最大メッセージバッファ数,
                .TMaxCycId      = 最大周期ハンドラ数,
                .TMaxAlmId      = 最大アラームハンドラ数,
                .TLstTskPri     = 指定可能な最低タスク優先度 ,
                .THstTskPri     = 指定可能な最高タスク優先度,
                .TMaxTskTim     = タスク最大連続実行時間,
                .TMaxTehTim     = タイムイベントハンドラ最大連続実行時間,
                .TMaxDmnTim     = ドメイン最大連続実行時間
                .TMaxDspTim     = ディスパッチ禁止最大連続実行時間
                .TIniTsk        = 初期タスク生成情報
        },
        /* 二番目の安全ドメインのドメイン情報 */
        {
                /* 省略 */
        },

        /* 繰り返し(省略) */

        /* TS_MAX_SDMN 番目の安全ドメインのドメイン情報 */
        {
                /* 省略 */
        }
};
```

安全ドメイン管理情報に記述された順番に、ドメイン情報に対応するドメイン ID が割り振られる。安全ドメインの ID は 2 から始まるので、最大の安全ドメイン ID は TS_MAX_SDMN＋1となる。

以下に安全ドメイン管理情報の記述例を示す。

```
/* 安全ドメイン管理情報 */
SAFE_DMN_CONF = {
        /* 一番目の安全ドメインのドメイン情報 */
        {
                .TDmnAtr        = TA_START,  /* ドメイン属性 */
                .TMemMap        = 2,      /* メモリマップ番号 */
                .TStackRes      = 2,      /* スタック資源番号 */
                .TMaxTskId      = 20,     /* 最大タスク数*/
                .TMaxSemId      = 10,     /* 最大セマフォ数 */
                .TMaxFlgId      = 5,      /* 最大イベントフラグ数 */
                .TMaxMtxId      = 5,      /* 最大ミューテックス数 */
                .TMaxMbfId      = 5,      /* 最大メッセージバッファ数 */
                .TMaxCycId      = 5,      /* 最大周期ハンドラ数 */
                .TMaxAlmId      = 5,      /* 最大アラームハンドラ数 */
                .TLstTskPri     = 100,    /* 指定可能な最低タスク優先度 */
                .THstTskPri     = 16,     /* 指定可能な最高タスク優先度 */
                .TMaxTskTim     = 500,    /* タスク最大連続実行時間 */
                .TMaxTehTim     = 500,    /* タイムイベントハンドラ最大連続実行時間 */
                .TMaxDmnTim     = 1000,   /* ドメイン最大連続実行時間 */
                .TMaxDspTim     = 1000,   /* ディスパッチ禁止最大連続実行時間 */
                .TIniTsk        =         /* 初期タスク生成情報 */
                {
                        (ATR)0,           /* タスク属性 */
                        (FP)SDTSK_MAIN,   /* タスク起動アドレス */
                        (PRI)16,          /* タスク起動時優先度 */
                        (RELTIM)100,      /* 最大実行時間 */
                        /* オブジェクト名 */
                        {'I','N','I','T','A','S','K',0}
                }
        },
        /* 二番目の安全ドメインのドメイン情報 */
         {
                /* 省略 */
        },

        /* 繰り返し(省略) */

        /* TS_MAX_SDMN 番目の安全ドメインのドメイン情報 */
         {
                /* 省略 */
        }
};
```

7.4.4　通常ドメイン管理情報

通常ドメインはシステム管理情報で定義した TS_MAX_NDMN の数だけ存在する。よって、通常ドメイン管理情報は TS_MAX_NDMN 個のドメイン情報の並びとなる。

通常ドメイン管理情報は、C 言語の定数型データ(const 型)として定義される。
通常ドメインの管理情報の記述形式を以下に示す。

```
NORM_DMN_CONF = {/* 通常ドメイン管理情報 */
        /* 一番目の通常ドメインのドメイン情報 */
        {
                /* 省略(安全ドメインと同じ) */
        },
        /* 二番目の通常ドメインのドメイン情報 */
        {
                /* 省略 */
        },

        /* 繰り返し(省略) */

        /* TS_MAX_NDMN 番目の通常ドメインのドメイン情報 */
        {
                /* 省略 */
        }
};
```

通常ドメイン管理情報に記述された順番に、ドメイン情報に対応するドメイン ID が割り振られる。通常ドメインの ID は安全ドメインの最大 ID の次の値から始まるので、(TS_MAX_SDMN＋2)から始まる。最大の通常ドメイン ID は(TS_MAX_SDMN＋TS_MAX_NDMN + 1)となる。
通常ドメインでは、システムドメインもしくは安全ドメインのタスクから起動するので、ドメイン属性の起動時実行属性は無視される。

以下に通常ドメイン管理情報の記述例を示す。

```
/* 通常ドメイン管理情報 */
NORM_DMN_CONF = {
        /* 一番目の通常ドメインのドメイン情報 */
        {
                .TDmnAtr        = (ATR)0, /* ドメイン属性 */
                .TMemMap        = 3,      /* メモリマップ番号 */
                .TStackRes      = 0,      /* スタック資源番号(通常ドメインでは無効) */
                .TMaxTskId      = 20,     /* 最大タスク数*/
                .TMaxSemId      = 10,     /* 最大セマフォ数 */
                .TMaxFlgId      = 5,      /* 最大イベントフラグ数 */
                .TMaxMtxId      = 5,      /* 最大ミューテックス数 */
                .TMaxMbfId      = 5,      /* 最大メッセージバッファ数 */
                .TMaxCycId      = 5,      /* 最大周期ハンドラ数 */
                .TMaxAlmId      = 5,      /* 最大アラームハンドラ数 */
                .TLstTskPri     = 256,    /* 指定可能な最低タスク優先度 */
                .THstTskPri     = 50,     /* 指定可能な最高タスク優先度 */
                .TMaxTskTim     = 500,    /* タスク最大連続実行時間 */
                .TMaxTehTim     = 500,    /* タイムイベントハンドラ最大連続実行時間 */
                .TMaxDmnTim     = 1000,   /* ドメイン最大連続実行時間 */
                .TMaxDspTim     = 1000,   /* ディスパッチ禁止最大連続実行時間 */
                .TIniTsk        =         /* 初期タスク生成情報 */
                {
                        (ATR)0,           /* タスク属性 */
                        (FP)NMTSK_MAIN,   /* タスク起動アドレス */
                        (PRI)50,          /* タスク起動時優先度 */
                        (RELTIM)100,      /* 最大実行時間 */
                        /* オブジェクト名 */
                        {'I','N','I','T','A','S','K',0}
                }
        },
        /* 二番目の通常ドメインのドメイン情報 */
         {
                /* 省略 */
        },

        /* 繰り返し(省略) */

        /* TS_MAX_NDMN 番目の安全ドメインのドメイン情報 */
         {
                /* 省略 */
        }
};
```

7.5 ユーザ定義システムソフトウェア登録情報

ユーザ定義システムソフトウェア登録情報は、TRON Safe Kernel の初期状態で登録されるユーザ定義システムソフトウェアの情報である。初期登録 API により記述される。

7.5.1 ユーザ定義システムソフトウェア登録情報の初期登録 API

ユーザ定義ソフトウェア登録情報に記述される初期登録 API の一覧を「表 7-7 ユーザ定義システムソフトウェア登録情報の初期登録 API 一覧」に示す。

表 7-7 ユーザ定義システムソフトウェア登録情報の初期登録 API 一覧

分類	API 名	機能
初期登録 API	TS_DEF_DEV	デバイスドライバの登録
	TS_DEF_INT	割込みハンドラの登録
	TS_DEF_SSY	サブシステムの登録
	TS_DEF_FDH	故障診断ハンドラの登録
	TS_DEF_AEH	異常例外ハンドラの登録

各初期登録 API の説明を以降に示す。

7.5.1.1 TS_DEF_DEV – デバイスドライバの登録

C 言語インタフェース

【初期登録 API】 void TS_DEF_DEV(CONST T_DDEV *pk_ddev);

パラメータ

CONST T_DDEV* pk_ddev デバイスドライバ登録情報

デバイスドライバ登録情報 typedef struct st_ddev {

```
    UB      devnm[8];       /* 物理デバイス名 */
    ATR     drvatr;         /* デバイスドライバ属性 */
    INT     nsub;           /* サブユニット数 */
    INT     blksz;          /* 固有データのブロックサイズ(-1:不明) */
    FP      initfn,         /* 初期化関数アドレス */
    FP      openfn;         /* オープン関数アドレス */
    FP      closefn;        /* クローズ関数アドレス */
    FP      execfn;         /* 処理開始関数アドレス */
    FP      waitfn;         /* 完了待ち関数アドレス */
    FP      cancelfn;       /* 要求無効関数アドレス */
    FP      abortfn;        /* 中止処理関数アドレス */
    FP      eventfn;        /* イベント関数アドレス */
} T_DDEV;
```

解説

pk_ddev で指定された値に従いデバイスドライバの登録を行う。pk_ddev に NULL(0)を指定した場合はエラーとする。

dennm は登録するデバイスドライバの物理デバイス名を指定する。デバイス名の命名規則は「4.1.4 デバイス名」に従う。
devatr は登録するデバイスドライバのデバイスドライバ属性を指定する。デバイスドライバ属性は「4.1.2 デバイスドライバ属性」の規定に従う。
nsub はデバイスドライバのサブユニット数を指定する。サブユニット数は 0 ～ 254 の値が指定可能である。ただし、実際に有効なサブユニット数はデバイスドライバ毎に規定される。
blksz は、固有データのブロックサイズをバイト数で設定する。ディスクデバイスの場合は、物理ブロックサイズとなる。シリアル回線などは1バイトとなる。固有データのないデバイスでは 0 とする。未フォーマットのディスクなど、ブロックサイズが不明の場合は-1 とする。blksz≦0 の場合は、固有データにアクセスできない。ts/tn_rea_dev, ts/tn_wri_dev で固有データをアクセスする場合に、size * blksz がアクセスする領域サイズ、つまり buf のサイズとならなければならない。
initfn、openfn、closefn、execfn、waitfn、cancelfn、abortfn、eventfn は、登録するデバイスドライバのドライバインタフェース関数の実行開始アドレスを設定する。デバイスドライバ関数の詳細は「4.1.5 デバイスドライバ・インタフェース」に記す。

デバイスドライバが登録されると、初期登録 API が呼ばれた順番にデバイス ID が割当てられる。デバイスドライバの ID は 1 から始まる。最大のデバイス ID は登録可能なデバイスドライバの合計であり、(TS_MAX_DEV_A + TS_MAX_DEV_N)となる。

登録されたデバイスドライバの操作は、TRON Safe Kernel の起動後にデバイス管理機能の API を用いて操作することができる。基本的な操作は、本初期登録 API で設定したデバイス名を用いて指定する。また、デバイス ID を含むデバイスの各種情報はデバイス情報取得 API(ts_ref_dev/tn_ref_dev)で取得できる。

本 API は戻り値を返さない。実行中にエラーがあった場合は異常例外を発生する。

記述例

```
TS_DEF_DEV ( &(T_DEV) {
        { 'i', 'o', 'p', 'a', 0, 0, 0, 0 },   /* デバイス名 */
        TDA_SEV | TDA_OPENREQ ,               /* デバイス属性 */
        0 ,                                   /* サブユニット数 */
        1,                                    /* 固有データのブロックサイズ*/
        fp_initfn;                            /* 初期化関数アドレス */
        fp_openfn;                            /* オープン関数アドレス */
        fp_closefn;                           /* クローズ関数アドレス */
        fp_execfn;                            /* 処理開始関数アドレス */
        fp_waitfn;                            /* 完了待ち関数アドレス */
        fp_cancelfn;                          /* 要求無効関数アドレス */
        fp_abortfn;                           /* 中止処理関数アドレス */
        fp_eventfn;                           /* イベント関数アドレス */
}) ;
```

なお、デバイスインタフェース関数は別途定義されているものとする。

7.5.1.2 TS_DEF_INT – 割込みハンドラの登録

C 言語インタフェース

【初期登録 API】 void TS_DEF_INT(UINT dintno, CONST T_DINT* pk_dint);

パラメータ

UINT	dintno	割込みハンドラ番号
CONST T_DINT*	pk_dint	割込みハンドラ定義情報

割込みハンドラ定義情報

```
typedef struct st_dint {
        ATR     intatr;         /* 割込みハンドラ属性 */
        FP      inthdr;         /* 割込みハンドラ起動アドレス */
        INT     priority;       /* 実行優先度 */
        RELTIM  maxrtim;        /* 最大連続実行時間 */
} T_DINT;
```

解説

dintno で指定される割込みハンドラ番号に対して割込みハンドラを登録する。

本初期登録 API の動作は、割込みハンドラの登録 API(ts_def_int)と同等である。ただし、以下の点は異なる。

- 初期登録 API は戻り値を返さない。実行時にエラーが発生した場合は異常例外を発生する。起こりうるエラーは、ts_def_int と同等である。
- 初期登録 API では、登録済みの割込みハンドラの再登録はできない。登録済みの割込みハンドラ番号を指定した場合はエラーとする。
- 初期登録 API では、登録済みの割込みハンドラの解除はできない。pk_dint に NULL(0)を指定した場合はエラーとする。

割込みハンドラの登録 API(ts_def_int)の詳細は「8.3.3.1 ts_def_int – 割込みハンドラの登録」を参照。

記述例

```
TS_DEF_INT ( DINT_1, &(T_DINT){ TIA_MLTINT, inthdr, 10 , 100 } );
```

なお、以下が定義されているものとする。

```
#define     DINT_1              1       /* 割込みハンドラ番号 */
void        int1_hdr( UINT );           /* 割込みハンドラ */
```

7.5.1.3 TS_DEF_SSY – サブシステム登録

C 言語インタフェース

【初期登録 API】 void TS_DEF_SSY(ID ssid, CONST T_DSSY* pk_dssy);

パラメータ

ID	ssid	サブシステム ID
CONST T_DSSY*	pk_dssy	サブシステム定義情報へのポインタ

サブシステム定義情報

```
typedef struct st_dssy {
        ATR     ssyatr;         /* サブシステム属性 */
        PRI     ssypri;         /* サブシステム優先度 */
        INT     svcnum;         /* 機能コード数 */
        FP      svchdr_s;       /* 安全 API 用拡張 SVC 起動アドレス */
        FP      svchdr_n;       /* 通常 API 用拡張 SVC 起動アドレス */
        FP      initfn;         /* 初期化関数アドレス */
        FP      startupfn;      /* スタートアップ関数アドレス */
        FP      cleanupfn;      /* クリーンアップ関数アドレス */
        FP      eventfn;        /* イベント処理関数アドレス */
        FP      breakfn;        /* ブレーク関数アドレス */
} T_DSSY;
```

解説

ssid で指定されるサブシステム ID でサブシステムを登録する。

サブシステム ID は、1～TS_MAX_SSY の値が指定できる。他のサブシステムと重複してはならない。既に登録済みのサブシステム ID が指定された場合は、エラーとして異常例外を発生する。

pk_dssy で登録するサブシステムの情報を指定する。

ssyatr はサブシステム属性を指定する。詳細は「4.3.4 サブシステムの属性」に記す。

ssypri はサブシステム優先度を指定する。1～TS_LST_SSYPRI の値を指定することができる。

svcnum はサブシステムが提供する機能コードの数を指定する。

svchdr_s は安全 API 用拡張 SVC の起動アドレス、svchdr_n は通常 API 用拡張 SVC の起動アドレスを指定する。拡張 SVC ハンドラの詳細は「4.3.5 拡張 SVC ハンドラ」に記す。svchdr_s と svchdr_n に同じ起動アドレスが指定された場合は、エラーとして異常例外を発生する。

initfn、startupfn、cleanupfn、eventfn、breakfn は、サブシステム I/F 関数の起動アドレスを指定する。サブシステム I/F 関数の詳細は「4.3.6 サブシステム・インタフェース」に記す。

登録されたサブシステムに対して、TRON Safe Kernel の起動後にサブシステム管理機能の API を用いて拡張 SVC を発行することができる。

本 API は戻り値を返さない。実行中にエラーがあった場合は異常例外を発生する。

記述例

```
TS_DEF_SSY ( SSID_1,
        &(T_DSSY){
                TSA_SSSY ,          /* サブシステム属性 */
                32 ,                /* サブシステム優先度 */
                10 ,                /* 機能コード数 */
                fp_svchdr ,         /* 安全 API 用拡張 SVC 起動アドレス */
                FP(0) ,             /* 通常 API 用拡張 SVC 起動アドレス */
                fp_initfn ,         /* 初期化関数アドレス */
                fp_tartupfn ,       /* スタートアップ関数アドレス */
                fp_cleanupfn,       /* クリーンアップ関数アドレス */
                fp_eventfn;         /* イベント処理関数アドレス */
                fp_breakfn;         /* ブレーク関数アドレス */
        }
);
```

なお、以下が定義されているものとする。

```
#define     SSID_1              1       /* サブシステム ID */
```

また、各拡張 SVC ハンドラ、サブシステム I/F 関数は別途定義されているものとする。

7.5.1.4 TS_DEF_FDH – 故障診断ハンドラ登録

C 言語インタフェース

【初期登録 API】 void TS_DEF_FDH(ID fdhid, CONST T_DFDH* pk_dfdh);

パラメータ

ID	fdhid	故障診断ハンドラ ID
CONST T_DFDH*	pk_dfdh	故障診断ハンドラ定義情報へのポインタ

故障診断ハンドラ定義情報

```
typedef struct st_dfdh {
        ATR     fdhatr;         /* 故障診断ハンドラ属性 */
        PRI     fdhpri;         /* 故障診断ハンドラの実行優先度 */
        RELTIM  fdhcyc;         /* 故障診断ハンドラを起動する間隔 */
        FP      fdhdr;          /* 故障診断ハンドラの起動アドレス */
        RELTIM  maxrtim;        /* 最大連続実行時間 */
} T_DFDH;
```

解説

fdhid で指定される故障診断ハンドラ ID で故障診断ハンドラを登録する。

故障診断ハンドラ ID は、1～TS_MAX_FDH の値が指定できる。他の故障診断ハンドラと重複してはならない。既に登録済みの故障診断ハンドラ ID が指定された場合は、エラーとして異常例外を発生する。

pk_dfdh で登録する故障診断ハンドラの情報を指定する。

fdhatr は故障診断ハンドラ属性を指定する。詳細は「4.4.3 故障診断ハンドラの属性」に記す。

fdhpri は故障診断ハンドラの実行優先度を指定する。タスク優先度の範囲の値を指定することができる。

fdhcycは故障診断ハンドラを起動する間隔時間を指定する。1 ～ TS_MAX_FDHCYC の値が指定できる。

fdhdr は故障診断ハンドラの起動アドレスを指定する。詳細は「4.4.6 故障診断ハンドラの実行プログラム」に記す。

maxrtim は故障診断ハンドラが連続実行できる時間の上限を指定する。1 ～ TS_MAX_DMNTIM の値が指定できる。詳細は「4.4.7 故障診断ハンドラの時間保護」に記す。

登録された故障診断ハンドラは、TRON Safe Kernel の起動後に、指定されたタイミングで実行される。詳細は「4.4.5 故障診断ハンドラの起動」に記す。

本 API は戻り値を返さない。実行中にエラーがあった場合は異常例外を発生する。

記述例

```
TS_DEF_FDH( FDHID_1,
        &(T_DFDH){
                TFA_CYC ,       /* 故障診断ハンドラ属性 */
                16 ,            /* 故障診断ハンドラの実行優先度 */
```

```
                100 ,               /* 故障診断ハンドラを起動する間隔 */
                fp_fdhdr,           /* 故障診断ハンドラの起動アドレス */
                1000                /* 最大連続実行時間 */
        }
);
```

なお、以下が定義されているものとする。

```
#define     FDHID_1             1       /* 故障診断ハンドラ ID */
```

また、故障診断ハンドラの実行関数は別途定義されているものとする。

7.5.1.5 TS_DEF_AEH – 異常例外ハンドラ登録

C 言語インタフェース

【初期登録 API】 void TS_DEF_AEH(UINT aexpno, CONST T_DAEH* pk_daeh);

パラメータ

UINT	aexpno	異常例外番号
CONST T_DAEH*	pk_daeh	異常例外ハンドラ定義情報へのポインタ

異常例外ハンドラ定義情報

```
typedef struct st_daeh {
        ATR     aehatr;         /* 異常例外ハンドラ属性 */
        FUNCP   aehdr;          /* 異常例外ハンドラの起動アドレス */
} T_DAEH;
```

解説

aexpno で指定される異常例外番号に対して異常例外ハンドラを登録する。

本初期登録 API では、登録済みの異常例外ハンドラの解除、再登録はできない。登録済みの異常例外番号を指定した場合はエラーとする。

aehatr は異常例外ハンドラ属性を指定する。詳細は「6.2.4 異常例外ハンドラの属性」に記す。

aehdr は異常例外ハンドラの起動アドレスを指定する。詳細は「6.2.6 異常例外ハンドラの実行プログラム」に記す。

本 API は戻り値を返さない。実行中にエラーがあった場合は異常例外を発生する。

記述例

```
TS_DEF_AEH( AEXP_DIVZ,          /* 異常例外番号 */
        &(T_DAEH){
                TAA_RETURN ,    /* 異常例外ハンドラ属性 */
                fp_aehdr,       /* 異常例外ハンドラの起動アドレス */
        }
);
```

なお、異常例外ハンドラの実行関数は別途定義されているものとする。

8. API 仕様

8.1 安全 API と通常 API の相違

「2.4.2 安全 API」及び「2.4.3 通常 API」に示す。

8.2 TRON Safe Kernel/OS の API

8.2.1 タスク管理機能の API

タスク管理機能の詳細については、「3.1 タスク管理機能」を参照。

8.2.1.1 ts_cre_tsk/tn_cre_tsk – タスク生成

C 言語インタフェース

【安全 API】 ID tskid = ts_cre_tsk(CONST T_CTSK *pk_ctsk);

【通常 API】 ID tskid = tn_cre_tsk(CONST T_CTSK *pk_ctsk);

パラメータ

CONST T_CTSK*	pk_ctsk	タスク生成情報へのポインタ

タスク生成情報

```
typedef struct st_ctsk {
        ATR     tskatr;         /* タスク属性 */
        FP      task;           /* タスク起動アドレス */
        PRI     itskpri;        /* タスク起動時優先度 */
        UINT    ustkno;         /* ユーザスタックの資源番号 */
        INT     ustksz;         /* ユーザスタックサイズ（バイト数）*/
        UINT    sstkno;         /* システムスタックの資源番号 */
        INT     sstksz;         /* システムスタックサイズ（バイト数）*/
        RELTIM  maxrtim;        /* 最大連続実行時間 */
        UB      oname[8];       /* オブジェクト名称 */
} T_CTSK;
```

リターンパラメータ

ID	tskid	Task ID	タスク ID
	または	Error Code	エラーコード

エラーコード

E_NOMEM	メモリ不足（指定したサイズのスタックがスタック用領域から確保できない）
E_LIMIT	タスクの数がシステムの制限を超えた
E_RSATR	予約属性エラー（tskatr が不正あるいは利用できない）、指定のコプロセッサは存在しない
E_PAR	パラメータエラー（pk_ctsk, task, maxrtim で指定されたアドレス、値が不正）
E_NOCOP	指定のコプロセッサが使用できない（動作中のハードウェアには搭載されていない、または動作異常が検出された）
E_CTX	コンテキストエラー（タスク部および準タスク部以外で実行）
E_ONAME	オブジェクト名エラー（指定されたオブジェクト名が既にドメイン内で使用されている）
E_MACV	メモリアクセス違反エラー（pk_ctsk、task に対するアクセス権限がない、または対象のメモリ領域が存在しない、または ustkno、sstkno で指定したメモリ領域が自ドメインのデータ領域でない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

タスクを生成してタスク ID 番号を割り当てる。

対象タスクは生成後、休止状態(DORMANT)となる。

tskatr には、対象タスクの動作を指定するためのタスク属性を設定する。

tskatr には、次のような指定を行う。

tskatr := [TA_ONAME]| [TA_COP0] | [TA_COP1] | [TA_COP2] | [TA_COP3] | [TA_FPU]

TA_ONAME	オブジェクト名称を指定する
TA_COPn	対象タスクが第 n 番目のコプロセッサを使用する （浮動小数点演算用コプロセッサや DSP を含む）
TA_FPU	対象タスクが浮動小数点演算用コプロセッサを使用する （TA_COPn による指定の内、特に浮動小数点演算を使用するための、CPU に依存しない汎用的な指定である）

```
#define TA_ONAME        (0x00000040U)    /* オブジェクト名称を指定 */
#define TA_COP0         (0x00001000U)    /* ID=0 のコプロセッサを使用 */
#define TA_COP1         (0x00002000U)    /* ID=1 のコプロセッサを使用 */
#define TA_COP2         (0x00004000U)    /* ID=2 のコプロセッサを使用 */
#define TA_COP3         (0x00008000U)    /* ID=3 のコプロセッサを使用 */
```

タスクは C 言語の関数として以下の形式で記述する。

```
void    task_main( INT stacd );
```

上記形式の関数へのポインタを、タスク起動アドレス task に指定する。

タスクの起動パラメータとして、ts_sta_tsk/tn_sta_tsk で指定するタスク起動コードが stacd に渡される。

関数からの単純なリターン(return)でタスクを終了することはできない(してはいけない)。必ず、ts_ext_tsk/tn_ext_tsk または ts_exd_tsk/tn_exd_tsk を発行してタスクを終了する。

タスクの関数を単純なリターン(return)で終了した場合は、対象タスクを終了(すなわち ts_ext_tsk/tn_ext_tsk を発行したのと同じことになる)した後、異常例外を発生させる。

itskpri には、タスクが起動する時の優先度の初期値を指定する。タスク優先度としては、1～250 の値を指定することができ、値の小さい方が高い優先度となる。実際に指定可能な優先度は、コンフィギュレーションによって以下のように決定される。

- システムドメインのタスクで指定可能な優先度は、1～TS_LST_PRI とする。
- 安全ドメインのタスクで指定可能な優先度は、TS_HST_DPRI_SA～TS_LST_DPRI_SA の範囲の中で、ドメイン毎に定められる。
- 通常ドメインのタスクで指定可能な優先度は、TS_HST_DPRI_NA～TS_LST_DPRI_NA の範囲の中で、ドメイン毎に定められる。

itskpri に自ドメインの最高優先度よりも高い優先度を指定した場合はエラーE_PAR となる。

タスクはユーザスタックとシステムスタックを持つ。ただし、システムドメインのタスクは、システムスタックのみであり、ユーザスタックはもたない。よって、システムドメインのタスク生成では、ユーザスタックの指定は無視される。
ユーザスタックは、タスクが所属するドメインのスタック用領域からメモリが確保される。ustkno にコンフィギュレーションで設定したスタック資源番号を指定する。ustksz にユーザスタックのサイズを指定する。ただし、通常ドメインではメモリ領域は動的に確保するので、通常 API では utskno の値は無視される。安全 API では ustksz は、ustkno で指定したスタック資源のメモリサイズ以下でなければならない。
システムスタックは、システムドメインのスタック用領域からメモリが確保される。sstkno にコンフィギュレーションで設定したスタック資源番号を指定する。sstksz にシステムスタックのサイズを指定する。sstksz は、sstkno で指定したスタック資源のメモリサイズ以下でなければならない。
以下の理由でスタックが確保できなかった場合、エラーE_NOMEM となる。
- 指定したスタックサイズが、スタック資源のサイズより大きい
- 指定したスタック資源が既にタスクまたはタイムイベントハンドラで使用されている
- 通常 API においてユーザスタックが確保できなかった（ユーザスタック領域の残りサイズが不足）

maxrtim には、タスクが連続して実行できる最大時間をミリ秒単位で設定する。maxrtim の最小値は 1 ミリ秒である。maxrtim の最大値は自ドメインの連続実行時間上限の値である。
maxrtim が自ドメインの連続実行時間上限より大きな値を指定した場合はエラーE_PAR となる。
タスクが maxrtim を超えて連続実行した場合は異常例外が発生し、タスクの実行は中断される。

TA_ONAME を指定した場合に oname が有効となる。
oname には、対象タスクのオブジェクト名称を設定する。TA_ONAME が指定されてない場合は、oname は無視される。

【補足事項】

TA_COPn の定義は、CPU などのハードウェアに依存して決められるため移植性はない。
TA_FPU は、TA_COPn の定義の内、浮動小数点演算の使用に関してのみ移植性のある指定方法として用意される。例えば、浮動小数点コプロセッサが TA_COP0 の場合は、TA_FPU=TA_COP0 となる。浮動小数点演算を行うのに特にコプロセッサの使用を指定する必要がない場合は、TA_FPU=0 となる。

8.2.1.2 ts_del_tsk/tn_del_tsk – タスク削除

C 言語インタフェース

【安全 API】 ER ercd = ts_del_tsk(ID tskid);

【通常 API】 ER ercd = tn_del_tsk(ID tskid);

パラメータ

ID	tskid	タスク ID

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（tskid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（tskid のタスクが存在しない）
E_OBJ	オブジェクトの状態が不正（対象タスクが休止状態(DORMANT)でない）
E_CTX	コンテキストエラー（タスク部および準タスク部以外で実行）
E_DACV	ドメインアクセス保護違反（対象タスクは他のドメインに所属してアクセス保護されている）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

tskid で指定されたタスクを削除する。

具体的には、tskid で指定されたタスクを休止状態(DORMANT)から未登録状態(NON-EXISTENT)へと移行させる。

休止状態(DORMANT)でないタスクに対して本 API を実行すると、E_OBJ のエラーとなる。

本 API で自タスクの指定はできない。自タスクを指定した場合には、自タスクが休止状態(DORMANT)ではないため、E_OBJ のエラーとなる。自タスクを削除するには、本 API ではなく、ts_exd_tsk/tn_exd_tsk を発行する。

8.2.1.3 ts_sta_tsk/tn_sta_tsk – タスク起動

C 言語インタフェース

【安全 API】 ER ercd = ts_sta_tsk(ID tskid, INT stacd);

【通常 API】 ER ercd = tn_sta_tsk(ID tskid, INT stacd);

パラメータ

ID	tskid	タスク ID
INT	stacd	タスク起動コード

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（tskid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（tskid のタスクが存在しない）
E_OBJ	オブジェクトの状態が不正（対象タスクが休止状態(DORMANT)でない）
E_DACV	ドメインアクセス保護違反（対象タスクは他のドメインに所属してアクセス保護されている）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

tskid で指定されたタスクを起動する。
具体的には、対象タスクを休止状態(DORMANT)から実行可能状態(READY)へと遷移させる。対象タスクが実行状態(RUNNING)となると、タスクのメイン関数から実行が開始される。

stacd には、タスクの起動時にタスクに渡すパラメータを設定する。このパラメータは、タスクのメイン関数の引数に渡される。

タスク起動時のタスク優先度は、対象タスクが生成された時に指定されたタスク起動時優先度(itskpri)となる。ただし、生成された後の休止状態の間に ts_chg_pri/tn_chg_pri でタスク優先度が変更された場合はその優先度となる。

本 API による起動要求はキューイングされない。すなわち、対象タスクが休止状態(DORMANT)でないのに本 API が発行された場合、発行タスクに E_OBJ のエラーが返る。

タスクが休止状態の間に設定された以下の設定は、タスクを起動する時にクリアされない。

- タスク優先度（タスク起動時優先度）
- 待ちを禁止されている待ち要因

8.2.1.4 ts_ext_tsk/tn_ext_tsk – 自タスク終了

C 言語インタフェース

【安全 API】 void ts_ext_tsk(void);

【通常 API】 void tn_ext_tsk(void);

パラメータ

なし

リターンパラメータ

なし(API を発行したコンテキストには戻らない)

エラーコード

なし(エラーを検出した場合は異常例外を発生)

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

自タスクを正常終了させ、休止状態(DORMANT)へと移行させる。

タスクが休止状態(DORMANT)に戻る時は、タスク優先度など TCB に含まれている情報を初期状態に戻す。
例えば、ts_chg_pri/tn_chg_pri によりタスク優先度を変更されているタスクが、本 API により終了した時、タスク優先度は ts_cre_tsk/tn_cre_tsk で指定したタスク起動時優先度(itskpri)に戻る。ts_sta_tsk/tn_sta_tsk によって再度タスクを起動した場合、本 API 実行時のタスク優先度になるわけではない。

本 API は発行元のコンテキストに戻らない API である。何らかのエラーを検出した場合でも、API を発行したコンテキストには戻らない。従って、API のリターンパラメータとして直接エラーコードを返すことはできない。万一エラーを検出した場合は異常例外が発生する。

タスク部および準タスク部以外で本 API を発行した場合は、エラーを検出し、異常例外が発生する。

自タスクが属するドメインがディスパッチ禁止されている状態で、本 API を発行した場合は、エラーを検出する。この場合は、TRON Safe Kernel は以下のように動作する。

(1) 本 API を発行したタスクを終了する。
(2) ディスパッチ禁止を解除する。
(3) 異常例外を発生する。

8.2.1.5 ts_exd_tsk/tn_exd_tsk – 自タスクの終了と削除

C 言語インタフェース

【安全 API】 void ts_exd_tsk(void);

【通常 API】 void tn_exd_tsk(void);

パラメータ

なし

リターンパラメータ

なし（API を発行したコンテキストには戻らない）

エラーコード

なし（エラーを検出した場合は異常例外を発生）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

自タスクを正常終了させ、さらに自タスクを削除する。すなわち、自タスクを未登録状態(NON-EXISTENT)へと移行させる。

本 API は発行元のコンテキストに戻らない API である。何らかのエラーを検出した場合でも、API を発行したコンテキストには戻らない。従って、API のリターンパラメータとして直接エラーコードを返すことはできない。万一エラーを検出した場合は異常例外が発生する。

タスク部および準タスク部以外で本 API を発行した場合は、エラーを検出し、異常例外が発生する。

自タスクが属するドメインがディスパッチ禁止されている状態で、本 API を発行した場合は、エラーを検出する。この場合は、TRON Safe Kernel は以下のように動作する。

(1) 本 API を発行したタスクを終了し削除する。
(2) ディスパッチ禁止を解除する。
(3) 異常例外を発生させる。

8.2.1.6 ts_ter_tsk/tn_ter_tsk – 他タスク強制終了

C 言語インタフェース

【安全 API】 ER ercd = ts_ter_tsk(ID tskid);

【通常 API】 ER ercd = tn_ter_tsk(ID tskid);

パラメータ

ID	tskid	タスク ID

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（tskid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（tskid のタスクが存在しない）
E_OBJ	オブジェクトの状態が不正（対象タスクが休止状態(DORMANT)または自タスク）
E_DACV	ドメインアクセス保護違反（対象タスクは他のドメインに所属してアクセス保護されている）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

tskid で指定されたタスクを強制的に終了させる。
具体的には、tskid で指定された対象タスクを休止状態(DORMANT)に移行させる。

タスクが休止状態(DORMANT)に戻る時は、タスク優先度などタスクの持つ情報を初期状態に戻す。
例えば、ts_chg_pri/tn_chg_pri によりタスク優先度を変更されているタスクが、本 API により終了した時、タスク優先度はタスク生成時に指定されたタスク起動時優先度(itskpri)に戻る。ts_sta_tsk/tn_sta_tsk によって再度タスクを起動した場合、本 API を実行して強制終了された時のタスク優先度になるわけではない。

対象タスクが待ち状態(強制待ち状態(SUSPENDED)を含む)にあった場合でも、対象タスクは待ち解除となって終了する。また、対象タスクが何らかの待ち行列(セマフォ待ちなど)につながれていた場合には、対象タスクはその待ち行列から削除される。

本 API では、自タスクは指定できない。自タスクを指定した場合には、E_OBJ のエラーとなる。

本 API の対象タスクの状態と実行結果との関係を、「表 8-1 対象タスクの実行と実行結果の関係」に示す。

表 8-1 対象タスクの実行と実行結果の関係

対象タスク状態	リターンコード	処理
実行できる状態（RUNNING,READY)(自タスク以外）	E_OK	強制終了
実行状態（RUNNING)(自タスク）	E_OBJ	何もしない
待ち状態（WAITING）	E_OK	強制終了
強制待ち状態（SUSPENDED）	E_OK	強制終了
二重待ち状態（WAITING-SUSPENDED）	E_OK	強制終了
休止状態（DORMANT）	E_OBJ	何もしない
未登録状態（NON-EXISTENT）	E_NOEXS	何もしない

8.2.1.7 ts_chg_pri/tn_chg_pri – タスク優先度変更

C 言語インタフェース

【安全 API】 ER ercd = ts_chg_pri(ID tskid, PRI tskpri);

【通常 API】 ER ercd = tn_chg_pri(ID tskid, PRI tskpri);

パラメータ

ID	tskid	タスク ID
PRI	tskpri	タスク優先度

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（tskid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（tskid のタスクが存在しない）
E_PAR	パラメータエラー（tskpri が不正あるいは利用できない値）
E_ILUSE	不正使用（上限優先度違反）
E_DACV	ドメインアクセス保護違反（対象タスクは他のドメインに所属してアクセス保護されている）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

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

tskpri には、タスク優先度としては、1～250 の値を指定することができ、数の小さい方が高い優先度となる。実際に指定可能な優先度は、ドメイン毎にコンフィギュレーションによって静的に決定する。またコンフィギュレーションによらず、タスク優先度として TPRI_INI(=0)を指定することが可能である。

tskpri に自ドメインの最高優先度よりも高い優先度を指定した場合はエラーE_PAR となる。

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

tskpri に TPRI_INI(=0)が指定されると、対象タスクのベース優先度を、タスクの起動時優先度(itskpri)に変更する。

この API で変更した優先度は、タスクが終了するまで有効である。タスクが休止状態(DORMANT)に戻る時、終了前のタスクの優先度は破棄され、タスク生成時に指定されたタスク起動時優先度 itskpri になる。

ただし、休止状態(DORMANT)中に変更した優先度は有効である。次にタスクを起動したときは、休止状態(DORMANT)中に設定された優先度で起動される。

この API を実行した結果、対象タスクの現在優先度がベース優先度に一致している場合(ミューテックス機能を使わない場合には、この条件は常に成り立つ)には、次の処理を行う。

- 対象タスクが実行できる状態である場合、タスクの優先順位を、変更後の優先度に従って変化させる。変更後の優先度と同じ優先度を持つタスクの間では、対象タスクの優先順位を最低とする。
- 対象タスクが何らかのタスク優先度順の待ち行列につながれている場合にも、その待ち行列の中での順序を、変更後の優先度に従って変化させる。変更後の優先度と同じ優先度を持つタスクの間では、対象タスクを最後につなぐ。
- 対象タスクが TA_CEILING 属性のミューテックスをロックしているか、ロックを待っている場合で、tskpri で指定されたベース優先度が、それらのミューテックスのいずれかの上限優先度よりも高い場合には、E_ILUSE エラーを返す。

この API を呼び出した結果、対象タスクのタスク優先度順の待ち行列の中での順序が変化した場合、対象タスクやその待ち行列で待っている他のタスクの待ち解除が起こる場合がある(メッセージバッファの送信待ち行列)。

対象タスクが、TA_INHERIT 属性のミューテックスのロック待ち状態である場合、この API でベース優先度を変更したことにより、推移的な優先度継承の処理が実行される場合がある。

ミューテックス機能を使わない場合には、対象タスクに自タスク、変更後の優先度に自タスクのベース優先度を指定してこの API が呼び出されると、自タスクの実行順位は同じ優先度を持つタスクの中で最低となる。

8.2.1.8 ts_inf_tsk/tn_inf_tsk – タスク統計情報参照

C 言語インタフェース

【安全 API】 ER ercd = ts_inf_tsk(ID tskid, T_ITSK *pk_itsk, BOOL clr);

【通常 API】 ER ercd = tn_inf_tsk(ID tskid, T_ITSK *pk_itsk, BOOL clr);

パラメータ

ID	tskid	タスク ID
T_ITSK*	pk_itsk	タスク統計情報を返す領域へのポインタ
BOOL	clr	タスク統計情報のクリアの有無（安全 API のみ有効）

リターンパラメータ

ER	ercd	エラーコード
T_ITSK*	itsk	タスク統計情報

タスク統計情報

```
typedef struct st_itsk {
        RELTIM  ctime;          /* 累積タスク実行時間（ミリ秒）*/
        RELTIM  stime;          /* 連続タスク実行時間（ミリ秒）*/
} T_ITSK;
```

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（tskid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（tskid のタスクが存在しない）
E_PAR	パラメータエラー（pk_itsk で指定されたアドレスが不正）
E_DACV	ドメインアクセス保護違反（対象タスクは他のドメインに所属してアクセス保護されている）
E_MACV	メモリアクセス違反エラー（pk_itsk に対するアクセス権限がない、または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

tskid で示された対象タスクの統計情報を参照する。

tskid=TSK_SELF(=0)によって自タスクの指定を行うことができる。

ctime は、累積タスク実行時間を示す。累積タスク実行時間とは、実行されていた時間を累積した値(単位ミリ秒)である。時間

の計測は ts_sta_tsk/tn_sta_tsk によりタスクが実行を開始するとともに開始される。加算されるのは対象タスクが実行している時間のみであり、割込みハンドラなどの他の実行単位のプログラムの実行時間は加算されない。なお、対象タスクから呼び出された拡張 SVC の実行時間は加算される。

ctime がリセット（0 クリア）されるのは以下の場合である。

- ts_sta_tsk/tn_sta_tsk でタスクが実行開始した
- ts_inf_tsk が clr=TRUE 指定で実行された（値を取得後にリセット）

stime は、連続タスク実行時間を示す。

連続タスク実行時間とは、対象タスクが連続実行している時間の値である(単位ミリ秒)。タスクの連続実行とは、対象タスクのプログラムが実行開始してから、実行状態以外に切り替わるまでである。タスクの実行中に割り込まれる割込みハンドラなどの実行は、連続実行には影響しない。ただし、連続実行時間として加算されるのは、対象タスクが実行している時間のみであり、割込みハンドラなどの他の実行単位のプログラムの実行時間は加算されない。なお、対象タスクから呼び出された拡張 SVC の実行時間は加算される。

stime がリセット（0 クリア）されるのは以下の場合である。

- 実行タスクがディスパッチし、実行状態以外に遷移した

安全 API では、clr=TRUE の場合は、統計情報を取り出した後、累積タスク実行時間をリセット(0 クリア)する。

通常 API では、clr の指定は無視される。

8.2.1.9 ts_ref_tsk/tn_ref_tsk – タスク状態参照

C 言語インタフェース

【安全 API】 ER ercd = ts_ref_tsk(ID tskid, T_RTSK *pk_rtsk);

【通常 API】 ER ercd = tn_ref_tsk(ID tskid, T_RTSK *pk_rtsk);

パラメータ

ID	tskid	タスク ID
T_RTSK*	pk_rtsk	タスク状態を返す領域へのポインタ

リターンパラメータ

ER	ercd	エラーコード
T_RTSK	rtsk	タスク状態

タスク状態

```
typedef     struct st_rtsk {
            ID      dmnno;      /* ドメイン番号 */
            ATR     tskatr;     /* タスク属性 */
            FP      task;       /* タスク起動アドレス */
            RELTIM  maxrtim;    /* 最大連続実行時間 */
            INT     stksz;      /* ユーザスタックサイズ(バイト数) */
            INT     sstksz;     /* システムスタックサイズ(バイト数) */
            PRI     itskpri;    /* 起動時優先度 */
            PRI     tskpri;     /* 現在の優先度 */
            PRI     tskbpri;    /* ベース優先度 */
            UINT    tskstat;    /* タスク状態 */
            UINT    tskwait;    /* 待ち要因 */
            ID      wid;        /* 待ちオブジェクト ID */
            INT     wupcnt;     /* 起床要求数 */
            INT     suscnt;     /* 強制待ち要求ネスト数 */
            UINT    waitmask;   /* 待ちを禁止されている待ち要因 */
            UB      oname[8];   /* オブジェクト名称 */
} T_RTSK;
```

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（tskid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（tskid のタスクが存在しない）
E_PAR	パラメータエラー（pk_rtsk で指定されたアドレスが不正）
E_DACV	ドメインアクセス保護違反（対象タスクは他のドメインに所属してアクセス保護されている）

E_MACV	メモリアクセス違反エラー（pk_rtsk に対するアクセス権限がない、または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

tskid で示された対象タスクの各種の状態を参照する。

tskid=TSK_SELF(=0)によって自タスクの指定を行うことができる。ただし、タスク独立部やタイムイベントハンドラから発行した API で tskid=TSK_SELF を指定した場合には、E_ID のエラーとなる。

dmnno は、属しているドメインの番号を示す。
tskpri は、対象タスクの現在優先度を示す。
tskbpri は、対象タスクのベース優先度を示す。

tskstat は、タスク状態を示す。
タスク状態としては次のような値をとる。

TTS_RUN	0x00000001	実行状態（RUNNING）
TTS_RDY	0x00000002	実行可能状態（READY）
TTS_WAI	0x00000004	待ち状態（WAITING）
TTS_SUS	0x00000008	強制待ち状態(SUSPENDED)
TTS_WAS	0x0000000c	二重待ち状態(WAITING-SUSPENDED)
TTS_DMT	0x00000010	休止状態（DORMANT）
TTS_NODISWAI	0x00000080	待ち禁止拒否状態

TTS_RUN, TTS_WAI などによるタスク状態の表現はビット対応となっている。
TTS_WAI の場合、ts_dis_wai/tn_dis_wai による待ち禁止を拒否している状態であれば、TTS_NODISWAI がセットされる。
TTS_WAI 以外と TTS_NODISWAI が組み合わされることはない。
割込みハンドラの中から、割り込まれたタスクを対象とした ts_ref_tsk を実行した場合は、tskstat として実行状態 TTS_RUN を返す。

tskwait は、対象タスクの待ち要因を示す。
wid は、対象タスクが待ち状態になっているオブジェクトの ID を示す。
対象タスクが待ち状態でない場合、tskwait, wid は共に 0 となる。
対象タスクが待ち状態の場合、すなわち tskstat が TTS_WAI の場合、tskwait, wid の値を「表 8-2 tskwait と wid の値」に示す。

表 8-2 tskwait と wid の値

tskwait	値	意味	wid
TTW_SLP	0x00000001	ts_slp_tsk による待ち	0
TTW_DLY	0x00000002	ts_dly_tsk による待ち	0
TTW_SEM	0x00000004	ts_wai_sem による待ち	待ち対象の semid
TTW_FLG	0x00000008	ts_wai_flg による待ち	待ち対象の flgid
TTW_MTX	0x00000080	ts_loc_mtx による待ち	待ち対象の mtxid
TTW_SMBF	0x00000100	ts_snd_mbf による待ち	待ち対象の mbfid
TTW_RMBF	0x00000200	ts_rcv_mbf による待ち	待ち対象の mbfid

wupcnt は、起床要求数を示す。

起床要求数とは、対象タスクが起床待ち状態でない状態で ts_wup_tsk/tn_wup_tsk が発行された回数である。

waitmask は、待ちを禁止されている待ち要因を示す。

対象タスクにおいて、待ちに入ることを禁止されている待ち要因であり、対象タスクに対して ts_dis_wai/tn_dis_wai のパラメータとして指定された待ち要因となる。waitmask は、tskwait と同じビット並びとなる。

休止状態(DORMANT)のタスクでは wupcnt=0 である。

8.2.1.10 ts_slp_tsk/tn_slp_tsk – 自タスクを起床待ち状態へ移行

C 言語インタフェース

【安全 API】 ER ercd = ts_slp_tsk(TMO tmout);

【通常 API】 ER ercd = tn_slp_tsk(TMO tmout);

パラメータ

TMO	tmout	タイムアウト指定（ミリ秒）

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_PAR	パラメータエラー（tmout の値が不正）
E_RLWAI	待ち状態強制解除（待ちの間に ts_rel_wai/tn_rel_wai を受け付け）
E_DISWAI	待ち禁止による待ち解除
E_TMOUT	ポーリング失敗またはタイムアウト
E_CTX	コンテキストエラー（タスク部および準タスク部以外、ディスパッチ禁止状態、割込み禁止状態で実行）
E_DOMAIN	ドメインエラー

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

自タスクを実行状態(RUNNING)から起床待ち状態(ts_wup_tsk/tn_wup_tsk を待つ状態)に移す。ただし、自タスクに対する起床要求がキューイングされている場合、具体的には自タスクの起床要求数が 1 以上の場合には、起床要求数から 1 を減じ、自タスクを待ち状態に移行させず、そのまま実行を継続する。

tmout で指定した時間が経過する前に本 API を発行したタスクを対象とした ts_wup_tsk/tn_wup_tsk が発行された場合、この API は正常終了する。

一方、tmout で指定した時間が経過する間に ts_wup_tsk/tn_wup_tsk が発行されなかった場合は、タイムアウトエラー E_TMOUT となる。

tmout として TMO_FEVR(=-1)を指定した場合は、タイムアウトまでの時間が無限大であることを示す。この場合は、ts_wup_tsk が発行されるまで永久に待ち状態になる。安全 API では TMO_FEVR を指定した場合は、ドメインエラーE_DOMAIN となる。

tmout として TMO_POL(=0)を指定した場合は、待ち状態に入ることなく本 API を終了する。

この時、起床要求数が 0 であれば、エラーE_TMOUT(ポーリング失敗)となる。起床要求数が 1 以上であれば、起床要求数から 1 を減じ、正常終了する。

tmout として、−2 以下の値を指定した場合、パラメータエラーE_PAR となる。さらに安全ドメインの場合、tmout として、最大タイムアウト時間を超えて指定した場合、パラメータエラーE_PAR となる。

8.2.1.11 ts_wup_tsk/tn_wup_tsk – 他タスクの起床

C 言語インタフェース

【安全 API】 ER ercd = ts_wup_tsk(ID tskid);

【通常 API】 ER ercd = tn_wup_tsk(ID tskid);

パラメータ

ID	tskid	タスク ID

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（tskid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（tskid のタスクが存在しない）
E_OBJ	オブジェクトの状態が不正（対象タスクが自タスクまたは休止状態(DORMANT)）
E_QOVR	キューイングまたはネストのオーバーフロー（キューイング数 wupcnt のオーバーフロー）
E_DACV	ドメインアクセス保護違反（対象タスクは他のドメインに所属してアクセス保護されている）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

tskid で指定されたタスクが起床待ち状態であった場合に、その待ち状態を解除する。

本 API では、自タスクを指定することはできない。自タスクを指定した場合には、E_OBJ のエラーとなる。

対象タスクが ts_slp_tsk/tn_slp_tsk を実行したことによる待ち状態でない場合には本 API による起床要求はキューイングされる。すなわち、対象タスクに対して本 API が発行されたという記録が残り、この後で対象タスクが ts_slp_tsk/tn_slp_tsk を実行した場合に、待ち状態にならず実行が継続されることになる。これを起床要求のキューイングと呼ぶ。
起床要求のキューイングの動作は、具体的には次のようになる。
各タスクは、起床要求数(wupcnt)という情報を持っており、その初期値(ts_sta_tsk/tn_sta_tsk 実行時の値)は 0 である。起床待ち状態でないタスクに対して本 API を実行することにより、対象タスクの起床要求数がプラス 1 される。一方、タスクが ts_slp_tsk/tn_slp_tsk を実行することにより、そのタスクの起床要求数がマイナス 1 される。そうして、起床要求数が 0 のタスクが ts_slp_tsk/tn_slp_tsk を実行した時に、起床要求数がマイナスになる代わりに、そのタスクが待ち状態になる。
起床要求数の最大値は TS_MAX_WUPCNT である。TS_MAX_WUPCNT は、コンフィギュレーションによって静的に決定する。
起床要求数の最大値の制限を超えて本 API を発行した場合には、E_QOVR のエラーとなる。

8.2.1.12 ts_can_wup/tn_can_wup – タスクの起床要求を無効化

C 言語インタフェース

【安全 API】 INT wupcnt = ts_can_wup(ID tskid);

【通常 API】 INT wupcnt = tn_can_wup(ID tskid);

パラメータ

ID	tskid	タスク ID

リターンパラメータ

INT	wupcnt	キューイングされていた起床要求回数
	または	エラーコード

エラーコード

E_ID	不正 ID 番号（tskid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（tskid のタスクが存在しない）
E_OBJ	オブジェクトの状態が不正（対象タスクが休止状態(DORMANT)）
E_DACV	ドメインアクセス保護違反（対象タスクは他のドメインに所属してアクセス保護されている）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

tskid で指定されたタスクの起床要求をすべてキャンセルする。

すなわち、対象タスクの起床要求数(wupcnt)を 0 にする。

正常終了時の戻り値は、本 API を発行した時点での起床要求数である。

tskid=TSK_SELF(=0)によって自タスクの指定になる。

8.2.1.13 ts_rel_wai/tn_rel_wai – 他タスクの待ち状態解除

C 言語インタフェース

【安全 API】 ER ercd = ts_rel_wai(ID tskid);

【通常 API】 ER ercd = tn_rel_wai(ID tskid);

パラメータ

ID	tskid	タスク ID

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（tskid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（tskid のタスクが存在しない）
E_OBJ	オブジェクトの状態が不正 （対象タスクが待ち状態ではない（自タスクや休止状態(DORMANT)の場合を含む)）
E_DACV	ドメインアクセス保護違反（対象タスクは他のドメインに所属してアクセス保護されている）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

tskid で示されるタスクが何らかの待ち状態(強制待ち状態(SUSPENDED)を除く)にある場合に、それを強制的に解除する。
本 API により待ち状態が解除されたタスクに対しては、エラーE_RLWAI が返る。このとき、対象タスクは資源を確保せずに(待ち解除の条件が満たされないまま)待ち状態が解除されることが保証される。

本 API では、待ち状態解除要求のキューイングは行わない。すなわち、tskid で示される対象タスクが既に待ち状態であればその待ち状態を解除するが、対象タスクが待ち状態でなければ、発行元にエラーE_OBJ が返る。本 API で自タスクを指定した場合にも、同様に E_OBJ のエラーとなる。
本 API では、強制待ち状態(SUSPENDED)の解除は行わない。強制待ち状態(SUSPENDED)のタスクを指定した場合はエラーE_OBJ となる。

二重待ち状態(WAITING-SUSPENDED)のタスクを対象として本 API を発行すると、対象タスクは強制待ち状態(SUSPENDED)となる。
ts_rel_wai/tn_rel_wai の対象タスクの状態と実行結果との関係を「表 8-3 対象タスクの状態と実行結果」に示す。

表 8-3 対象タスクの状態と実行結果

対象タスク状態	リターンコード	処理
実行できる状態(RUNNING,READY)(自タスク以外)	E_OBJ	何もしない
実行状態(RUNNING)(自タスク)	E_OBJ	何もしない
待ち状態(WAITING)	E_OK	待ち解除
強制待ち状態(SUSPENDED)	E_OBJ	何もしない
二重待ち状態(WAITING-SUSPENDED)	E_OK	強制待ち状態に移行
休止状態(DORMANT)	E_OBJ	何もしない
未登録状態(NON-EXISTENT)	E_NOEXS	何もしない

8.2.1.14 ts_sus_tsk/tn_sus_tsk – 他タスクを強制待ち状態へ移行

C 言語インタフェース

【安全 API】 ER ercd = ts_sus_tsk(ID tskid);

【通常 API】 ER ercd = tn_sus_tsk(ID tskid);

パラメータ

ID	tskid	タスク ID

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（tskid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（tskid のタスクが存在しない）
E_OBJ	オブジェクトの状態が不正（対象タスクが自タスクまたは休止状態(DORMANT)）
E_CTX	ディスパッチ禁止状態で実行状態のタスクを指定した
E_QOVR	キューイングまたはネストのオーバーフロー（ネスト数 suscnt のオーバーフロー）
E_DACV	ドメインアクセス保護違反（対象タスクは他のドメインに所属してアクセス保護されている）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

tskid で指定されたタスクを強制待ち状態(SUSPENDED)に移し、タスクの実行を中断させる。

強制待ち状態(SUSPENDED)は、ts_rsm_tsk/tn_rsm_tsk、ts_frsm_tsk/tn_frsm_tsk の発行によって解除される。

対象タスクが既に待ち状態であった場合には、本 API の実行により、対象タスクは待ち状態と強制待ち状態が複合した二重待ち状態(WAITING-SUSPENDED)となる。その後、このタスクの待ち解除の条件が満たされると、対象タスクは強制待ち状態(SUSPENDED)に移行する。一方、このタスクに対して ts_rsm_tsk/tn_rsm_tsk が発行されると、対象タスクは前と同じ待ち状態に戻る。

強制待ち状態(SUSPENDED)は他タスクの発行した API による中断状態を意味するので、本 API で自タスクを指定することはできない。自タスクを指定した場合には、E_OBJ のエラーとなる。

タスク独立部やタイムイベントハンドラからの本 API の発行において、ディスパッチ禁止状態で実行状態(RUNNING)のタスクを指定した場合は E_CTX のエラーとなる。

あるタスクに対して複数回の本 API が発行された場合、そのタスクは多重の強制待ち状態(SUSPENDED)になる。これを強制待ち要求のネストと呼ぶ。この場合、本 API が発行された回数(suscnt)と同じ回数の ts_rsm_tsk/ tn_rsm_tsk を発行することにより、対象タスクが元の状態に戻る。従って、ts_sus_tsk～ts_rsm_tsk/tn_sus_tsk～tn_rsm_tsk の対をネストすることが可能である。

強制待ち要求のネスト回数の最大値は TS_MAX_SUSCNT である。TS_MAX_SUSCNT は、コンフィギュレーションによって静的に決定する。
強制待ち要求のネスト回数の最大値の制限を超えて ts_sus_tsk/tn_sus_tsk を発行した場合には、E_QOVR のエラーとなる。

【補足事項】
あるタスクが資源獲得のための待ち状態(セマフォ待ちなど)で、かつ強制待ち状態(SUSPENDED)の場合でも、強制待ち状態(SUSPENDED)でない時と同じ条件によって資源の割り当て(セマフォの割り当てなど)が行われる。強制待ち状態(SUSPENDED)であっても、資源割り当ての遅延などが行われるわけではなく、資源割り当てや待ち状態の解除に関する条件や優先度は全く変わらない。すなわち、強制待ち状態(SUSPENDED)は、他の処理やタスク状態と直交関係にある。

8.2.1.15 ts_rsm_tsk/tn_rsm_tsk – 強制待ち状態のタスクを再開

C 言語インタフェース

【安全 API】 ER ercd = ts_rsm_tsk(ID tskid);

【通常 API】 ER ercd = tn_rsm_tsk(ID tskid);

パラメータ

ID	tskid	タスク ID

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（tskid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（tskid のタスクが存在しない）
E_OBJ	オブジェクトの状態が不正（対象タスクが強制待ち状態(SUSPENDED)ではない(自タスクや休止状態(DORMANT)の場合を含む)）
E_DACV	ドメインアクセス保護違反（対象タスクは他のドメインに所属してアクセス保護されている）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

tskid で指定されたタスクの強制待ち状態(SUSPENDED)を解除する。
すなわち、対象タスクが以前に発行された ts_sus_tsk/tn_sus_tsk によって強制待ち状態(SUSPENDED)に入り、その実行が中断している場合に、その状態を解除し、実行を再開させる。

対象タスクが待ち状態(WAITING)と強制待ち状態(SUSPENDED)の複合した二重待ち状態(WAITING-SUSPENDED)であった場合には、本 API の実行により強制待ち状態(SUSPENDED)のみが解除され、対象タスクは待ち状態となる。(「図 3-1 タスクの状態遷移」を参照)

本 API では、自タスクを指定することはできない。自タスクを指定した場合には、E_OBJ のエラーとなる。

本 API では、強制待ち要求のネスト(suscnt)を 1 回分だけ解除する。従って、対象タスクに対して 2 回以上の ts_sus_tsk/tn_sus_tsk が発行されていた場合(suscnt>=2)には、本 API の実行が終った後も、対象タスクはまだ強制待ち状態(SUSPENDED)のままである。

実行状態(RUNNING)もしくは実行可能状態(READY)のタスクが ts_sus_tsk/tn_sus_tsk により強制待ち状態(SUSPENDED)にな

った後、本 API によって実行を再開した場合、そのタスクは同じ優先度を持つタスクの中で最低の優先順位となる。

例えば、同じ優先度の task_A と task_B に対して以下の API を実行した場合、次のような動作をする。

```
ts_sta_tsk ( tskid=task_A, stacd_A );
ts_sta_tsk ( tskid=task_B, stacd_B );
/* この時は FCFS（First Come First Served）の原則により、優先順位は task_A、task_B の順になる。 */

ts_sus_tsk ( tskid=task_A );
ts_rsm_tsk ( tskid=task_A );
/* この時の優先順位は task_B、task_A の順になる。 */
```

8.2.1.16 ts_frsm_tsk/tn_frsm_tsk – 強制待ち状態のタスクを強制再開

C 言語インタフェース

【安全 API】 ER ercd = ts_frsm_tsk(ID tskid);

【通常 API】 ER ercd = tn_frsm_tsk(ID tskid);

パラメータ

ID	tskid	タスク ID

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（tskid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（tskid のタスクが存在しない）
E_OBJ	オブジェクトの状態が不正（対象タスクが強制待ち状態(SUSPENDED)ではない(自タスクや休止状態(DORMANT)の場合を含む)）
E_DACV	ドメインアクセス保護違反（対象タスクは他のドメインに所属してアクセス保護されている）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

tskid で指定されたタスクの強制待ち状態(SUSPENDED)をすべて解除する。
すなわち、対象タスクが以前に発行された ts_sus_tsk/tn_sus_tsk によって強制待ち状態(SUSPENDED)に入り、その実行が中断している場合に、その状態を解除し、実行を再開させる。

対象タスクが待ち状態(WAITING)と強制待ち状態(SUSPENDED)の複合した二重待ち状態(WAITING-SUSPENDED)であった場合には、本 API の実行により強制待ち状態(SUSPENDED)のみが解除され、対象タスクは待ち状態となる。

本 API では、自タスクを指定することはできない。自タスクを指定した場合には、E_OBJ のエラーとなる。

本 API は、強制待ち要求のネスト(suscnt)をすべて解除する(suscnt=0)。従って、対象タスクに対して 2 回以上の ts_sus_tsk/tn_sus_tsk が発行されていた場合(suscnt>=2)でも、それらの要求をすべて解除(suscnt=0)する。すなわち、強制待ち状態(SUSPENDED)は必ず解除され、対象タスクが二重待ち状態(WAITING-SUSPENDED)でない限り実行を再開できる。

実行状態(RUNNING)もしくは実行可能状態(READY)のタスクが ts_sus_tsk/tn_sus_tsk により強制待ち状態(SUSPENDED)になった後、本 API によって実行を再開した場合、そのタスクは同じ優先度を持つタスクの中で最低の優先順位となる。

例えば、同じ優先度の task_A と task_B に対して以下の API を実行した場合、次のような動作をする。

```
ts_sta_tsk ( tskid=task_A, stacd_A );
ts_sta_tsk ( tskid=task_B, stacd_B );
/* この時は FCFS の原則により、優先順位は task_A、task_B の順になる。 */

ts_sus_tsk ( tskid=task_A );
ts_frsm_tsk ( tskid=task_A );
/* この時の優先順位は task_B、task_A の順になる。 */
```

8.2.1.17 ts_dly_tsk/tn_dly_tsk – タスク遅延

C 言語インタフェース

【安全 API】 ER ercd = ts_dly_tsk(RELTIM dlytim);

【通常 API】 ER ercd = tn_dly_tsk(RELTIM dlytim);

パラメータ

RELTIM	dlytim	遅延時間（ミリ秒）

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_CTX	コンテキストエラー（タスク部および準タスク部以外、ディスパッチ禁止状態、割込み禁止状態で実行）
E_RLWAI	待ち状態強制解除（待ちの間に ts_rel_wai/tn_rel_wai を受け付け）
E_DISWAI	待ち禁止による待ち解除

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

自タスクの実行を一時的に停止し、時間経過待ちの状態に入る。

タスクの実行を停止する時間は、dlytim により指定される。

時間経過待ちの状態は待ち状態の一つであり、ts_rel_wai/tn_rel_wai により時間経過待ちを解除することができる。

本 API を発行したタスクが二重待ち状態(WAITING-SUSPENDED)になっている間も、時間経過のカウントは行われる。

【補足事項】

本 API は、ts_slp_tsk/tn_slp_tsk とは異なり、遅延時間として指定された時間が経過した場合に正常終了となる。また、遅延時間中に ts_wup_tsk/tn_wup_tsk が実行されても、待ち解除とはならない。遅延時間が経過する前に本 API が終了するのは、ts_rel_wai/tn_rel_wai、ts_dis_wai/tn_dis_wai、ts_ter_tsk/tn_ter_tsk が発行された場合に限られる。

8.2.1.18 ts_dis_wai/tn_dis_wai – タスク待ち状態の禁止

C 言語インタフェース

【安全 API】 INT tskwait = ts_dis_wai(ID tskid, UINT waitmask);

【通常 API】 INT tskwait = tn_dis_wai(ID tskid, UINT waitmask);

パラメータ

ID	tskid	タスク ID
UINT	waitmask	タスク待ち禁止設定

リターンパラメータ

INT	tskwait	タスク待ち禁止後のタスク待ち状態
	または	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（tskid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（tskid のタスクが存在しない）
E_PAR	パラメータエラー（waitmask が不正）
E_DACV	ドメインアクセス保護違反（対象タスクは他のドメインに所属してアクセス保護されている）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

tskid で指定されたタスクに対し、waitmask で指定した待ち要因による待ちに入ることを禁止する。

対象タスクがすでに waitmask で指定した待ち要因の待ちに入っていた場合は、その待ちを解除する。

waitmask には、以下の待ち要因を任意に組み合わせた論理和(OR)を指定する。

```
#define TTW_SLP     (0x00000001U)   /* 起床待ちによる待ち */
#define TTW_DLY     (0x00000002U)   /* タスクの遅延による待ち */
#define TTW_SEM     (0x00000004U)   /* セマフォ待ち */
#define TTW_FLG     (0x00000008U)   /* イベントフラグ待ち */
#define TTW_MTX     (0x00000080U)   /* ミューテックス待ち */
#define TTW_SMBF    (0x00000100U)   /* メッセージバッファ送信待ち */
#define TTW_RMBF    (0x00000200U)   /* メッセージバッファ受信待ち */
#define TTX_SVC     (0x80000000U)   /* 拡張 SVC 呼び出し禁止 */
```

上記以外のビットが設定されていた場合、または上記のビットが一つも指定されていない場合は、エラーE_PAR が発生する。

すでに対象タスクに待ち禁止状態が設定されていた場合は、新たに waitmask で指定された待ち禁止状態に変更する。待ち禁止状態を全て解除したい場合は、ts_ena_wai/tn_ena_wai を使用する。

TTX_SVC はタスクの待ちではないが、拡張 SVC の呼び出しを禁止する特殊な指定となる。
TTX_SVC が指定されている場合、対象タスクが拡張 SVC を呼び出そうとすると、拡張 SVC を呼び出さずに E_DISWAI を返す。
すでに呼び出している拡張 SVC を終了させることはできない。

戻り値(tskwait)には、本 API によって待ち禁止処理が行われた後の対象タスクの待ち状態を返す。
この値は、ts_ref_tsk/tn_ref_tsk の tskwait と同じ値となる。なお、tskwait には TTX_SVC に関する情報は返されない。

tskwait が 0 であれば、対象タスクは待ち状態に入っていない(または待ちが解除された)ことを示す。tskwait が 0 でなければ、waitmask に指定した以外の要因で待っていることになる。

対象タスクの待ちが本 API により解除された場合、または待ち禁止状態で新たに待ちに入ろうとした場合は、E_DISWAI が返される。

待ち禁止状態でその待ちに入る可能性がある API を呼び出したとき、待ちに入らずに処理できる場合であっても E_DISWAI が返される。これは TMO_POL を指定している場合も同様である。
例えば、メッセージバッファに空きがあり、待ちに入ることなく送信できるような状況で、メッセージバッファへ送信(ts_snd_mbf/tn_snd_mbf)を行った場合も、メッセージの送信は行われずに E_DISWAI が返される。

拡張 SVC を呼び出したとき、待ち禁止は自動的に解除され、拡張 SVC から戻ってきたときに元の設定に復帰する。また、拡張 SVC の実行中に設定された待ち禁止は、拡張 SVC から呼び出し元に戻る際に自動的に解除される。

対象タスクが休止状態(DORMANT)に戻るときにも、待ち禁止は自動的に解除される。ただし、休止状態(DORMANT)である間の設定は有効であり、次に対象タスクが起動したときは休止状態(DORMANT)の間に設定された待ち禁止が適用される。
待ち禁止の設定が有効な区間を、「図 8-1 待ち禁止の有効期間」に示す。

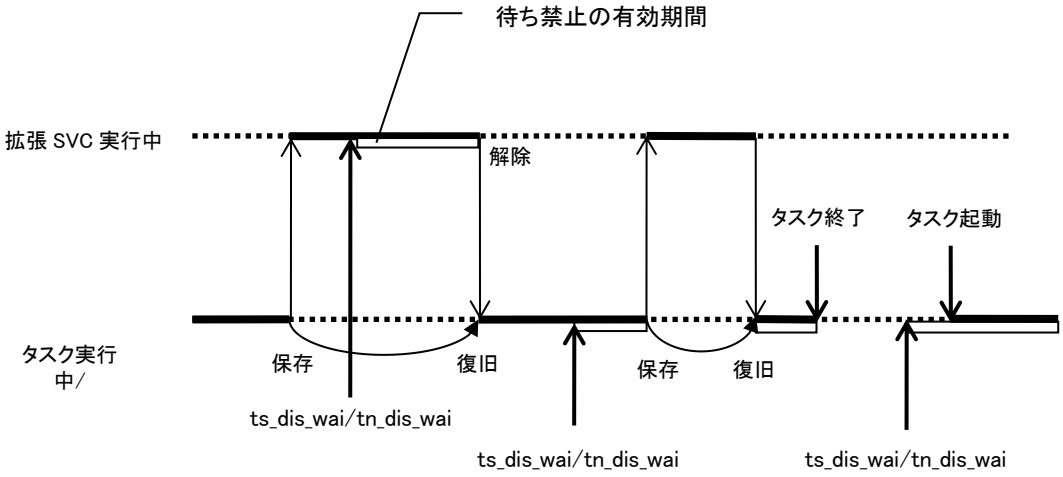

図 8-1 待ち禁止の有効期間

セマフォなど主なオブジェクトでは、オブジェクト生成時にオブジェクト属性に TA_NODISWAI を指定することができる。
TA_NODISWAI を指定して生成されたオブジェクトでは、本 API による待ち禁止は拒否され、待ちは禁止されない。
「待ちは禁止されない」とは、本 API によって特定のオブジェクトでの待ちを禁止されたタスクが、TA_NODISWAI 属性を指定して生成されたオブジェクトに対する操作を行った場合、本 API による待ち禁止指定は無視されて通常の動作になるという意味である。本 API の発行時に待ち禁止指定が拒否されてエラーが返されるという意味ではない。

対象となるオブジェクトに TA_NODISWAI 属性が指定されている場合も戻り値は 0 にならない。
例えば、TA_NODISWAI 属性が指定されているセマフォで待っているタスクに対して本 API を発行した場合、対象タスクのセマフォ待ち状態は解除されず、戻り値は TTW_SEM(0x00000004)となる。

tskid=TSK_SELF(=0)によって自タスクの指定になる。

8.2.1.19 ts_ena_wai/tn_ena_wai – タスク待ち禁止の解除

C 言語インタフェース

【安全 API】 ER ercd = ts_ena_wai(ID tskid);

【通常 API】 ER ercd = tn_ena_wai(ID tskid);

パラメータ

ID	tskid	タスク ID

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（tskid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（tskid のタスクが存在しない）
E_DACV	ドメインアクセス保護違反（対象タスクは他のドメインに所属してアクセス保護されている）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

tskid で指定されたタスクに設定されている待ち禁止をすべて解除する。

tskid=TSK_SELF(=0)によって自タスクの指定になる。

8.2.2 システム状態管理機能の API

システム状態管理機能の詳細については、「3.2 システム状態管理機能」を参照。

8.2.2.1 ts_rot_rdq/tn_rot_rdq – タスクの優先順位の回転

C 言語インタフェース

【安全 API】 ER ercd = ts_rot_rdq(PRI tskpri);

【通常 API】 ER ercd = tn_rot_rdq(PRI tskpri);

パラメータ

PRI	tskpri	タスク優先度

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_PAR	パラメータエラー（tskpri が不正）
E_CTX	コンテキストエラー

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	×

解説

自ドメイン内で、tskpri で指定される優先度のタスクの優先順位を回転する。
具体的には、自ドメインにおいて、優先度が tskpri であるタスクの中で最も高い優先順位を持つタスクを、同じ優先度を持つタスクの中で最低の優先順位とする。

タスク優先度 tskpri には 1～250 の値を指定することができ、値の小さい方が高い優先度となる。
実際に指定可能な優先度は、ドメイン毎にコンフィギュレーションによって静的に決定する。
tskpri に自ドメインの最高優先度よりも高い優先度を指定した場合はエラーE_PAR となる。

tskpri=TPRI_RUN(=0)が指定されると、本 API 発行時に自ドメインで実行状態(RUNNING)にあるタスク優先度の優先順位を回転する。
一般のタスクから発行される ts_rot_rdq/tn_rot_rdq では、この操作は自タスクの持つ優先度のタスクの優先順位を回転するのと同じ意味であるが、タイムイベントハンドラから ts_rot_rdq/tn_rot_rdq (tskpri=TPRI_RUN)を発行することも可能である。

対象優先度を持った実行できる状態のタスクが一つしかない場合には、結果的に優先順位は変化しない。ただし、タスクの連続実行時間はクリアされる。
対象優先度を持った実行できる状態のタスクがない場合は、何もしない(エラーとはしない)。

【補足事項】

ディスパッチが許可されている状態で、対象優先度に TPRI_RUN または自タスクの現在優先度を指定して本 API が呼び出されると、自タスクの実行順位は同じ優先度を持つタスクの中で最低となる。そのため、本 APIを用いて、実行権の放棄を行うことができる。

ディスパッチが禁止されている状態では、同じ優先度を持つタスクの中で最高の優先順位を持ったタスクが実行されているとは限らないため、この方法で自タスクの実行順位が同じ優先度を持つタスクの中で最低となるとは限らない。

8.2.2.2 ts_get_tid/tn_get_tid – 実行状態タスクのタスク ID 参照

C 言語インタフェース

【安全 API】 ID tskid = ts_get_tid(void);

【通常 API】 ID tskid = tn_get_tid(void);

パラメータ

なし

リターンパラメータ

ID	tskid	実行状態タスクの ID

エラーコード

なし

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

自ドメインで現在実行状態にあるタスクの ID 番号を取得する。

自ドメインに現在実行状態のタスクがない場合は、0 が返される。

タイムイベントハンドラの実行中を除けば、現在実行状態にあるタスクは自タスクである。タイムイベントハンドラから発行した場合も、自ドメインで現在実行状態にあるタスクのタスク ID が返される。

【補足事項】

本 API で返されるタスク ID は、ts_ref_sys/tn_ref_sys で返される runtskid と同一である。

8.2.2.3 ts_dis_dsp/tn_dis_dsp – ディスパッチ禁止

C 言語インタフェース

【安全 API】 ER ercd = ts_dis_dsp(TMO tmout);

【通常 API】 ER ercd = tn_dis_dsp(TMO tmout);

パラメータ

TMO	tmout	ディスパッチ禁止時間

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_PAR	パラメータエラー（tmout の値が不正）
E_CTX	コンテキストエラー（タスク部および準タスク部以外から実行）
E_DOMAIN	ドメインエラー

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

自ドメインのタスクのディスパッチを禁止する。

tmout には、ディスパッチ禁止時間を指定する。
tmout として TMO_FEVR(=-1)を指定した場合は、ディスパッチ禁止時間が無限大であることを示す。
ただし、安全 API では TMO_FEVR を指定した場合は、ドメインエラーE_DOMAIN となる。また、本 API で tmout として TMO_POL(=0)を指定した場合は、パラメータエラーE_PAR となる。
tmout として、-2 以下の値を指定した場合、パラメータエラーE_PAR となる。さらに安全ドメインの場合、tmout として、最大タイムアウト時間を超えて指定した場合、パラメータエラーE_PAR となる。

本 API 発行後、ディスパッチ禁止時間を経過もしくは、ts_ena_dsp/tn_ena_dsp が実行されディスパッチ禁止状態が解除されるまでの間は、自ドメインはディスパッチ禁止状態となり、自タスクが実行状態(RUNNING)から実行可能状態(READY)に移ることはなくなる。また、待ち状態などの実行状態以外の状態にも移ることもできなくなる。

ディスパッチ禁止状態の間は、具体的には次のような動作をする。

- 自ドメインの中で、本 API を実行したタスクより高い優先度を持つタスクが実行可能状態(READY)となっても、そのタスクにはディスパッチされない。優先度の高いタスクへのディスパッチは、ディスパッチ禁止状態が終了するまで遅延される。

- 本 API を実行したタスクが、自タスクを待ち状態に移す可能性のあるシステムコール(ts_slp_tsk/tn_slp_tsk、ts_wai_sem/tn_wai_sem など)を発行した場合には、E_CTX のエラーとなる。
- ts_ref_sys/tn_ref_sys によってシステム状態を参照した場合、sysstat として TSS_DDSP が返る。

ディスパッチ禁止状態では、実行状態のタスクは休止状態(DORMANT)や未登録状態(NON-EXISTENT)に移行することはできない。ディスパッチ禁止状態において ts_ext_tsk/tn_ext_tsk や ts_exd_tsk/tn_exd_tsk を発行した場合は、エラーE_CTX となる。この場合は以下のように動作する。

- ts_ext_tsk/tn_ext_tsk を発行したタスクは終了する。(呼び出し元には戻らない。)
- ts_exd_tsk/tn_exd_tsk を発行したタスクは終了、削除する。(呼び出し元には戻らない。)
- ディスパッチ禁止状態は自動的に解除される。
- 異常例外を発生する。

既にディスパッチ禁止状態にあるドメインで本 API を発行した場合は、ディスパッチ禁止状態がそのまま継続するだけで、エラーとはならない。ts_dis_dsp/tn_dis_dsp が再発行された時点からの tmout で、ディスパッチ禁止時間は上書き更新される。
また、本 API を何回か発行しても、その後 ts_ena_dsp/tn_ena_dsp を 1 回発行するだけでディスパッチ禁止状態が解除される。
従って、ts_dis_dsp/tn_dis_dsp～ts_ena_dsp/tn_ena_dsp の対がネストした場合の動作は、必要に応じてユーザ側で管理しなければならない。

tmout で指定した時間が経過してもディスパッチ禁止状態が解除されなかった場合は、以下のように動作する。

- ディスパッチ禁止状態は自動的に解除される。
- 異常例外を発生する。

【補足事項】
ディスパッチ禁止状態では、周期ハンドラとアラームハンドラの実行を禁止しない。ディスパッチ禁止したドメインであっても周期ハンドラとアラームハンドラは動作する。
ディスパッチ禁止は自ドメインにおいてのみ有効である。よって、他のドメインの自タスクより優先度の高いタスクは実行される。

8.2.2.4 ts_ena_dsp/tn_ena_dsp – ディスパッチ許可

C 言語インタフェース

【安全 API】 ER ercd = ts_ena_dsp(void);

【通常 API】 ER ercd = tn_ena_dsp(void);

パラメータ

なし

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_CTX	コンテキストエラー（タスク部または準タスク部以外から発行）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

自ドメインのタスクのディスパッチを許可する。すなわち、ts_dis_dsp/tn_dis_dsp によって設定されていた自ドメインのディスパッチ禁止状態を解除する。

自ドメインがディスパッチ禁止状態ではないタスクが本 API を発行した場合は、ディスパッチを許可した状態がそのまま継続するだけで、エラーとはならない。

8.2.2.5 ts_ref_sys/tn_ref_sys – システム状態参照

C 言語インタフェース

【安全 API】 ER ercd = ts_ref_sys(T_RSYS *pk_rsys);

【通常 API】 ER ercd = tn_ref_sys(T_RSYS *pk_rsys);

パラメータ

T_RSYS*	pk_rsys	システム状態を返す領域へのポインタ

リターンパラメータ

ER	ercd	エラーコード
T_RSYS	rsys	システム状態

システム状態

```
typedef struct st_rsys {
        UINT    sysstat;        /* システム状態 */
        ID      runtskid;       /* 現在実行状態にあるタスクの ID 番号 */
        ID      schedtskid;     /* 次に実行状態にすべきタスクの ID 番号 */
} T_RSYS;
```

エラーコード

E_OK	正常終了
E_PAR	パラメータエラー（pk_rsys で指定されたアドレスが不正）
E_MACV	メモリアクセス違反エラー（pkrsys に対するアクセス権限がない、または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

システム状態を参照し、情報をリターンパラメータとして返す。

sysstat は本 API が呼ばれた時点のシステム状態を示す。

sysstat は次のような値をとる。

```
sysstat := ( TSS_TSK | [TSS_DDSP] | [TSS_DINT] )
        || ( TSS_QTSK | [TSS_DDSP] | [TSS_DINT] )
        || ( TSS_TEH | [TSS_DDSP] | [TSS_DINT] )
        || ( TSS_INDP )
```

TSS_TSK	タスク部実行中
TSS_DDSP	ディスパッチ禁止中
TSS_DINT	割込み禁止中
TSS_INDP	タスク独立部実行中
TSS_QTSK	準タスク部実行中
TSS_TEH	タイムイベントハンドラ部実行中

システム状態は、ヘッダファイルにおいて以下のように定義されている。

```
#define TSS_TSK         (0x00000000U)   /* During execution of task part(context) */
#define TSS_DDSP        (0x00000001U)   /* During dispatch disable */
#define TSS_DINT        (0x00000002U)   /* During Interrupt disable */
#define TSS_INDP        (0x00000004U)   /* During execution of task independent part */
#define TSS_QTSK        (0x00000008U)   /* During execution of quasi-task part */
#define TSS_TEH         (0x00000010U)   /* During execution of time event handler */
```

runtskid には自ドメインで現在実行中のタスクの ID、schedtskid には自ドメインで次に実行状態にすべきタスクの ID が返される。なお、該当タスクがない場合は 0 を返す。
通常は runtskid = schedtskid となるが、ディスパッチ禁止中により優先度の高いタスクが起床された場合などに runtskid != schedtskid となる場合がある。
本 API は割込みハンドラやタイムイベントハンドラからも発行できる。

8.2.2.6 ts_ref_ver/tn_ref_ver – バージョン参照

C 言語インタフェース

【安全 API】 ER ercd = ts_ref_ver(T_RVER *pk_rver);

【通常 API】 ER ercd = tn_ref_ver(T_RVER *pk_rver);

パラメータ

T_RVER* pk_rver	バージョン情報を返す領域へのポインタ

リターンパラメータ

ER	ercd	エラーコード
T_RVER	rver	バージョン情報

バージョン情報

```
typedef struct st_rver {
        UH      maker;          /* カーネルのメーカーコード */
        UH      prid;           /* カーネルの識別番号 */
        UH      spver;          /* 仕様書バージョン番号 */
        UH      prver;          /* カーネルのバージョン番号 */
        UH      prno[4];        /* カーネル製品の管理情報 */
} T_RVER;
```

エラーコード

E_OK	正常終了
E_PAR	パラメータエラー（pk_rver で指定されたアドレスが不正）
E_MACV	メモリアクセス違反エラー（pk_rver に対するアクセス権限がない、または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

使用しているカーネルのバージョン情報を参照し、pk_rver で指定されるパケットに返す。

各情報の内容は「3.2.2 システムの情報取得」を参照のこと。

8.2.3 同期・通信機能（セマフォ）の API

セマフォ関連の同期・通信機能の詳細は、「3.3.1 セマフォ」を参照。

8.2.3.1 ts_cre_sem/tn_cre_sem – セマフォ生成

C 言語インタフェース

【安全 API】 ID semid = ts_cre_sem(CONST T_CSEM *pk_csem);

【通常 API】 ID semid = tn_cre_sem(CONST T_CSEM *pk_csem);

パラメータ

CONST T_CSEM* pk_csem セマフォ生成情報へのポインタ

セマフォ生成情報

```
typedef struct st_csem {
        ATR     sematr;         /* セマフォ属性 */
        INT     isemcnt;        /* セマフォカウントの初期値 */
        INT     maxsem;         /* セマフォカウントの最大値 */
        UB      oname[8];       /* オブジェクト名称 */
} T_CSEM;
```

リターンパラメータ

ID	semid	セマフォ ID
	または	エラーコード

エラーコード

E_LIMIT	セマフォの数がシステムの上限を超えた
E_RSATR	予約属性エラー（sematr が不正あるいは利用できない）
E_PAR	パラメータエラー（pk_csem で指定されたアドレスが不正、isemcnt が負または不正、maxsem が 0 以下または不正）
E_CTX	コンテキストエラー（タスク部または準タスク部以外から発行）
E_ONAME	オブジェクト名エラー（指定されたオブジェクト名が既にドメイン内で使用されている）
E_MACV	メモリアクセス違反エラー（pk_csem に対するアクセス権限がない、または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

セマフォを生成してセマフォ ID 番号を割り当てる。

sematr には、対象セマフォの動作を指定するためのセマフォ属性を設定する。

sematr には、次のような指定を行う。

sematr := (TA_TFIFO || TA_TPRI) | (TA_FIRST || TA_CNT) | [TA_ONAME] | [TA_NODISWAI]

TA_TFIFO	待ちタスクのキューイングは FIFO
TA_TPRI	待ちタスクのキューイングは優先度順
TA_FIRST	待ち行列先頭のタスクを優先
TA_CNT	要求数の少ないタスクを優先
TA_ONAME	オブジェクト名称を指定する
TA_NODISWAI	ts_dis_wai/tn_dis_wai による待ち禁止を拒否する

セマフォ属性は、ヘッダファイルにおいて以下のように定義されている。

```
#define TA_TFIFO        (0x00000000U)    /* 待ちタスクを FIFO で管理 */
#define TA_TPRI         (0x00000001U)    /* 待ちタスクを優先度順で管理 */
#define TA_FIRST        (0x00000000U)    /* 待ち行列先頭のタスクを優先 */
#define TA_CNT          (0x00000002U)    /* 要求数の少ないタスクを優先 */
#define TA_ONAME        (0x00000040U)    /* オブジェクト名称を指定 */
#define TA_NODISWAI     (0x00000080U)    /* 待ち禁止拒否 */
```

TA_TFIFO、TA_TPRI では、タスクがセマフォの待ち行列に並ぶ際の並び方を指定する。
セマフォ属性に TA_TFIFO が指定されていればタスクの待ち行列は FIFO となり、TA_TPRI が指定されていればタスクの待ち行列はタスク優先度順となる。

TA_FIRST、TA_CNT では、資源獲得の優先順を指定する。TA_FIRST および TA_CNT の指定によって待ち行列の並び順が変わることはない。待ち行列の並び順は TA_TFIFO、TA_TPRI によってのみ決定される。
TA_FIRST では、要求資源数に関係なく待ち行列の先頭のタスクから順に資源を割り当てる。このため、待ち行列の先頭のタスクが要求分の資源を獲得できない限り、待ち行列の後ろのタスクが資源を獲得することはない。
TA_CNT では、要求資源数分の資源が獲得できるタスクから順に割り当てる。具体的には、待ち行列の先頭のタスクから順に要求資源数を検査し、要求資源数分の資源を割り当て可能なタスクがあれば、そのタスクに対して資源を割り当てる。割り当て後、資源が残っていれば(資源数が 1 以上であれば)、待ち行列につながれているタスクの要求資源数の検査を継続する。要求資源数の少ない順に割り当てる訳ではない。

TA_ONAME を指定した場合に oname が有効となる。
oname には、対象セマフォのオブジェクト名称を設定する。

TA_NODISWAI を指定した場合、ts_dis_wai/tn_dis_wai による待ち禁止は拒否される。

isemcnt には、対象セマフォのセマフォカウントの初期値を指定する。isemcnt に指定可能な値は、0～TS_MAX_SEMCNT である。
maxsem には、対象セマフォのセマフォカウントの最大値(対象セマフォが保持できる資源数の最大値)を指定する。maxsem に指定可能な値は 1～TS_MAX_SEMCNT である。
TS_MAX_SEMCNT は、コンフィギュレーションによって静的に決定する。

isemcnt は、maxsem 以下でなければならない。isemcnt に maxsem よりも大きな値を指定した場合は、エラーE_PAR となる。

8.2.3.2 ts_del_sem/tn_del_sem – セマフォ削除

C 言語インタフェース

【安全 API】 ER ercd = ts_del_sem(ID semid);

【通常 API】 ER ercd = tn_del_sem(ID semid);

パラメータ

ID	semid	セマフォ ID

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（semid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（semid のセマフォが存在しない）
E_CTX	コンテキストエラー（タスク部または準タスク部以外から発行）
E_DACV	ドメインアクセス保護違反（対象セマフォは他のドメインに所属してアクセス保護されている）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

semid で指定されたセマフォを削除する。

対象セマフォにおいて条件成立を待っているタスクがあった場合、タスクの待ち状態は解除されエラーE_DLT が返される。

8.2.3.3 ts_sig_sem/tn_sig_sem – セマフォ資源返却

C 言語インタフェース

【安全 API】 ER ercd = ts_sig_sem(ID semid, INT cnt);

【通常 API】 ER ercd = tn_sig_sem(ID semid, INT cnt);

パラメータ

ID	semid	セマフォ ID
INT	cnt	返却資源数

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（semid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（semid のセマフォが存在しない）
E_QOVR	キューイングまたはネストのオーバーフロー（セマフォカウントのオーバーフロー）
E_PAR	パラメータエラー（cnt が 0 以下）
E_DACV	ドメインアクセス保護違反（対象セマフォは他のドメインに所属してアクセス保護されている）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

semid で指定されるセマフォに対して、資源を返却する操作を行う。

cnt には返却する資源数を指定する。cnt に指定可能な値は 1～TS_MAX_SEMCNT である。TS_MAX_SEMCNT は、コンフィギュレーションによって静的に決定する。

対象セマフォではセマフォカウントが cnt 個増加する。

ただし、本 API を発行して cnt 個の資源を返却することにより、対象セマフォのセマフォカウントが最大値を超える場合は、エラーE_QOVR を返す。この場合、資源の返却は一切行われず、対象セマフォのセマフォカウントは変化しない。

対象セマフォのカウントが増加した場合、対象セマフォにおいてセマフォ獲得待ち状態になっているタスクがあれば、要求資源数を確認して割り当て可能であれば資源を割り当てる。資源を割り当てられたタスクはセマフォ待ち状態が解除される。条件によっては、複数のタスクに資源が割り当てられてセマフォ待ち状態が解除される場合がある。

8.2.3.4 ts_wai_sem/tn_wai_sem – セマフォ資源獲得

C 言語インタフェース

【安全 API】 ER ercd = ts_wai_sem(ID semid, INT cnt, TMO tmout);

【通常 API】 ER ercd = tn_wai_sem(ID semid, INT cnt, TMO tmout);

パラメータ

ID	semid	セマフォ ID
INT	cnt	要求資源数
TMO	tmout	タイムアウト指定(ミリ秒)

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号 (semid が不正あるいは利用できない)
E_NOEXS	オブジェクトが存在していない (semid のセマフォが存在しない)
E_PAR	パラメータエラー (cnt<=0, maxsem<cnt, tmout の値が不正)
E_DLT	待ちオブジェクトが削除された (待ちの間に対象セマフォが削除)
E_RLWAI	待ち状態強制解除 (待ちの間に ts_rel_wai を受け付け)
E_DISWAI	待ち禁止による待ち解除
E_TMOUT	ポーリング失敗またはタイムアウト
E_CTX	コンテキストエラー (タスク部および準タスク部以外、ディスパッチ禁止状態、割込み禁止状態で実行)
E_DACV	ドメインアクセス保護違反 (対象セマフォは他のドメインに所属してアクセス保護されている)
E_DOMAIN	ドメインエラー

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

semid で指定されたセマフォから、cnt 個の資源を獲得する操作を行う。

cnt には、対象セマフォから獲得する資源数を指定する。cnt に指定可能な値は 1～セマフォカウントの最大値(maxsem)である。

資源が獲得できれば、本 API を発行したタスクは待ち状態にならずに、実行を継続する。この場合、対象セマフォのカウントは cnt 個減少する。

資源が獲得できなければ、本 API を発行したタスクはセマフォ待ち状態になる。すなわち、対象セマフォに対する待ち行列につながれる。この場合、対象セマフォのカウントは不変である。

tmout には、タイムアウト時間を指定する。指定したタイムアウト時間が経過する間に資源が獲得できなかった場合は、タイムアウトエラーE_TMOUT となる。

tmout として TMO_FEVR(=-1)を指定した場合は、タイムアウトまでの時間が無限大であることを示す。この場合は、資源が獲得できるまで永久に待ち状態になる。安全 API では TMO_FEVR を指定した場合は、ドメインエラーE_DOMAIN となる。

tmout として TMO_POL(=0)を指定した場合は、待ち状態に入ることなくエラーE_TMOUT(ポーリング失敗)となる。

tmout として、-2 以下の値を指定した場合、パラメータエラーE_PAR となる。さらに安全ドメインの場合、tmout として、最大タイムアウト時間を超えて指定した場合、パラメータエラーE_PAR となる。

8.2.3.5 ts_ref_sem/tn_ref_sem – セマフォ状態参照

C 言語インタフェース

【安全 API】 ER ercd = ts_ref_sem(ID semid, T_RSEM *pk_rsem);

【通常 API】 ER ercd = tn_ref_sem(ID semid, T_RSEM *pk_rsem);

パラメータ

ID	semid	セマフォ ID
T_RSEM*	pk_rsem	セマフォ状態を返す領域へのポインタ

リターンパラメータ

ER	ercd	エラーコード
T_RSEM	rsem	セマフォ状態

セマフォ状態

```
typedef     struct st_rsem {
            ID      dmnno;          /* ドメイン番号 */
            ID      wtsk;           /* 待ちタスクの ID */
            INT     semcnt;         /* 現在のセマフォカウントの値 */
} T_RSEM;
```

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（semid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（semid のセマフォが存在しない）
E_PAR	パラメータエラー（pk_rsem で指定されたアドレスが不正）
E_DACV	ドメインアクセス保護違反（対象セマフォは他のドメインに所属してアクセス保護されている）
E_MACV	メモリアクセス違反エラー（pk_rsem に対するアクセス権限がない、または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

semid で指定されるセマフォの各種状態を参照し、リターンパラメータとして返す。

dmnno は、属しているドメインの番号を示す。

wtsk は、このセマフォで待っているタスクの ID を示す。複数のタスクが待っている場合は、待ち行列の先頭のタスクの ID を返す。待ちタスクが無い場合は wtsk=0 となる。

semcnt は、現在のセマフォのカウント値を示す。

8.2.4 同期・通信機能（イベントフラグ）の API

イベントフラグ関連の同期・通信機能の詳細は、「3.3.2 イベントフラグ」を参照。

8.2.4.1 ts_cre_flg/tn_cre_flg – イベントフラグ生成

C 言語インタフェース

【安全 API】 ID flgid = ts_cre_flg(CONST T_CFLG *pk_cflg);

【通常 API】 ID flgid = tn_cre_flg(CONST T_CFLG *pk_cflg);

パラメータ

CONST T_CFLG*	pk_cflg	イベントフラグ生成情報へのポインタ

イベントフラグ生成情報

```
typedef    struct st_cflg {
           ATR     flgatr;          /* イベントフラグ属性 */
           UINT    iflgptn;         /* イベントフラグの初期値 */
           UB      oname[8];        /* オブジェクト名称 */
} T_CFLG;
```

リターンパラメータ

ID	flgid	イベントフラグ ID
	または	エラーコード

エラーコード

E_LIMIT	イベントフラグの数がシステムの上限を超えた
E_RSATR	予約属性エラー（flgatr が不正あるいは利用できない）
E_PAR	パラメータエラー（pk_cflg で指定されたアドレスが不正）
E_CTX	コンテキストエラー（タスク部および準タスク部以外で実行）
E_ONAME	オブジェクト名エラー（指定されたオブジェクト名が既にドメイン内で使用されている）
E_MACV	メモリアクセス違反エラー（pk_cflg に対するアクセス権限がない、または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

イベントフラグを生成してイベントフラグ ID 番号を割り当てる。

flgatr には、対象イベントフラグの動作を指定するためのイベントフラグ属性を設定する。

flgatr には、次のような指定を行う。

flgatr := (TA_TFIFO || TA_TPRI) | (TA_WMUL || TA_WSGL) | [TA_ONAME] | [TA_NODISWAI]

TA_TFIFO	待ちタスクのキューイングは FIFO
TA_TPRI	待ちタスクのキューイングは優先度順
TA_WSGL	複数タスクの待ちを許さない(Wait Single Task)
TA_WMUL	複数タスクの待ちを許す(Wait Multiple Task)
TA_ONAME	オブジェクト名称を指定する
TA_NODISWAI	ts_dis_wai による待ち禁止を拒否する

イベントフラグ属性は、ヘッダファイルにおいて以下のように定義されている。

```
#define TA_TFIFO        (0x00000000U)    /* 待ちタスクを FIFO で管理 */
#define TA_TPRI         (0x00000001U)    /* 待ちタスクを優先度順で管理 */
#define TA_WSGL         (0x00000000U)    /* 複数タスクの待ちを許さない */
#define TA_WMUL         (0x00000008U)    /* 複数タスクの待ちを許す */
#define TA_ONAME        (0x00000040U)    /* オブジェクト名称を指定 */
#define TA_NODISWAI     (0x00000080U)    /* 待ち禁止拒否 */
```

TA_WSGL を指定した場合は、複数のタスクが同時に待ち状態になることを禁止する。TA_WMUL を指定した場合は、同時に複数のタスクが待ち状態となることが許される。

TA_TFIFO、TA_TPRI では、タスクが対象イベントフラグで待ち行列に並ぶ際の並び方を指定する。
イベントフラグ属性に TA_TFIFO が指定されていればタスクの待ち行列は FIFO となり、TA_TPRI が指定されていればタスクの待ち行列はタスク優先度順となる。ただし、TA_WSGL を指定した場合は待ち行列を作らないため、TA_TFIFO、TA_TPRI のどちらを指定しても動作に変わりはない。

複数のタスクがイベントフラグ待ち状態になっている場合、イベントフラグのビットパターンがセットされるとタスクの待ち行列の先頭から順に待ち条件が成立しているか検査し、待ち条件が成立しているタスクのイベントフラグ待ちを順に解除する。従って、待ち行列の先頭のタスクが必ずしも最初に待ち解除されるとは限らない。
待ち条件が成立したタスクが複数あれば、複数のタスクの待ちが解除される。

TA_ONAME を指定した場合に oname が有効となる。
oname には、対象イベントフラグのオブジェクト名称を設定する。

TA_NODISWAI を指定した場合、ts_dis_wai/tn_dis_wai による待ち禁止は拒否される。

iflgptn には、対象イベントフラグのビットパターンの初期値を指定する。

8.2.4.2 ts_del_flg/tn_del_flg – イベントフラグ削除

C 言語インタフェース

【安全 API】 ER ercd = ts_del_flg(ID flgid);

【通常 API】 ER ercd = tn_del_flg(ID flgid);

パラメータ

ID	flgid	イベントフラグ ID

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（flgid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（flgid のイベントフラグが存在しない）
E_DACV	ドメインアクセス保護違反（対象イベントフラグは他のドメインに所属してアクセス保護されている）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

flgid で指定されたイベントフラグを削除する。

対象イベントフラグにおいて条件成立を待っているタスクがあった場合、タスクの待ち状態は解除されエラーE_DLT が返される。

8.2.4.3 ts_set_flg/tn_set_flg – イベントフラグのセット

C 言語インタフェース

【安全 API】 ER ercd = ts_set_flg(ID flgid, UINT setptn);

【通常 API】 ER ercd = tn_set_flg(ID flgid, UINT setptn);

パラメータ

ID	flgid	イベントフラグ ID
UINT	setptn	セットするビットパターン

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（flgid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（flgid のイベントフラグが存在しない）
E_DACV	ドメインアクセス保護違反（対象イベントフラグは他のドメインに所属してアクセス保護されている）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

flgid で指定されるイベントフラグのビットパターン(EventFlagPattern とする)のうち、setptn で 1 が指定されているビットをセットする。すなわち、EventFlagPattern と setptn の論理和をとり、その値を対象イベントフラグの新しいビットパターンとする。
C 言語で記述すると以下の式で表現される。

```
EventFlagPattern |= setptn;
```

本 API を発行することによって対象イベントフラグのビットパターンが変更された結果が、ts_wai_flg/tn_wai_flg で対象イベントフラグを待っていたタスクの待ち解除の条件を満たせば、そのタスクのイベントフラグ待ち状態が解除される。
イベントフラグ待ち状態が解除されたタスクは、実行状態(RUNNING)または実行可能状態(READY)へと移行する。
もし、イベントフラグ待ち状態が解除されたタスクが二重待ち状態(WAITING-SUSPENDED)であった場合には強制待ち状態(SUSPENDED)に移行する。

setptn の全ビットを 0 とした場合には、対象イベントフラグに対して何の操作も行わないことになる。ただし、この場合でもエラーとはならない。

TA_WMUL 属性を持つイベントフラグに対しては、同一のイベントフラグに対して複数のタスクが同時に待つことができる。従っ

て、イベントフラグでもタスクが待ち行列を作ることになる。この場合、本 API の一回の発行で複数のタスクが待ち解除されることがある。
具体的には、本 API によって更新されたビットパターンを用いて、対象イベントフラグの待ち行列につながれているタスクの待ち解除条件を順に確認していく。

イベントフラグ待ち状態のタスクの待ちが解除される場合、「解除条件が成立したらフラグをクリアする」という指定がなされていれば、当該タスクの待ちを解除した後に対象イベントフラグのビットパターンが変化することになる。以降は変化したビットパターンを用いて処理することになるので、待ち行列に繋がれているタスクの待ち状態が必ずしも解除されるとは限らない。
なお、タスクの待ちを解除した後にビットパターンをクリアするか否かについては、ts_wai_flg/tn_wai_flg のパラメータとして指定する。指定の詳細については、「8.2.4.4 ts_clr_flg/tn_clr_flg － イベントフラグのクリア」を参照のこと。

8.2.4.4 ts_clr_flg/tn_clr_flg – イベントフラグのクリア

C 言語インタフェース

【安全 API】 ER ercd = ts_clr_flg(ID flgid, UINT clrptn);

【通常 API】 ER ercd = tn_clr_flg(ID flgid, UINT clrptn);

パラメータ

ID	flgid	イベントフラグ ID
UINT	clrptn	クリアするビットパターン

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（flgid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（flgid のイベントフラグが存在しない）
E_DACV	ドメインアクセス保護違反（対象イベントフラグは他のドメインに所属してアクセス保護されている）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

flgid で指定されるイベントフラグのビットパターン(EventFlagPattern とする)のうち、clrptn で 0 が指定されているビットをクリアする。すなわち、EventFlagPattern と clrptn の論理積をとり、その値を対象イベントフラグの新しいビットパターンとする。
C 言語で記述すると以下の式で表現される。

```
EventFlagPattern &= clrptn;
```

本 API を発行することによって対象イベントフラグのビットパターンが変更された結果、新たにセットされるビットはない。このため、対象イベントフラグを待っているタスクが待ち解除となることはない。

clrptn の全ビットを 1 とした場合には、対象イベントフラグに対して何の操作も行わないことになる。ただし、この場合でもエラーとはならない。

8.2.4.5 ts_wai_flg/tn_wai_flg – イベントフラグ待ち

C 言語インタフェース

【安全 API】 ER ercd = ts_wai_flg(ID flgid, UINT waiptn, UINT wfmode, UINT *p_flgptn, TMO tmout);

【通常 API】 ER ercd = tn_wai_flg(ID flgid, UINT waiptn, UINT wfmode, UINT *p_flgptn, TMO tmout);

パラメータ

ID	flgid	イベントフラグ ID
UINT	waiptn	待ちビットパターン
UINT	wfmode	待ちモード
UINT*	p_flgptn	リターンパラメータ flgptn を返す領域へのポインタ
TMO	tmout	タイムアウト指定(ミリ秒)

リターンパラメータ

ER	ercd	エラーコード
UINT	flgptn	待ち解除時のビットパターン

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号 (flgid が不正あるいは利用できない)
E_NOEXS	オブジェクトが存在していない (flgid のイベントフラグが存在しない)
E_PAR	パラメータエラー (waiptn = 0, wfmode が不正, tmout の値が不正, p_flgptn で指定されたアドレスが不正)
E_OBJ	オブジェクトの状態が不正 (TA_WSGL 属性のイベントフラグに対する複数タスクの待ち)
E_DLT	待ちオブジェクトが削除された (待ちの間に対象イベントフラグが削除された)
E_RLWAI	待ち状態強制解除 (待ちの間に ts_rel_wai を受け付け)
E_DISWAI	待ち禁止による待ち解除
E_TMOUT	ポーリング失敗またはタイムアウト
E_CTX	コンテキストエラー (タスク部および準タスク部以外、ディスパッチ禁止状態、割込み禁止状態で実行)
E_DACV	ドメインアクセス保護違反 (対象イベントフラグは他のドメインに所属してアクセス保護されている)
E_MACV	メモリアクセス違反エラー (p_flgptn に対するアクセス権限がない、または対象のメモリ領域が存在しない)
E_DOMAIN	ドメインエラー

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

wfmode で示される待ち解除の条件に従って、flgid で指定されるイベントフラグのビットパターンがセットされるのを待つ。
対象イベントフラグのビットパターンが、既に wfmode で示される待ち解除条件を満たしている場合には、本 API を発行したタスクは待ち状態にならずに実行を続ける。

wfmode には、イベントフラグを待つ際の条件と、待ち解除条件が成立した際にビットをクリアする方法を指定する。

wfmode := (TWF_ANDW || TWF_ORW) | [TWF_CLR || TWF_BITCLR]

TWF_ANDW	AND 待ち
TWF_ORW	OR 待ち
TWF_CLR	全ビットクリア指定
TWF_BITCLR	条件ビットのみクリア指定

イベントフラグの待ちモードは、ヘッダファイルにおいて以下のように定義されている。

```
#define TWF_ANDW        (0x00000001U)    /* AND wait */
#define TWF_ORW         (0x00000002U)    /* OR wait */
#define TWF_CLR         (0x00000010U)    /* All clear specify */
#define TWF_BITCLR      (0x00000020U)    /* Only condition bit clear specify */
```

TWF_ORW を指定した場合には、flgid で指定されるイベントフラグのビットパターン(EventFlagPattern とする)のうち、waiptn で指定したビットのいずれかがセットされるのを待つ(OR 待ち)。
TWF_ANDW を指定した場合には、flgid で指定されるイベントフラグのビットパターンのうち、waiptn で指定したビットすべてがセットされるのを待つ(AND 待ち)。

TWF_CLR、TWF_BITCLR が指定されていない場合は、イベントフラグの待ち解除条件が満足されて対象タスクが待ち解除となった場合にも、イベントフラグのビットパターンは変化しない。
TWF_CLR が指定された場合は、イベントフラグの待ち解除条件が満足されて対象タスクが待ち解除となった場合、対象イベントフラグのビットパターン(全部のビット)が 0 にクリアされる。
TWF_BITCLR が指定された場合は、イベントフラグの待ち解除条件が満足されて対象タスクが待ち解除となった場合、対象イベントフラグのビットパターンのうち、waiptn で 1 が指定されたビットのみ 0 クリアされる。
C 言語で記述すると以下の式で表現される。

```
EventFlagPattern &= ~waiptn;            /* TWF_BITCLR が指定された場合 */
```

flgptn は、リターンパラメータであり、本 API による待ち状態が解除される時のイベントフラグのビットパターン(TWF_CLR または TWF_BITCLR が指定されている場合は、イベントフラグがクリアされる前のビットパターン)が設定される。すなわち、flgptn で返される値は、本 API 発行時に指定した待ち解除の条件 waiptn を満たすビットパターンになっている。
なお、タイムアウト等でイベントフラグ待ちが解除された場合は、flgptn の内容は不定となる。

tmout には、タイムアウト時間を指定する。指定したタイム時間が経過する間に待ち解除条件が成立しなかった場合は、タイムアウトエラーE_TMOUT となる。
tmout として TMO_FEVR(=-1)を指定した場合は、タイムアウトまでの時間が無限大であることを示す。この場合は、ts_set_flg/tn_set_flg、ts_del_flg/tn_del_flg、ts_rel_wai/tn_rel_wai、ts_dis_wai/tn_dis_wai が発行されるまで永久に待ち状態になる。
安全 API では TMO_FEVR を指定した場合は、ドメインエラーE_DOMAIN となる。
tmout として TMO_POL(=0)を指定した場合は、待ち状態に入ることなくエラーE_TMOUT(ポーリング失敗)となる。
タイムアウトした場合は、TWF_CLR または TWF_BITCLR の指定があってもイベントフラグのビットパターンはクリアされない。
tmout として、-2 以下の値を指定した場合、パラメータエラーE_PAR となる。さらに安全ドメインの場合、tmout として、最大タイムアウト時間を超えて指定した場合、パラメータエラーE_PAR となる。

waiptn を 0 とした場合は、パラメータエラーE_PAR になる。
waiptn = 0 の指定を許した場合、イベントフラグのビットパターンがどのような値に変わってもイベントフラグ待ちの状態が解除されることがないため、このような使い方はエラーとしている。

既に待ちタスクの存在する TA_WSGL 属性のイベントフラグに対して、別のタスクが ts_wai_flg を実行することはできない。この場合は、後から ts_wai_flg/tn_wai_flg を実行したタスクが待ち状態に入るかどうか(待ち解除条件が満たされているかどうか)にかかわらず、後から ts_wai_flg/tn_wai_flg を実行したタスクにはエラーE_OBJ が返される。

一方、TA_WMUL の属性を持つイベントフラグに対しては、同一のイベントフラグに対して複数のタスクが同時に待つことができる。従って、イベントフラグでもタスクが待ち行列を作ることになる。この場合、一回の ts_set_flg/tn_set_flg で複数のタスクが待ち解除となることがある。
TA_WMUL 属性を持つイベントフラグに対して複数のタスクが待ち行列を作った場合、次のような動作をする。

- 待ち行列の順番は FIFO またはタスク優先度順である。(ただし、waiptn や wfmode との関係により、必ずしも行列先頭のタスクから待ち解除になるとは限らない。)
- 待ち行列中にクリア指定のタスクがあれば、対象タスクの待ち状態を解除した後で、イベントフラグのビットパターンの全部、または一部をクリアする。
- クリア指定を行っていたタスクよりも後ろの待ち行列にあったタスクは、クリアされた後のイベントフラグを用いて待ち解除条件を判定することになる。

ts_set_flg によって同じ優先度を持つ複数のタスクが同時に待ち解除となる場合、待ち解除後のタスクの優先順位は、元のイベントフラグの待ち行列の順序を保存する。

8.2.4.6 ts_ref_flg/tn_ref_flg – イベントフラグ状態参照

C 言語インタフェース

【安全 API】 ER ercd = ts_ref_flg(ID flgid, T_RFLG *pk_rflg);

【通常 API】 ER ercd = tn_ref_flg(ID flgid, T_RFLG *pk_rflg);

パラメータ

ID	flgid	イベントフラグ ID
T_RFLG*	pk_rflg	イベントフラグ状態を返す領域へのポインタ

リターンパラメータ

ER	ercd	エラーコード
T_RFLG	rflg	イベントフラグ状態

イベントフラグ状態

```
typedef    struct st_rflg {
           ID      dmnno;      /* ドメイン番号 */
           ID      wtsk;       /* 待ちタスクの ID */
           UINT    flgptn;     /* 現在のイベントフラグのビットパターン */
} T_RFLG;
```

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（flgid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（flgid のイベントフラグが存在しない）
E_PAR	パラメータエラー（pk_rflg で指定されたアドレスが不正）
E_DACV	ドメインアクセス保護違反（対象イベントフラグは他のドメインに所属してアクセス保護されている）
E_MACV	メモリアクセス違反エラー（pk_rflg に対するアクセス権限がない、または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

flgid で指定されるイベントフラグの各種状態を参照し、リターンパラメータとして返す。

dmnno は、属しているドメインの番号を示す。

wtsk は、このイベントフラグで待っているタスクの ID を示す。複数のタスクが待っている場合は、待ち行列の先頭のタスクの ID を返す。待ちタスクが無い場合は wtsk=0 となる。

flgptn は、現在のイベントフラグのビットパターンを示す。

8.2.5　同期・通信機能（ミューテックス）の API

ミューテックス関連の同期・通信機能の詳細は、「3.3.3 ミューテックス」を参照。

8.2.5.1 ts_cre_mtx/tn_cre_mtx – ミューテックス生成

C 言語インタフェース

【安全 API】 ID mtxid = ts_cre_mtx(CONST T_CMTX *pk_cmtx);

【通常 API】 ID mtxid = tn_cre_mtx(CONST T_CMTX *pk_cmtx);

パラメータ

CONST T_CMTX* pk_cmtx ミューテックス生成情報へのポインタ

ミューテックス生成情報

```
typedef struct st_cmtx {
        ATR     mtxatr;         /* ミューテックス属性 */
        PRI     ceilpri;        /* ミューテックスの上限優先度 */
        UB      oname[8];       /* オブジェクト名称 */
} T_CMTX;
```

リターンパラメータ

ID	mtxid	ミューテックス ID
	または	エラーコード

エラーコード

E_LIMIT	ミューテックスの数がシステムの上限を超えた
E_RSATR	予約属性エラー（mtxatr が不正あるいは利用できない）
E_PAR	パラメータエラー（pk_cmtx で指定されたアドレスが不正, ceilpri が不正）
E_CTX	コンテキストエラー（タスク部および準タスク部以外で実行）
E_ONAME	オブジェクト名エラー（指定されたオブジェクト名が既にドメイン内で使用されている）
E_MACV	メモリアクセス違反エラー（pk_cmtx に対するアクセス権限がない、または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

ミューテックスを生成してミューテックス ID 番号を割り当てる。

mtxatr には、対象ミューテックスの動作を指定するためのミューテックス属性を設定する。

mtxatr には、次のような指定を行う。

mtxatr:= (TA_TFIFO || TA_TPRI || TA_INHERIT || TA_CEILING) | [TA_ONAME] | [TA_NODISWAI]

TA_TFIFO	待ちタスクのキューイングは FIFO
TA_TPRI	待ちタスクのキューイングは優先度順
TA_INHERIT	優先度継承プロトコル
TA_CEILING	優先度上限プロトコル
TA_ONAME	オブジェクト名称を指定する
TA_NODISWAI	ts_dis_wai による待ち禁止を拒否する

ミューテックス属性は、ヘッダファイルにおいて以下のように定義されている。

```
#define TA_TFIFO        (0x00000000U)   /* 待ちタスクを FIFO で管理 */
#define TA_TPRI         (0x00000001U)   /* 待ちタスクを優先度順で管理 */
#define TA_INHERIT      (0x00000002U)   /* 優先度継承プロトコル */
#define TA_CEILING      (0x00000003U)   /* 優先度上限プロトコル */
#define TA_ONAME        (0x00000040U)   /* オブジェクト名称を指定 */
#define TA_NODISWAI     (0x00000080U)   /* 待ち禁止拒否 */
```

TA_TFIFO を指定した場合、ミューテックスのタスクの待ち行列は FIFO となる。TA_TPRI、TA_INHERIT、TA_CEILING を指定した場合は、タスクの優先度順となる。
TA_INHERIT を指定した場合は優先度継承プロトコル、TA_CEILING を指定した場合は優先度上限プロトコルが適用される。

TA_CEILING を指定した場合のみ ceilpri が有効となる。
ceilpri には、ミューテックスの上限優先度を設定する。ただし、上限優先度は、ミューテックスが属するドメインの最高実行優先度を超える優先度は設定できない。よって、ceilpri に指定可能な値は、「所属するドメインの最高実行優先度」～TS_LST_PRI である。

TA_ONAME を指定した場合に oname が有効となる。
oname には、対象ミューテックスのオブジェクト名称を設定する。

TA_NODISWAI を指定した場合、ts_dis_wai/tn_dis_wai による待ち禁止は拒否される。

8.2.5.2 ts_del_mtx/tn_del_mtx – ミューテックス削除

C 言語インタフェース

【安全 API】 ER ercd = ts_del_mtx(ID mtxid);

【通常 API】 ER ercd = tn_del_mtx(ID mtxid);

パラメータ

ID	mtxid	ミューテックス ID

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（mtxid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（mtxid のミューテックスが存在しない）
E_DACV	ドメインアクセス保護違反（対象ミューテックスは他のドメインに所属してアクセス保護されている）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

mtxid で指定されたミューテックスを削除する。

対象ミューテックスにおいて条件成立を待っているタスクがあった場合、タスクの待ち状態は解除されエラーE_DLT が返される。

削除するミューテックスが TA_INHERIT または TA_CEILING 属性の場合には、ロックしていたタスクの優先度が変更される場合がある。

8.2.5.3 ts_loc_mtx/tn_loc_mtx – ミューテックスのロック

C 言語インタフェース

【安全 API】 ER ercd = ts_loc_mtx(ID mtxid, TMO tmout);

【通常 API】 ER ercd = tn_loc_mtx(ID mtxid, TMO tmout);

パラメータ

ID	mtxid	ミューテックス ID
TMO	tmout	タイムアウト指定(ミリ秒)

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（mtxid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（mtxid のミューテックスが存在しない）
E_PAR	パラメータエラー（tmout の値が不正）
E_DLT	待ちオブジェクトが削除された（待ちの間に対象ミューテックスが削除）
E_RLWAI	待ち状態強制解除（待ちの間に ts_rel_wai/tn_rel_wai を受け付け）
E_DISWAI	待ち禁止による待ち解除
E_TMOUT	ポーリング失敗またはタイムアウト
E_CTX	コンテキストエラー（タスク部および準タスク部以外、ディスパッチ禁止状態、割込み禁止状態で実行）
E_ILUSE	不正使用（多重ロック、上限優先度違反）
E_DACV	ドメインアクセス保護違反（対象ミューテックスは他のドメインに所属してアクセス保護されている）
E_DOMAIN	ドメインエラー

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

mtxid で指定されたミューテックスをロックする。

ミューテックスがロックできれば、本 API の発行タスクは待ち状態に入らず、実行を継続する。この場合、そのミューテックスはロック状態になる。

すでに対象ミューテックスがロックされている場合は、本 API を発行したタスクは待ち状態に入る。すなわち、そのミューテックスに対する待ち行列につながれる。

自タスクがすでに対象ミューテックスをロックしていた場合には、エラーE_ILUSE(多重ロック)を返す。

tmout には、タイムアウト時間を指定する。指定したタイムアウト時間が経過する間にロックできなかった場合は、タイムアウト

エラーE_TMOUT となる。
tmout として TMO_FEVR(=-1)を指定した場合は、タイムアウトまでの時間が無限大であることを示す。この場合は、ts_wup_tsk/tn_wup_tsk が発行されるまで永久に待ち状態になる。安全 API では TMO_FEVR を指定した場合は、ドメインエラーE_DOMAIN となる。
tmout として TMO_POL(=0)を指定した場合は、待ち状態に入ることなくエラーE_TMOUT(ポーリング失敗)となる。
tmout として、-2 以下の値を指定した場合、パラメータエラーE_PAR となる。さらに安全ドメインの場合、tmout として、最大タイムアウト時間を超えて指定した場合、パラメータエラーE_PAR となる。

TA_INHERIT 属性のミューテックスの場合、自タスクがロック待ち状態になるときに、そのミューテックスをロックしているタスクの現在優先度が自タスクより低ければ、ロックしているタスクの優先度を自タスクと同じ優先度まで引き上げる。ロックを待っているタスクがロックを獲得せずに待ちを終了した場合(タイムアウトなど)、そのミューテックスをロック中のタスクの優先度を、次の内の最も高い優先度まで引き下げる。

a. そのミューテックスでロック待ちしているタスクの現在優先度の内の最も高い優先度。
b. そのミューテックスをロック中のタスクがロックしている他のすべてのミューテックスの内の最も高い優先度。
c. ロック中のタスクのベース優先度。

TA_CEILING 属性のミューテックスの場合、自タスクがロックを獲得したときに、自タスクの現在優先度がミューテックスの上限優先度より低ければ、自タスクの優先度をミューテックスの上限優先度まで引き上げる。自タスクのベース優先度が対象ミューテックスの上限優先度より高い場合にはエラーE_ILUSE(上限優先度違反)を返す。
自タスクのベース優先度とは、ミューテックスによって自動的に引き上げられる前のタスクの優先度を示す。最後(ミューテックスのロック中も含む)に ts_chg_pri/tn_chg_pri によって設定された優先度、または ts_chg_pri/tn_chg_pri を一度も発行していない場合はタスク生成時に指定したタスク優先度が、ベース優先度である。

【補足事項】
ミューテックスの多重ロックを許さないのは以下の理由による。
ミューテックスのロック解除はロックしたタスクからしか行えない。つまり、既に対象ミューテックスをロックしているタスクが、再度同じミューテックスをロックしようとしてミューテックスのロック待ち状態になってしまうと、ミューテックスのロックを解除できないことになる。このため、このような操作は仕様として禁止している。

8.2.5.4 ts_unl_mtx/tn_unl_mtx – ミューテックスのアンロック

C 言語インタフェース

【安全 API】 ER ercd = ts_unl_mtx(ID mtxid);

【通常 API】 ER ercd = tn_unl_mtx(ID mtxid);

パラメータ

ID	mtxid	ミューテックス ID

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（mtxid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（mtxid のミューテックスが存在しない）
E_ILUSE	不正使用（自タスクがロックしたミューテックスではない）
E_CTX	コンテキストエラー（タスク部および準タスク部以外で実行）
E_DACV	ドメインアクセス保護違反（対象ミューテックスは他のドメインに所属してアクセス保護されている）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

mtxid で指定されたミューテックスのロックを解除する。
対象ミューテックスにおいてロック待ち状態になっているタスクがあれば、待ち行列の先頭のタスクの待ちを解除し、そのタスクをロック獲得状態にする。

自タスクがロックしていないミューテックスを指定した場合、エラーE_ILUSE を返す。

ロック解除したミューテックスが TA_INHERIT または TA_CEILING 属性の場合、以下のようにタスク優先度が変更される。
ロックを解除することにより、自タスクがロックしているミューテックスがすべてなくなった場合は、自タスクの優先度はベース優先度まで引き下げられる。
自タスクがロック中のミューテックスが残っている場合、自タスクの優先度は次の内の最も高い優先度まで引き下げられる。

a.自タスクがロックしている TA_INHERIT 属性を持つミューテックスの待ち行列につながれているタスクの現在優先度の中で最も高い優先度
b.自タスクがロックしている TA_CEILING 属性を持つミューテックスに設定されている上限優先度の中で最も高い優先度
c.自タスクのベース優先度

ミューテックスをロックした状態でタスクを終了した(休止状態(DORMANT)または未登録状態(NON-EXISTENT)になった)場合、当該タスクがロックしているすべてのミューテックスはカーネルによって自動的にロック解除される。

8.2.5.5 ts_ref_mtx/tn_ref_mtx – ミューテックス状態参照

C 言語インタフェース

【安全 API】 ER ercd = ts_ref_mtx(ID mtxid, T_RMTX *pk_rmtx);

【通常 API】 ER ercd = tn_ref_mtx(ID mtxid, T_RMTX *pk_rmtx);

パラメータ

ID	mtxid	ミューテックス ID
T_RMTX*	pk_rmtx	ミューテックス状態を返す領域へのポインタ

リターンパラメータ

ER	ercd	エラーコード
T_RMTX	rmtx	ミューテックス状態

ミューテックス状態

```
typedef struct st_rmtx {
        ID      dmnno;          /* ドメイン番号 */
        ID      htsk;           /* ロックしているタスクの ID */
        ID      wtsk;           /* ロック待ちタスクの ID */
} T_RMTX;
```

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（mtxid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（mtxid のミューテックスが存在しない）
E_PAR	パラメータエラー（pk_rmtx で指定されたアドレスが不正）
E_DACV	ドメインアクセス保護違反（対象ミューテックスは他のドメインに所属してアクセス保護されている）
E_MACV	メモリアクセス違反エラー（pk_rmtx に対するアクセス権限がない、または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

mtxid で示された対象ミューテックスの各種の状態を参照し、リターンパラメータを返す。

dmnno は、属しているドメインの番号を示す。

htsk は、このミューテックスをロックしているタスクの ID を示す。ロックしているタスクがない場合は htsk=0 となる。

wtsk は、このミューテックスで待っているタスクの ID を示す。複数のタスクが待っている場合は、待ち行列の先頭のタスクの ID を返す。待ちタスクが無い場合は wtsk=0 となる。

8.2.6　同期・通信機能（メッセージバッファ）の API

メッセージバッファ関連の同期・通信機能の詳細は、「3.3.4 メッセージバッファ」を参照。

8.2.6.1 ts_cre_mbf/tn_cre_mbf – メッセージバッファ生成

C 言語インタフェース

【安全 API】 ID mbfid = ts_cre_mbf(CONST T_CMBF *pk_cmbf);

【通常 API】 ID mbfid = tn_cre_mbf(CONST T_CMBF *pk_cmbf);

パラメータ

CONST T_CMBF* pk_cmbf メッセージバッファ生成情報へのポインタ

メッセージバッファ生成情報

```
typedef struct st_cmbf {
        ATR     mbfatr;         /* メッセージバッファ属性 */
        INT     mbfno;          /* メッセージバッファ管理番号 */
        INT     bufsz;          /* メッセージバッファのサイズ（バイト数）*/
        INT     maxmsz;         /* メッセージの最大長（バイト数）*/
        UB      oname[8];       /* オブジェクト名称 */
} T_CMBF;
```

リターンパラメータ

ID	mbfid	メッセージバッファ ID
	または	エラーコード

エラーコード

E_LIMIT	メッセージバッファの数がシステムの上限を超えた
E_RSATR	予約属性エラー（mbfatr が不正あるいは利用できない）
E_PAR	パラメータエラー（pk_cmbf で指定されたアドレスが不正, mbfno, bufsz, maxmsz が負または不正）
E_NOMEM	メモリ不足エラー（メッセージバッファが使用できるメモリ領域より bufsz が大きい）
E_CTX	コンテキストエラー（タスク部および準タスク部以外で実行）
E_ONAME	オブジェクト名エラー（指定されたオブジェクト名が既にドメイン内で使用されている）
E_MACV	メモリアクセス違反エラー（pk_cmbf に対するアクセス権限がない、または対象のメモリ領域が存在しない、または mbfno で指定したメモリ領域が自ドメインのデータ領域でない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

メッセージバッファを生成してメッセージバッファ ID 番号を割り当てる。

mbfatr には、対象メッセージバッファの動作を指定するためのメッセージバッファ属性を設定する。

mbfatr には、次のような指定を行う。

mbfatr := (TA_TFIFO || TA_TPRI) | [TA_ONAME] | [TA_NODISWAI]

TA_TFIFO	送信待ちタスクのキューイングは FIFO
TA_TPRI	送信待ちタスクのキューイングは優先度順
TA_ONAME	オブジェクト名称を指定する
TA_NODISWAI	ts_dis_wai/tn_dis_wai による待ち禁止を拒否する

メッセージバッファ属性は、ヘッダファイルにおいて以下のように定義されている。

```
#define TA_TFIFO        (0x00000000U)    /* 送信待ちタスクを FIFO で管理 */
#define TA_TPRI         (0x00000001U)    /* 送信待ちタスクを優先度順で管理 */
#define TA_ONAME        (0x00000040U)    /* オブジェクト名称を指定 */
#define TA_NODISWAI     (0x00000080U)    /* 待ち禁止拒否 */
```

TA_TFIFO、TA_TPRI では、バッファが一杯の場合にメッセージを送信するタスクがメッセージ送信の待ち行列に並ぶ際の並び方を指定することができる。属性が TA_TFIFO であればタスクの待ち行列は FIFO となり、属性が TA_TPRI であればタスクの待ち行列はタスクの優先度順となる。
メッセージ受信待ちのタスクの待ち行列の順序は FIFO のみである。

TA_ONAME を指定した場合に oname が有効となる。
oname には、対象メッセージバッファのオブジェクト名称を設定する。

TA_NODISWAI を指定した場合、ts_dis_wai/tn_dis_wai による待ち禁止は拒否される。

mbfno には、コンフィギュレーションで設定したメッセージバッファ管理情報の番号を指定する。メッセージバッファ管理情報には、メッセージバッファで使用するメモリ資源の情報が設定されている。ただし、bufsz=0 の場合は、mbfno は指定しなくてよい。指定した値は無視される。
既に他のメッセージバッファで使用されている番号を指定した場合はエラーE_PAR が返される。
また、指定したメッセージバッファ管理情報に、生成するメッセージバッファから使用できないメモリ領域が設定されている（自ドメインのデータ領域でない）場合は、エラーE_MACV を返す。

bufsz には、対象メッセージバッファが使用するバッファ領域のサイズ(バイト数)を指定する。
使用するバッファ領域にはメッセージだけでなく、個々のメッセージを管理するための情報も合わせて格納する。このため、bufsz で指定されたバッファ領域のサイズとキューに入るメッセージのサイズの合計は一致しない。後者の方が小さい値をとることになる。
bufsz に指定可能な値は 0 以上の値である。負の値は設定できない。最大値は mbfno で指定したメッセージバッファ管理情報のメモリ領域のサイズ以下でなくてはならない。超えた場合はエラーE_NOMEM を返す。
bufsz=0 のメッセージバッファを生成することも可能である。この場合、このメッセージバッファでは送受信側が完全に同期した通信を行うことになる。すなわち、ts_snd_mbf/tn_snd_mbf と ts_rcv_mbf/tn_rcv_mbf の一方のシステムコールが先に実行される

と、それを実行したタスクは待ち状態となる。もう一方のシステムコールが実行された段階で、メッセージの受け渡し(コピー)が行われ、その後双方のタスクが実行を再開する。

maxmsz には、送受信するメッセージの最大長(バイト数)を指定する。maxmsz には、1 以上の値を設定する。最大値はコンフィギュレーションで設定される。
送信側では、送信メッセージのサイズは maxmsz で指定した値以下にしなければならない。送信メッセージのサイズが maxmsz を超えていた場合はエラーとなる。
受信側では、maxmsz で指定したバイト数以上のサイズの受信バッファ領域を用意しなければならない。

maxmsz は、bufsz よりも大きな値を指定しても構わない。
例えば、maxmsz が bufsz よりも大きなメッセージバッファに対して、bufsz を超えるサイズのメッセージを送信しようとした場合、送信メッセージはバッファ領域に入りきらないことになる。この時、対象メッセージバッファにおいて受信待ち状態のタスクがあれば、メッセージは送信バッファから受信バッファに直接コピーされる。一方、対象メッセージバッファにおいて受信待ち状態のタスクがなければ、送信側タスクは一旦メッセージ送信待ち状態となった後、対象メッセージバッファに対するメッセージの受信が実行された時点で、メッセージが送信バッファから受信バッファに直接コピーされる。ただし、既にバッファ領域にメッセージが格納されていた場合は、格納されていたメッセージが先に全て処理されていなければならない(メッセージキューの並び順は FIFO のみである)。

8.2.6.2 ts_del_mbf/tn_del_mbf – メッセージバッファ削除

C 言語インタフェース

【安全 API】 ER ercd = ts_del_mbf(ID mbfid);

【通常 API】 ER ercd = tn_del_mbf(ID mbfid);

パラメータ

ID	mbfid	メッセージバッファ ID

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（mbfid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（mbfid のメッセージバッファが存在しない）
E_DACV	ドメインアクセス保護違反（対象メッセージバッファは他のドメインに所属してアクセス保護されている）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

mbfid で指定されたメッセージバッファを削除する。

対象メッセージバッファの中にメッセージが残っている場合でもエラーとはならず、メッセージバッファの削除が行われ、中にあったメッセージは消滅する。

対象メッセージバッファにおいて条件成立を待っているタスクがあった場合、タスクの待ち状態は解除されエラーE_DLT が返される。

8.2.6.3 ts_snd_mbf/tn_snd_mbf – メッセージバッファへ送信

C 言語インタフェース

【安全 API】 ER ercd = ts_snd_mbf(ID mbfid, CONST void *msg, INT msgsz, TMO tmout);

【通常 API】 ER ercd = tn_snd_mbf(ID mbfid, CONST void *msg, INT msgsz, TMO tmout);

パラメータ

ID	mbfid	メッセージバッファ ID
CONST void*	msg	送信メッセージの先頭アドレス
INT	msgsz	送信メッセージのサイズ（バイト数）
TMO	tmout	タイムアウト指定（ミリ秒）

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（mbfid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（mbfid のメッセージバッファが存在しない）
E_PAR	パラメータエラー（msg で指定されたアドレスが不正, msgsz=<0, maxmsz<msgsz, tmout の値が不正）
E_DLT	待ちオブジェクトが削除された(待ちの間に対象メッセージバッファが削除)
E_RLWAI	待ち状態強制解除（待ちの間に ts_rel_wai を受け付け）
E_DISWAI	待ち禁止による待ち解除
E_TMOUT	ポーリング失敗またはタイムアウト
E_CTX	コンテキストエラー（タスク部および準タスク部以外、ディスパッチ禁止状態、割込み禁止状態で実行）
E_DACV	ドメインアクセス保護違反（対象メッセージバッファは他のドメインに所属してアクセス保護されている）
E_MACV	メモリアクセス違反エラー（msg に対するアクセス権限がない。または対象のメモリ領域が存在しない）
E_DOMAIN	ドメインエラー

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

mbfid で指定されるメッセージバッファに対してメッセージを送信する。

msg には、送信するメッセージを格納してあるメモリ領域へのポインタを指定する。

msgsz には、メッセージのサイズ(バイト数)を指定する。msgsz に指定可能な値は 1～maxmsz である。

すなわち、msg 以下の msgsz バイトが、mbfid で指定されたメッセージバッファのバッファ領域にコピーされ、メッセージキューにつながれることになる。

バッファの空き領域が少なく、メッセージをバッファ領域にコピーできない場合、本 API を発行したタスクはメッセージ送信待ち状態となり、バッファの空きを待つための待ち行列(送信待ち行列)につながれる。送信待ち行列の順序は ts_cre_mbf/tn_cre_mbf 時の指定により FIFO またはタスク優先度順となる。

tmout には、タイムアウト時間を指定する。指定したタイムアウト時間が経過する間にメッセージを送信できなかった場合は、タイムアウトエラーE_TMOUT となる。
tmout として TMO_FEVR(=-1)を指定した場合は、タイムアウトまでの時間が無限大であることを示す。この場合は、メッセージが送信されるまで永久に待ち状態になる。安全 API では TMO_FEVR を指定した場合は、ドメインエラーE_DOMAIN となる。
tmout として TMO_POL(=0)を指定した場合は、待ち状態に入ることなくエラーE_TMOUT(ポーリング失敗)となる。
tmout として、-2 以下の値を指定した場合、パラメータエラーE_PAR となる。さらに安全ドメインの場合、tmout として、最大タイムアウト時間を超えて指定した場合、パラメータエラーE_PAR となる。

8.2.6.4 ts_rcv_mbf/tn_rcv_mbf – メッセージバッファから受信

C 言語インタフェース

【安全 API】 INT msgsz = ts_rcv_mbf(ID mbfid, void *msg, TMO tmout);

【通常 API】 INT msgsz = tn_rcv_mbf(ID mbfid, void *msg, TMO tmout);

パラメータ

ID	mbfid	メッセージバッファ ID
void*	msg	受信メッセージを入れるアドレス
TMO	tmout	タイムアウト指定（ミリ秒）

リターンパラメータ

INT	msgsz	受信したメッセージのサイズ(バイト数)
	または	エラーコード

エラーコード

E_ID	不正 ID 番号（mbfid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（mbfid のメッセージバッファが存在しない）
E_PAR	パラメータエラー（msg で指定されたアドレスが不正, tmout の値が不正）
E_DLT	待ちオブジェクトが削除された（待ちの間に対象メッセージバッファが削除）
E_RLWAI	待ち状態強制解除（待ちの間に ts_rel_wai/tn_rel_wai を受け付け）
E_DISWAI	待ち禁止による待ち解除
E_TMOUT	ポーリング失敗またはタイムアウト
E_CTX	コンテキストエラー（タスク部および準タスク部以外、ディスパッチ禁止状態、割込み禁止状態で実行）
E_DACV	ドメインアクセス保護違反（対象メッセージバッファは他のドメインに所属してアクセス保護されている）
E_MACV	メモリアクセス違反エラー（msg に対するアクセス権限がない。または対象のメモリ領域が存在しない）
E_DOMAIN	ドメインエラー

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

mbfid で指定されるメッセージバッファからメッセージを受信し、msg で指定したメモリ領域に入れる。
すなわち、対象メッセージバッファのメッセージキューの先頭のメッセージを、msg で指定されたメモリ領域にコピーする。コピーされたメッセージはメッセージキューから削除される。

msg には、受信したメッセージを格納するためのメモリ領域へのポインタを指定する。
msg で指定するメモリ領域は、ts_cre_mbf/tn_cre_mbf で指定した maxmsz で指定したバイト数以上のサイズが確保されていなければならない。

対象メッセージバッファのメッセージキューが空の場合には、本 APIを発行したタスクは待ち状態となり、メッセージの到着を待つ待ち行列(受信待ち行列)につながれる。受信待ちタスクの待ち行列は FIFO のみである。

tmout には、タイムアウト時間を指定する。
指定したタイムアウト時間が経過する間にメッセージが受信できなかった場合は、タイムアウトエラーE_TMOUT となる。
tmout として TMO_FEVR(=-1)を指定した場合は、タイムアウトまでの時間が無限大であることを示す。この場合は、メッセージが受信できるまで永久に待ち状態になる。安全 API では TMO_FEVR を指定した場合は、ドメインエラーE_DOMAIN となる。
tmout として TMO_POL(=0)を指定した場合は、待ち状態に入ることなくエラーE_TMOUT(ポーリング失敗)となる。
tmout として、-2 以下の値を指定した場合、パラメータエラーE_PAR となる。さらに安全ドメインの場合、tmout として、最大タイムアウト時間を超えて指定した場合、パラメータエラーE_PAR となる。

8.2.6.5 ts_ref_mbf/tn_ref_mbf – メッセージバッファ状態参照

C 言語インタフェース

【安全 API】 ER ercd = ts_ref_mbf(ID mbfid, T_RMBF *pk_rmbf);

【通常 API】 ER ercd = tn_ref_mbf(ID mbfid, T_RMBF *pk_rmbf);

パラメータ

ID	mbfid	メッセージバッファ ID
T_RMBF*	pk_rmbf	メッセージバッファ状態を返す領域へのポインタ

リターンパラメータ

ER	ercd	エラーコード
T_RMBF	rmbf	メッセージバッファ状態

メッセージバッファ状態

```
typedef struct st_rmbf {
        ID      dmnno;      /* ドメイン番号 */
        ID      wtsk;       /* 受信待ちタスクの ID */
        ID      stsk;       /* 送信待ちタスクの ID */
        INT     msgsz;      /* 次に受信されるメッセージのサイズ（バイト数）*/
        INT     frbufsz;    /* 空きバッファのサイズ（バイト数）*/
        INT     maxmsz;     /* メッセージの最大長（バイト数）*/
} T_RMBF;
```

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（mbfid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（mbfid のメッセージバッファが存在しない）
E_PAR	パラメータエラー（pk_rmbf で指定されたアドレスが不正）
E_DACV	ドメインアクセス保護違反（対象メッセージバッファは他のドメインに所属してアクセス保護されている）
E_MACV	メモリアクセス違反エラー（pk_rmbf に対するアクセス権限がない。または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

mbfid で指定されるメッセージバッファの各種の状態を参照し、リターンパラメータとして返す。

dmnno は、属しているドメインの番号を示す。

wtsk には、対象メッセージバッファで受信待ち行列の先頭につながれているタスクのタスク ID が返る。受信待ちタスクが無い場合は wtsk=0 となる。

stsk には、対象メッセージバッファで送信待ち行列の先頭につながれているタスクのタスク ID が返る。送信待ちタスクが無い場合は stsk=0 となる。

msgsz には、メッセージキューの先頭のメッセージ(次に受信されるメッセージ)のサイズが返る。メッセージキューにメッセージが無い場合には、msgsz=0 となる。

frbufsz には、メッセージキューを構成するバッファ領域の空きのサイズが返る。この値は、あとどの程度の量のメッセージを送信できるかを知る手掛かりになる。ただし、バッファ領域にはメッセージを管理するための情報も格納されるため、frbufsz のメッセージを送信できるとは限らない。

maxmsz には、ts_cre_mbf/tn_cre_mbf で指定したメッセージの最大長を返す。

8.2.7　時間管理機能の API

時間管理機能の詳細は、「3.4 時間管理機能」を参照。

8.2.7.1 ts_set_tim/tn_set_tim – システム時刻設定

C 言語インタフェース

【安全 API】 ER ercd = ts_set_tim(CONST SYSTIM *pk_tim);

【通常 API】 ER ercd = tn_set_tim(CONST SYSTIM *pk_tim);

パラメータ

CONST SYSTIM*	pk_tim	現在時刻(ミリ秒)を示すパケット tim へのポインタ

tim の内容

W	hi	システム時刻設定用現在時刻の上位 32 ビット
UW	lo	システム時刻設定用現在時刻の下位 32 ビット

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_PAR	パラメータエラー（pk_tim で指定されたアドレスが不正、設定時間が不正）
E_MACV	メモリアクセス違反エラー（pk_tim に対するアクセス権限がない。または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

tim で指定される値を現在時刻としてシステム時刻に設定する。

システム時刻は、1985 年 1 月 1 日 0 時(GMT)からの通算のミリ秒数であり、64 ビットの符号付整数で表す。

tim に指定可能な値は、0～9,223,372,036,854,775,807 (=0x7FFFFFFFFFFFFFFF) である。負の値を指定した場合はエラー E_PAR となる。

システムの動作中に本 API を使ってシステム時刻を変更した場合にも、RELTIM や TMO で指定された相対時間は変化しない。

例えば、60 秒後にタイムアウトする様に指定した場合、タイムアウト待ちの間に本 API で時間を 60 秒進めてもそこでタイムアウトすることはなく、60 秒後にタイムアウトする。従って、本 API によってタイムアウトするシステム時刻は変化することになる。

【補足事項】

本 API により設定される時刻が、システムタイマの周期時間によって制約を受けることはないが、その後の ts_get_tim/tn_get_tim で読み出される時刻は、システムタイマの時間分解能で変化する。例えば、システムタイマの周期が 10

ミリ秒のシステムにおいて、本 API で 5(ミリ秒)の時刻を設定した場合に、その後の ts_get_tim/tn_get_tim で得られる時刻は、5(ミリ秒)→15(ミリ秒)→25(ミリ秒)のように変化する。

8.2.7.2 ts_get_tim/tn_get_tim – システム時刻参照

C 言語インタフェース

【安全 API】 ER ercd = ts_get_tim(SYSTIM *pk_tim);

【通常 API】 ER ercd = tn_get_tim(SYSTIM *pk_tim);

パラメータ

SYSTIM*	pk_tim	現在時刻 tim を返す領域へのポインタ

リターンパラメータ

ER	ercd	エラーコード
SYSTIM	tim	現在時刻

tim の内容

W	hi	システムの現在時刻の上位 32 ビット
UW	lo	システムの現在時刻の下位 32 ビット

エラーコード

E_OK	正常終了
E_PAR	パラメータエラー（pk_tim で指定されたアドレスが不正）
E_MACV	メモリアクセス違反エラー（pk_tim に対するアクセス権限がない。または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

システム時刻の現在の値を読み出し、リターンパラメータ tim に返す。

システム時刻は、1985 年 1 月 1 日 0 時(GMT)からの通算のミリ秒数であり、64 ビットの符号付整数で表す。

【補足事項】

本システムコールで読み出されるシステムの現在時刻の分解能は、システムタイマの周期時間の分解能で決められる。具体的な値は実装定義である。

システムタイマの周期時間より短い時間経過を知ることはできない。

8.2.7.3 ts_get_otm/tn_get_otm – システム稼働時間参照

C 言語インタフェース

【安全 API】 ER ercd = ts_get_otm(SYSTIM *pk_tim);

【通常 API】 ER ercd = tn_get_otm(SYSTIM *pk_tim);

パラメータ

SYSTIM*	pk_tim	Packet of Operating Time	稼働時間 tim を返す領域へのポインタ

リターンパラメータ

ER	ercd	エラーコード
SYSTIM	tim	稼働時間

tim の内容

W	hi	システム稼働時間の上位 32 ビット
UW	lo	システム稼働時間の下位 32 ビット

エラーコード

E_OK	正常終了
E_PAR	パラメータエラー（pk_tim で指定されたアドレスが不正）
E_MACV	メモリアクセス違反エラー（pk_tim に対するアクセス権限がない。または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

システム稼働時間を取得し、リターンパラメータ tim に返す。

システム稼働時間はシステム時刻(時刻)と異なり、システム起動時からの単純増加する稼働時間を表す。

ts_set_tim/tn_set_tim による時刻設定に影響されない。

システム稼働時間はシステム時刻と同じ精度になる。

8.2.7.4 ts_cre_cyc/tn_cre_cyc – 周期ハンドラの生成

C 言語インタフェース

【安全 API】 ID cycid = ts_cre_cyc(CONST T_CCYC *pk_ccyc);

【通常 API】 ID cycid = tn_cre_cyc(CONST T_CCYC *pk_ccyc);

パラメータ

CONST T_CCYC* pk_ccyc 周期ハンドラ定義情報へのポインタ

周期ハンドラ定義情報

```
typedef struct st_ccyc {
        ATR     cycatr;         /* 周期ハンドラ属性 */
        INT     stacd;          /* 周期ハンドラ起動コード */
        FP      cychdr;         /* 周期ハンドラ起動アドレス */
        UINT    ustkno;         /* ユーザスタックの資源番号 */
        INT     ustksz;         /* ユーザスタックサイズ（バイト数）*/
        UINT    sstkno;         /* システムスタックの資源番号 */
        INT     sstksz;         /* システムスタックサイズ（バイト数）*/
        RELTIM  cyctim;         /* 周期起動時間間隔（ミリ秒）*/
        RELTIM  cycphs;         /* 周期起動位相（ミリ秒）*/
        RELTIM  maxrtim;        /* 最大連続実行時間 */
        UB      oname[8];       /* オブジェクト名称 */
} T_CCYC;
```

リターンパラメータ

ID	cycid	周期ハンドラ ID
	または	エラーコード

エラーコード

E_NOMEM	メモリ不足（指定したサイズのスタックがスタック用領域から確保できない）
E_LIMIT	周期ハンドラの数がシステムの制限を超えた
E_RSATR	予約属性エラー（cycatr が不正あるいは利用できない）
E_PAR	パラメータエラー（pk_ccyc, cychdr で指定されたアドレスが不正。または maxrtim が自ドメインの連続実行時間上限や cyctim より大きい）
E_CTX	コンテキストエラー（タスク部および準タスク部以外で実行）
E_ONAME	オブジェクト名エラー（指定されたオブジェクト名が既にドメイン内で使用されている）
E_MACV	メモリアクセス違反エラー（pk_ccyc、cychdr に対するアクセス権限がない。または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

周期ハンドラを生成して周期ハンドラ ID を割り当てる。

cycatr には、対象周期ハンドラの動作を指定するための周期ハンドラ属性を設定する。
cycatr には、次のような指定を行う。

cycatr := [TA_STA] | [TA_PHS] | [TA_ONAME]

TA_STA	周期ハンドラ生成後、動作状態とする
TA_PHS	起動位相を保存する
TA_ONAME	オブジェクト名称を指定する

```
#define TA_STA          (0x00000002U)   /* 周期ハンドラ起動 */
#define TA_PHS          (0x00000004U)   /* 周期ハンドラ起動位相を保存 */
#define TA_ONAME        (0x00000040U)   /* オブジェクト名称を指定 */
```

TA_STA を指定した場合は、周期ハンドラの生成時から周期ハンドラは動作状態となり、指定された時間間隔で周期ハンドラが起動される。
TA_STA を指定しなかった場合は、周期ハンドラは停止状態となり、起動周期の計測は行われるが、周期ハンドラは起動されない。

周期ハンドラは C 言語の関数として以下の形式で記述する。

```
void    cychdr( INT stacd);
```

上記の関数へのポインタを、周期ハンドラ起動アドレス cychdr に指定する。
周期ハンドラの起動パラメータとして、周期ハンドラ起動コード stacd が引数に渡される。

周期ハンドラはユーザスタックとシステムスタックを持つ。ただし、システムドメインの周期ハンドラは、システムスタックのみであり、ユーザスタックはもたない。よって、システムドメインの周期ハンドラ生成では、ユーザスタックの指定は無視される。
ユーザスタックは、周期ハンドラが所属するドメインのスタック用領域からメモリが確保される。ustkno にコンフィギュレーションで設定したスタック資源番号を指定する。ustksz にユーザスタックのサイズを指定する。ただし、通常ドメインではメモリ領域は動的に確保するので、通常 API では utskno の値は無視される。安全 API では ustksz は、ustkno で指定したスタック資源のメモリサイズ以下でなければならない。
システムスタックは、システムドメインのスタック用領域からメモリが確保される。sstkno にコンフィギュレーションで設定したスタック資源番号を指定する。sstksz にシステムスタックのサイズを指定する。sstksz は、sstkno で指定したスタック資源のメモリ

サイズ以下でなければならない。
以下の理由でスタックが確保できなかった場合、エラーE_NOMEM となる。

- 指定したスタックサイズが、スタック資源のサイズより大きい
- 指定したスタック資源が既にタスクまたはタイムイベントハンドラで使用されている
- 通常 API においてユーザスタックが確保できなかった（ユーザスタック領域の残りサイズが不足）

cyctim は周期起動の時間間隔、cycphs は起動位相の時間（ミリ秒）を指定する。
cyctim に指定可能な値は、1～TS_MAX_CYCTIM である。cycphs に指定可能な値は、0～TS_MAX_CYCPHS である。
TS_MAX_CYCTIM および TS_MAX_CYCPHS は実装定義である。
cyctim と cycphs の値により、周期ハンドラが起動する時刻が決定される。本 API により周期ハンドラが生成されてから cycphs 時間後が最初の周期ハンドラの起動時刻となる。その後は、cyctim 間隔で繰り返し起動時刻となる。
cycphs に 0 を指定した場合は、周期ハンドラの生成直後が周期ハンドラの起動時刻となる。cyctim に 0 を指定することはできない。周期ハンドラの n 回目の起動時刻は、周期ハンドラを生成してから cycphs + cyctim * (n − 1) 以上の時間が経過した時刻である。
ただし、TA_STA が指定されていない場合は、起動周期の計測は行われるが、周期ハンドラは起動されない。

TA_PHS が指定されている場合は、ts_sta_cyc/tn_sta_cyc が発行されても周期ハンドラの起動位相を保存する。すなわち、ts_sta_cyc/tn_sta_cyc により周期ハンドラが動作状態となっても、起動周期はリセットされず、周期ハンドラの生成時から計測している周期を維持する。
TA_PHS が指定されていない場合は、ts_sta_cyc/tn_sta_cyc によって起動周期がリセットされ、ts_sta_cyc/tn_sta_cyc の呼び出しから cyctim 間隔で周期ハンドラが起動される。ts_sta_cyc/tn_sta_cyc によるリセットでは、cycphs は適用されない。この場合、ts_sta_cyc/tn_sta_cyc から n 回目の周期ハンドラの起動は、ts_sta_cyc/tn_sta_cyc 呼び出し時から cyctim*n 以上の時間が経過した時刻となる。

maxrtim には、周期ハンドラが連続して実行できる最大時間をミリ秒単位で設定する。maxrtim の最小値は 1 ミリ秒である。
maxrtim の最大値は自ドメインの連続実行時間上限の値である。また、maxrtim は cyctim 以下の値でなければならない。
maxrtim が自ドメインの連続実行時間上限または cyctim より大きな値を指定した場合はエラーE_PAR となる。
周期ハンドラが maxrtim を超えて連続実行した場合は異常例外が発生し、周期ハンドラの実行は中断される。

TA_ONAME を指定した場合に oname が有効となる。
oname には、対象周期ハンドラのオブジェクト名称を設定する。

周期ハンドラは、所属するドメインの最高実行優先度で動作し、同一優先度のタスクよりも優先される。つまり、同一優先度のタスク実行中に周期ハンドラの起動時刻となった場合は、タスクの動作は中断され周期ハンドラが起動する。また、周期ハンドラが自ドメインの他のタイムイベントハンドラやタスクに実行を中断されることはない。

【補足事項】
周期ハンドラは自ドメインの中では最優先で実行され、また実行を奪われることもない。ただし、他のドメインのより優先度の高いタスクやタイムイベントハンドラの方が優先される。通常ドメインは安全ドメインより実行優先度が低いので、通常ドメインの周期ハンドラは、安全ドメインのタスクやタイムイベントハンドラより低い優先度となる。これは通常ドメインのソフトにより安

全ドメインのソフトの実行が阻害されることを防ぐためである。

8.2.7.5 ts_del_cyc/tn_del_cyc – 周期ハンドラの削除

C 言語インタフェース

【安全 API】 ER ercd = ts_del_cyc(ID cycid);

【通常 API】 ER ercd = tn_del_cyc(ID cycid);

パラメータ

ID	cycid	周期ハンドラ ID

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（cycid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（cycid の周期ハンドラが存在しない）
E_DACV	ドメインアクセス保護違反（対象周期ハンドラは他のドメインに所属してアクセス保護されている）
E_CTX	コンテキストエラー（タスク部および準タスク部以外で実行）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

cycid で指定された周期ハンドラを削除する。

周期ハンドラが動作状態でも削除することはできる。

8.2.7.6 ts_sta_cyc/tn_sta_cyc – 周期ハンドラの動作開始

C 言語インタフェース

【安全 API】 ER ercd = ts_sta_cyc(ID cycid);

【通常 API】 ER ercd = tn_sta_cyc(ID cycid);

パラメータ

ID	cycid	周期ハンドラ ID

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（cycid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（cycid の周期ハンドラが存在しない）
E_DACV	ドメインアクセス保護違反（対象周期ハンドラは他のドメインに所属してアクセス保護されている）
E_CTX	コンテキストエラー（タスク部および準タスク部以外で実行）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

cycid で指定された周期ハンドラを動作状態にする。

周期ハンドラの生成時に、周期ハンドラ属性に TA_PHS 属性を指定している場合は、周期ハンドラの起動周期はリセットされず、周期ハンドラの生成時から計測している周期を維持する。cycid で指定された周期ハンドラがすでに動作状態になっている場合は、何もせずに動作状態を維持する。

周期ハンドラ属性に TA_PHS 属性を指定していない場合は、起動周期をリセットしたうえで、周期ハンドラを動作状態にする。cycid で指定された周期ハンドラがすでに動作状態になっている場合は、起動周期をリセットした上で、動作状態を維持する。従って、次に周期ハンドラが起動されるのは、本 API を発行してから cyctim 後となる。

8.2.7.7 ts_stp_cyc/tn_stp_cyc – 周期ハンドラの動作停止

C 言語インタフェース

【安全 API】 ER ercd = ts_stp_cyc(ID cycid);

【通常 API】 ER ercd = tn_stp_cyc(ID cycid);

パラメータ

ID	cycid	周期ハンドラ ID

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（cycid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（cycid の周期ハンドラが存在しない）
E_DACV	ドメインアクセス保護違反（対象周期ハンドラは他のドメインに所属してアクセス保護されている）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

cycid で指定された周期ハンドラを停止状態にする。

周期ハンドラがすでに停止状態になっている場合は何もしない。

8.2.7.8 ts_ref_cyc/tn_ref_cyc – 周期ハンドラ状態参照

C 言語インタフェース

【安全 API】 ER ercd = ts_ref_cyc(ID cycid, T_RCYC *pk_rcyc);

【通常 API】 ER ercd = tn_ref_cyc(ID cycid, T_RCYC *pk_rcyc);

パラメータ

ID	cycid	周期ハンドラ ID
T_RCYC*	pk_rcyc	周期ハンドラ状態を返す領域へのポインタ

リターンパラメータ

ER	ercd	エラーコード
T_RCYC	rcyc	周期ハンドラ状態

周期ハンドラ状態

```
typedef struct st_rcyc {
        ID      dmnno;    /* ドメイン番号 */
        PRI     cycpri;   /* 周期ハンドラの優先度 */
        RELTIM  lfttim;   /* 次のハンドラ起動までの残り時間（ミリ秒）*/
        UINT    cycstat;  /* 周期ハンドラの状態 */
} T_RCYC;
```

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（cycid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（cycid の周期ハンドラが存在していない）
E_PAR	パラメータエラー（pk_rcyc で指定されたアドレスが不正）
E_DACV	ドメインアクセス保護違反（対象周期ハンドラは他のドメインに所属してアクセス保護されている）
E_MACV	メモリアクセス違反エラー（pk_rcyc に対するアクセス権限がない。または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

cycid で指定された周期ハンドラの各種の状態を参照する。

dmnno は、属しているドメインの番号を示す。

cycpri は、周期ハンドラの優先度を示す。生成時に所属するドメインの最高優先度に設定されている。

lfttim は、次に周期ハンドラが起動される予定の時刻までの残り時間(ミリ秒)を示す。周期ハンドラが現在動作中か停止中かは関係しない。

cycstat は、周期ハンドラの状態を示す。
周期ハンドラの状態は以下のように表される。

cycstat:= (TCYC_STP | TCYC_STA | TCYC_EXE)

```
#define TCYC_STP    (0x00000000U)        /* 周期ハンドラは停止状態 */
#define TCYC_STA    (0x00000001U)        /* 周期ハンドラは動作状態 */
#define TCYC_EXE    (0x00000002U)        /* 周期ハンドラは実行状態 */
```

8.2.7.9 ts_cre_alm/tn_cre_alm – アラームハンドラの生成

C 言語インタフェース

【安全 API】 ID almid = ts_cre_alm(CONST T_CALM *pk_calm);

【通常 API】 ID almid = tn_cre_alm(CONST T_CALM *pk_calm);

パラメータ

CONST T_CALM*	pk_calm	アラームハンドラ定義情報へのポインタ

アラームハンドラ定義情報

```
typedef struct st_calm {
        ATR     almatr;         /* アラームハンドラ属性 */
        INT     stacd;          /* アラームハンドラ起動コード */
        FP      almhdr;         /* アラームハンドラアドレス */
        UINT    ustkno;         /* ユーザスタックの資源番号 */
        INT     ustksz;         /* ユーザスタックサイズ（バイト数）*/
        UINT    sstkno;         /* システムスタックの資源番号 */
        INT     sstksz;         /* システムスタックサイズ（バイト数）*/
        RELTIM  maxrtim;        /* 最大連続実行時間 */
        UB      oname[8];       /* オブジェクト名称 */
} T_CALM;
```

リターンパラメータ

ID	almid	アラームハンドラ ID
	または	エラーコード

エラーコード

E_NOMEM	メモリ不足（指定したサイズのスタックがスタック用領域から確保できない）
E_LIMIT	アラームハンドラの数がシステムの制限を超えた
E_RSATR	予約属性エラー（almatr が不正あるいは利用できない）
E_PAR	パラメータエラー（pk_calm, almhdr で指定されたアドレスが不正。または maxrtim が自ドメインの連続実行時間上限より大きい）
E_ONAME	オブジェクト名エラー（指定されたオブジェクト名が既にドメイン内で使用されている）
E_CTX	コンテキストエラー（タスク部および準タスク部以外で実行）
E_MACV	メモリアクセス違反エラー（pk_calm、almhdr に対するアクセス権限がない。または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

アラームハンドラを生成してアラームハンドラ ID を割り当てる。

almatr には、対象アラームハンドラの動作を指定するためのアラームハンドラ属性を設定する。
almatr には、次のような指定を行う。

almatr := [TA_ONAME]

TA_ONAME　　オブジェクト名称を指定する

#define TA_ONAME　(0x00000040U)　　/* オブジェクト名称を指定 */

アラームハンドラは C 言語の関数として以下の形式で記述する。

void　　almhdr(INT stacd);

上記の関数へのポインタ almhdr を、アラームハンドラアドレス almhdr に指定する。
アラームハンドラの起動パラメータとして、アラームハンドラ起動コード stacd が引数に渡される。

アラームハンドラはユーザスタックとシステムスタックを持つ。ただし、システムドメインのアラームハンドラは、システムスタックのみであり、ユーザスタックはもたない。よって、システムドメインのアラームハンドラ生成では、ユーザスタックの指定は無視される。
ユーザスタックは、アラームハンドラが所属するドメインのスタック用領域からメモリが確保される。ustkno にコンフィギュレーションで設定したスタック資源番号を指定する。ustksz にユーザスタックのサイズを指定する。ただし、通常ドメインではメモリ領域は動的に確保するので、通常 API では utskno の値は無視される。安全 API では ustksz は、ustkno で指定したスタック資源のメモリサイズ以下でなければならない。
システムスタックは、システムドメインのスタック用領域からメモリが確保される。sstkno にコンフィギュレーションで設定したスタック資源番号を指定する。sstksz にシステムスタックのサイズを指定する。sstksz は、sstkno で指定したスタック資源のメモリサイズ以下でなければならない。
以下の理由でスタックが確保できなかった場合、エラーE_NOMEM となる。

- 指定したスタックサイズが、スタック資源のサイズより大きい
- 指定したスタック資源が既にタスクまたはタイムイベントハンドラで使用されている
- 通常 API においてユーザスタックが確保できなかった（ユーザスタック領域の残りサイズが不足）

maxrtim には、アラームハンドラが連続して実行できる最大時間をミリ秒単位で設定する。maxrtim の最小値は 1 ミリ秒である。
maxrtim の最大値は自ドメインの連続実行時間上限の値である。
maxrtim が自ドメインの連続実行時間上限より大きな値を指定した場合はエラーE_PAR となる。
アラームハンドラが maxrtim を超えて連続実行した場合は異常例外が発生し、アラームハンドラの実行は中断される。

TA_ONAME を指定した場合に oname が有効となる。

oname には、対象アラームハンドラのオブジェクト名称を設定する。

アラームハンドラは、所属するドメインの最高実行優先度で動作し、同一優先度のタスクよりも優先される。つまり、同一優先度のタスク実行中にアラームハンドラの起動時刻となった場合は、タスクの動作は中断されアラームハンドラが起動する。また、アラームハンドラが自ドメインの他のタイムイベントハンドラやタスクに実行を中断されることはない。

【補足事項】

アラームハンドラは自ドメインの中では最優先で実行され、また実行を奪われることもない。ただし、他のドメインのより優先度の高いタスクやタイムイベントハンドラの方が優先される。通常ドメインは安全ドメインより実行優先度が低いので、通常ドメインのアラームハンドラは、安全ドメインのタスクやタイムイベントハンドラより低い優先度となる。これは通常ドメインのソフトにより安全ドメインのソフトの実行が阻害されることを防ぐためである。

8.2.7.10 ts_del_alm/tn_del_alm – アラームハンドラの削除

C 言語インタフェース

【安全 API】 ER ercd = ts_del_alm(ID almid);

【通常 API】 ER ercd = tn_del_alm(ID almid);

パラメータ

ID	almid	アラームハンドラ ID

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（almid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（almid のアラームハンドラが存在しない）
E_DACV	ドメインアクセス保護違反（対象アラームハンドラは他のドメインに所属してアクセス保護されている）
E_CTX	コンテキストエラー（タスク部および準タスク部以外で実行）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

almid で指定されたアラームハンドラを削除する。

アラームハンドラが動作状態でも削除することはできる。

8.2.7.11 ts_sta_alm/tn_sta_alm – アラームハンドラの動作開始

C 言語インタフェース

【安全 API】 ER ercd = ts_sta_alm(ID almid, RELTIM almtim);

【通常 API】 ER ercd = tn_sta_alm(ID almid, RELTIM almtim);

パラメータ

ID	almid	アラームハンドラ ID
RELTIM	almtim	アラームハンドラ起動時刻（ミリ秒）

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（almid が不正あるいは利用できない）
E_PAR	パラメータエラー（almtim が不正な値）
E_NOEXS	オブジェクトが存在していない（almid のアラームハンドラが存在しない）
E_DACV	ドメインアクセス保護違反（対象アラームハンドラは他のドメインに所属してアクセス保護されている）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

almid で指定されたアラームハンドラの起動時刻を設定して、動作状態にする。

almtim には、アラームハンドラを起動する時刻を指定する。

almtim に指定可能な値は、0～TS_MAX_ALMTIM である。TS_MAX_ALMTIM は実装定義である。

アラームハンドラは、本 API が呼び出された時刻を基準とし、almtim で指定した時間が経過した後に起動される。

almtim に 0 を指定した場合、本 API の実行直後にアラームハンドラが起動される。

既に本 API が発行されて、起動待ち状態になっているアラームハンドラに対して、再度本 API を発行した場合は、既に設定されていた起動時刻を一旦解除した後、再度アラームハンドラを設定する。

8.2.7.12 ts_stp_alm/tn_stp_alm – アラームハンドラの動作停止

C 言語インタフェース

【安全 API】 ER ercd = ts_stp_alm(ID almid);

【通常 API】 ER ercd = tn_stp_alm(ID almid);

パラメータ

ID	almid	アラームハンドラ ID

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（almid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（almid のアラームハンドラが存在しない）
E_DACV	ドメインアクセス保護違反（対象アラームハンドラは他のドメインに所属してアクセス保護されている）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

almid で指定されたアラームハンドラの起動時刻を解除し、停止状態にする。

すでに停止状態であれば何もしない。

8.2.7.13 ts_ref_alm/tn_ref_alm – アラームハンドラ状態参照

C 言語インタフェース

【安全 API】 ER ercd = ts_ref_alm(ID almid, T_RALM *pk_ralm);

【通常 API】 ER ercd = tn_ref_alm(ID almid, T_RALM *pk_ralm);

パラメータ

ID	almid	アラームハンドラ ID
T_RALM*	pk_ralm	アラームハンドラ状態を返す領域へのポインタ

リターンパラメータ

ER	ercd	エラーコード
T_RALM	ralm	アラームハンドラ状態

アラームハンドラ状態

```
typedef struct st_ralm {
        ID      dmnno;    /* ドメイン番号 */
        PRI     almpri;   /* アラームハンドラの優先度 */
        RELTIM  lfttim;   /* ハンドラ起動までの残り時間（ミリ秒）*/
        UINT    almstat;  /* アラームハンドラの状態 */
} T_RALM;
```

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（almid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（almid のアラームハンドラが存在しない）
E_PAR	パラメータエラー（pk_ralm で指定されたアドレスが不正）
E_DACV	ドメインアクセス保護違反（対象アラームハンドラは他のドメインに所属してアクセス保護されている）
E_MACV	メモリアクセス違反エラー（pk_ralm に対するアクセス権限がない。または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

almid で示されたアラームハンドラの各種状態を参照する。

dmnno は、属しているドメインの番号を示す。

almpri は、アラームハンドラの優先度を示す。

lfttim は、アラームハンドラが起動される予定の時刻までの残り時間(ミリ秒)を示す。アラームハンドラが停止状態のときは、lfttim は不定である。

almstat は、アラームハンドラの状態を示す。
ハンドラの状態は以下のように表される。

almstat:= (TALM_STP | TALM_STA | TALM_EXE)

```
#define TALM_STP    (0x00000000U)       /* アラームハンドラが停止状態 */
#define TALM_STA    (0x00000001U)       /* アラームハンドラが動作状態 */
#define TALM_EXE    (0x00000002U)       /* アラームハンドラが実行状態 */
```

8.2.8　システム構成情報管理機能の API

システム構成情報管理機能の詳細は、「3.5 システム構成情報管理機能」を参照。

8.2.8.1 ts_get_cfn/tn_get_cfn – システム構成情報(数値列)の取得

C 言語インタフェース

【安全 API】 INT ct = ts_get_cfn(CONST UB *name, INT *val, INT max);

【通常 API】 INT ct = tn_get_cfn(CONST UB *name, INT *val, INT max);

パラメータ

CONST UB*	name	システム構成情報の名称
INT*	val	システム構成情報の数値列を格納する配列
INT	max	val の配列の要素数

リターンパラメータ

INT	ct	定義されている数値情報の個数 または　エラーコード

エラーコード

E_NOEXS	指定したシステム構成情報が定義されていない
E_OBJ	指定したシステム構成情報が数値列でない、または範囲外
E_PAR	パラメータエラー（name, val で指定されたアドレスが不正、または max が負の値）
E_MACV	メモリアクセス違反エラー（val に対するアクセス権限がない。または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

システム構成情報から数値列の情報を取得する。

name の名称で定義されているシステム構成情報の数値列の値を、最大 max 個まで取得し、val へ格納する。

戻り値として、定義されている数値列の情報の数を返す。

戻り値が max より大きな値であれば、すべての情報を格納しきれていないことを示す。

この時、数値の取得は max 個まで実施し、max+1 以降の数値の取得は行わない。つまり、max+1 以降の数値にフォーマットや数値の大きさの誤りがあってもエラーは発生しない。

max=0 を指定することで、val に格納せずに数値列の数を取得することができる。

指定された名称のシステム構成情報が定義されていないときは、エラーE_NOEXS を返す。

指定された名称のシステム構成情報が、数値列でない場合、または INT 型に格納できない(オーバーフローする)場合は、エラーE_OBJ となる。

エラーが返された場合は、val に格納された値は不定である。

8.2.8.2 ts_get_cfs/tn_get_cfs – システム構成情報(文字列)の取得

C 言語インタフェース

【安全 API】 INT rlen = ts_get_cfs(CONST UB *name, UB *buf, INT max);

【通常 API】 INT rlen = tn_get_cfs(CONST UB *name, UB *buf, INT max);

パラメータ

CONST UB*	name	名称
UB*	buf	文字列を格納する配列
INT	max	buf の最大長(バイト数)

リターンパラメータ

INT	rlen	定義されている文字列情報の長さ(バイト数)
	または	エラーコード

エラーコード

E_NOEXS	指定されたシステム構成情報が定義されていない
E_PAR	パラメータエラー (name, buf で指定されたアドレスが不正, または max が負の値)
E_MACV	メモリアクセス違反エラー (name に対するアクセス権限がない。または対象のメモリ領域が存在しない)

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

システム構成情報から文字列の情報を取得する。

name の名称で定義されているシステム構成情報の文字列を、最大 max 文字まで取得し、buf へ格納する。

格納した文字列が max 文字未満であれば、最後に'¥0'を格納する。

戻り値として、定義されている文字列情報の長さ('¥0'は含まない)を返す。

戻り値が max より大きな値であれば、すべての情報を格納しきれていないことを示す。

max=0 を指定することで、buf に格納せずに文字列の長さを知ることができる。

指定された名称のシステム構成情報が定義されていないときは、エラーE_NOEXS を返す。

指定された名称のシステム構成情報が数値列であった場合も文字列として処理する。数値変換は行わない。

8.2.9　ドメイン管理機能の API

ドメイン管理機能の詳細は、「3.6 ドメイン管理機能」を参照。

8.2.9.1 ts_sta_dmn – ドメインの実行開始

C 言語インタフェース

【安全 API】 ER ercd = ts_sta_dmn(ID dmnid);

【通常 API】 なし

パラメータ

ID	dmnid	ドメイン ID

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（dmnid が不正または利用できない）
E_OBJ	ドメインの状態が不正（対象ドメインが停止状態ではない）
E_DACV	ドメインアクセス保護違反（対象ドメインはシステムドメイン）
E_CTX	コンテキストエラー（タスク部および準タスク部以外で実行）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

dmnid で指定されたドメインを実行開始する。

具体的には、初期タスクを実行可能状態に遷移させる。対象ドメインが実行状態となると、そのドメインの初期タスクが実行を開始する。

本 API による実行開始要求はキューイングされない。すなわち、対象ドメインが停止状態でないのに本 API が発行された場合、エラーE_OBJ となる。

8.2.9.2 ts_ext_dmn/tn_ext_dmn – 自ドメインの実行終了

C 言語インタフェース

【安全 API】 void ts_ext_dmn(void);

【通常 API】 void tn_ext_dmn(void);

パラメータ

なし

リターンパラメータ

なし

エラーコード

なし（API を発行したコンテキストには戻らない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

自ドメインを正常終了させ、停止状態へと移行させる。

自ドメインが停止状態となるとき、以下の終了処理が行われる。

- そのドメインに属するすべてのタスクは終了し削除される。つまり、API を発行したタスクは ts_exd_tsk/tn_exd_tsk を実行したのと同等の処理を行う。その他のタスクは、ts_ter_tsk/tn_ter_tsk により強制終了したのち、ts_del_tsk/tn_del_tsk によりタスク削除したのと同等の処理を行う。
- そのドメインに属するすべてのカーネルオブジェクトは削除される。タイムイベントハンドラも含まれる。
- そのドメインのタスクからオープンされているデバイスは、すべてクローズされる。

ドメインの終了処理は、そのドメインの初期タスクのコンテキストにて実行される。ただし、本 API の処理中、対象ドメインはディスパッチ禁止状態となり、また対象ドメインに属するタイムイベントハンドラの実行は抑止される。

タスク部および準タスク部以外で本 API が発行された場合は、エラーを検出し、異常例外が発生する。

8.2.9.3 ts_ter_dmn – 他ドメインの強制終了

C 言語インタフェース

【安全 API】 ER ercd = ts_ter_dmn(ID dmnid);

【通常 API】 なし

パラメータ

ID	dmnid	ドメイン ID

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（dmnid が不正あるいは利用できない）
E_OBJ	ドメインの状態が不正（対象ドメインが停止状態、または自ドメイン）
E_DACV	ドメインアクセス保護違反（対象ドメインはシステムドメインである）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

dmnid で指定されたドメインを強制的に終了させる。

具体的には、dmnid で指定された対象ドメインを終了処理したのち停止状態に移行させる。

ドメインの終了処理は、ts_ext_dmn/tn_ext_dmn による自ドメインの終了と同等である。

終了処理は対象ドメインの初期タスクにて実行される。その際、対象ドメインはディスパッチ禁止状態となり、また対象ドメインに属するタイムイベントハンドラの実行は抑止される。

本 API は対象ドメインの終了処理の完了を待たない。つまり、本 API から返ったときに対象ドメインの終了処理が完了しているとは限らない。ただし、本 API 発行後に対象ドメインにて終了処理以外のプログラムは実行されないことは保証される。対象ドメインの状態は、ts_ref_dmn/tn_ref_dmn にて取得できる。

本 API では、自ドメインは指定できない。自ドメインを指定した場合には、エラーE_OBJ となる。

システムドメインは強制終了することはできない。システムドメインを指定した場合は、エラーE_DACV となる。

8.2.9.4 ts_sus_dmn – 他ドメインを強制停止状態へ移行

C 言語インタフェース

【安全 API】 ER ercd = ts_sus_dmn(ID dmnid);

【通常 API】 なし

パラメータ

ID	dmnid	ドメイン ID

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（dmnid が不正あるいは利用できない）
E_OBJ	ドメインの状態が不正（対象ドメインが自ドメインまたは強制停止禁止ドメイン）
E_DACV	ドメインアクセス保護違反（対象ドメインはシステムドメイン）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

dmnid で指定されたドメインを強制停止状態に移し、ドメインに所属するタスクおよびタイムイベントハンドラの実行を中断させる。

本 API で自ドメインを指定することはできない。自ドメインを指定した場合には、エラーE_OBJ となる。
システムドメインは強制停止することはできない。また、安全ドメインは強制停止可能か否か、生成時の設定にて静的に決められる。強制停止できないドメインを指定した場合はエラーE_OBJ となる。

強制停止状態のドメインに所属するタスクおよびタイムイベントハンドラは、TRON Safe Kernel のプログラム実行のスケジューリングから外され、実行されることはない。タスクおよびタイムイベントハンドラの状態は、強制停止状態になった時点の状態が維持される。
ドメインの強制停止状態は、ts_rsm_dmn の発行によって解除される。

すでに強制停止状態のドメインに対し本 API を発行した場合は、そのドメインの強制停止状態は継続され、正常終了 E_OK となる。強制停止の要求はネストされない。よって、本 API が同一ドメインに複数回発行されていても、一回の ts_rsm_dmn で強制停止は解除される。

8.2.9.5 ts_rsm_dmn – 強制停止状態のドメインを再開

C 言語インタフェース

【安全 API】 ER ercd = ts_rsm_dmn(ID dmnid);

【通常 API】 なし

パラメータ

ID	dmnid	ドメイン ID

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（dmnid が不正あるいは利用できない）
E_OBJ	ドメインの状態が不正（対象ドメインが強制停止状態ではない）
E_DACV	ドメインアクセス保護違反（対象ドメインはシステムドメイン）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

dmnid で指定されたドメインの強制停止状態を解除する。

すなわち、対象ドメインが以前に発行された ts_sus_dmn によって強制停止状態に入り、その実行が中断している場合に、その状態を解除し、実行を再開させる。

強制停止状態の間、そのドメインに所属するタスクおよびタイムイベントハンドラの状態は保持されている。よって、本 API より強制停止状態が解除されると、タスクおよびタイムイベントハンドラの実行は再開される。

8.2.9.6 ts_get_oid/tn_get_oid – カーネルオブジェクトの ID 取得

C 言語インタフェース

【安全 API】 ID oid = ts_get_oid(ID dmnid, UINT otype, CONST UB *oname);

【通常 API】 ID oid = tn_get_oid(ID dmnid, UINT otype, CONST UB *oname);

パラメータ

ID	dmnid	ドメイン ID
UINT	otype	カーネルオブジェクトのタイプ
CONST UB	*oname	オブジェクト名称

リターンパラメータ

ID	oid	指定したカーネルオブジェクトの ID またはエラーコード

エラーコード

E_ID	不正 ID 番号（dmnid が不正あるいは利用できない）
E_PAR	パラメータエラー（otype で指定されたカーネルオブジェクトのタイプが不正、oname が不正、oname が文字列長 0 の場合もしくは 9 文字以上の場合）
E_OBJ	ドメインの状態が不正（対象ドメインが停止状態）
E_NOEXS	oname で指定したオブジェクトが存在していない
E_DACV	ドメインアクセス保護違反（通常 API で他ドメインを指定した、安全 API で安全ドメインからシステムドメインを指定した）
E_MACV	メモリアクセス違反エラー（oname に対するアクセス権限がない、または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

dmnid で指定されたドメインに所属する、otype と oname で指定したオブジェクト名のカーネルオブジェクトの ID を取得する。

カーネルオブジェクトは、オブジェクトタイプ otype とオブジェクト名称 oname で指定する。
オブジェクトタイプは以下の通りである。

TN_TSK (0x01)	タスク
TN_SEM (0x02)	セマフォ
TN_FLG (0x03)	イベントフラグ
TN_MBF (0x05)	メッセージバッファ

TN_MTX (0x07)	ミューテックス
TN_CYC (0x0a)	周期ハンドラ
TN_ALM (0x0b)	アラームハンドラ

dmnid=DMN_SELF(=0)のとき、対象ドメインは自ドメインとなる。通常 API では自ドメインの指定のみが許される。

8.2.9.7 ts_inf_dmn/tn_inf_dmn – ドメイン統計情報参照

C 言語インタフェース

【安全 API】 ER ercd = ts_inf_dmn(ID dmnid, T_IDMN *pk_idmn, BOOL clr);

【通常 API】 ER ercd = tn_inf_dmn(ID dmnid, T_IDMN *pk_idmn, BOOL clr);

パラメータ

ID	dmnid	ドメイン ID
T_IDMN*	pk_idmn	ドメイン統計情報を返す領域へのポインタ
BOOL	clr	ドメイン統計情報のクリアの有無（安全 API のみ有効）

リターンパラメータ

ER	ercd	エラーコード
T_IDMN	*pk_idmn	タスク統計情報

タスク統計情報

```
typedef struct st_idmn {
        RELTIM  ctime;          /* 累積ドメイン実行時間（ミリ秒）*/
        RELTIM  stime;          /* 連続ドメイン実行時間（ミリ秒）*/
} T_IDMN;
```

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（dmnid が不正あるいは利用できない）
E_PAR	パラメータエラー（pk_idmn で指定されたアドレスが不正）
E_DACV	ドメインアクセス保護違反（通常 API で他ドメインを指定した、安全 API で安全ドメインからシステムドメインを指定した）
E_MACV	メモリアクセス違反エラー（pk_idmn に対するアクセス権限がない、または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

dmnid で示された対象ドメインの統計情報を参照する。

dmnid=DMN_SELF(=0)によって自ドメインの指定を行うことができる。

通常 API は自ドメインの指定しかできない。他のドメインを指定した場合はエラーE_DACV となる。

安全 API で安全ドメインからシステムドメインを指定した場合はエラーE_DACV となる。

ctime は、累積ドメイン実行時間を示す。累積ドメイン実行時間とは、ドメインに所属するタスクおよびタイムイベントハンドラが実行されていた時間を累積した値(単位ミリ秒)である。時間の計測は ts_sta_dmn によりドメインが実行を開始するとともに開始される。加算されるのは対象ドメインのタスクおよびタイムイベントハンドラが実行している時間のみであり、割込みハンドラなどの他の実行単位のプログラムの実行時間は加算されない。なお、対象タスクから呼び出された拡張 SVC の実行時間は加算される。
ctime がリセット(0 クリア)されるのは以下の場合である。

- ts_sta_dmn でドメインが実行開始した
- ts_inf_dmn/tn_inf_dmn が clr=TRUE 指定で実行された(値を取得後にリセット)

stime は、連続ドメイン実行時間を示す。
連続ドメイン実行時間とは、対象ドメインが連続実行している時間の値である(単位ミリ秒)。すなわち、対象ドメインのタスクまたはタイムイベントハンドラが実行開始してから、他のドメインのタスクやタイムイベントハンドラに切り替わるまでの時間である。自ドメインのタスクおよびタイムイベントハンドラが切り替わっても、連続実行には影響しない。対象タスクから呼び出された拡張 SVC の実行時間は加算される。割込みハンドラなどタスク独立部の実行時間は除外される。
stime がリセット(0 クリア)されるのは以下の場合である。

- 対象ドメインのタスクまたはタイムイベントハンドラが、他のドメインのタスクやタイムイベントハンドラへ切り替わった

安全 API では、clr=TRUE の場合は、統計情報を取り出した後、累積ドメイン実行時間をリセット(0 クリア)する。
通常 API では、clr の指定は無視される。

8.2.9.8 ts_ref_dmn/tn_ref_dmn – ドメイン状態参照

C 言語インタフェース

【安全 API】 ER ercd = ts_ref_dmn(ID dmnid, T_RDMN *pk_rdmn);

【通常 API】 ER ercd = tn_ref_dmn(ID dmnid, T_RDMN *pk_rdmn);

パラメータ

ID	dmnid	ドメイン ID
T_RDMN*	pk_rdmn	ドメイン状態を返す領域へのポインタ

リターンパラメータ

ER	ercd	エラーコード
T_RDMN	rdmn	ドメイン状態

ドメイン状態

```
typedef    struct st_rdmn {
           ID      dmnno;          /* ドメイン番号 */
           UINT    dmnstat;        /* ドメインの状態 */
} T_RDMN;
```

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（dmnid が不正あるいは利用できない）
E_PAR	パラメータエラー（pk_rdmn で指定されたアドレスが不正）
E_DACV	ドメインアクセス保護違反（通常 API で他ドメインを指定した、安全 API で安全ドメインからシステムドメインを指定した）
E_MACV	メモリアクセス違反エラー（pk_rdmn に対するアクセス権限がない、または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

dmnid で示された対象ドメインの各種の状態を参照する。

dmnid=DMN_SELF(=0)によって自ドメインの指定を行うことができる。

通常 API は自ドメインの指定しかできない。他のドメインを指定した場合はエラーE_DACV となる。

安全 API で安全ドメインからシステムドメインを指定した場合はエラーE_DACV となる。

dmnno は、自身のドメインの番号を示す。

dmnstat は、ドメインの状態を示す。

ドメインの状態としては次のような値をとる。

TDS_RUN	0x00000001	ドメイン実行状態
TDS_WAI	0x00000008	ドメイン終了処理実行状態
TDS_DMT	0x00000010	ドメイン停止状態
TDS_DISDSP	0x00000080	ディスパッチ禁止状態
TDS_SUS	0x00000100	ドメイン強制停止状態

ディスパッチ禁止(TDS_DISDSP)は実行状態(TDS_RUN)とのみ一緒にセットされる。ドメイン実行状態でかつ、

ts_dis_dsp/tn_dis_dsp によりディスパッチ禁止状態であれば、TDS_DISDSP がセットされる。

8.3 TRON Safe Kernel/SM の API

8.3.1 サブシステム管理機能の API

サブシステム管理機能の詳細については、「4.3 サブシステム管理機能」を参照。

8.3.1.1 ts_cal_svc/tn_cal_svc – 拡張 SVC の呼び出し

C 言語インタフェース

【安全 API】 ER ercd = ts_cal_svc(ID ssid, FN fncd, T_SVCPARA *pk_svcpara);

【通常 API】 ER ercd = tn_cal_svc(ID ssid, FN fncd, T_SVCPARA *pk_svcpara);

パラメータ

ID	ssid	サブシステム ID
FN	fncd	機能コード
T_SVCPARA	*pk_svcpara	拡張 SVC に渡されるパラメータへのポインタ

拡張 SVC に渡されるパラメータ

```
typedef struct st_svcpara {
        UW      par[TS_MAX_SVCPARA];        拡張 SVC の各パラメータ
} T_SVCPARA;
```

リターンパラメータ

ER	ercd	エラーコード
		または、各 SVC の戻り値

エラーコード

E_OK	正常終了
E_RSFN	予約機能コード番号エラー（ssid, fncd が不正）
E_PAR	パラメータエラー（pk_svcpara で指定されたアドレスが不正）
E_DISWAI	拡張 SVC 呼び出し禁止（ts_dis_wai/tn_dis_wai で TTX_SVC が指定されている）
E_DACV	ドメインアクセス保護違反（API 呼び出し元のドメインでは使用できない）
E_MACV	メモリアクセス違反エラー（pk_svcpara に対するアクセス権限がない。または対象のメモリ領域が存在しない）
その他	各拡張 SVC から返されたエラーコード

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

ssid で指定したサブシステムの拡張 SVC ハンドラを呼び出す。

呼び出した拡張 SVC ハンドラには、fncd および pk_svcpara で示した拡張 SVC のパラメータが渡される。TS_MAX_SVCPARA は、4 以上の実装定義の数である。T_SVCPARA の各メンバーの内容は、サブシステム毎に決められる。

指定した ssid または fncd が不正であった場合は、予約機能コード番号エラーE_RSFN となる。

pk_svcpara で指定されたアドレスが不正の場合はパラメータエラーE_PAR となる。

指定した拡張 SVC の属性により、本 API を呼び出した実行プログラムの所属するドメインから、対象の拡張 SVC が使用できない場合は、ドメインアクセス保護違反エラーE_DACV となる。

上記以外の場合は、拡張 SVC ハンドラが呼び出され、その戻り値が本 API の戻り値となる。

【補足事項】

一般的に、各サブシステムはそれぞれの拡張 SVC を呼び出すための C 言語関数またはマクロを用意する。その C 言語関数またはマクロの中から本 API が呼ばれる。よって、拡張 SVC を使用するプログラムは本 API を直接呼び出すことはない。ただし、直接呼び出してもかまわない。

ssid や fncd が不正の場合に、パラメータエラーE_PAR や ID 番号エラーE_ID ではなく、未サポートエラーE_RSFN を返すのは、指定した拡張 SVC がそのシステムソフトウェアに存在していないことを示すためである。また、前述の C 言語関数またはマクロを用いた場合、直接 ssid や fncd を指定しないので、エラーの原因が不明確になることを避けるためもある。

8.3.2 デバイス管理機能の API

デバイス管理機能の詳細については、「4.1 デバイス管理機能」を参照。

8.3.2.1 ts_opn_dev/tn_opn_dev – デバイスのオープン

C 言語インタフェース

【安全 API】 ID dd = ts_opn_dev(CONST UB *devnm, UINT omode);

【通常 API】 ID dd = tn_opn_dev(CONST UB *devnm, UINT omode);

パラメータ

CONST UB*	devnm	デバイス名
UINT	omode	オープンモード

リターンパラメータ

ID	dd	デバイスディスクリプタ
	または	エラーコード

エラーコード

E_PAR	パラメータエラー（devnm で指定されたアドレスが不正、omode が不正な値）
E_BUSY	デバイスは使用中（排他オープン中）
E_NOEXS	デバイスは存在しない
E_LIMIT	オープン可能な最大数を超えた
E_CTX	コンテキストエラー（タスク部および準タスク部以外、ディスパッチ禁止状態、割込み禁止状態で実行）
E_DACV	ドメインアクセス保護違反（対象デバイスはこのドメインからオープンできない）
E_MACV	メモリアクセス違反エラー（devnm に対するアクセス権限がない。または対象のメモリ領域が存在しない）
E_OBJ	オブジェクトの状態が不正（セマフォの生成でエラーが発生した）
その他	デバイスドライバから返されたエラーコード

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

devnm で指定したデバイスを omode で指定したモードでオープンし、デバイスへのアクセスを準備する。

戻り値として、デバイスディスクリプタを返す。

omode には、デバイスのオープンモードを設定する。

omode には、次のような指定を行う。

omode := (TD_READ || TD_WRITE || TD_UPDATE) | [TD_EXCL || TD_WEXCL || TD_REXCL] | [TD_NOLOCK]

TD_READ	読込み専用

TD_WRITE	書込み専用
TD_UPDATE	読込みおよび書込みが可能
TD_EXCL	排他(一切の同時オープンを禁止する)
TD_WEXC	排他書込み(TD_WRITE または TD_UPDATE による同時オープンを禁止)
TD_REXCL	排他読込み(TD_READ または TD_UPDATE による同時オープンを禁止)
TD_NOLOCK	ロック(常駐化)不要

オープンモードは、ヘッダファイルにおいて以下のように定義されている。

```
#define TD_READ     (0x00000001U)     /* 読込み専用 */
#define TD_WRITE    (0x00000002U)     /* 書込み専用 */
#define TD_UPDATE   (0x00000003U)     /* 読込みおよび書込み */
#define TD_EXCL     (0x00000100U)     /* 排他 */
#define TD_WEXCL    (0x00000200U)     /* 排他書込み */
#define TD_REXCL    (0x00000400U)     /* 排他読込み */
#define TD_NOLOCK   (0x00001000U)     /* ロック(常駐化)不要 */
```

既にオープンされているデバイスをさらにオープンしようとした際の動作は、それぞれのオープンモードの組合せにより決まる。「表 8-4 デバイスドライバオープンモード」に、オープンモードの組合せを示す。

表 8-4 デバイスドライバオープンモード

現在オープンモード		同時オープンモード											
		排他指定なし			TD_WEXCL			TD_REXCL			TD_EXCL		
		R	U	W	R	U	W	R	U	W	R	U	W
排他指定なし	R	○	○	○	○	○	○	×	×	×	×	×	×
	U	○	○	○	×	×	×	×	×	×	×	×	×
	W	○	○	○	×	×	×	○	○	○	×	×	×
TD_WEXCL	R	○	×	×	○	×	×	×	×	×	×	×	×
	U	○	×	×	×	×	×	×	×	×	×	×	×
	W	○	×	×	×	×	×	○	×	×	×	×	×
TD_REXCL	R	×	×	○	×	×	○	×	×	×	×	×	×
	U	×	×	○	×	×	×	×	×	×	×	×	×
	W	×	×	○	×	×	×	×	×	○	×	×	×
TD_EXCL	R	×	×	×	×	×	×	×	×	×	×	×	×
	U	×	×	×	×	×	×	×	×	×	×	×	×
	W	×	×	×	×	×	×	×	×	×	×	×	×

R = TD_READ

W = TD_WRITE

U = TD_UPDATE

○ = オープン可

× = オープン不可 (E_BUSY)

デバイスは、安全ソフトウェアまたは通常ソフトウェアのどちらからしかオープンできない。どちらからオープンできるかは、デバイス毎に静的に決められている。
安全ソフトウェアからのみオープンできるデバイスは、安全ドメインまたはシステムドメインのソフトウェアから ts_opn_dev によりオープンできる。通常ソフトウェアからのみオープンできるデバイスは、通常ドメインのソフトウェアから tn_opn_dev によりオープンできる。

本 API が正常に実行されるとデバイスディスクリプタが返される。以降、本デバイスに対するリード、ライトなどの操作はデバイスディスクリプタを用いて行う。デバイスディスクリプタは、オープンしたタスクのドメインに所属する。他のドメインからディスクリプタは使用できない。

物理デバイスをオープンした場合、その物理デバイスに属する論理デバイスをすべて同じモードでオープンしたのと同様に扱い、排他オープンの処理が行われる。

安全 API は安全ソフトウェアからオープン可能なデバイスのみをオープンできる。通常 API は通常ソフトウェアからオープン可能なデバイスのみをオープンできる。それ以外をオープンしようとした場合は、エラーE_DACV を返す。
本 API では、セマフォを1つ生成する。生成したセマフォは、クローズ処理における要求の完了待ちに使用する。セマフォ生成時にエラーが発生した場合はエラーE_OBJ となる。生成したセマフォは ts_cls_dev/tn_cls_dev で削除される。

8.3.2.2 ts_cls_dev/tn_cls_dev – デバイスのクローズ

C 言語インタフェース

【安全 API】 ER ercd = ts_cls_dev(ID dd, UINT option, TMO tmout);

【通常 API】 ER ercd = tn_cls_dev(ID dd, UINT option, TMO tmout);

パラメータ

ID	dd	デバイスディスクリプタ
UINT	option	クローズオプション
TMO	tmout	タイムアウト時間（ミリ秒）

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_PAR	パラメータエラー（tmout の値が不正）
E_ID	dd が不正またはオープンされていない
E_TMOUT	タイムアウトエラー（指定時間内にクローズ処理が完了しなかった）
E_CTX	コンテキストエラー（タスク部および準タスク部以外、ディスパッチ禁止状態、割込み禁止状態で実行）
E_DACV	ドメインアクセス保護違反（dd は他のドメインでオープン）
E_DOMAIN	ドメインエラー
その他	デバイスドライバから返されたエラーコード

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

dd で指定したデバイスディスクリプタのデバイスをクローズする。

処理中の要求があった場合は、その処理を中止させてクローズする。

option には、クローズオプション(デバイスをクローズする際の動作)を設定する。

option は対象デバイスを制御するデバイスドライバに伝えられる。option の意味はデバイスドライバの仕様で規定される。

dd は、自ドメインのソフトウェアによりオープンされたデバイスのデバイスディスクリプタでなければならない。他のドメインでオープンされたデバイスディスクリプタの場合はエラーE_DACV となる。

tmout にタイムアウト時間を指定する。

タイムアウト時間を超えた場合は、エラーE_TMOUT を返すとともに、異常例外を発生する。

安全 API では tmout として TMO_FEVR または TMO_POL を指定した場合は、ドメインエラーE_DOMAIN となる。

tmout として、-2 以下の値を指定した場合、パラメータエラーE_PAR となる。さらに安全ドメインの場合、tmout として、最大タイムアウト時間を超えて指定した場合、パラメータエラーE_PAR となる。

【補足事項】

デバイスは、本 API 以外に以下の場合にクローズされる。

- ドメインの実行終了（ts_ext_dmn/tn_ext_dmn、ts_ter_dmn）により、対象ドメインに属するデバイスディスクリプタはすべてクローズされる。

8.3.2.3 ts_rea_dev/tn_rea_dev – デバイスの読込み要求

C 言語インタフェース

【安全 API】 ID reqid = ts_rea_dev(ID dd, W start, void *buf, W size, TMO tmout);

【通常 API】 ID reqid = tn_rea_dev(ID dd, W start, void *buf, W size, TMO tmout);

パラメータ

ID	dd	デバイスディスクリプタ
W	start	読込み開始位置（≧0:固有データ，＜0:属性データ）
void*	buf	読み込んだデータを格納するバッファ
W	size	読み込むサイズ
TMO	tmout	タイムアウト時間（ミリ秒）

リターンパラメータ

ID	reqid	リクエスト ID
	または	エラーコード

エラーコード

E_ID	dd が不正またはオープンされていない
E_PAR	パラメータエラー（buf で指定されたアドレスが不正、size の値が不正、tmout の値が不正）
E_OACV	オープンモードが不正（読込みが許可されていない）
E_LIMIT	最大リクエスト数を超えた
E_TMOUT	タイムアウトエラー（他の要求を処理中で受け付けられない）
E_ABORT	処理中止
E_CTX	コンテキストエラー（タスク部および準タスク部以外、ディスパッチ禁止状態、割込み禁止状態で実行）
E_DACV	ドメインアクセス保護違反（dd は他のドメインでオープン）
E_MACV	メモリアクセス違反エラー（buf に対するアクセス権限がない、または対象のメモリ領域が存在しない）
E_DOMAIN	ドメインエラー
その他	デバイスドライバから返されたエラーコード

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

dd で指定されたデバイスからの固有データまたは属性データの読込みを開始する。

具体的には対象とするデバイスを制御するデバイスドライバに読込みの要求を通知する。正常に要求が通知されれば、リクエスト ID を返して終了する。

本 API は読込みを開始するのみで、読込み完了を待たずに呼び出し元へ戻る。読込み完了は ts_wai_dev/tn_wai_dev により確

認することができる。読込みが完了するまで、buf を保持しなければならない。

start が 0 以上の場合は、デバイスの固有データを読み込む。固有データの start および size の単位はデバイス毎に決められる。
start が負の値の場合は、デバイスの属性データを読み込む。start は属性データ番号、size はバイト数となり、start のデータ番号の属性データを読み込む。size は読み込む属性データのサイズ以上でなければならない。複数の属性データを一度に読み込むことはできない。size が読み込む属性データのサイズ未満の場合はエラーE_PAR となる。
size=0 を指定した場合、実際の読込みは行わず、現時点で読込み可能なサイズを調べる。

読込みまたは書込みの動作中である場合、新たな要求を受け付けられるか否かはデバイスドライバによる。新たな要求を受け付けられない状態の場合、本 API の処理の中で要求受付待ちとなる。要求受付待ちのタイムアウト時間を tmout に指定する。要求受付待ちが指定されたタイムアウト時間を超えた場合、エラーE_TMOUT を返す。この場合、デバイスの読込み要求は破棄される。
安全 API では tmout として TMO_FEVR または TMO_POL を指定した場合は、ドメインエラーE_DOMAIN となる。
tmout として、−2 以下の値を指定した場合、パラメータエラーE_PAR となる。さらに安全ドメインの場合、tmout として、最大タイムアウト時間を超えて指定した場合、パラメータエラーE_PAR となる。

dd は、自ドメインのソフトウェアによりオープンされたデバイスのデバイスディスクリプタでなければならない。他のドメインでオープンされたデバイスディスクリプタの場合はエラーE_DACV となる。

【補足事項】
本 API の処理は、デバイスへの読込みの要求を、デバイスを制御するデバイスドライバに渡すことである。読込み要求をデバイスドライバに渡すと本 API は終了し呼び出し元に戻る。実際のデバイスからの読込み処理はデバイスドライバにおいて行われ、具体的な処理はそれぞれのデバイスドライバにて定められる。
本 API のタイムアウト時間は、デバイスドライバに読込み要求を渡すまでの時間であり、デバイスドライバの処理におけるタイムアウト時間は、それぞれのデバイスドライバにて定められる。

8.3.2.4 ts_wri_dev/tn_wri_dev – デバイスの書込み要求

C 言語インタフェース

【安全 API】 ID reqid = ts_wri_dev(ID dd, W start, CONST void *buf, W size, TMO tmout);

【通常 API】 ID reqid = tn_wri_dev(ID dd, W start, CONST void *buf, W size, TMO tmout);

パラメータ

ID	dd	デバイスディスクリプタ
W	start	書込み開始位置（≧0:固有データ，＜0:属性データ）
CONST void*	buf	書き込むデータを格納したバッファ
W	size	書き込むサイズ
TMO	tmout	要求受付待ちタイムアウト時間（ミリ秒）

リターンパラメータ

ID	reqid	リクエスト ID
	または	エラーコード

エラーコード

E_ID	dd が不正またはオープンされていない
E_PAR	パラメータエラー（buf で指定されたアドレスが不正、size の値が不正、tmout の値が不正）
E_OACV	オープンモードが不正（書込みが許可されていない）
E_RONLY	書き込めないデバイス
E_LIMIT	最大リクエスト数を超えた
E_TMOUT	タイムアウトエラー（他の要求を処理中で受け付けられない）
E_ABORT	処理中止
E_CTX	コンテキストエラー（タスク部および準タスク部以外、ディスパッチ禁止状態、割込み禁止状態で実行）
E_DACV	ドメインアクセス保護違反（dd は他のドメインでオープン）
E_MACV	メモリアクセス違反エラー（buf に対するアクセス権限がない、または対象のメモリ領域が存在しない）
E_DOMAIN	ドメインエラー
その他	デバイスドライバから返されたエラーコード

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

dd で指定されたデバイスへの固有データまたは属性データの書込みを開始する。

具体的には対象とするデバイスを制御するデバイスドライバに書込みの要求を通知する。正常に要求が通知されれば、リクエスト ID を返して終了する。

本 API は書込みを開始するのみで、書込み完了を待たずに呼び出し元へ戻る。書込み完了は ts_wai_dev/tn_wai_dev により確認することができる。書込みが完了するまで、buf を保持しなければならない。

start が 0 以上の場合は、デバイスの固有データを書き込む。固有データの場合、start および size の単位はデバイス毎に決められる。
start が負の値の場合は、デバイスの属性データを書き込む。start は属性データ番号、size はバイト数となり、start のデータ番号の属性データに書き込む。size は書き込む属性データのサイズと同じでなければならない。複数の属性データを一度に書き込むことはできない。
size=0 を指定した場合、実際の書込みは行わず、現時点で書込み可能なサイズを調べる。

読込みまたは書込みの動作中である場合、新たな要求を受け付けられるか否かはデバイスドライバによる。新たな要求を受け付けられない状態の場合、本 API の処理の中で要求受付待ちとなる。要求受付待ちのタイムアウト時間を tmout に指定する。要求受付待ちが指定されたタイムアウト時間を超えた場合、エラーE_TMOUT を返す。この場合、デバイスの書込み要求は破棄される。
安全 API では tmout として TMO_FEVR または TMO_POL を指定した場合は、ドメインエラーE_DOMAIN となる。
tmout として、-2 以下の値を指定した場合、パラメータエラーE_PAR となる。さらに安全ドメインの場合、tmout として、最大タイムアウト時間を超えて指定した場合、パラメータエラーE_PAR となる。

dd は、自ドメインのソフトウェアによりオープンされたデバイスのデバイスディスクリプタでなければならない。他のドメインでオープンされたデバイスディスクリプタの場合はエラーE_DACV となる。

【補足事項】
本 API の処理は、デバイスへの書込みの要求を、デバイスを制御するデバイスドライバに渡すことである。書込み要求をデバイスドライバに渡すと本 API は終了し呼び出し元に戻る。実際のデバイスへの書込み処理はデバイスドライバにおいて行われ、具体的な処理はそれぞれのデバイスドライバにて定められる。
本 API のタイムアウト時間は、デバイスドライバに書込み要求を渡すまでの時間であり、デバイスドライバの処理におけるタイムアウト時間は、それぞれのデバイスドライバにて定められる。

8.3.2.5 ts_wai_dev/tn_wai_dev – デバイスの要求完了待ち

C 言語インタフェース

【安全 API】 ID creqid = ts_wai_dev(ID dd, ID reqid, W *asize, ER *ioer, TMO tmout);

【通常 API】 ID creqid = tn_wai_dev(ID dd, ID reqid, W *asize, ER *ioer, TMO tmout);

パラメータ

ID	dd	デバイスディスクリプタ
ID	reqid	リクエスト ID
W*	asize	読込み/書込みサイズを返す領域へのポインタ
ER*	ioer	入出力エラーを返す領域へのポインタ
TMO	tmout	タイムアウト時間（ミリ秒）

リターンパラメータ

ID	creqid	完了したリクエスト ID
	または	エラーコード
W	asize	実際の読込み/書込みサイズ
ER	ioer	入出力エラー

エラーコード

E_ID	dd が不正またはオープンされていない、reqid が不正または dd に対する要求ではない
E_PAR	パラメータエラー（asize, ioer で指定されたアドレスが不正、tmout の値が不正）
E_OBJ	reqid の要求は他のタスクで完了待ちもしくはキャンセル待ちをしている
E_NOEXS	処理中の要求はない（reqid＝0 の場合のみ）
E_TMOUT	タイムアウトエラー（処理継続中）
E_ABORT	処理を中止した
E_CTX	コンテキストエラー（タスク部および準タスク部以外、ディスパッチ禁止状態、割込み禁止状態で実行）
E_DACV	ドメインアクセス保護違反（dd は他のドメインでオープン）
E_MACV	メモリアクセス違反エラー（asize、ioer に対するアクセス権限がない、または対象のメモリ領域が存在しない）
E_DOMAIN	ドメインエラー
その他	デバイスドライバから返されたエラーコード

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

dd で指定されたデバイスに対する reqid の要求の完了を待つ。

reqid は、対象の要求を示すリクエスト ID であり、ts_rea_dev/tn_rea_dev または ts_wri_dev/tn_wri_dev の戻り値で返される。

reqid=0 の場合は、dd で指定されたデバイスに対する要求の内のいずれかが完了するのを待つ。本 API を呼び出した時点で処理中の要求のみが完了待ちの対象となる。本 API の呼び出し後に要求された処理は完了待ちの対象とならない。
デバイスが複数の要求を受けている場合、その要求の完了の順序は必ずしも要求した順序ではなく、デバイスドライバに依存する。

reqid で指定した処理が終了すると、本 API は正常終了する。その際に、asize には要求により読込みまたは書込みされたデータのサイズが返される。また、ioer にはデバイスドライバからの処理結果（エラーコード）が返される。

tmout に完了待ちのタイムアウト時間を指定する。
タイムアウト時間を超えた場合は、エラーE_TMOUT を返す。ただし、タイムアウトした場合でも要求された処理は継続中である。処理の終了を確認するには、再度本 API により完了を待つ必要がある。reqid > 0 かつ tmout=TMO_FEVR(=-1)の場合はタイムアウトすることはなく、必ず処理が完了する。なお、安全 API では TMO_FEVR を指定した場合は、ドメインエラーE_DOMAIN となる。タイムアウトした場合の asize の内容はそれぞれのデバイスドライバの仕様で規定される。ioer の内容は無効である。tmout として、-2 以下の値を指定した場合、パラメータエラーE_PAR となる。さらに安全ドメインの場合、tmout として、最大タイムアウト時間を超えて指定した場合、パラメータエラーE_PAR となる。

本 API による要求の完了待ちが正しくできなかった場合にエラーを返す。本 API の戻り値にエラーが返された場合、asize および ioer の内容は無効である。また、戻り値にエラーが返された場合は要求された処理を継続中なので、処理の終了を確認するには、再度本 API により完了を待つ必要がある。

同じリクエスト ID に対して、複数のタスクから同時に完了待ちすることはできない。reqid=0 で待っているタスクがあれば、同じ dd に対して他のタスクは完了待ちできない。同様に、reqid > 0 で待っているタスクがあれば、他のタスクで reqid=0 の完了待ちはできない。

dd は、自ドメインのソフトウェアによりオープンされたデバイスのデバイスディスクリプタでなければならない。他のドメインでオープンされたデバイスディスクリプタの場合はエラーE_DACV となる。

8.3.2.6 ts_can_dev/tn_can_dev – デバイスの要求無効化

C 言語インタフェース

【安全 API】 ID creqid = ts_can_dev(ID dd, ID reqid, TMO tmout);

【通常 API】 ID creqid = tn_can_dev(ID dd, ID reqid, TMO tmout);

パラメータ

ID	dd	デバイスディスクリプタ
ID	reqid	リクエスト ID
TMO	tmout	タイムアウト時間（ミリ秒）

リターンパラメータ

ID	creqid	完了したリクエスト ID
	または	エラーコード

エラーコード

E_ID	dd が不正またはオープンされていない、reqid が不正または dd に対する要求ではない
E_PAR	パラメータエラー（tmout の値が不正）
E_TMOUT	タイムアウトエラー
E_CTX	コンテキストエラー（タスク部および準タスク部以外、ディスパッチ禁止状態、割込み禁止状態で実行）
E_DACV	ドメインアクセス保護違反（dd は他のドメインでオープン）
E_DOMAIN	ドメインエラー
その他	デバイスドライバから返されたエラーコード

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

dd で指定されたデバイスに対する reqid の要求をキャンセルする。

reqid は、対象の要求を示すリクエスト ID であり、ts_rea_dev/tn_rea_dev または ts_wri_dev/tn_wri_dev の戻り値で返される。

tmout にタイムアウト時間を指定する。

タイムアウト時間を超えた場合は、エラーE_TMOUT を返すとともに、異常例外を発生する。

安全 API では tmout として TMO_FEVR または TMO_POL を指定した場合は、ドメインエラーE_DOMAIN となる。

指定したリクエスト ID に対して、他のタスクが ts_wai_dev/tn_wai_dev で完了待ちしていた場合は、その API はエラーE_ABORT で終了する。

tmout として、-2 以下の値を指定した場合、パラメータエラーE_PAR となる。さらに安全ドメインの場合、tmout として、最大タイムアウト時間を超えて指定した場合、パラメータエラーE_PAR となる。

dd は、自ドメインのソフトウェアによりオープンされたデバイスのデバイスディスクリプタでなければならない。他のドメインでオープンされたデバイスディスクリプタの場合はエラーE_DACV となる。

【補足事項】

本 API はデバイスドライバに処理の中断を要求する。デバイスドライバは、この中断要求には速やかに対応しなくてはならないが、その時間はデバイス毎の仕様にて規定される。本 API のタイムアウト時間には、通常は対象のデバイスドライバの中断要求を受け付けるまでの時間以上の値を設定する。タイムアウトが発生したということは、デバイスドライバが正常に動作していないことを意味するので、異常例外を発生させている。

8.3.2.7 ts_srea_dev/tn_srea_dev – デバイスへの同期読込み

C 言語インタフェース

【安全 API】 ER ercd = ts_srea_dev(ID dd, W start, void *buf, W size, W *asize, TMO tmout);

【通常 API】 ER ercd = tn_srea_dev(ID dd, W start, void *buf, W size, W *asize, TMO tmout);

パラメータ

ID	dd	デバイスディスクリプタ
W	start	読込み開始位置（≧0:固有データ，＜0:属性データ）
void*	buf	読み込んだデータを格納するバッファ
W	size	読み込むサイズ
W*	asize	読込みサイズを返す領域へのポインタ
TMO	tmout	タイムアウト時間（ミリ秒）

リターンパラメータ

ER	ercd	エラーコード
W	asize	実際の読込みサイズ

エラーコード

E_ID	dd が不正またはオープンされていない
E_PAR	パラメータエラー（buf, asize で指定されたアドレスが不正、size の値が不正、tmout の値が不正）
E_TMOUT	タイムアウトエラー
E_OACV	オープンモードが不正（読込みが許可されていない）
E_LIMIT	最大リクエスト数を超えた
E_ABORT	処理が中止された
E_CTX	コンテキストエラー（タスク部および準タスク部以外、ディスパッチ禁止状態、割込み禁止状態で実行）
E_DACV	ドメインアクセス保護違反（dd は他のドメインでオープン）
E_MACV	メモリアクセス違反エラー（buf に対するアクセス権限がない、または対象のメモリ領域が存在しない）
E_DOMAIN	ドメインエラー
その他	デバイスドライバから返されたエラーコード

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

dd で指定されたデバイスから固有データまたは属性データを読込む。

具体的には対象とするデバイスを制御するデバイスドライバに読込みの要求を通知したのち、要求した処理の完了待ちとなる。デバイスドライバが要求した処理を完了すると本 API は終了する。

start および size の内容は、ts_rea_dev/tn_rea_dev と同じである。
buf には読み込んだデータが格納される。asize には読み込んだデータのサイズが返される。

デバイスドライバの読込み処理でエラーが発生した場合は、デバイスドライバからの処理結果（エラーコード）を返して本 API は終了する。その際の buf、asize の内容はデバイスドライバの仕様にて規定される。

tmout にはタイムアウト時間を指定する。
タイムアウト時間を超えた場合は、エラーE_TMOUT を返す。その際に、まだ読込み要求がデバイスドライバに受け付けられていない場合（要求受付待ち）、要求が破棄される。すでにデバイスドライバが要求を受け付けている場合は、デバイスドライバの処理を中断する。この処理の中断は、ts_can_dev/tn_can_dev と同等の処理が行われる。
安全 API では tmout として TMO_FEVR または TMO_POL を指定した場合は、ドメインエラーE_DOMAIN となる。
tmout として、-2 以下の値を指定した場合、パラメータエラーE_PAR となる。さらに安全ドメインの場合、tmout として、最大タイムアウト時間を超えて指定した場合、パラメータエラーE_PAR となる。

dd は、自ドメインのソフトウェアによりオープンされたデバイスのデバイスディスクリプタでなければならない。他のドメインでオープンされたデバイスディスクリプタの場合はエラーE_DACV となる。

8.3.2.8 ts_swri_dev/tn_swri_dev – デバイスの同期書込み

C 言語インタフェース

【安全 API】 ER ercd = ts_swri_dev(ID dd, W start, CONST void *buf, W size, W *asize, TMO tmout);

【通常 API】 ER ercd = tn_swri_dev(ID dd, W start, CONST void *buf, W size, W *asize, TMO tmout);

パラメータ

ID	dd	デバイスディスクリプタ
W	start	書込み開始位置（≧0:固有データ，＜0:属性データ）
CONST void*	buf	書き込むデータを格納したバッファ
W	size	書き込むサイズ
W*	asize	書込みサイズを返す領域へのポインタ
TMO	tmout	タイムアウト時間（ミリ秒）

リターンパラメータ

ER	ercd	エラーコード
W	asize	実際の書込みサイズ

エラーコード

E_ID	dd が不正またはオープンされていない
E_PAR	パラメータエラー（buf、asize で指定されたアドレスが不正、size の値が不正、tmout の値が不正）
E_TMOUT	タイムアウトエラー
E_OACV	オープンモードが不正（書込みが許可されていない）
E_RONLY	書き込めないデバイス
E_LIMIT	最大リクエスト数を超えた
E_ABORT	処理中止
E_CTX	コンテキストエラー（タスク部および準タスク部以外、ディスパッチ禁止状態、割込み禁止状態で実行）
E_DACV	ドメインアクセス保護違反（dd は他のドメインでオープン）
E_MACV	メモリアクセス違反エラー（buf に対するアクセス権限がない、または対象のメモリ領域が存在しない）
E_DOMAIN	ドメインエラー
その他	デバイスドライバから返されたエラーコード

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

dd で指定されたデバイスへ固有データまたは属性データを書込む。

具体的には対象とするデバイスを制御するデバイスドライバに書込みの要求を通知したのち、要求した処理の完了待ちとなる。デバイスドライバが要求した処理を完了すると本 API は終了する。

start および size の内容は、ts_wri_dev/tn_wri_dev と同じである。
asize には書き込んだデータのサイズが返される。

デバイスドライバの書込み処理でエラーが発生した場合は、デバイスドライバからの処理結果（エラーコード）を返して本 API は終了する。その際の buf、asize の内容はデバイスドライバの仕様にて規定される。

tmout にはタイムアウト時間を指定する。
タイムアウト時間を超えた場合は、エラーE_TMOUT を返す。その際に、まだ書込み要求がデバイスドライバに受け付けられていない場合（要求受付待ち）、要求が破棄される。すでにデバイスドライバが要求を受け付けている場合は、デバイスドライバの処理を中断する。この処理の中断は、ts_can_dev/tn_can_dev と同等の処理が行われる。
安全 API では tmout として TMO_FEVR または TMO_POL を指定した場合は、ドメインエラーE_DOMAIN となる。
tmout として、-2 以下の値を指定した場合、パラメータエラーE_PAR となる。さらに安全ドメインの場合、tmout として、最大タイムアウト時間を超えて指定した場合、パラメータエラーE_PAR となる。

dd は、自ドメインのソフトウェアによりオープンされたデバイスのデバイスディスクリプタでなければならない。他のドメインでオープンされたデバイスディスクリプタの場合はエラーE_DACV となる。

8.3.2.9 ts_evt_dev/tn_evt_dev – デバイス要求イベントの送信

C 言語インタフェース

【安全 API】 INT retcode = ts_evt_dev(ID devid, INT evttyp, void *evtinf, TMO tmout);

【通常 API】 INT retcode = tn_evt_dev(ID devid, INT evttyp, void *evtinf, TMO tmout);

パラメータ

ID	devid	イベント送信先のデバイス ID
INT	evttyp	ドライバ要求イベントのタイプ
void*	evtinf	イベントタイプ別の情報
TMO	tmout	タイムアウト時間（ミリ秒）

リターンパラメータ

INT	retcode	eventfn からの戻り値
	または	エラーコード

エラーコード

E_ID	devid が不正あるいは利用できない
E_NOEXS	devid のデバイスは存在しない
E_PAR	パラメータエラー（evtinf で指定されたアドレスが不正、evttyp の値が不正、tmout の値が不正）
E_TMOUT	タイムアウトエラー
E_CTX	コンテキストエラー（タスク部および準タスク部以外、ディスパッチ禁止状態、割込み禁止状態で実行）
E_DACV	ドメインアクセス保護違反（対象デバイスはこのドメインからアクセスできない）
E_MACV	メモリアクセス違反エラー（evtinf に対するアクセス権限がない、または対象のメモリ領域が存在しない）
E_DOMAIN	ドメインエラー
その他	デバイスドライバから返されたエラーコード

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	×

解説

devid で指定されるデバイス(デバイスドライバ)に、ドライバ要求イベントを送る。

本 API は、送ったドライバ要求イベントの応答を受けて終了する。

evttyp は、ドライバ要求イベントタイプである。

ドライバ要求イベントの機能(処理内容)および evtinf の内容はイベントタイプ毎に定義される。

ドライバ要求イベントの処理時間は、デバイスドライバ毎に定義される。

tmout にタイムアウト時間を指定する。
タイムアウト時間を超えた場合は、エラーE_TMOUT を返すとともに、異常例外を発生する。
安全 API では tmout として TMO_FEVR または TMO_POL を指定した場合は、ドメインエラーE_DOMAIN となる。
tmout として、−2 以下の値を指定した場合、パラメータエラーE_PAR となる。さらに安全ドメインの場合、tmout として、最大タイムアウト時間を超えて指定した場合、パラメータエラーE_PAR となる。

安全 API は安全ソフトウェアからオープン可能なデバイスのみを対象とできる。通常 API は通常ソフトウェアからオープン可能なデバイスのみを対象とできる。それ以外に対して操作しようとした場合は、エラーE_DACV を返す。

【補足事項】
本 API はデバイスドライバにドライバ要求イベントで指定した処理を要求する。デバイスドライバは、ドライバ要求イベントの処理は速やかに実行しなくてはならないが、その時間はデバイス毎の仕様にて規定される。本 API のタイムアウト時間には、通常は対象のデバイスドライバのドライバ要求イベントの処理時間以上の値を設定する。タイムアウトが発生したということは、デバイスドライバが正常に動作していないことを意味するので、異常例外を発生させている。

8.3.2.10 ts_get_dev/tn_get_dev – デバイスのデバイス名取得

C 言語インタフェース

【安全 API】 ID pdevid = ts_get_dev(ID devid, UB *devnm);

【通常 API】 ID pdevid = tn_get_dev(ID devid, UB *devnm);

パラメータ

ID	devid	デバイス ID
UB*	devnm	デバイス名の格納領域へのポインタ

リターンパラメータ

ID	pdevid	物理デバイスのデバイス ID
	または	エラーコード
UB	devnm	デバイス名

エラーコード

E_ID	devid が不正あるいは利用できない
E_NOEXS	devid のデバイスは存在しない
E_PAR	パラメータエラー（devnm で指定されたアドレスが不正）
E_CTX	コンテキストエラー（タスク部および準タスク部以外、ディスパッチ禁止状態、割込み禁止状態で実行）
E_DACV	ドメインアクセス保護違反（対象デバイスはこのドメインからアクセスできない）
E_MACV	メモリアクセス違反エラー（devnm に対するアクセス権限がない、または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	×

解説

devid で指定されたデバイスのデバイス名を取得し、devnm に格納する。

devid が物理デバイスのデバイス ID であれば、devnm には物理デバイス名が格納される。devid が論理デバイスのデバイス ID であれば、devnm には論理デバイス名が格納される。
devnm は L_DEVNM+1 バイト以上の領域が必要である。
戻り値には、devid のデバイスが属する物理デバイスのデバイス ID を返す。

安全 API は安全ソフトウェアからオープン可能なデバイスのみを対象とできる。通常 API は通常ソフトウェアからオープン可能なデバイスのみを対象とできる。それ以外に対して操作しようとした場合は、エラーE_DACV を返す。

8.3.2.11 ts_ref_dev/tn_ref_dev – デバイスのデバイス情報取得

C 言語インタフェース

【安全 API】 ID devid = ts_ref_dev(CONST UB *devnm, T_RDEV *pk_rdev);

【通常 API】 ID devid = tn_ref_dev(CONST UB *devnm, T_RDEV *pk_rdev);

パラメータ

CONST UB*	devnm	デバイス名
T_RDEV*	pk_rdev	デバイス情報を返す領域へのポインタ

リターンパラメータ

ID	devid	デバイス ID
	または	エラーコード
T_RDEV	pk_rdev	デバイス情報

デバイス情報

```
typedef struct st_rdev {
        ATR     devatr;         /* デバイス属性 */
        INT     blksz;          /* 固有データのブロックサイズ（-1:不明）*/
        INT     nsub;           /* サブユニット数 */
        INT     subno;          /* 0 : 物理デバイス　1～nsub : サブユニット番号+1 */
} T_RDEV;
```

エラーコード

E_NOEXS	devnm のデバイスは存在しない
E_PAR	パラメータエラー（devnm, pk_rdev で指定されたアドレスが不正）
E_CTX	コンテキストエラー（タスク部および準タスク部以外、ディスパッチ禁止状態、割込み禁止状態で実行）
E_DACV	ドメインアクセス保護違反（対象デバイスはこのドメインからアクセスできない）
E_MACV	メモリアクセス違反エラー（devnm に対するアクセス権限がない、または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	×

解説

devnm で指定するデバイスのデバイス情報を取得し、pk_rdev に格納する。

devatr は、デバイスドライバを登録する際に指定したデバイス属性である。

blksz は、固有データのブロックサイズ(バイト数)である。

nsub は、devnm で示すデバイスが属する物理デバイスのサブユニット数である。

subno は、サブユニットの番号である。subno は、0 ～ nsub−1 の値となる。

物理デバイスの場合は subno=0、サブユニットの場合は subno=(サブユニット番号+1)=(1 ～ nsub)となる。

pk_rdev=NULL とした場合には、デバイス情報は格納されない。

戻り値に devnm のデバイスのデバイス ID を返す。

安全 API は安全ソフトウェアからオープン可能なデバイスのみを対象とできる。通常 API は通常ソフトウェアからオープン可能なデバイスのみを対象とできる。それ以外に対して操作しようとした場合は、エラーE_DACV を返す。

8.3.2.12 ts_oref_dev/tn_oref_dev – デバイスディスクリプタからデバイス情報取得

C 言語インタフェース

【安全 API】 ID devid = ts_oref_dev(ID dd, T_RDEV *pk_rdev);

【通常 API】 ID devid = tn_oref_dev(ID dd, T_RDEV *pk_rdev);

パラメータ

ID	dd	デバイスディスクリプタ
T_RDEV*	pk_rdev	デバイス情報を返す領域へのポインタ

リターンパラメータ

ID	devid	デバイス ID
	または	エラーコード
T_RDEV	pk_rdev	デバイス情報

デバイス情報

```
typedef struct st_rdev {
        ATR     devatr;         /* デバイス属性 */
        INT     blksz;          /* 固有データのブロックサイズ（-1:不明）*/
        INT     nsub;           /* サブユニット数 */
        INT     subno;          /* 0:物理デバイス　1～nsub:サブユニット番号+1 */
} T_RDEV;
```

エラーコード

E_ID	dd が不正またはオープンされていない
E_PAR	パラメータエラー（rdev で指定されたアドレスが不正）
E_CTX	コンテキストエラー（タスク部および準タスク部以外、ディスパッチ禁止状態、割込み禁止状態で実行）
E_DACV	ドメインアクセス保護違反（dd は他のドメインでオープン）
E_MACV	メモリアクセス違反エラー（rdev に対するアクセス権限がない、または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

dd で指定したデバイスディスクリプタのデバイスのデバイス情報を取得し、pk_rdev に格納する。

pk_rdev の内容は、ts_ref_dev/tn_ref_dev と同じである。

pk_rdev=NULL とした場合には、デバイス情報は格納されない。

戻り値に dd のデバイスのデバイス ID を返す。

dd は、自ドメインのソフトウェアによりオープンされたデバイスのデバイスディスクリプタでなければならない。他のドメインでオープンされたデバイスディスクリプタの場合はエラーE_DACV となる。

8.3.3 割込み管理機能の API

割込み管理機能の詳細については、「4.2 割込み管理機能」を参照。

8.3.3.1 ts_def_int – 割込みハンドラの登録

C 言語インタフェース

【安全 API】 ER ercd = ts_def_int(UINT dintno, CONST T_DINT *pk_dint);

【通常 API】 なし

パラメータ

UINT	dintno	割込みハンドラ番号
CONST T_DINT*	pk_dint	割込みハンドラ定義情報へのポインタ

割込みハンドラ定義情報

```
typedef struct st_dint {
        ATR     intatr;         /* 割込みハンドラ属性 */
        FP      inthdr;         /* 割込みハンドラ起動アドレス */
        INT     priority;       /* 実行優先度 */
        RELTIM  maxrtim;        /* 最大連続実行時間 */
} T_DINT;
```

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_RSATR	予約属性エラー（intatr が不正あるいは利用できない）
E_PAR	パラメータエラー（dintno, pk_dint, inthdr, maxrtim が不正あるいは利用できない）
E_CTX	コンテキストエラー（タスク部および準タスク部以外で実行）
E_MACV	メモリアクセス違反エラー（pk_dint、inthdr に対するアクセス権限がない、または対象のメモリ領域が存在しない）
E_DOMAIN	ドメインエラー

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

dintno で指定される割込みハンドラ番号に対して割込みハンドラを定義し、割込みハンドラを使用可能にする。すなわち、dintno で指定される割込みハンドラ番号と割込みハンドラのアドレスや属性との対応付けを行う。

割込みハンドラ番号は実装定義である。

本 API はシステムドメインのタスクからのみ発行できる。他のドメインから使用された場合は、エラーE_DOMAIN とする。また、システムドメインにおいてタスク部および準タスク部以外から使用された場合はエラーE_CTX とする。

intatr には、対象割込みハンドラに対する割込みハンドラ属性を設定する。
intatr には、次のような指定を行う。

intatr := [TIA_MLTINT]

TIA_MLTINT　　多重割込みを許す

```
#define TIA_MLTINT          (0x00000008U)       /* 多重割込みを許可 */
```

割込みハンドラは C 言語の関数として以下の形式で記述する。以下の関数へのポインタ inthdr を、割込みハンドラ起動アドレス inthdr に指定する。

```
void    inthdr( UINT dintno )
{
        /*
                割込み処理
         */
        return;             /* 割込みハンドラの終了 */
}
```

割込みハンドラに渡される dintno は、発生した割込みの割込みハンドラ番号で、本 API で指定するものと同一である。

maxrtim には、割込みハンドラが連続して実行できる最大時間をミリ秒単位で設定する。maxrtim の最小値は 1 ミリ秒である。また、maxrtim の最大値はシステムドメインの連続実行時間上限値である。
割込みハンドラが maxrtim を超えて連続実行した場合は異常例外が発生する。割込みハンドラの実行時間の確認は、割込みハンドラの終了時に行われる。よって割込みハンドラの実行中にその処理が中断されるわけではない。

割込みハンドラに入った時点で、システムコールの呼び出しが可能な状態となっている。
割込みハンドラはタスク独立部として実行される。従って、割込みハンドラの中では、待ち状態に入るシステムコールや、自タスクの指定を意味するシステムコールを実行することはできない。
割込みハンドラの中でシステムコールを発行することにより、それまで実行状態(RUNNING)であったタスクがその他の状態に移行し、代わりに別のタスクが実行状態(RUNNING)となった場合でも、割込みハンドラ実行中はディスパッチ(実行タスクの切り替え)が起こらない。ディスパッチが必要になっても、まず割込みハンドラを最後まで実行することが優先され、割込みハンドラを終了する時にはじめてディスパッチが行われる。すなわち、割込みハンドラ実行中に生じたディスパッチ要求はすぐに処理されず、割込みハンドラ終了までディスパッチが遅らされる。これを遅延ディスパッチの原則と呼ぶ。

pk_dint=NULL とした場合には、前に定義した割込みハンドラの定義解除を行う。割込みハンドラの定義解除状態では、デフォルトハンドラが設定される。
デフォルトハンドラは、システムが定義する割込みハンドラである。カーネル初期化状態以降であれば、常に利用可能である。

既に定義済みの割込みハンドラ番号に対して、割込みハンドラを再定義することも可能である。
再定義のときにも、あらかじめその番号のハンドラの定義解除を行っておく必要はない。既に割込みハンドラが定義された dintno に対して新しいハンドラを再定義しても、エラーにはならない。

補足説明

割込みハンドラは、TRON Safe Kernel の起動時に初期登録 API を使用して登録することもできる。本 API は、TRON Safe Kernel 起動後に割込みハンドラの定義を解除、または再定義するために使用する。

8.3.4　故障診断機能の API

故障診断機能の詳細については、「6.3 故障診断機能」を参照。

8.3.4.1 ts_sta_fdh – 故障診断ハンドラの動作開始

C 言語インタフェース

【安全 API】 ER ercd = ts_sta_fdh(ID fdhid);

【通常 API】 なし

パラメータ

ID	fdhid	故障診断ハンドラ ID

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（fdhid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（fdhid の故障診断ハンドラが存在しない）
E_OACV	オブジェクトアクセス違反（対象の故障診断ハンドラが API 操作不可）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

fdhid で指定された故障診断ハンドラの動作開始し、起動待ち状態にする。

起動待ち状態となった故障診断ハンドラは、指定されている周期または API による起動で実行が可能である。

対象の故障診断ハンドラの属性が TA_API 属性（API による操作可能）ではない場合は、本 API はエラーE_OACV となる。

対象の故障診断ハンドラがすでに、起動待ち状態であれば何もせずに起動待ち状態を維持する。

8.3.4.2 ts_stp_fdh – 故障診断ハンドラの動作停止

C 言語インタフェース

【安全 API】 ER ercd = ts_stp_fdh(ID fdhid);

【通常 API】 なし

パラメータ

ID	fdhid	故障診断ハンドラ ID

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（fdhid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（fdhid の故障診断ハンドラが存在しない）
E_OACV	オブジェクトアクセス違反（対象の故障診断ハンドラが API 操作不可）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

fdhid で指定された故障診断ハンドラを停止状態にする。

対象の故障診断ハンドラの属性が TA_API 属性（API による操作可能）ではない場合は、本 API はエラーE_OACV となる。

対象の故障診断ハンドラがすでに停止状態になっている場合は何もしない。

8.3.4.3 ts_act_fdh – 故障診断ハンドラの起動

C 言語インタフェース

【安全 API】 ER ercd = ts_act_fdh(ID fdhid);

【通常 API】 なし

パラメータ

ID	fdhid	故障診断 ID

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_ID	不正 ID 番号（fdhid が不正あるいは利用できない）
E_NOEXS	オブジェクトが存在していない（fdhid の故障診断ハンドラが存在しない）
E_OBJ	オブジェクトの状態が不正（対象の故障診断ハンドラが起動待ち状態でない）
E_OACV	オブジェクトアクセス違反（対象の故障診断ハンドラが API 操作不可）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

fdhid で指定された故障診断ハンドラを起動する。

具体的には、対象の故障診断ハンドラを起動待ち状態から実行状態へと遷移させる。

対象の故障診断ハンドラの属性が TA_API 属性（API による操作可能）ではない場合は、本 API はエラーE_OACV となる。

対象の故障診断ハンドラが起動待ち状態でない場合は、本 API はエラーE_OBJ となる。

8.3.5　異常例外機能の API

異常例外機能の詳細については、「6.2 異常例外機能」を参照。

8.3.5.1 ts_def_aeh – 異常例外ハンドラ登録

C 言語インタフェース

【安全 API】 ER ercd = ts_def_aeh(UINT aexpno, CONST T_DAEH* pk_daeh);

【通常 API】 なし

パラメータ

UINT	aexpno	異常例外番号
CONST T_DAEH*	pk_daeh	異常例外ハンドラ定義情報へのポインタ

異常例外ハンドラ定義情報

```
typedef struct st_daeh {
        ATR     aehatr;         /* 異常例外ハンドラ属性 */
        FP      aehdr;          /* 異常例外ハンドラの起動アドレス */
} T_DAEH;
```

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_RSATR	予約属性エラー（aehatr が不正あるいは利用できない）
E_PAR	パラメータエラー（aexpno, pk_deah, aehdr,が不正あるいは利用できない）
E_CTX	コンテキストエラー（タスク部および準タスク部以外で実行）
E_DACV	ドメインアクセス保護違反（システムドメイン以外から実行）
E_MACV	メモリアクセス違反エラー（pk_daeh、aehdr に対するアクセス権限がない、または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	×	×

解説

aexpno で指定される異常例外ハンドラ番号に対して異常例外ハンドラを定義する。
すなわち、aexpno で指定される異常例外ハンドラ番号と異常例外ハンドラのアドレスや属性との対応付けを行う。
異常例外ハンドラ番号は実装定義である（異常例外ハンドラ番号の種類は「6.2.2 異常例外の種類」を参照）。

本 API はシステムドメインのタスクからのみ発行できる。他のドメインから使用された場合は、エラーE_DACV とする。また、システムドメインにおいてタスク部および準タスク部以外から使用された場合はエラーE_CTX とする。

aehatr には、対象異常例外ハンドラに対する異常例外ハンドラ属性を設定する。
aehatr には、次のような指定を行う。

aehatr := [TAA_UPDATE | TAA_RETURN]

TAA_UPDATE	安全 API による異常例外ハンドラの変更を許可
TAA_RETURN	異常例外ハンドラの終了後に、元の状態に復帰可能

```
#define TAA_UPDATE          (0x00000001U)       /* 安全 API による異常例外ハンドラの変更を許可 */
#define TAA_RETURN          (0x00000002U)       /* 異常例外ハンドラの終了後に、元の状態に復帰可能 */
```

TAA_UPDATE 属性の異常例外ハンドラは、安全 APIにより他の異常例外ハンドラへ変更が可能である。TAA_UPDATE 属性でない異常例外ハンドラは、本 API による異常例外ハンドラの変更が許されない。

TAA_RETURN 属性の異常例外ハンドラは、戻り値として AE_RETURN(=1)を返すことにより、ハンドラを終了し、元の状態に復帰することができる。戻り値として AE_NORETURN(=0、または AE_RETURN 以外の値)を返した場合は、TRON Safe Kernel は安全状態に遷移し、すべてのプログラムの実行を停止する。

異常例外ハンドラは C 言語の関数として以下の形式で記述する。以下の関数へのポインタ aehdr を、異常例外ハンドラ起動アドレス aehdr に指定する。

```
INT     aehdr( UINT aexpno,   INT tskstat,   VP aexppar )
{
        /*
                異常例外処理
         */
        return AE_RETURN;               /* 戻り値として AE_RETURN もしくは AE_NORETURN を返す */
}
```

関数のパラメータとして、発生した異常例外の異常例外番号が axepno に渡される。tskstat には異常例外が発生したときのカーネル状態が渡される。aexppar は異常例外毎に定められた情報である。
異常例外ハンドラに渡される aexpno は、発生した異常例外のハンドラ番号で、本 API で指定するものと同一である。

異常例外ハンドラに入った時点で、システムコールの呼び出しが可能な状態となっている。
ただし異常例外ハンドラから使用可能な API は限定される。詳細は、「6.2.7 異常例外ハンドラから使用可能な API」を参照。

異常例外が発生すると TRON Safe Kernel は最優先で異常例外ハンドラを実行する。その際、すべての割込みが禁止される。
異常例外ハンドラは実行中に割込みを許可してはならない。

pk_aehdr=NULL とした場合には、前に定義した異常例外ハンドラの定義解除を行う。異常例外ハンドラの定義解除状態では、デフォルトハンドラが設定される。
デフォルトハンドラは、システムが定義する異常例外ハンドラである。カーネル初期化状態以降であれば、常に利用可能である。

既に定義済みの異常例外ハンドラ番号に対して、異常例外ハンドラを再定義することも可能である。
再定義のときにも、あらかじめその番号のハンドラの定義解除を行っておく必要はない。既に異常例外ハンドラが定義された aexpno に対して新しいハンドラを再定義しても、エラーにはならない。

補足説明

異常例外ハンドラは、TRON Safe Kernel の起動時に初期登録 API を使用して登録することもできる。本 API は、TRON Safe Kernel 起動後に異常例外ハンドラの定義を解除、または再定義するために使用する。

8.3.5.2 ts_ras_aexp – 異常例外の発生

C 言語インタフェース

【安全 API】 ER ercd = ts_ras_aexp(UINT aexpno, VP aexppar);

【通常 API】 なし

パラメータ

UINT	aexpno	異常例外番号
VP	aexppar	異常例外の情報

リターンパラメータ

ER	ercd	エラーコード

エラーコード

E_OK	正常終了
E_PAR	パラメータエラー（aexpno の値が不正）
E_MACV	メモリアクセス違反エラー（aexppar のメモリ領域に対するアクセス権限がない、または対象のメモリ領域が存在しない）

利用可能なコンテキスト

タスク部	タイムイベントハンドラ	タスク独立部
○	○	○

解説

aexpno で指定されたユーザ定義の異常例外を発生させる。

異常例外が発生すると、他のプログラムの実行は中断され、対応する異常例外ハンドラが実行される。aexppar で指定された情報は異常例外ハンドラにパラメータとして渡される。

もし、発生させたユーザ定義の異常例外に対して、異常例外ハンドラが定義されていなかった場合は、未定義異常例外として処理される。

9. 付録

9.1 仕様策定の経緯

本仕様書は、トロンフォーラムにおいて、T－Kernel2.0 仕様をもととした機能安全に対応したリアルタイム OS の仕様の策定を目的に、TRON Safe Kernel ワーキンググループ（2016 年 4 月～2017 年 12 月）を設けて仕様検討を行い、作成された。

TRON Safe Kernel ワーキンググループにご参加を戴いたメンバーを以下に記載致しますと共に、そのご協力に対して謝意を表します。

TRON Safe Kernel ワーキンググループ委員リスト(順不同)

座長:	豊山祐一	株式会社日立超 LSI システムズ
	小滝総一朗	株式会社日立超 LSI システムズ
	田丸厚司	株式会社日立超 LSI システムズ
	大島訓	株式会社日立製作所
	竹下若菜	株式会社日立製作所
	峯博史	株式会社日立製作所
	若林昇	株式会社日立製作所
	川野敏弘	ルネサス エレクトロニクス株式会社
	毛利裕二	ルネサス エレクトロニクス株式会社
	林兼治郎	イーソル株式会社
	小林康浩	株式会社ソシオネクスト
	松為彰	パーソナルメディア株式会社
	神尾真人	ユーシーテクノロジ株式会社
	山田浩之	ユーシーテクノロジ株式会社
幹事:	由良修二	ユーシーテクノロジ株式会社