Partition OS
Safety Requirements Specification

Ver.1.00

2012/12/25

株式会社ヴィッツ　組込制御開発部

一致性検証	承認	審査	作成

改訂履歴

Version	Date	Detail	Editor	所属
0.01	2011/10/25	・ 新規作成	原	株式会社ヴィッツ
0.10	2011/02/14	・ TUV からの指摘内容を反映.	原	株式会社ヴィッツ
0.11	2011/04/20	・ TUV Review Report の内容を反映	本田	名古屋大学
0.20	2011/09/13	・ TUV からの指摘内容を反映.	本田	名古屋大学
0.30	2011/09/24	・ TUV からの指摘内容を反映.	本田	名古屋大学
0.40	2011/11/04	・ TUV からの指摘内容を反映. ・ BCC+を追加	本田	名古屋大学
0.5	2012/09/13	・ TUV review report of the ParOS 2012/07 の指摘内容を反映.	本田	名古屋大学
0.6	2012/09/18	・ 2012/09/18 の TUV ミーティングでの指摘内容を反映.	本田	名古屋大学
0.61	2012/09/19	・ 2012/09/19 の TUV ミーティングでの指摘内容を反映.	本田	名古屋大学
0.62	2012/09/30	・ TUV reportv ver 1.1[2012/09/27] での指摘内容を反映	本田	名古屋大学
1.00	2012/12/25	・ コンセプトレポート向けのリリース	原	株式会社ヴィッツ

目次

1. 本文書の位置づけ

本ドキュメントは，"平成 22 年度 戦略的基盤技術高度化支援事業 故障未然防衛機能を有した高信頼ソフトウェアプラットフォームの開発" プロジェクトにおいて策定したパーティション OS（以下，ParOS）の要求仕様と外部仕様についてまとめたものである．

本仕様書は，ParOS の機能レベルの仕様について延べている．実装レベルの仕様（設計）については，実装時に記述する．

OS 仕様のコンセプト（要求仕様）に関しては，Reliable OS Safety Concept を参照のこと．

コンセプトレポート取得版では文書間の要件管理のためタグを記載しておりますが，本公開版は一部文書のみの公開のためタグを記載しておりません．

2. 関連文書

以下の文書は，本仕様書から参照しているもの，または本仕様書を理解するために必要なものである

表 1 関連文章

Reference ID	Title	Revision	Data
[IEC]	IEC 61508:2010 Edition 2.0 Part1-7	2.0	2010/04
[ISO]	ISO 26262:2011 Part10	First	2011/11/15
[SafeOS_SC]	Safe OS Safety Concept	0.90	2009/12/09
[SafeOS_SRS]	Safe OS Software Safety Requirement Specification	0.90	2010/01/13
[SafeOS_SA]	Safe OS Safety Analysis	0.90	2009/12/10
	Safe OS Safety Analysis(detail)	0.90	2009/03/24
[SafeOS_SM]	Safe OS Safety Manual	1.00	2010/02/12
[ParOS_SC]	Partition OS Safety Concept	1.00	2012/12/15
[ParOS_SA]	Partition OS Safety Requirements Analysis Plan And Results Report	1.00	2012/12/15
	Partition OS Safety Requirement Analysis Result Report(detail)	1.00	2012/12/25
[ParOS_SM]	Partition OS Safety Manual	1.00	2012/12/25
[WITZ_S]	Function Safety Management Standard	2.3.0	2012/01/16

3. 安全関連規格

以下の規格は，安全関連機能を達成するため適用される．

・IEC 62508 ed2.0[IEC]
E/E/PE 安全関連システムの機能安全

4. 安全目標

ParOS の Safety Goal を以下に示す．【SG-1】と【SG-2】は，ISO26262 Part6 D.2 に記載される 2 個のパーティショニングの目的と同等である．なお，パーティショニングされているプログラムの単位をパーティションと呼ぶ．

Safety Goal 1【SG-1】
パーティションの故障が，他のパーティションに影響を及ぼさないこと．異なる安全度水準のパーティションを同一のシステム上で実行できること

Safety Goal 2【SG-2】
あるパーティションに変更があった場合，他のパーティションの再検証の必要がない，もしくは検証レベル（労力）を下げることが可能であること．パーティションを個別に検証可能（検証範囲や工数を減らせる）であること．

Safety Goal 3【SG-3】
故障診断率（Diagnosys Coverage）90%以上の診断方法を含む故障検出ライブラリを提供する．
故障検出ライブラリを含む ParOS 自身は，SIL3 を満たすために必要なプロセスで開発すること．同一の ParOS モジュールを用い，デュアルチャンネル・ハードウェア構成としハードウェアのフォールトトレランスが 1 の場合は SIL 3 を満たすことができ，シングルチャンネル・ハードウェア構成でフォールトトレランスが 0 の場合は SIL 2 を満たすことができること．

Safety Goal 4【SG-4】
パーティション間通信で通信相手に問題が発生し停止した場合に，その停止を検知できること．

5. ハードウェア要件

ParOS を実行するために，システムは次のハードウェアを備える必要がある．

基本アーキテクチャ

MCU は中央処理演算処理装置(CPU)，割込みコントローラ(INTC)，アドレスバス(Bus)，内蔵 ROM(ROM)，内蔵 RAM(RAM)，タイマ，ウォッチドッグ機能(MCU 内蔵である場合)を有すること．【HW:SIR-1】

メモリ保護ハードウェア

MCU は実行しているプログラムに対して特定のメモリへのアクセスを制限する機構(MPU や MMU)を持つこと．【HW:SIR-2】

ユーザーモードデバイスアクセス

メモリアクセスの制限のために，非特権モードを持つ場合，非特権モードであっても，許可されたデバイスへは直接アクセス可能であること．【HW:SIR-3】

クロック

周辺回路としてはメインクロック，サブクロック，内蔵クロック，電源，ウォッチドッグ(MCU 内蔵でない場合)を有すること．【HW:SIR-5】

ウオッチドック

ウォッチドッグは，MCU 内蔵機能または外部部品のいずれかを用いる．フォールトトレランス 1 となるハードウェアを用いること．【HW:SIR-6】
※SIL2 を満たすことを目標とする場合，フォールトトレランス 0 のハードウェアを用いること．

パーティションスケジュール用タイマ

タイマは 1 つのカウンタに対してコンペア値を 2 個持ち，カウンタの値をクリアせずにコンペア値を変更可能であること．【HW:SIR-7】

さらに，実行性能のため以下の機能を持つことを推奨する．

低オーバヘッドの割込み禁止許可

割込みを低オーバヘッドで個別に許可・禁止することが可能であること．【HW:SIR-4】

6. コンフォーマンスクラス

ParOS は適用するシステムに要求されるパーティションの強度（アプリケーション変更時のインパクトアナリシスの容易性）に応じて次の 3 種類のコンフォーマンスクラスを定める．

・BCC(Basic Conformance Class)
・BCC+(Basic Conformance Class Plus)
・ECC(Extended Conformance Class)
・DCC(Development Conformance Class)

それぞれのコンフォーマンスクラスがサポートする機能は次の通りである．ParOS の実装はいずれかのコンフォーマンスクラスに従う．

表 2 コンフォーマンスクラス

		BCC	BCC+	ECC	DCC
パーティショニング機能	サービス保護機能	○	○	○	○
	メモリ保護機能	○	○	○	○
	SafeOS 互換安全維持機能	○	○	○	○
	システム(OS)情報保護機能	○	○	○	○
	周期実行ポリシー	○	○	○	○
	SafeOS 互換機能	○	○	○	○
一般機能	アプリケーション割込み	○	○	○	○
	タイムウィンドウハンドラ	○	○	○	○
	アトミックハンドラ	○	○	○	○
	システム例外ハンドラ	○	○	○	○
	パーティション間通信	○	○	○	○
	アイドル属性			○	○
	スケジューリングモード変更			○	○
	システム割込み		△	○	○
	システムタスク				○
	システムタイムイベントハンドラ				○
パーティションレベル		Lv4	Lv3	Lv3	－

BCC は最もパーティション強度が高く ECC は OS としての機能は BCC と比較して多いが，パーティションの強度は低い．DCC ではパーティショニングが適用されないシステムパーティションのタスクやタイムイベントハンドラをサポートするため，既存システムの ParOS を用いたパーティショニング環境への移行を容易にするため，開発時に用いることを想定している．

[ParOS_SC]で定義されているパーティションレベルとの関係としては，BCC は ParLv4，ECC は ParLv3 を実現できる．DCC はどのパーティションレベルも実現できない．

仕様書内で ECC/DCC でのみ使用可能な機能や API については，その記述の後に[ECC]ないし[DCC]を付加する．

BCC+は，BCC に機能制限付きシステム割込みをサポートしたものである．BCC+は ECC と同じパーティションレベルであるが，ECC と比較して，機能が少ないため，実装が容易であり，ほぼ BCC と同様の実装となる．ECC では実装が複雑化して，オーバヘッドが大きくなり，BCC では，システム割込みが使えず，組込みシステムの高リアルタイム性を実現することが困難であるため定めた．機能制限付きシステム割込みは，システム割込みのセクションで解説する．

7. 全体構成

ParOS の全体構成を示す．ParOS は，Executive, SafeOS，COM から構成されている．これらのメモリ領域は，アプリケーションから保護されている．アプリケーションはパーティションと呼ばれる単位で管理される．本仕様書では，アプリケーションとパーティションは同意語である．

7.1. Executive

Executive は，各パーティショング機能を実現する．また，ランダムハードウェア故障検出を行うための故障検出ライブラリを持つ．各パーティションの実行シーケンスモニタに対して仮想化されたタイマや W/D タイマを提供する．

7.2. SafeOS

SafeOS は，パーティションに SafeOS 互換の機能を提供する．ただし，【SG1】，【SG2】に反する以下の機能は使用することができない．

・タイムイベントハンドラ（周期ハンドラ・アラームハンドラ）
・割込みハンドラ・割込みサービスルーチン
・CPU 例外管理機能

7.3. COM

COM はパーティション間の通信を実現する．送受信データはチャネルと呼ばれるオブジェクトに格納される．各パーティションはインタフェースを介してチャネルに接続する．

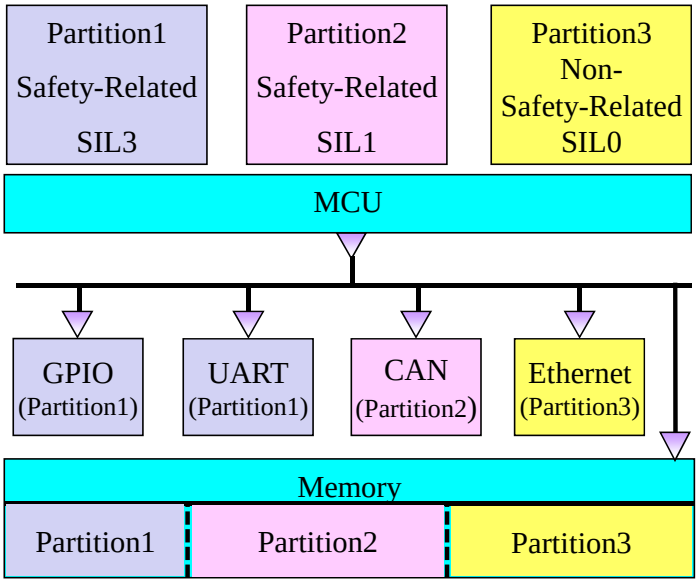

図 7-1. ParOS の全体構成

8. システムコンフィギュレーション

8.1.1. 静的なオブジェクト生成

ParOS では，ベースとなる SafeOS と同様にオブジェクトは静的に（設計時）のみ生成される．オブジェクトの生成は後述するコンフィギュレーションファイルに静的 API により記述する．

8.1.2. コンフィギュレーション方法

オブジェクトの生成情報は，静的 API をコンフィギュレーションファイルに記述することで行う．ParOS の静的 API は SafeOS の静的 API を拡張したものである．

8.2. 処理単位の優先度

8.2.1. SafeOSの優先度

SafeOS では，タスクに正の優先度，負の優先度は割込みハンドラに設定可能であり，値が大きい程優先して実行される．

8.2.2. ParOSの優先度

ParOS では，タスクは SafeOS と同様に正の優先度を設定可能である．負の優先度に関しては-1 から標準では-16 までの値をアプリケーション割込みハンドラやタイムウィンドウハンドラの優先度として設定可能である．優先度の上限は実装で拡張可能である．アプリケーション割込みハンドラやタイムウィンドウハンドラの優先度の最大値より 1 高い優先度(-16 の場合は-17)はアプリケーション割込みに設定される．さらに高い優先度(-16 の場合は-18 から)をシステム割込みに設定可能である．システム割込みに設定可能な優先度の上限は実装依存である．

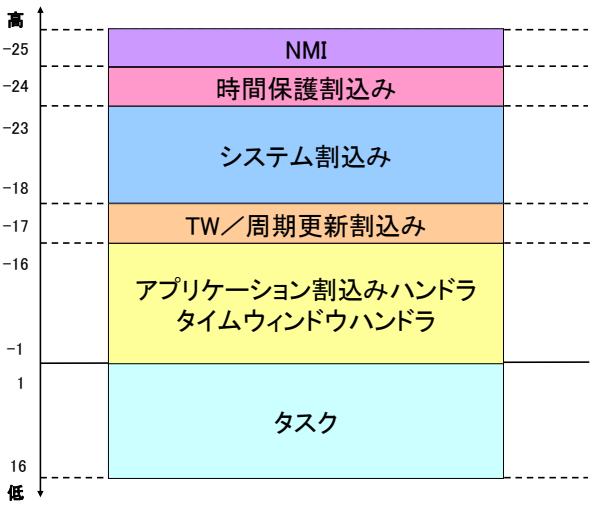

図 8-1. ParOS の優先度

8.3. API

8.3.1. ParOS APIとSafeOS互換API

ParOS は, ユーザーに対して ParOS の機能を呼び出す API を提供する. また, SafeOS と互換の API（SafeOS 互換 API）も提供する. 使用可能な SafeOS 互換 API には制限ある. 詳細については, SafeOS 互換機能の節を参照のこと.

8.3.2. ParOS APIの名称

SafeOS 互換 API は ITRON 仕様をベースとしており, 各 API はタスクコンテキストまたは非タスクコンテキストから呼び出し可能であり, 非タスクコンテキストから呼び出し可能な API はプリフィックスとして"i"が付加されている. 一方, ParOS API は, 呼び出し可能なコンテキストとして, タスクコンテキスト, 非タスクコンテキスト以外に, パーティションやパーティション初期化/終了処理ルーチン等の処理単位が追加されており, 呼び出し可能な処理単位の組み合わせが複雑であるため, プリフィックスとして"i"を付加するルールは用いない.

9. システム（ParOS）関連

9.1. 概要

本節では，システム全体(ParOS)に関わる機能について説明する．

9.2. ParOSの状態

ParOS は状態を持つ．取り得る状態は以下の通り．

未定義状態(TSS_UNDEF)

- ParOS が一度も動作していない状態．
- 電源投入直後にこの状態となる．

システム初期化中状態(TSS_INIT)

- ParOS が起動し初期化処理を実行している状態．
- 全ての割込みは禁止されている

システム終了処理中状態(TSS_TER)

- ParOS が終了処理を実行している状態．
- 全ての割込みは禁止されている．

システム停止状態(TSS_STOP)

- ParOS が動作したのちに終了した状態．
- 全ての割込みは禁止されている．

システム通常状態(TSS_NORMAL)

- パーティションスケジューリングが実施され，パーティションが実行されている状態．
- 許可されている割込みは受け付けられる．

9.3. ParOSの状態遷移

未定義状態

・電源投入時/リセット割込みルーチン実行時に遷移

システム初期化中状態

- 未定義状態 から遷移
  - ➢ システム起動 API の呼び出し．
- システム終了処理中状態 から遷移
  - ➢ システム起動 API の呼び出し．

システム通常状態

- システム初期化中状態 から遷移
  - システム初期化ルーチン終了

システム終了処理中状態

- システム通常状態から
  - システム終了 API 呼び出し

システム停止状態

- システム終了処理中状態から
- システム終了ルーチン終了

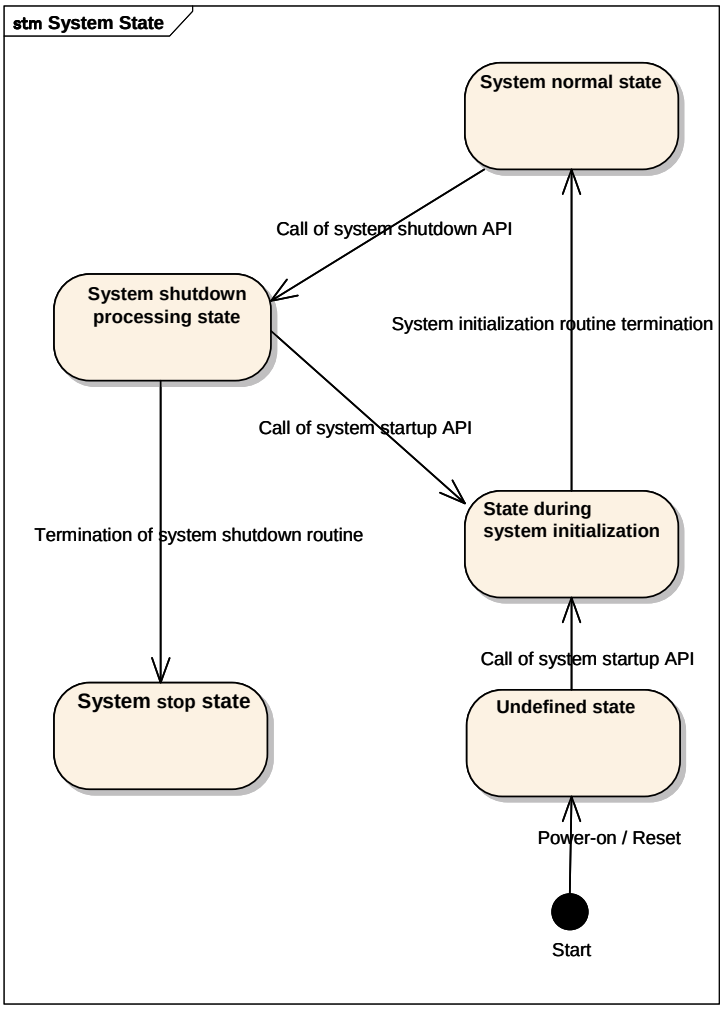

図 9-1. ParOS の状態

9.4. システム起動API

システムを起動する API．電源投入後等に実行されるスタートアップコードから呼び出されることを想定している．システム起動 API が呼び出されると ParOS が動作を開始する．

9.5. システム初期化ルーチン

システム初期化中状態で実行される関数．システムに 1 個登録可能．コンフィギュレーション時にシステム初期化ルーチンとする関数を指定する．指定されない場合はコンフィギュレーションエラーとなる．引数には，システム初期化ルーチン定義 API で指定された拡張情報が渡される．システム初期化ルーチンからはいかなる API も呼び出すことはできない．

9.6. システム終了API

システムを終了する API．再起動時にも呼び出される．

9.7. システム終了処理ルーチン

システム終了 API を呼び出すことにより実行される．システムに 1 個登録可能．コンフィギュレーション時にシステム終了処理ルーチンとする関数を指定する．指定されない場合はコンフィギュレーションエラーとなる．引数には，システム終了ルーチン定義 API で指定された拡張情報が渡される．システム終了処理ルーチンからは，システム起動 API 以外の API を呼び出すことはできない．

9.8. システム起動の流れ

1. 電源投入
   - システム状態は未定義状態へ
   - スタータアップルーチンが実行される．

2. システム起動 API 呼び出し．
   - スタートアップルーチンから呼び出す．
   - システム状態はシステム初期化中状態へ
   - ParOS の内部の初期化

3. システム初期化ルーチンの実行
   - ユーザー定義の初期化処理．

4. システム初期化ルーチンの終了
   - システム状態はシステム通常状態へ．

5. システムパーティションの初期化処理実行

6. システムパーティションの初期化処理終了
   - システムパーティションは通常状態へ

7. 通常動作
   - スケジューリングモードで指定されたパーティションスケジューリングに従い，起動時実行属性を指定されたパーティションが初期化中状態で実行される．起動時実行属性を指定されていないパーティションは停止状態に遷移する．
   - 各パーティションで初期化処理を行い，通常状態に遷移．

9.9. システム終了・再起動処理の流れ

1. システム終了 API の呼び出し．
   - システム状態はシステム終了処理中状態へ

2. 各パーティションの終了処理の実行

3. システムパーティションの終了処理実行

4. システム終了処理ルーチンの実行
   - ユーザー定義の終了処理．

5. 再起動か終了を選択
   - 再起動する場合
     - システム終了処理ルーチンからシステム起動 API を呼び出す．
     - システム状態はシステム初期化中状態へ
     - システム起動の流れの 2 へ
   - 終了する場合
     - システム終了ルーチンからリターン．
     - システム状態はシステム停止状態へ
     - システム停止

9.10. システム（ParOS）関連 ：API

9.10.1. システム起動API

【C 言語 API】

ER ercd = StartSystem(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_SYS	システムエラー（カーネルの誤動作）
E_CTX〔s〕	コンテキストエラー（未定義状態もしくはシステム終了処理中状態以外からの呼び出し以外）

【機能】

システム（ParOS）を起動する．呼び出すとシステム状態をシステム初期化中状態へ遷移させ，ParOS 内部の初期化処理を行う．その後，システム初期化ルーチンが呼び出される．

呼び出し可能なコンテキストは，未定義状態(スタートアップルーチンを想定)，もしくは，システム終了中状態(システム終了処理ルーチンを想定).

9.10.2. システム終了API

【C 言語 API】

ER ercd = ShutdownSystem(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_SYS	システムエラー（カーネルの誤動作）
E_CTX〔s〕	コンテキストエラー（システムパーティションの処理単位以外からの呼び出し）

【機能】

システム（ParOS）を終了する．システム状態をシステム終了中状態へ．各パーティションの終了処理を実行して，システム終了ルーチンを呼び出す．

呼び出し可能なコンテキストは，システムパーティションの処理単位．

9.10.3. システム状態参照API

【C 言語 API】

ER ercd = GetSystemState(T_SSTATE *pk_sstate)

【パラメータ】

T_SSTATE	*pk_sstate	システム状態を入れるパケットへのポインタ

【リターンパラメータ】

ER	ercd	エラーコード

*システム状態（パケットの内容）

SYS_STAT	sys_stat	現在のシステム状態
SCH_MODE	current_sch_mode	現在のスケジューリングモード.
SCH_MODE	next_sch_mode	次システム周期のスケジューリングモード
uint_t	count	起動回数（システム初期化中状態から システム通常状態への遷移回数）

【機能】

システム状態を参照する．取得可能な内容は以下の通り．

- 現在のシステム状態
- 現在のスケジューリングモード.
- 次システム周期のスケジューリングモード.
- 起動回数（システム初期化中状態からシステム通常状態への遷移回数）

9.10.4. システム初期化ルーチン定義API〔S〕

【静的 API】

DEF_SYSTEM_INI(ATR iniatr, intptr_t exinf, INIRTN inirtn)

【パラメータ】

ATR	iniatr	システム初期化ルーチン属性
intptr_t	exinf	システム初期化ルーチンの拡張情報
INIRTN	inirtn	システム初期化ルーチンの先頭番地

【エラーコード】

E_RSATR	予約属性（iniatr が不正または使用できない，システムパーティションの囲み以外に記述）
E_PAR	パラメータエラー（inirtn が不正）

【機能】

各パラメータで指定したシステム初期化ルーチンの追加情報に従って，システム初期化ルーチンを追加する．DEF_SYSTEM_INI をシステムパーティションの囲み以外に記述した場合には，E_RSATR エラーとなる．

9.10.5. システム終了処理ルーチン定義API〔S〕

【静的 API】

DEF_SYSTEM_TER(ATR teratr, intptr_t exinf, TERRTN terrtn)

【パラメータ】

ATR	teratr	システム終了処理ルーチン属性
intptr_t	exinf	システム終了処理ルーチンの拡張情報
TERRTN	terrtn	システム終了処理ルーチンの先頭番地

【エラーコード】

E_RSATR	予約属性（teratr が不正または使用できない，システムパーティションの囲み以外に記述）
E_PAR	パラメータエラー（inirtn が不正）

【機能】

各パラメータで指定したシステム終了処理ルーチン追加情報に従って，システム終了処理ルーチンを追加する. DEF_SYSTEM_TER をシステムパーティションの囲み以外に記述した場合には, E_RSATR エラーとなる.

10. パーティション

10.1. 概要

パーティションはタスクやカーネルオブジェクトの集合であり，複数のパーティションによりシステムが構成される．処理単位は必ずいずれかのパーティションに属する必要がある．

パーティションからは，SafeOS の API を呼び出すことが可能である．なお，SafeOS の状態は，パーティション毎に独立している．

10.2. パーティション状態

パーティションは個別に以下の状態を持つ.

通常状態

- パーティションに属するアプリケーション割込みが有効に.
- パーティションに属するタイムウィンドウハンドラが有効に.

実行状態(TPS_NORMAL)

- 割り当てられたタイムウィンドウが実行され，かつ実行するべき処理単位が存在し実行されている状態.

休止状態(TPS_IDLE)

- 割り当てられたタイムウィンドウが実行され，かつ実行するべき処理単位が存在しない状態.

実行可能状態(TPS_RUNNABLE)

- システム周期内に未実行の割り当てられたタイムウィンドウが存在する状態.

満了状態(TPS_EXPIRE)

- ステム周期内の割り当てられたタイムウィンドウが全て実行された状態．もしくはシステム周期内に割り当てられたタイムウィンドウが存在しない場合.

初期化中状態(TPS_INIT)

- パーティションの初期化を実行している状態.
- パーティションに属する割込みが無効に.
- パーティションに属するタイムイベントハンドラが無効に.

終了処理中状態(TPS_TER)

- パーティションの終了処理を実行している状態.
- パーティションに属する割込みが無効に.
- パーティションに属するタイムイベントハンドラが無効に.

停止状態(TPS_STOP)

- パーティションを終了した状態.
- パーティションに属する割込みが無効に.
- パーティションに属するタイムイベントハンドラが無効に.

10.3. パーティションの状態遷移

パーティションは以下のような状態遷移を行う.

停止状態

- 電源投入時/リセット割込みルーチン実行時に遷移
- 終了処理中状態 から遷移
  - パーティション終了処理ルーチンの終了

初期化中状態

- 停止状態 から遷移
  - 起動時実行属性が指定されており，システム状態が通常状態となった場合.
  - システムパーティションからのパーティション起動 API 呼び出しにより遷移

通常状態

- 初期化中状態 から遷移
  - パーティション初期化ルーチン終了
  - 休止状態へ遷移

終了処理中状態

- 通常状態 から遷移
  - パーティション終了 API 呼び出し.

実行可能状態

- 実行状態 から遷移
  - 割り当てられたタイムウィンドウが終了した場合.
- 満了状態 から遷移
  - 新たなシステム周期となった場合.
- 休止状態 から遷移
  - 実行すべき処理単位が発生した場合.

実行状態

- 実行可能状態 から遷移
  - タイムウィンドウが割当てられた場合.

満了状態

- 実行状態 から遷移

- システム周期内の自タイムウィンドウが全て割り当てられ，実行が終了した場合.

・ 休止状態 から遷移

- システム周期内の自タイムウィンドウが全て割り当てられ，実行が終了した場合.

休止状態

・ 実行状態 から遷移

- 実行するべき処理単位が存在しない場合.

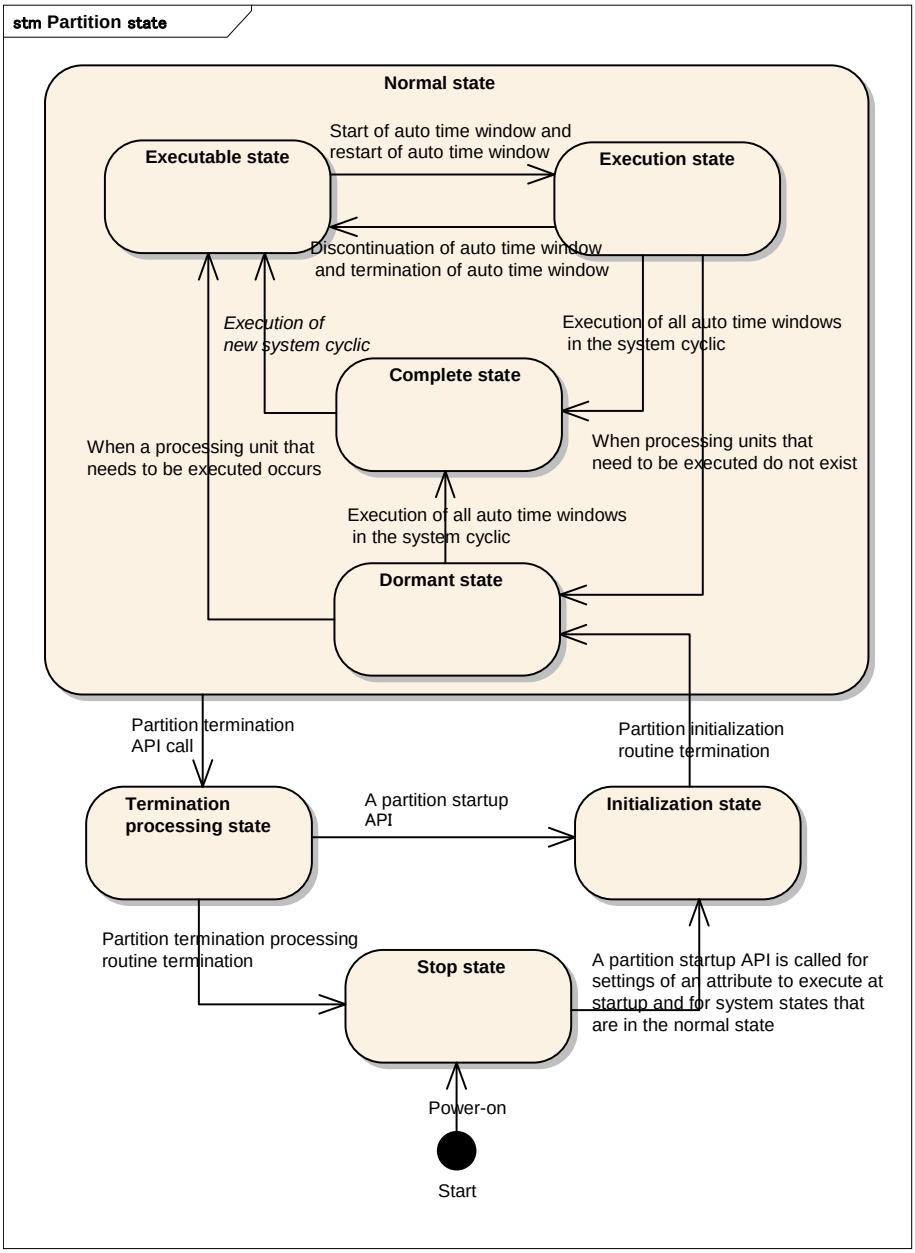

図 10-1. パーティションの状態

【仕様決定の理由】

満了状態は，想定以上のシステムパーティションの処理単位の動作等により，システム周期終了までに，パーティションに対してタイムウィンドウが割当てられなった場合にパーティション未満了エラーを発生させるために必要である．満了状態を用意することで，どのパーティションにタイムウィンドウが割り当てられなかったかを知ることができる．

10.4. パーティションID

パーティションにはそれぞれユニークな ID が付加される．システムパーティションの ID は PID_SYSTEM(-1)に予約されている．

10.5. メモリ領域

個々のパーティションの処理単位がアクセス可能な空間（メモリ，デバイス）には制限がある．許可されていない空間にアクセスした場合には，エラーが発生する．

アクセス可能な空間の設定については，"メモリ保護"節を参照のこと．

10.6. データの初期化

メモリ上のデータ（DATA/BSS）は幾つかのグループに分けられる．グループを特定のパーティションにのみアクセス可能なメモリに配置することで，パーティション専用のデータ（変数）とすることができる．

システム起動 API 呼び出し後に，ParOS は全てのデータ(DATA/BSS)を初期化する．このタイミング以外で，データが ParOS により自動的に初期化されることはない．

変数グループを指定して初期化する API を提供する．パーティション初期化・終了処理ルーチンで呼び出すことを想定している．また，API には正しく初期化できたか（メモリが故障してないか）を確認するオプションを指定可能である．

【仕様決定の理由】

API で提供するのは，ユーザーが変数を個別に初期化するより高速なためである．

10.7. OS状態の独立性

OS の振る舞いに影響を与える次の状態はパーティション毎に独立である．

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

そのため，あるパーティションでディスパッチ禁止状態とするシステムコールを呼び出したとしても，他のパーティションには影響を与えない．

全割込みロックフラグ，CPU ロックフラグ，割込み優先度マスクは，そのパーティションに所属する割込みにのみ影響を与える．

10.8. API発行の制限

API は同じパーティションに属する処理単位ないしカーネルオブジェクトにのみ発行可能である．

他のパーティションに対して API を発行することはできない．他のパーティションと通信したい場合は，後述のパーティション間通信を用いること．

他のパーティションに属するオブジェクトに対してカーネル API を発行した場合にはエラーコードとして E_OACV が返される．

この制限はシステムパーティションに対しても適用される．すなわち，システムパーティションの処理単位からは，システムパーティションに所属する処理単位ないしカーネルオブジェクトに対してのみ API を発行可能である．

10.9. システムパーティションとアプリケーションパーティション

システムパーティションは，システム上に 1 個存在する特別なパーティションである．システムパーティション以外のパーティションはアプリケーションパーティションと呼ぶ．

システムパーティションに属する処理単位は，特権モードで実行され，メモリアクセスの制限がない．また，実行するべき処理単位が存在すればタイムウィンドウを超えて実行される．そのため，システムパーティションの処理単位のシステム周期内での実行時間が事前に分かっている場合は，その合計時間分のタイムウィンドウをシステム周期の最後に確保し，アイドルタイムウィンドウとすることで，システムパーティションの処理単位の実行時間をリザーブし，後述のパーティション未満了エラーの発生を抑制することを推奨する．システムパーティションはシステム全体に影響を与えるため，使用時には[ParOS_SM]の 7.3.5 の注意事項を参照のこと．

アプリケーションパーティションは通常のパーティションであり，処理単位は非特権モードで実行される．それぞれのパーティションで属することができる処理単位は異なり，次に示す表のようになっている．

表 3 各パーティションに属する処理単位

	Partition	
	Application	System
タスク	○	○[DCC]
タイムウィンドウハンドラ	○	
アプリケーション割込みハンドラ	○	
パーティション初期化ルーチン	○	○
パーティション終了処理ルーチン	○	○
初期化ルーチン	○	○
終了処理ルーチン	○	○
パーティション例外ハンドラ	○	
システム初期化ルーチン		○
システム終了ルーチン		○
システム例外ハンドラ		○
システム割込みハンドラ		○[BCC+] [ECC][DCC]
システムタイムイベントハンドラ		○[DCC]
チェックフック		○

10.10.　処理単位の優先度

各パーティションに所属する処理単位の優先度は次のようになる．

システムパーティション

- システム例外ハンドラ
- システム割込み/システムタイムウィンドウハンドラ
- タスク
- アイドル処理

アプリケーションパーティション

- チェックフック（OS レベル相当，特権で動作）
- パーティション例外ハンドラ
- アプリケーション割込みハンドラ/タイムウィンドウハンドラ（タスクレベルで実行）
- タスク
- アイドル処理

＊注意事項

アトミックハンドラは呼び出した処理単位の優先度で起動される．起動後は最高優先度で実行する．システム割込みは入らない．最悪実行時間監視が有効となる．

10.11.　起動時実行属性

起動時実行属性が指定されたパーティションは，システム状態が通常状態となった場合に，自動的に初期化中状態となる．システムパーティションは，常に起動時実行属性が付加されている．

10.12.　オブジェクトID

カーネル状態はパーティション毎に独立であるが, ID は全てのパーティションで一意であるとする．

【仕様決定の理由】

仮想アドレス空間を使わないため，ID もこの考えに合わせる．また，自動割り付けするため，問題ない．

10.13.　割込みレベルの拡張

アプリケーション割込みハンドラとタイムイベントタスクには，負の割込み優先度を設定可能である．

10.14.　パーティション初期化ルーチン

パーティション初期化中状態で実行される関数．コンフィギュレーション時に，パーティション初期化ルーチンとする関数を指定する．指定されない場合はコンフィギュレーションエラーとなる．

パーティション毎に定義され，そのパーティションのタイムウィンドウ内で実行される．パーティション初期化ルーチンが実行されている間は，そのパーティションでは，他の処理単位は動作しない．

パーティション初期化ルーチンから呼び出し可能な API は次の通りである．

- チャネル開始
- データグループ初期化
- システム状態参照

10.15.　パーティション終了処理ルーチン

パーティション終了中状態で実行される関数．コンフィギュレーション時に，パーティション終了処理ルーチンとする関数を指定する．指定されない場合はコンフィギュレーションエラーとなる．

パーティション毎に定義され，そのパーティションのパーティションウィンドウ内で実行される．パーティション終了処理ルーチンが実行されている間は，そのパーティションでは，他の処理単位は動作しない．

パーティション終了処理ルーチンからリターンした場合，パーティションの状態は停止状態となる．

10.15.1. スケジューリングモードの変更[ECC]

ECC では，パーティション終了処理ルーチンからのリターン時にスケジューリングモードを変更することが可能である．パーティション終了処理ルーチンからの戻り値が 0 の場合，スケジューリングモードは変更とならない．戻り値として変更したいスケジューリングモードの ID を指定すると，システムパーティションに属するスケジューリングモード変更チェックフック が呼び出され引数として渡される．スケジューリングモード切り替えチェックフックフックは，指定されたスケジューリングをチェックして，許可される操作なら OK でリターンし，許可されない操作ならエラーでリターンする．OK でリターンした場合のみスケジューリングモードが変更される．その際，スケジューリングモードの切り替えタイミングは，次のシステム周期からである．

スケジューリングモードを変更すると，システム全体に影響を与える．使用上の注意点に関しては，[ParOS_SM] 7.3.3 を参照のこと．

10.15.2. パーティションの再起動

パーティションを再起動する場合は，パーティション起動 API を呼び出す.

呼び出し可能なAPI

パーティション終了処理ルーチンから呼び出し可能な API は次の通りである.

- チャネル停止
- データグループ初期化パーティション起動 API

10.16.　初期化ルーチン・終了処理ルーチン

初期化ルーチンと終了処理ルーチンは，他のカーネルオブジェクトと同様のいずれかのパーティションに属する.

初期化ルーチンは，パーティション初期化ルーチンの終了後，パーティションの状態が通常状態に遷移した後に実行される.

終了処理ルーチンは，パーティション終了 API の呼び出し後に，パーティション終了処理ルーチン実行前に実行される.

10.17.　パーティション初期化処理の流れ

パーティション初期化処理はパーティションに割り当てられたタイムウィンドウ内で実行される．この流れはアプリケーションパーティションに適応される．システムパーティションについては，"システム"節のシステム起動の流れを参照のこと．

1. パーティション起動 API の呼び出し or システム初期化ルーチンの終了時に起動時実行属性が付加されている場合
   - パーティションの状態は初期化中状態へ

2. アプリケーションパーティション:割り当てられたタイムウィンドウの実行
   - パーティション初期化ルーチンの起動

3. パーティション初期化ルーチンの実行
   - 必要に応じてパーティションに関連した初期化を行う

   例)
   - データの初期化.
   - パーティション間チャネルを通常状態に.

4. パーティション初期化ルーチンの終了
   - パーティション状態を通常状態へ.
   - 初期化ルーチン(ATT_INI で登録)の起動

5. 初期化ルーチンの実行

6. 初期化ルーチンの終了
   - パーティションに属する割込みが有効に.
   - パーティションに属するタイムイベントハンドラが有効に.
   - 初期起動タスクを起動.

10.18.　パーティション終了・再起動処理の流れ

パーティション終了・再起動処理はパーティションに割り当てられたタイムウィンドウ内で実行される．この流れはアプリケーションパーティションに適応される．システムパーティションについては，"システム"節のシステム終了・再起動処理の流れを参照のこと．

1. パーティション終了 API の呼び出し
   - パーティションに属する割込みが無効に．
   - パーティションに属するタイムイベントハンドラが無効に．
   - 終了処理ルーチン(ATT_TER で登録)の起動

2. 終了処理ルーチン(ATT_TER で登録)の実行

3. 終了処理ルーチン終了
   - パーティション終了ルーチン起動

4. パーティション終了ルーチン実行
   - パーティションの状態は終了中状態に
   - 必要に応じてパーティションに関連した終了処理を行う

   例)
   - データの初期化
   - パーティション間チャネルを停止状態に．

5. パーティション終了ルーチン終了．
   - 再起動か終了を選択
     - 再起動する場合はパーティション起動 API を呼び出す．
     - 終了する場合はパーティション終了ルーチンからリターン．

10.19.　パーティションの指定

カーネルオブジェクトではいずれかのパーティションに属する必要がある．パーティションの指定は，カーネルオブジェクトを登録する静的 API 等を，そのオブジェクトが属するパーティションの囲みの中に記述する．パーティションに属すべきオブジェクトを登録する静的 API 等を，パーティションの囲みの外に記述した場合には，コンフィギュレータが E_RSATR エラーを報告する．

パーティションの囲みの文は次の通り．

```
PARTITION(パーティション ID) {
        パーティションに属するオブジェクトを登録する静的 API 等
}
```

同じパーティション ID を指定したパーティションの囲みを複数回記述してもよい．

10.20.　パーティション：API

10.20.1. パーティション起動API

【C 言語 API】

ER ercd = StartPartition(ID parid)

【パラメータ】

ID	parid	パーティション ID (有効値 1 - システムに存在するパーティションの最大数)

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_CTX〔s〕	コンテキストエラー
E_OBJ	オブジェクト状態エラー（parid で指定したパーティションが停止状態または終了処理中状態以外）
E_ID	不正 ID 番号（parid が不正）

【機能】

指定した ID のパーティションを起動する．パーティションは初期化中状態となり，初期化処理を開始する．parid には，アプリケーションパーティションのみ指定可能．

呼び出し可能なコンテキストは次の通り．

- パーティション終了処理ルーチンから
  - 自パーティションに対して呼び出し．
- システムパーティションの処理単位
  - 全てのアプリケーションパーティションに対して呼び出し可能．対象のパーティションがどのような状態でも呼び出し可能．この場合，API はリターンし，対象パーティションのタイムウィンドウでパーティション初期化ルーチンが実行される．

10.20.2. パーティション終了API

【C 言語 API】

ER ercd = ShutDownPartition(ID parid)

【パラメータ】

ID	parid	パーティション ID (有効値 1 - システムに存在するパーティションの最大数)

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_CTX〔s〕	コンテキストエラー
E_OBJ	オブジェクト状態エラー（parid で指定したパーティションが通常状態以外）
E_ID	不正 ID 番号（parid が不正）

【機能】

指定した ID のパーティションを終了する．指定されたアプリケーションパーティションは，そのアプリケーションパーティションのタイムウィンドウ内で，パーティ　ションの終了処理（終了処理ルーチン，パーティション終了ルーチンの実行）がなされる．parid には，アプリケーションパーティションのみ指定可能．

呼び出し可能なコンテキストは次の通り．

- アプリケーションパーティションの処理単位
  - 自パーティションに対して呼び出し可能．
- システムパーティションの処理単位
  - 全てのアプリケーションパーティションに対して呼び出し可能．この場合は対象パーティションの状態を終了中状態として，API はリターンする．

10.20.3. パーティションの状態取得API

【C 言語 API】

ER ercd = GetPartitionState(ID parid, T_PSTATE *pk_pstate)

【パラメータ】

ID	parid	パーティション ID (有効値 1 - システムに存在するパーティションの最大数)
T_PSTATE	*pk_pstate	パーティション状態を入れるパケットへのポインタ

【リターンパラメータ】

ER	ercd	エラーコード

*パーティション状態（パケットの内容）

PAR_STATE	par_state	パーティションの状態
uint_t	count	起動回数（初期化中状態から通常状態への遷移の回数）

【エラーコード】

E_ID	不正 ID 番号（parid が不正）

【機能】

指定した ID のパーティションの状態を取得する．取得可能な内容は以下の通り．

- パーティションの状態
- 起動回数

10.20.4. データグループ初期化

【C 言語 API】

ER ercd = InitDataGroup(ID dgid, ATR init_atr)

【パラメータ】

ID	dgid	データグループ ID (有効値 1 - システムに存在するデータグループの最大数)
ATR	init_atr	初期化属性（INIT_DG_VERIFY or INIT_DG_NONE）

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_ID	不正 ID 番号（dgid が不正）
E_PAR	パラメータエラー（init_atr が不正）
E_OACV	アクセス違反

【機能】

指定したデータグループへのアクセス権があれば初期化する．オプションに VERIFY が指定されると初期化後に初期化されたかチェックを行う．

10.20.5. パーティション初期化ルーチン定義API〔S〕

【静的 API】

DEF_PARTITION_INI(ATR iniatr, intptr_t exinf, INIRTN inirtn)

【パラメータ】

ATR	iniatr	パーティション初期化ルーチン属性
intptr_t	exinf	パーティション初期化ルーチン拡張情報
INIRTN	inirtn	パーティション初期化ルーチン先頭番地

【エラーコード】

E_RSATR	予約属性（iniatr が不正または使用できない，アプリケーションパーティションの囲みの外に記述）
E_PAR	パラメータエラー（inirtn が不正）

【機能】

各パラメータで指定したパーティション初期化ルーチンの追加情報に従って，パーティション初期化ルーチンを追加する．

DEF_PARTITION_INI をアプリケーションパーティションの囲みの外に記述した場合には，E_RSATR エラーとなる．

10.20.6. パーティション終了処理ルーチン定義API〔S〕

【静的 API】

DEF_PARTITION_TER(ATR teratr, intptr_t exinf, TERRTN terrtn)

【パラメータ】

ATR	teratr	パーティション終了処理ルーチン属性
intptr_t	exinf	パーティション終了処理ルーチン拡張情報
TERRTN	terrtn	パーティション終了処理ルーチン先頭番地

【エラーコード】

E_RSATR	予約属性（teratr が不正または使用できない，アプリケーションパーティションの囲みの外に記述）
E_PAR	パラメータエラー（terrtn が不正）

【機能】

各パラメータで指定したシステム終了処理ルーチンの追加情報に従って，システム終了処理ルーチンを追加する．

DEF_PARTITION_TER をアプリケーションパーティションの囲みの外に記述した場合には，E_RSATR エラーとなる．

11. パーティションスケジューリング

11.1. 概要

システムには，固定時間のシステム周期が設定される．システム周期内の時間は，複数のタイムウィンドウに分けられる．

個々のタイムウィンドウは，いずれかのパーティションに対応付けられており，タイムウィンドウの時間になると，そのパーティションが実行される．

パーティション内の処理単位は，所属するパーティションに割り当てられたタイムウィンドウ内で実行され，そのパーティションのカーネル状態がどのような状態であっても，タイムウィンドウが終了すると，次のタイムウィンドウに割り当てられたパーティションに切り替わる．

タイムウィンドウの割り当てを細かくすると，OS によるタイムウィンドウの切り替えオーバヘッドが増加するため注意すること（[ParOS_SM] 7.1.4.1）．

11.2. タイムウィンドウ

各タイムウィンドウには，システム周期相対の開始時刻，実行時間，割り当てるパーティションを定義することが可能である．

タイムウィンドウには，コンフィギュレーション時にユニークな ID が割り当てられる．

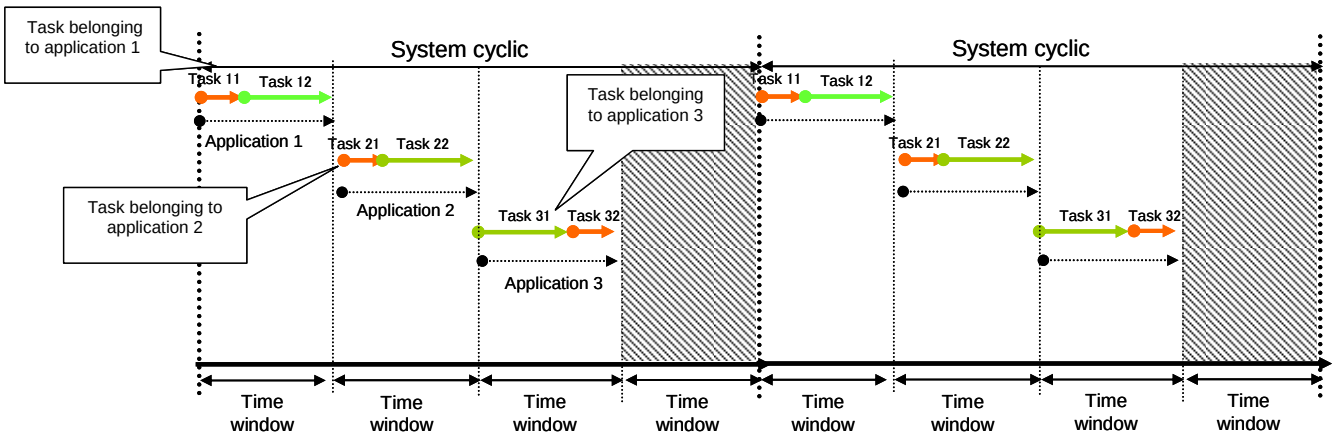

図 11-1. スケジューリング

11.3. スケジューリングモード　[ECC][DCC]

システム通常状態では，システムはモード（スケジューリングモード）を持つ．スケジューリングモードはコンフィギュレーション時に任意の個数定義可能である．モード毎にパーティションのスケジューリング（タイムウィンドウ割当て）を定義することができる．なお，全てのモードでシステム周期は固定である．

スケジューリングモード毎に実行するタイムウィンドウを指定する．時間的に矛盾が無ければ，1 つのスケジューリングモードに複数のタイムウィンドウを割り当てることができる．
スケジューリングモードには，コンフィギュレーション時にユニークな ID が割り当てられる．タイムウィンドウの割当てがない区間はアイドルタイムウィンドウと呼ぶ．

コンフィギュレーション時にディフォルトのスケジューリングモードを指定する必要がある．ディフォルトのスケジューリングモードを指定しなかった場合は，コンフィギュレーション時にエラーとなる．システムの起動後，スケジューリングモードはディフォルトのモードとなる．再起動時等で，ディフォルトモード以外のモードでシステムを開始したい場合は，システム初期化ルーチンで，次のスケジューリングモード変更 API を呼び出せばよい．

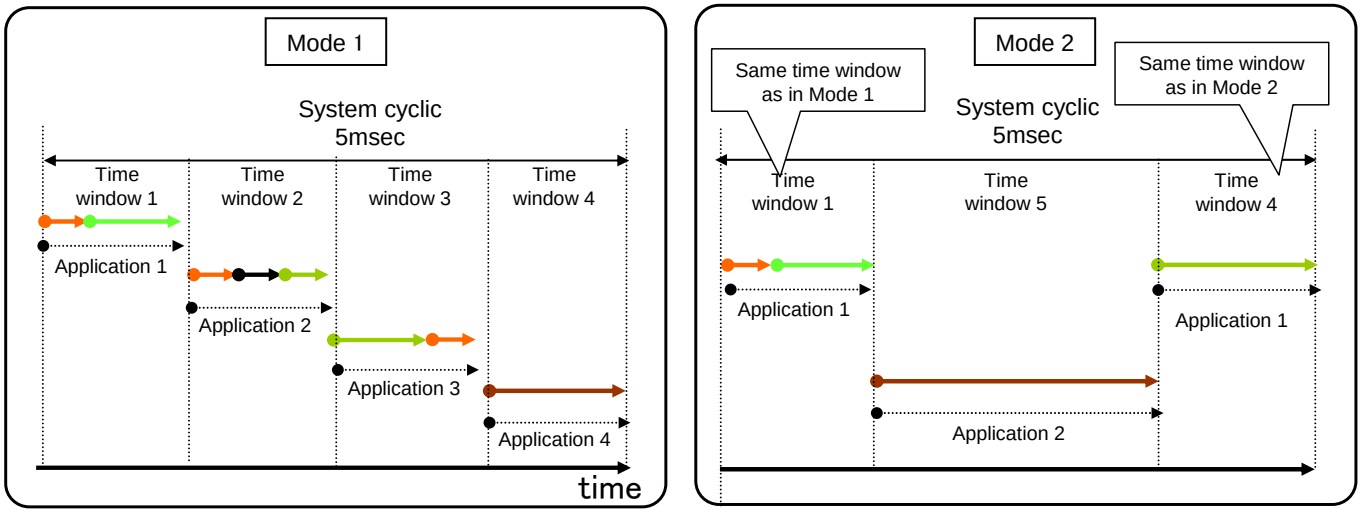

図 11-2. スケジューリングモード

スケジューリングモードはスケジューリングモード変更 API により変更可能である．スケジューリングモード変更 API は，API 呼び出し後に即座にモードを変更し，新しいモードのシステム周期の先頭から実行を開始するか，現在のシステム周期終了後に新しいモードに切り替えるかを指定可能である．

スケジューリングモード変更 API が呼び出されると，スケジューリングモード変更チェックフックが呼び出され，スケジューリングモード変更の正当性をチェックする．チェックをパスすると，スケジューリングモードが指定したモードに変更される．パスしない場合は，エラーが返され，スケジューリングモードは変化しない．

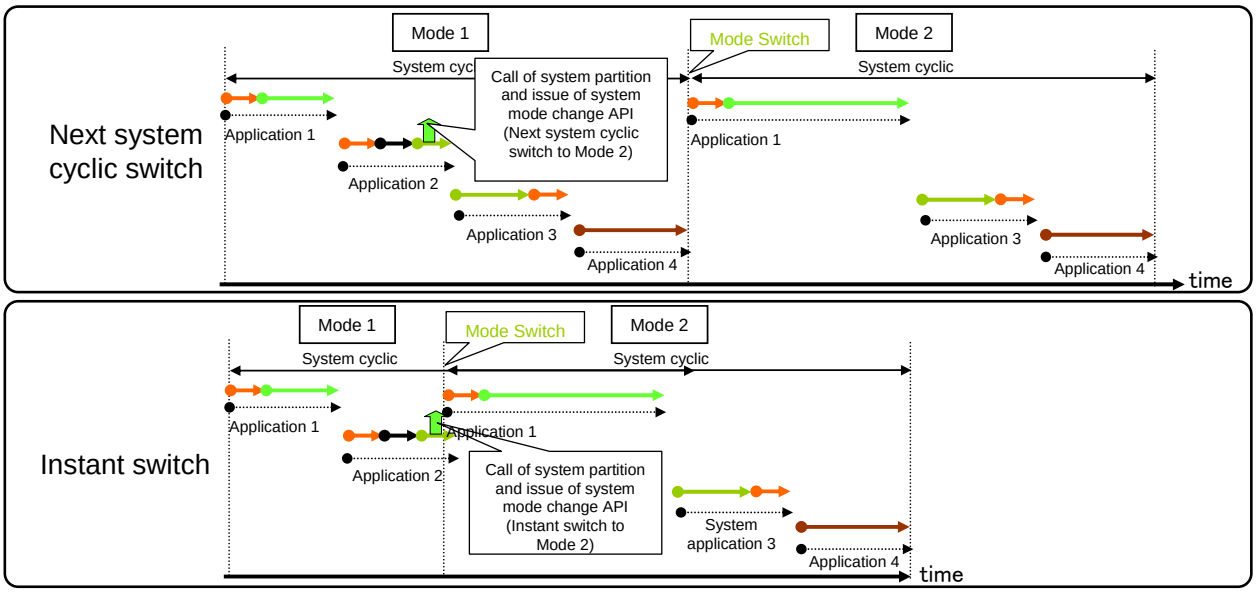

図 11-3. スケジューリングモード切り替え

スケジューリングモードの変更は，"CPU 利用率保護"，"実行順序保護"，"実行タイミング保護"の設定を変更する可能性がある．そのため，パーティションに不正に変更されないように，後述のスケジューリングモードチェックフックにより変更内容をチェックすること．

【仕様決定の理由】

システム周期をスケジューリングモード毎に変更する方法もあるが，ユースケースが考えられないことや，システム割込みハンドラの実行頻度監視をシステム周期のバリエーション毎に定義しなければならないため，本仕様では固定とする．

11.4. パーティションのアイドル属性 [ECC][DCC]

アプリケーションパーティションにはアイドル属性を付加することが可能である．アイドル属性を付加されたパーティションは，実行するべきパーティションが存在しないタイムウィンドウや，パーティションが割り当てられているタイムウィンドウであっても，そのパーティションが休止状態や停止状態の場合に，そのタイムウィンドウで実行される．

複数のパーティションにアイドル属性が付加された場合，どのパーティションがどの程度実行されるかは，実装依存である．また，アイドル属性のパーティションのアプリケーション割込みが受け付けられるかは，実装依存である．

実行するべき処理単位が存在しないパーティション（休止状態）のタイムウィンドウで実行される場合は，元のパーティションの割込みやタイムイベントが発生すると，元のパーティションに処理が移る．

システムパーティションには，アイドル属性を付加することはできない．

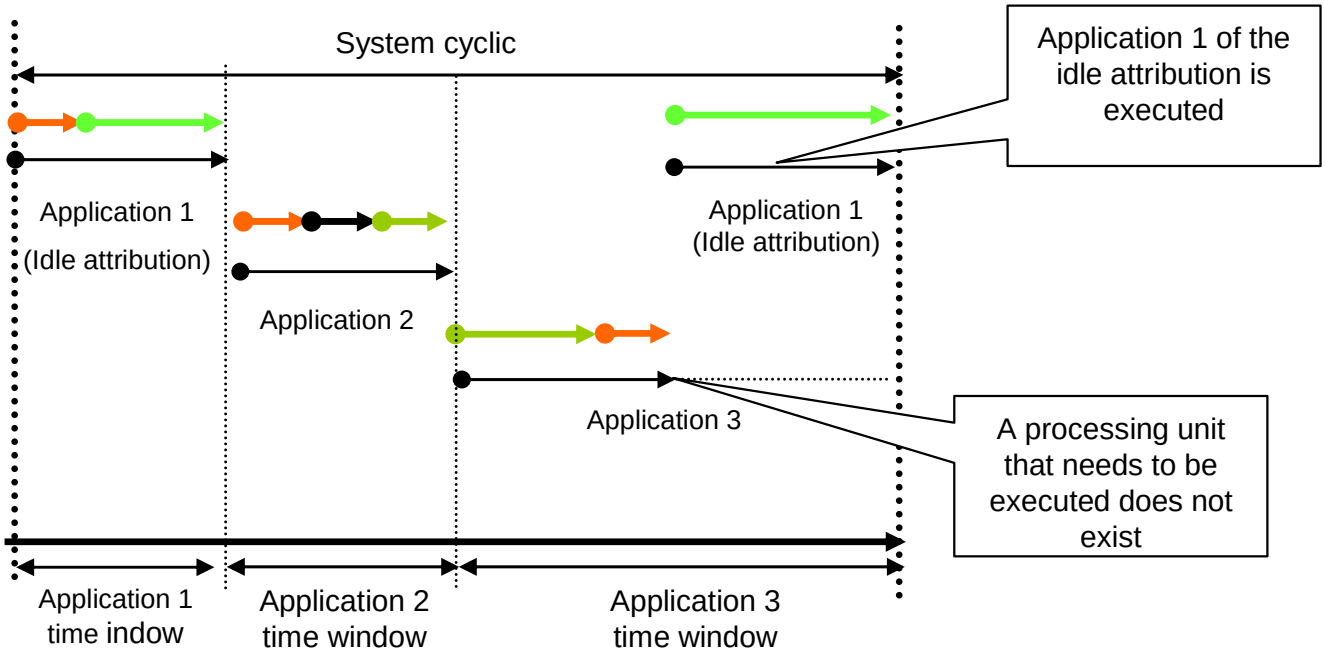

図 11-4. アイドル属性

アイドル属性を付加したパーティションは，"CPU 利用率保護"，"実行順序保護"，"実行タイミング保護"保証されないため注意が必要である（[ParOS_SM] 7.1.4.2）．

11.5. パーティション未満了エラー

システムパーティションの処理単位の実行等により，アプリケーションパーティションに割り当てられたタイムウィンドウを全て実行できず，システム周期終了時に実行状態ないし実行可能状態のパーティションが存在した場合には，システムレベルエラーである，パーティション未満了エラーが発生する．

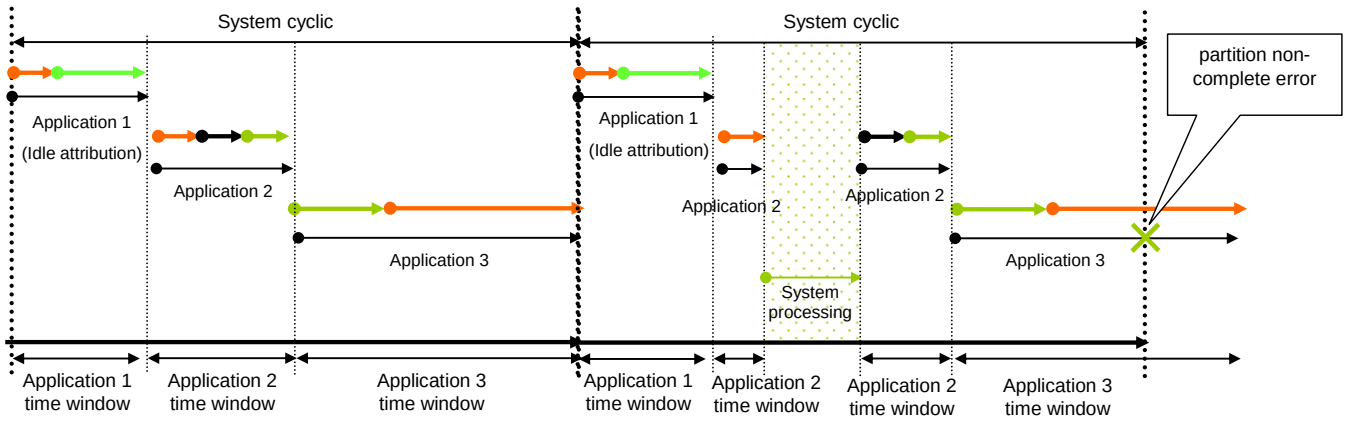

図 11-5. パーティション未満量エラー

11.6. パーティションスケジューリング ：API

11.6.1. スケジューリングモード変更API　[ECC][DCC]

【C 言語 API】

ER ercd = ChangeSchedulingMode(ID schmid, ATR csmatr)

【パラメータ】

ID	schmid	スケジューリングモード ID (有効値 1 - システムに存在するスケジューリングモードの最大数)
ATR	csmatr	切り替えタイミング （CSM_NEXT_CYCLE or CSM_IMMIDIATE）

【リターンパラメータ】

ER	ercd	エラーコード

【エラーコード】

E_ID	不正 ID 番号（symid が不正）
E_PAR	パラメータエラー（csmatr が不正）
E_OACV	アクセス違反（チェックフックがエラーを返した）

【機能】

システムのスケジューリングモードを変更する．csmatr に CSM_NEXT_CYCLE を指定した場合には，次のシステム周期からスケジューリングモードが変更される．

csmatr に CSM_IMMIDIATE を指定した場合は，即座にモードが変更され，変更後のモードのシステム周期の先頭から実行が開始される．

全てのコンテキストから可能だが，アプリケーションパーティションから呼び出した場合はスケジューリングモード変更チェックフックが呼び出される．

11.6.2. システム周期定義API〔S〕

【静的 API】

DEF_SYSTEM_CYCLE(UTIM syscyc)

【パラメータ】

UTIM	syscyc	システム周期

【エラーコード】

E_PAR	パラメータエラー（syscyc が不正）
E_RSATR	予約属性（パーティションの囲みの中に記述，複数記述）

【機能】

システム周期を定義する．本静的 API はパーティションの囲みの外に記述する必要ある．また，定義可能な回数は 1 回である．

11.6.3. タイムウィンドウ定義API〔S〕

【静的 API】

CRE_TWINDOW(ID twid, UTIM start, UTIM dulation)

【パラメータ】

ID	twid	生成するタイムウィンドウの ID
UTIM	start	開始時刻（システム周期相対） (有効値 0 以上)
UTIM	dulation	実行時間 (有効値 1 以上)

【エラーコード】

E_PAR	パラメータエラー（start, dulation が不正）
E_RSATR	予約属性（パーティションの囲みの外に記述）
E_OBJ	オブジェクト状態エラー（twid で指定したタイムウィンドウが登録済み）

【機能】

タイムウィンドウを定義する．定義するタイムウィンドウに属させるパーティションの囲みの中に記述する必要がある．

11.6.4. スケジューリングモード定義API〔S〕[ECC][DCC]

【静的 API】

CRE_SCHMODE(ID schmid, ATR schmatr)

【パラメータ】

ID	schmid	スケジューリングモード ID
ATR	schmatr	属性（ディフォルトスケジューリングモードの場合 SCHM_DEFAULT を指定）

【エラーコード】

E_RSATR	予約属性（symatr が不正または使用できない，パーティションの囲みの中に記述，SCHM_DEFAULT を付加したスケジューリングモードを複数記述）
E_OBJ	オブジェクト状態エラー（schmid で指定したスケジューリングモードが登録済み）

【機能】

スケジューリングモードを定義する．本静的 API はパーティションの囲みの外に記述する必要ある．

11.6.5. タイムウィンドウ割り当て〔S〕

【静的 API】

ATT_TW(ID schmid, ID twid)

【パラメータ】

ID	symid	スケジューリングモード ID (有効値 1 - システムに存在するスケジューリングモードの最大数)
ID	twid	タイムウィンドウ ID (有効値 1 - システムに存在するタイムウィンドウのードの最大数)

【エラーコード】

E_ID	不正 ID 番号(symid, twid が不正)
E_RSATR	予約属性（パーティションの囲みの中に記述）
E_OBJ	オブジェクト状態エラー（登録済みのタイムウィンドウと時間が重なる）

【機能】

スケジューリングモード ID で指定したスケジューリングモードにタイムウィンドウ ID で指定したタイムウィンドウを割り付ける.

本静的 API はパーティションの囲みの外に記述する必要ある.

11.6.6. パーティションの属性指定〔S〕

【静的 API】

SET_PAR_ATTR(ATR paratr)

【パラメータ】

ATR	paratr	パーティション属性

【エラーコード】

E_RSATR	予約属性（パーティションの囲みの外に記述,複数記述）
E_PAR	パラメータ指定

【機能】

パーティションの属性を指定する．指定可能な属性値は以下の通り．

TA_PAR_IDLE：アイドルパーティション　[ECC][DCC]
TA_PAR_STA　：起動時実行属性　[BCC][ECC][DCC]

TA_PAR_IDLE をシステムパーティションに指定した場合には，E_PAR エラーが返る．
SET_PAR_ATTR はアプリケーションパーティションの囲みの外に記述する必要ある．

12. 時間管理

12.1. 概要

ParOS ではシステム周期でパーティションをスケジューリングするため，SafeOS と異なる時間管理を行う．

12.2. システム時刻

SafeOS ではカーネルが管理する時刻であるシステム時刻の単位はミリ秒である．一方，ParOS においては，システム時刻の単位はシステム周期に設定した時間を 1 単位とする．例えば，システム周期 500 マイクロ秒とした場合には，システム時刻の単位は 500 マイクロ秒となる．そして，システム周期が進む毎にシステム時刻が進む．

12.3. システム周期の長さ以下の時間単位

ParOS で追加された機能の時間指定はシステム周期より小さい単位となる．これらは符号無しの整数型である UTIM 型で表し，単位はマイクロ秒とする．UTIM 型で指定する時間には以下のものがある．実際に設定可能な精度は実装依存とする．

- システム周期の時間
- タイムウィンドウの開始時刻
- タイムウィンドウの実行時間
- アトミックハンドラ最悪実行時間
- システム割込みハンドラ最悪実行時間
- 周期ハンドラ最悪実行時間
- アラームハンドラ最悪実行時間

13. 処理単位実行時のプロセッサ動作モード

13.1. 概要

タスクや割込みハンドラの処理単位はプロセッサの特権モードないし非特権モードで動作させる．特権モードとはメモリアクセスの制約がないモードで，非特権モード（ユーザーモード）とはメモリアクセスの制約があるモードである．

特権モードしか持たないプロセッサであっても MPU によりメモリアクセス制約を持たせることができる場合は，便宜上，メモリアクセスの制約がある状態を非特権モード，メモリアクセスの制約がない状態を特権モードと呼ぶ．

13.2. 特権モードで動作する処理単位

システムパーティションに属する処理単位は特権モードで動作する．

13.3. 非特権モードで動作する処理単位

アプリケーションパーティションに属する処理単位は非特権モードで動作する．

14. パーティションメモリ保護

14.1. 概要

パーティションが使用可能なアドレス領域は，メモリオブジェクトを定義して，そのメモリオブジェクトへのアクセス権を許可することにより実現する．

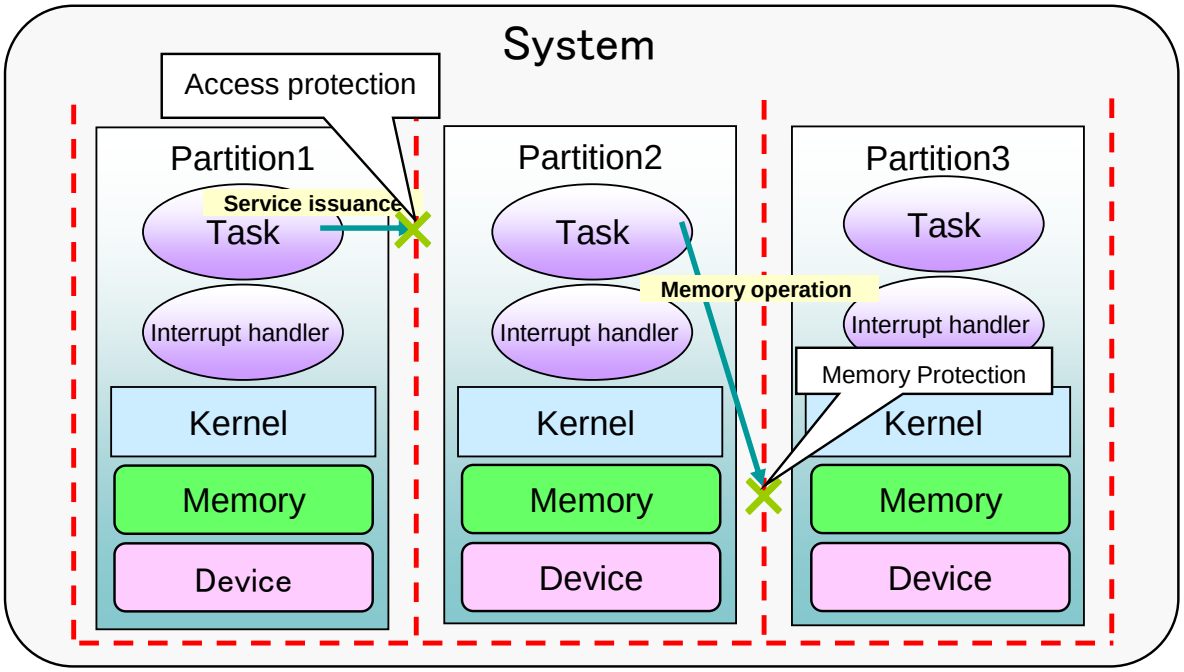

図 14-1. パーティションメモリ保護

14.2. メモリオブジェクト

ParOS ではメモリ領域をカーネルオブジェクトとして扱う．連続したメモリ領域を，メモリオブジェクト（memory object）と呼ぶ．メモリオブジェクトは，互いに重なりあうことはない．

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

メモリオブジェクトはいずれかのパーティションに属する必要がある．所属するパーティションの指定は，コンフィギュレーション時にメモリオブジェクトの生成 API をパーティションの囲みの中に記述すればよい．

所属するパーティションの処理単位のみメモリオブジェクトに対してアクセスすることができる．その他のパーティションの処理単位がアクセスを行うとメモリアクセス違反となる．

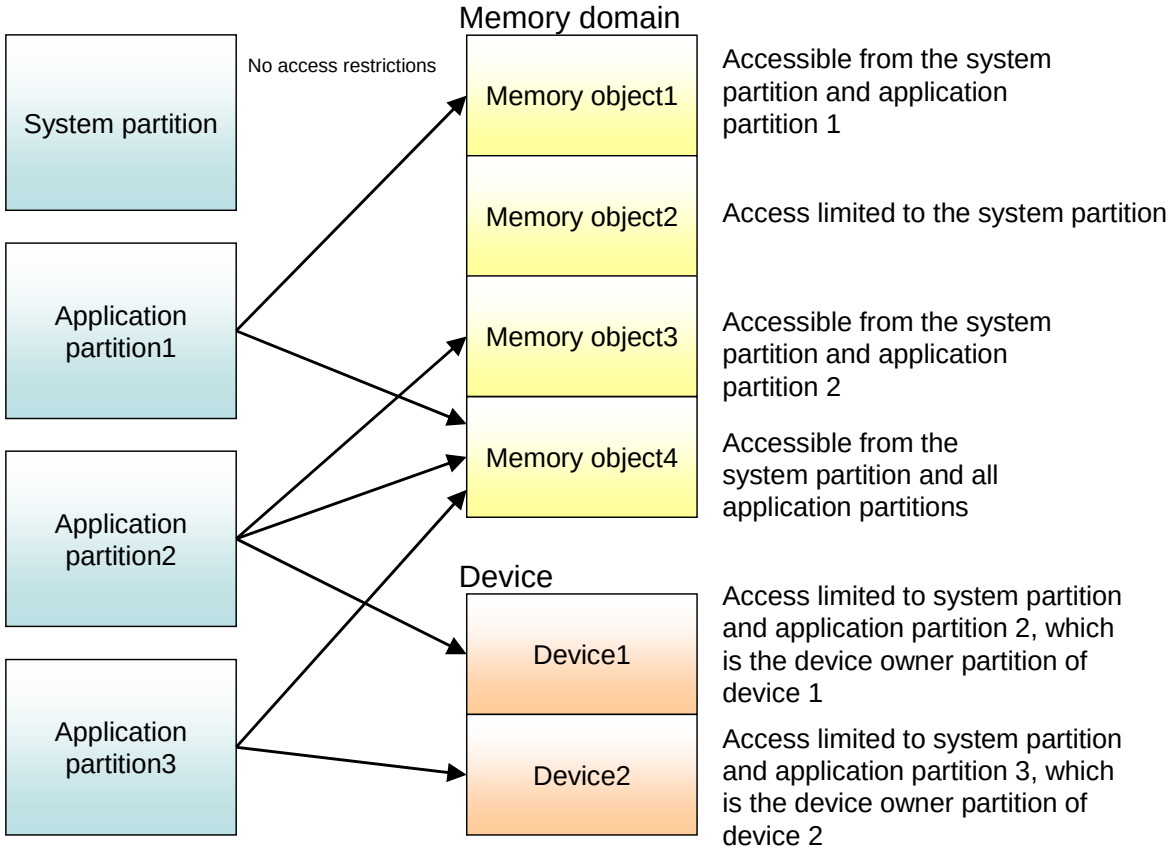

図 14-2. メモリオブジェクト

各メモリオブジェクトが持つ情報は次の通り.

- 先頭番地
- サイズ
- メモリオブジェクト属性
- 属するパーティション

メモリオブジェクト属性には，次の属性を指定することができる.

TA_NOWRITE	0x01U	書込みアクセス禁止
TA_NOREAD	0x02U	読出しアクセス禁止
TA_EXEC	0x04U	実行アクセス許可
TA_MEMINI	0x08U	メモリの初期化を行う
TA_MEMPRSV	0x10U	メモリの初期化を行わない
TA_SDATA	0x20U	ショートデータ領域に配置
TA_UNCACHE	0x40U	キャッシュ禁止
TA_IODEV	0x80U	周辺デバイスの領域

メモリオブジェクトに対して書込みアクセスできるのは，メモリオブジェクト属性に書込みアクセス禁止（TA_NOWRITE 属性）が指定されておらず，アクセス許可ベクタにより書込みアクセスが許可されている場合である．また，読出しアクセスできるのは，メモリオブジェクト属性に読出しアクセス禁止（TA_NOREAD 属性）が指定されておらず，アクセス許可ベクタにより読出し・実行アクセス
が許可されている場合である．実行アクセスできるのは，メモリオブジェクト属性に実行アクセス許可（TA_EXEC 属性）が指定されており，アクセス許可ベクタにより読出し・実行アクセスが許可されている場合である．

ただし，ターゲットハードウェアの制約によってこれらの属性を実現できない場合には，次のように扱われる．書込みアクセス禁止が実現できない場合には，TA_NOWRITE を指定しても無視される．また，読出しアクセス禁止が実現できない場合には，TA_NOREAD を指定しても無視される．実行アクセス禁止が実現できない場合には，TA_EXEC を指定しなくても実行アクセス許可となり，TA_EXEC は無視される．どのような場合にどの属性の指定が無視されるかは，ターゲット定義である．

TA_MEMINI 属性は，システム初期化時に初期化するメモリオブジェクトであることを，TA_MEMPRSV 属性は, システム初期化時に初期化を行わないメモリオブジェクトであることを示す．いずれの属性も指定しない場合，そのメモリオブジェクトは，システム初期化時にクリア（すなわち，0 で初期化）される．TA_MEMINI と TA_MEMPRSV の両方を指定した場合には，E_RSATR エラーとなる．

TA_SDATA 属性は，メモリオブジェクトをショートデータ領域に配置することを示す．具体的な扱いはターゲット定義であるが，ショートデータ領域がサポートされていないターゲットでは，この属性は無視される．また，ターゲットによっては，TA_NOWRITE を指定した場合に，TA_SDATA が無視される場合がある．

TA_UNCACHE 属性は，メモリオブジェクトをキャッシュ禁止に設定することを，TA_IODEV 属性は，メモリオブジェクトを周辺デバイスの領域として扱うことを示す．これらの属性を指定しても意味がないターゲット（例えば，キャッシュを持たないターゲットプロセッサでの TA_UNCACHE）では，これらの属性は無視される．逆に，キャッシュ禁止にできないメモリオブジェクトに対して TA_UNCACHE を指定した場合や，周辺デバイスの領域として扱うことができないメモリオブジェクトに対して TA_IODEV を指定した場合には，E_RSATR エラーとなる．

ターゲットによっては，ターゲット定義のメモリオブジェクト属性を指定できる場合がある．ターゲット定義のメモリオブジェクト属性として，次の属性を予約している．

TA_WTHROUGH	ライトスルーキャッシュを用いる

属性として登録されたアクセス権以外のアクセス，アクセス権が登録されていないメモリオブジェクトへのアクセス及び，メモリオブジェクトとして登録されていないメモリへのアクセスはメモリ違反となる．

システムパーティションは全てのメモリ領域に対してアクセス可能である．

14.3. 共有メモリオブジェクト

共有ライブラリを実現するために全てのパーティションからアクセス可能なメモリオブジェクトを作成する場合には，パーティションの囲みの外でメモリオブジェクトを生成すればよい．

14.4. データグループ

パーティション再起動時のメモリ初期化のためにデータグループを定義することができる．データグループはデータ領域とするメモリオブジェクトを含めることができ，データグループ初期化 API で指定することにより指定したメモリオブジェクトの集合を初期化する．

データグループはいずれのパーティションに所属し，他のパーティションから呼び出された場合にはエラーとなる．

14.5. パーティションメモリ保護 ：API

14.5.1. セクションの登録〔S〕

【静的 API】

ATT_SEC("セクション名", ATR mematr, "メモリリージョン名")

【パラメータ】

"セクション名"	登録するセクションを指定する文字列
ATR　　mematr	メモリオブジェクト属性
"メモリリージョン名"	セクションを配置するメモリリージョンを指定する文字列

【エラーコード】

E_RSATR	予約属性（mematr が不正または使用できない）
E_PAR	パラメータエラー（セクション名，メモリリージョン名が不正）
E_OBJ	オブジェクト状態エラー（登録済みのセクションの再登録）

【機能】

各パラメータで指定した情報に従って，指定したセクションをカーネルに登録する．具体的な振舞いは以下の通り．

各オブジェクトモジュール中に含まれるセクション名で指定したセクションが，メモリリージョン名で指定したメモリリージョンに配置され，メモリオブジェクトとして登録される．登録されるメモリオブジェクトには，mematr で指定したメモリオブジェクト属性が設定される．

指定したメモリリージョンが，書込みを行えないメモリリージョンである場合，メモリオブジェクト属性の内，TA_NOWRITE，TA_MEMINI，TA_MEMPRSV は無視される（指定しても指定しなくても，同じ振舞いとなる）．

ATT_MOD／ATA_MOD がサポートされているターゲットでは，セクション名として，標準のセクションを指定することはできない．指定した場合には，E_PAR エラーとなる．

14.5.2. オブジェクトモジュールの登録〔S〕

【静的 API】

ATT_MOD("オブジェクトモジュール名", ATR mematr)

【パラメータ】

"オブジェクトモジュール名"		登録するオブジェクトモジュールを指定する文字列
ATR	mematr	メモリオブジェクト属性

【エラーコード】

E_RSATR	予約属性（mematr が不正または使用できない）
E_NOSPT	未サポート機能（ATT_MOD／ATA_MOD がサポートされていない）
E_OBJ	オブジェクト状態エラー（登録済みのオブジェクトモジュールの再登録）

【機能】

各パラメータで指定した情報に従って，指定したオブジェクトモジュールをカーネルに登録する．具体的な振舞いは以下の通り．

オブジェクトモジュール名で指定したオブジェクトモジュール中に含まれる標準のセクションが，それぞれ，ターゲット定義でセクション毎に定まるメモリリージョンに配置され，メモリオブジェクトとして登録される．登録されるメモリオブジェクトには，ターゲット定義でセクション毎に定まるメモリオブジェクト属性が設定される．

ターゲット定義で，ATA_MOD により登録できるオブジェクトモジュールが属する保護ドメインや登録できる数に制限がある場合がある．

14.5.3. メモリオブジェクトの登録〔S〕

【静的 API】

ATT_MEM(ATR mematr, void *base, SIZE size)

【パラメータ】

ATR	mematr	メモリオブジェクト属性
void *	base	登録するメモリ領域の先頭番地 (有効値 0 以上)
SIZE	size	登録するメモリ領域のサイズ（バイト数） (有効値 1 以上)

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_RSATR	予約属性（mematr が不正または使用できない)
E_NOSPT	未サポート機能（条件はターゲット定義）
E_PAR	パラメータエラー（base，size が不正）
E_OBJ	オブジェクト状態エラー（登録済みのメモリオブジェクトとメモリ領域が重なる）

【機能】

各パラメータで指定したメモリオブジェクト登録情報に従って，メモリオブジェクトを登録する．具体的な振舞いは以下の通り．

base と size で指定したメモリ領域が，メモリオブジェクトとして登録される．登録されるメモリオブジェクトには，mematr で指定したメモリオブジェクト属性が設定される．

mematr に，TA_MEMINI と TA_MEMPRSV を指定することはできない．指定した場合には，E_RSATR エラーとなる．

ターゲット定義で，この機能により登録できるメモリオブジェクトが属する保護ドメインや登録できる数に制限がある場合がある．

base や size に，ターゲット定義の制約に合致しない先頭番地やサイズを指定した時には，E_PAR エラーとなる．また，size が 0 の場合には，E_PAR エラーとなる．登録しようとしたメモリオブジェクトが，登録済みのメモリオブジェクトとメモリ領域が重なる場合には，E_OBJ エラーとなる．

【使用上の注意】

ATT_MEM／ATA_MEM は，ユーザタスクからメモリ空間にマッピングされた I/O 領域にアクセスできるようにするために使用することを想定した静的 API である．メモリ領域に対しては，ATT_SEC／ATA_SEC か ATT_MOD／ATA_MOD を使用することを推奨する．

14.5.4. データグループの生成〔S〕

【静的 API】

ER ercd = CRE_DTG(ID dtgid)

【パラメータ】

ID	dtqid	データグループ ID

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_RSATR	予約属性（パーティションの囲みの外に書かれている）

【機能】

データグループを生成する．パーティションの囲みの外に書かれている場合には，E_RSATR エラーが返る．

14.5.5. メモリオブジェクトのデータグループへの追加〔S〕

【静的 API】

ER ercd = ATT_DTG(ID dtgid, void *base)

【パラメータ】

ID	dtgid	データグループの ID
void	*base	追加するメモリオブジェクトの先頭番地 (有効値 0 以上)

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正 ID 番号(dtgid が不正)
E_PAR	パラメータエラー（size が不正，パーティションの囲みの外に記述）

【機能】

メモリオブジェクトをデータグループへ追加する．dtgid に存在しないデータグループを追加するとエラーとなる．

ATT_DTG は，パーティションの囲みの中に記述する必要がある．

15. SafeOS互換機能

15.1. 概要

各パーティションは，SafeOS 互換の機能を呼び出すことが可能である．

一部の機能は，ParOS で追加された機能で代替可能であるため使用できない．ParOS により代替機能が提供されているものはその機能を用いること（[ParOS_SM] 7.1.5）．

15.2. 使用できない機能と代替機能

SafeOS のもつ以下の機能を使用することはできない，ParOS で導入される代替機能を用いること．

- 割込みハンドラ・割込みサービスルーチン
  - システム割込みハンドラ
  - アプリケーション割込みハンドラ
- タイムイベントハンドラ
  - システムタイムイベントハンドラ
  - タイムウィンドウハンドラ
- CPU 例外管理機能
  - 例外ハンドラ

15.3. タスク生成APIの保護対応カーネル対応版

SafeOS のタスクを生成する静的 API は保護機能対応でないカーネルの場合の引数になっているが，ParOS ではメモリ保護を持つため，保護対応カーネルの場合の引数となる．

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

15.4. 他のパーティションのオブジェクトアクセス違反

ParOS では他のパーティションのカーネルオブジェクトに対する API 呼び出しが許可されていない．他のパーティションのカーネルオブジェクトを指定して API を呼び出した場合には，オブジェクトアクセス違反（E_OACV）が返る．

16. パーティション間通信（COM）

16.1. 概要

パーティション間通信は，パーティション間でデータをやりとりするための仕組みである．

通信路本体をチャネルとして宣言し，パーティション毎にチャネルに接続するためのインタフェースを宣言する．インタフェースとチャネルの接続はシステム構成時に指定する．チャネルとインタフェースには，それぞれ ID 番号が付加される．

チャネルはいずれかのパーティションに属する．チャネルの属するパーティションをチャネルのオーナーと呼ぶ．

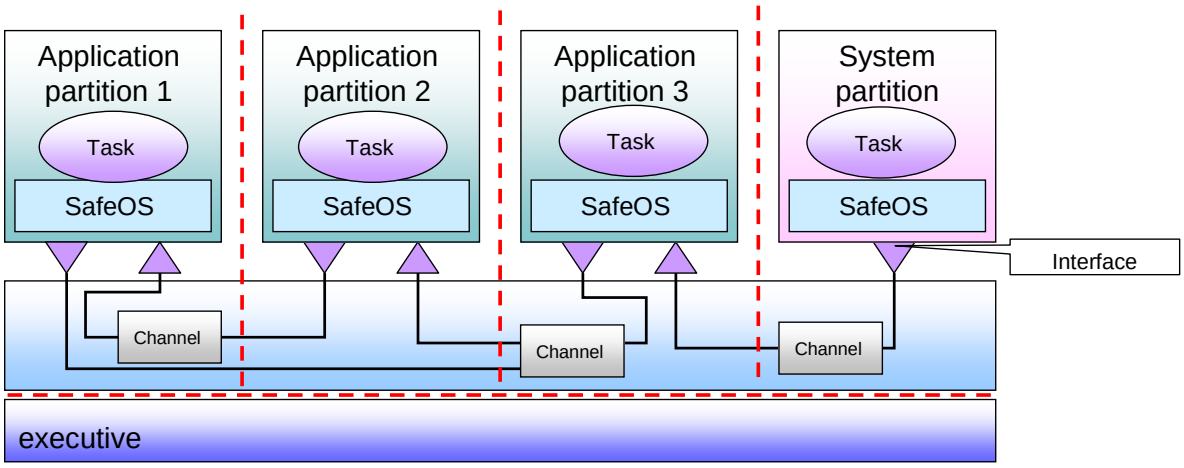

図 16-1. パーティション間通信

パーティション間通信の使用時の注意点については，[ParOS_SM]の 7.2 を参照のこと．

16.2. メッセージキューチャネル

メッセージキューチャネルは，キューイング可能な固定長メッセージをやり取りするためのチャネルである．メッセージサイズとバッファ（キューイング可能なメッセージ数）はコンフィギュレーション時に指定する．

メッセージはキューイングされるため，書き込み時にバッファがフルの場合，読み込み時にバッファがエンプティブの場合，タスクは待ち状態となる．待ち状態のタスクは rel_wai() により強制待ち解除が可能である．

n 対 n 通信が可能である．受信側はどのインタフェース（パーティション）から送られたメッセージか判断するため，受信 API には，そのメッセージを送信したパーティションの ID が付加される．

読み込み・書き込み API にはタイムアウトが指定可能である．タイムアウト値を超えて待ち状態となった場合は，待ち状態が解除されてエラーが返る．

メッセージキューチャネルは停止状態と通常状態 2 種類の状態を持つ．停止状態のチャネルに対するアクセスはエラーとなる．通常状態ではインタフェースを介したアクセスが可能．

状態はチャネルが所属するパーティションないしシステムパーティションから開始・停止 API により操作可能である．システム起動時には停止状態となっている．

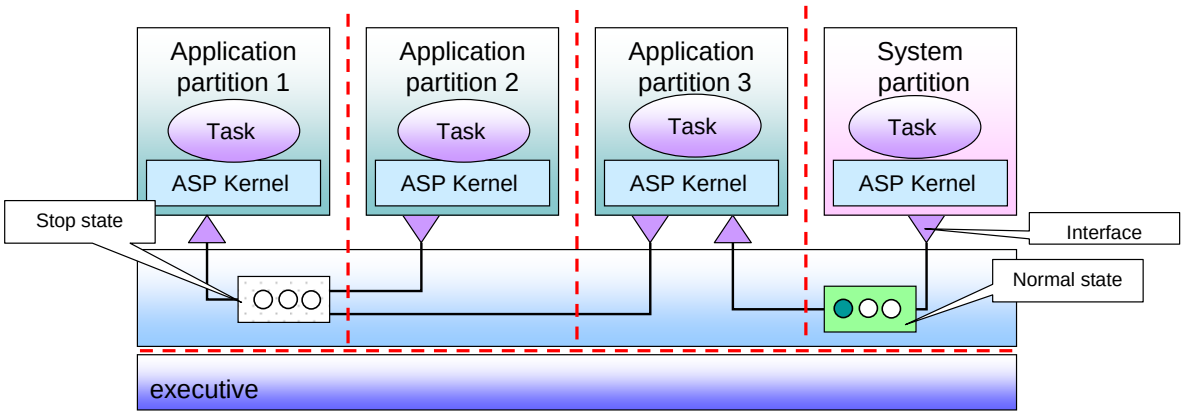

図 16-2. メッセージキューチャネル

16.3. 状態変数チャネル

状態変数チャネルは，キューイングしない固定長メッセージをやり取りするためのチャネルである．メッセージサイズはコンフィギュレーション時にバイトサイズで指定する．

メッセージはキューイングされず，最新のデータが常に読み込める．そのため，読み込み書き込み共にノンブロックである．

n 対 n 通信が可能である．読み込みまたは書き込み可能なパーティションはコンフィギュレーション時にインタフェースの属性で指定する．

状態変数チャネルは停止状態と通常状態 2 種類の状態を持つ．停止状態のチャネルに対するアクセスはエラーとなる．通常状態ではインタフェースを介したアクセスが可能．

状態はチャネルが所属するパーティションないしシステムパーティションから開始･停止 API により操作可能である．システム起動時には停止状態となっている．また，状態はチャネルが所属するパーティションないしシステムパーティションから書き込み API により書き込んだ場合にも通常状態とする．

更新監視が可能である．具体的には，生成時に最長更新間隔時間を指定可能．指定された時間の間に書き込みがなされなかった場合には，チャネルの状態を停止状態とする．また，オーナパーティションに状態変数更新エラーを発生させる．

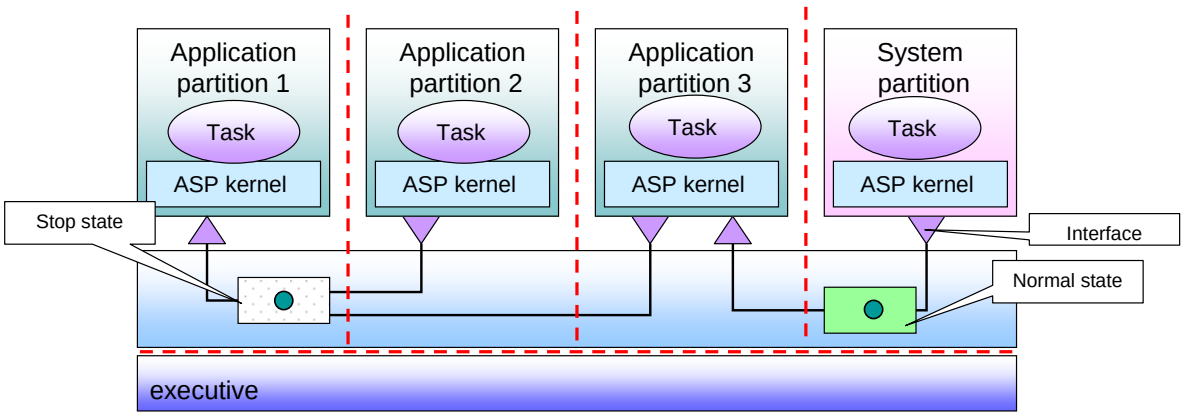

図 16-3. 状態変数チャネル

16.4. パーティション間通信（COM）: API

16.4.1. メッセージキューチャネル生成〔S〕

【静的 API】

CRE_MSGQ(ID msgqid, ATR msgqatr, uint_t msgsize, uint_t msgcnt)

【パラメータ】

ID	msgqid	メッセージキューチャネル ID
ATR	msgqatr	メッセージキューチャネル属性
uint_t	msgsize	メッセージサイズ（バイト） (有効値 1 以上)
uint_t	msgcnt	メッセージ数（キューの数） (有効値 1 以上)

【リターンパラメータ】

ER_ID	msgqid	生成されたデータキューの ID 番号（正の値） またはエラーコード

【エラーコード】

E_RSATR	予約属性（msgatr が不正または使用できない，属するパーティションが不正）
E_NOSPT	未サポート機能（dtqmb がサポートされていない値）
E_NOMEM	メモリ不足（管理領域が確保できない）
E_OBJ	オブジェクト状態エラー（msgqid で指定したメッセージキューチャネルが登録済み）

【機能】

各パラメータで指定したメッセージキューチャネル生成情報に従って，メッセージキューチャネルを生成する．msgsize と msgcnt からデータキューが設定され，格納されているデータがない状態に初期化される．また，送信待ち行列と受信待ち行列は，空の状態に初期化される．

本 API はパーティションの囲みの中に記述する必要があり，指定したパーティションがオーナパーティションとなる．

16.4.2. 状態変数チャネル生成〔S〕

【静的 API】

CRE_STVA(ID stvaid, ATR stvaatr, uint_t varsize)

【パラメータ】

ID	stvaid	状態変数チャネル ID
ATR	stvaatr	状態変数チャネル属性
uint_t	varsize	変数サイズ（バイト） (有効値 1 以上)
RELTIM	updatetim	最長更新間隔時間 (有効値 1 以上)

【リターンパラメータ】

ER_ID	stvaid	生成されたデータキューの ID 番号（正の値） またはエラーコード

【エラーコード】

E_RSATR	予約属性（stvaatr が不正または使用できない，属するパーティションが不正）
E_NOSPT	未サポート機能（varsize がサポートされていない値）
E_NOMEM	メモリ不足（管理領域が確保できない）
E_OBJ	オブジェクト状態エラー（stvaid で指定した状態変数チャネルが登録済み：CRE_DTQ の場合

【機能】

各パラメータで指定した状態変数チャネル生成情報に従って，状態変数チャネルを生成する．varsize から変数のサイズが設定される．

updatetim を超えて内容が更新されない場にはチャネルの状態が停止状態となる．

本 API はパーティションの囲みの中に記述する必要があり，指定したパーティションがオーナパーティションとなる．

16.4.3. インタフェース生成〔S〕

【静的 API】

CRE_INF(ID ifid, ATR ifatr)

【パラメータ】

ID	ifid	インタフェース ID
ATR	ifiatr	インタフェース属性

【リターンパラメータ】

ER_ID	ifid	生成されたデータキューの ID 番号（正の値） またはエラーコード

【エラーコード】

E_RSATR	予約属性（ifatr が不正または使用できない，属するパーティションが不正）
E_OBJ	オブジェクト状態エラー（ifid で指定した状態変数チャネルが登録済み：CRE_DTQ の場合

【機能】

各パラメータで指定したインタフェース生成情報に従って，インタフェースを生成する．
インタフェース属性には，次の属性を指定することが可能である．

TA_IN	0x01U	入力インタフェース
TR_OUT	0x02U	出力インタフェース

本 API はパーティションの囲みの中に記述する必要があり，指定したパーティションに所属する．

16.4.4. メッセージキューチャネル・インタフェース接続定義〔S〕

【静的 API】

ATT_IF_MSGQ(ID msgqid ID ifid)

【パラメータ】

ID	msgqid	メッセージキューチャネル ID (有効値 1 - システムに存在するメッセージキューチャネルの最大数)
ID	ifid	インタフェース ID (有効値 1 - システムに存在するインタフェースの最大数)

【リターンパラメータ】

ER_ID	ifid	生成されたデータキューの ID 番号（正の値） またはエラーコード

【エラーコード】

E_RSATR	予約属性（パーティションの囲みの中に記述）
E_ID	不正 ID 番号(msgqid, ifid が不正)

【機能】

チャネルとインタフェースの接続を指定する.

本静的 API はパーティションの囲みの外に記述する必要ある.

16.4.5. 状態変数チャネル・インタフェース接続定義〔S〕

【静的 API】

ATT_IF_STVA(ID stvaid, ID ifid)

【パラメータ】

ID	stvaid	情報変数チャネル ID (有効値 1 - システムに存在する情報変数チャネルの最大数)
ID	ifid	インタフェース ID (有効値 1 - システムに存在するインタフェースの最大数)

【リターンパラメータ】

ER_ID	ifid	生成されたデータキューの ID 番号（正の値） またはエラーコード

【エラーコード】

E_RSATR	予約属性（パーティションの囲みの中に記述）
E_ID	不正 ID 番号(stvaid ifid が不正)

【機能】

チャネルとインタフェースの接続を指定する.

本静的 API はパーティションの囲みの外に記述する必要ある.

16.4.6. メッセージキューチャネル開始

【C 言語 API】

ER　ercd = StartMessageQueue(ID msgqid)

【パラメータ】

ID	msgqid	メッセージキューチャネル ID (有効値 1 - システムに存在するメッセージキューチャネルの最大数)

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正 ID 番号（msgqid が不正）
E_OBJ	オブジェクト状態エラー（対象チャネルが通常状態）
E_OACV	オブジェクトアクセス違反（対象チャネルに対する操作が許可されていない）

【機能】

msgqid で指定したメッセージキューチャネル（対象チャネル）を通常状態とする．操作可能なのは，対象チャネルのオーナーないしシステムパーティションのみである．

16.4.7. メッセージキューチャネル停止

【C 言語 API】

ER ercd = StopMessageQueue(ID msgqid)

【パラメータ】

ID	msgqid	メッセージキューチャネル ID (有効値 1 - システムに存在するメッセージキューチャネルの最大数)

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正 ID 番号（msgqid が不正）
E_OBJ	オブジェクト状態エラー（対象チャネルが停止状態）
E_OACV	オブジェクトアクセス違反（対象チャネルに対する操作が許可されていない）

【機能】

msgqid で指定したメッセージキューチャネル（対象チャネル）を停止状態とする．操作可能なのは，対象チャネルのオーナーないしシステムパーティションのみである．

バッファ（メッセージキュー）を全てクリアして待ち状態のタスクにエラー通知する．

16.4.8. メッセージキューチャネル書き込み

- メッセージキューチャネル書き込み(タイムアウトなし)
- メッセージキューチャネル書き込み(ポーリング)
- メッセージキューチャネル書き込み(タイムアウト付き)

【C 言語 API】

```
ER ercd = SendMessageQueue(ID ifid, intptr_t *p_data)
ER ercd = PSendMessageQueue(ID ifid, intptr_t *p_data)
ER ercd = TSendMessageQueue(ID ifid, intptr_t *p_data, TMO tmout)
```

【パラメータ】

ID	ifid	インタフェース ID (有効値 1 - システムに存在するインタフェースの最大数)
intptr_t *	p_data	送信データを入れているメモリ領域へのポインタ
TMO	tmout	タイムアウト時間

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_CTX	コンテキストエラー
E_ID	不正 ID 番号（ifid が不正）
E_PAR	パラメータエラー（tmout が不正）
E_OACV	オブジェクトアクセス違反（対象メッセージキューに対する操作が許可されていない）
E_MACV	メモリアクセス違反（p_data が指すメモリ領域への書込みアクセスが許可されていない）
E_TMOUT	ポーリング失敗またはタイムアウト
E_RLWAI	待ち状態の強制解除
E_OBJ	オブジェクト状態エラー（対象メッセージキューが停止状態）

【機能】

ifid で指定したインタフェースを経由してメッセージキューチャネル（対象メッセージキューチャネル）に，p_data で指定したメモリ領域のデータを送信する．

対象メッセージキューチャネルが停止状態の場合には，E_OBJ が返されエラーとなる．対象メッセージキューチャネルが通常状態の場合の具体的な振舞いは以下の通り．

対象メッセージキューチャネルの受信待ち行列にタスクが存在する場合には，受信待ち行列の先頭のタスクが，p_data で指定したデータを受信し，待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_OK が返る．

対象メッセージキューチャネルの受信待ち行列にタスクが存在せず，データキューにデータを格納するスペースがある場合には，data で指定したデータが，FIFO 順でデータキューに格納される．

対象メッセージキューチャネルの受信待ち行列にタスクが存在せず，データキューにデータを格納するスペースがない場合には，自タスクはメッセージキューチャネルへの送信待ち状態となり，対象メッセージキューチャネルの送信待ち行列につながれる．対象メッセージキューチャネルが停止状態になった場合は，エラーとなり待ち状態が解除され，E_RELWAI が返る．

16.4.9. メッセージキューチャネル読み込み

- メッセージキューチャネル読み込み(タイムアウトなし)
- メッセージキューチャネル読み込み(ポーリング)
- メッセージキューチャネル読み込み(タイムアウト付き)

【C 言語 API】

ER ercd = ReciveMessageQueue(ID ifid, ID *p_parid, intptr_t *p_data)
ER ercd = PReciveMessageQueue(ID ifid, ID *p_parid, intptr_t *p_data)
ER ercd = TReciveMessageQueue(ID ifid, ID *p_parid, intptr_t *p_data, TMO tmout)

【パラメータ】

ID	ifid	インタフェース ID (有効値 1 - システムに存在するインタフェースの最大数)
ID	*p_parid	受信データの送信パーティション ID を入れるメモリ領域へのポインタ
intptr_t *	p_data	受信データを入れるメモリ領域へのポインタ
TMO	tmout	タイムアウト時間

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード
ID	*p_parid	受信データの送信パーティション ID
intptr_t	*p_data	受信データ

【エラーコード】

E_CTX	コンテキストエラー
E_ID	不正 ID 番号（ifid が不正）
E_PAR	パラメータエラー（tmout が不正）
E_OACV	オブジェクトアクセス違反（対象メッセージキューに対する操作が許可されていない）
E_MACV	メモリアクセス違反（p_data が指すメモリ領域への書込みアクセスが許可されていない）
E_TMOUT	ポーリング失敗またはタイムアウト
E_RLWAI	待ち状態の強制解除
E_OBJ	オブジェクト状態エラー（対象メッセージキューが停止状態）

【機能】

ifid で指定したインタフェースを経由してメッセージキューチャネル（対象メッセージキューチャネル）からデータを受信する．受信したデータは，p_data で指定したメモリ領域に返される．

対象メッセージキューチャネルが停止状態の場合には，E_OBJ が返されエラーとなる．対象メッセージキューチャネルが通常状態の場合の具体的な振舞いは以下の通り．

対象メッセージキューチャネルのデータキューにデータが格納されている場合には，データキューの先頭に格納されたデータが取り出され，p_data で指定したメモリ領域に返される．また，送信待ち行列にタスクが存在する場合には，送信待ち行列の先頭のタスクの送信データが，FIFO 順でデータキューに格納され，そのタスクは待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_OK が返る．

対象メッセージキューチャネルのデータキューにデータが格納されておらず，送信待ち行列にタスクが存在する場合には，送信待ち行列の先頭のタスクの送信データが，p_data で指定したメモリ領域に返される．送信待ち行列の先頭のタスクは，待ち解除される．待ち解除されたタスクには，待ち状態となったサービスコールから E_OK が返る．

対象メッセージキューチャネルのデータキューにデータが格納されておらず，送信待ち行列にタスクが存在しない場合には，自タスクはメッセージキューチャネルからの受信待ち状態となり，対象メッセージキューチャネルの受信待ち行列につながれる．対象メッセージキューチャネルが停止状態になった場合は，エラーとなり待ち状態が解除され，E_RELWAI が返る．

16.4.10. メッセージキューチャネルの状態参照

【C 言語 API】

ER ercd = RefMessageQueue(ID msgqid, T_RMSGQ *pk_rmsgq)

【パラメータ】

ID	msgqid	対象メッセージキューチャネルの ID (有効値 1 - システムに存在するメッセージキューチャネルの最大数)
T_RBSGQ	*pk_rmsgq	メッセージキューチャネルの現在情報を入れるパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

*メッセージキューチャネルの現在状態（パケットの内容）

STAT	chstat	対象メッセージキューチャネルの状態
ID	stskid	送信待ち行列の先頭のタスクの ID 番号
ID	rtskid	受信待ち行列の先頭のタスクの ID 番号
uint_t	sdtqcnt	データキューに格納されているデータの数

【エラーコード】

E_ID	不正 ID 番号（msgqid が不正）
E_MACV〔P〕	メモリアクセス違反（pk_rmsgq が指すメモリ領域への書込みアクセスが許可されていない）

【機能】

msgqid で指定したメッセージキューチャネル（対象メッセージキューチャネル）の現在状態を参照する．参照した現在状態は pk_rmsgq で指定したパケットに返される．

対象メッセージキューチャネルの送信待ち行列にタスクが存在しない場合，stskid には TSK_NONE（＝0）が返る．また，受信待ち行列にタスクが存在しない場合，rtskid には TSK_NONE（＝0）が返る．

16.4.11. 状態変数チャネル開始

【C 言語 API】

ER　ercd = StartStateVariable(ID stvaid)

【パラメータ】

ID	stvaid	状態変数チャネル ID (有効値 1 - システムに存在する情報変数チャネルの最大数)

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正 ID 番号（stvaid が不正）
E_OBJ	オブジェクト状態エラー（対象チャネルが通常状態）
E_OACV	オブジェクトアクセス違反（対象チャネルに対する操作が許可されていない）

【機能】

stvaid で指定した状態変数チャネル（対象チャネル）を通常状態とする．操作可能なのは，対象チャネルのオーナーないしシステムパーティションのみである．

16.4.12. 状態変数チャネル停止

【C 言語 API】

ER ercd = StopStateVariable(ID stvaid)

【パラメータ】

ID	msgqid	情報変数チャネル ID (有効値 1 - システムに存在する状態変数チャネルの最大数)

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正 ID 番号（stvaid が不正）
E_OBJ	オブジェクト状態エラー（対象チャネルが停止状態）
E_OACV	オブジェクトアクセス違反（対象チャネルに対する操作が許可されていない）

【機能】

stvaid で指定した状態変数チャネル（対象チャネル）を停止状態とする．操作可能なのは，対象チャネルのオーナーないしシステムパーティションのみである．

16.4.13. 状態変数チャネル書き込み

【C 言語 API】

ER ercd = WriteStateVariable(ID ifid, intptr_t *p_data)

【パラメータ】

ID	ifid	インタフェース ID (有効値 1 - システムに存在するインタフェースの最大数)
intptr_t *	p_data	書き込みデータを入れているメモリ領域へのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_ID	不正 ID 番号（ifid が不正）
E_OACV	オブジェクトアクセス違反（対象状態変数に対する操作が許可されていない）
E_MACV	メモリアクセス違反（p_data が指すメモリ領域への読み込みアクセスが許可されていない）
E_OBJ	オブジェクト状態エラー（対象メッセージキューが停止状態）

【機能】

ifid で指定したインタフェースを経由して状態変数チャネル（対象状態変数チャネル）に，p_data で指定したメモリ領域のデータを書き込む．

対象状態変数チャネルが停止状態の場合には，E_OBJ が返されエラーとなる．

16.4.14. 状態変数チャネル読み込み

【C 言語 API】

ER ercd = ReadStateVariable(ID ifid, intptr_t *p_data)

【パラメータ】

ID	ifid	インタフェース ID (有効値 1 - システムに存在するインタフェースの最大数)
intptr_t *	p_data	読み込みデータを入れているメモリ領域へのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード
intptr_t	*p_data	受信データ

【エラーコード】

E_ID	不正 ID 番号（ifid が不正）
E_OACV	オブジェクトアクセス違反（対象状態変数に対する操作が許可されていない）
E_MACV	メモリアクセス違反（p_data が指すメモリ領域への書込みアクセスが許可されていない）
E_OBJ	オブジェクト状態エラー（対象メッセージキューが停止状態）

【機能】

ifid で指定したインタフェースを経由して状態変数チャネル（対象状態変数チャネル）からデータを読み込む．読み込んだデータは，p_data で指定したメモリ領域に書き込まれる．

対象状態変数チャネルが停止状態の場合には，E_OBJ が返されエラーとなる．

16.4.15. 状態変数チャネル状態参照

【C 言語 API】

ER ercd = RefStateVariable(ID stvaid, T_RSTVA *pk_rstva)

【パラメータ】

ID	stvaid	対象状態変数チャネルの ID (有効値 1 - システムに存在する状態変数チャネルの最大数)
T_RBSGQ	*pk_rstva	状態変数の現在情報を入れるパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

*メッセージキューチャネルの現在状態（パケットの内容）

STAT	chstat	対象状態変数チャネルの状態

【エラーコード】

E_ID	不正 ID 番号（stvaid が不正）
E_MACV〔P〕	メモリアクセス違反（pk_rstva が指すメモリ領域への書込みアクセスが許可されていない）

【機能】

stvaid で指定した状態変数チャネル（対象状態変数チャネル）の現在状態を参照する．参照した現在状態は pk_rstva で指定したパケットに返される．

17. アトミックハンドラ

17.1. 概要

アトミックハンドラは，アプリケーションパーティションにおいて一連の処理を中断されることなく実行することが可能なハンドラである．

アプリケーションパーティションの処理単位は，割り当てられたタイムウィンドウが終了した場合や，システム割込みを受け付けた場合には，中断される．しかしながら，アプリケーションパーティションにおいても，デバイスの操作等で一連の処理を中断することなく連続して行いたいケースがあると考えられる．

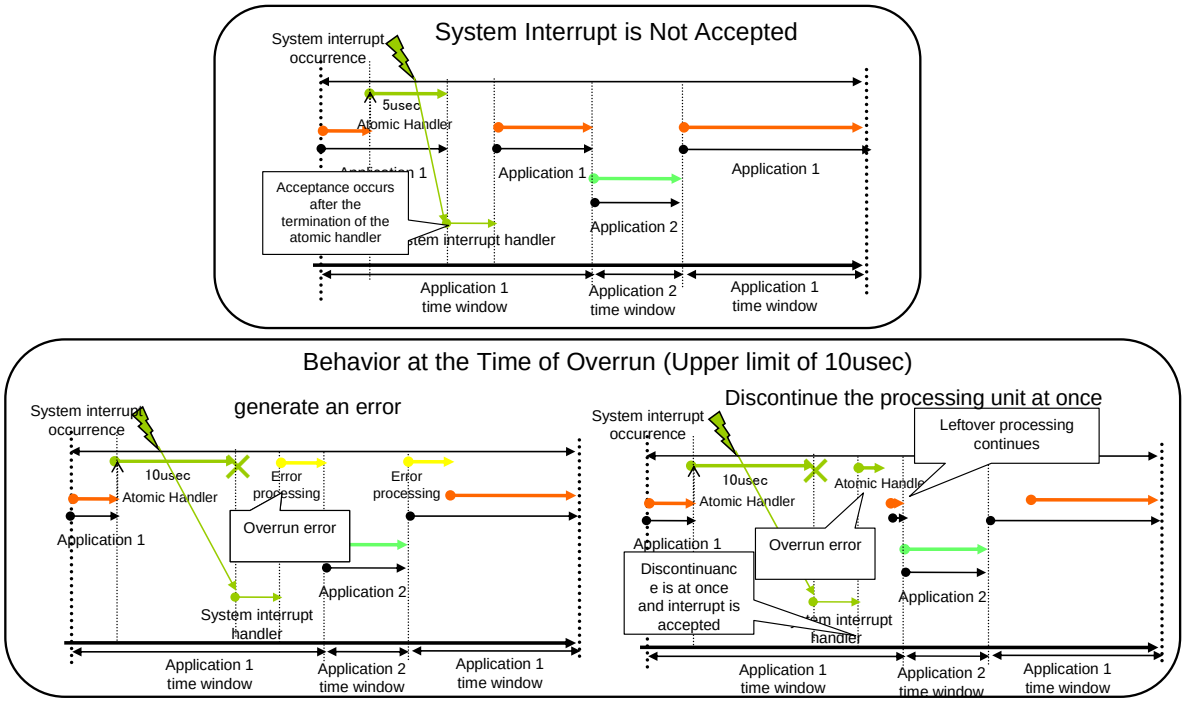

図 17-1. アトミックハンドラ

17.2. 呼び出し制約

アトミックハンドラは，いずれかのアプリケーションパーティションに所属し，所属するパーティションの処理単位からのみ呼び出し可能である．

17.3. メモリ制約

アトミックハンドラは，所属するアプリケーションパーティションの処理単位と同等のメモリ制約が適用される．

17.4. 最大実行時間制約

アトミックハンドラは生成時に最大実行時間を指定する必要がある．

全てのスケジューリングモードにおいて，アトミックハンドラが属するアプリケーションに割り当てられたタイムウィンドウの最大時間がアトミックハンドラの最大実行時間より短い場合は，コンフィグレーション時にエラーとなる．

現在のスケジューリングモードのタイムウィンドウの最大時間がアトミックハンドラの最大実行時間より短い場合は，アトミックハンドラ呼び出し API 実行時にエラーとなる．

アトミックハンドラが，指定された最大実行時間を超えて実行された場合の振る舞いは，次のいずれかから選択する．

- 処理を中断しエラーとする．呼び出し元にはエラーが返る．

- 処理を一旦中断し，割り当てられたタイムウィンドウを超えていればパーティション切り替えが行われ，システム割込みが発生していればシステム割込みが受け付けられる．その後，再び割り当てられたタイムウィンドウが来ると実行を再開する．呼び出し元にはエラーが返る．

アトミックハンドラが呼び出された時刻からパーティション切り替えまでの時間が，アトミックハンドラの最大実行時間より短い場合は，実行が延期され，次の十分な時間があるタイムウィンドウで呼び出される．実行が延期されている間は，アトミックハンドラ API を呼び出した処理単位が呼び出し時の状態でアトミックハンドラの実行を待っている（ビジーループにより）状態となり，パーティションは実行状態のままである．

アトミックハンドラ使用時の注意点については，[ParOS_SM]の 7.1.6 を参照のこと．

17.5. OS呼び出し制約

アトミックハンドラからは，OS の API は呼び出すことができない．呼び出した場合はコンテキストエラーが返る．

17.6. アトミックハンドラ ：API

17.6.1. アトミックハンドラ生成〔S〕

【静的 API】

CRE_ATH(ID athid, ATR athatr, ATMHDR atmhdr, UTIM exectim)

【パラメータ】

ID	athid	アトミックハンドラ ID
ATR	athatr	アトミックハンドラ属性(ATH_ERROR, ATH_PAUSE)
ATMHDR	atmhdr	アトミックハンドラ先頭アドレス
UTIM	exectim	最大実行時間 (有効値 1 以上)

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_RSATR	予約属性（athatr が不正または使用できない，パーティションに所属していない）
E_PAR	パラメータエラー（atmhdr, exectim が不正）
E_OBJ	オブジェクト状態エラー（条件は機能の項を参照すること）

【機能】

各パラメータで指定したアトミックハンドラ生成情報に従って，アトミックハンドラを生成する．

アトミックハンドラ属性（atmatr）に指定可能な属性は次の通りである．

TA_ATH_ERROR	0x01U	エラー
TA_ATH_PAUSE	0x02U	中断

アトミックハンドラが属するアプリケーションに割り当てられたタイムウィンドウがアトミックハンドラの最大実行時間より長いスケジューリングモードが存在しない場合は，オブジェクト状態エラーが返る．

パーティションの囲み外に記述した場合には，E_RSATR エラーが返る．

17.6.2. アトミックハンドラ呼び出し

【C 言語 API】

ER ercd = CallAtomicHandler(ID athid)

【パラメータ】

ID	athid	アトミックハンドラ ID (有効値 1 - システムに存在するアトミックハンドラの最大数)

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_CTX	コンテキストエラー
E_ID	不正 ID 番号（athid が不正）
E_OACV	オブジェクトアクセス違反（対象アトミックハンドラの所属するパーティションではない）
E_OBJ	オブジェクト状態エラー
E_TMOUT	タイムアウト

【機能】

athid のアトミックハンドラ（対象アトミックハンドラ）を実行する．具体的な振る舞いは以下の通り．

呼び出し元のパーティションと対象アトミックハンドラの属するパーティションが異なる場合には，エラーとなり E_OACV が返る．

呼び出し元のパーティションがシステムパーティションの場合は，エラーとなり，E_CTX が返る．

現在のスケジューリングモードにおける呼び出し元のパーティションに属するタイムウィンドウの最大時間と比較してアトミックハンドラの最大実行時間が長い場合はエラーとなり，E_OBJ が返る．

呼び出し時点のタイムウィンドウの残り時間がアトミックハンドラの最大実行時間より短い場合には，呼び出しは中断されパーティションは実行状態のまま，ビジーループでアトミックハンドラの最大実行時間より長いタイムウィンドウが実行されるのを待ち，その後にアトミックハンドラが実行される．

最大実行時間を超えて対象アトミックハンドラが実行された場合の振る舞いは，コンフィギュレーション時に指定された属性に従う．

ATH_ERROR が指定されている場合には，処理を中断しエラーとする．呼び出し元にはエラーとして，E_TMOUT が返る．

ATH_PAUSE が指定されている場合には，処理を一旦中断し，割り当てられたタイムウィンドウを超えていればパーティション切り替えが行われ，システム割込みが発生していればシステム割込みが受け付けられる．その後，再び割り当てられたタイムウィンドウが来ると実行を再開する．呼び出し元にはエラーとして，E_TMOUT が返る．

18. チェックフック

18.1. 概要

チェックフックは，アプリケーションパーティションから呼び出し可能で，他のパーティションに影響を与える可能性のある操作 (API 等)を呼び出した場合に，その要求を実行するかを判断するフックである．チェックフックの種類は以下の通りである．

- スケジューリングモード変更チェックフック　[ECC][DCC]
- システム例外ハンドラ呼び出しチェックフック

チェックフックは, チェックフックを実行する API を呼び出したアプリケーションパーティションのタイムウィンドウで実行される．

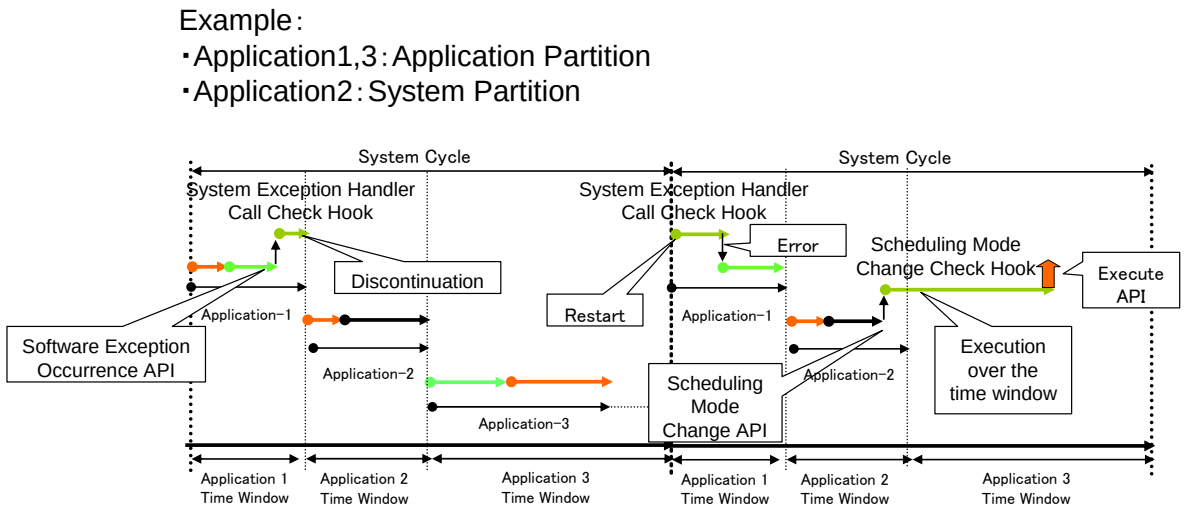

図 18-1. チェックフック

18.2. スケジューリングモード変更チェックフック　[ECC][DCC]

アプリケーションパーティションからのスケジューリングモード変更要求 API（ChangeSchedulingModee()呼び出し時，もしくはパーティション終了処理ルーチンの戻り値が正（スケジューリングモード変更要求）の場合に実行される．

スケジューリングモード変更チェックフック実行時には，スケジューリングモード変更要求 API の引数として指定されたスケジューリングモード（変更したいスケジューリングモード）と切り替えタイミング及び呼び出し元のパーティションの ID が引数として渡される．

パーティション終了処理ルーチンの戻り値が正の場合には，切り替えタイミングは，CSM_NEXT_CYCLE が渡される．

スケジューリングモード変更チェックフックではこれらの情報を用いて，アプリケーションパーティションからの要求を受け付けるか判断する．要求を受け付ける場合は OK を，受け付けない場合はエラーをリターンする．

18.3. システム例外ハンドラ呼び出しチェックフック

パーティション例外ハンドラからシステム例外ハンドラを実行するためのソフトウェア例外発生 API（RaiseSoftwareException）呼び出し時に実行される．

システム例外ハンドラ呼び出しチェックフック実行時には，ソフトウェア例外発生 API の引数として指定された，例外要因番号と呼び出し元のパーティションの ID が渡される．

システム例外ハンドラ呼び出しチェックフックではこれらの情報を用いて，アプリケーションパーティションからの要求を受け付けるか判断する．要求を受け付ける場合は OK を，受け付けない場合はエラーをリターンする．

チェックフックエラーをリターンした場合，アプリケーションパーティションからの要求は実行されず，API 呼び出しの場合はエラーが返る．

パーティションが故障して，ソフトウェア例外発生 API を不正に実行すると，システム例外ハンドラが呼び出され，システム全体に影響を与えてしまう．この対策のために，ソフトウェア例外発生 API 呼び出し時に，システム例外ハンドラの前に実行されるシステム例外ハンドラ呼び出しチェックフックにて，システム例外ハンドラを実行するかどうかユーザーにて判断すること．詳細や対応方法については，[ParOS_SM]の 7.3.1 を参照のこと．

18.4. チェックフック ：API

18.4.1. スケジューリングモード変更チェックフックの定義〔S〕[ECC][DCC]

【静的 API】

DEF_SCHMODE_CHECKHOOK(ATR hookatr, CHKHOOK chkhook)

【パラメータ】

ATR	hookatr	スケジューリングモード変更チェックフック属性
CHKHOOK	chkhook	スケジューリングモード変更チェックフックの先頭番地

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_RSATR	予約属性（hookatr が不正または使用できない，システムパーティション以外に所属している）
E_PAR	パラメータエラー（chkhook が不正）
E_OBJ	オブジェクト状態エラー（定義済みのスケジューリングモード変更チェックフックに対する再定義）

【機能】

スケジューリングモード変更チェックフックを各パラメータで指定したスケジューリングモード変更チェックフック定義情報に従って，定義する．

スケジューリングモード変更チェックフックを 2 重以上に定義した場合には, E_OBJ エラーとなる．
本静的 API はシステムパーティションの囲みの中に記述する必要ある．

18.4.2. システム例外ハンドラ呼び出しチェックフックの定義

【静的 API】

DEF_SYSERRORHDR_CHECKHOOK(ATR hookatr, CHKHOOK chkhook)

【パラメータ】

ATR	hookatr	システム例外ハンドラ呼び出しチェックフック属性
CHKHOOK	chkhook	システム例外ハンドラ呼び出しチェックフックの先頭番地

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_RSATR	予約属性（hookatr が不正または使用できない，システムパーティション以外に所属している）
E_PAR	パラメータエラー（chkhook が不正）
E_OBJ	オブジェクト状態エラー（定義済みのシステムエラーハンドラ呼び出しチェックフックに対する再定義）

【機能】

システム例外ハンドラ呼び出しチェックフックを各パラメータで指定したシステム例外ハンドラ呼び出しチェックフック定義情報に従って，定義する．

システム例外ハンドラ呼び出しチェックフックを 2 重以上に定義した場合には，E_OBJ エラーとなる．

本静的 API はシステムパーティションの囲みの中に記述する必要ある．

19. 例外ハンドリング

19.1. 概要

メモリ違反や 0 除算等の例外が発生すると，例外ハンドラが実行され，例外処理を行う．例外ハンドラはパーティション例外ハンドラ，システム例外ハンドラの 2 種類があり，例外要因や例外が発生した場所によって，どちらかが呼び出される．

【仕様決定の理由】

処理レベル例外ハンドラ，パーティション例外ハンドラ，システム例外ハンドラの 3 段階に分ける方法もあるが，扱いや静的 API の記述が複雑になるのに対してメリットが低いと考え，2 段階にすることにした．

19.2. 例外要因

例外要因としては， 0 除算，メモリ保護違反，未定義命令実行といった，ハードウェア的に発生するハードウェア例外や，ソフトウェア例外発生 API により発生させるソフトウェア例外，パーティション未満了エラー等のシステムパーティションの誤動作により発生する例外がある．例外は実装毎に定義される．

例外要因にはユニークな ID（例外要因番号）が設定される．ParOS では以下の例外を OS が提供する．以下の例外名称に対応する例外要因番号を，ターゲット依存部にて定義を行う．

例外要因番号	名称	例外種類
x	EXCNO_SWEXC	ソフトウェア例外
x	EXCNO_PARNCOMP	パーティション未満了エラー
x	EXCNO_SINTCNT	システム割込み受け付け回数エラー
x	EXCNO_SINTEXCT	システム割込み最大実行時間オーバーエラー
x	EXCNO_INVMEMACCESS	メモリアクセス違反
x	EXCNO_STVANONUPDATE	状態変数更新エラー

19.3. パーティション例外ハンドラ

パーティション例外ハンドラは，アプリケーションパーティション毎に登録することが可能である．システムパーティションに対しては，パーティション例外ハンドラは登録できない．

コンフィギュレーションにおいて，あるパーティションのパーティションレベル例外ハンドラの定義は，そのアプリケーションパーティションの囲みの中に記述する必要がある．

あるパーティション実行時に例外が発生した場合，そのパーティションにおいて，発生した例外要因に対してパーティション例外ハンドラが定義されていれば，そのパーティション例外ハンドラが実行される．

パーティション例外ハンドラが定義されていない場合には，システム例外ハンドラが実行される．

パーティション例外ハンドラは例外が発生したパーティション内のどの処理単位よりも優先して実行され，パーティションに割り当てられたタイムウィンド内でのみ実行される．

メモリアクセスに関しては例外発生元のパーティションの処理単位と同様のアクセス制約となる．

パーティション例外ハンドラ内で例外が発生した場合は，システム例外ハンドラが呼び出される．

発生した例外の情報（例外情報）は例外情報取得 API により取得可能である．

19.4. システム例外ハンドラ

システム例外ハンドラは，例外要因毎に一つ登録可能である．

コンフィギュレーションにおいて，システム例外ハンドラの定義はシステムパーティションの囲みの中に書く必要がある．アプリケーションパーティションに対してはシステム例外ハンドラを登録できない．

システム上のどの処理単位より優先して実行され，タイムウィンドウの概念は適用されない．システムパーティションの処理単位と同様にメモリアクセスの制約は発生しないため，注意すること（[ParOS_SM] 7.3.2）．

発生した例外の情報（例外情報）は例外情報取得 API により取得可能である．システム例外ハンドラが実行されると，全ての処理単位より優先的に実行されるため，注意が必要である．

19.5. 例外情報

パーティション例外ハンドラ及びシステム例外ハンドラから礼状情報取得 API により以下の情報を取得可能である．

- 例外要因番号
- ソフトウェアエラー番号
- エラー発生元のパーティション ID
- 実行中の処理単位の情報
- 実行中の処理単位の ID（ない場合は NULL）

19.6. ソフトウェア例外

ソフトウェア例外は，ソフトウェア例外発生 API により発生させる例外である．ソフトウェア例外には，例外要因番号 0 が予約されている．

ソフトウェア例外発生 API の引数にはソフトウェアエラー番号を指定することが可能である．

システムパーティションの処理単位，または，パーティション例外ハンドラからソフトウェア例外発生 API を呼び出した場合には，システム例外ハンドラが呼び出される．

パーティション例外ハンドラから呼び出された場合には，システム例外ハンドラ呼び出しチェックフックにより呼び出しの是非がチェックされる．

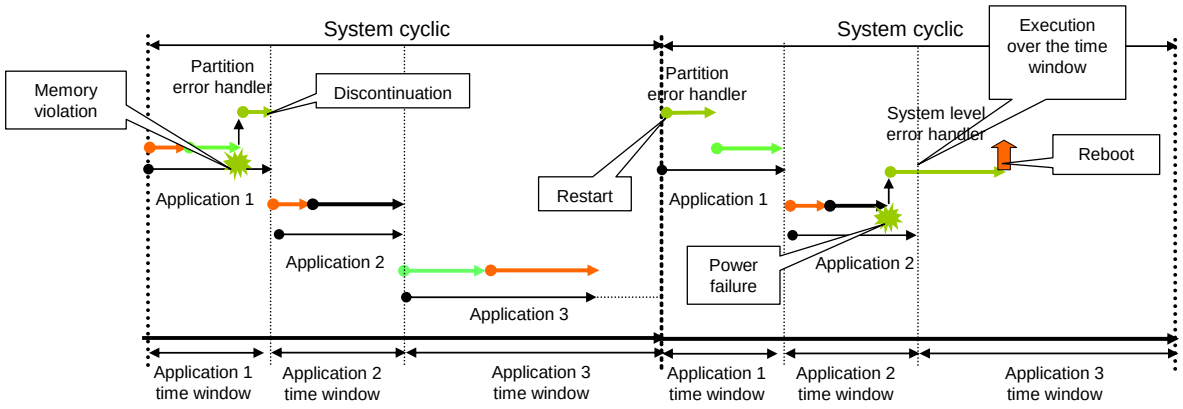

図 19-1. 例外ハンドラ

19.7. 例外ハンドリング ：API

19.7.1. 例外ハンドラ定義〔S〕

【静的 API】

DEF_EXC(EXCNO excno, ATR excatr, EXCHDR exchdr)

【パラメータ】

EXCNO	excno	例外ハンドラ番号（例外要因番号） (有効値 1 - 存在する例外の最大数)
ATR	excatr	例外ハンドラ属性
EXCHDR	exchdr	例外ハンドラの先頭番地

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_RSATR	予約属性（excatr が不正または使用できない，パーティションの囲みの外に記述されている）
E_PAR	パラメータエラー（excno，exchdr が不正）
E_OBJ	オブジェクト状態エラー（定義済みの例外要因番号に対する再定義，未定義の例外要因番号に対する定義解除）

【機能】

excno で指定した例外要因番号（対象例外要因番号）に対して，各パラメータで指定した例外ハンドラ定義情報に従って，例外ハンドラを定義する．

本 API がパーティションの囲みに記述された場合には，パーティション例外ハンドラとなり，システムパーティションの囲みの中に記述された場合には，システム例外ハンドラとなる．

例外ハンドラを定義する場合で，同じパーティションに対象例外要因番号に対してすでにパーティション例外ハンドラが定義されている場合には，E_OBJ エラーとなる．同様に対象例外要因番号に対して既にシステム例外ハンドラが定義されている場合にも E_OBJ エラーとなる．

19.7.2. ソフトウェア例外発生

【C 言語 API】

ER ercd = RaiseSoftwareException(SWERNO swerno);

【パラメータ】

SWERNO	swerno	ソフトウェアエラー番号 (有効値 1 以上)

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_CTX	コンテキストエラー（システム例外ハンドラからの呼び出し）
E_PAR	パラメータエラー（swerno が不正）
E_OACV	アクセス違反（チェックフックがエラーを返した）

【機能】

ソフトウェア例外を発生させる．

パーティションの処理単位から呼び出した場合，例外要因番号 0 のパーティション例外ハンドラが登録されていれば例外要因番号 0 のパーティション例外ハンドラを実行する．例外要因番号 0 のパーティション例外ハンドラが登録されていなければ，例外要因番号 0 のシステム例外ハンドラが実行される．

パーティション例外ハンドラやシステムパーティションに所属する処理単位から呼び出した場合には，システム例外ハンドラが実行される．

パーティション例外ハンドラからシステム例外ハンドラを呼び出した場合には，システム例外ハンドラ呼び出しチェックフックが呼び出される．

19.7.3. 例外情報取得

【C 言語 API】

ER ercd = GetExceptionInformation(T_EXCINFO pk_excinfo);

【パラメータ】

T_EXCINFO	pk_excinfo	例外情報を入れるパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

*例外情報（パケットの内容）

EXCNO	excno	例外要因番号
SWERNO	swerno	ソフトウェアエラー番号
ID	parid	エラーが発生元のパーティション ID
PRCTYPE	prctype	実行中の処理単位の情報
ID	prcid	実行中の処理単位の ID（ない場合は NULL）

【エラーコード】

E_CTX	コンテキストエラー（例外ハンドラ以外からの呼び出し）
E_MACV	メモリアクセス違反（pk_excinfo が指すメモリ領域への書き込みアクセスが許可されていない）

【機能】

例外情報を取得する．参照した例外情報は，pk_excinfo で指定したメモリ領域に返される．
例外ハンドラ以外から呼び出した場合には，E_CTX エラーが返る．

20. アプリケーション割込み

20.1. 概要

アプリケーション割込みは，アプリケーションパーティションに所属する割込みである．所属するアプリケーションパーティションのタイムウィンドウの実行中にのみ受け付けられる．

他のパーティションが実行されている間は，割込みは受け付けられず（ハードウェア的に可能な場合），所属するパーティションのタイムウィンドウが実行されると受け付けられる．

割込みが受け付けられると，アプリケーション割込みハンドラが実行される．アプリケーション割込み処理ハンドラはパーティション切り替えが発生すると，プリエンプションされる．

20.2. システム割込みとの違い

所属するアプリケーションパーティションのタイムウィンドウの実行中にのみ受け付けられる．そのため，システム割込みとは異なり，他のパーティションの CPU 利用率や実行タイミングに影響を与えない．

20.3. 割込み要求ラインの設定

割込み要求ラインの設定は，CFG_INT()により行う．ただし，割込み優先度(intpri)は無視される．

20.4. アプリケーション割込みハンドラ

アプリケーション割込みハンドラは，アプリケーション割込みが受け付けられると実行される．通常のタスクより優先度が高く，コンテキストは非タスクコンテキストとなる．呼び出し可能な SafeOS 互換 API は i が頭に付属したシステムコールのみである．呼び出し可能な ParOS API は"各 API の呼び出し可能コンテキスト"の節の情報を参照のこと．

アプリケーション割込み処理ハンドラはタスクと同様に，所属するパーティションに対して許可されていない空間にアクセスした場合には，エラーが発生する．所属するパーティションのタイムウィンドウ内で実行される．ディスパッチ禁止中であっても動作する．割込み禁止中は動作しない．

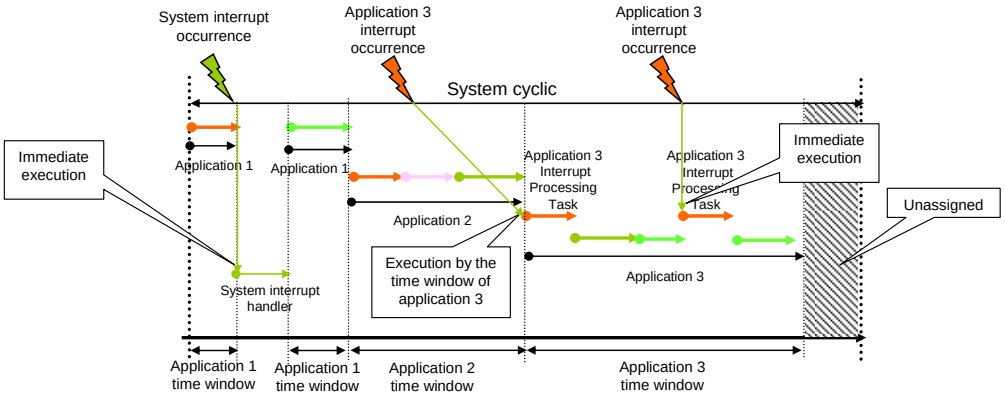

図 20-1. アプリケーション割込み

20.5. アプリケーション割込み ：API

20.5.1. アプリケーション割込みハンドラの定義〔S〕

【静的 API】

DEF_AINH(INHNO inhno, ATR inttskatr, INTTSK inttsk, PRI ainhpri,
SIZE stksz, STK_T *stk, SIZE sstksz, STK_T *sstk)
※ sstksz および sstk の記述は省略することができる

【パラメータ】

INHNO	inhno	割込みハンドラ番号 (有効値 1 - 存在する割込みの最大数)
ATR	inttskatr	アプリケーション割込みハンドラ属性
INTTSK	inttsk	アプリケーション割込みハンドラの先頭番地
PRI	ainhpri	優先度(有効値 -1 以下)
SIZE	stksz	アプリケーション割込みハンドラのスタック領域のサイズ（バイト数）(有効値 1 以上)
STK_T	*stk	アプリケーション割込みハンドラのスタック領域の先頭番地 (有効値 0 以上)
SIZE	sstksz	アプリケーション割込みハンドラのシステムスタック領域のサイズ（バイト数，省略可）(有効値 1 以上)
STK_T	*sstk	アプリケーション割込みハンドラのシステムスタック領域の先頭番地（省略可）(有効値 0 以上)

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_RSATR	予約属性（inhatr が不正または使用できない，パーティション囲みの外に記述）
E_PAR	パラメータエラー（inhno，inttask, stsz, stk sstskz, sstk が不正）
E_OBJ	オブジェクト状態エラー（条件は機能の項を参照すること）
E_NOMEM	メモリ不足（スタック領域やシステムスタック領域が確照すること）

【機能】

inhno で指定した割込みハンドラ番号（対象割込みハンドラ番号）に対して，各パラメータで指定した割込み処置タスク定義情報に従って，アプリケーション割込みハンドラを定義する．

対象割込みハンドラ番号に対応する割込み要求ラインの属性が設定されていない場合には，E_OBJ エラーとなる．また，対象割込みハンドラ番号に対してすでにアプリケーション割込みハンドラやシステム割込みハンドラが定義されている場合は，E_OBJ エラーとなる．

パーティションの囲み外に記述した場合には，E_RSATR エラーが返る．

まず，stk と stksz からタスクが用いるスタック領域が設定される．stksz に 0 を指定した時や，ターゲット定義の最小値よりも小さい値を指定した時には，E_PAR エラーとなる．sstk と sstksz からシステムスタック領域が設定される．この場合，sstksz に 0 を指定した時や，ターゲット定義の最小値よりも小さい値を指定した時には，E_PAR エラーとなる．

スタックに対する扱いは SafeOS の CRE_TSK のパラメータと同等であるため，SafeOS の CRE_TSK 仕様を参考のこと．

21. タイムウィンドウハンドラ

21.1. 概要

タイムウィンドウハンドラは，特定のタイムウィンドウの開始時に実行されるハンドラである．SafeOS のアラームハンドラと周期ハンドラをタイムウィンドウに対応させたもので，次の 2 種類がある．

- タイムウィンドウ周期ハンドラ
- タイムウィンドウアラームハンドラ

アプリケーションパーティションでは，SafeOS のタイムイベントハンドラは使用できず，タイムウィンドウハンドラのみを使用可能である．

21.2. タイムウィンドウハンドラ

タイムウィンドウハンドラは，いずれかのパーティションに所属する．また，指定可能なタイムウィンドウは所属するパーティションに割り当てられたタイムウィンドウのみである．

アプリケーションパーティションに所属するタイムウィンドウハンドラは，他の処理単位と同様に所属するタイムウィンドウ内でのみ実行される．また，他の処理単位と同様のメモリ制約となる．

タイムウィンドウハンドラのコンテキストは非タスクコンテキストとなる．呼び出し可能な SafeOS 互換 API は i が頭に付属したシステムコールのみである．呼び出し可能な ParOS API は"各 API の呼び出し可能コンテキスト"の節の情報を参照のこと．

21.3. タイムウィンドウ周期ハンドラ

タイムウィンドウ周期ハンドラは，指定したシステム周期で特定のタイムウィンドウの先頭で実行されるタイムウィンドウハンドラである．タイムウィンドウ周期ハンドラは，タイムウィンドウ周期ハンドラ ID と呼ぶ ID 番号によって識別する．

各周期ハンドラが持つ情報は次の通り．

- タイムウィンドウ周期ハンドラ属性
- タイムウィンドウ周期ハンドラの動作状態
- 次に周期ハンドラを起動するシステムティック
- 拡張情報
- タイムウィンドウ周期ハンドラの先頭番地
- 起動周期
- 属するパーティション
- 実行するタイムウィンドウ
- 優先度

システムティックで起動周期を指定可能．システムティックとシステム周期は同等であるため，例えば，1 と指定した場合には，毎システム周期実行され，2 と指定した場合は，システム周期 2 回毎に実行される．

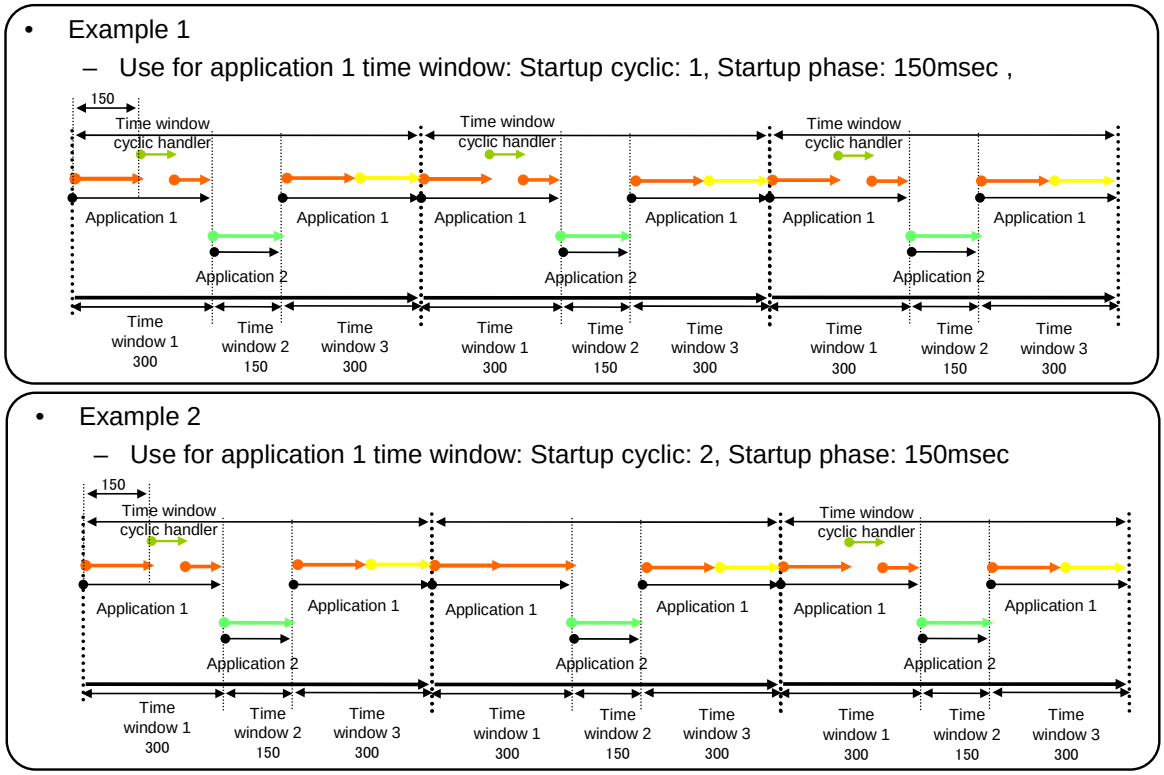

図 21-1. タイムウィンドウ周期ハンドラ

タイムウィンドウ周期ハンドラの動作状態は，動作している状態と動作していない状態のいずれかをとる．タイムウィンドウ周期ハンドラを動作している状態にすることを動作開始，動作していない状態にすることを動作停止という．

タイムウィンドウ周期ハンドラが動作している状態の場合には，タイムウィンドウ周期ハンドラを起動する時刻になると，タイムウィンドウ周期ハンドラの起動処理が行われる．具体的には，拡張情報をパラメータとして，タイムウィンドウ周期ハンドラが呼び出される．

タイムウィンドウ周期ハンドラ属性には，次の属性を指定することができる．

TA_STA	0x02U	タイムウィンドウ周期ハンドラの生成時にタイムウィンドウ周期ハンドラを動作開始する

TA_STA を指定しない場合，タイムウィンドウ周期ハンドラの生成直後には，周期ハンドラは動作していない状態となる．

タイムウィンドウ周期ハンドラはアプリケーション割込みハンドラと同様に負の優先度を設定可能である．

C 言語によるタイムウィンドウ周期ハンドラの記述形式は次の通り．

```
void twcyclic_handler(intptr_t exinf)
{
        タイムウィンドウ周期ハンドラ本体
}
```

exinf には，タイムウィンドウ周期ハンドラの拡張情報が渡される．

21.4. タイムウィンドウアラームハンドラ

指定したシステムティック後に特定のタイムウィンドウの先頭で実行されるタイムウィンドウハンドラ．タイムウィンドウアラームハンドラは，タイムウィンドウアラームハンドラ ID と呼ぶ ID 番号によって識別する．

各タイムウィンドウアラームハンドラが持つ情報は次の通り．

- タイムウィンドウアラームハンドラ属性
- タイムウィンドウアラームハンドラの動作状態
- タイムウィンドウアラームハンドラを起動するシステム周期
- 拡張情報
- タイムウィンドウアラームハンドラの先頭番地
- 属するアプリケーションパーティション
- 実行するタイムウィンドウ
- 優先度

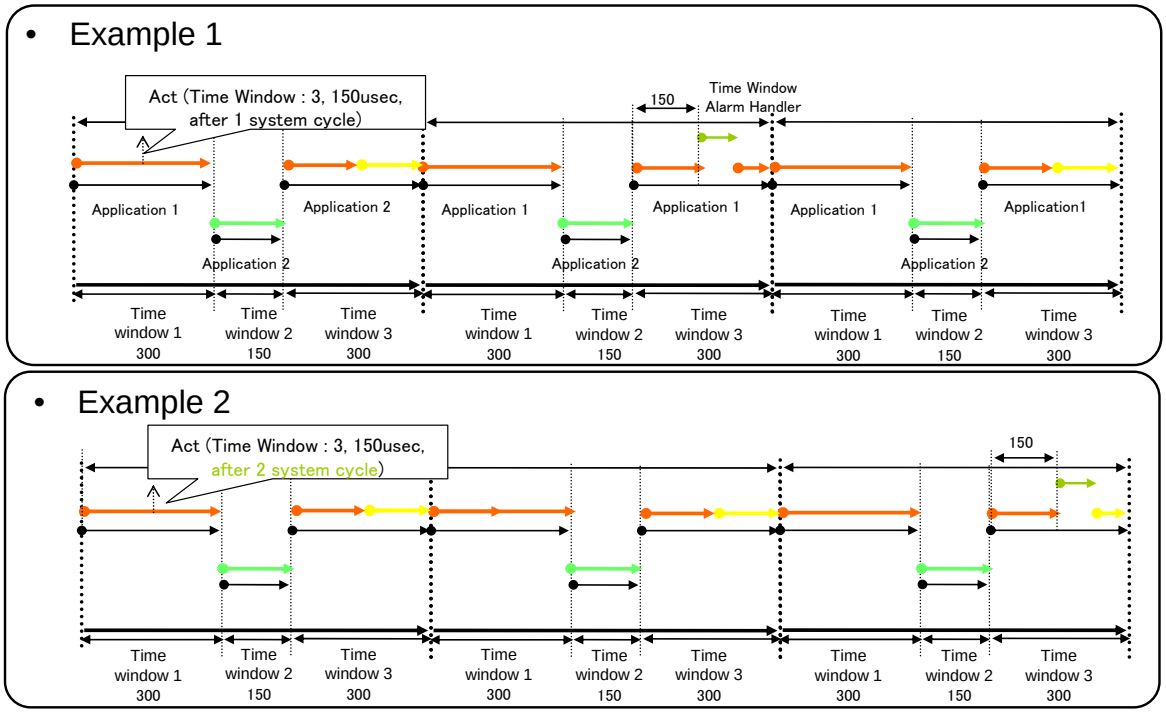

図 21-2. タイムウィンドウアラームハンドラ

タイムウィンドウアラームハンドラの動作状態は，動作している状態と動作していない状態のいずれかをとる．タイムウィンドウアラームハンドラを動作している状態にすることを動作開始，動作していない状態にすることを動作停止という．

タイムウィンドウアラームハンドラを起動するシステム周期は，タイムウィンドウアラームハンドラを動作開始する時に設定される．

タイムウィンドウアラームハンドラが動作している状態の場合には，アラームハンドラを起動するシステム周期になると，タイムウィンドウアラームハンドラの起動処理が行われる．具体的には，まず，タイムウィンドウアラームハンドラが動作していない状態にされる．その後に，拡張情報をパラメータとして，タイムウィンドウアラームハンドラが呼び出される．

タイムウィンドウアラームハンドラはアプリケーション割込みハンドラと同様に負の優先度を設定可能である．

C 言語によるタイムウィンドウアラームハンドラの記述形式は次の通り．

```
void twalarm_handler(intptr_t exinf)
{
        タイムウィンドウタイムウィンドウアラームハンドラ本体
}
```

exinf には，タイムウィンドウアラームハンドラの拡張情報が渡される．

21.5. タイムウィンドウハンドラ ：API

21.5.1. タイムウィンドウ周期ハンドラの生成〔S〕

【静的 API】

CRE_TWCYC(ID twcycid, ATR twcycatr, intptr_t exinf,
CYCHDR tcychdr,uint_t cyccnt, ID twid, PRI cycpri,
SIZE stksz, STK_T *stk, SIZE sstksz, STK_T *sstk)

【パラメータ】

ID	twcycid	生成するタイムウィンドウ周期ハンドラの ID 番号
ATR	twcycatr	タイムウィンドウ周期ハンドラ属性
intptr_t	exinf	タイムウィンドウ周期ハンドラの拡張情報
CYCHDR	twcychdr	タイムウィンドウ周期ハンドラの先頭番地
uint_t	cyccnt	タイムウィンドウ周期ハンドラの起動周期(有効値 1 以上)
ID	twid	実行タイムウィンドウ ID (有効値 1 - システムに存在するタイムウィンドウの ID)
PRI	cycpri	優先度(有効値 -1 以下)
SIZE	stksz	タイムウィンドウ周期ハンドラのスタック領域のサイズ （バイト数）(有効値 1 以上)
STK_T	*stk	タイムウィンドウ周期ハンドラのスタック領域の先頭番地 (有効値 0 以上)
SIZE	sstksz	タイムウィンドウ周期ハンドラのシステムスタック領域 のサイズ（バイト数，省略可）(有効値 1 以上)
STK_T	*sstk	タイムウィンドウ周期ハンドラのシステムスタック領域 の先頭番地（省略可）(有効値 1 以上)

【リターンパラメータ】

ER_ID	twcycid	生成された周期ハンドラの ID 番号（正の値） またはエラーコード

【エラーコード】

E_RSATR	予約属性（twcycatr が不正または使用できない，パーティション の囲みの中に記述されていない）
E_PAR	パラメータエラー（twcychdr，cyctim，cyccnt, cycpri が不正）
E_OBJ	オブジェクト状態エラー（システム周期で指定したタイム イベント周期ハンドラが登録済み）
E_ID	不正 ID 番号(twid)

【機能】

各パラメータで指定したタイムウィンドウ周期ハンドラ生成情報に従って，タイムウィンドウ周期ハンドラを生成する．具体的な振舞いは以下の通り．

twcycatr に TA_STA を指定した場合，対象タイムウィンドウ周期ハンドラは動作している状態となる．twcycatr に TA_STA を指定しない場合，対象タイムウィンドウ周期ハンドラは動作していない状態に初期化される．

CRE_TWCYC は，パーティションの囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．

cyccnt は 0 より大きくてはならない．

21.5.2. タイムウィンドウ周期ハンドラの動作開始

【C 言語 API】

ER ercd = StartTWCyclicHandler(ID twcycid)

【パラメータ】

ID	twcycid	対象タイムウィンドウ周期ハンドラの ID 番号 (有効値 1 - システムに存在するタイムウィンドウ周期ハンドラの最大数)

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し，システムパーティションからの呼び出し）
E_ID	正 ID 番号（twcycid が不正）
E_OACV	オブジェクトアクセス違反（対象タイムウィンドウ周期ハンドラに対する操作が許可されていない）

【機能】

twcycid で指定したタイムウィンドウ周期ハンドラ（対象タイムウィンドウ周期ハンドラ）を動作開始する．具体的な振舞いは以下の通り．

対象タイムウィンドウ周期ハンドラが動作していない状態であれば，対象タイムウィンドウ周期ハンドラは動作している状態となる．次にタイムウィンドウ周期ハンドラが起動されるシステム周期は，TWCRE_CYC で指定した cyccnt 周期後である．

対象周期ハンドラが動作している状態であれば，次に周期ハンドラを起動するシステム周期の再設定のみが行われる．

21.5.3. タイムウィンドウ周期ハンドラの動作停止

【C 言語 API】

ER ercd = StopTWCyclicHandler(ID twcycid)

【パラメータ】

ID	twcycid	対象タイムウィンドウ周期ハンドラの ID 番号 (有効値 1 - システムに存在するタイムウィンドウ周期ハンドラの最大数)

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し，システムパーティションからの呼び出し）
E_ID	不正 ID 番号（twcycid が不正）
E_OACV	オブジェクトアクセス違反（対象タイムウィンドウ周期ハンドラに対する通常操作が許可されていない）

【機能】

twcycid で指定したタイムウィンドウ周期ハンドラ（対象タイムウィンドウ周期ハンドラ）を動作停止する．具体的な振舞いは以下の通り．

対象タイムウィンドウ周期ハンドラが動作している状態であれば，動作していない状態になる．対象タイムウィンドウ周期ハンドラが動作していない状態であれば，何も行われずに正常終了する．

21.5.4. タイムウィンドウ周期ハンドラの状態参照

【C 言語 API】

ER ercd = RefTWCyclicHandler(ID twcycid, T_RTWCYC *pk_rtwcyc)

【パラメータ】

ID	twcycid	対象タイムウィンドウ周期ハンドラの ID 番号 (有効値 1 - システムに存在するタイムウィンドウ周期ハンドラの最大数)
T_RTWCYC *pk_rtwcyc		タイムウィンドウ周期ハンドラの現在状態を入れるパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

＊周期ハンドラの現在状態（パケットの内容）

STAT	twcycstat	タイムウィンドウ周期ハンドラの動作状態
uint_t	leftcnt	次に周期ハンドラを起動するシステム周期までのカウント

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し，システムパーティションからの呼び出し）
E_ID	不正 ID 番号（twcycid が不正）
E_OACV	オブジェクトアクセス違反（対象周期ハンドラに対する操作が許可されていない）
E_MACV	メモリアクセス違反（pk_rcyc が指すメモリ領域への書込みアクセスが許可されていない）

【機能】

twcycid で指定したタイムウィンドウ周期ハンドラ（対象タイムウィンドウ周期ハンドラ）の現在状態を参照する．参照した現在状態は，pk_rtwcyc で指定したパケットに返される．

cycstat には，対象タイムウィンドウ周期ハンドラの現在の動作状態を表す次のいずれかの値が返される．

TCYC_STP	0x01U	タイムウィンドウ周期ハンドラが動作していない状態
TCYC_STA	0x02U	タイムウィンドウ周期ハンドラが動作している状態

対象タイムウィンドウ周期ハンドラが動作している状態である場合には，leftcnt に，次にタイムウィンドウ周期ハンドラ起動するシステム周期のカウントが返される．対象周期ハンドラが動作していない状態である場合には，leftcnt の値は保証されない．

マルチプロセッサ対応カーネルでは，prcid に，対象周期ハンドラの割付けプロセッサの ID 番号が返される．

21.5.5. タイムウィンドウアラームハンドラの生成〔S〕

【静的 API】

CRE_TWALM(ID twalmid, ATR twalmatr, intptr_t exinf,
ALMHDR twalmhdr, ID twid, PRI cycpri
SIZE stksz, STK_T *stk, SIZE sstksz, STK_T *sstk)

【パラメータ】

ID	twalmid	生成するタイムイベントアラームハンドラの ID 番号
ATR	twalmatr	タイムイベントアラームハンドラ属性
intptr_t	exinf	タイムイベントアラームハンドラの拡張情報
ALMHDR	twalmhdr	タイムイベントアラームハンドラの先頭番地
ID	twid	実行タイムウィンドウ ID (有効値 1 - システムに存在するタイムウィンドウの最大数)
PRI	cycpri	優先度 (有効値 -1 以下)
SIZE	stksz	タイムウィンドウアラームハンドラのスタック領域のサイズ （バイト数）(有効値 1 以上)
STK_T	*stk	タイムウィンドウアラームハンドラのスタック領域の先頭番地 (有効値 0 以上)
SIZE	sstksz	タイムウィンドウアラームハンドラのシステムスタック 領域のサイズ（バイト数，省略可）(有効値 1 以上)
STK_T	*sstk	タイムウィンドウアラームハンドラのシステムスタック 領域の先頭番地（省略可）(有効値 0 以上)

【リターンパラメータ】

ER_ID	twalmid	生成されたアラームハンドラの ID 番号（正の値） またはエラーコード

【エラーコード】

E_RSATR	予約属性（twalmatr が不正または使用できない， パーティションの囲みの中に記述されていない）
E_PAR	パラメータエラー（twalmhdr, twid, cycpri が不正）
E_OBJ	オブジェクト状態エラー（twalmid で指定したタイム イベントアラームハンドラが登録済み）
E_ID	不正 ID 番号(twid)

【機能】
各パラメータで指定したタイムイベントアラームハンドラ生成情報に従って，タイムイベントアラームハンドラを生成する．対象アラームハンドラは，動作していない状態に初期化される．

CRE_TWALM は，パーティションの囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．

21.5.6. タイムウィンドウアラームハンドラの動作開始

【C 言語 API】

ER ercd = StartTWAlarmHandler(ID twalmid, uint_t twalmcnt)

【パラメータ】

ID	twalmid	対象タイムウィンドウアラームハンドラの ID 番号 (有効値 1 - システムに存在するタイムウィンドウアラームハンドラの最大数)
uint_t	twalmcnt	タイムウィンドウアラームハンドラの起動時刻（相対周期）

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_CTX	コンテキストエラー（CPU ロック状態からの呼出し，システムパーティションからの呼び出し）
E_ID	不正 ID 番号（twalmid が不正）
E_PAR	パラメータエラー（twalmcnt が不正）
E_OACV	オブジェクトアクセス違反（対象アラームハンドラに対する操作が許可されていない）

【機能】

twalmid で指定したタイムウィンドウアラームハンドラ（対象タイムウィンドウアラームハンドラ）を動作開始する．具体的な振舞いは以下の通り．

対象タイムウィンドウアラームハンドラが動作していない状態であれば，対象タイムウィンドウアラームハンドラは動作している状態となる．タイムウィンドウアラームハンドラを起動するシステム周期は，StartTWAlarmHandler を呼び出してから，twalmcnt で指定した相対周期後（システム周期実行回数）に設定される．

対象アラームハンドラが動作している状態であれば，タイムウィンドウアラームハンドラを起動するシステム周期時刻の再設定のみが行われる．

almcnt は，0 より大きくてはなければならない．

21.5.7. タイムウィンドウアラームハンドラの動作停止

【C 言語 API】

ER ercd = StopTWAlarmHandler(ID twalmid)

【パラメータ】

ID	twalmid	対象タイムウィンドウアラームハンドラの ID 番号 (有効値 1 - システムに存在するタイムウィンドウアラームハンドラの最大数)

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_CTX	コンテキストエラー（CPU ロック状態からの呼出し，システムパーティションからの呼び出し）
E_ID	不正 ID 番号（twalmid が不正）
E_OACV〔P〕	オブジェクトアクセス違反（対象アラームハンドラに対する操作が許可されていない）

【機能】

twalmid で指定したタイムウィンドウアラームハンドラ（対象タイムウィンドウアラームハンドラ）を動作停止する．具体的な振舞いは以下の通り．

対象タイムウィンドウアラームハンドラが動作している状態であれば，動作していない状態となる．対象タイムウィンドウアラームハンドラが動作していない状態であれば，何も行われずに正常終了する．

21.5.8. タイムウィンドウアラームハンドラの状態参照

【C 言語 API】

ER ercd = RefTWAlaramHandler(ID twalmid, T_RTWALM *pk_rtwalm)

【パラメータ】

ID	twalmid	対象タイムウィンドウアラームハンドラの ID 番号 (有効値 1 - システムに存在するアラームハンドラの最大数)
T_RTWALM *pk_rtwalm		タイムウィンドウアラームハンドラの現在状態を入れるパケットへのポインタ

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

＊タイムウィンドウアラームハンドラの現在状態（パケットの内容）

STAT	twalmstat	タイムウィンドウアラームハンドラの動作状態
uint_t	leftcnt	タイムウィンドウアラームハンドラを起動するシステム周期までのカウント

【エラーコード】

E_CTX	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し，システムパーティションからの呼び出し）
E_ID	不正 ID 番号（twalmid が不正）
E_OACV	オブジェクトアクセス違反（対象アラームハンドラに対する操作が許可されていない）
E_MACV	メモリアクセス違反（pk_rtwalm が指すメモリ領域への書込みアクセスが許可されていない）

【機能】

twalmid で指定したタイムウィンドウアラームハンドラ（対象タイムウィンドウアラームハンドラ）の現在状態を参照する．参照した現在状態は，pk_rtwalm で指定したパケットに返される．

twalmstat には，対象タイムウィンドウアラームハンドラの現在の動作状態を表す次のいずれかの値が返される．

TALM_STP	0x01U	タイムウィンドウアラームハンドラが動作していない状態
TALM_STA	0x02U	タイムウィンドウアラームハンドラが動作している状態

対象アラームハンドラが動作している状態である場合には，leftcnt に，タイムウィンドウアラームハンドラ起動するシステム周期でのカウントが返される．対象タイムウィンドウアラームハンドラが動作していない状態である場合には，leftcnt の値は保証されない．

22. システム割込み　[BCC+][ECC][DCC]

22.1. 概要

システム割込みは，システムパーティションに所属し，割込み発生後，システムパーティションで実行されている処理単位が割込みを禁止しているか，より優先度が高いシステム割込みが実行されている以外は，即座に受け付けられる割込みである．

システム割込みは，全てのパーティションの実行タイミングに影響を与える．使用時の注意点に関しては，[ParOS_SM]の 7.3.4 を参照のこと．

22.2. システム割込みの時間保護

システム割込みは，割り込んだパーティションの実行タイミングを変化させる．また，オーバランや想定以上の頻度で割込み発生すると，各パーティションの"CPU 利用率"，"実行タイミング"を変化させる．

デバイスの故障により，システム割込みが想定以上に発生し，システムの実行に悪影響を及ぼすことを防ぐため，システム割込みには，時間保護を必ず設定すること．

個々のシステム割込み毎に以下を設定可能である．

(1) システム周期内での発生回数

(2) 発生毎の割込みハンドラの最悪実行時間

時間保護のため，(1)x(2)時間分の未割り当てのタイムウィンドウを各モードの最後に配置することを推奨する．

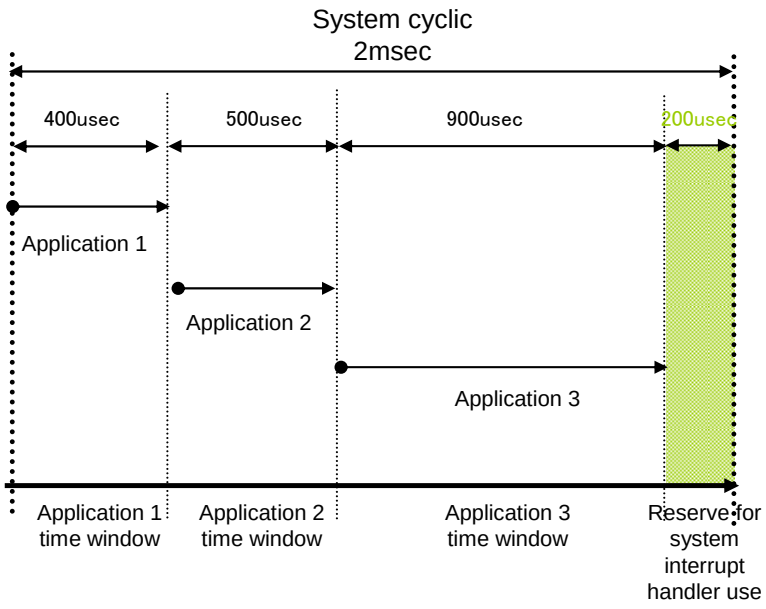

図 22-1. タイムウィンドウのリザーブ

システム周期内で(1)で指定した回数の割込みを受けた場合の振る舞いは，コンフィギュレーション時に次のいずれかに指定可能である．

- 次のシステム周期まで割込みの受け付けを禁止
- 同一システム周期内で次の割込みが発生したらエラーとする

(2)を超える時間で割込みハンドラが実行された場合は，システムレベルエラーを発生させる．

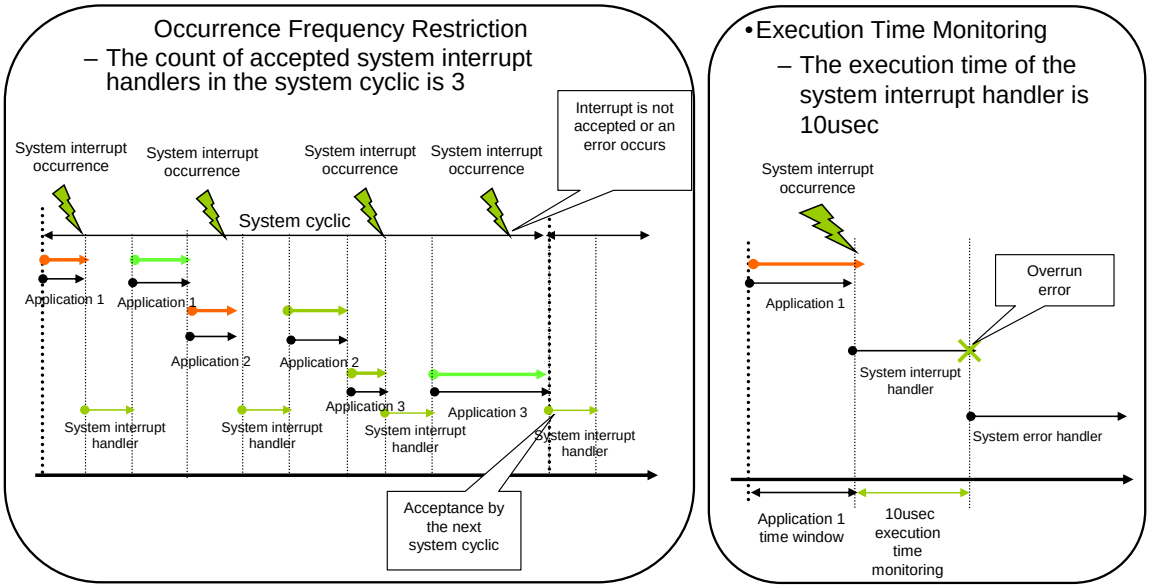

図 22-2. システム割込みの時間保護

【仕様決定の理由】

システム周期内で(1)で指定した回数の割込みを受けた場合の振る舞いを 2 種類から選択可能な理由は，割込みの発生が設計値以上の原因が 2 種類考えられるためである．一つは，外的な要因によって設計以上の頻度で割込みが入る可能性がある場合である（ネットワークからのデータ受信等）．この場合，受け付けを禁止する方法が適切である．もう一つは，デバイスの性質上，設定以上の割込みが発生する場合はデバイスの設定ミスや故障の可能性が高い場合であり，この場合は，エラーとする方法が適切である．

22.3. システム割込みの終了通知

システム割込みの終了通知 API を呼び出すと，割込み 1 回分の処理が終わったものとして，割込み受け付け回数を 1 増やし，新たな割込み処理として実行時間監視を行う．

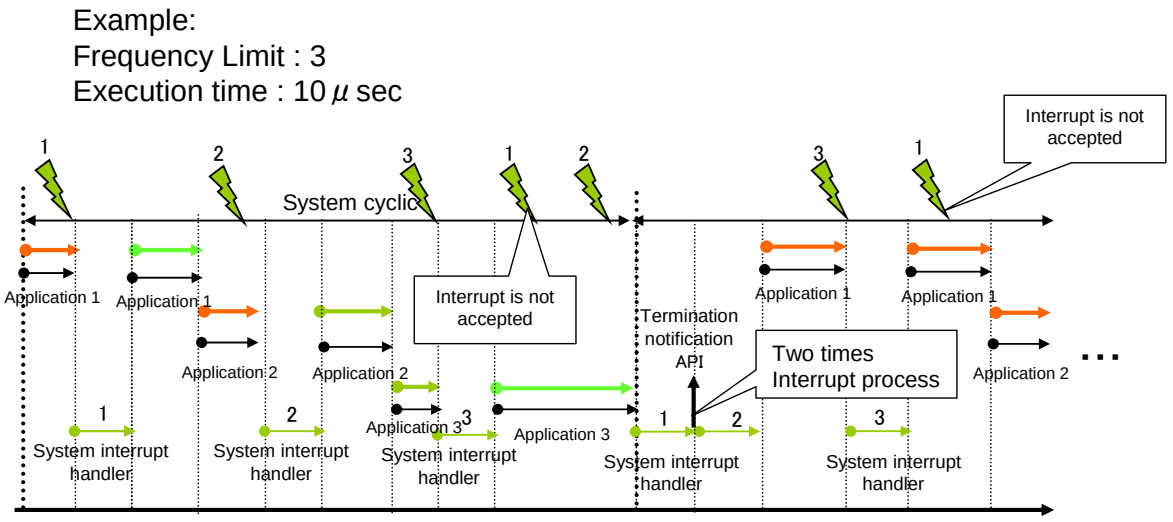

図 22-3. システム割込みの終了通知

【仕様決定の理由】

システム周期内での発生回数が規定値以上となり，時間保護により割込みが禁止となった場合を考える．この状態で次のシステム周期となり割込みを受け付けた場合，複数の割込み要求がペンディングする可能性がある．その場合，1 回の割込みハンドラの起動で割込み複数回分の処理を行う必要がある．

しかしながら，割込み複数回分の処理を行うと実行時間監視によりエラーとなるため，割込み 1 回分の処理が終わった時点でシステム割込みの終了通知を呼び出すことでこのエラーを回避する．

22.4. 機能制限付きシステム割込み

機能制限付きシステム割込みは BCC+でのみサポートされる．通常のシステム割込みとは異なり，機能制限付きシステム割込みから起動される割込みハンドラでは，一部のチャネル関連の API を除き，SafeOS/ParOS/COM のいずれの API も呼び出すことができない．そのため，機能制限付きシステム割込みにより起動された割込みハンドラが終了すると，必ず割り込んだパーティションの実行を再開する．

機能制限付きシステム割込みから呼び出し可能な API は次の通りである．

- メッセージキューチャネル開始
  StartMessageQueue(ID msgqid)
- メッセージキューチャネル停止
  StopMessageQueue(ID msgqid)
- メッセージキューチャネル書き込み(ポーリング)
  PSendMessageQueue(ID ifid, intptr_t *p_data)
- メッセージキューチャネル読み込み(ポーリング)
  PReciveMessageQueue(ID ifid, ID *p_parid, intptr_t *p_data)

- 状態変数チャネル開始
  StartStateVariable(ID stvaid)
- 状態変数チャネル停止
  StopStateVariable(ID stavaid)
- 状態変数チャネル書き込み
  WriteStateVariable(ID ifid, intptr_t *p_data)
- 状態変数チャネル読み込み
  ReadStateVariable(ID ifid, intptr_t *p_data)

22.5. システム割込み ：API

22.5.1. システム割込みハンドラの定義〔S〕〔S〕

【静的 API】

DEF_SINH(INHNO inhno, ATR inhatr, INTHDR inthdr, uint_t count, UTIM exectim)

【パラメータ】

INHNO	inhno	システム割込みハンドラ番号 (有効値 有効なシステム割込みハンドラ番号)
ATR	inhatr	システム割込みハンドラ属性
INTHDR	inthdr	システム割込みハンドラの先頭番地
uint_t	count	システム周期内での割込みの発生回数(有効値 1 以上)
UTIM	exectim	システム割込みハンドラの最悪実行時間(有効値 1 以上)

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_RSATR	予約属性（inhatr が不正または使用できない，システム パーティションの囲み以外に記述）
E_PAR	パラメータエラー（inhno，inthdr, count exectim が不正）
E_OBJ	オブジェクト状態エラー（条件は機能の項を参照すること）

【機能】

inhno で指定した割込みハンドラ番号（対象割込みハンドラ番号）に対して，各パラメータで指定したシステム割込みハンドラ定義情報に従って，システム割込みハンドラを定義する．

システム割込み属性（inhatr）に指定可能な属性は次の通りである．

TA_SYSINH_COVR_DIS	0x01U	割込みの受付けを禁止
TA_SYSINH_COVR_ERROR	0x02U	エラー

対象割込みハンドラ番号に対応する割込み要求ラインの属性が設定されていない場合には，E_OBJ エラーとなる．また，対象割込みハンドラ番号に対してすでにシステム割込みハンドラが定義されている場合は E_OBJ エラーとなる．

DEF_SINH は，システムパーティション囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．

22.5.2. システム割込み終了処理通知

【C 言語 API】

```
ER ercd = EndSystemInterruptHandler();
```

【パラメータ】

なし

【リターンパラメータ】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_CTX	コンテキストエラー（システム割込みハンドラ以外からの呼び出し）

【機能】

システム割込みハンドラが 1 回分終了したことを OS に通知．1 回のシステム割込みハンドラの起動で複数回分の処理を実行するときに呼び出す．

システム割込みハンドラ以外が呼び出した場合には E_CTX エラーが返る．

23. システムタイムイベントハンドラ　[DCC]

23.1. 概要

システムパーティションは，SafeOS との互換性を考慮して SafeOS の持つシステムタイムイベントハンドラ（システム周期ハンドラ，システムアラームハンドラ）を使用することが可能である．

使用可能な API は SafeOS の持つタイムイベントハンドラ機能と同等であるが，生成 API のみは，実行時間や実行間隔が可能なように拡張がなされている．

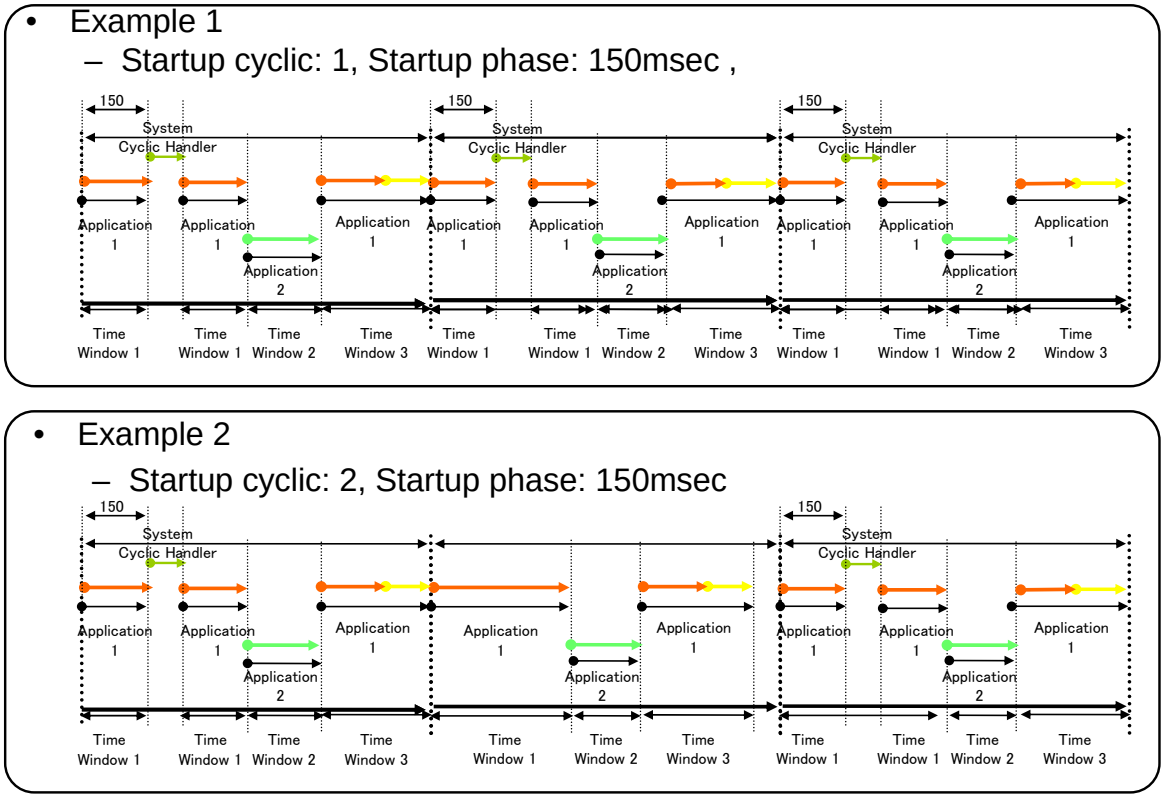

図 23-1. システム周期ハンドラ

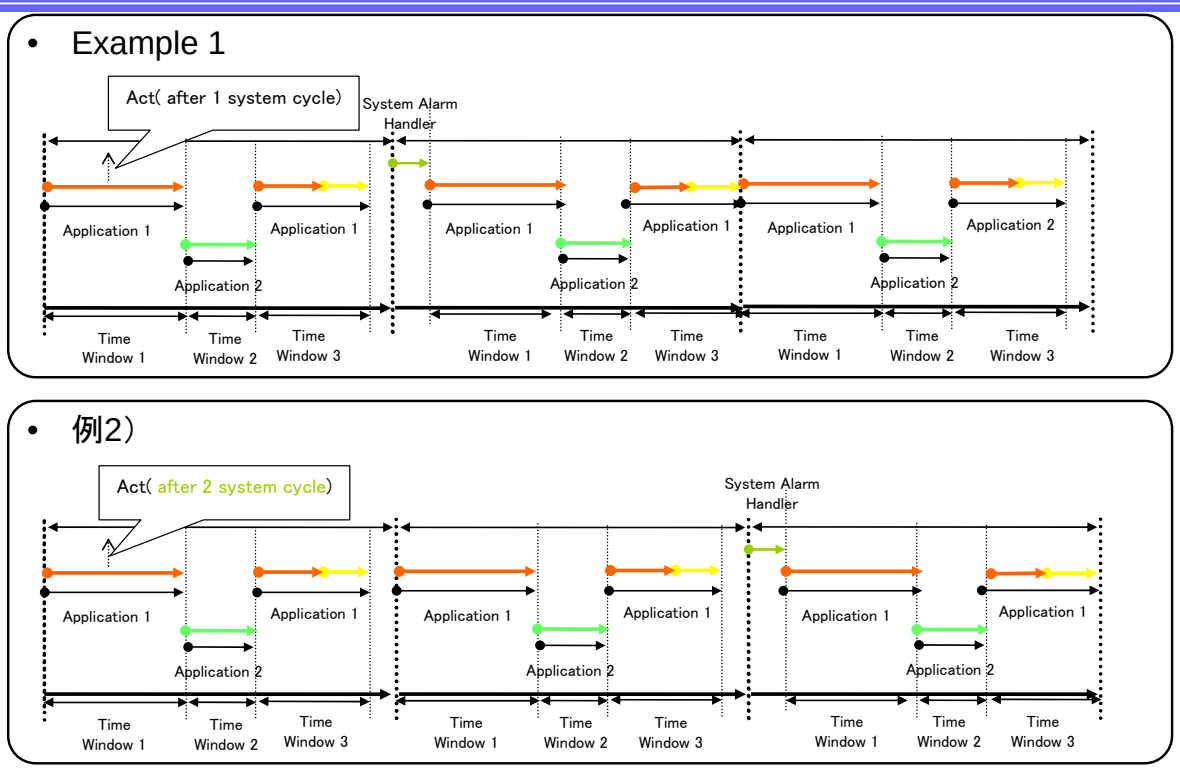

図 23-2. システムアラームハンドラ

23.2. システムタイムイベントハンドラの時間保護

システム割込みハンドラと同様に実行時間の監視を行う.

個々のシステムタイムイベントハンドラ毎に以下を設定可能である.

(1) 発生毎のシステムタイムイベントハンドラの最悪実行時間

時間保護のため, (1)の時間分の未割り当てのタイムウィンドウを各モードの最初に配置することを推奨する.

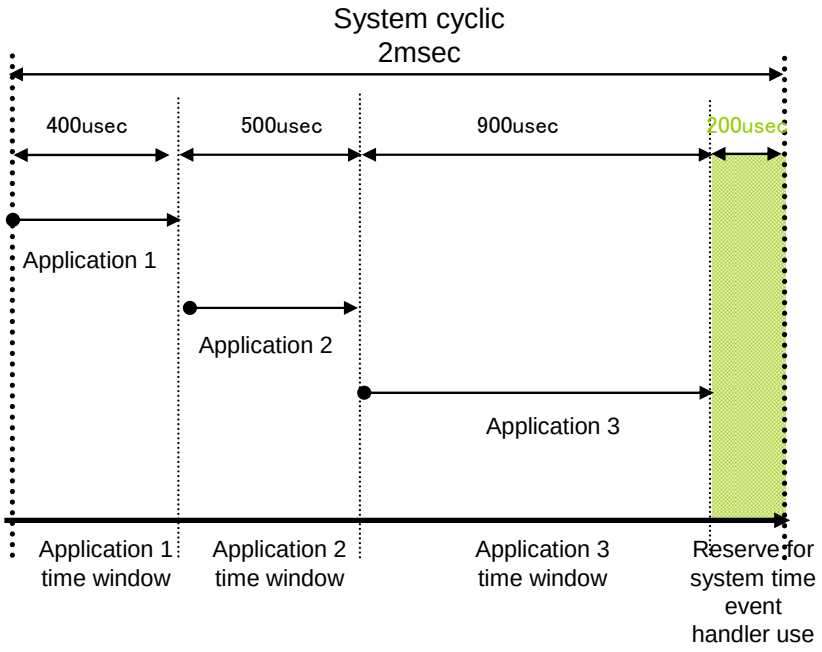

図 23-3. システムタイムイベントハンドラの時間保護

(1)を超える時間でシステムタイムイベントハンドラが実行された場合は, システムレベルエラーを発生させる.

23.3. システムタイムイベントハンドラ ：API

23.3.1. システム周期ハンドラの生成〔S〕[DCC]

【静的 API】

CRE_SCYC(ID cycid, ATR cycatr, intptr_t exinf, CYCHDR cychdr,
RELTIM cyctim, RELTIM cycphs, UTIM exectim)

【パラメータ】

ID	cycid	生成するシステム周期ハンドラの ID 番号
ATR	cycatr	システム周期ハンドラ属性
intptr_t	exinf	システム周期ハンドラの拡張情報
CYCHDR	cychdr	システム周期ハンドラの先頭番地
RELTIM	cyctim	システム周期ハンドラの起動周期(有効値 1 以上)
RELTIM	cycphs	システム周期ハンドラの起動位相(有効値 0 以上
UTIME	exectim	システム周期ハンドラの最大実行時間(有効値 1 以上)

【リターンパラメータ】

ER_ID	cycid	生成されたシステム周期ハンドラの ID 番号 （正の値）またはエラーコード

【エラーコード】

E_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し）
E_RSATR	予約属性（cycatr が不正または使用できない，システムパーティションの囲み以外に記述されている)
E_PAR	パラメータエラー（cychdr，cyctim，cycphs,exectim が不正）
E_OBJ	オブジェクト状態エラー（cycid で指定した周期ハンドラが登録済み）

【機能】

各パラメータで指定したシステム周期ハンドラ生成情報に従って，システム周期ハンドラを生成する．具体的な振舞いは以下の通り．

cycatr に TA_STA を指定した場合，対象システム周期ハンドラは動作している状態となる．次に周期ハンドラを起動する時刻は，サービスコールを呼び出した時刻（静的 API の場合はカーネルの起動時刻）から，cycphs で指定した相対時間後に設定される．cycatr に TA_STA を指定しない場合，対象システム周期ハンドラは動作していない状態に初期化される．

静的 API においては，cycid はオブジェクト識別名，cycatr，cyctim，cycphs は整数定数式パラメータ，exinf と cychdr は一般定数式パラメータである．

cyctim は，0 より大きく，TMAX_RELTIM 以下の値でなければならない．また，cycphs は，TMAX_RELTIM 以下でなければならない．cycphs に cyctim より大きい値を指定してもよい．exectim は，0 より大きくなければならない．

CRE_SCYC は，システムパーティションの囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．

23.3.2. システムアラームハンドラの生成〔S〕[DCC]

【静的 API】

CRE_SALM(ID almid, ATR almatr, intptr_t exinf, ALMHDR almhdr,
UTIM exectim)

【パラメータ】

ID	almid	生成するシステムアラームハンドラの ID 番号（CRE_ALM の場合）
ATR	almatr	システムアラームハンドラ属性
intptr_t	exinf	システムアアラームハンドラの拡張情報
ALMHDR	almhdr	システムアアラームハンドラの先頭番地
UTIM	exectim	システムアアラームハンドラの最大実行時間(有効値 1 以上)

【リターンパラメータ】

ER_ID	almid	生成されたシステムアアラームハンドラの ID 番号 （正の値）またはエラーコード

【エラーコード】

E_CTX〔s〕	コンテキストエラー（非タスクコンテキストからの呼出し， CPU ロック状態からの呼出し）
E_RSATR	予約属性（almatr が不正または使用できない，システム パーティションの囲み以外に記述されている)
E_PAR	パラメータエラー（almhdr が不正）
E_OBJ	オブジェクト状態エラー（almid で指定したアラームハンドラが 登録済み：CRE_ALM の場合）

【機能】

各パラメータで指定したシステムアラームハンドラ生成情報に従って，システムアラームハンドラを生成する．対象システムアラームハンドラは，動作していない状態に初期化される．

静的 API においては，almid はオブジェクト識別名，almatr は整数定数式パラメータ，exinf と almhdr は一般定数式パラメータである．exectim は，0 より大きくなければならない．

CRE_CYC は，システムパーティションの囲みの中に記述しなければならない．そうでない場合には，E_RSATR エラーとなる．

24. タスク保護（パーティション内保護）

24.1. 概要

これらの機能はパーティション内のタスク間の時間やメモリ（スタック）の保護機能である．これらの機能はオプションとする．

24.2. タスク実行時間監視（オーバランハンドラ機能）

TOPPERS/ASP カーネルの拡張機能のオーバランハンドラ機能と同等.

24.3. タスクスタック保護

タスクのユーザスタック領域に対しては，そのタスクのみが書込みアクセスおよび読出しアクセスを行うことができる．

【仕様決定の理由】

パーティション間の独立性を確保することを第一目的とするなら，パーティション内のタスク間の保護は必要ないという考え方もあるため，実装するかはオプションとする．

25. OSが用いるスタック（非タスクコンテキスト用スタック）

OS が使用するスタック（SafeOS における非タスクコンテキスト用スタック）は，パーティション毎に用意する．また，システム全体で使用する非タスクコンテキスト用スタックも必要である．

パーティションにおいてパーティション毎の非タスクコンテキスト用スタックを使用する処理単位としては，パーティション初期化ルーチン・パーティション終了処理ルーチン・パーティション例外ハンドラ等がある．

システム全体で使用する非タスクコンテキスト用スタックを使用する処理単位としては，システム例外ハンドラ・システム初期化ルーチン・システム終了処理ルーチン等がある．

パーティション毎に必要となるスタックのサイズやスタックとして使用するメモリの指定は，コンフィギュレーションファイルにおいて，パーティションの囲みに DEF_ICS を記述することで指定する．

システム全体で使用する非タスクコンテキスト用スタックの指定は，コンフィギュレーションファイルにおいて，パーティションの囲みの外に DEF_ICS を記述することで指定する．

それぞれ，DEF_ICS により指定しなかった場合は，ディフォルト値のサイズのスタック領域が自動的に生成され，使用される．

26. APIー覧

26.1. C言語API

各 API は返り値としてエラーコードを返す．API 呼び出し時の注意点については，[ParOS_SM]の 7.1.3 を参照のこと．

- システム起動 API
  StartSystem(void)
- システム終了 API
  ShutdownSystem(void)
- システム状態参照 API
  GetSystemState(T_SSTATE *pk_sstate)
- パーティション起動 API
  StartPartition(ID parid)
- パーティション終了 API
  ShutDownPartition(ID parid)
- パーティションの状態取得 API
  GetPartitionState(ID parid, T_PSTATE *pk_pstate)
- データグループ初期化
  InitDataGroup(ID dgid, ATR init_atr)
- スケジューリングモード変更 API　[ECC][DCC]
  ChangeSchedulingMode(ID schmid, ATR csmatr)
- メッセージキューチャネル開始
  StartMessageQueue(ID msgqid)
- メッセージキューチャネル停止
  StopMessageQueue(ID msgqid)
- メッセージキューチャネル書き込み(タイムアウトなし)
  SendMessageQueue(ID ifid, intptr_t *p_data)
- メッセージキューチャネル書き込み(ポーリング)
  PSendMessageQueue(ID ifid, intptr_t *p_data)
- メッセージキューチャネル書き込み(タイムアウト付き)
  TSendMessageQueue(ID ifid, intptr_t *p_data, TMO tmout)
- メッセージキューチャネル読み込み(タイムアウトなし)
  ReciveMessageQueue(ID ifid, ID *p_parid, intptr_t *p_data)
- メッセージキューチャネル読み込み(ポーリング)
  PReciveMessageQueue(ID ifid, ID *p_parid, intptr_t *p_data)
- メッセージキューチャネル読み込み(タイムアウト付き)

TReciveMessageQueue(ID ifid, ID *p_parid, intptr_t *p_data, TMO tmout)

- メッセージキューチャネルの状態参照

  RefMessageQueue(ID msgqid, T_RMSGQ *pk_rmsgq)

- 状態変数チャネル開始

  StartStateVariable(ID stvaid)

- 状態変数チャネル停止

  StopStateVariable(ID stavaid)

- 状態変数チャネル書き込み

  WriteStateVariable(ID ifid, intptr_t *p_data)

- 状態変数チャネル読み込み

  ReadStateVariable(ID ifid, intptr_t *p_data)

- 状態変数チャネル状態参照

  RefStateVariable(ID stvaid, T_RSTVA *pk_rstva)

- アトミックハンドラ呼び出し

  CallAtomicHandler(ID athid)

- ソフトウェア例外発生

  RaiseSoftwareException(SWERNO swerno);

- 例外情報取得

  GetExceptionInformation(T_EXCINFO pk_excinfo);

- タイムウィンドウ周期ハンドラの動作開始

  StartTWCyclicHandler(ID twcycid)

- タイムウィンドウ周期ハンドラの動作停止

  StopTWCyclicHandler(ID twcycid)

- タイムウィンドウ周期ハンドラの状態参照

  RefTWCyclicHandler(ID twcycid, T_RTWCYC *pk_rtwcyc)

- タイムウィンドウアラームハンドラの動作開始

  StartTWAlarmHandler(ID twalmid, uint_t twalmcnt)

- タイムウィンドウアラームハンドラの動作停止

  StopTWAlarmHandler(ID twalmid)

- タイムウィンドウアラームハンドラの状態参照

  RefTWAlaramHandler(ID twalmid, T_RTWALM *pk_rtwalm)

- システム割込み終了処理通知

  EndSystemInterruptHandler();

26.2. 静的API

- システム初期化ルーチン定義 API〔S〕

  DEF_SYSTEM_INI(ATR iniatr, intptr_t exinf, INIRTN inirtn)
- システム終了処理ルーチン定義 API〔S〕

  DEF_SYSTEM_TER(ATR teratr, intptr_t exinf, TERRTN terrtn)
- パーティション初期化ルーチン定義 API〔S〕

  DEF_PARTITION_INI(ATR iniatr, intptr_t exinf, INIRTN inirtn)
- パーティション終了処理ルーチン定義 API〔S〕

  DEF_PARTITION_TER(ATR teratr, intptr_t exinf, TERRTN terrtn)
- システム周期定義 API〔S〕

  DEF_SYSTEM_CYCLE(UTIM syscyc)
- タイムウィンドウ定義 API〔S〕

  CRE_TWINDOW(ID twid, UTIM start, UTIM dulation)
- スケジューリングモード定義 API〔S〕　[ECC][DCC]

  CRE_SCHMODE(ID schmid, ATR schmatr)
- タイムウィンドウ割り当て〔S〕

  ATT_TW(ID schmid, ID twid)
- パーティションの属性指定〔S〕

  SET_PAR_ATTR(ATR paratr)
- セクションの登録〔S〕

  ATT_SEC("セクション名", ATR mematr, "メモリリージョン名")
- オブジェクトモジュールの登録〔S〕

  ATT_MOD("オブジェクトモジュール名", ATR mematr)
- メモリオブジェクトの登録〔S〕

  ATT_MEM(ATR mematr, void *base, SIZE size)
- データグループの生成〔S〕

  ER ercd = CRE_DTG(ID dtgid)
- メモリオブジェクトのデータグループへの追加〔S〕

  ER ercd = ATT_DTG(ID dtgid, void *base)
- メッセージキューチャネル生成〔S〕

  CRE_MSGQ(ID msgqid, ATR msgqatr, uint_t msgsize, uint_t msgcnt)
- 状態変数チャネル生成〔S〕

  CRE_STVA(ID stvaid, ATR stvaatr, uint_t varsize)
- インタフェース生成〔S〕

  CRE_INF(ID ifid, ATR ifatr)
- メッセージキューチャネル・インタフェース接続定義〔S〕

ATT_IF_MSGQ(ID msgqid ID ifid)

- 状態変数チャネル・インタフェース接続定義〔S〕

  ATT_IF_STVA(ID stvaid, ID ifid)

- アトミックハンドラ生成〔S〕

  CRE_ATH(ID athid, ATR athatr, ATMHDR atmhdr, UTIM exectim)

- スケジューリングモード変更チェックフックの定義〔S〕　[ECC][DCC]

  DEF_SCHMODE_CHECKHOOK(ATR hookatr, CHKHOOK chkhook)

- システム例外ハンドラ呼び出しチェックフックの定義

  DEF_SYSERRORHDR_CHECKHOOK(ATR hookatr, CHKHOOK chkhook)

- 例外ハンドラ定義〔S〕

  DEF_EXC(EXCNO excno, ATR excatr, EXCHDR exchdr)

- アプリケーション割込みハンドラの定義〔S〕

  DEF_AINH(INHNO inhno, ATR inttskatr, INTTSK inttsk, PRI ainhpri,
  SIZE stksz, STK_T *stk, SIZE sstksz, STK_T *sstk)

- タイムウィンドウ周期ハンドラの生成〔S〕

  CRE_TWCYC(ID twcycid, ATR twcycatr, intptr_t exinf,
  CYCHDR tcychdr,uint_t cyccnt, ID twid, PRI cycpri,
  SIZE stksz, STK_T *stk, SIZE sstksz, STK_T *sstk)

- タイムウィンドウアラームハンドラの生成〔S〕

  CRE_TWALM(ID twalmid, ATR twalmatr, intptr_t exinf,
  ALMHDR twalmhdr, ID twid, PRI cycpri
  SIZE stksz, STK_T *stk, SIZE sstksz, STK_T *sstk)

- システム割込みハンドラの定義〔S〕

  DEF_SINH(INHNO inhno, ATR inhatr, INTHDR inthdr,
  uint_t count, UTIM exectim)

- システム周期ハンドラの生成〔S〕[DCC]

  CRE_SCYC(ID cycid, ATR cycatr, intptr_t exinf, CYCHDR cychdr,
  RELTIM cyctim, RELTIM cycphs, UTIM exectim)

- システムアラームハンドラの生成〔S〕[DCC]

  CRE_SALM(ID almid, ATR almatr, intptr_t exinf, ALMHDR almhdr,
  UTIM exectim)

26.3. 各APIの呼び出し可能コンテキスト

別紙を参照のこと.

27. アプリケーション要件

ParOS の幾つかの機能は，使用方法を誤るとシステムの不具合を起こす可能性ある．これらの機能を使用するには，[ParOS_SM]の 7 章"アプリケーション要件・制約"に示している使用方法を参照して正しい使い方をすること.

このような機能と使用方法を誤った場合に発生する問題, [ParOS_SM]の使用上の注意への参照を示す.

Section	Function	Application Requirements/Influence/Limitation	Corresponding [ParOS_SM] section	Violate Safety Goal
26	API	各 API は返り値としてエラーコードを返すため，ユーザープログラムはエラーコードをチェックする必要がある.	7.1.3	No
11	パーティションスケジューリング	タイムウィンドウの割り当てを細かくすると，OS によるタイムウィンドウの 切り替えオーバヘッドが増加するため注意すること.	7.1.4.1	No
11.4.	パーティションのアイドル属性	アイドル属性を付加したパーティションは，"CPU 利用率保護"，"実行順序保護"， "実行タイミング保護"保証されないため注意が必要である.	7.1.4.2	No
15	SafeOS 互換機能	一部の機能は，ParOS で追加された機能で代替可能であるため使用できない．ParOS により代替機能が提供されているものはその機能を用いること.	7.1.5	No
16	パーティション間通信（COM）	引数のチェック，タイムアウト，データの内容チェックが必要	7.2	Yes (他のパーティションに影響を与える)

Section	Function	Application Requirements/Influence/Limitation	Corresponding [ParOS_SM] section	Violate Safety Goal
17	アトミックハンドラ	時間保護が有効な場合，戻り値をチェックして成功したか確認すること	7.1.6	Yes(システムに影響を与える)
18	チェックフック	パーティションが故障して、ソフトウェア例外発生 API を不正に実行すると、 システム例外ハンドラが呼び出され、システム全体に影響を与えてしまう。こ の対策のために、ソフトウェア例外発生 API 呼び出し時に、システム例外ハン ドラの前に実行されるシステム例外ハンドラ呼び出しチェックフックにて、シ ステム例外ハンドラを実行するかどうかユーザーにて判断すること。	7.3.1	Yes(システムに影響を与える)
19	例外処理	システムパーティションの処理単位と同様にメモリアクセスの制約は発生しないため，注意すること.	7.3.2	Yes(システムに影響を与える)
22	システム割込み	システム割込みは，全てのパーティションの実行タイミングに影響を与える.	7.3.4	Yes(システムに影響を与える)
10.15.1	スケジューリングモード変更	スケジューリングモードを変更すると，システム全体に影響を与える.	7.3.3	Yes(システムに影響を与える)
10.9	システムパーティション	システムパーティションはシステム全体に影響を与える	7.3.5	Yes(システムに影響を与える)

28. リファレンス

28.1.1. 変数型

- ID　：パーティション ID，データグループ ID，スケジューリングモード ID
- SYS_STAT　：システム状態
- CHE_MODE　：スケジューリングモード
- PAR_STAT　：パーティション状態
- SYSTIM　：システム周期
- INTTSK　：アプリケーション割込みハンドラの先頭アドレス
- ATMHDR　：アトミックハンドラの先頭アドレス
- UTIM　：マイクロ秒指定
- SYSTIM　：システム周期
- SWERNO　：ソフトウェア例外番号
- PRCTYPE　：処理単位タイプ

28.1.2. 処理単位タイプ

- PRCTYPE_TASK　：タスク
- PRCTYPE_SINH　：システム割込みハンドラ
- PRCTYPE_AINH　：アプリケーション割込みハンドラ
- PRCTYPE_TIH　：タイムイベントハンドラ（タイムウィンドウハンドラも含む）
- PRCTYPE_TIH　：タイムイベントハンドラ

28.1.3. 属性指定

- システム状態
  - 未定義状態　TSS_UNDEF
  - システム初期化中状態　TSS_INIT
  - システム終了処理中状態　TSS_TER
  - システム停止状態　TSS_STOP
  - システム通常状態　TSS_NORMAL

- パーティション状態
  - 実行状態　TPS_NORMAL
  - 実行可能状態　TPS_RUNNABLE
  - 満了状態　TPS_EXPIRE
  - 休止状態　TPS_IDLE
  - 初期化中状態　TPS_INIT
  - 終了処理中状態　TPS_TER

- 停止状態　　TPS_STOP

・ システムパーティション ID
- PID_SYSTEM　　(-1)

・ データグループ初期化属性
- INIT_DG_VERIFY
- INIT_DG_NONE

・ スケジューリングモード切り替えタイミング
- CSM_NEXT_CYCLE
- CSM_IMMIDIATE

・ スケジューリングモード
- SCHM_DEFAULT　　：ディフォルトスケジューリングモード

・ チャネルの状態
- 停止状態：TCH_STOP
- 通常状態：TCH_NORMAL

・ チャネルの方向
- 入力：TA_IN
- 出力：TA_OUT

・ アトミックハンドラ属性
- TA_ATH_ERROR　　：エラー
- TA_ATH_PAUSE　　：中断

・ システム割込み受け付け回数オーバー時の振る舞い
- TA_SYSINH_COVR_DIS　　：次のシステム周期まで割込みの受け付けを禁止
- TA_SYSINH_COVR_ERROR　：エラー

・ パーティションの属性
- TA_PAR_IDLE　　：アイドルパーティション

28.1.4. パケット

システム状態のパケット形式

```
typedef struct t_sstate {
    SYS_STAT        sys_stat            現在のシステム状態
    SCH_MODE        current_sch_mode;   現在のスケジューリングモード.
    SCH_MODE        next_sch_mode;      次システム周期のスケジューリングモード
    uint_t          count;              起動回数（システム初期化中状態から
                                        システム通常状態への遷移回数）
}T_SSTATE;
```

パーティション状態のパケット形式

```
typedef struct t_psate {
        PAR_STATE   par_state;          パーティションの状態
        uint_t      count;              起動回数（初期化中状態から通常状態への
                                        遷移の回数）
}T_PSTATE;
```

メッセージキューチャネル状態のパケット形式

```
typedef struct t_rmsgq {
        STAT        chstat;             メッセージキューチャネルの状態
        ID          stskid;             送信待ち行列の先頭のタスクの ID 番号
        ID          rtskid;             受信待ち行列の先頭のタスクの ID 番号
        uint_t      sdtqcnt;            データキューに格納されているデータの数
}T_RMSGQ;
```

状態変数チャネル状態のパケット形式

```
typedef struct t_rstva {
        STAT        chstat;             状態変数チャネルの状態
}T_RSTVA;
```

例外情報のパケット形式

```
typdef struct t_excinfo {
        EXCNO       excno;              例外要因番号
        SWERNO      swerno;             ソフトウェアエラー番号
        ID          parid;              エラーが発生元のパーティション ID
}T_EXCINFO;
```

タイムウィンドウ周期ハンドラの状態のパケット形式

```
typdef struct t_rtwcyc {
        STAT        twcycstat;      タイムウィンドウ周期ハンドラの動作状態
        uint_t      leftcnt;        次に周期ハンドラを起動するシステム
                                    周期までのカウント
}T_RTWCYC;
```

タイムウィンドウアラームハンドラの状態のパケット形式

```
typdef struct t_rtwalm {
        STAT        twalmstat;      タイムウィンドウアラームハンドラの動作状態
        uint_t      leftcnt;        タイムウィンドウアラームハンドラを起動する
                                    システム周期までのカウント
}T_RTWALM
```

29. コンフィギュレーションファイルの記述例

29.1. サンプルシステム１（メモリ保護なし）

29.1.1. 構成

- システム初期化ルーチン
  - sys_init

- システム終了処理ルーチン
  - sys_ter

- システム周期：1msec

- アプリケーションパーティション：PARTITION_A
  - タスク
    - TASK_A1
    - TASK_A2
    - TASK_A3
  - パーティション初期化ルーチン
    - par_init_a
  - パーティション終了処理ルーチン
    - par_ter_a
- アプリケーションパーティション：PARTITION_B
  - タスク
    - TASK_B1
    - TASK_B2
    - TASK_B3
  - パーティション初期化ルーチン
    - par_init_b
  - パーティション終了処理ルーチン
    - par_ter_b
  - アプリケーションパーティション：PARTITION_C
    - TASK_C1
    - TASK_C2
    - TASK_C3
  - パーティション初期化ルーチン
    - par_init_c
  - パーティション終了処理ルーチン

- par_ter_c

- メッセージキューチャネル：MSGQ_1
  - 所属パーティション：PARTITION_C
  - インタフェース
    - IF_MSGQ_1_PAR_A：出力
    - IF_MSGQ_1_PAR_B：出力
    - IF_MSGQ_1_PAR_C：入力
- 状態変数チャネル：STVA_1
  - 所属パーティション：PARTITION_A
  - インタフェース
    - IF_STVA_1_PAR_A：出力
    - IF_STVA_1_PAR_B：入力
    - IF_STVA_1_PAR_C：入力

- スケジューリングモード：SCHMODE_1
  - タイムウィンドウ
    - TWIN_1_S：50usec　システムパーティション
    - TWIN_1_1：300usec　PARTITION_A
    - TWIN_1_2：300usec　PARTITION_B
    - TWIN_1_3：300usec　PARTITION_C
    - TWIN_1_R：50usec　リザーブ
- スケジューリングモード：SCHMODE_2
  - タイムウィンドウ
    - TWIN_2_S：50usec　システムパーティション
    - TWIN_2_1：450usec　PARTITION_A
    - TWIN_2_2：450usec　PARTITION_B
    - TWIN_2_R：50usec　リザーブ

- アトミックハンドラ：ATH_1
  - 所属パーティション 1：PARTITION_A
  - 実行時間　　：100usec
  - ハンドラアドレス　　：ath_1

- スケジューリングモード変更チェックフック
  - フックアドレス：schm_check_hdr_1
- システム例外ハンドラ呼び出しチェックフック

- フックアドレス：syserror_check_hdr_1

・ 0 除算例外
- 例外番号：EXCNO_ZERODIV
- パーティション例外ハンドラ
  - zediv_hdr_para
  - zediv_hdr_parb
  - zediv_hdr_parc
- システム例外ハンドラ
  - zediv_hdr_sys

・ メモリアクセス例外
- 例外番号：EXCNO_INVMEMACCESS
- パーティション例外ハンドラ
  - なし
- システム例外ハンドラ
  - invmemaccess_hdr_sys

・ アプリケーション割込み：DEV_A
- 割込みハンドラ番号：INHNO_DEV_A
- アプリケーション割込みハンドラ　：inh_dev_a
- 所属パーティション：PARTITION_A

・ アプリケーション割込み：DEV_B
- 割込みハンドラ番号：INHNO_DEV_B
- 割込みハンドラ　　：inh_dev_b
- 所属パーティション：PARTITION_B

・ アプリケーション割込み：DEV_C
- 割込みハンドラ番号：INHNO_DEV_C
- アプリケーション割込みハンドラ　：inh_dev_c
- 所属パーティション：PARTITION_C

・ タイムウィンドウハンドラ
- タイムウィンドウ周期ハンドラ：TWCYCID_1
  - 属性　　：TA_NULL
  - 周期　　：10
  - タイムウィンドウ　：TWIN_1_1
  - 所属パーティション：PARTITION_A
  - ハンドラ　　：twcyc_1_hdr

- タイムウィンドウ周期ハンドラ：TWCYCITWCYCID_2
  - 属性　　：TA_STA
  - 周期　　：2
  - タイムウィンドウ　：TWIN_1_2
  - 所属パーティション：PARTITION_B
  - ハンドラ　　：twcyc_2_hdr
- タイムウィンドウアラームハンドラ：TWALMID_1
  - 属性　　：TA_NULL
  - タイムウィンドウ　：TWIN_1_1
  - 所属パーティション：PARTITION_A
  - ハンドラ　　：twalm_1_hdr

・ システム割込み：DEV_SYS
- 割込みハンドラ番号：INHNO_DEV_SYS
- 割込みハンドラ　　：inh_dev_sys
- 発生回数　　：2
- 最悪実行時間　　：10

・ システムタイムイベントハンドラ
- システム周期ハンドラ：SCYC_1
  - ハンドラ：scyc_1_hdr
  - 周期　　：2
  - 位相　　：0
  - 最大実行時間：15
- システムアラームハンドラ：SALM_1
  - ハンドラ：salm_1_hdr
  - 最大実行時間：15

29.1.2. 静的API

```
/* システム周期 */
DEF_SYSTEM_CYCLE(1000);

/* スケジューリングモード */
CRE_SCHMODE(SCHMODE_1, SCHM_DEFAULT);
CRE_SCHMODE(SCHMODE_2, TA_NULL);

/* スケジューリングモードとタイムウィンドウの割り付け */
ATT_TW(SCHMODE_1, TWIN_1_S);
ATT_TW(SCHMODE_1, TWIN_1_1);
ATT_TW(SCHMODE_1, TWIN_1_2);
ATT_TW(SCHMODE_1, TWIN_1_3);
ATT_TW(SCHMODE_1, TWIN_1_R);
ATT_TW(SCHMODE_2, TWIN_2_S);
ATT_TW(SCHMODE_2, TWIN_2_1);
ATT_TW(SCHMODE_2, TWIN_2_2);
ATT_TW(SCHMODE_2, TWIN_2_R);

/* リザーブする未割り当てのタイムウィンドウ */
CRE_TWINDOW(TWIN_R_4, 950,   50);
CRE_TWINDOW(TWIN_R_3, 950,   50);

/* チャネルとインタフェース接続 */
ATT_IF_MSGQ(MSGQ_1, IF_MSGQ_1_PAR_A);
ATT_IF_MSGQ(MSGQ_1, IF_MSGQ_1_PAR_B);
ATT_IF_MSGQ(MSGQ_1, IF_MSGQ_1_PAR_C);
ATT_IF_STVA(STVA_1, IF_STVA_1_PAR_A);
ATT_IF_STVA(STVA_1, IF_STVA_1_PAR_B);
ATT_IF_STVA(STVA_1, IF_STVA_1_PAR_C);

/* システムパーティション */
PARTITION(PID_SYSTEM) {
    /* システム初期化ルーチンと終了処理ルーチン */
    DEF_SYSTEM_INI(TA_NULL, 1, sys_init_a);
    DEF_SYSTEM_TER(TA_NULL, 1, sys_ter_a);
```

```
    /* タイムウィンドウの定義 */
    CRE_TWINDOW(TWIN_1_S,    0,   50);
    CRE_TWINDOW(TWIN_2_S,    0,   50);
    /* チェックフック */
    DEF_SCHMODE_CHECKHOOK(TA_NULL, schm_check_hdr_1);
    DEF_SYSERRORHDR_CHECKHOOK(TA_NULL, syserror_check_hdr_1);
    /* システム例外ハンドラ */
    DEF_EXC(EXCNO_ZERODIV, zediv_hdr_sys);
    DEF_EXC(EXCNO_INVMEMACCESS, invmemaccess_hdr_sys);
    /* システム割込み */
    DEF_SINH(INHNO_DEV_SYS, TA_NULL, inh_dev_sys, 2, 10);
    CFG_INT(INHNO_DEV_SYS, {INTATR_DEV_SYS, INTPRI_DEV_SYS});
    /* システム周期ハンドラ */
    CRE_SCYC(SCYC_1, TA_NULL, 0, scyc_1_hdr, 2, 0, 15);
    /* システムアラームハンドラ */
    CRE_SALM(SALM_1, TA_NULL, 0, salm_1_hdr, 15);
}

/* アプリケーションパーティション */
PARTITION(PARTITION_A) {
    /* パーティションの属性 */
    SET_PAR_ATTR(TA_PAR_IDLE);
    /* パーティション初期化ルーチンと終了処理ルーチン */
    DEF_PARTITION_INI(TA_NULL, 1, par_init_a);
    DEF_PARTITION_TER(TA_NULL, 1, par_ter_a);
    /* タイムウィンドウの定義 */
    CRE_TWINDOW(TWIN_1_1, 50, 300);
    CRE_TWINDOW(TWIN_2_1, 50, 450);
    /* タスクの定義 */
    CRE_TSK(TASK_A1, { TA_NULL, 1, task, MID_PRI, ST_SIZE, NULL,SST_SIZE, NULL});
    CRE_TSK(TASK_B1, { TA_NULL, 2, task, MID_PRI, ST_SIZE, NULL, SST_SIZE, NULL});
    CRE_TSK(TASK_C1, { TA_NULL, 2, task, MID_PRI, ST_SIZE, NULL, SST_SIZE, NULL});
    /* チャネル */
    CRE_STVA(STVA_1, TA_NULL, 4);
    /* インタフェース */
    CRE_INF(IF_MSGQ_1_PAR_A, TA_OUT);
    CRE_INF(IF_STVA_1_PAR_A, TA_OUT);
```

```
    /* アトミックハンドラ */
    CRE_ATH(ATH_1, TA_NULL, ath_1, 100);
    /* パーティション例外ハンドラ */
    DEF_EXC(EXCNO_ZERODIV, zediv_hdr_para);
    /* アプリケーション割込みハンドラ */
    DEF_AINH(INHNO_DEV_A, TA_NULL, inh_dev_a, S_SIZE, TA_NULL, SS_SIZE, TANULL);
    CFG_INT(INHNO_DEV_A, {INTATR_DEV_A, INTPRI_DEV_A});
    /* タイムウィンドウ周期ハンドラ */
    CRE_TWCYC(TWCYCID_1, TA_NULL, 1, twcyc_1_hdr, 10, TWIN_1_1);
    /* タイムウィンドウアラームハンドラ */
    CRE_TWALM(TWALMID_1, TA_NULL, 1, twalm_1_hdr, TWIN_1_1);
}

PARTITION(PARTITION_B) {
    /* パーティションの属性 */
    SET_PAR_ATTR(TA_PAR_IDLE);
    /* パーティション初期化ルーチンと終了処理ルーチン */
    DEF_PARTITION_INI(TA_NULL, 1, par_init_b);
    DEF_PARTITION_TER(TA_NULL, 1, par_ter_b);
    /* タイムウィンドウの定義 */
    CRE_TWINDOW(TWIN_1_2, 350, 300);
    CRE_TWINDOW(TWIN_2_2, 500, 900);
    /* タスクの定義 */
    CRE_TSK(TASK_A1, { TA_NULL, 1, task, MID_PRI, ST_SIZE, NULL, SST_SIZE, NULL});
    CRE_TSK(TASK_B1, { TA_NULL, 2, task, MID_PRI, ST_SIZE, NULL, SST_SIZE, NULL});
    CRE_TSK(TASK_C1, { TA_NULL, 2, task, MID_PRI, ST_SIZE, NULL, SST_SIZE, NULL});
    /* インタフェース */
    CRE_INF(IF_MSGQ_1_PAR_B, TA_OUT);
    CRE_INF(IF_STVA_1_PAR_B, TA_IN);
    /* パーティション例外ハンドラ */
    DEF_EXC(EXCNO_ZERODIV, zediv_hdr_parb);
    /* アプリケーション割込みハンドラ */
    DEF_AINH(INHNO_DEV_B, TA_NULL, inh_dev_b, S_SIZE, TA_NULL, SS_SIZE, TANULL);
    CFG_INT(INHNO_DEV_B, {INTATR_DEV_B, INTPRI_DEV_B});
    /* タイムウィンドウ周期ハンドラ */
    CRE_TWCYC(TWCYCID_2, TA_STA, 2, twcyc_2_hdr, 2, TWIN_1_2);
}
```

```
PARTITION(PARTITION_C) {
    /* パーティションの属性 */
    SET_PAR_ATTR(TA_TANULL);
    /* パーティション初期化ルーチンと終了処理ルーチン */
    DEF_PARTITION_INI(TA_NULL, 1, par_init_c);
    DEF_PARTITION_TER(TA_NULL, 1, par_ter_c);
    /* タイムウィンドウの定義 */
    CRE_TWINDOW(TWIN_1_3, 650, 300);
    /* タスクの定義 */
    CRE_TSK(TASK_A1, { TA_NULL, 1, task, MID_PRI, ST_SIZE, NULL, SST_SIZE, NULL});
    CRE_TSK(TASK_B1, { TA_NULL, 2, task, MID_PRI, ST_SIZE, NULL, SST_SIZE, NULL});
    CRE_TSK(TASK_C1, { TA_NULL, 2, task, MID_PRI, ST_SIZE, NULL, SST_SIZE, NULL});
    /* チャネル */
    CRE_MSGQ(MSGQ_1, TA_NULL, 4, 6);
    /* インタフェース */
    CRE_INF(IF_MSGQ_1_PAR_C, TA_IN);
    CRE_INF(IF_STVA_1_PAR_C, TA_IN);
    /* パーティション例外ハンドラ */
    DEF_EXC(EXCNO_ZERODIV, zediv_hdr_parc);
    /* アプリケーション割込みハンドラ */
    DEF_AINH(INHNO_DEV_C, TA_NULL, inh_dev_c, S_SIZE, TA_NULL, SS_SIZE, TANULL);
    CFG_INT(INHNO_DEV_C, {INTATR_DEV_C, INTPRI_DEV_C});
}
```

以上.