カーネル勉強会

μITRON4.0 の概略

2001 年 8 月 20 日(月)

竹田良之

1．ITRON とは

ITRON は機器組み込み制御システム用のリアルタイム OS のアーキテクチャである。

2. μITRON4.0 仕様の策定にあたって

μITRON4.0 仕様の策定が必要な理由は、次の 4 点に集約されている。

(1) ソフトウェアの移植性の向上

近年の組み込みソフトウェアの大規模化･複雑化により、アプリケーションソフトウェアを異なる ITRON 仕様カーネルへ容易に移植できることが、従来よりも強く求められるようになってきた。また、ソフトウェア部品の流通を促すためにも、ITRON 仕様カーネル上に構築されるソフトウェアの移植性の向上は、極めて重要な課題となっている。

(2) ソフトウェア部品向けの機能の追加

従来の ITRON 仕様では、外販することを前提にソフトウェア部品を構築するには、機能的に不足している部分があった。例えば、ソフトウェア部品のあるサービスルーチンが、どのようなコンテキストから呼び出されたか調べる方法は、拡張レベルでしか用意されていなかった。

(3) 新しい要求･検討成果の反映

ITRON プロジェクトでは、ハードリアルタイムサポート研究会においてハードリアルタイムシステムの構築を容易にするためにリアルタイムカーネルが持つべき機能の検討を、RTOS 自動車応用技術委員会において自動車制御応用におけるリアルタイムカーネルに対する要求事項の整理を行ってきた。これらの新しい要求や検討成果を、リアルタイムカーネル仕様に反映する必要がある。

(4) 半導体技術の進歩への対応

μITRON3.0 仕様を策定してから約 6 年が経過しており、その間に半導体技術は大きく進歩し、組み込みシステムに用いられるプロセッサの性能向上やメモリ容量の増加も著しい。そのために、有用な機能であるが、μITRON3.0 仕様策定時にはオーバーヘッドが大きいため標準化を見送ったが、現在の技術では許容できるオーバーヘッドで実現できると考えられるものがある。

3. ITRON 仕様共通データ型

ITRON 仕様では、ソフトウェア技術者の教育を重視し、標準化を通じて一度覚えたことが広く活用できるよう配慮している。具体的には、使用における用語の使い方や、データ型、システムコールなどの名称の決め方などは、できる限り一貫性を持つように配慮し、教育用のテキストの作成にも力を入れている。

ITRON 仕様で使用されているデータ型一覧については付録 1 を参照してください。

4. 用語の説明

(1) タスクと自タスク

プログラムの並列実行の単位を「タスク」と呼び、サービスコールを呼び出したタスクを「自タスク」と呼ぶ。

(2) ディスパッチとディスパッチャ

プロセッサが実行するタスクを切り替えることを「ディスパッチ」と呼び、ディスパッチを実現するカーネル内の機構を「ディスパッチャ」と呼ぶ。

(3) スケジューリングとスケジューラ

次に実行すべきタスクを決定する処理を「スケジューリング」と呼び、スケジューリングを実現するカーネル内の機構を「スケジューラ」をと呼ぶ。

(4) コンテキスト

一般に、プログラムの実行される環境を「コンテキスト」と言う。

(5) 優先順位

処理が実行される順序を決める順序関係を「優先順位」と言う。

5. スケジューリング

(1)タスクの状態

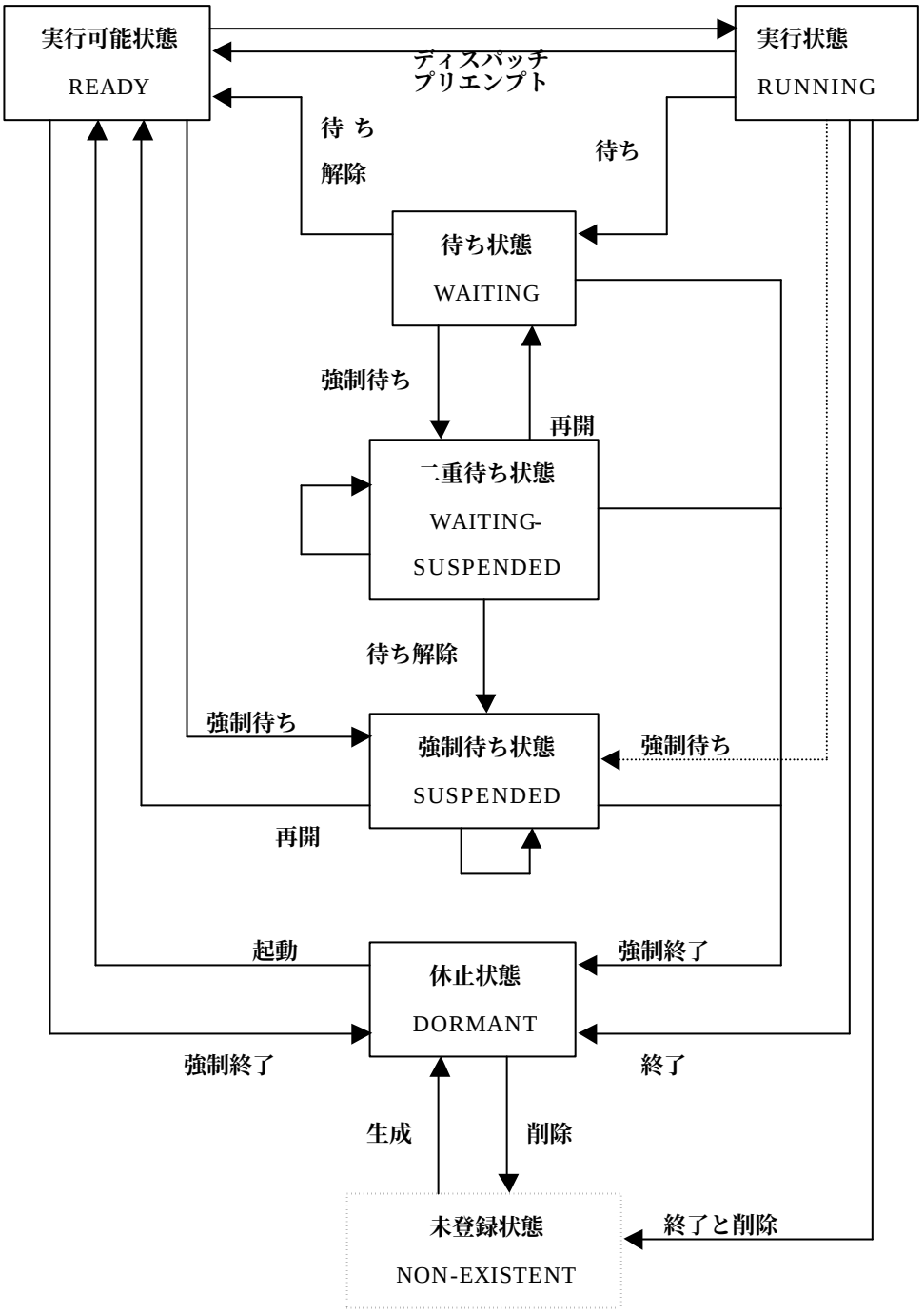


図 1．タスクの状態

実行状態：現在そのタスクを実行中であるという状態

実行可能状態：そのタスクを実行する準備は整っているが、そのタスクより優先度の高いタスクが実行中であるために、そのタスクを実行できない状態

広義の待ち状態：そのタスクを実行できる条件が整わないために、実行できない状態。何らかの条件が満

たされるのを待っている状態。

(a)待ち状態：何らかの条件が整うまで自タスクの実行を中断するサービスコールを呼び出したことにより、実行が中断された状態。

(b)強制待ち状態：他のタスクによって強制的に実行を中断させられた状態。

(c)二重待ち状態：待ち状態と強制待ち状態が重なった状態

休止状態：タスクがまだ起動されてないか、実行を終了した後の状態。

未登録状態：タスクがまだ生成されてないか、削除された後の、システムに登録されていない仮想的な状態。

(2)スケジューリング規則

ITRON 仕様では、タスクに与えられた優先度に基づくプリエンプティブな優先度ベースのスケジューリング方式を採用している。同じ優先度を持つタスク間では、FCFS（First Come First Served）方式によりスケジューリングを行う。

ITRON 仕様のスケジューリング規則では、優先順位の高いタスクが実行できる状態にある限り、それより優先順位の低いタスクはまったく実行されない。つまり、最も高い優先順位を持つタスクが待ち状態に入るなどの理由で実行できない状態とならない限り、他のタスクは実行されない。この点で、複数のタスクを公平に実行しようという TSS(Time Sharing System)のスケジューリング方式とは根本的に異なっている。但し、同じ優先度を持つタスク間の優先順位は、サービスコールを用いて変更することが可能であり、これを用いて TSS のスケジューリング方式を実現することが可能である。

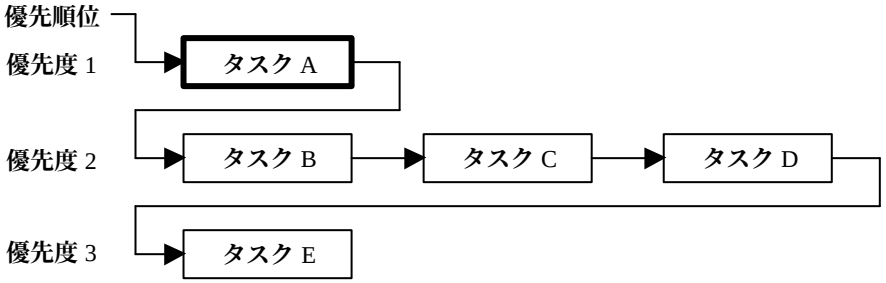
例：同じ優先度を持つタスクの実行順序

図 2(a)は優先度 1 のタスク A、優先度 3 のタスク E、優先度 2 のタスク B、タスク C、タスク D がこの順番で起動された後のタスク間の優先順位を示す。この状態では、最も優先度の高いタスク A が実行状態となっている。

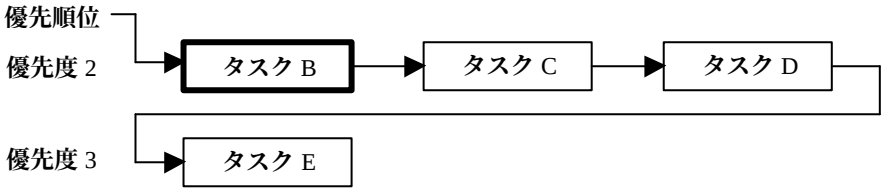
ここで、タスク A が終了すると、次に優先度の高いタスク B が実行状態に遷移する(図 2(b))。その後タスク A が再び起動されると、タスク B は中断され実行可能状態に戻るが、このときタスク B はタスク C とタスク D のいずれよりも先に実行できる状態になっていたことから、同じ優先度を持つタスクの中で最高の優先度を持つことになる。即ち、タスク間の優先順位は図 2(a)の状態に戻る。

次に図 2(b)の状態で、タスク B が待ち状態になった場合を考える。タスクの優先順位は実行できるタスクの間で定義されるため、タスク間の優先順位は図 2(c)の状態となる。その後タスク B が待ち解除されると、タスク B はタスク C とタスク D のいずれよりも後に実行できる状態になったことから、同じ優先度を持つタスクの中で最低の優先順位となる(図 2(d))。

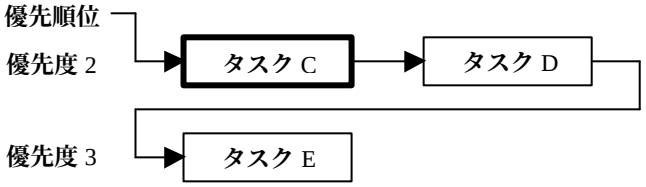
以上を整理すると、実行可能状態のタスクが実行状態になった後に実行可能状態に戻った直後には、同じ優先度を持つタスクの中で最高の優先順位を持っているのに対して、実行状態のタスクが待ち状態になった後に待ち解除されて実行できる状態になった直後には、同じ優先度を持つタスクの中で最低の優先順位になる。



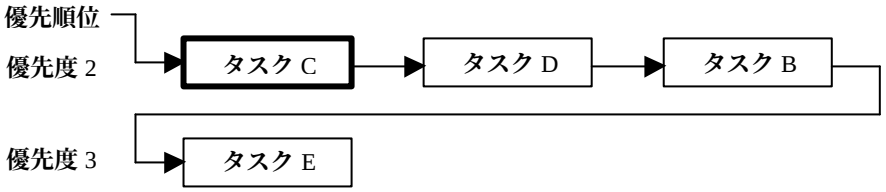
(a)最初の状態の優先順位



(b)タスク B が実行状態になった後の優先順位



(c)タスク B が待ち状態になった後の優先順位



(d)タスク B が待ち解除された後の優先順位

図 2 タスク間の優先順位

6. スタンダードプロファイル

ソフトウェアの移植性を向上させるためには、実装が備えるべき機能セットや、それぞれのサービスコールの機能仕様を、厳格に定める必要がある。言い換えると、使用の標準化の度合いを強くする必要がある。

一方、μITRON 仕様はこれまで、ソフトウェアの移植性よりも、ハードウェアやプロセッサへの適応化を可能にし、実行時のオーバーヘッドや使用メモリの削減を重視する「弱い標準化」の方針に基づいて標準化を行ってきた。この方針によりμITRON 使用は、8 ビットから 64 ビットまでの広い範囲のプロセッサに適用可能なスケーラビリティを実現しており、マイクロ ITRON 仕様が広く受け入れられている重要な理由の一つとなっている。ところが、ソフトウェア移植性の向上とスケーラビリティの実現は本質的に矛盾する面が多く、一つの仕様で両者を同時に実現することは困難である。

そこでμITRON4.0 仕様では、仕様全体としては「弱い標準化」の方針を維持しつつ、ソフトウェアの移植性を向上させるために、標準的な機能セットとその仕様を厳密に定めると言うアプローチを採用している。この標準的な機能セットを「スタンダードプロファイル」と呼ぶ。標準的な機能セットを定めるにあたっては、μITRON 仕様のカーネルの応用分野としては大き目のシステムを想定する。これは、一般に、システムの規模が大きいほど、ソフトウェアの移植性が重視されるためである。

スタンダードプロファイルを定義することにより、ソフトウェア部品などの移植性を重視するソフトウェアはスタンダードプロファイルの機能のみを用いて構築することを推奨し、逆にソフトウェアの移植性を重視する分野向けのカーネルはスタンダードプロファイルに準拠して実装することを推奨することになる。

さらに、スタンダードプロファイル内でも、考えられる限り、ソフトウェアの移植性を向上させるとともに、スケーラビリティも実現できるような仕様としている。具体的には、小さいオーバーヘッドで実現できる範囲で、割り込みハンドラの移植性が向上するような仕組みを導入している。例えば従来のμITRON 仕様では、割り込みハンドラの中でより優先度の高い割り込みハンドラが多重起動されるのを禁止するための、移植性を確保できる方法が用意されていなかったが、μITRON4.0 仕様ではこれを可能にしている。

スケーラビリティの実現に関しては、従来のμITRON 仕様と同様、サービスコールの単機能化などの方針により、豊富な機能を用意した上でライブラリリンクの機構で必要の無い機能がリンクされないような工夫を行っている。さらに、ライブラリリンクの機構だけでは必要な機能だけをリンクすることが難しい場面では、カーネルではより複雑な機能を実現するのに必要なプリミティブのみを提供するという方針を採っている。これによりカーネルに改造を加えず複雑な機能を実現することを可能にする一方、複雑な機能を必要としないアプリケーションでのオーバーヘッドを最小限にすることができる。

スタンダードプロファイルが想定するシステムイメージは、具体的には次のようなものである。

- ハイエンドの 16 ビットないしは 32 ビットプロセッサを使用。
- カーネルのプログラムサイズが 10～20KB 程度(全ての機能を使った場合)
- システム全体が一つのモジュールにリンクされる。
- カーネルオブジェクトは静的に生成される。

システム全体が一つのモジュールにリンクされることから、サービスコールはサブルーチンコールで呼び出すことになる。また、プロテクションの機構はもたない。

7. 静的 API

システムコンフィグレーションファイル中に記述し、カーネルやソフトウェア部品の構成を決定したり、オブジェクトの初期状態を定義するためのインターフェースを、静的 API と呼ぶ。ITRON 仕様では、静的 API の名称、機能、パラメータの種類･順序･名称･データ型を標準化する。

オブジェクトを登録するサービスコールなどに対して、それに対応する静的 API を規定する。サービスコールに対応する静的 API は、システムコンフィグレーションファイル中に記述された順序で、それぞれに対応するサービスコールをシステム初期化時に実行するのと等価な機能を持つ。また、サービスコールに対応しない静的 API､カーネルやソフトウェア部品で共通に利用する ITRON 仕様共通静的 API もある。

静的 API の名称は、同じ機能を持つサービスコールの名称を大文字で記述したものとしており、静的 API とサービスコールでパラメータを一致させている。ただし、パケットのポインタの変わりに、パケットの各要素の値を“{”と“}”の中に列挙する。

例：タスクの生成

静的 API

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

サービスコール

```
ER ercd = cre_tsk ( ID tskid, T_CTSK *pk_ctsk );
```

pk_ctsk の内容(T_CTSK 型)

ATR	tskatr	タスクの属性
VP_INT	exinf	タスクの拡張情報
FP	task	タスクの起動番地
PRI	itskpri	タスクの起動時優先度
SIZE	stksz	タスクのスタック領域のサイズ(バイと数)
VP	stk	タスクのスタック領域の先頭番地

(実装独自に他の情報を追加しても良い)

8. コンフィグレーション方法

カーネルやソフトウェア部品の構成やオブジェクトの初期状態を定義するためのファイルを、コンフィグレーションファイルと呼ぶ。システムコンフィグレーションファイルには、カーネルやソフトウェア部品の静的 API と ITRON 仕様共通静的 API に加えて、C 言語処理系のプリプロセッサディレクティブを記述することができる。システムコンフィグレーションファイル中の静的 API を解釈して、カーネルやソフトウェア部品を構成するためのツールを、コンフィグレータと呼ぶ。

システムコンフィグレーションファイルの処理手順は次のとおりである(図 3)。システムコンフィグレーションファイルは、まず、C 言語のプリプロセッサに通される。次に、ソフトウェア部品のコンフィグレータによって順に処理され、最後にカーネルのコンフィグレータによって処理される。

ソフトウェア部品のコンフィグレータは、渡されたファイル中に含まれる自分自身に対する静的 API と共通静的 API を解釈し、自分自身の構成や初期化に必要なファイルを C 言語のソースファイルの形で、ID 自動割付け結果ヘッダファイルを C 言語のヘッダファイルの形で生成する。また、渡されたファイルから自分自身に対する静的 API を取り除き、以降のコンフィグレータに対する静的 API を追加し、次のコンフィグレータに渡す。

カーネルコンフィグレータは、渡されたファイル中の全ての静的 API を解釈し、自分自身の構成や初期化に必要なファイルを C 言語のソースファイルの形で、ID 自動割付け結果ヘッダファイルを C 言語のヘッダファイルの形で生成する。自分自身に対する静的 API または共通静的 API として解釈できない記述が含まれている場合には、エラーを報告する。

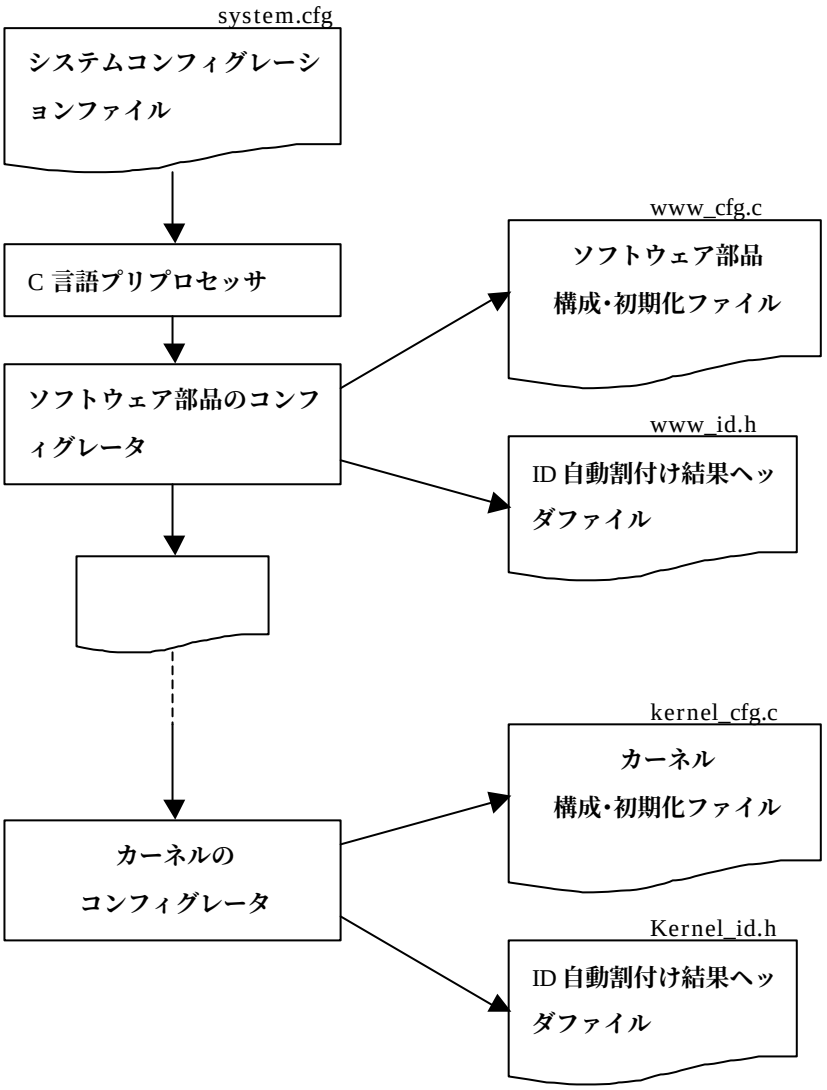


図 3．システムコンフィグレーションファイルの処理手順

付録 1．データ型一覧

B	符号付き 8 ビット整数
H	符号付き 16 ビット整数
W	符号付き 32 ビット整数
D	符号付き 64 ビット整数
UB	符号無し 8 ビット整数
UH	符号無し 16 ビット整数
UW	符号無し 32 ビット整数
UD	符号無し 64 ビット整数
VB	データタイプの定まらない 8 ビットの値
VH	データタイプの定まらない 16 ビットの値
VW	データタイプの定まらない 32 ビットの値
VD	データタイプの定まらない 64 ビットの値
VP	データタイプの定まらないものへのポインタ
FP	プログラムの起動番地
INT	プロセッサに自然なサイズの符号付き整数
UINT	プロセッサに自然なサイズの符号無し整数
BOOL	真偽値（TRUE または FALSE）
FN	機能コード(符号付き整数)
ER	エラーコード(符号付き整数)
ID	オブジェクトの ID 番号(符号付き整数)
ATR	オブジェクトの属性(符号無し整数)
STAT	オブジェクトの状態(符号無し整数)
MODE	サービスコールの動作モード(符号無し整数)
PRI	優先度(符号付き整数)
SIZE	メモリの領域サイズ(符号無し整数)
TMO	タイムアウト指定(符号付き整数、時間は実装定義)
RELTIM	相対時間(符号無し整数、時間は実装定義)
SYSTIM	システム時刻(符号無し整数、時間は実装定義)
VP_INT	データタイプの定まらないものへのポインタまたはプロセッサに自然なサイズの符号付き整数
ER_BOOL	エラーコードまたは真偽値(符号付き整数)
ER_ID	エラーコードまたは ID 番号(符号付き整数、負の ID 番号は表現できない)
ER_UINT	エラーコードまたは符号無し整数(符号付き整数、符号無し整数を表現する場合の有効ビット数は UINT より 1 ビット少ない)

付録 2．静的 API 一覧

ATT_INI	初期化ルーチンの追加
ATT_ISR	割り込みサービスルーチンの追加
CRE_ALM	アラームハンドラの生成
CRE_CYC	周期ハンドラの生成
CRE_DTQ	データキューの生成
CRE_FLG	イベントフラグの生成
CRE_MBF	メッセージバッファの生成
CRE_MBX	メールボックスの生成
CRE_MPF	固定長メモリプールの生成
CRE_MPL	可変長メモリプールの生成
CRE_MTX	ミューテックスの生成
CRE_POR	ランデブポートの生成
CRE_SEM	セマフォの生成
CRE_TSK	タスクの生成
DEF_EXC	CPU 例外ハンドラの定義
DEF_INH	割り込みハンドラの定義
DEF_OVR	オーバランハンドラの定義
DEF_SVC	拡張サービスコールの定義
DEF_TEX	タスク例外処理ルーチンの定義
INCLUDE	ファイルのインクルード

付録３．スタンダードプロファイルでサポートする静的 API とサービスコール

CRE_TSK	タスクの生成(静的 API)
act_tsk／iact_tsk	タスクの起動
can_tsk	タスク起動要求のキャンセル
ext_tsk	自タスクの終了
ter_tsk	タスクの強制終了
chg_pri	タスクの優先度の変更
get_pri	タスクの優先度の参照
slp_tsk	起床待ち
tslp_tsk	起床待ち(タイムアウトあり)
wup_tsk／iwup_tsk	タスクの起床
can_wup	タスクの起床要求のキャンセル
rel_wai／irel_wai	待ち状態の強制解除
sus_tsk	強制待ち状態への移行
rsm_tsk	強制待ち状態からの再開
frsm_tsk	強制待ち状態からの強制再開
dly_tsk	自タスクの遅延
DEF_TEX	タスク例外処理ルーチンの定義(静的 API)
ras_tex／iras_tex	タスク例外処理の要求
dis_ tex	タスク例外処理の禁止
ena_tex	タスク例外処理の許可
sns_tex	タスク例外処理禁止状態の参照
CRE_SEM	セマフォの生成(静的 API)
sig_sem／isig_sem	セマフォ資源の返却
wai_sem	セマフォ資源の獲得
pol_sem	セマフォ資源の獲得(ポーリグ)
twai_sem	セマフォ資源の獲得(タイムアウトあり)
CRE_FLG	イベントフラグの生成(静的 API)
set_flg／iset_flg	イベントフラグのセット
clr_flg	イベントフラグのクリア
wai_flg	イベントフラグ待ち
pol_flg	イベントフラグ待ち(ポーリグ)
tawi_flg	イベントフラグ待ち(タイムアウトあり)
DEF_EXC	CPU 例外ハンドラの定義(静的 API)
ATT_INI	初期化ルーチンの追加(静的 API)

CRE_DTQ	データキューの生成(静的 API)
snd_dtq	データキューへの送信
psnd_dtq/ ipsnd_dtq	データキューへの送信(ポーリグ)
tsnd_dtq	データキューへの送信(タイムアウトあり)
fsnd_dtq /ifsnd_dtq	データキューへの強制送信
rcv_dtq	データキューからの受信
prcv_dtq	データキューからの受信(ポーリグ)
trcv_dtq	データキューからの受信(タイムアウトあり)
CRE_MBX	メールボックスの生成(静的 API)
snd_mbx	メールボックスへの送信
rcv_mbx	メールボックスからの受信
prcv_mbx	メールボックスからの受信(ポーリグ)
trcv_mbx	メールボックスからの受信(タイムアウトあり)
CRE_MPF	固定長メモリプールの生成(静的 API)
get_mpf	固定長メモリプールの獲得
pget_mpf	固定長メモリプールの獲得(ポーリグ)
tget_mpf	固定長メモリプールの獲得(タイムアウトあり)
rel_mpf	固定長メモリプールの返却
set_tim	システム時刻の設定
get_tim	システム時刻の参照
isig_tim	タイムティックの供給
CRE_CYC	周期ハンドラの生成(静的 API)
sta_cyc	周期ハンドラの動作開始
stp_cyc	周期ハンドラの動作停止
rot_rdq/irot_rdq	タスクの優先順位の回転
get_tid/iget_tid	実行状態のタスク ID の参照
loc_cpu/iloc_cpu	CPU ロック状態への移行
nul_cpu/inul_cpu	CPU ロック状態の解除
dis_dsp	ディスパッチの禁止
ena_dsp	ディスパッチの許可
sns_ctx	コンテキストの参照
sns_loc	CPU ロック状態の参照
sns_dsp	ディスパッチ禁止状態の参照
sns_dpn	ディスパッチ保留状態の参照
DEF_INH	割り込みハンドラの定義(静的 API)