リアルタイムカーネル勉強会
タスク管理機能

飯山 真一 (shin@ertl.ics.tut.ac.jp)
豊橋技術科学大学 情報工学専攻
平成 13 年 9 月 3 日（月)

1 概要

ここでは，JSP カーネルにおけるタスク管理機能について解説する．タスク管理機能には，./kernel/ 以下の task.h，task.c，queue.h，task_manage.c の 4 つのファイルが主に関連する．task.h や task.c には，タスク制御ブロックやタスクの状態遷移に関する操作が定義されていて，具体的なサービスコールは，./kernel/task_manage.c に宣言されている．レディキューなどのキュー操作に関する queue.h に定義されている．

2 タスク管理機能

タスク管理機能は，タスクの状態を直接的に操作／参照するための機能である．タスクを生成／削除する機能，タスクを起動／終了する機能，タスクに対する起動要求をキャンセルする既往，タスクの優先度を変更する機能，タスクの状態を参照する機能が含まれる．タスクは ID 番号で識別されるオブジェクトである．([仕様書],p.77)

2.1 サービスコール

スタンダードプロファイルでは，タスクを動的に生成／削除する機能，起動コードを指定してタスクを起動する機能，タスクを終了と同時に削除する機能，タスクの状態を参照する機能を除いて，タスク管理機能をサポートしなければならない．([仕様書],p.79)

つまり，スタンダードプロファイルでは，タスク管理機能として，表 1 に示すサービスコールをサポートしなければならない．

表 1: サービスコール (タスク管理機能)

サービスコール名	機能
(CRE_TSK)	タスクの生成（静的 API）
act_tsk, iact_tsk	タスクの起動
can_act	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	タスク優先度の変更
get_pri	タスク優先度の参照

3　タスク制御ブロックの実装

タスクを管理するための主な情報は，タスクの状態，タスクの優先度などがある．これらの管理情報はタスク制御ブロック (TCB) と呼ばれる領域に，タスクごとにまとめて格納される．

3.1　タスク状態

JSP カーネルでは、タスクの状態として，実行状態 (RUNNING)，実行可能状態 (READY)，待ち状態 (WAITING)，二重待ち状態 (WAITING-SUSPENDED)，強制待ち状態 (SUSPENDED)，休止状態 (DORMANT) の 6 通りが存在する[1] (図 2)．このうち，タスク管理機能として深く関わるタスク状態は，RUNNING，READY，DORMANT の 3 つである．

JSP カーネルでは，TCB 中のタスク状態として，実行状態 (RUNNING) と実行可能状態 (READY) は区別しない．また，二重待ち状態は，(TS_WAITING | TS_SUSPENDED) で表す．TS_WAIT_???? は，待ち状態に伴う付属状態を表し，待ち状態（二重待ち状態を含む）の場合にのみ設定される．

```
/* 休止状態 */
#define TS_DORMANT      0x00
/* 実行できる状態 */
define TS_RUNNABLE      0x01
/* 待ち状態 */
#define TS_WAITING      0x02
/* 強制待ち状態 */
#define TS_SUSPENDED    0x04
/* 起床待ち状態 */
#define TS_WAIT_SLEEP   0x08
/* 同期・通信オブジェクトに対する待ち状態 */
#define TS_WAIT_WOBJ    0x10
/* 共通部分の待ちキューにつながっている */
#define TS_WAIT_WOBJCB  0x20
```

図 1: JSP でサポートするタスクの内部状態

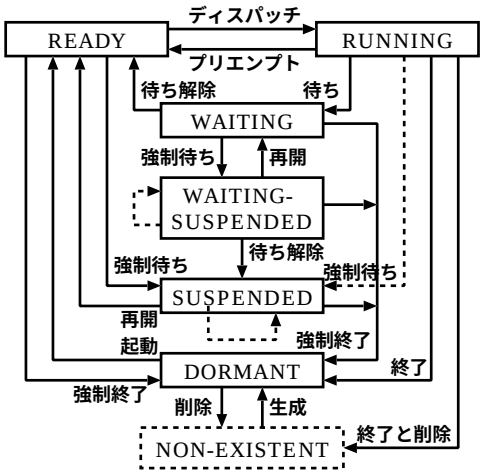

図 2: タスクの状態遷移

(参考) **タスクのスケジューリング規則**

ITRON 仕様でのタスクのスケジューリングは，タスクの優先度を基準にして使われ，実行可能状態にあるタスク中で最も高い優先度をもつタスクが実行される．ITRON 仕様では，優先度ベースのスケジューリングを採用しており，高い優先度のタスクの実行中は，それより低い優先度のタスクは全く実行されない．逆に，低い優先度のタスクの実行中は，高い優先度のタスクが実行可能状態になると，実行中のタスクは中断され，高い優先度のタスクが実行される．

3.2　各種定数の定義

JSP（Just Standard Profile) では，スタンダードプロファイルに準拠のための**最小限の機能**をサポートしている．各種定数の値も最小限となるように定義されている．

[1] 過渡的な状態を含まない

スタンダードプロファイルでは，少なくとも 1〜16 の 16 段階のタスク優先度をサポートしなければならない．([仕様書] p.27)

```
#define TMIN_TPRI  1   /* タスク優先度の最小値 */
#define TMAX_TPRI  16  /* タスク優先度の最大値 */
```

スタンダードプロファイルでは，少なくとも 1 回のタスクの起動要求キューイングをサポートしなければならない．したがって，TMAX_ACTCNT は 1 以上でなければならない．([仕様書] p.27)

スタンダードプロファイルでは，少なくとも 1 回のタスクの起床要求キューイングをサポートしなければならない．また，タスクの強制待ち状態をサポートしなければならない．したがって，TMAX_WUPCNT と TMAX_SUSCNT は，ともに 1 以上でなければならない．([仕様書] pp.99-100)．

```
#define TMAX_ACTCNT  1  /* 起動要求キューイング数の最大値 */
#define TMAX_WUPCNT  1  /* 起床要求キューイング数の最大値 */
#define TMAX_SUSCNT  1  /* 強制待ち要求ネスト数の最大値 */
```

3.3 タスク初期化ブロック

タスクに関する情報を，値が変わらないために ROM に置ける部分（タスク初期化ブロック）と，値が変化するために RAM に置かなければならない部分（タスク制御ブロック，TCB）に分離し，TCB 内に対応するタスク初期化ブロックを指すポインタを入れる．([JSP],task.h)

```
typedef struct task_initialization_block {
        ATR     tskatr;         /* タスク属性 */
        VP_INT  exinf;          /* タスクの拡張情報 */
        FP      task;           /* タスクの起動番地 */
        UINT    ipriority;      /* タスクの起動時優先度（内部表現） */
        SIZE    stksz;          /* スタック領域のサイズ（丸めた値） */
        VP      stk;            /* スタック領域の先頭番地 */
        ATR     texatr;         /* タスク例外処理ルーチン属性 */
        FP      texrtn;         /* タスク例外処理ルーチンの起動番地 */
} TINIB;
```

タスク初期化ブロックはコンフィギュレータにより生成される．例えば，次の静的 API をコンフィギュレータで処理すると，

```
CRE_TSK(TASK1, { TA_HLNG, (VP_INT) 1, task, MID_PRIORITY, STACK_SIZE, NULL });
```

カーネル初期化ブロックの初期値は次のように定義される．

```
const TINIB _kernel_tinib_table[TNUM_TSKID] = {
        {TA_HLNG, (VP_INT) 1, task, INT_PRIORITY(MID_PRIORITY),
STACK_SIZE, _stack1, TA_HLNG, tex_routine}
};
```

3.4 タスク制御ブロック (TCB)

JSP では，TCB は図 3 のように定義されている．大きな特徴としては，タスクキューが先頭に配置されていることである．これにより，TCB の内部構造とキュー操作を独立して扱うことができる．

```
typedef struct task_control_block {
    QUEUE   task_queue;     /* タスクキュー */
    const TINIB *tinib;     /* タスク初期化ブロックへのポインタ */

    UINT    tstat : TBIT_TCB_TSTAT;         /* タスク状態（内部表現）*/
    UINT    priority : TBIT_TCB_PRIORITY;   /* 現在の優先度（内部表現）*/
    BOOL    actcnt : 1;                     /* 起動要求キューイング */
    BOOL    wupcnt : 1;                     /* 起床要求キューイング */
    BOOL    enatex : 1;                     /* タスク例外処理許可状態 */

    TEXPTN  texptn;         /* 保留例外要因 */
    WINFO   *winfo;         /* 待ち情報ブロックへのポインタ */
    CTXB    tskctxb;        /* タスクコンテキストブロック */
} TCB;
```

図 3: タスク制御ブロック (TCB)

TCB のサイズを小さく押えることは，メモリサイズの点で有利である[2]．TCB のサイズを小さく押えるために，各メンバ変数をビットフィールド指定付きで定義している．

JSP では，タスクの起動要求キューイング数の最大値（TMAX_ACTCNT）と起動要求キューイング数の最大値（TMAX_WUPCNT）は 1 に固定されているため，キューイングされているかどうかの真偽値で表現することができる．また，強制待ち要求ネスト数の最大値（TMAX_SUSCNT）が 1 に固定されているので，強制待ち要求ネスト数（suscnt）は必要ない．([JSP],task.h)

また，タスク状態は 6 ビットで表現できるし，優先度は 4 ビット（16 段階) で十分である[3]．

```
#define TBIT_TCB_TSTAT     6  /* tstat フィールドのビット幅 */
#define TBIT_TCB_PRIORITY  4  /* priority フィールドのビット幅 */
```

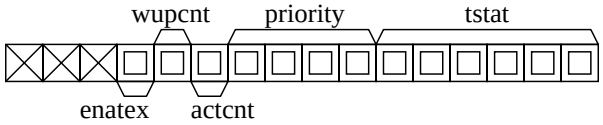

図 4: TCB のビットフィールド指定部の例

すべてのタスクのタスク制御ブロックやタスク初期化ブロックは，コンフィギュレータにより生成される kernel_cfg.c 内で静的に確保および定義されるが，タスクの状態などは，関数 task_initialize で初期化される．タスクの生成後の状態のデフォルトは休止状態であり，タスク初期化ブロック内で TA_ACT が指定されている場合は，実行可能状態とされる．

[2] システムに依存して，実行性能が悪くなる可能性がある
[3] sh では， TBIT_TCB_PRIORITY を 8 で指定しているので，図 4 のようにパディングされない．

4 レディキューの構造

レディキューは，実行できる状態のタスクを管理するためのデータ構造で，タスクスケジューリングを効率良く実現するためには，その構造に工夫が必要となる．本節では，JSP で実現されているレディキューのデータ構造と，効率良く実現するための工夫を示す．

レディキューを実現するためには，まずその基本となるキューのデータ構造とキューの操作を定義する必要がある．JSP では **queue.h** で定義されている[4] ．

4.1 キューのデータ構造

JSP では，循環型の構造をもつダブルリンクキューが採用されている．キューのデータ構造を**図** 5 に示す．

```
typedef struct queue {
 struct queue *next; //次のエントリ
 struct queue *prev; //前のエントリ
} QUEUE;
```

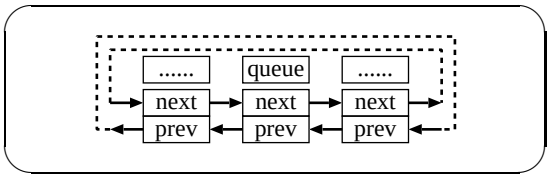

図 5: キューのデータ構造

4.2 キュー操作関数

キューの操作するための関数として，queue_initialize (キューの初期化)，queue_insert_prev (キューの前エントリへの挿入)，queue_delete(エントリの削除)，queue_delete_next (キューの次エントリの取出し)，queue_empty (キューが空かどうかのチェック) の 5 つが用意されている．

- `Inline void queue_initialize(QUEUE *queue)`
  キューの初期化を行う関数．指定された queue の next と prev を queue 自身に向けることで，キューの初期化を行う．

- `Inline void queue_insert_prev(QUEUE *queue, QUEUE *entry)`
  queue の前に entry を挿入する．queue にキューヘッダを指定した場合には，キューの末尾に entry を挿入することになる．([JSP],task.h)

- `Inline void queue_delete(QUEUE *entry)`
  キューから entry を削除するための関数．entry に前後にあるエントリのポインタをそれぞれのエントリに向けさせることで，entry をキューから外す．entry のポインタは操作していない．

- `Inline QUEUE * queue_delete_next(QUEUE *queue)`
  キューの次エントリを取出すための関数．レディキューでは，最高優先度のタスクキューの先頭にある TCB を取り出す操作などで使われる．queue の次エントリをキューから削除し，削除したエントリを返す．queue にキューヘッダを指定した場合には，キューの先頭のエントリを取り出すことになる．queue に空のキューを指定して呼び出してはならない．([JSP],task.h) queue に空のキューを指定した場合を，アサートにかけている．

[4] ここで定義されたキューはレディキューだけでなく，待ちキューなどにも使われている．

- `Inline BOOL queue_empty(QUEUE *queue)`
  キューが空かどうかのチェックするための関数．queue には，キューヘッダを指定する．queue に指定したキューの next が自分自身を指していれば，そのキューは空であることが分かる．next は自分自身を指しているのに，prev が指していない場合は，アサートをかけている．

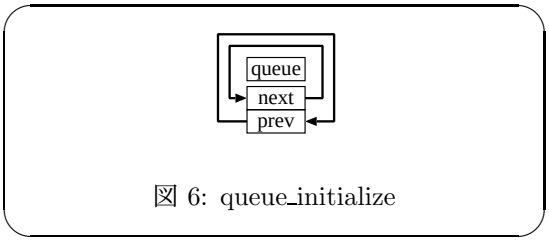

図 6: queue_initialize

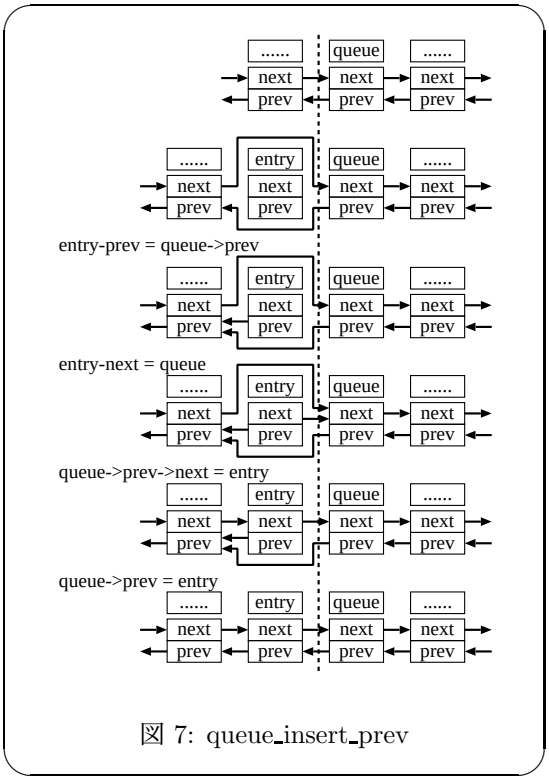

図 7: queue_insert_prev

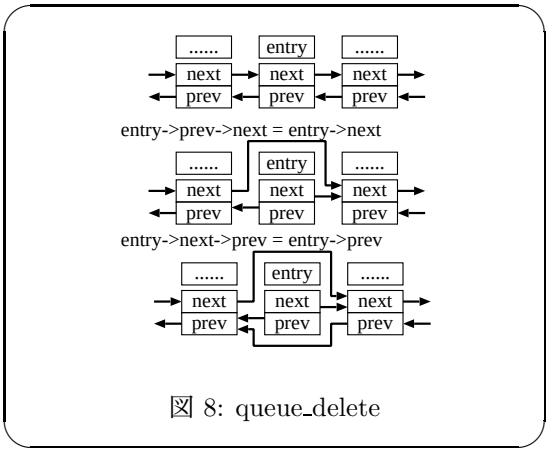

図 8: queue_delete

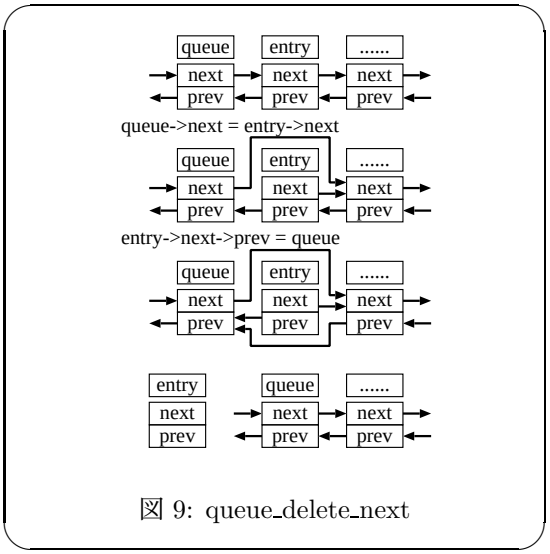

図 9: queue_delete_next

4.3 レディキュー

JSP のレディキューは，優先度ごとのタスクキューで構成されている (図 10)．タスクの TCB は，該当する優先度のキューに登録される．([JSP],task.c)

```
static QUEUE    ready_queue[TNUM_TPRI];
```

μITRON 仕様では，実行可能状態にあるタスクの中で最も高い優先順位をもつタスク，つまりレディキューの先頭のタスクが，占有的に実行状態になることが許される．同じ優先度をもつタスク同士では，First Come First Served 方式でスケジューリングされる．

実行できるタスクが複数ある場合には，その中でも最も優先順位の高いタスクが実行状態となり，他は実行可能状態となる．タスク間の優先順位は，異なる優先度を持つタスク間では，高い優先度を持つタスクの方が高い優先順位を持つ．同じ優先度を持つタスク間では，先に実行できる状態になったタスクの方が高い優先順位を持つ．([仕様書],p.53)

レディキューの先頭のタスクが最高優先順位のタスク (schedtsk) である．最高優先順位のタスク (schedtsk) が，実行状態 (runtsk) になる．最高優先順位のタスク (schedtsk) が変更された場合は，ディスパッチ要求フラグ (reqflg) を立てる．システムがディスパッチ許可状態になった時点で，ディスパッチャは実行状態のタスク (runtsk) を最高優先順位のタスク (schedtsk) にする．システムがディスパッチ許可状態であるかは，enadsp で判断する．

```
TCB   *runtsk;   /* 実行状態のタスク */
TCB   *schedtsk; /* 最高優先順位のタスク */
BOOL  reqflg;    /* タスクディスパッチ／タスク例外処理ルーチン起動要求フラグ */
BOOL  enadsp;    /* タスクディスパッチ許可状態 */
```

レディキューの効率化

レディキューのサーチを効率よく行うために，優先度ごとのタスクキューにタスクが入っているかどうかを示すビットマップを用意している．ビットマップを使うことで，メモリアクセスの回数を減らすことができるが，ビット操作命令が充実していないプロセッサで，優先度の段階数が少ない場合には，ビットマップ操作のオーバーヘッドのために，逆に効率が落ちる可能性もある．([JSP],task.c)

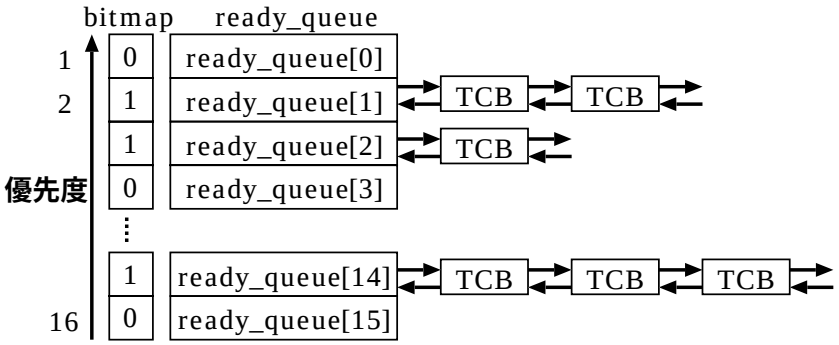

図 10: レディーキューの構造とビットマップ

レディキューを扱う関数として，次のものが用意されている．

- `BOOL rotate_ready_queue(UNIT pri)`
  レディキューの回転を行うための関数である．pri に指定した優先度のタスクキューの回転させる．pri に最高優先順位のタスクと同じ優先度が指定された場合，つまり，最高優先順位のタスクがタスクキューの末尾に移動する場合は最高優先度のタスクを更新する．

  タスクキューの中身が空であったり，最高優先順位のタスクしかなかった場合は，何もしない．そうでない場合は，最高優先順位のタスクをタスクキューの末尾に移動する．

5 タスク管理機能の実装

5.1 状態遷移のための関数

状態遷移のための関数として，make_???? 関数が用意されている．タスクが状態遷移をすることで，最高優先順位のタスクを更新する必要が発生する場合もある．

- `BOOL make_runnable(TCB *tcb)`
  tcb に指定したタスクを実行可能状態へ移行する．最高優先順位のタスクを更新するのは，次のときである．
  - 実行できるタスクがなかった場合
  - tcb の優先度が最高優先順位のタスクの優先度よりも高い場合
- `BOOL make_non_runnable(TCB *tcb)`
  tcb に指定したタスクを実行可能状態から他の状態へ移行する．最高優先順位のタスクを更新するのは，次のときである．
  - tcb が最高優先順位のタスクであった場合
- `void make_dormant(TCB *tcb)`
  tcb で指定したタスクを休止状態へ移行する．具体的には，TCB の中身を初期状態にする．
- `BOOL make_active(TCB *tcb)`
  tcb で指定したタスクを実行可能状態へ移行する．中で，make_runnable を呼ぶ．
- `BOOL change_priority(TCB *tcb, UINT newpri)`
  tcb がレディキューに接続されている場合に，その優先度を変更するための関数．指定された tcb と最高優先順位のタスクとの関係などで，動作が異なる．
  - 最高優先順位のタスク (schedtsk) が指定されたタスク (tcb) であって，優先度が下がった場合．この場合はレディキューを改めてサーチする必要がある．
  - 最高優先順位のタスク (schedtsk) が指定されたタスク (tcb) とは異なる場合で，指定されたタスクの優先度が最高優先順位のタスクよりも高い場合．この場合は，指定されたタスクが新たな最高優先順位のタスクとなるので，レディキューをサーチする必要はない．
- `void exit_task()`
  実行状態のタスクの終了．まず，実行可能状態以外の状態に移行して，そのあと休止状態に移行する．起動要求がキューイングされていれば，再び実行可能状態に移行する．そして，ディスパッチャが起動される．

5.2 サービスコールの機能の概略

- `ER act_tsk(ID tskid)`
  tskid で指定されるタスクを起動する．具体的には，対象タスクを休止状態から実行可能状態に移行させ，タスクの起動時に行うべき処理を行う．対象タスクが休止状態でない場合には，タスクに対する起動要求をキューイングする．tskid に自タスク指定 (TSK_SELF) 可能．

- `ER_UINT can_tsk(ID tskid)`
  tskid で指定されるタスクに対してキューイングされている起動要求をキャンセル (起動要求キューイング数を 0 に) し，キューイングされていた起動要求の回数を返す．tskid に自タスク指定 (TSK_SELF) 可能．

- `void ext_tsk()`
  自タスクを終了させる．具体的には，自タスクを休止状態に移行させ，タスクの終了時に行うべき処理を行う．

  ここでは，使用中のスタック領域が activate_context(./config/sh3/cpu_context.h) で破壊されないように，現在使用しているスタック上に alloca でダミー領域を確保して保護している．実際の操作は，exit_tsk で行われる．

- `ER ter_tsk(ID tskid)`
  tskid で指定されるタスクを強制的に終了させる．具体的には，対象タスクを休止状態にさせ，タスクの終了時に行うべき処理を行う．対象タスクが休止状態の時は E_OBJ エラーを返す．また．このサービスコールで自タスクを終了させることはできない．対象タスクが自タスクの場合には，E_ILUSE 返す．([仕様書],p.90)

- `ER chg_pri(ID tskid, PRI tskpri)`
  tskid で指定されるタスクのベース優先度を，tskpri で指定される値に変更する．tskid で指定されたタスクがレディキューや待ちキューにつながっている場合は，それぞれのキューに優先度の変更を反映しなけらばならない．例えば，レディキューにつながっている場合は，change_priority が呼ばれる．

5.3 エラーチェック

各サービスコールに渡されたパラメータをチェックするためのマクロが kernel/check.h に定義されている．

- `CHECK_TSKID(tskid)`, `CHECK_TSKID_SELF(tskid)`
  tskid に指定されたタスク ID が有効であるかをチェックする．具体的には，tskid が TMIN_TSKID から tmax_tskid までの範囲に入っていなければ，E_ID を返す．入っていれば何もしない．CHECK_TSKID_SELF は，有効な範囲に自タスク指定である TSK_SELF を含めたもの．

- `CHECK_NONSELF(tcb)`
  tcb に指定されたタスクが自タスクであるかをチェックする．具体的には，tcb が実行状態にあるタスクであれば，E_ILUSE を返す．そうでなければ何もしない．

- `CHECK_TSKCTX_UNL()`
  現在のシステムの状態をチェックする．具体的には，現在のコンテキストが非タスクコンテキストであるか，もしくは，CPU ロック状態にあれば，E_CTX を返す．そうでない場合は何もしない．

- `CHECK_INTCTX_UNL()`
  現在のシステムの状態をチェックする．具体的には，現在のコンテキストがタスクコンテキストであるか，もしくは，CPU ロック状態にあれば，E_CTX を返す．そうでない場合は何もしない．

- `CHECK_TPRI_INI(tpri)`
  tpri に指定された優先度が有効であるかをチェックする．具体的には，tpri が TPRI_INI(起動優先

度) でなくて，かつ，TMIN_TPRI から TMAX_TPRI までの範囲に入ってなければ，E_PAR を返す．そうでない場合は何もしない．

参考文献

[仕様書] 坂村 健監修，高田 広章編:μITRON4.0 仕様 Ver．4.00.00(1999)

[JSP] TOPPERS/JSP kernel v1.1, Embedded and Real-Time Systems Laboratory, Toyohashi Univ. of Technology, JAPAN