株式会社 ヴィッツ

2001 年 9 月 17 日

# カーネル勉強会資料

プロセッサ非依存部６

＊＊＊ 割込み管理 システム状態管理機能 タスク例外処理機能 ＊＊＊

株式会社ヴィッツ　森川 聡久

morikawa@witz-inc.co.jp

## 1　関連ファイル

| | |
|---|---|
| ./kernel/interrupt.h | 割込み管理機能 |
| ./kernel/interrupt.c | 割込み管理機能 |
| ./kernel/exception.h | CPU 例外管理機能 |
| ./kernel/exception.c | CPU 例外管理機能 |
| ./kernel/task_except.c | タスク例外処理機能 |
| ./kernel/sys_manage.c | システム状態管理機能 |

## 2　システム状態管理機能

### 2.1　システム状態の種類

- タスクコンテキストと非タスクコンテキスト
  - SH3 では割込みネスト回数が 1 以上なら、非タスクコンテキスト
- CPU ロック状態とロック解除状態
  - SH3 では最高優先度の割込み(MAX_IPM)かを判断
- ディスパッチ禁止状態と許可状態
  - enadsp フラグ
- ディスパッチ保留状態
  - 非タスクコンテキストのとき
  - CPU ロック状態のとき
  - ディスパッチ禁止状態のとき

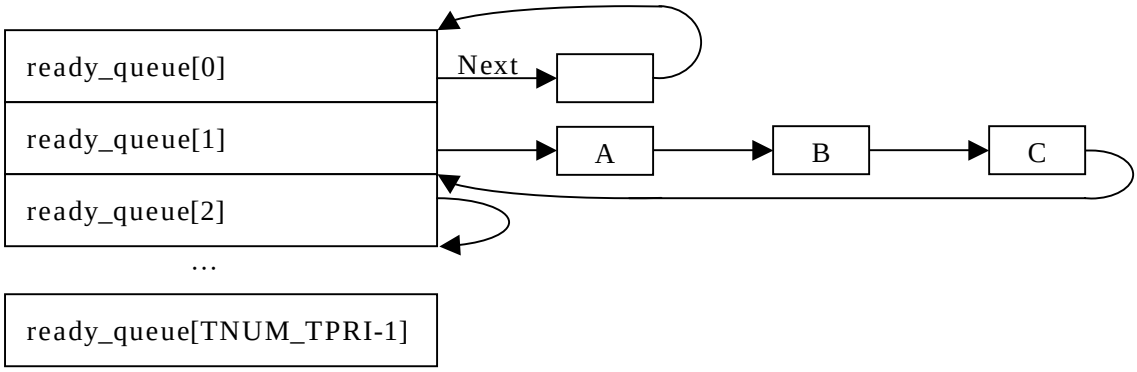
### 2.2　サービスコール一覧（sys_manage.c）

| ｻｰﾋﾞｽｺｰﾙ | 機能 | 備考 |
|---|---|---|
| rot_rdq | タスクの優先順位の回転 | タスクコンテキスト用 |
| irot_rdq | タスクの優先順位の回転 | 非タスクコンテキスト用 |
| get_tid | 実行状態のタスク ID の参照 | タスクコンテキスト用 |

| | | |
|---|---|---|
| iget_tid | 実行状態のタスク ID の参照 | 非タスクコンテキスト用 |
| loc_cpu | CPU ロック状態への移行 | タスクコンテキスト用 |
| iloc_cpu | CPU ロック状態への移行 | 非タスクコンテキスト用 |
| unl_cpu | CPU ロック状態の解除 | タスクコンテキスト用 |
| iunl_cpu | CPU ロック状態の解除 | 非タスクコンテキスト用 |
| 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 | |

### 2.3 レディーキュー

- 優先度毎に管理
- リング状に連なっている
- rotate_ready_queue()で、指定優先度の最高優先順位タスクを最後に変更できる。
  (ex) A,B,C の優先順位であるものを、B,C,A とする。




## 3 タスク例外処理機能

### 3.1 タスク例外処理について

- タスク毎に１つのタスク例外処理ルーチンを登録することができる。
- タスク例外処理要求時に、タスク例外処理ルーチンを実行する。

### 3.2 タスク例外処理情報

#### 3.2.1 タスク初期化ブロック

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;
```

タスクに関する情報のうち、値が変わらないために ROM に置ける部分をタスク初期化ブロックとした。タスク初期化ブロックには，DEF_TEX で定義されるタスク例外処理ルーチンに関する情報も含む。TCB にタスク初期化ブロックへのポインタをもつ。

```
CRE_TSK(tskid, { tskatr, exinf, task, itskpri, stksz, stack });
DEF_TEX(tskid, { texatr, texrtn });
```

texatr には、（TA_HLNG | TA_ASM）の指定ができる。TA_HLNG (=0x00) が指定された場合には高級言語用のインタフェースで、TA_ASM (=0x01) が指定された場合にはアセンブリ言語用のインタフェースでタスク例外処理ルーチンを起動する。(μITRON4.0 仕様)

### 3.2.2 タスクコントロールブロック（TCB）

```
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;
```

### 3.3 サービスコール一覧（task_except.c）

| ｻｰﾋﾞｽｺｰﾙ | 機能 | 備考 |
|---|---|---|
| ras_tex | タスク例外処理の要求 | タスクコンテキスト用 |
| iras_tex | タスク例外処理の要求 | 非タスクコンテキスト用 |
| dis_tex | タスク例外処理の禁止 | タスクコンテキスト用 |
| ena_tex | タスク例外処理の許可 | タスクコンテキスト用 |
| sns_tex | タスク例外処理禁止状態の参照 | |

# 4 割込み管理

## 4.1 割込みについて

- HW からの割込み → カーネル → 割込みハンドラ → 割込みサービスルーチン
- スタンダードプロファイルでは、割込みハンドラか割込みサービスルーチンのどちらかを登録する機能を実装すればよい。（JSP カーネルでは割込みハンドラのみ）

### 4.2 割込みハンドラ初期化ブロック（interrupt.h, interrupt.c）

```
typedef struct interrupt_handler_initialization_block {
        INHNO inhno;            /* 割込みハンドラ番号 */
        ATR   inhatr;           /* 割込みハンドラ属性 */
        FP    inthdr;           /* 割込みハンドラの起動番地 */
} INHINIB;
```

割込みハンドラの定義を行う静的 API

```
DEF_INH(INHNO inhno, { ATR inhatr, FP inthdr });
```

で宣言し、

```
const INHINIB inhinib_table[TNUM_INHNO];
```

に格納される。

INHNO 型の定義と inhno の意味はターゲット毎に定める。inhatr には、TA_HLNG のみを指定することができる。(user.txt)

## 4.3 割込み処理の流れ

(1) 割込み発生(_interrupt)

(2) 割込み入口処理

- ✧ レジスタ保存
- ✧ 例外/割り込みのネスト回数をインクリメント
- ✧ スタック切替え

(3) 割込みハンドラの呼び出し(int_table[]より取得)

- ✧ タスク例外処理の要求

(4) 割込みハンドラから復帰

(5) 割込み出口処理

- ✧ 例外/割り込みのネスト回数をデクリメント
- ✧ スタック戻す

(6) reqflg＝TRUE のとき、タスク例外処理の実行(ret_int)

- ✧ reqflg クリア
- ✧ ディスパッチ必要な場合(enadsp=TRUE かつ runtsk≠schedtsk)、ディスパッチ処理
- ✧ タスク例外処理の実行(calltex)

(7) レジスタ復帰

### 4.4 サービスコール一覧

| ｻｰﾋﾞｽｺｰﾙ | 機能 | 備考 |
|---|---|---|
| dis_int | 割込みの禁止 | SH3 ではサポートしていない |
| ena_int | 割込みの許可 | SH3 ではサポートしていない |
| chg_ixx | 割込みマスクの変更 | SH3 では chg_ipm |
| get_ixx | 割込みマスクの参照 | SH3 では get_ipm |

これらのサービスコールがサポートされているかどうか、サポートされている場合の仕様（xx の部分の名称，型とパラメータの名称と意味、CPU ロック状態やディスパッチ状態との関連）については、ターゲット依存である。(user.txt)

## 5 CPU 例外管理

### 5.1 CPU 例外処理について

- プロセッサが CPU 例外を検出した場合に、CPU 例外ハンドラを起動。
- CPU 例外ハンドラ内で、タスク例外処理を要求される。
- CPU 例外ハンドラの処理は発生したコンテキストで実行されるが、タスク例外処理ルーチンはそのタスクのコンテキストで実行される。

### 5.2 CPU 例外ハンドラ初期化ブロック（exception.h, exception.c）

typedef struct cpu_exception_handler_initialization_block {

```
    EXCNO excno;        /* CPU 例外ハンドラ番号 */
    ATR   excatr;       /* CPU 例外ハンドラ属性 */
    FP    exchdr;       /* CPU 例外ハンドラの起動番地 */
} EXCINIB;
```

CPU 例外ハンドラの定義を行う静的 API

```
DEF_EXC(EXCNO excno, { ATR excatr, FP exchdr });
```

で宣言し、

```
const EXCINIB   excinib_table[TNUM_EXCNO];
```

に格納される。

EXCNO 型の定義と excno の意味はターゲット毎に定める。excatr には、TA_HLNG のみを指定することができる。(user.txt)

## 5.3 CPU 例外処理の流れ

(1) CPU 例外発生(_general_exception)
(2) 例外入口処理
- ✧ レジスタ保存
- ✧ 例外/割り込みのネスト回数をインクリメント
- ✧ スタック切替え

(3) CPU 例外ハンドラの呼び出し（MS7709ASE01/sample1.c 参照）
- ✧ CPU 例外が発生したコンテキストや状態の参照(vxsns_loc, vxsns_ctx)
- ✧ 実行状態のタスク ID の取得(iget_tid)
- ✧ タスク例外処理の要求(iras_tex)

(4) CPU 例外ハンドラから復帰
(5) 例外出口処理
- ✧ 例外/割り込みのネスト回数をデクリメント
- ✧ スタック戻す

(6) reqflg＝TRUE のとき、タスク例外処理の実行(ret_exc)
- ✧ reqflg クリア
- ✧ ディスパッチ必要な場合(enadsp=TRUE かつ runtsk≠schedtsk)、ディスパッチ処理
- ✧ タスク例外処理の実行(calltex)

(7) レジスタ復帰

## 5.4 サービスコール一覧（exception.c）

| ｻｰﾋﾞｽｺｰﾙ | 機能 | 備考 |
|---|---|---|
| vxsns_ctx | CPU 例外の発生したコンテキストの参照 | JSP カーネル独自のサービスコール |
| vxsns_loc | CPU 例外の発生した時の CPU ロック状態の参照 | JSP カーネル独自のサービスコール |
| vxsns_dsp | CPU 例外の発生した時のディスパッチ禁止状態の参照 | JSP カーネル独自のサービスコール |
| vxsns_dpn | CPU 例外の発生した時のディスパッチ保留状態の参照 | JSP カーネル独自のサービスコール |
| vxsns_tex | CPU 例外の発生した時のタスク例外処理禁止状態の参照 | JSP カーネル独自のサービスコール |

これらのサービスコールを用いて、CPU 例外が発生する前の状態を取り出すことができる。