```
/*
 *  TOPPERS/JSP Kernel
 *      Toyohashi Open Platform for Embedded Real-Time Systems/
 *      Just Standard Profile Kernel
 *
 *  Copyright (C) 2000 by Embedded and Real-Time Systems Laboratory
 *                              Toyohashi Univ. of Technology, JAPAN
 *
 *  上記著作権者は，以下の条件を満たす場合に限り，本ソフトウェア（本ソ
 *  フトウェアを改変したものを含む．以下同じ）を使用・複製・改変・再配
 *  布（以下，利用と呼ぶ）することを無償で許諾する．
 *  (1) 本ソフトウェアをソースコードの形で利用する場合には，上記の著作
 *      権表示，この利用条件および下記の無保証規定が，そのままの形でソー
 *      スコード中に含まれていること．
 *  (2) 本ソフトウェアをバイナリコードの形または機器に組み込んだ形で利
 *      用する場合には，次のいずれかの条件を満たすこと．
 *    (a) 利用に伴うドキュメント（利用者マニュアルなど）に，上記の著作
 *        権表示，この利用条件および下記の無保証規定を掲載すること．
 *    (b) 利用の形態を，別に定める方法によって，上記著作権者に報告する
 *        こと．
 *  (3) 本ソフトウェアの利用により直接的または間接的に生じるいかなる損
 *      害からも，上記著作権者を免責すること．
 *
 *  本ソフトウェアは，無保証で提供されているものである．上記著作権者は，
 *  本ソフトウェアに関して，その適用可能性も含めて，いかなる保証も行わ
 *  ない．また，本ソフトウェアの利用により直接的または間接的に生じたい
 *  かなる損害に関しても，その責任を負わない．
 *
 *  @(#) $Id: task_sync.c,v 1.1 2000/11/14 14:44:21 hiro Exp $
 *
 *
 *  リアルタイムカーネル勉強会用にコメントを追加した
 *  Sun Sep 02 01:03:21 2001 modified by Keizyu Anada YAMAHA CORPORATION
 *
 *
 */

/*
 * タスク付属同期機能
 */
#include "jsp_kernel.h"
#include "check.h"
#include "task.h"
#include "wait.h"
```



```
/* ============ */
/*  起床待ちする */
/* ============ */
SYSCALL ER slp_tsk(void)
{
  WINFO winfo;
  ER ercd;

  /*
   *  非タスクコンテキストから呼ばれた時、CPUロック状態の時、
   *  ディスパッチ禁止状態の時は、コンテキストエラーE_CTX を返す
   */
  CHECK_DISPATCH();

  /* CPUロック状態にする */
  t_lock_cpu();

  /* 起床要求キューイング数がTRUE の時は、正常終了E_OK を返す */
  if (runtsk->wupcnt) {

    /* 起床要求キューイング数を FALSE に（クリア）する */
    runtsk->wupcnt = FALSE;

    ercd = E_OK;
  }
  else {

    /* タスクを「待ち状態」に、その要因を「起床待ち状態」に設定する */
    runtsk->tstat = (TS_WAITING | TS_WAIT_SLEEP);

    /* 待ち状態へ移行する */
    make_wait(&winfo); /* => make_non_runnable, winfo->tmevtb = NULL */

    /* ディスパッチする */
    dispatch();

    /* 再び実行状態になるとここに戻ってくる */

    /*
     *  エラーコードを設定する。この変数は
     *  wup_tsk/iwup_tsk 呼び出しにより wait_complete が正常終了E_OK
     *  rel_wai/irel_wai 呼び出しにより wait_release  が待ち状態の強制解除E_RLWAI
     *  に設定する可能性がある
     */
    ercd = winfo.wercd;
  }

  /* CPUロック解除状態にする */
  t_unlock_cpu();

  return(ercd);
}
```

```
103: /* ============================ */
104: /*  タイムアウト付で起床待ちする  */
105: /* ============================ */
106: SYSCALL ER tslp_tsk(TMO tmout)
107: {
108:   WINFO winfo;
109:   TMEVTB tmevtb;
110:   ER ercd;
111:
112:   /*
113:    *  非タスクコンテキストから呼ばれた時、CPUロック状態の時、
114:    *  ディスパッチ禁止状態の時は、コンテキストエラーE_CTX を返す
115:    */
116:   CHECK_DISPATCH();
117:
118:   /*
119:    *  タイムアウト指定値が「TMO_FEVR = -1」以外の負数の時は、
120:    *  パラメータエラーE_PAR を返す。
121:    */
122:   CHECK_TMOUT(tmout);
123:
124:   /* CPUロック状態にする */
125:   t_lock_cpu();
126:
127:   /* 起床要求キューイング数が TRUE の時は、正常終了E_OK を返す */
128:   if (runtsk->wupcnt) {
129:
130:     /* 起床要求キューイング数をFALSEに（０クリア）する */
131:     runtsk->wupcnt = FALSE;
132:
133:     ercd = E_OK;
134:   }
135:   /* タイムアウト指定値が「TMO_POL = 0」の時は、ポーリング失敗エラー E_TMOUT を返す */
136:   else if (tmout == TMO_POL) {
137:
138:     ercd = E_TMOUT;
139:   }
140:   /* タイムアウト指定値が正常な値の時 */
141:   else {
142:
143:     /* タスク状態を「待ち状態」に、その要因を「起床待ち状態」に設定する */
144:     runtsk->tstat = (TS_WAITING | TS_WAIT_SLEEP);
145:
146:     /* タイムアウト指定付きで待ち状態に移行する */
147:     make_wait_tmout(&winfo, &tmevtb, tmout);
148:
149:     /*
150:      *  => make_non_runnable
151:      *  => tmevtb_enqueue
152:      *  現在時刻から tmout 後に、このタスクのTCBを引数として
153:      *  wait_tmout がコールバックされるように、
154:      *  タイムイベントブロックのキューに登録する。
155:      *  指定時間後に wait_tmout がコールバックされた場合は
156:      *  winfo.wercd にタイムアウトエラーE_TMOUT が設定される
157:      */
158:
159:     /* ディスパッチする */
160:     dispatch();
161:
162:     /* 再び実行状態になるとここに戻ってくる */
163:
164:     /*
165:      *  エラーコードを設定する。この変数は
166:      *  wup_tsk/iwup_tsk 呼び出しにより wait_complete が正常終了E_OK
167:      *  rel_wai/irel_wai 呼び出しにより wait_release  が待ち状態の強制解除E_RLWAI
168:      *  に設定する可能性がある
```



```
169:      */
170:     ercd = winfo.wercd;
171:   }
172:
173:   /* CPUロック解除状態にする */
174:   t_unlock_cpu();
175:
176:   return(ercd);
177: }
178:
179:
```

```
/* =============== */
/*  タスクを起床する */
/* =============== */
SYSCALL ER wup_tsk(ID tskid)
{
  TCB *tcb;
  UINT tstat;
  ER ercd;

  /*
   * 非タスクコンテキストから呼ばれた時、CPUロック状態の時は、
   * コンテキストエラーE_CTX を返す
   */
  CHECK_TSKCTX_UNL();

  /*
   * 存在しないタスクID の時、指定tskid が自分自身の時は、
   * 不正ID番号エラーE_ID を返す
   */
  CHECK_TSKID_SELF(tskid);

  /* 指定tskid のTCBへのポインタを得る */
  tcb = get_tcb_self(tskid);

  /* CPUロック状態にする */
  t_lock_cpu();

  /* タスクが「休止状態」の時は、オブジェクト状態エラーE_OBJ を返す */
  if (TSTAT_DORMANT(tstat = tcb->tstat)) {
    ercd = E_OBJ;
  }
  /* タスクが（「待ち状態」でその要因が）slp_tsk/tslp_tsk による「起床待ち状態」の時 */
  else if ((tstat & TS_WAIT_SLEEP) != 0) {

    /* タスクの待ち状態を解除する */
    if (wait_complete(tcb)) { /* => make_non_wait => make_runnable */

      /* tskidで指定したタスクが最高優先順位になる場合はディスパッチする */
      dispatch();
    }

    /* 正常終了を返す */
    ercd = E_OK;
  }
  /* 起床要求キューイング数が FALSE の場合は TRUE を設定して、正常終了E_OK を返す */
  else if (!(tcb->wupcnt)) {
    tcb->wupcnt = TRUE;
    ercd = E_OK;
  }
  /* 起床要求キューイング数が TRUE の場合は、キューイングオーバーフローE_QOVR を返す */
  else {
    ercd = E_QOVR;
  }

  /* CPUロック解除状態にする */
  t_unlock_cpu();

  return(ercd);
}
```



```
/* ======================================== */
/*  タスクを起床する（非タスクコンテキスト用） */
/* ======================================== */
SYSCALL ER iwup_tsk(ID tskid)
{
  TCB *tcb;
  UINT tstat;
  ER ercd;

  /*
   * タスクコンテキストから呼ばれた時、CPUロック状態の時は
   * コンテキストエラーE_CTX を返す
   */
  CHECK_INTCTX_UNL();

  /* 存在しないタスクID の時は、不正ID番号エラー E_ID を返す */
  CHECK_TSKID(tskid);

  /* 指定tskid のTCBへのポインタを得る */
  tcb = get_tcb(tskid);

  /* CPUロック状態にする */
  i_lock_cpu();

  /* タスクが「休止状態」の時は、オブジェクト状態エラーE_OBJ を返す */
  if (TSTAT_DORMANT(tstat = tcb->tstat)) {
    ercd = E_OBJ;
  }
  /* タスクが（「待ち状態」でその要因が）slp_tsk/tslp_tsk による「起床待ち状態」の時 */
  else if ((tstat & TS_WAIT_SLEEP) != 0) {

    /* タスクの待ち状態を解除する */
    if (wait_complete(tcb)) { /* => make_non_wait => make_runnable */

      /*
       *  tskidで指定したタスクが最高優先順位になる場合は
       *  タスクディスパッチ要求フラグをセットする
       */
      reqflg = TRUE;
    }

    /* 正常終了を返す */
    ercd = E_OK;
  }
  /* 起床要求キューイング数が FALSE の場合は TRUE を設定して、正常終了E_OK を返す */
  else if (!(tcb->wupcnt)) {
    tcb->wupcnt = TRUE;
    ercd = E_OK;
  }
  /* 起床要求キューイング数が TRUE の場合は、キューイングオーバーフローE_QOVR を返す */
  else {
    ercd = E_QOVR;
  }

  /* CPUロック解除状態にする */
  i_unlock_cpu();

  return(ercd);
}
```

```
302: /* ============================ */
303: /*  タスク起床要求をキャンセルする */
304: /* ============================ */
305: SYSCALL ER_UINT can_wup(ID tskid)
306: {
307:   TCB *tcb;
308:   ER_UINT wupcnt;
309:
310:   /*
311:    *  非タスクコンテキストから呼ばれた時、CPUロック状態の時は
312:    *  コンテキストエラーE_CTX を返す
313:    */
314:   CHECK_TSKCTX_UNL();
315:
316:   /*
317:    *  存在しないタスクID の時、指定tskid が自分自身の時は
318:    *  不正ID番号エラーE_ID を返す
319:    */
320:   CHECK_TSKID_SELF(tskid);
321:
322:   /* 指定tskid のTCBへのポインタを得る */
323:   tcb = get_tcb_self(tskid);
324:
325:   /* CPUロック状態にする */
326:   t_lock_cpu();
327:
328:   /* タスクが「休止状態」の時は、オブジェクト状態エラーE_OBJ を返す */
329:   if (TSTAT_DORMANT(tcb->tstat)) {
330:     wupcnt = E_OBJ;
331:   }
332:   else {
333:
334:     /* 起床要求キューイング数を ... する */
335:     wupcnt = tcb->wupcnt ? 1 : 0;
336:
337:     /* 起床要求キューイング数を０クリアする */
338:     tcb->wupcnt = FALSE;
339:   }
340:
341:   /* CPUロック解除状態にする */
342:   t_unlock_cpu();
343:
344:   /* 起床要求キューイング数を０クリアする前の値を返す */
345:   return(wupcnt);
346: }
347:
348:
```



```
349: /* ====================== */
350: /*  待ち状態を強制解除する  */
351: /* ====================== */
352: SYSCALL ER rel_wai(ID tskid)
353: {
354:   TCB *tcb;
355:   ER ercd;
356:
357:   /*
358:    *  非タスクコンテキストから呼ばれた時、CPUロック状態の時は
359:    *  コンテキストエラーE_CTX を返す
360:    */
361:   CHECK_TSKCTX_UNL();
362:
363:   /* 存在しないタスクID の時は、不正ID番号エラーE_ID を返す  */
364:   CHECK_TSKID(tskid);
365:
366:   /* 指定tskid のTCBへのポインタを得る */
367:   tcb = get_tcb(tskid);
368:
369:   /* CPUロック状態にする */
370:   t_lock_cpu();
371:
372:   /* タスクが「待ち状態」でない時は、オブジェクト状態エラーE_OBJ を返す */
373:   if (!(TSTAT_WAITING(tcb->tstat))) {
374:     ercd = E_OBJ;
375:   }
376:   /* タスクが「待ち状態」、もしくは「二重待ち状態」の時 */
377:   else {
378:     /*
379:      *  タスクの状態が、「待ち状態」だったならば「実行できる状態」に
380:      *  「二重待ち状態」だったならば「強制待ち状態」に設定する。
381:      *  待ち解除されたタスクのサービスコールの返値を
382:      *  待ち状態の強制解除E_RLWAI に設定する。
383:      */
384:     if (wait_release(tcb)) { /*
385:                  *  (待ちの原因がdly_tsk ならば => tmevtb_dequeue)
386:                  *  => make_non_wait
387:                  *  (「待ち状態」らなば => make_runnable)
388:                  */
389:
390:       /* tskidで指定したタスクが最高優先順位になる場合はディスパッチする */
391:       dispatch();
392:     }
393:
394:     ercd = E_OK;
395:   }
396:
397:   /* CPUロック解除状態にする */
398:   t_unlock_cpu();
399:
400:   return(ercd);
401: }
402:
403:
```

```
/* ========================================== */
/*  待ち状態を強制解除する（非タスクコンテキスト用）  */
/* ========================================== */
SYSCALL ER irel_wai(ID tskid)
{
  TCB *tcb;
  ER ercd;

  /*
   *  タスクコンテキストから呼ばれた時、CPUロック状態の時は、
   *  コンテキストエラーE_CTX を返す
   */
  CHECK_INTCTX_UNL();

  /* 存在しないタスクID の時は、不正ID番号エラーE_ID を返す */
  CHECK_TSKID(tskid);

  /* 指定tskid のTCBへのポインタを得る */
  tcb = get_tcb(tskid);

  /* CPUロック状態にする */
  i_lock_cpu();

  /* タスクが「待ち状態」でない時は、オブジェクト状態エラーE_OBJ を返す */
  if (!(TSTAT_WAITING(tcb->tstat))) {
    ercd = E_OBJ;
  }
  /* タスクが「待ち状態」、もしくは「二重待ち状態」の時 */
  else {

    /*
     *  タスクの状態が、「待ち状態」だったならば「実行できる状態」に
     *  「二重待ち状態」だったならば「強制待ち状態」に設定する。
     *  待ち解除されたタスクのサービスコールの返値を
     *  待ち状態の強制解除E_RLWAI に設定する。
     */
    if (wait_release(tcb)) { /*
                  * (待ちの原因が dly_tsk ならば => tmevtb_dequeue)
                  *  => make_non_wait
                  * (「待ち状態」らなば => make_runnable)
                  */
      /*
       *  tskidで指定したタスクが最高優先順位になる場合は
       *  タスクディスパッチ要求フラグをセットする
       */
      reqflg = TRUE;
    }

    ercd = E_OK;
  }

  /* CPUロック解除状態にする */
  i_unlock_cpu();

  return(ercd);
}
```



```
/* ==================== */
/*  強制待ち状態への移行  */
/* ==================== */
SYSCALL ER sus_tsk(ID tskid)
{
  TCB *tcb;
  UINT tstat;
  ER ercd;

  /*
   * 非タスクコンテキストから呼ばれた時、CPUロック状態の時は、
   * コンテキストエラーE_CTX を返す
   */
  CHECK_TSKCTX_UNL();

  /*
   *  存在しないタスクID の時、指定tskid が自分自身の時は、
   *  不正ID番号エラーE_ID を返す
   */
  CHECK_TSKID_SELF(tskid);

  /* 指定tskid のTCBへのポインタを得る */
  tcb = get_tcb_self(tskid);

  /* CPUロック状態にする */
  t_lock_cpu();

  /* タスクが自分自身かディスパッチ禁止状態の時は、コンテキストエラーを返す */
  if (tcb == runtsk && !(enadsp)) {
    ercd = E_CTX;
  }
  /* タスクが「休止状態」の時は、オブジェクト状態エラーを返す */
  else if (TSTAT_DORMANT(tstat = tcb->tstat)) {
    ercd = E_OBJ;
  }
  /* タスクが「実行できる状態」の時 */
  else if (TSTAT_RUNNABLE(tstat)) {

    /* タスク状態を「強制待ち状態」に設定する */
    tcb->tstat = TS_SUSPENDED;

    /*
     *  このタスクをレディキューから削除する。
     *  このタスクと同じ優先度のタスクがなければ、
     *  レディキューサーチマップからこの優先度ビットをクリアし
     *  このタスクが最高優先順位タスクならば最高優先順位のタスクを更新する。
     *  このタスクと同じ優先度のタスクがあれば、
     *  次の優先順位のタスクを最高優先順位のタスクに設定する。
     */
    if (make_non_runnable(tcb)) {

      /* ディスパッチする */
      dispatch();
    }

    ercd = E_OK;
  }
  /* タスクが既に「強制待ち状態」の時は、キューイングオーバーフローE_QOVR を返す */
  else if (TSTAT_SUSPENDED(tstat)) {
    ercd = E_QOVR;
  }
  /* タスクが「待ち状態」の時は、「二重待ち状態」に設定して、正常終了E_OK を返す */
  else {
    tcb->tstat |= TS_SUSPENDED;
    ercd = E_OK;
  }
```

```
528:
529:    /* CPUロック解除状態にする */
530:    t_unlock_cpu();
531:
532:    return(ercd);
533: }
534:
535:
```



```
536: /* ====================== */
537: /*  強制待ち状態からの再開  */
538: /* ====================== */
539: SYSCALL ER rsm_tsk(ID tskid)
540: {
541:    TCB *tcb;
542:    UINT tstat;
543:    ER ercd;
544:
545:    /*
546:     *  非タスクコンテキストから呼ばれた時、CPUロック状態の時は、
547:     *  コンテキストエラーE_CTXを返す
548:     */
549:    CHECK_TSKCTX_UNL();
550:
551:    /* 存在しないタスクID の時は、不正ID番号エラーE_ID を返す */
552:    CHECK_TSKID(tskid);
553:
554:    /* 指定tskid のTCBへのポインタを得る */
555:    tcb = get_tcb(tskid);
556:
557:    /* CPUロック状態にする */
558:    t_lock_cpu();
559:
560:    /* タスクが「強制待ち状態」でない時は、オブジェクト状態エラーを返す */
561:    if (!(TSTAT_SUSPENDED(tstat = tcb->tstat))) {
562:      ercd = E_OBJ;
563:    }
564:    /* タスクが「二重待ち状態」でない時 */
565:    else if (!(TSTAT_WAITING(tstat))) {
566:
567:      /*
568:       *  タスクを「実行できる状態」にして、
569:       *  タスクを優先度毎のレディキューの最後尾につなぎ、
570:       *  レディキューサーチマップにこの優先度ビットをセットする。
571:       *  最高優先順位のタスクが設定されていなかった時、もしくは、
572:       *  このタスクの優先度が最高優先順位タスクの優先度より高い場合は、
573:       *  最高優先順位タスクをこのタスクに設定する。
574:       */
575:      if (make_runnable(tcb)) {
576:
577:        /*
578:         *  最高優先順位タスクが変更になった時、かつ、
579:         *  ディスパッチ許可状態ならばディスパッチする
580:         */
581:        dispatch();
582:      }
583:
584:      ercd = E_OK;
585:    }
586:    /* タスクが「二重待ち状態」の時は「待ち状態」にして正常終了E_OK を返す */
587:    else {
588:
589:      tcb->tstat &= ~TS_SUSPENDED;
590:
591:      ercd = E_OK;
592:    }
593:
594:    /* CPUロック解除状態にする */
595:    t_unlock_cpu();
596:
597:    return(ercd);
598: }
599:
600:
```

```
601: /* ================================================== */
602: /*  強制待ち状態から強制再開させる                    */
603: /*  JSPカーネルでは，frsm_tsk と rsm_tsk は同一の処理となる */
604: /* ================================================== */
605: #ifdef LABEL_ALIAS
606: LABEL_ALIAS(frsm_tsk, rsm_tsk)
607: #else /* LABEL_ALIAS */
608: SYSCALL ER frsm_tsk(ID tskid)
609: {
610:   return(rsm_tsk(tskid));
611: }
612: #endif /* LABEL_ALIAS */
613:
614:
```



```
615: /* ================== */
616: /*  自タスクを遅延する  */
617: /* ================== */
618: SYSCALL ER dly_tsk(RELTIM dlytim)
619: {
620:   WINFO winfo;
621:   TMEVTB tmevtb;
622:
623:   /*
624:    *  非タスクコンテキストから呼ばれた時、CPUロック状態の時、
625:    *  ディスパッチ禁止状態の時は、コンテキストエラーE_CTX を返す
626:    */
627:   CHECK_DISPATCH();
628:
629:   /* 負数ならば、パラメータエラーE_PAR を返す */
630:   CHECK_PAR(dlytim <= TMAX_RELTIM);
631:
632:   /* CPUロック状態にする */
633:   t_lock_cpu();
634:
635:   /* タスクを「待ち状態」にする */
636:   runtsk->tstat = TS_WAITING;
637:
638:   /*
639:    *  このタスクをレディキューから削除する。
640:    *  このタスクと同じ優先度のタスクがなければ、
641:    *  レディキューサーチマップからこの優先度ビットをクリアし
642:    *  このタスクが最高優先順位タスクならば最高優先順位のタスクを更新する。
643:    *  このタスクと同じ優先度のタスクがあれば、
644:    *  次の優先順位のタスクを最高優先順位のタスクに設定する。
645:    */
646:   make_non_runnable(runtsk);
647:
648:   runtsk->winfo = &winfo;
649:
650:   winfo.tmevtb = &tmevtb;
651:
652:   /*
653:    *  現在時刻から dlytim 後に、このタスクのTCBを引数として
654:    *  wait_tmout_ok がコールバックされるように、
655:    *  タイムイベントブロックのキューに登録する。
656:    *  指定時間後に wait_tmout がコールバックされた場合は
657:    *  winfo.wercd に正常終了E_OK が設定される。
658:    *  rel_wai/irel_wai 呼び出しにより「待ち状態」を強制解除された時は
659:    *  => wait_release が待ち状態の強制解除E_RLWAI を設定する。
660:    */
661:   tmevtb_enqueue(&tmevtb, dlytim, (CBACK)wait_tmout_ok, (VP) runtsk);
662:
663:   /* ディスパッチする */
664:   dispatch();
665:
666:   /* CPUロック解除状態にする */
667:   t_unlock_cpu();
668:
669:   return(winfo.wercd);
670: }
671:
```