```
/*
 *  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: wait.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
 *  Mon Sep 03 00:14:30 2001 modified by Keizyu Anada YAMAHA CORPORATION
 *
 *
 */

/*
 *  待ち状態管理モジュール
 */

#include "jsp_kernel.h"
#include "wait.h"


/*
 *  待ち状態への移行（タイムアウト指定）
 *
 *  実行中のタスクを，タイムアウト指定付きで待ち状態に移行させる．具体
 *  的には，実行中のタスクをレディキューから削除し，TCB の winfoフィー
 *  ルド，WINFO の tmevtbフィールドを設定する．また，タイムイベントブ
 *  ロックを登録する．
 *
 *  この関数は tslp_tsk から呼ばれる可能性がある。
 *
 */
inline void make_wait_tmout(WINFO *winfo,   /* 待ち情報ブロックへのポインタ       */
                TMEVTB *tmevtb, /* タイムイベントブロックへのポインタ */
                TMO tmout)      /* タイムアウト時間                   */
{
  /*
    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。
  */
  make_non_runnable(runtsk);

  runtsk->winfo = winfo;

  if (tmout > 0) {

    winfo->tmevtb = tmevtb;

    /* 現在時刻から tmout 後に、実行状態のタスクのTCBを引数として */
    /* wait_tmout がコールバックされるように、                     */
    /* タイムイベントブロックのキューに登録する                    */
    tmevtb_enqueue(tmevtb,
          (RELTIM)tmout,
          (CBACK)wait_tmout,
          (VP) runtsk);
  }
  else {

    /* 永久待ちでない、かつ、DEBUG ならばシステムログを残してカーネルを終了する */
    assert(tmout == TMO_FEVR);

    winfo->tmevtb = NULL;
  }
}

/*
 *  待ち解除のためのタスク状態の更新
 *
 *  tcb で指定されるタスクを，待ち解除するようタスク状態を更新する．待
 *  ち解除するタスクが実行できる状態になる場合は，レディキューにつなぐ．
 *  また，ディスパッチが必要な場合には TRUE を返す．
 *
 *  この関数は wup_tsk/rel_wai から呼ばれる。
 *
 */
Inline BOOL make_non_wait(TCB *tcb)
{
  /* 「待ち状態」でない、かつ、DEBUG ならばシステムログを残してカーネルを終了する */
  assert(TSTAT_WAITING(tcb->tstat));

  if (!(TSTAT_SUSPENDED(tcb->tstat))) {

    /*
      タスクを「実行できる状態」にして、
      タスクを優先度毎のレディキューの最後尾につなぎ、
      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。
      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、
      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、
      最高優先順位タスクをこのタスクに設定する。
    */
    return(make_runnable(tcb));
  }
  else {

    /* 「二重待ち」の時は「強制待ち状態」に設定する */
    tcb->tstat = TS_SUSPENDED;

    return(FALSE);
  }
}

/*
 *  待ち解除
 *
```

```
 133:  *  tcb で指定されるタスクの待ち状態を解除する．具体的には，タイムイベ
 134:  *  ントブロックが登録されていれば，それを登録解除する．また，タスク状
 135:  *  態を更新し，待ち解除したタスクからの返値を E_OK とする．待ちキュー
 136:  *  からの削除は行わない．待ち解除したタスクへのディスパッチが必要な場
 137:  *  合には TRUE を返す．
 138:  *
 139:  *  この関数は slp_tsk/tslp_tsk による「起床待ち状態」のタスクを
 140:  *  wup_tsk/iwup_tsk で起床させる時に呼ばれる。
 141:  *
 142:  */
 143: BOOL wait_complete(TCB *tcb)
 144: {
 145:   if (tcb->winfo->tmevtb != NULL) {
 146:     /* タイムイベントブロックのキューから削除する */
 147:     tmevtb_dequeue(tcb->winfo->tmevtb);
 148:   }
 149:
 150:   /* slp_tsk/tslp_tsk を発行したタスクに正常終了E_OK が返るように設定する */
 151:   tcb->winfo->wercd = E_OK;
 152:
 153:   /* 待ち解除のためにタスクの状態を更新する */
 154:   return(make_non_wait(tcb));
 155: }
 156:
 157: /*
 158:  *  タイムアウトに伴う待ち解除
 159:  *
 160:  *  tcb で指定されるタスクが，待ちキューにつながれていれば待ちキューか
 161:  *  ら削除し，タスク状態を更新する．また，待ち解除したタスクからの返値
 162:  *  を，wait_tmoutでは E_TMOUT，wait_tmout_ok では E_OK とする．待ち解
 163:  *  除したタスクへのディスパッチが必要な時は，reqflg を TRUE にする．
 164:  *  wait_tmout_ok は，dly_tsk で使うためのもので，待ちキューから削除す
 165:  *  る処理を行わない．
 166:  *  いずれの関数も，タイムイベントのコールバック関数として用いるための
 167:  *  もので，割込みハンドラから呼び出されることを想定している．
 168:  *
 169:  *  この関数は tslp_tsk によりタイムイベントブロックのキューに積まれた時、
 170:  *  指定時間後に割り込みハンドラから呼ばれる。
 171:  *
 172:  */
 173: void wait_tmout(TCB *tcb)
 174: {
 175:   if ((tcb->tstat & TS_WAIT_WOBJ) != 0) {
 176:     queue_delete(&(tcb->task_queue));
 177:   }
 178:
 179:   /* tslp_tsk を発行したタスクにタイムアウトエラーE_TMOUT が返るように設定する */
 180:   tcb->winfo->wercd = E_TMOUT;
 181:
 182:   /* 待ち解除のためにタスクの状態を更新する */
 183:   if (make_non_wait(tcb)) {
 184:     reqflg = TRUE;
 185:   }
 186: }
 187:
 188: /*
 189:  * この関数は dly_tsk によりタイムイベントブロックのキューに積まれた時、
 190:  * 指定時間後に割り込みハンドラから呼ばれる。
 191:  */
 192: void wait_tmout_ok(TCB *tcb)
 193: {
 194:   /* dly_tsk を発行したタスクに正常終了E_OK が返るように設定する */
 195:   tcb->winfo->wercd = E_OK;
 196:
 197:   /* 待ち解除のためにタスクの状態を更新する */
 198:   if (make_non_wait(tcb)) {
```



```
 199:     reqflg = TRUE;
 200:   }
 201: }
 202:
 203: /*
 204:  *  待ち状態の強制解除
 205:  *
 206:  *  tcb で指定されるタスクの待ち状態を強制的に解除する．具体的には，タ
 207:  *  スクが待ちキューにつながれていれば待ちキューから削除し，タイムイベ
 208:  *  ントブロックが登録されていればそれを登録解除する．
 209:  *  wait_cancel は，タスクの状態は更新しない．
 210:  *  wait_release は，タスクの状態を更新し，待ち解除したタスクからの返
 211:  *  値を E_RLWAI とする．また，待ち解除したタスクへのディスパッチが必
 212:  *  要な場合には TRUE を返す．
 213:  *
 214:  *  この関数は rel_wai/irel_wai の対象タスクが「待ち状態」か
 215:  *  「二重待ち状態」の時に呼ばれる可能性がある。
 216:  *
 217:  */
 218: inline void wait_cancel(TCB *tcb)
 219: {
 220:   if (tcb->winfo->tmevtb != NULL) {
 221:     tmevtb_dequeue(tcb->winfo->tmevtb);
 222:   }
 223:
 224:   if ((tcb->tstat & TS_WAIT_WOBJ) != 0) {
 225:     queue_delete(&(tcb->task_queue));
 226:   }
 227: }
 228:
 229: /*
 230:  * この関数は rel_wai/irel_wai の対象タスクが「待ち状態」か
 231:  *  「二重待ち状態」の時に呼ばれる可能性がある。
 232:  */
 233: BOOL wait_release(TCB *tcb)
 234: {
 235:   wait_cancel(tcb);
 236:
 237:   /* 待ち状態にあるタスクに待ち状態の強制解除E_RLWAIが返るように設定する */
 238:   tcb->winfo->wercd = E_RLWAI;
 239:
 240:   /* 待ち解除のためにタスクの状態を更新する */
 241:   return(make_non_wait(tcb));
 242: }
 243:
 244:
```

```
245: /*
246:  *  タスクの優先度順の待ちキューへの挿入
247:  */
248: Inline void queue_insert_tpri(TCB *tcb, QUEUE *queue)
249: {
250:   QUEUE *entry;
251:   UINT  priority = tcb->priority;
252:
253:   for (entry = queue->next; entry != queue; entry = entry->next) {
254:
255:     if (priority < ((TCB *) entry)->priority) {
256:       break;
257:     }
258:   }
259:
260:   queue_insert_prev(entry, &(tcb->task_queue));
261: }
262:
263: /*
264:  *  実行中のタスクの同期・通信オブジェクトの待ちキューへの挿入
265:  */
266: Inline void wobj_queue_insert(WOBJCB *wobjcb)
267: {
268:   if ((wobjcb->wobjinib->wobjatr & TA_TPRI) != 0) {
269:     queue_insert_tpri(runtsk, &(wobjcb->wait_queue));
270:   }
271:   else {
272:     queue_insert_prev(&(wobjcb->wait_queue),
273:               &(runtsk->task_queue));
274:   }
275: }
276:
277: /*
278:  *  同期・通信オブジェクトに対する待ち状態への移行
279:  */
280: void wobj_make_wait(WOBJCB *wobjcb, WINFO_WOBJ *winfo)
281: {
282:   runtsk->tstat = (TS_WAITING | TS_WAIT_WOBJ | TS_WAIT_WOBJCB);
283:
284:   make_wait(&(winfo->winfo));
285:
286:   wobj_queue_insert(wobjcb);
287:
288:   winfo->wobjcb = wobjcb;
289: }
290:
291: void wobj_make_wait_tmout(WOBJCB *wobjcb, WINFO_WOBJ *winfo,
292:             TMEVTB *tmevtb, TMO tmout)
293: {
294:   runtsk->tstat = (TS_WAITING | TS_WAIT_WOBJ | TS_WAIT_WOBJCB);
295:
296:   make_wait_tmout(&(winfo->winfo), tmevtb, tmout);
297:
298:   wobj_queue_insert(wobjcb);
299:
300:   winfo->wobjcb = wobjcb;
301: }
302:
303: /*
304:  *  タスクの優先度変更時の処理
305:  */
306: void wobj_change_priority(WOBJCB *wobjcb, TCB *tcb)
307: {
308:   if ((wobjcb->wobjinib->wobjatr & TA_TPRI) != 0) {
309:
310:     queue_delete(&(tcb->task_queue));
```



```
311:
312:     queue_insert_tpri(tcb, &(wobjcb->wait_queue));
313:   }
314: }
315:
```