YAMAHA

リアルタイムカーネル勉強会
非プロセッサ依存部（2）

ヤマハ株式会社
穴田 啓樹
keizyu@beat.yamaha.co.jp

YAMAHA

目次

YAMAHA

1. はじめに

- 本勉強会の目的は、タスクの状態管理、タスク付属同期機能を理解することです。

YAMAHA

1.1 対象ファイル・サービスコール

- ファイル
  ./kernel/wait.h
  ./kernel/wait.c
  ./task_sync.c
- サービスコール
  slp_tsk, tslp_tsk, wup_tsk, iwup_tsk, can_wup, rel_wai, irel_wai, sus_tsk, rem_tsk, frsm_tsk, dly_tsk

YAMAHA

1.2 本勉強会の進め方

本勉強会では下記の手順で進めます。

- 概要編
  - サービスコールの機能を確認する
  - サービスコールによるタスクの状態変化を確認する
- 詳細編
  - 各サービスコールのソースを読む

YAMAHA

2. サービスコールの機能確認

- **slp_tsk タスクを起床待ちにする**
- **wup_tsk タスクを起床する**
- **can_wup タスクの起床要求をキャンセルする**
- **rel_wai タスクの待ち状態を強制解除する**
- **sus_ysk タスクを強制待ち状態へ移行する**
- **rsm_tsk タスクを強制待ち状態から再開する**
- **dly_tsk 自タスクの実行を遅延する機能**

YAMAHA

2.1 要求のキューイングについて

- 起床要求のキューイング
  起床待ち状態でないタスクを起床しようとすると、そのタスクを起床しようとしたという記録が残り、後でそのタスクが起床待ちに移行しようとした時に、タスクを起床待ち状態にしない。
- 強制待ち要求のネスト
  強制待ち状態（二重待ち状態を含む）になっているタスクを再度強制待ち状態に移行させようとすると、そのタスクを強制待ち状態に移行させようとしたという記録が残り、後でそのタスクを強制待ち状態（二重待ち状態を含む）から再開させようとしたときに、強制待ちからの再開を行わない。

YAMAHA

3. タスク状態について

タスク状態の内部表現はビットマップになっています。

- TS_DORMANT 休止状態
- TS_RUNNABLE 実行できる状態
- TS_WAITING 待ち状態
- TS_SUSPENDING 強制待ち状態

更に待ち状態の付属状態として下記があります。

- TS_WAIT_SLEEP 起床待ち状態
- TS_WAIT_WOBJ 同期/通信オブジェクトに対する待ち状態
- TS_WAIT_WOBJCB 共通部分の待ちキューにつながっている

YAMAHA

3.1 タスク状態の変化

- タスク状態の遷移表
  サービスコールよ呼び出しに対する、タスク状態の変化を状態遷移表にまとめましたので、CASE tool を使って動作確認してみます。
- この表の読み方
  あるタスク状態（列）の時にサービスコール（行）が発行されると、交わるセルの処理欄が実行され、遷移欄に書かれた状態に遷移します。セルが十字で仕切られているものは「if」文を表現しています。

YAMAHA

4. カーネル読書会

- サービスコールは割込み禁止で実行します
- タスクコンテキストのサービスコール基本形

```
xxx_xxx()
{
    エラーチェック
    t_lock_cpu();
    クリティカルセクション実行
    t_unlock_cpu();
}
```

- 非タスクコンテキストのサービスコール基本形

```
ixxx_xxx()
{
    i_lock_cpu();
    クリティカルセクション実行
    i_unlock_cpu();
}
```


YAMAHA

4.1 前提知識1 ～構造体1～

```
typedef struct time_event_block {
    UINT  index;    /* タイムイベントヒープ中での位置 */
    CBACK callback;          /* コールバック関数 */
    VP    arg;      /* コールバック関数へ渡す引数 */
} TMEVTB; /* time_event.h */

typedef union waiting_information {
    ER    wercd;    /* 待ち解除時のエラーコード */
    TMEVTB        *tmevtb; /* 待ち状態用のタイムイベントブロック */
} WINFO; /* task.h */
```


YAMAHA

4.1 前提知識2 ～構造体2～

```
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;
```

4.2 前提知識3 ～変数～

```
/* 実行状態のタスク */
TCB    * runtsk;
/* 最高優先順位のタスク */
TCB    * schedtsk;
/* 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 reqflg;
/* タスクディスパッチ許可状態 */
BOOL enadsp;
```


4.3 読書手順

ソースをトップダウンで読んでいきます。

- task_sync.c
  サービスコールが定義してあります
- wait.h
  プロトタイプ宣言、同期/通信オブジェクト用の構造体が定義してあります
- wait.c
  サービスコール用の内部関数が定義してあります

5. 参考文献

- μITRON4.0仕様書　社団法人トロン協会
- TOPPERS/JSPのドキュメント　ERTL 豊橋技術科学大学
  - TOPPERS/JSPカーネル ユーザズマニュアル
  - JSPカーネル 設計メモ
  - JSPカーネル ターゲット依存部インタフェース仕様