トロンフォーラム

実習：組込みリアルタイム・プログラミング

（ITRON 初級編）

# プログラミング演習テキスト

トロンフォーラム 学術・教育 WG

株式会社グレープシステム　宮下 光明

μITRON4.0 仕様は、トロンフォーラムが定めたオープンなリアルタイムカーネル仕様です。μITRON4.0 仕様の仕様書は、トロンフォーラム Web サイトから入手することができます。

- TRON は”The Real-time Operating system Nucleus”の略称です。
- ITRON は”Industrial TRON”の略称です。
- μITRON は”Micro Industrial TRON”の略称です。
- TRON,ITRON,およびμITRON は、特定の商品ないしは商品群を指す名称ではありません。

組込み型リアルタイム OS 実技コース:プログラミング演習テキスト

目次

# 1 課題：タスクの起動と優先度

## 1.1 概要・課題

### 1.1.1 扱うプログラム(ex_pri.c)の概要

- task1, task2, task3 の 3 つのタスクから構成。
- プログラム内で chg_pri()を発行することで優先度を変更し、優先度の高いタスクへの切り替えが起こる。
- exd_tsk()により、タスクの終了を行う(task2)。

### 1.1.2 課題

1) サンプルプログラム ex_pri.c を動作させて、ITRON/T-Kernel のスケジューリング規則に従ってタスクが切替る様子を確認せよ。

## 1.2 実習環境の前提

本課題では、コンソールと ITRON/T-Kernel のサービスコールだけで行う実習である。

### 1.2.1 使用する外部ライブラリ

- printf():コンソールへの文字列出力/標準 C ライブラリの仕様と同様

## 1.3 サンプルプログラム

### 1.3.1 system.cfg

```
/************************************/
/* Sample configuration file        */
/************************************/

INCLUDE("task_pri.h");

#define STACK_SIZE 2048

CRE_TSK(TASK1, {TA_HLNG|TA_ACT, TASK1, task1, 20, STACK_SIZE, NULL});
CRE_TSK(TASK2, {TA_HLNG|TA_ACT, TASK2, task2, 21, STACK_SIZE, NULL});
CRE_TSK(TASK3, {TA_HLNG|TA_ACT, TASK3, task3, 22, STACK_SIZE, NULL});
```

### 1.3.2 ex_pri.c

```
#include  "kernel.h"

#include  "kernel_id.h"
#include  "ex_pri.h"

/*********************************************/
/* task1                                     */
/*********************************************/
void task1(VP_INT exinf)
{
    ER ercd;
    PRI tskpri;

    printf("Start task1()¥n");

    ercd = get_pri(TSK_SELF, &tskpri);

    printf("task1: priority change = %d¥n", tskpri+1);
    ercd = chg_pri(TSK_SELF, tskpri+1);                 /* priority 20 -> 21 */

    printf("task1: priority change = %d¥n", tskpri+2);
    ercd = chg_pri(TSK_SELF, tskpri+2);                 /* priority 21 -> 22 */

    printf("End task1()¥n");
}

/*********************************************/
/* task2                                     */
/*********************************************/
void task2(VP_INT exinf)
{
```

```
    ER ercd;
    PRI itskpri;
    PRI tskpri;
    int cnt = 0;

    printf("    Start task2()¥n");

    ercd = get_pri(TSK_SELF, &itskpri);

    while(cnt<10)
    {
        ercd = get_pri(TSK_SELF, &tskpri);

        if(tskpri!=itskpri)
        {
            /* タスク起動時の priority(itskpri)と違う場合、タスクを終了する */
            printf("    **** Exit task2()¥n");
            ext_tsk();
        }

        printf("    task2: waiting¥n");
        ercd = dly_tsk(2000);
        cnt++;
    }

    printf("    End task2()¥n");
}

/*********************************************/
/* task3                                     */
/*********************************************/
void task3(VP_INT exinf)
{
    ER ercd;
    PRI tskpri;

    printf("        Start task3()¥n");

    /* TASK2 の優先度を変更する */
    ercd = get_pri(TASK2, &tskpri);
    printf("        task3: TASK2 priority change = %d¥n", tskpri+1);
    ercd = chg_pri(TASK2, tskpri+1);

    printf("        End task3()¥n");
}
```

# 2 課題：起床待ちと起床

## 2.1 概要・課題

### 2.1.1 扱うプログラム(ex_slp.c)の概要

- main_tsk, sub_tsk の 2 つのタスクから構成。
- slp_tsk()を発行し起床待ち状態に移行した sub_tsk に対し、main_tsk から wup_tsk()によって sub_tsk の起床待ち状態を解除する。
- main_tsk は sub_tsk に対して 1 回 wup_tsk を呼び出し、遅延後 3 回 wup_tsk を呼び出す。

### 2.1.2 課題

1) サンプルプログラム ex_slp.c を動作させて、起床待ちおよびタスクの起床操作によってタスクが切替る様子を確認せよ。
2) sub_tsk の優先度を main_tsk より高くする事で、動作タイミングが変わることを確認せよ。

## 2.2 実習環境の前提

本課題では、コンソールと ITRON/T-Kernel のサービスコールだけで行う実習である。

### 2.2.1 使用する外部ライブラリ

- printf():コンソールへの文字列出力/標準 C ライブラリの仕様と同様

## 2.3 サンプルプログラム

### 2.3.1 system.cfg

```
/************************************/
/* Sample configuration file        */
/************************************/

INCLUDE("ex_slp.h");

#define STACK_SIZE 2048

CRE_TSK(TASK_MAIN, {TA_HLNG|TA_ACT, TASK_MAIN, main_tsk, 20, STACK_SIZE, NULL});
CRE_TSK(TASK_SUB, {TA_HLNG|TA_ACT, TASK_MAIN, sub_tsk, 22, STACK_SIZE, NULL});
```

### 2.3.2 ex_slp.c

```
#include  "kernel.h"

#include  "kernel_id.h"
#include  "ex_slp.h"

/*********************************************/
/* global datas                              */
/*********************************************/

/*********************************************/
/* main_tsk                                  */
/*********************************************/
void main_tsk(VP_INT exinf)
{
    ER ercd;
    VP_INT data;

    printf("Start main_tsk()¥n");

    while(1)
    {
        ercd = wup_tsk(TASK_SUB);
        printf("wakeup call¥n");

        ercd = dly_tsk(1000);

        ercd = wup_tsk(TASK_SUB);
        ercd = wup_tsk(TASK_SUB);
        ercd = wup_tsk(TASK_SUB);
        printf("wakeup call(3)¥n");

        ercd = dly_tsk(1000);
```

```
    }
}

/*********************************************/
/* sub_tsk                                   */
/*********************************************/
void sub_tsk(VP_INT exinf)
{
    ER ercd;

    printf("    Start sub_tsk()¥n");

    while(1)
    {
        ercd = slp_tsk();
        printf("    sub_tsk:wakeup sub_tsk()¥n");
    }
}
```

# 3 課題：セマフォ

## 3.1 概要・課題

### 3.1.1 扱うプログラム(ex_sem.c)の概要

- main_tsk, sub_tsk の 2 つのタスクから構成。
- wai_sem()によってセマフォ獲得待ち状態の sub_tsk に対し、main_tsk から sig_sem()によってセマフォを返却する。
- セマフォを獲得した sub_tsk はしばらく遅延する。
- その後 main_tsk はセマフォ獲得待ち状態に入り、sub_tsk が遅延から抜けてくると、セマフォを返却する。

### 3.1.2 課題

1) サンプルプログラム ex_sem.c を動作させて、セマフォの獲得および返却操作によってタスクが切替る様子を確認せよ。
2) sub_tsk の優先度を main_tsk より高くする事で、動作タイミングが変わることを確認せよ。

## 3.2 実習環境の前提

本課題では、コンソールと ITRON/T-Kernel のサービスコールだけで行う実習である。

### 3.2.1 使用する外部ライブラリ

- printf():コンソールへの文字列出力/標準 C ライブラリの仕様と同様

## 3.3 サンプルプログラム

### 3.3.1 system.cfg

```
/************************************/
/* Sample configuration file        */
/************************************/

INCLUDE("ex_sem.h");

#define STACK_SIZE 2048

CRE_TSK(TASK_MAIN, {TA_HLNG|TA_ACT, TASK_MAIN, main_tsk, 20, STACK_SIZE, NULL});
CRE_TSK(TASK_SUB, {TA_HLNG|TA_ACT, TASK_MAIN, sub_tsk, 22, STACK_SIZE, NULL});

CRE_SEM(ID_SEM1, {TA_TFIFO, 0, 10});
```

### 3.3.2 ex_sem.c

```
#include  "kernel.h"

#include  "kernel_id.h"
#include  "ex_sem.h"

/*********************************************/
/* global datas                              */
/*********************************************/
int data1 = 0;

/*********************************************/
/* main_tsk                                  */
/*********************************************/
void main_tsk(VP_INT exinf)
{
    ER ercd;

    printf("Start main_tsk()¥n");

    while(1)
    {
        ercd = wai_sem(ID_SEM1);
        printf("Get semaphore at main_tsk()¥n");

        data1++;

        printf("Release semaphore at main_tsk()¥n");
        ercd = sig_sem(ID_SEM1);

        ercd = dly_tsk(500);
```

```
    }
}

/*********************************************/
/* sub_tsk                                   */
/*********************************************/
void sub_tsk(VP_INT exinf)
{
    ER ercd;

    printf("    Start sub_tsk()¥n");

    ercd = dly_tsk(3000);

    ercd = sig_sem(ID_SEM1);

    while(1)
    {
        ercd = wai_sem(ID_SEM1);
        printf("    Get semaphore at sub_tsk()¥n");

        printf("    data1 = %d¥n", data1);

        ercd = dly_tsk(1000);

        printf("    Release semaphore at sub_tsk()¥n");
        ercd = sig_sem(ID_SEM1);
    }
}
```

# 4 課題：イベントフラグ

## 4.1 概要・課題

### 4.1.1 扱うプログラム(ex_flg.c)の概要

- main_tsk, sub_tsk の 2 つのタスクから構成。
- wai_flg()によってイベントフラグ待ち(0x00000101/ TWF_ORW)状態の sub_tsk に対し、main_tsk から set_flg()によって 0x00000001 をセットする。
- イベントフラグを獲得した sub_tsk は再びイベントフラグ待ち(0x00000101/ TWF_ANDW)に入る。
- main_tsk から set_flg()によって 0x00000001 をセットするだけでは sub_tsk には切り替わらない。
- main_tsk から set_flg()によって 0x00000100 をセットすることによって、sub_tsk はイベントフラグを受取る。

### 4.1.2 課題

1) サンプルプログラム ex_flg.c を動作させて、イベントフラグ待ちおよびセット操作によってタスクが切替る様子を確認せよ。
2) sub_tsk の優先度を main_tsk より高くする事で、動作タイミングが変わることを確認せよ。
3) もうひとつタスクを作成し、そのタスクで main_tsk がセットしていた 0x00000100 をセットするように変更(main_tsk がセットしていた 0x00000100 は削除)せよ。

## 4.2 実習環境の前提

本課題では、コンソールと ITRON/T-Kernel のサービスコールだけで行う実習である。

### 4.2.1 使用する外部ライブラリ

- printf():コンソールへの文字列出力/標準 C ライブラリの仕様と同様

## 4.3 サンプルプログラム

### 4.3.1 system.cfg

```
/************************************/
/* Sample configuration file        */
/************************************/

INCLUDE("ex_flg.h");

#define STACK_SIZE 2048

CRE_TSK(TASK_MAIN, {TA_HLNG|TA_ACT, TASK_MAIN, main_tsk, 20, STACK_SIZE, NULL});
CRE_TSK(TASK_SUB, {TA_HLNG|TA_ACT, TASK_MAIN, sub_tsk, 22, STACK_SIZE, NULL});

CRE_FLG(ID_FLG1, {TA_TFIFO|TA_CLR|TA_WMUL, 0});
```

### 4.3.2 ex_flg.c

```
#include  "kernel.h"

#include  "kernel_id.h"
#include  "ex_flg.h"

/*********************************************/
/* global datas                              */
/*********************************************/

/*********************************************/
/* main_tsk                                  */
/*********************************************/
void main_tsk(VP_INT exinf)
{
    ER ercd;

    printf("Start main_tsk()¥n");

    ercd = dly_tsk(3000);

    while(1)
    {
        printf("set_flg() 0x00000001¥n");
        ercd = set_flg(ID_FLG1, (FLGPTN)0x00000001);
        ercd = dly_tsk(1000);

        printf("set_flg() 0x00000001¥n");
        ercd = set_flg(ID_FLG1, (FLGPTN)0x00000001);
        ercd = dly_tsk(1000);
```

```
        printf("set_flg() 0x00000100¥n");
        ercd = set_flg(ID_FLG1, (FLGPTN)0x00000100);
        ercd = dly_tsk(1000);
    }
}

/*********************************************/
/* sub_tsk                                   */
/*********************************************/
void sub_tsk(VP_INT exinf)
{
    ER ercd;
    FLGPTN flgptn;

    printf("    Start sub_tsk()¥n");

    while(1)
    {
        ercd = wai_flg(ID_FLG1, (FLGPTN)0x00000101, TWF_ORW, &flgptn);
        printf("    Event flag(OR) %08x¥n", flgptn);

        ercd = wai_flg(ID_FLG1, (FLGPTN)0x00000101, TWF_ANDW, &flgptn);
        printf("    Event flag(AND) %08x¥n", flgptn);

        printf("¥n");
    }
}
```

# 5 課題：データキュー

## 5.1 概要・課題

### 5.1.1 扱うプログラム(ex_dtq.c)の概要

・ main_tsk, sub_tsk の 2 つのタスクから構成。
・ rcv_dtq()によってデータキュー受信待ち状態の main_tsk に対し、sub_tsk から snd_dtq()によってデータキューにデータを送信する。

### 5.1.2 課題

1) サンプルプログラム ex_dtq.c を動作させて、データキューの送信および受信操作によってタスクが切替る様子を確認せよ。
2) 送信されたデータが、受信側に伝わる様子を確認せよ。
3) データキューサイズを 0 に変更し、動作タイミングが変わることを確認せよ。

## 5.2 実習環境の前提

本課題では、コンソールと ITRON/T-Kernel のサービスコールだけで行う実習である。

### 5.2.1 使用する外部ライブラリ

- printf():コンソールへの文字列出力/標準 C ライブラリの仕様と同様

## 5.3 サンプルプログラム

### 5.3.1 system.cfg

```
/************************************/
/* Sample configuration file        */
/************************************/

INCLUDE("ex_dtq.h");

#define STACK_SIZE 2048

CRE_TSK(TASK_MAIN, {TA_HLNG|TA_ACT, TASK_MAIN, main_tsk, 20, STACK_SIZE, NULL});
CRE_TSK(TASK_SUB, {TA_HLNG|TA_ACT, TASK_MAIN, sub_tsk, 22, STACK_SIZE, NULL});

CRE_DTQ(ID_DTQ1, {TA_TPRI, 10, NULL});
```

### 5.3.2 ex_dtq.c

```
#include  "kernel.h"

#include  "kernel_id.h"
#include  "ex_dtq.h"

/*********************************************/
/* global datas                              */
/*********************************************/

/*********************************************/
/* main_tsk                                  */
/*********************************************/
void main_tsk(VP_INT exinf)
{
    ER ercd;
    VP_INT data;

    printf("Start main_tsk()¥n");

    ercd = dly_tsk(3000);

    while(1)
    {
        ercd = rcv_dtq(ID_DTQ1, &data);
        printf("rcv_dtq() %d¥n", (int)data);
    }
}

/*********************************************/
/* sub_tsk                                   */
```

```
/*********************************************/
void sub_tsk(VP_INT exinf)
{
    ER ercd;

    printf("    Start sub_tsk()¥n");

    while(1)
    {
        printf("    snd_dtq() 500¥n");
        ercd = snd_dtq(ID_DTQ1, (VP_INT)500);

        printf("    snd_dtq() 123¥n");
        ercd = snd_dtq(ID_DTQ1, (VP_INT)123);

        printf("    snd_dtq() 999¥n");
        ercd = snd_dtq(ID_DTQ1, (VP_INT)999);

        ercd = dly_tsk(1000);
        printf("¥n");
    }
}
```

# 6 課題：メールボックスと固定長メモリプール

## 6.1 概要・課題

### 6.1.1 扱うプログラム(ex_mbx.c)の概要

- main_tsk, sub_tsk の 2 つのタスクから構成。
- rcv_mbx()によってメールボックス受信待ち状態の main_tsk に対し、sub_tsk から snd_mbx()によってメールボックスに、固定長メモリプールから獲得したメッセージデータを送信する。
- メッセージを受信した main_tsk はデータを表示し、メッセージデータを固定長メモリプールへ返却する。

### 6.1.2 課題

1) サンプルプログラム ex_mbx.c を動作させて、メールボックスの送信および受信操作によってタスクが切替る様子を確認せよ。
2) 送信されたメッセージが、受信側に伝わる様子を確認せよ。
3) メールボックスの代わりにデータキューを使用して、同じ動作になるよう変更せよ。またメールボックスとデータキューの違いを考察せよ。

## 6.2 実習環境の前提

本課題では、コンソールと ITRON/T-Kernel のサービスコールだけで行う実習である。

### 6.2.1 使用する外部ライブラリ

- printf():コンソールへの文字列出力/標準 C ライブラリの仕様と同様

## 6.3 サンプルプログラム

### 6.3.1 system.cfg

```
/************************************/
/* Sample configuration file        */
/************************************/

INCLUDE("ex_mbx.h");

#define STACK_SIZE 2048

CRE_TSK(TASK_MAIN, {TA_HLNG|TA_ACT, TASK_MAIN, main_tsk, 20, STACK_SIZE, NULL});
CRE_TSK(TASK_SUB, {TA_HLNG|TA_ACT, TASK_MAIN, sub_tsk, 22, STACK_SIZE, NULL});

CRE_MPF(ID_MPF1, {TA_TFIFO, 10, MSG_PACKET, NULL});
CRE_MBX(ID_MBX1, {TA_MFIFO, 10, NULL});
```

### 6.3.2 ex_mbx.c

```
#include  "kernel.h"

#include  "kernel_id.h"
#include  "ex_mbx.h"

/************************************************/
/* global datas                                 */
/************************************************/

/************************************************/
/* main_tsk                                     */
/************************************************/
void main_tsk(VP_INT exinf)
{
    ER ercd;
    T_MSG_PACKET* p_msg_data;

    printf("Start main_tsk()¥n");

    ercd = dly_tsk(3000);

    while(1)
    {
        ercd = rcv_mbx(ID_MBX1, (T_MSG**)&p_msg_data);
        printf("rcv_mbx() address<%08x>¥n", p_msg_data);
        printf("rcv_data : <%s>¥n¥n", p_msg_data->msg);

        /* Release a mailbox packet */
        ercd = rel_mpf(ID_MPF1, p_msg_data);
```

```
    }
}

/*********************************************/
/* sub_tsk                                   */
/*********************************************/
void sub_tsk(VP_INT exinf)
{
    ER ercd;
    T_MSG_PACKET* p_msg_data;
    int count = 0;

    printf("    Start sub_tsk()¥n");

    while(1)
    {
        /* Get mailbox packet area */
        ercd = get_mpf(ID_MPF1, (VP*)&p_msg_data);
        sprintf(&p_msg_data->msg, "count=%d", ++count);

        /* Set mailbox */
        printf("    snd_mbx()¥n");
        ercd = snd_mbx(ID_MBX1, (T_MSG*)p_msg_data);

        ercd = dly_tsk(1000);
        printf("¥n");
    }
}
```

# 7 課題：周期ハンドラ

## 7.1 概要・課題

### 7.1.1 扱うプログラム(ex_cyc.c)の概要

・ main_tsk(タスク)および cyc_func(周期ハンドラ)から構成。
・ cyc_func で一定周期にインクリメントされるデータを、main_tsk が定期的に更新状態を検査し、変化があれば表示する。

### 7.1.2 課題

1) サンプルプログラム ex_cyc.c を動作させて、定期的にデータが更新される様子を確認せよ。
2) main_tsk で定期的に監視する時間はどの程度が最適か考察し、実際に変更して確認せよ。
3) main_tsk を待ち状態とし、cyc_func でデータが更新された時に待ち解除する方法を検討し、動作確認せよ。

## 7.2 実習環境の前提

本課題では、コンソールと ITRON/T-Kernel のサービスコールだけで行う実習である。

### 7.2.1 使用する外部ライブラリ

- printf():コンソールへの文字列出力/標準 C ライブラリの仕様と同様

## 7.3 サンプルプログラム

### 7.3.1 system.cfg

```
/************************************/
/* Sample configuration file        */
/************************************/

INCLUDE("ex_cyc.h");

#define STACK_SIZE 2048

CRE_TSK(TASK_MAIN, {TA_HLNG|TA_ACT, TASK_MAIN, main_tsk, 20, STACK_SIZE, NULL});
CRE_CYC(ID_CYC1, {TA_HLNG|TA_STA, ID_CYC1, cyc_func, 2000, 2000});
```

### 7.3.2 ex_cyc.c

```
#include  "kernel.h"

#include  "kernel_id.h"
#include  "ex_cyc.h"

/*********************************************/
/* global datas                              */
/*********************************************/
int data = 0;

/*********************************************/
/* main_tsk                                  */
/*********************************************/
void main_tsk(VP_INT exinf)
{
    ER ercd;
    int ldata;

    printf("Start main_tsk()¥n");

    ercd = loc_cpu();
    ldata = data;
    ercd = unl_cpu();

    while(1)
    {
        ercd = dly_tsk(500);

        ercd = loc_cpu();
        if(ldata!=data)
        {
            printf("ldata <%d>¥n", ldata);
```

```
            ldata = data;
        }
        ercd = unl_cpu();
    }
}

/*********************************************/
/* cyc_func                                  */
/*********************************************/
void cyc_func(VP_INT exinf)
{
    ER ercd;

    data++;
}
```

# 8 課題：周期処理の比較

## 8.1 概要・課題

### 8.1.1 扱うプログラム(ex_polling.c)の概要

・ task1, task2 の 2 つのタスクおよび cyc_func(周期ハンドラ)から構成。
・ task1, task2, cyc_func それぞれで、定期的にメッセージを表示する。

### 8.1.2 課題

1) サンプルプログラム ex_polling.c を動作させて、それぞれ定期的にメッセージが表示される様子を確認せよ。
2) 定期的に動作するそれぞれの方法について、それぞれの特徴を考察せよ。

## 8.2 実習環境の前提

本課題では、コンソールと ITRON/T-Kernel のサービスコールだけで行う実習である。

### 8.2.1 使用する外部ライブラリ

- printf():コンソールへの文字列出力/標準 C ライブラリの仕様と同様

## 8.3 サンプルプログラム

### 8.3.1 system.cfg

```
/************************************/
/* Sample configuration file        */
/************************************/

INCLUDE("ex_polling.h");

#define STACK_SIZE 2048

CRE_TSK(TASK1, {TA_HLNG|TA_ACT, TASK1, task1, 20, STACK_SIZE, NULL});
CRE_TSK(TASK2, {TA_HLNG|TA_ACT, TASK2, task2, 20, STACK_SIZE, NULL});

CRE_CYC(ID_CYC1, {TA_HLNG|TA_STA, ID_CYC1, cyc_func, 2000, 2000});
```

### 8.3.2 ex_polling.c

```
#include  "kernel.h"

#include  "kernel_id.h"
#include  "ex_polling.h"

/*********************************************/
/* task1                                     */
/*********************************************/
void task1(VP_INT exinf)
{
    ER ercd;

    while(1)
    {
        ercd = dly_tsk(1000);

        /* dly_tsk を使った監視処理 */
        printf("1:polling by dly_tsk()¥n");
    }
}

/*********************************************/
/* task2                                     */
/*********************************************/
void task2(VP_INT exinf)
{
    ER ercd;

    while(1)
    {
```

```
        ercd = tslp_tsk(1000);

        /* tslp_tsk を使った監視処理 */
        printf("    2:polling by tslp_tsk()¥n");
    }
}

/*********************************************/
/* cyc_func                                  */
/*********************************************/
void cyc_func(VP_INT exinf)
{
    /* 周期ハンドラでの監視処理 */
    printf("        *** polling by cyclic handler¥n");
}
```

# 9 課題：同じ処理を行う複数のタスク(コードシェア)

## 9.1 概要・課題

### 9.1.1 扱うプログラム(ex_tsk2.c)の概要

- test_tsk 1 つのみで構成。
- test_tsk は ID=TEST_TSK1 および ID=TEST_TSK1 の二つのタスクとして起動され、定期的にメッセージを表示する。

### 9.1.2 課題

1) サンプルプログラム ex_tsk2.c を動作させて、それぞれ定期的にメッセージが表示される様子を確認せよ。
2) 同一コードで複数のタスクを動作させる利点を考察せよ。

## 9.2 実習環境の前提

本課題では、コンソールと ITRON/T-Kernel のサービスコールだけで行う実習である。

### 9.2.1 使用する外部ライブラリ

- printf():コンソールへの文字列出力/標準 C ライブラリの仕様と同様

## 9.3 サンプルプログラム

### 9.3.1 system.cfg

```
/************************************/
/* Sample configuration file        */
/************************************/

INCLUDE("ex_tsk2.h");

#define STACK_SIZE 2048

CRE_TSK(TEST_TSK1, {TA_HLNG|TA_ACT, TEST_TSK1, test_tsk, 15, STACK_SIZE, NULL});
CRE_TSK(TEST_TSK2, {TA_HLNG|TA_ACT, TEST_TSK2, test_tsk, 15, STACK_SIZE, NULL});
```

### 9.3.2 ex_tsk2.c

```
#include  "kernel.h"

#include  "kernel_id.h"
#include  "ex_tsk2.h"

/*********************************************/
/* test_tsk                                  */
/*********************************************/
void test_tsk(VP_INT exinf)
{
    ER ercd;
    ID tskid;

    ercd = get_tid(&tskid);

    while(1)
    {
        printf("task id = %d¥n", tskid);

        ercd = dly_tsk(1000);
    }
}
```

# 10課題：チャタリングの除去

## 10.1　概要・課題

### 10.1.1 扱うプログラム(ex_noize.c)の概要

・ main_tsk(タスク)および button_handler(ボタン押下時の割込みハンドラ)で構成。
・ ボタンが押された場合 button_handler が呼び出され、isig_sem()を利用してイベントを送信する。
・ main_tsk は wai_sem()によって、イベントの発生を受け取る。

### 10.1.2 課題

1) サンプルプログラム ex_noize.c を動作させて、ボタン押下時に発生しているチャタリング(スイッチの接点接触時に、複数回 ON/OFF が繰り返される現象)が発生している様子を確認せよ。
2) #if 0 を#if 1 に変更(コードを有効に)し、チャタリングが除去される事を確認せよ。
3) isig_sem()/wai_sem()の組み合わせ以外の、別の方法を検討し、動作確認せよ。

## 10.2　実習環境の前提

本課題では、コンソールとボタン押下シミュレーション(チャタリング発生)および ITRON/T-Kernel のサービスコールで行う実習である。

### 10.2.1 使用する外部ライブラリ

- printf():コンソールへの文字列出力/標準 C ライブラリの仕様と同様
- チャタリング割り込みが発生する、割込みシミュレーション処理

## 10.3　サンプルプログラム

### 10.3.1 system.cfg

```
/************************************/
/* Sample configuration file        */
/************************************/
INCLUDE("ex_noize.h");

#define STACK_SIZE 2048

DEF_INH(BUTTON_INT, {0, button_handler});

CRE_TSK(TASK_MAIN, {TA_HLNG|TA_ACT, TASK_MAIN, main_tsk, 20, STACK_SIZE, NULL});
CRE_SEM(ID_SEM1, {TA_TFIFO, 0, 10});

ATT_INI({TA_HLNG, 0, init_demo});
```

### 10.3.2 ex_noize.c

```
#include  "kernel.h"

#include  "kernel_id.h"
#include  "ex_noize.h"

/*********************************************/
/* button_handler                            */
/*********************************************/
void button_handler(void)
{
    isig_sem(ID_SEM1);
}

/*********************************************/
/* init_demo                                 */
/*********************************************/
void init_demo(VP_INT exinf)
{
    /* Initialize demo dialog. */
    init_window();
}

/*********************************************/
/* main_tsk                                  */
/*********************************************/
void main_tsk(VP_INT exinf)
{
    int count = 0;
```

```
    while(1)
    {
        if(wai_sem(ID_SEM1)==E_OK)
        {
#if 0
            /* この部分がチャタリングを除去するための処理になります */
            while(twai_sem(ID_SEM1, 100)!=E_TMOUT)
            {
                ;
            }
#endif
            /* ボタンが押されました */
            count++;
            printf("Button %d times pushed!¥n", count);
        }
    }
}
```

# 11課題：自動ドアシミュレーション

## 11.1　概要・課題

### 11.1.1 扱うプログラム(door.c)の概要

・ 本課題は自動ドアの動作のシミュレーションを行うものである。
・ gate_tsk （タスク）、opn_handler （ドアセンサーの割込みハンドラ）および stp_handler(非常停止ボタン押下時の割込みハンドラ)で構成。
・ ドアセンサーが押された場合 opn_handler が呼び出され、ipsnd_dtq ()によってイベントを送信する。非常停止ボタンが押された場合は stp_handler()が呼び出される。stp_handler()の初期状態では何も行っていない。
・ main_tsk はドア操作の仮プログラムが入っている。

### 11.1.2 課題

1) サンプルプログラム door.c の gate_tsk()を変更し、ドアセンサーによって動作する自動ドアとして機能させよ。
2) 非常停止ボタンによって動作するタスクを追加し、gate_tsk()と連携した非常停止動作を作成せよ。非常停止からの復帰についても、安全対策のために必要な処理を考慮して追加せよ。

本課題は、タスク間通信、タスク優先度の設定、複数タスクで連携して動作させる等の総合的なプログラミングを実習するものである。

## 11.2　実習環境の前提

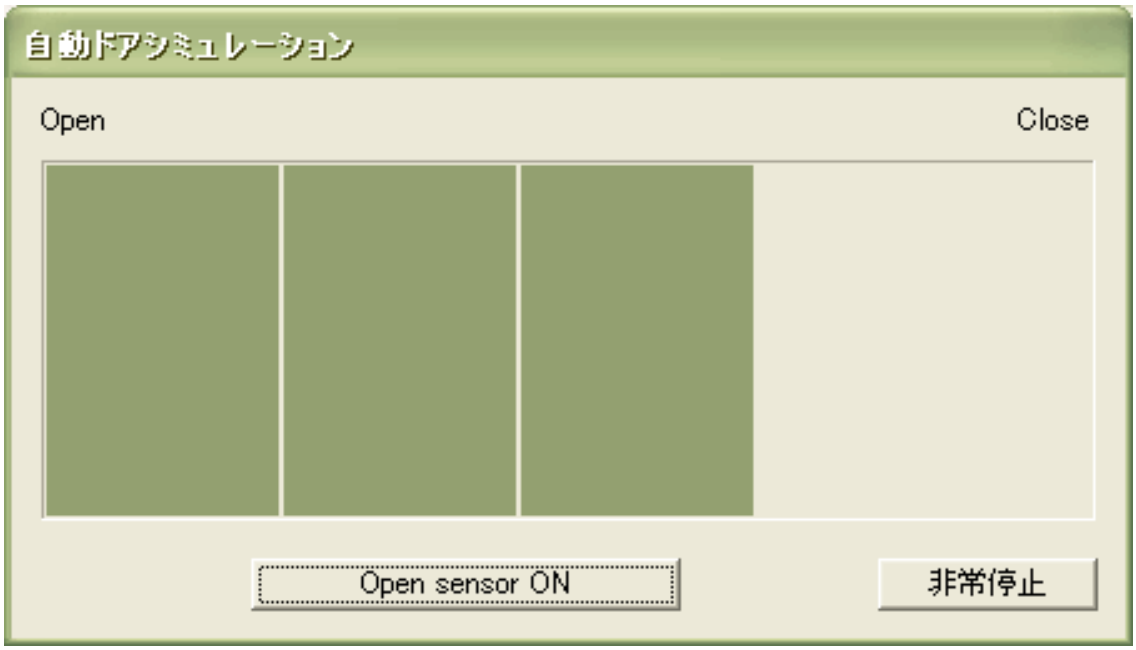
本課題では、自動ドアシミュレーション環境および ITRON/T-Kernel のサービスコールで行う実習である。

### 11.2.1 使用する外部ライブラリ

- printf():コンソールへの文字列出力/標準 C ライブラリの仕様と同様
- ドアセンサーおよび非常停止ボタンを含む、自動ドアシミュレーション処理

- void init_window(void);　　シミュレーション初期化
- void set_pos(int pos);　　ドア位置の指定(0～100)

## 11.3　シミュレーション画面



## 11.4　サンプルプログラム

### 11.4.1 system.cfg

```
/************************************/
/* Sample configuration file        */
/************************************/

INCLUDE("door.h");

#define STACK_SIZE 2048

CRE_TSK(GATE_TSK, {TA_HLNG|TA_ACT, 0, gate_tsk, 20, STACK_SIZE, NULL});

CRE_DTQ(BTN_DTQ, {TA_TPRI, 5, NULL});

DEF_INH(OPN_INT, {0, opn_handler});
DEF_INH(STP_INT, {0, stp_handler});


/* User initialize function */
ATT_INI({TA_HLNG, 0, init_demo});
```

### 11.4.2 door.c

```
#include  "kernel.h"

#include  "kernel_id.h"
#include  "door.h"

void init_window(void);
void set_pos(int pos);

/*********************************************/
/* opn_handler                               */
/*********************************************/
void opn_handler(void)
{
    ipsnd_dtq(BTN_DTQ, (VP_INT)OPN_INT);
}

/*********************************************/
/* stp_handler                               */
/*********************************************/
void stp_handler(void)
{
    ;
}
```

```
/*********************************************/
/* init_demo                                 */
/*********************************************/
void init_demo(VP_INT exinf)
{
    ER ercd;

    /* Initialize demo dialog. */
    init_window();
}

/*********************************************/

int gate_pos = 0;   /* ドアの位置 */

/*********************************************/
/* gate_tsk                                  */
/*********************************************/
void gate_tsk(VP_INT exinf)
{
    ER ercd;
    VP_INT data;
    int ctrl = 0;
    int wait_count = 0;

    /* ドアの最初の位置を設定 */
    set_pos(gate_pos);

    while(1)
    {
        /* TODO:ドアセンサーからの情報を得るために、*/
        /* このあたりの処理を修正する必要があります */
        dly_tsk(50);

        if(ctrl==0 && 100<wait_count)
        {
            /* ドア open 開始 */
            printf("door sensor on¥n");
            ctrl = -1;
        }

        if(ctrl!=0)
        {
            if(((int)ctrl<0 && 0<gate_pos) || (0<(int)ctrl && gate_pos<100))
            {
                gate_pos += (int)ctrl;

                /* ドア位置の設定 */
                set_pos(gate_pos);
            }
            else
            {
                /* ドア動作の停止 */
                ctrl = 0;
                wait_count = 0;
```

```
                printf("door is stoped¥n");
            }
        }
        else
        {
            if(gate_pos==0)
            {
                /* ドアを閉じるまでしばらく待つ */
                wait_count++;
                if(100<wait_count)
                {
                    /* ドアclose開始 */
                    printf("door close¥n");
                    ctrl = 1;
                }
            }
            else
            {
                /* TODO:ドアを開くための、テスト処理です             */
                /* ドアセンサーで制御ができれば、この処理は不要です */
                wait_count++;
            }
        }
    }
}
```

# 解答例

課題解決方法の一例を示したものであり、別の方法でも解決可能と思われる。必ずしも本解答例が唯一の解決方法では無い事に留意願いたい。

また最初のサンプルからの変更点は、#if ディレクティブが記述されている部分になるので、元のコードと比較して確認してください。

## 11.5　自動ドアシミュレーション

### 11.5.1 system.cfg

```
/************************************/
/* Sample configuration file        */
/************************************/

INCLUDE("door.h");

#define STACK_SIZE 2048

CRE_TSK(GATE_TSK, {TA_HLNG|TA_ACT, 0, gate_tsk, 20, STACK_SIZE, NULL});
#if 1
    CRE_TSK(EMG_TSK, {TA_HLNG|TA_ACT, 0, emg_tsk, 10, STACK_SIZE, NULL});
#endif

CRE_DTQ(BTN_DTQ, {TA_TPRI, 5, NULL});

DEF_INH(OPN_INT, {0, opn_handler});
DEF_INH(STP_INT, {0, stp_handler});


/* User initialize function */
ATT_INI({TA_HLNG, 0, init_demo});
```

### 11.5.2 door.c

```
#include  "kernel.h"

#include  "kernel_id.h"
#include  "door.h"

void init_window(void);
void set_pos(int pos);

/*********************************************/
/* opn_handler                               */
/*********************************************/
void opn_handler(void)
{
    ipsnd_dtq(BTN_DTQ, (VP_INT)OPN_INT);
}

/*********************************************/
/* stp_handler                               */
/*********************************************/
void stp_handler(void)
```

```
{
#if 1
    iwup_tsk(EMG_TSK);
#endif
}

/*********************************************/
/* init_demo                                 */
/*********************************************/
void init_demo(VP_INT exinf)
{
    ER ercd;

    /* Initialize demo dialog. */
    init_window();
}

/*********************************************/

int gate_pos = 0;   /* ドアの位置 */

/*********************************************/
/* gate_tsk                                  */
/*********************************************/
void gate_tsk(VP_INT exinf)
{
    ER ercd;
    VP_INT data;
    int ctrl = 0;
    int wait_count = 0;

    /* ドアの最初の位置を設定 */
    set_pos(gate_pos);

#if 1
    /* 溜まっているイベントをクリア */
    while((ercd=prcv_dtq(BTN_DTQ, &data))==E_OK)
    {
        ;
    }

    /* 安全対策のため、最初のドア open まで待つ */
    if((ercd=rcv_dtq(BTN_DTQ, &data))==E_OK)
    {
        printf("1st door sensor on¥n");
        ctrl = -1;
    }
#endif

    while(1)
    {
#if 1
        if((ercd=trcv_dtq(BTN_DTQ, &data, 50))==E_OK)
        {
            /* ドア open 開始 */
```

```
            printf("door sensor on¥n");
            ctrl = -1;
        }
#endif

        if(ctrl!=0)
        {
            if(((int)ctrl<0 && 0<gate_pos) || (0<(int)ctrl && gate_pos<100))
            {
                gate_pos += (int)ctrl;

                /* ドア位置の設定 */
                set_pos(gate_pos);
            }
            else
            {
                /* ドア動作の停止 */
                ctrl = 0;
                wait_count = 0;
                printf("door is stoped¥n");
            }
        }
        else
        {
            if(gate_pos==0)
            {
                /* ドアを閉じるまでしばらく待つ */
                wait_count++;
                if(100<wait_count)
                {
                    /* ドア close 開始 */
                    printf("door close¥n");
                    ctrl = 1;
                }
            }
#if 0
            /* センサーからの処理を入れたので、このテスト処理は削除します */
            else
            {
                /* TODO:ドアを開くための、テスト処理です            */
                /* ドアセンサーで制御ができれば、この処理は不要です */
                wait_count++;
            }
#endif
        }
    }
}

#if 1
/*********************************************/
/* emg_tsk                                   */
/*********************************************/
void emg_tsk(VP_INT exinf)
{
    ER ercd;
```

```
    /* 非常停止対応処理 */
    while(1)
    {
        ercd = slp_tsk();

        printf("emg. button pushed!¥n");

        /* GATE_TSK を強制終了し、再度起動する */
        ercd = ter_tsk(GATE_TSK);
        ercd = act_tsk(GATE_TSK);
    }
}
#endif
```

トロンフォーラム
実習：組込みリアルタイム・プログラミング(ITRON 初級編)
プログラミング演習テキスト

---

2016 年 8 月 24 日発行

発行所
トロンフォーラム
(YRP ユビキタス・ネットワーキング研究所内)
〒141-0031 東京都品川区西五反田 2-12-3 第一誠実ビル 9F
URL: http://www.tron.org/
TEL: 03-5437-0572　FAX: 03-5437-2399

---



---