Consignes pour les TP

Temps Réel

SEEC & SETIA

1. Préparation des TP

Cette préparation consiste à comprendre le principe de fonctionnement de principe des noyaux temps réels par l'étude d'exemples fournis par Texas Instruments.

Chaque TP correspond à un projet de Code Composer Studio à récupérer dans les répertoires :

C:\CCStudio_V3.1\examples\dsk6416

C:\CCStudio_V3.1\tutorial\dsk6416

Copier ces dossiers complets dans votre compte « Z:\ »

Le but des TP est de comprendre les programmes fournis par Texas Instruments

Il faut donc lire les listing de programmes et les comprendre : c'est le travail de préparation

2. Compte rendu de TP

Pour chaque compte rendu de TP sur les exemples de Texas instruments on demande de :

- Étudier les objets crées par l'outil de configuration de DSP/BIOS.
- Donner l'analyse structurée du programme.
- Donner la liste et le rôle de toutes les variables.
- Analyser le fonctionnement de toutes les tâches crées.
- Visualiser et commenter les diagrammes de commutation des tâches.
- Faire des modifications en prévoir les conséquences et vérifier.

TP1-A Les LEDS

Ce programme ce trouve dans le sous dossier :

C:\CCStudio_V3.1\examples\dsk6416\bsl\led

```
/*
 *  Copyright 2003 by Spectrum Digital Incorporated.
 *  All rights reserved. Property of Spectrum Digital Incorporated.
 */

/*
 *  ======== led.c ========
 *
 *  This example blinks LED #0 at a rate of about 2.5 times per second using
 *  the LED module of the the DSK6416 Board Support Library.  The example
 *  also reads the state of DIP switch #3 and lights LED #3 if the switch
 *  is depressed or turns it off if the switch is not depressed.
 *
 *  The purpose of this example is to demonstrate basic BSL usage as well
 *  as provide a project base for your own code.
 *
 *  Please see the DSK6416 help file for more detailed information.
 */

/*
 *  DSP/BIOS is configured using the DSP/BIOS configuration tool.  Settings
 *  for this example are stored in a configuration file called led.cdb.  At
 *  compile time, Code Composer will auto-generate DSP/BIOS related files
 *  based on these settings.  A header file called ledcfg.h contains the
 *  results of the autogeneration and must be included for proper operation.
 *  The name of the file is taken from led.cdb and adding cfg.h.
 */
#include "ledcfg.h"

/*
 *  The Board Support Library is divided into several modules, each
 *  of which has its own include file.  The file dsk6416.h must be included
 *  in every program that uses the BSL.  This example also includes
 *  dsk6416_led.h and dsk6416_dip.h because it uses the LED and DIP modules.
 */
#include "dsk6416.h"
#include "dsk6416_led.h"
#include "dsk6416_dip.h"


/*
 *  main() - Main code routine, initializes BSL and runs LED application
 */

/*
 *  EXTRA: Pressing DIP switch #3 changes LED #3 from off to on.
 */
```

```
void main()
{
   /* Initialize the board support library, must be first BSL call */
   DSK6416_init();

   /* Initialize the LED and DIP switch modules of the BSL */
   DSK6416_LED_init();
   DSK6416_DIP_init();

   while(1)
   {
      /* Toggle LED #0 */
      DSK6416_LED_toggle(0);

      /* Check DIP switch #3 and light LED #3 accordingly, 0 = switch pressed */
      if (DSK6416_DIP_get(3) == 0)
         /* Switch pressed, turn LED #3 on */
         DSK6416_LED_on(3);
      else
         /* Switch not pressed, turn LED #3 off */
         DSK6416_LED_off(3);

      /* Spin in a software delay loop for about 200ms */
      DSK6416_waitusec(200000);
   }
}
```

TP1-B Les LEDS Périodiques

Ce programme ce trouve dans le sous dossier :

C:\CCStudio_V3.1\examples\dsk6416\bsl\ledprd

```
/*
 *  Copyright 2002 by Spectrum Digital Incorporated.
 *  All rights reserved. Property of Spectrum Digital Incorporated.
 */
/*
 *  ======== ledprd.c ========
 *  This example blinks LED #0 at a rate of about 4 times per second using
 *  the LED module of the the 6416 DSK Board Support Library.  The example
 *  also reads the state of DIP switch #3 and lights LED #3 if the switch
 *  is depressed or turns it off if the switch is not depressed.
 *
 *  When the program is run, DSP/BIOS initializes itself and calls the main()
 *  function.  Main() initializes the BSL then exits and returns control back
 *  to DSP/BIOS.  The real work is done inside blinkLED0() which is a DSP/BIOS
 *  periodic thread that is run every 125ms.
 *
 *  A second thread named blinkLED1() is also included that blinks LED #1
 *  asynchronously with blinkLED0() to demonstrate DSP/BIOS multitasking.
 *  It is not enabled by default but can be added by creating a new periodic
 *  thread entry for it in the DSP/BIOS scheduler.
 *
 *  Please see the 6416 DSK help file under Software/Examples for more
 *  detailed information.
 */
/*
 *  DSP/BIOS is configured using the DSP/BIOS configuration tool.  Settings
 *  for this example are stored in a configuration file called ledprd.cdb.  At
 *  compile time, Code Composer will auto-generate DSP/BIOS related files
 *  based on these settings.  A header file called ledprdcfg.h contains the
 *  results of the autogeneration and must be included for proper operation.
 *  The name of the file is taken from ledprd.cdb and adding cfg.h.
 */
#include "ledprdcfg.h"

/*
 *  The 6416 DSK Board Support Library is divided into several modules, each
 *  of which has its own include file.  The file dsk6416.h must be included
 *  in every program that uses the BSL.  This example also includes
 *  dsk6416_led.h and dsk6416_dip.h because it uses the LED and DIP modules.
 */

#include "dsk6416.h"
#include "dsk6416_led.h"
#include "dsk6416_dip.h"
```

```
/*
 *  blinkLED0() - Blink LED #0 and set LED #3 based on the state of DIP switch
 *              #3.  If the switch is down, the LED is turned on.  If the
 *              switch is up, the LED is turned off.
 *
 *              blinkLED0 is a periodic thread that is called every 200ms
 *              from the DSP/BIOS scheduler.  It is configured in the
 *              DSP/BIOS configuration file (ledprd.cdb) under Scheduling
 *              --> PRD --> PRD_blinkLED0.  Right click PRD_blinkLED0 and
 *              select Properties to view its settings.
 */
void blinkLED0()
{
   /* Toggle LED #0 */
   DSK6416_LED_toggle(0);

   /* Check DIP switch #3 and light LED #3 accordingly, 0 = switch pressed */
   if (DSK6416_DIP_get(3) == 0)
      /* Switch pressed, turn LED #3 on */
      DSK6416_LED_on(3);
   else
      /* Switch pressed, turn LED #3 off */
      DSK6416_LED_off(3);
}
/*
 *  blinkLED1() - Blink LED #1.
 *
 *              blinkLED1 is a periodic thread that can be called from the
 *              DSP/BIOS scheduler.  By default, it is not active.  To make
 *              it active, create a new PRD entry called PRD_blinkLED1 in
 *              the DSP/BIOS configuration file (ledprd.cdb) under
 *              Scheduling --> PRD.  Right click PRD_blinkLED1 and select
 *              properties to configure it.  Change the function field to
 *              _blinkLED1 and the period field to 100 to make blinkLED1()
 */
void blinkLED1()
{
   /* Toggle LED #1 */
   DSK6416_LED_toggle(1);
}

/*
 *  main() - Initialize BSL then drop into DSP/BIOS idle loop
 */
void main()
{
   /* Initialize the board support library, must be first BSL call */
   DSK6416_init();

   /* Initialize the LED and DIP switch modules of the BSL */
   DSK6416_LED_init();
   DSK6416_DIP_init();
}
```

TP2 Horloge

Ce programme ce trouve dans le sous dossier :

C:\CCStudio_V3.1\examples\dsk6416\bios\clktest

```
/*
 *  Copyright 2003 by Texas Instruments Incorporated.
 *  All rights reserved. Property of Texas Instruments Incorporated.
 *  Restricted rights to use, duplicate or disclose this code are
 *  granted through contract.
 *
 *  ======== clktest.c =======
 *  In this example, a task goes to sleep # number of ticks and
 *  prints the time after it wakes up.
 *
 */
#include <std.h>

#include <log.h>
#include <clk.h>
#include <tsk.h>

#include "clktestcfg.h"

/*
 *  ======== main ========
 */
Void main()
{
    LOG_printf(&trace, "clktest example started.\n");
}

Void taskFxn(Arg value_arg)
{
    Int value = ArgToInt (value_arg);
    Uns ticks;
#ifdef _28_
    LOG_printf(&trace, "The time in task is: %d ticks", (Arg)TSK_time());
#else
    LOG_printf(&trace, "The time in task is: %d ticks", (Int)TSK_time());
#endif

    ticks = (value * CLK_countspms()) / CLK_getprd();

#ifdef _28_
    LOG_printf(&trace, "task going to sleep for %d ticks... ", (Arg)ticks);
#else
    LOG_printf(&trace, "task going to sleep for %d ticks... ", ticks);
#endif

    TSK_sleep(ticks);

#ifdef _28_
    LOG_printf(&trace, "...awake! Time is: %d ticks", (Arg)TSK_time());
#else
    LOG_printf(&trace, "...awake! Time is: %d ticks", (Int)TSK_time());
#endif
}
```

TP 3 TSK

Ce programme ce trouve dans le sous dossier :

C:\CCStudio_V3.1\tutorial\dsk6416\tsktest

```
/*
 *  ======== tsktest.c =======
 *  In this example, 3 tasks have been created with the Configuration
 *  Tool.  Each task has a computation loop consisting of a LOG_printf()
 *  followed by a TSK_yield().  This causes a round robin scheduling for
 *  these tasks of equal priority.
 */

#include <std.h>

#include <log.h>
#include <tsk.h>

#define NLOOPS  5

extern LOG_Obj trace;   /* Created with the Config Tool */

Void task(Int id);      /* Function for tasks created with Config Tool */

/*
 *  ======== main ========
 */
Void main()
{
}

/*
 *  ======== task ========
 */
Void task(Int id)
{
    Int     i;

    for (i = 0; i < NLOOPS ; i++) {
        LOG_printf(&trace, "Loop %d: Task %d Working", i, id);
        TSK_yield();
    }
    LOG_printf(&trace, "Task %d DONE", id);
}
```

TP4 - MEM

Ce programme ce trouve dans le sous dossier :

C:\CCStudio_V3.1\tutorial\dsk6416\memtest

```
/*
 *  Copyright 2003 by Texas Instruments Incorporated.
 *  All rights reserved. Property of Texas Instruments Incorporated.
 *  Restricted rights to use, duplicate or disclose this code are
 *  granted through contract.
 *
 */
/* "@(#) DSP/BIOS 4.90.270 01-05-04 (bios,dsk6416-f06)" */
/*
 *  ======== memtest.c ========
 *  This program allocates and frees memory from
 *  different memory segments.
 */

#include <std.h>

#include <log.h>
#include <mem.h>

#include "memtestcfg.h"


#define NALLOCS 2       /* # of allocations from each segment */
#define BUFSIZE 128     /* size of allocations */

extern Int SEG0;

static Void printmem(Int segid);

/*
 *  ======== main ========
 */
Void main()
{
    LOG_printf(&trace, "memtest example started.\n");
}

/*
 *  ======== initTask ========
 */
Void initTask()
{
    Int i;
    Ptr ram[NALLOCS];

    LOG_printf(&trace, "before allocating ...");

    /* print initial memory status */
    printmem(SEG0);

    LOG_printf(&trace, "allocating ...");

    /* allocate some memory from each segment */
```

```
    for (i = 0; i < NALLOCS; i++) {
        ram[i] = MEM_alloc(SEG0, BUFSIZE, 0);
#ifdef _28_
        LOG_printf(&trace, "seg %d: ptr = %p", (Arg)SEG0, (Arg)ram[i]);
#else
        LOG_printf(&trace, "seg %d: ptr = %p", SEG0, ram[i]);
#endif
    }

    LOG_printf(&trace, "after allocating ...");

    /* print memory status */
    printmem(SEG0);

    /* free memory */
    for (i = 0; i < NALLOCS; i++) {
        MEM_free(SEG0, ram[i], BUFSIZE);
    }

    LOG_printf(&trace, "after freeing ...");

    /* print memory status */
    printmem(SEG0);

    LOG_printf(&trace, "Test Complete");
}

/*
 *  ======== printmem ========
 */
static Void printmem(Int segid)
{
    MEM_Stat    statbuf;

    MEM_stat(segid, &statbuf);

#ifdef _28_
    LOG_printf(&trace, "seg %d: O 0x%x", (Arg)segid,(Arg)statbuf.size);
    LOG_printf(&trace, "\tU 0x%x\tA 0x%x", (Arg)statbuf.used,
(Arg)statbuf.length);
#else
    LOG_printf(&trace, "seg %d: O 0x%x", segid, statbuf.size);
    LOG_printf(&trace, "\tU 0x%x\tA 0x%x", statbuf.used, statbuf.length);
#endif
}
```

TP5 QUE

Ce programme ce trouve dans le sous dossier :

C:\CCStudio_V3.1\examples\dsk6416\bios\quetest

```
/*
 *  Copyright 2003 by Texas Instruments Incorporated.
 *  All rights reserved. Property of Texas Instruments Incorporated.
 *  Restricted rights to use, duplicate or disclose this code are
 *  granted through contract.
 *
 */
/* "@(#) DSP/BIOS 4.90.270 01-05-04 (bios,dsk6416-f06)" */
/*
 *  ======== quetest.c ========
 *  Use a QUE queue to send messages from a writer() to a reader().
 *
 *  The queue is created by the Configuration Tool.
 *  For simplicity, we use MEM_alloc() and MEM_free() to manage
 *  the MsgObj structures.  It would be way more efficient to
 *  preallocate a pool of MsgObj's and keep them on a 'free'
 *  queue. Using the Config tool, create 'freeQueue'.  Then in
main(),
 *  allocate the MsgObj's with MEM_alloc() and add them to
'freeQueue'
 *  with QUE_put().
 *  You can then replace the MEM_alloc() calls with
 *  QUE_get(freeQueue) and MEM_free() with QUE_put(freeQueue, msg).
 *
 *  A queue can hold an arbitrary number of messages or elements.
 *  Each message must, however, be a structure with a QUE_Elem as
 *  its first field.
 */

#include <std.h>

#include <log.h>
#include <mem.h>
#include <que.h>
#include <sys.h>

#include "quetestcfg.h"

#define NUMMSGS     5      /* number of messages */

typedef struct MsgObj {
    QUE_Elem    elem;      /* first field for QUE */
    Char        val;       /* message value */
} MsgObj, *Msg;


Void reader();
Void writer();
```

```
/*
 *  ======== main ========
 */
Void main()
{
    /* The writer() must be called before reader() to ensure that
    the queue is non-empty for the reader. */
    writer();
    reader();
}

/*
 *  ======== reader ========
 */
Void reader()
{
    Msg         msg;
    Int         i;

    for (i=0; i < NUMMSGS; i++) {

        /* The queue should never be empty */
        if (QUE_empty(&queue)) {
            SYS_abort("queue error\n");
        }

        /* dequeue message */
        msg = QUE_get(&queue);

        /* print value */
#ifdef _28_
        LOG_printf(&trace, "read '%c'.", (Arg)msg->val);
#else
        LOG_printf(&trace, "read '%c'.", msg->val);
#endif

        /* free msg */
        MEM_free(0, msg, sizeof(MsgObj));
    }
}
```

```
/*
 *  ======== writer ========
 */
Void writer()
{
    Msg         msg;
    Int         i;

    for (i=0; i < NUMMSGS; i++) {
        /* allocate msg */
        msg = MEM_alloc(0, sizeof(MsgObj), 0);
        if (msg == MEM_ILLEGAL) {
            SYS_abort("Memory allocation failed!\n");
        }
        /* fill in value */
        msg->val = i + 'a';

#ifdef  _28_
        LOG_printf(&trace, "writing '%c' ...",(Arg) msg->val);
#else
        LOG_printf(&trace, "writing '%c' ...", msg->val);
#endif
        /* enqueue message */
        QUE_put(&queue, msg);
    }
}
```

TP 6 SEM

Ce programme ce trouve dans le sous dossier :

C:\CCStudio_V3.1\tutorial\dsk6416\semtest

```
/*
 *  Copyright 2003 by Texas Instruments Incorporated.
 *  All rights reserved. Property of Texas Instruments Incorporated.
 *  Restricted rights to use, duplicate or disclose this code are
 *  granted through contract.
 *
 */
/* "@(#) DSP/BIOS 4.90.270 01-05-04 (bios,dsk6416-f06)" */
/*
 *  ======== semtest.c ========
 *
 *  Use a QUE queue and SEM semaphore to send messages from
 *  multiple writer() tasks to a single reader() task.  The reader task,
 *  the three writer tasks, queues, and semaphore are created by the
 *  Configuration tool.
 *
 *  The MsgObj's are preallocated in main(), and put on the free queue.
 *  The writer tasks get free message structures from the free queue,
 *  write the message, and then put the message structure onto the message
 *  queue.
 *  This example builds on quetest.c.  The major differences are:
 *    - one reader() and multiple writer() tasks.
 *    - SEM_pend() and SEM_post() are used to synchronize access
 *      to the message queue.
 *    - 'id' field was added to MsgObj to specify writer() task id.
 *
 *  Unlike a mailbox, a queue can hold an arbitrary number of
 *  messages or elements.  Each message must, however, be a
 *  structure with a QUE_Elem as its first field.
 */
#include <std.h>

#include <log.h>
#include <mem.h>
#include <que.h>
#include <sem.h>
#include <sys.h>
#include <tsk.h>
#include <trc.h>

#include "semtestcfg.h"

#define NUMMSGS     3    /* number of messages */
#define NUMWRITERS  3    /* number of writer tasks created with Config Tool
*/

typedef struct MsgObj {
    QUE_Elem    elem;        /* first field for QUE */
    Int         id;          /* writer task id */
    Char        val;         /* message value */
} MsgObj, *Msg;

Void reader();
Void writer(Arg id_arg);
```

```
QUE_Obj msgQueue;
QUE_Obj freeQueue;

/*
 *  ======== main ========
 */
Void main()
{
    LOG_printf(&trace, "semtest example started.\n");
}
/*
 *  ======== initTask ========
 */
Void initTask()
{
    Int         i;
    MsgObj      *msg;
    Uns         mask;

    QUE_new(&msgQueue);
    QUE_new(&freeQueue);

    mask = TRC_LOGTSK | TRC_LOGSWI | TRC_STSSWI | TRC_LOGCLK;
    TRC_enable(TRC_GBLHOST | TRC_GBLTARG | mask);

    msg = (MsgObj *)MEM_alloc(0, NUMMSGS * sizeof(MsgObj), 0);
    if (msg == MEM_ILLEGAL) {
        SYS_abort("Memory allocation failed!\n");
    }

    /* Put all messages on freequeue */
    for (i = 0; i < NUMMSGS; msg++, i++) {
        QUE_put(&freeQueue, msg);
    }
}
/*
 *  ======== reader ========
 */
Void reader()
{
    Msg         msg;
    Int         i;

    for (i = 0; i < NUMMSGS * NUMWRITERS; i++) {
        /*
         * Wait for semaphore to be posted by writer().
         */
        SEM_pend(&sem, SYS_FOREVER);

        /* dequeue message */
        msg = QUE_get(&msgQueue);

        /* print value */
#ifdef _28_
        LOG_printf(&trace, "read '%c' from (%d).", (Arg)msg->val, (Arg)msg-
>id);
#else
        LOG_printf(&trace, "read '%c' from (%d).", msg->val, msg->id);
#endif
```

```
        /* free msg */
        QUE_put(&freeQueue, msg);
    }
    LOG_printf(&trace, "reader done.");
}

/*
 *  ======== writer ========
 */
Void writer(Arg id_arg)
{
    Msg         msg;
    Int         i;
    Int id =    ArgToInt (id_arg);

    for (i = 0; i < NUMMSGS; i++) {
        /*
         *  Get msg from the free queue.  Since reader is higher priority
         *  and only blocks on sem, this queue will never be empty.
         */
        if (QUE_empty(&freeQueue)) {
            SYS_abort("Empty free queue!\n");
        }
        msg = QUE_get(&freeQueue);

        /* fill in value */
        msg->id = id;
        msg->val = (i & 0xf) + 'a';
#ifdef _28_
        LOG_printf(&trace, "(%d) writing '%c' ...", (Arg)id, (Arg)msg-
>val);
#else
        LOG_printf(&trace, "(%d) writing '%c' ...", id, msg->val);
#endif

        /* enqueue message */
        QUE_put(&msgQueue, msg);

        /* post semaphore */
        SEM_post(&sem);

        /* what happens if you call TSK_yield() here? */
        /* TSK_yield(); */
    }
#ifdef _28_
    LOG_printf(&trace, "writer (%d) done.", (Arg)id);
#else
    LOG_printf(&trace, "writer (%d) done.", id);
#endif
}
```

TP7 MBX

Ce programme ce trouve dans le sous dossier :

C:\CCStudio_V3.1\tutorial\dsk6416\mbxtest

```
/*
 *  Copyright 2003 by Texas Instruments Incorporated.
 *  All rights reserved. Property of Texas Instruments Incorporated.
 *  Restricted rights to use, duplicate or disclose this code are
 *  granted through contract.
 *
 */
/* "@(#) DSP/BIOS 4.90.270 01-05-04 (bios,dsk6416-f06)" */
/*
 *  ======== mbxtest.c ========
 *  Use a MBX mailbox to send messages from multiple writer()
 *  tasks to a single reader() task.
 *  The mailbox, reader task, and three writer tasks are created by the
 *  Configuration Tool.
 *
 *  This example is similar to semtest.c.  The major differences
 *    are:
 *    - MBX is used in place of QUE and SEM.
 *    - the 'elem' field is removed from MsgObj.
 *    - reader() task is *not* higher priority than writer task.
 *    - reader() looks at return value from MBX_pend() for timeout
 */

#include <std.h>

#include <log.h>
#include <mbx.h>
#include <tsk.h>

#include "mbxtestcfg.h"


#define NUMMSGS         3       /* number of messages */

#define TIMEOUT         10

typedef struct MsgObj {
    Int         id;             /* writer task id */
    Char        val;            /* message value */
} MsgObj, *Msg;


Void reader(Void);
Void writer(Arg id_arg);

/*
 *  ======== main ========
 */
Void main()
{
    /* Does nothing */
}

/*
```

```
 *  ======== reader ========
 */
Void reader(Void)
{
    MsgObj      msg;
    Int         i;

    for (i=0; ;i++) {

        /* wait for mailbox to be posted by writer() */
        if (MBX_pend(&mbx, &msg, TIMEOUT) == 0) {
            LOG_printf(&trace, "timeout expired for MBX_pend()");
            break;
        }

        /* print value */
#ifdef _28_
        LOG_printf(&trace, "read '%c' from (%d).", (Arg)msg.val,
(Arg)msg.id);
#else
        LOG_printf(&trace, "read '%c' from (%d).", msg.val, msg.id);
#endif
    }
    LOG_printf(&trace, "reader done.");
}

/*
 *  ======== writer ========
 */
Void writer(Arg id_arg)
{
    MsgObj      msg;
    Int         i;
    Int id =    ArgToInt (id_arg);

    for (i=0; i < NUMMSGS; i++) {
        /* fill in value */
        msg.id = id;
        msg.val = i % NUMMSGS + (Int)('a');

        /* enqueue message */
        MBX_post(&mbx, &msg, TIMEOUT);

#ifdef _28_
        LOG_printf(&trace, "(%d) writing '%c' ...", (Arg)id, (Arg)msg.val);
#else
        LOG_printf(&trace, "(%d) writing '%c' ...", id, (Int)msg.val);
#endif

        /* what happens if you call TSK_yield() here? */
        /* TSK_yield(); */
    }
#ifdef _28_
    LOG_printf(&trace, "writer (%d) done.", (Arg)id);
#else
    LOG_printf(&trace, "writer (%d) done.", id);
#endif
}
```

TP8 SWI

Ce programme ce trouve dans le sous dossier :

C:\CCStudio_V3.1\examples\dsk6416\bios\switest

```
/*
 *  Copyright 2003 by Texas Instruments Incorporated.
 *  All rights reserved. Property of Texas Instruments Incorporated.
 *  Restricted rights to use, duplicate or disclose this code are
 *  granted through contract.
 *
 */
/* "@(#) DSP/BIOS 4.90.270 01-05-04 (bios,dsk6416-f06)" */
/*
 *  ======== switest.c =======
 *
 */

#include <std.h>

#include <log.h>
#include <swi.h>
#include <sys.h>

#include "switestcfg.h"

Void swiFxn0(Void);
Void swiFxn1(Void);

/*
 *  ======== main ========
 */
Void main(Int argc, Char *argv[])
{
    LOG_printf(&trace,"switest started!\n");
    LOG_printf(&trace,"Main posts SWI0\n");
    SWI_post(&SWI0);
        LOG_printf(&trace,"Main done!\n");
}

/*
 *  ======== swiFxn0 ========
 */
Void swiFxn0(Void)
{
    LOG_printf(&trace,"swiFxn0 posts SWI1\n");
        SWI_post(&SWI1);
        LOG_printf(&trace,"SWI0 done!\n");
}

/*
 *  ======== swiFxn1 ========
 */
Void swiFxn1(Void)
{
        LOG_printf(&trace,"SWI1 done!\n");
}
```

TP9 MUTEX

Ce programme ce trouve dans le sous dossier :

C:\CCStudio_V3.1\tutorial\dsk6416\mutex

```
/*
 *  Copyright 2003 by Texas Instruments Incorporated.
 *  All rights reserved. Property of Texas Instruments Incorporated.
 *  Restricted rights to use, duplicate or disclose this code are
 *  granted through contract.
 *
 */
/* "@(#) DSP/BIOS 4.90.270 01-05-04 (bios,dsk6416-f06)" */
/*
 *  ======== mutex.c ========
 */

#include <std.h>

#include <clk.h>
#include <log.h>
#include <tsk.h>
#include <sem.h>

#include "mutexcfg.h"

Void    mutex1();
Void    mutex2();

LgUns   tsk1count = 0;
LgUns   tsk2count = 0;
LgUns   maincount = 0;

/*
 *  ======== main ========
 */
Void main()
{
     /* Does nothing */
}
/*
 *  ======== mutex1 ========
 */
Void mutex1()
{
    LgUns tempvar;
    LgUns time;

    for (;;) {
        LOG_printf(&trace, "\n  Running mutex1 function");
        if (SEM_count(&sem) == 0) {
            LOG_printf(&trace, "Sem blocked in mutex1");
        }
        SEM_pend(&sem, SYS_FOREVER);
        tempvar = maincount;
        time = CLK_getltime();
        /* wait for two system ticks to pass */
        while (CLK_getltime() <= (time + 1)) {
             ;
        }
```

```
        tsk1count++;
        maincount = ++tempvar;
        /* combine hex values to display LgUns */
#ifdef _28_
        LOG_printf(&trace, "mutex1: loop 0x%04x%04x;", (Arg)(tsk1count >>
16), (Arg)(tsk1count & 0xffff));
        LOG_printf(&trace, "              total count = 0x%04x%04x", (Arg)
(maincount >> 16), (Arg)(maincount & 0xffff));
#else
        LOG_printf(&trace, "mutex1: loop 0x%04x%04x;", (Int)(tsk1count >>
16), (Int)(tsk1count & 0xffff));
        LOG_printf(&trace, "              total count = 0x%04x%04x", (Int)
(maincount >> 16), (Int)(maincount & 0xffff));
#endif

        SEM_post(&sem);
        TSK_sleep(1);
    }
}

/*
 *  ======== mutex2 ========
 */
Void mutex2()
{
    LgUns tempvar;

    for (;;) {
        LOG_printf(&trace, "\n  Running mutex2 function");
        if (SEM_count(&sem) == 0) {
            LOG_printf(&trace, "Sem blocked in mutex2");
        }
        SEM_pend(&sem, SYS_FOREVER);
        tempvar = maincount;
        tsk2count++;
        maincount = ++tempvar;
        /* combine hex values to display LgUns */
#ifdef _28_
        LOG_printf(&trace, "mutex2: loop 0x%04x%04x;", (Arg)(tsk2count >>
16), (Arg)(tsk2count & 0xffff));
        LOG_printf(&trace, "              total count = 0x%04x%04x", (Arg)
(maincount >> 16), (Arg)(maincount & 0xffff));
#else
        LOG_printf(&trace, "mutex2: loop 0x%04x%04x;", (Int)(tsk2count >>
16), (Int)(tsk2count & 0xffff));
        LOG_printf(&trace, "              total count = 0x%04x%04x", (Int)
(maincount >> 16), (Int)(maincount & 0xffff));
#endif
        SEM_post(&sem);
        TSK_sleep(1);
    }
}
```