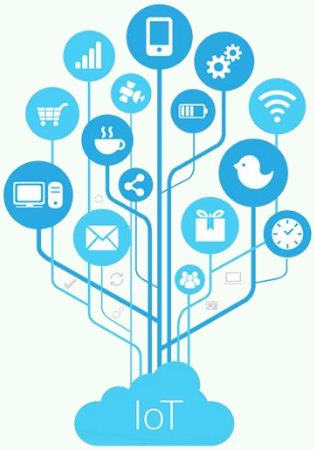
UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

TP n°1 : Solutions

MASTER RÉSEAUX ET TÉLÉCOM



VERSION : 10 juin 2021

Florent NOLOT
UNIVERSITÉ DE REIMS CHAMPAGNE ARDENNE

Table des matières

I. Solutions

A. Exercice 1 – Tâches et priorités

1. En utilisant les APIs TI-RTOS et TI Drivers, écrivez un programme possédant 2 tâches et une ISR. La tâche 1 doit avoir une priorité inférieure à la seconde tâche.
    - Quand la tâche 1 est en mode « Running », la LED verte doit être allumée et éteinte dans les autre modes.
    - Quand la tâche 2 est en mode « Running », la LED rouge doit être allumée et éteinte dans les autre modes.
    - L'ISR envoie un message sur le port série

Modification du fichier « main.c »

```c
/* Includes */
#include <xdc/runtime/System.h>         /* For System abort */
#include <ti/sysbios/knl/Task.h>        /* For Module task manager*/
#include <ti/sysbios/knl/Clock.h>       /* For Clock tick period */
#include <ti/drivers/GPIO.h>            /* For GPIO manager */
#include <ti/drivers/UART.h>            /* For UART manager */
#include <ti/sysbios/BIOS.h>            /* For RTOS header files */
#include <ti/drivers/Board.h>           /* For Example/Board Header files */
#include "board.h"                      /* For the board configuration */

/* Defines */
#define STACK_SIZE 1024                 /* Size of the stack */
#define LED_RED Board_GPIO_LED0         /* Led 0 */
#define LED_GREEN Board_GPIO_LED1       /* Led 1 */
#define LED_ON Board_GPIO_LED_ON        /* Switch Led on */
#define LED_OFF Board_GPIO_LED_OFF      /* Switch Led off */
#define BUTTON_0 Board_GPIO_BUTTON0    /* Button 0 */

/* Global */
UART_Handle uart;                                   /* UART Handle */
const char  UARTprompt[] = "Button pressed\r\n";    /* MSG interrupt */
```

```c
/* main */
int main(void){
    Task_Params taskParams;
    UART_Params uartParams;

    /* Initialization drivers */
    Board_init();
    GPIO_init();
    UART_init();

    /* Initialization LEDs */
    GPIO_setConfig(LED_RED, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
    GPIO_setConfig(LED_GREEN, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);

    /* Initialization Interrupt */
    GPIO_setConfig(BUTTON_0, GPIO_CFG_IN_PU | GPIO_CFG_IN_INT_FALLING);
    GPIO_setCallback(BUTTON_0, InterruptServiceRoutine);
    GPIO_enableInt(BUTTON_0);

    /* Initialization Task 1 */
    Task_Params_init(&taskParams);
    taskParams.stackSize = STACK_SIZE;
    taskParams.priority = 1;
    if( (Task_create((Task_FuncPtr) task1_fxn, &taskParams, NULL)) == NULL ){
        System_abort("Error creating task1\n");
    }

    /* Initialization Task 2 */
    Task_Params_init(&taskParams);
    taskParams.stackSize = STACK_SIZE;
    taskParams.priority = 2;
    if( (Task_create((Task_FuncPtr) task2_fxn, &taskParams, NULL)) == NULL ){
        System_abort("Error creating task2\n");
    }
```

Tâche 1 : Priorité 1

Tâche 2 : Priorité 2

```
    /* Initialization UART channel */
    UART_Params_init(&uartParams);
    uartParams.writeDataMode = UART_DATA_BINARY;
    uartParams.readDataMode = UART_DATA_BINARY;
    uartParams.readReturnMode = UART_RETURN_FULL;
    uartParams.readEcho = UART_ECHO_OFF;
    uartParams.baudRate = 115200;
    if( (uart = UART_open(Board_UART0, &uartParams)) == NULL ){
        System_abort("Error opening UART channel\n");
    }

    /* Start the TI-RTOS scheduler */
    BIOS_start();

    return (0);
}
```

```
/* Task 1 */
void task1_fxn(){
    while(1)
        GPIO_write(LED_GREEN, LED_ON);
}

/* Task 2 */
void task2_fxn(){
    int i;
    while(1){
        Task_sleep(3000000 / Clock_tickPeriod);
        GPIO_write(LED_GREEN, LED_OFF);
        GPIO_write(LED_RED, LED_ON);
        for(i=0; i < 2000000 ; i++);
        GPIO_write(LED_RED, LED_OFF);
    }
}

/* ISR */
void InterruptServiceRoutine(uint_least8_t index){
    if( (UART_write(uart, UARTprompt, sizeof(UARTprompt))) == UART_STATUS_ERROR ){
        System_abort("Error writing UART\n");
    }
}
```

La durée de mise en pause du Thread 2 est exprimée en nombre de « tick »
Dans cet exemple : pause de 3 secondes

Pour visualiser le changement d'état
Amélioration avec « clock »

2. Développez la même application avec les APIs POSIX

Modification du fichier « main.c »

```c
/* Includes */
#include <xdc/runtime/System.h>         /* For System abort */
#include <pthread.h>                    /* For POSIX Thread header files */
#include <ti/sysbios/knl/Clock.h>       /* For Clock tick period */
#include <ti/drivers/GPIO.h>            /* For GPIO manager */
#include <ti/drivers/UART.h>            /* For UART manager */
#include <ti/sysbios/BIOS.h>            /* For RTOS header files */
#include <ti/drivers/Board.h>           /* For Example/Board Header files */
#include <unistd.h>                     /* For sleep */
#include "board.h"                      /* For the board configuration */

/* Defines */
#define THREAD_STACK_SIZE 1024                  /* Size of the stack */
#define LED_RED Board_GPIO_LED0         /* Led 0 */
#define LED_GREEN Board_GPIO_LED1       /* Led 1 */
#define LED_ON Board_GPIO_LED_ON        /* Switch Led on */
#define LED_OFF Board_GPIO_LED_OFF      /* Switch Led off */
#define BUTTON_0 Board_GPIO_BUTTON0     /* Button 0 */

/* Global */
UART_Handle uart;                                       /* UART Handle */
const char  UARTprompt[] = "Button pressed\r\n";        /* MSG interrupt */
```

```c
/* main */
int main(void){
    UART_Params uartParams;
    pthread_t           thread;
    pthread_attr_t      attrs;
    struct sched_param  priParam;
    int                 retc;

    /* Initialization drivers */
    Board_init();
    GPIO_init();
    UART_init();

    /* Initialization Interrupt */
    GPIO_setConfig(BUTTON_0, GPIO_CFG_IN_PU | GPIO_CFG_IN_INT_FALLING);
    GPIO_setCallback(BUTTON_0, InterruptServiceRoutine);
    GPIO_enableInt(BUTTON_0);

    /* Initialization UART channel */
    UART_Params_init(&uartParams);
    uartParams.writeDataMode = UART_DATA_BINARY;
    uartParams.readDataMode = UART_DATA_BINARY;
    uartParams.readReturnMode = UART_RETURN_FULL;
    uartParams.readEcho = UART_ECHO_OFF;
    uartParams.baudRate = 115200;
    if( (uart = UART_open(Board_UART0, &uartParams)) == NULL ){
        System_abort("Error opening UART channel\n");
    }
```

```
    /* Initialize the attributes structure with default values */
    pthread_attr_init(&attrs);

    /* Thread 1 : Set priority, detach state, and stack size attributes */
    priParam.sched_priority = 1;
    retc = pthread_attr_setschedparam(&attrs, &priParam);
    retc |= pthread_attr_setdetachstate(&attrs, PTHREAD_CREATE_DETACHED);
    if( (retc |= pthread_attr_setstacksize(&attrs, THREAD_STACK_SIZE)) != 0 ){
        System_abort("Error setting stack size thread 1\n");
    }
    /* Thread 1 : creating thread */
    if( (retc = pthread_create(&thread, &attrs, thread1_fxn, NULL)) != 0 ){
        System_abort("Error creating thread 1\n");
    }

    /* Thread 2 : Set priority, detach state, and stack size attributes */
    priParam.sched_priority = 2;
    retc = pthread_attr_setschedparam(&attrs, &priParam);
    retc |= pthread_attr_setdetachstate(&attrs, PTHREAD_CREATE_DETACHED);
    if( (retc |= pthread_attr_setstacksize(&attrs, THREAD_STACK_SIZE)) != 0 ){
        System_abort("Error setting stack size thread 1\n");
    }
    /* Thread 2 : creating thread */
    if( (retc = pthread_create(&thread, &attrs, thread2_fxn, NULL)) != 0 ){
        System_abort("Error creating thread 2\n");
    }

    /* Start the TI-RTOS scheduler */
    BIOS_start();

    return (0);
}
```

```
/* Thread 1 */
void *thread1_fxn(void * arg0){
    while(1)
        GPIO_write(LED_GREEN, LED_ON);
}

/* Thread 2 */
void *thread2_fxn(void * arg0){
    int i;
    while(1){
        sleep(3);
        GPIO_write(LED_GREEN, LED_OFF);
        GPIO_write(LED_RED, LED_ON);
        for(i=0; i < 2000000 ; i++);
        GPIO_write(LED_RED, LED_OFF);
    }
}

/* ISR */
void InterruptServiceRoutine(uint_least8_t index){
    if( (UART_write(uart, UARTprompt, sizeof(UARTprompt))) == UART_STATUS_ERROR ){
        System_abort("Error writing UART\n");
    }
}
```

B. Exercice 2 – Sémaphores

En implémentant un sémaphore binaire modifiez un des programmes de l'exercice précédent (TI-RTOS ou TI-POSIX) afin de débloquer un thread (lors d'une pression du bouton 0) qui fait clignoter la LED rouge en continue.

Modification du fichier « main.c »

```c
/* Includes */
#include <xdc/runtime/System.h>             /* For System abort */
#include <ti/drivers/GPIO.h>                /* For GPIO header files */
#include <ti/drivers/Board.h>               /* For Example/Board Header files */
#include <ti/sysbios/knl/Task.h>            /* For Module task manager*/
#include <ti/sysbios/knl/Clock.h>           /* For Clock tick period */
#include <ti/sysbios/BIOS.h>                /* For RTOS header files */
#include <ti/sysbios/knl/semaphore.h>       /* For semaphore header files*/
#include "board.h"                          /* For the board configuration */

/* Defines */
#define STACK_SIZE 1024                     /* Size of the stack */
#define LED_RED Board_GPIO_LED0             /* Led 0 */
#define LED_ON Board_GPIO_LED_ON            /* Switch Led on */
#define LED_OFF Board_GPIO_LED_OFF          /* Switch Led off */
#define BUTTON_0 Board_GPIO_BUTTON0         /* Button 0 */

/* Global */
Semaphore_Handle S1_h;
```

```c
/* main */
int main(void){
    Task_Params taskParams;
    Semaphore_Params semParams;
    enum Semaphore_Mode semMode = Semaphore_Mode_BINARY;

    /* Initialization drivers */
    Board_init();
    GPIO_init();

    /* Initialization LEDs */
    GPIO_setConfig(LED_RED, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);

    /* Initialization Interrupt */
    GPIO_setConfig(BUTTON_0, GPIO_CFG_IN_PU | GPIO_CFG_IN_INT_FALLING);
    GPIO_setCallback(BUTTON_0, InterruptServiceRoutine);
    GPIO_enableInt(BUTTON_0);

    /* Initialization Semaphore */
    Semaphore_Params_init(&semParams);
    semParams.mode = semMode;
    if( (S1_h = Semaphore_create(0, &semParams, NULL)) == NULL ){
        System_abort("Error creating semaphore 1\n");
    }

    /* Initialization Task 1 */
    Task_Params_init(&taskParams);
    taskParams.stackSize = STACK_SIZE;
    taskParams.priority = 1;
    if( (Task_create((Task_FuncPtr) task1_fxn, &taskParams, NULL)) == NULL ){
        System_abort("Error creating task1\n");
    }

    /* Start the TI-RTOS scheduler */
    BIOS_start();

    return (0);
}
```

```c
/* Task 1 */
void task1_fxn(){
    Semaphore_pend(S1_h, BIOS_WAIT_FOREVER);
    while(1){
        GPIO_toggle(LED_RED);
        Task_sleep(100000 / Clock_tickPeriod);
    }
}

/* ISR */
void InterruptServiceRoutine(uint_least8_t index){
    Semaphore_post(S1_h);
}
```

C. Exercice 3 – Mailbox

Modifiez le programme de l'exercice précédent, afin d'obtenir une ISR et deux tâches.

- L'ISR doit incrémenter un sémaphore de 1 à chaque pression sur le bouton 0
- La tâche 1 décrémente le même sémaphore, lis l'état de la LED rouge et transmet la valeur à l'aide d'une Mailbox à la seconde tâche après un délai. (task_sleep(), ou boucle).
- La tâche 2 lis le message de reçu dans la Mailbox et positionne l'état de la LED verte au même état que celui de la LED rouge

Modification du fichier « main.c »

```c
/* Includes */
#include <xdc/runtime/System.h>              /* For System abort */
#include <ti/drivers/GPIO.h>                 /* For GPIO header files */
#include <ti/drivers/Board.h>                /* For Example/Board Header files */
#include <ti/sysbios/knl/Task.h>             /* For Module task manager*/
#include <ti/sysbios/knl/Clock.h>            /* For Clock tick period */
#include <ti/sysbios/BIOS.h>                 /* For RTOS header files */
#include <ti/sysbios/knl/semaphore.h>        /* For semaphore header files*/
#include <ti/sysbios/knl/Mailbox.h>          /* For mailbox header files*/
#include "board.h"                           /* For the board configuration */

/* Defines */
#define STACK_SIZE 1024                      /* Size of the stack */
#define LED_RED Board_GPIO_LED0              /* Led 0 */
#define LED_GREEN Board_GPIO_LED1            /* Led 1 */
#define LED_ON Board_GPIO_LED_ON             /* Switch Led on */
#define LED_OFF Board_GPIO_LED_OFF           /* Switch Led off */
#define BUTTON_0 Board_GPIO_BUTTON0          /* Button 0 */

/* Global */
uint_fast8_t state_LED_RED=0;
Semaphore_Handle S1_h;
Mailbox_Handle mailHandle;
```

```c
/* main */
int main(void){
    Task_Params taskParams;
    Semaphore_Params semParams;
    enum Semaphore_Mode semMode = Semaphore_Mode_BINARY;

    /* Initialization drivers */
    Board_init();
    GPIO_init();

    /* Initialization LEDs */
    GPIO_setConfig(LED_RED, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
    GPIO_setConfig(LED_GREEN, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);

    /* Initialization Interrupt */
    GPIO_setConfig(BUTTON_0, GPIO_CFG_IN_PU | GPIO_CFG_IN_INT_FALLING);
    GPIO_setCallback(BUTTON_0, InterruptServiceRoutine);
    GPIO_enableInt(BUTTON_0);

    /* Initialization Semaphore */
    Semaphore_Params_init(&semParams);
    semParams.mode = semMode;
    if( (S1_h = Semaphore_create(0, &semParams, NULL)) == NULL ){
        System_abort("Error creating semaphore 1\n");
    }
```

```c
    /* Initialization Mailbox */
    if( (mailHandle = Mailbox_create(sizeof(state_LED_RED), sizeof(state_LED_RED), NULL, NULL)) == NULL ){
        System_abort("Error creating Mailbox 1\n");
    }

    /* Initialization Task 1 */
    Task_Params_init(&taskParams);
    taskParams.stackSize = STACK_SIZE;
    taskParams.priority = 1;
    if( (Task_create((Task_FuncPtr) task1_fxn, &taskParams, NULL)) == NULL ){
        System_abort("Error creating task1\n");
    }

    /* Initialization Task 2 */
    Task_Params_init(&taskParams);
    taskParams.stackSize = STACK_SIZE;
    taskParams.priority = 1;
    if( (Task_create((Task_FuncPtr) task2_fxn, &taskParams, NULL)) == NULL ){
        System_abort("Error creating task2\n");
    }

    /* Start the TI-RTOS scheduler */
    BIOS_start();

    return (0);
}
```

```c
/* Task 1 */
void task1_fxn(){
    uint_fast8_t state_LED_RED=0;
    while(1){
        Semaphore_pend(S1_h, BIOS_WAIT_FOREVER);
        GPIO_toggle(LED_RED);
        state_LED_RED = GPIO_read(LED_RED);
        Task_sleep(1000000 / Clock_tickPeriod);
        if((Mailbox_post(mailHandle, &state_LED_RED, BIOS_NO_WAIT)) == FALSE ){
            System_abort("Error posting mailbox 1\n");
        }
    }
}

/* Task 2 */
void task2_fxn(){
    uint_fast8_t state_LED_RED=0;
    while(1){
        if((Mailbox_pend(mailHandle, &state_LED_RED, BIOS_WAIT_FOREVER)) == FALSE ){
            System_abort("Error posting mailbox 1\n");
        }
        GPIO_write(LED_GREEN,state_LED_RED);
    }
}


/* ISR */
void InterruptServiceRoutine(uint_least8_t index){
    Semaphore_post(S1_h);
}
```

D. Exercice 4 – Horloge

Modifiez le programme de l'exercice précédent et remplacez l'interruption (ISR) par une horloge.

Modification du fichier « main.c »

```c
/* Includes */
#include <xdc/runtime/System.h>             /* For System abort */
#include <ti/drivers/GPIO.h>                /* For GPIO header files */
#include <ti/drivers/Board.h>               /* For Example/Board Header files */
#include <ti/sysbios/knl/Task.h>            /* For Module task manager*/
#include <ti/sysbios/knl/Clock.h>           /* For clock header files */
#include <ti/sysbios/BIOS.h>                /* For RTOS header files */
#include <ti/sysbios/knl/semaphore.h>       /* For semaphore header files*/
#include <ti/sysbios/knl/Mailbox.h>         /* For mailbox header files*/
#include "board.h"                          /* For the board configuration */

/* Defines */
#define STACK_SIZE 1024                     /* Size of the stack */
#define LED_RED Board_GPIO_LED0             /* Led 0 */
#define LED_GREEN Board_GPIO_LED1           /* Led 1 */
#define LED_ON Board_GPIO_LED_ON            /* Switch Led on */
#define LED_OFF Board_GPIO_LED_OFF          /* Switch Led off */
#define BUTTON_0 Board_GPIO_BUTTON0         /* Button 0 */

/* Global */
uint_fast8_t state_LED_RED=0;
Semaphore_Handle S1_h;
Mailbox_Handle mailHandle;
Clock_Handle clockHandle;
```

```c
/* main */
int main(void){
    Task_Params taskParams;
    Semaphore_Params semParams;
    enum Semaphore_Mode semMode = Semaphore_Mode_BINARY;
    Clock_Params clockParams;
    UInt timeout = 10000000/Clock_tickPeriod;

    /* Initialization drivers */
    Board_init();
    GPIO_init();

    /* Initialization LEDs */
    GPIO_setConfig(LED_RED, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
    GPIO_setConfig(LED_GREEN, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);

    /* Initialization Semaphore */
    Semaphore_Params_init(&semParams);
    semParams.mode = semMode;
    if( (S1_h = Semaphore_create(0, &semParams, NULL)) == NULL ){
        System_abort("Error creating semaphore 1\n");
    }

    /* Initialization Mailbox */
    if( (mailHandle = Mailbox_create(sizeof(state_LED_RED), sizeof(state_LED_RED), NULL, NULL)) == NULL ){
        System_abort("Error creating Mailbox 1\n");
    }
```

```c
    /* Initialization Task 1 */
    Task_Params_init(&taskParams);
    taskParams.stackSize = STACK_SIZE;
    taskParams.priority = 1;
    if( (Task_create((Task_FuncPtr) task1_fxn, &taskParams, NULL)) == NULL ){
        System_abort("Error creating task1\n");
    }

    /* Initialization Task 2 */
    Task_Params_init(&taskParams);
    taskParams.stackSize = STACK_SIZE;
    taskParams.priority = 1;
    if( (Task_create((Task_FuncPtr) task2_fxn, &taskParams, NULL)) == NULL ){
        System_abort("Error creating task2\n");
    }

    /* Initialization Clock */
    Clock_Params_init(&clockParams);
    clockParams.period = 1000000/Clock_tickPeriod;
    if( (clockHandle = Clock_create((Clock_FuncPtr) clock_fxn, timeout, &clockParams, NULL)) == NULL ){
        System_abort("Error creating clock\n");
    }
    Clock_start(clockHandle);

    /* Start the TI-RTOS scheduler */
    BIOS_start();

    return (0);
}
```

```c
/* Task 1 */
void task1_fxn(){
    uint_fast8_t state_LED_RED=0;
    while(1){
        Semaphore_pend(S1_h, BIOS_WAIT_FOREVER);
        GPIO_toggle(LED_RED);
        state_LED_RED = GPIO_read(LED_RED);
        Task_sleep(100000/Clock_tickPeriod);
        if((Mailbox_post(mailHandle, &state_LED_RED, BIOS_NO_WAIT)) == FALSE ){
            System_abort("Error posting mailbox \n");
        }
    }
}

/* Task 2 */
void task2_fxn(){
    uint_fast8_t state_LED_RED=0;
    while(1){
        if((Mailbox_pend(mailHandle, &state_LED_RED, BIOS_WAIT_FOREVER)) == FALSE ){
            System_abort("Error pending mailbox\n");
        }
        GPIO_write(LED_GREEN,state_LED_RED);
    }
}

/* Clock function */
void clock_fxn(){
    if(Semaphore_getCount(S1_h) == 0)
        Semaphore_post(S1_h);
}
```