Emanuel Donie Ypon
CMPE 146 Sec 03
12/1/2020

Lab 8 Report

Exercise 1

Task functions:

Task1()

```
Void task1(UArg arg0, UArg arg1)
{
    printf("Task1\n");

    // control timing
    while (1) {
        // sleep for 0.4 Hz = 2500 ms
        // 1250 ms on/off
        Task_sleep(1250);

        // increment semaphore
        Semaphore_post(semaphoreHandle);
    }
}
```

Task2()

```
Void task2(UArg arg0, UArg arg1)
{
    printf("Task2\n");

    // setup LED
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);

    // control ON/OFF
    while (1) {
        // try to get semaphore
        Semaphore_pend(semaphoreHandle, BIOS_WAIT_FOREVER);

        // on or off
        MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);
    }
}
```

To control the LED, we split the blinking into two different tasks. Task 1 determines how long the LED stays on or off, in this case, we have a frequency of 0.4 Hz. To find the respective time in ms, I used the formula T = 1 / F. Since we want a frequency of 0.4 Hz, this will result in a time of 2500 ms with 1250 ms on and off. Task 2 toggles the LED on or off. The program executes Task 1 first since it

is created first in main, and will then enter Task 2. Task 1 calls the function `Semaphore_post()` which will increment the semaphore's count and release it. Task 2 calls the function `Semaphore_pend()` which will wait for the semaphore to be available. Once it is available, the task will change the LEDs state by toggling it on or off. It then calls the function again to try and get the semaphore again.

The entire program is in the appendix.

Exercise 2

Creating the tasks:

```
    // Construct tasks
    Task_Params taskParams1, taskParams2, taskParams3;

    Task_Params_init(&taskParams1);
    Task_Params_init(&taskParams2);
    Task_Params_init(&taskParams3);

    taskParams1.stackSize = TASKSTACKSIZE;
    taskParams2.stackSize = TASKSTACKSIZE;
    taskParams3.stackSize = TASKSTACKSIZE;

    // create 3 tasks
    // task 1
    taskParams1.arg0 = 1;
    taskParams1.arg1 = 1;
    taskParams1.stack = &taskStacks[0];
    Task_construct(&taskStructs[0], (Task_FuncPtr)taski, &taskParams1, NULL);

    // task 2
    taskParams2.arg0 = 2;
    taskParams2.arg1 = 2;
    taskParams2.stack = &taskStacks[1];
    Task_construct(&taskStructs[1], (Task_FuncPtr)taski, &taskParams2, NULL);

    // task 3
    taskParams3.arg0 = 3;
    taskParams3.arg1 = 3;
    taskParams3.stack = &taskStacks[2];
    Task_construct(&taskStructs[2], (Task_FuncPtr)taski, &taskParams3,
NULL)
```

To distinguish which tasks we are running, I changed the arguments of the parameters respectively (e.g., Task 1 arg0 = 1, arg1 = 1; Task 2 arg0 = 2, arg1 = 2; Task 3 arg0 = 3, arg1 = 3).

Screenshot of console outputs:

Lab8_Ex2:CIO

```
[CORTEX_M4_0] Task: arg0=1, arg1=1
Task: arg0=2, arg1=2
Task: arg0=3, arg1=3
Task: arg0=1, arg1=1
Task: arg0=2, arg1=2
Task: arg0=3, arg1=3
Task: arg0=1, arg1=1
Task: arg0=2, arg1=2
Task: arg0=3, arg1=3
```

The entire program is in the appendix.

Exercise 3

Timing task function:

```
Void task_timing(UArg arg0, UArg arg1) {
    // control timing
    while (1) {
        // random time freq between 0.2 Hz to 1 Hz
        // **between 5000 ms (0.2 Hz) to 1000 ms (1 Hz)
        int ms = (rand() % (5000 + 1 - 1000) + 1000) / 2;

        uint32_t i;
        for (i = 0; i < NTASKS - 1; i++) {
            Semaphore_post(semaphoreHandle);
        }

        Task_sleep(ms);
    }
}
```

Common LED control task function:

```
Void task_LEDcontrol(UArg arg0, UArg arg1)
{
    // setup LED
    MAP_GPIO_setAsOutputPin(arg0, arg1);
    MAP_GPIO_setOutputLowOnPin(arg0, arg1);

    while(1)
    {
        Semaphore_pend(semaphoreHandle, BIOS_WAIT_FOREVER);

        // on or off
        MAP_GPIO_toggleOutputOnPin(arg0, arg1);

        // suspend itself
        Task_yield();
    }
}
```

To make all three LEDs blink syncrhonously, after getting the random frequency time I then call `Semaphore_post()` to make the semaphore available for the control tasks to retrieve. One thing to note, we post the semaphore **three** times since only the three control tasks require the semaphore. In the control task, I first call `Semaphore_pend()` to try and retrieve the semaphore if it is available. Once acquired, I toggle the respective LED passed as an argument and call `Task_yield()` which will allow the other LEd control tasks to retrieve the semaphore. This will then enable all LEDs to be toggled synchronously.

The entire program is in the appendix.

Appendix

Exercise 1

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

/* XDC module Headers */
#include <xdc/std.h>
#include <xdc/runtime/System.h>

/* BIOS module Headers */
#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Clock.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Semaphore.h>

#include <ti/drivers/Board.h>
#define __MSP432P4XX__
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>

#define TASKSTACKSIZE   2048

Void task1(UArg arg0, UArg arg1);
Void task2(UArg arg0, UArg arg1);

Task_Struct task1Struct, task2Struct;
Char task1Stack[TASKSTACKSIZE], task2Stack[TASKSTACKSIZE];

Semaphore_Struct semaStruct;
Semaphore_Handle semaphoreHandle = NULL;

int main()
{
    // Init drivers
```

```
    Board_init();

    srand(time(NULL));      // Set seed for random number generator

    // Construct tasks
    Task_Params taskParams;
    Task_Params_init(&taskParams);
    taskParams.stackSize = TASKSTACKSIZE;

    // Task 1
    taskParams.stack = &task1Stack;
    Task_construct(&task1Struct, (Task_FuncPtr)task1, &taskParams, NULL);

    // Task 2
    taskParams.stack = &task2Stack;
    Task_construct(&task2Struct, (Task_FuncPtr)task2, &taskParams, NULL);

    // Construct a semaphore object
    Semaphore_Params semaParams;
    Semaphore_Params_init(&semaParams);                 // Initialize
structure with default parameters
    Semaphore_construct(&semaStruct, 0, &semaParams);   // Create an instance
of semaphore object
    semaphoreHandle = Semaphore_handle(&semaStruct);

    BIOS_start();    // Jump to the OS and won't return
    return(0);
}

Void task1(UArg arg0, UArg arg1)
{
    printf("Task1\n");

    // control timing
    while (1) {
        // sleep for 0.4 Hz = 2500 ms
        // 1250 ms on/off
        Task_sleep(1250);

        // increment semaphore
        Semaphore_post(semaphoreHandle);
    }
}

Void task2(UArg arg0, UArg arg1)
{
    printf("Task2\n");

    // setup LED
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);

    // control ON/OFF
    while (1) {
        // try to get semaphore
        Semaphore_pend(semaphoreHandle, BIOS_WAIT_FOREVER);
```

```
        // on or off
        MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);
    }
}
```

Exercise 2

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

/* XDC module Headers */
#include <xdc/std.h>
#include <xdc/runtime/System.h>

/* BIOS module Headers */
#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Clock.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Semaphore.h>

#include <ti/drivers/Board.h>
#define __MSP432P4XX__
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>

#define TASKSTACKSIZE   2048

//Void task1(UArg arg0, UArg arg1);
//Void task2(UArg arg0, UArg arg1);
Void taski(UArg arg0, UArg arg1);

//Task_Struct task1Struct, task2Struct;
//Char task1Stack[TASKSTACKSIZE], task2Stack[TASKSTACKSIZE];

// Ex 2
#define NTASKS 3
Task_Struct taskStructs[NTASKS];
Char taskStacks[NTASKS][TASKSTACKSIZE];

Semaphore_Struct semaStruct;
Semaphore_Handle semaphoreHandle = NULL;

int main()
{
    // Init drivers
    Board_init();

    srand(time(NULL));      // Set seed for random number generator

    // Construct tasks
    Task_Params taskParams1, taskParams2, taskParams3;
```

```
    Task_Params_init(&taskParams1);
    Task_Params_init(&taskParams2);
    Task_Params_init(&taskParams3);

    taskParams1.stackSize = TASKSTACKSIZE;
    taskParams2.stackSize = TASKSTACKSIZE;
    taskParams3.stackSize = TASKSTACKSIZE;

    // create 3 tasks
    // task 1
    taskParams1.arg0 = 1;
    taskParams1.arg1 = 1;
    taskParams1.stack = &taskStacks[0];
    Task_construct(&taskStructs[0], (Task_FuncPtr)taski, &taskParams1, NULL);

    // task 2
    taskParams2.arg0 = 2;
    taskParams2.arg1 = 2;
    taskParams2.stack = &taskStacks[1];
    Task_construct(&taskStructs[1], (Task_FuncPtr)taski, &taskParams2, NULL);

    // task 3
    taskParams3.arg0 = 3;
    taskParams3.arg1 = 3;
    taskParams3.stack = &taskStacks[2];
    Task_construct(&taskStructs[2], (Task_FuncPtr)taski, &taskParams3, NULL);

    // Construct a semaphore object
    Semaphore_Params semaParams;
    Semaphore_Params_init(&semaParams);                 // Initialize
structure with default parameters
    Semaphore_construct(&semaStruct, 0, &semaParams);   // Create an instance
of semaphore object
    semaphoreHandle = Semaphore_handle(&semaStruct);

    BIOS_start();    // Jump to the OS and won't return
    return(0);
}

Void taski(UArg arg0, UArg arg1)
{
    while(1)
    {
        printf("Task: arg0=%u, arg1=%u\n", (uint32_t)arg0, (uint32_t)arg1);
        Task_sleep(1000);
    }
}
```

Exercise 3

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
```

```
/* XDC module Headers */
#include <xdc/std.h>
#include <xdc/runtime/System.h>

/* BIOS module Headers */
#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Clock.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Semaphore.h>

#include <ti/drivers/Board.h>
#define __MSP432P4XX__
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>

#define TASKSTACKSIZE   2048

Void task_timing(UArg arg0, UArg arg1);
Void task_LEDcontrol(UArg arg0, UArg arg1);

// Ex 2
#define NTASKS 4
Task_Struct taskStructs[NTASKS];
Char taskStacks[NTASKS][TASKSTACKSIZE];

Semaphore_Struct semaStruct;
Semaphore_Handle semaphoreHandle = NULL;

int main()
{
    // Init drivers
    Board_init();

    srand(time(NULL));      // Set seed for random number generator

    // Construct tasks
    Task_Params timingParams, redParams, greenParams, blueParams;

    Task_Params_init(&timingParams);
    Task_Params_init(&redParams);
    Task_Params_init(&greenParams);
    Task_Params_init(&blueParams);

    timingParams.stackSize = TASKSTACKSIZE;
    redParams.stackSize = TASKSTACKSIZE;
    greenParams.stackSize = TASKSTACKSIZE;
    blueParams.stackSize = TASKSTACKSIZE;

    // task 1 - blinking task
    timingParams.stack = &taskStacks[0];
    Task_construct(&taskStructs[0], (Task_FuncPtr)task_timing, &timingParams,
NULL);

    // task 2 - Red LED
    redParams.arg0 = GPIO_PORT_P1;
    redParams.arg1 = GPIO_PIN0;
```

```
    redParams.stack = &taskStacks[1];
    Task_construct(&taskStructs[1], (Task_FuncPtr)task_LEDcontrol, &redParams,
NULL);

    // task 3 - Green LED
    greenParams.arg0 = GPIO_PORT_P2;
    greenParams.arg1 = GPIO_PIN1;
    greenParams.stack = &taskStacks[2];
    Task_construct(&taskStructs[2], (Task_FuncPtr)task_LEDcontrol,
&greenParams, NULL);

    // task 4 - Blue LED
    blueParams.arg0 = GPIO_PORT_P2;
    blueParams.arg1 = GPIO_PIN2;
    blueParams.stack = &taskStacks[3];
    Task_construct(&taskStructs[3], (Task_FuncPtr)task_LEDcontrol,
&blueParams, NULL);

    // Construct a semaphore object
    Semaphore_Params semaParams;
    Semaphore_Params_init(&semaParams);                 // Initialize
structure with default parameters
    Semaphore_construct(&semaStruct, 0, &semaParams);   // Create an instance
of semaphore object
    semaphoreHandle = Semaphore_handle(&semaStruct);

    BIOS_start();    // Jump to the OS and won't return
    return(0);
}

Void task_timing(UArg arg0, UArg arg1) {
    // control timing
    while (1) {
        // random time freq between 0.2 Hz to 1 Hz
        // **between 5000 ms (0.2 Hz) to 1000 ms (1 Hz)
        int ms = (rand() % (5000 + 1 - 1000) + 1000) / 2;

        uint32_t i;
        for (i = 0; i < NTASKS - 1; i++) {
            Semaphore_post(semaphoreHandle);
        }

        Task_sleep(ms);
    }
}

Void task_LEDcontrol(UArg arg0, UArg arg1)
{
    // setup LED
    MAP_GPIO_setAsOutputPin(arg0, arg1);
    MAP_GPIO_setOutputLowOnPin(arg0, arg1);

    while(1)
    {
        Semaphore_pend(semaphoreHandle, BIOS_WAIT_FOREVER);
```

```
        // on or off
        MAP_GPIO_toggleOutputOnPin(arg0, arg1);

        // suspend itself
        Task_yield();
    }
}
```