**Angelica Semenec**

**012179180**

**Lab 8**

**TI-RTOS and semaphores on MSP432 MCU**

**Purpose**

The purpose of this lab is to become familiarized with the TI real-time operating system as well as the semaphore system object to control LEDs.

**Exercise 1**

|  |
| --- |
| **hello.c** |
| .  .  .  Void task1(UArg arg0, UArg arg1)  {  while (1)  {  Task\_sleep(1000);  Semaphore\_post(semaphoreHandle);  }  }  Void task2(UArg arg0, UArg arg1)  {  MAP\_GPIO\_setAsOutputPin(GPIO\_PORT\_P1, GPIO\_PIN0);  while (1)  {  Semaphore\_pend(semaphoreHandle, BIOS\_WAIT\_FOREVER);  MAP\_GPIO\_toggleOutputOnPin(GPIO\_PORT\_P1, GPIO\_PIN0);  }  } |

Upon entering the infinite loop in *task1*, the *Task\_sleep()* function is used to sleep the task for 1000ms (1s). Upon sleeping, *task2* will set up the GPIO port to control the LED before entering an infinite loop. The function checks the availability of the resource. When it detects that the resource is not available, the task will be suspended and wait until it is. After the delay in *task1* is complete, the *task1* resumes and the *Semaphore\_post()* function increments the semaphore count by 1, making the resource available. The task then loops back to the *Task\_sleep()* function, passing control to *task2*, which grabs the resource and decrements the semaphore count back to 0 through the *Semaphore\_pend()* function. The task toggles the red LED and then tries to access the resource again. It detects that the resource is unavailable (as it decremented back to 0) and then waits for it to become available again. Eventually, *task1* awakes from its sleep again and increments the semaphore count, making it available for the task to use. This cycle repeats on an infinite loop with the 1000ms delay causing the red LED to toggle at a frequency of 1Hz.

**Exercise 2**

|  |
| --- |
| **hello.c** |
| .  .  .  Void task1(UArg arg0, UArg arg1)  {  int ts;  while (1)  {  //Range 1000 - 5000ms --> [ 0 : 4000 ] + 1000  Task\_sleep((rand() % 4001) + 1000);  Semaphore\_post(semaphoreHandle);  Semaphore\_post(semaphoreHandle);  }  }  Void task2(UArg arg0, UArg arg1)  {  MAP\_GPIO\_setAsOutputPin(GPIO\_PORT\_P1, GPIO\_PIN0);  MAP\_GPIO\_setOutputHighOnPin(GPIO\_PORT\_P1, GPIO\_PIN0);  while (1)  {  Semaphore\_pend(semaphoreHandle, BIOS\_WAIT\_FOREVER);  MAP\_GPIO\_toggleOutputOnPin(GPIO\_PORT\_P1, GPIO\_PIN0);  }  }  Void task3(UArg arg0, UArg arg1)  {  MAP\_GPIO\_setAsOutputPin(GPIO\_PORT\_P2, GPIO\_PIN1);  MAP\_GPIO\_setOutputHighOnPin(GPIO\_PORT\_P2, GPIO\_PIN1);  while (1)  {  Semaphore\_pend(semaphoreHandle, BIOS\_WAIT\_FOREVER);  MAP\_GPIO\_toggleOutputOnPin(GPIO\_PORT\_P2, GPIO\_PIN1);  }  } |

*Task1* uses the modulus function ( *%* ) to limit the range of the random number generated to between 0 and 4000. By adding 1000 to this, the limit for the timing (in ms for the *Task\_sleep()* function) is changed to 1000 to 5000. This corresponds to 1 cycle every 1000 to 5000 ms or 1 Hz to 0.2 Hz. *Task1* then sleeps and the next two task functions can begin processing. Each task function initializes its LED as high so that each LED begins in the same state. *Task2* first sets up then checks the availability of the resource. The resource is unavailable, so it waits and passes control. *Task3* then sets up its LED and checks the availability of the resource. The resource is also unavailable, so it also waits and passes control. After the randomly generated delay, *task1* wakes up and then increments the semaphore twice by calling the *\_post()* function twice. This allows for 2 accesses of the resource. *Task1* then sleeps again for a random amount of time. *Task2* and *task3* grab the resource and toggle their LEDs.

**Exercise 3**

|  |
| --- |
| **hello.c** |
| Void task1(UArg arg0, UArg arg1)  {  while (1)  {  //Range 1000 - 5000ms --> [ 0 : 4000 ] + 1000  Task\_sleep((rand() % 4001) + 1000);  Semaphore\_post(semaphoreHandle);  Semaphore\_post(semaphoreHandle);  Semaphore\_post(semaphoreHandle);  }  }  Void task2(UArg arg0, UArg arg1)  {  MAP\_GPIO\_setAsOutputPin(GPIO\_PORT\_P1, GPIO\_PIN0);  MAP\_GPIO\_setOutputHighOnPin(GPIO\_PORT\_P1, GPIO\_PIN0);  while (1)  {  Semaphore\_pend(semaphoreHandle, BIOS\_WAIT\_FOREVER);  MAP\_GPIO\_toggleOutputOnPin(GPIO\_PORT\_P1, GPIO\_PIN0);  Task\_yield();  }  }  Void task3(UArg arg0, UArg arg1)  {  MAP\_GPIO\_setAsOutputPin(GPIO\_PORT\_P2, GPIO\_PIN1);  MAP\_GPIO\_setOutputHighOnPin(GPIO\_PORT\_P2, GPIO\_PIN1);  while (1)  {  Semaphore\_pend(semaphoreHandle, BIOS\_WAIT\_FOREVER);  MAP\_GPIO\_toggleOutputOnPin(GPIO\_PORT\_P2, GPIO\_PIN1);  Task\_yield();  }  }  Void task4(UArg arg0, UArg arg1)  {  MAP\_GPIO\_setAsOutputPin(GPIO\_PORT\_P2, GPIO\_PIN2);  MAP\_GPIO\_setOutputHighOnPin(GPIO\_PORT\_P2, GPIO\_PIN2);  while (1)  {  Semaphore\_pend(semaphoreHandle, BIOS\_WAIT\_FOREVER);  MAP\_GPIO\_toggleOutputOnPin(GPIO\_PORT\_P2, GPIO\_PIN2);  Task\_yield();  }  } |

**Appendix**

**Exercise 1**

|  |
| --- |
| **hello.c** |
| #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)  {  while (1)  {  Task\_sleep(1000);  Semaphore\_post(semaphoreHandle);  }  }  Void task2(UArg arg0, UArg arg1)  {  MAP\_GPIO\_setAsOutputPin(GPIO\_PORT\_P1, GPIO\_PIN0);  while (1)  {  Semaphore\_pend(semaphoreHandle, BIOS\_WAIT\_FOREVER);  MAP\_GPIO\_toggleOutputOnPin(GPIO\_PORT\_P1, GPIO\_PIN0);  }  } |

**Exercise 2**

|  |
| --- |
| **hello.c** |
| #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); //red LED  Void task3(UArg arg0, UArg arg1); //green LED  Void task4(UArg arg0, UArg arg1); //blue LED  Task\_Struct task1Struct, task2Struct, task3Struct, task4Struct;  Char task1Stack[TASKSTACKSIZE], task2Stack[TASKSTACKSIZE], task3Stack[TASKSTACKSIZE], task4Stack[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);  // Task 3  taskParams.stack = &task3Stack;  Task\_construct(&task3Struct, (Task\_FuncPtr)task3, &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)  {  while (1)  {  //Range 1000 - 5000ms --> [ 0 : 4000 ] + 1000  Task\_sleep((rand() % 4001) + 1000);  Semaphore\_post(semaphoreHandle);  Semaphore\_post(semaphoreHandle);  }  }  Void task2(UArg arg0, UArg arg1)  {  MAP\_GPIO\_setAsOutputPin(GPIO\_PORT\_P1, GPIO\_PIN0);  MAP\_GPIO\_setOutputHighOnPin(GPIO\_PORT\_P1, GPIO\_PIN0);  while (1)  {  Semaphore\_pend(semaphoreHandle, BIOS\_WAIT\_FOREVER);  MAP\_GPIO\_toggleOutputOnPin(GPIO\_PORT\_P1, GPIO\_PIN0);  }  }  Void task3(UArg arg0, UArg arg1)  {  MAP\_GPIO\_setAsOutputPin(GPIO\_PORT\_P2, GPIO\_PIN1);  MAP\_GPIO\_setOutputHighOnPin(GPIO\_PORT\_P2, GPIO\_PIN1);  while (1)  {  Semaphore\_pend(semaphoreHandle, BIOS\_WAIT\_FOREVER);  MAP\_GPIO\_toggleOutputOnPin(GPIO\_PORT\_P2, GPIO\_PIN1);  }  } |

**Exercise 3**

|  |
| --- |
| **hello.c** |
| #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); //red LED  Void task3(UArg arg0, UArg arg1); //green LED  Void task4(UArg arg0, UArg arg1); //blue LED  Task\_Struct task1Struct, task2Struct, task3Struct, task4Struct;  Char task1Stack[TASKSTACKSIZE], task2Stack[TASKSTACKSIZE], task3Stack[TASKSTACKSIZE], task4Stack[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);  // Task 3  taskParams.stack = &task3Stack;  Task\_construct(&task3Struct, (Task\_FuncPtr)task3, &taskParams, NULL);  // Task 4  taskParams.stack = &task4Stack;  Task\_construct(&task4Struct, (Task\_FuncPtr)task4, &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)  {  while (1)  {  //Range 1000 - 5000ms --> [ 0 : 4000 ] + 1000  Task\_sleep((rand() % 4001) + 1000);  Semaphore\_post(semaphoreHandle);  Semaphore\_post(semaphoreHandle);  Semaphore\_post(semaphoreHandle);  }  }  Void task2(UArg arg0, UArg arg1)  {  MAP\_GPIO\_setAsOutputPin(GPIO\_PORT\_P1, GPIO\_PIN0);  MAP\_GPIO\_setOutputHighOnPin(GPIO\_PORT\_P1, GPIO\_PIN0);  while (1)  {  Semaphore\_pend(semaphoreHandle, BIOS\_WAIT\_FOREVER);  MAP\_GPIO\_toggleOutputOnPin(GPIO\_PORT\_P1, GPIO\_PIN0);  Task\_yield();  }  }  Void task3(UArg arg0, UArg arg1)  {  MAP\_GPIO\_setAsOutputPin(GPIO\_PORT\_P2, GPIO\_PIN1);  MAP\_GPIO\_setOutputHighOnPin(GPIO\_PORT\_P2, GPIO\_PIN1);  while (1)  {  Semaphore\_pend(semaphoreHandle, BIOS\_WAIT\_FOREVER);  MAP\_GPIO\_toggleOutputOnPin(GPIO\_PORT\_P2, GPIO\_PIN1);  Task\_yield();  }  }  Void task4(UArg arg0, UArg arg1)  {  MAP\_GPIO\_setAsOutputPin(GPIO\_PORT\_P2, GPIO\_PIN2);  MAP\_GPIO\_setOutputHighOnPin(GPIO\_PORT\_P2, GPIO\_PIN2);  while (1)  {  Semaphore\_pend(semaphoreHandle, BIOS\_WAIT\_FOREVER);  MAP\_GPIO\_toggleOutputOnPin(GPIO\_PORT\_P2, GPIO\_PIN2);  Task\_yield();  }  } |