**CG2271 Real Time Operating Systems**

**Lab 4**

**Using an RTOS**

|  |  |
| --- | --- |
| Name: Vu Ngoc Quang | Matric No: A0099878W |
| Name: Prachi Singhal | Matric No: A0103527N |

**Question 1 (3 marks):**

OSInit function will initialise the ArdOS. OSCreateTask function registers a new task. OSRun function executes the task.

**Question 2 (3 marks):**

The parameter for OSInit function is the amount of processes that you plan to implement. IT is of type unsigned character of which range is 0-255.

The parameters for the function OSCreate are the priority, the function pointer to the task and the argument to be passed to the task function. The type for the priority is unsigned character, function pointer, and a void pointer for the argument.

**Question 3 (3 marks):**

The empty loop is to do nothing. A basic characteristic of the microprocessor is that it always has to do something. However, our requirement in this situation is to not do anything. Hence, to ensure that the microprocessor continuously receives something to do, we put an empty loop.

**Question 4 (3 marks):**

RTOS systems usually have a lot of sensors which continuously provide information to calculate and distribute to the actuators. Hence, it has to receive input from the sensors continuously. Therefore, we implement the tasks in an infinite loop.

**Question 5 (5 marks):**

OSSleep puts tasks to the ready state, whereby the tasks execution is paused for a set period of time. This period is determined by the user, by means of the parameter. The parameter is of type unsigned long which ranges from 0 to (2^64)-1. This is because the time is calculated in milliseconds and hence we need a type which can hold large values. After the set period of time is passed, the OS will check the priority of the current task and only run the paused task if it has the highest priority.

This function pauses the task in whichever state it was being implemented. For example,

digitalWrite(6, HIGH);

OSSleep(100);

After the task writes a HIGH value to pin6, then the OSSleep will pause the task for 100ms and the task will stay at the HIGH value state for 100ms (i.e. the LED is on for 100ms).

**Question 6 (4 marks):**

The LED attached to pin6 flashes for five times first with frequency 2Hz, followed by the LED attached to pin7, which will flash for two times with frequency 1Hz. This sequence will repeat itself.

OSTakeSema function takes in a semaphore. If the semaphore value is equal to zero, the task will not be executed. However, if the semaphore value is equal to one, the task will be executed.

OSGiveSema function gives a semaphore. It unblocks the highest priority task that is waiting on that same semaphore. If there is no task waiting for that semaphore, the value of the semaphore is changed to one (assuming the semaphore’s value is binary).

For example, the semaphore ‘task1Go’ is initialised to be one. Then, task1 will take this semaphore to be executed. Then task 1 gives the semaphore ‘task2Go’ after execution. This is because the task2 is waiting on task2Go semaphore, now task2 is being executed. After task2 is completed, it will give the semaphore task1Go, which will execute task1 again.

Hence, each task will be executed in sequence and no two tasks will be implemented at the same time.

**Question 7 (10 marks):**

#include <avr/io.h>

#include <Arduino.h>

#include "kernel.h"

#include "sema.h"

const int INT0\_PRIORITY = 0; //priority of INT0

const int INT1\_PRIORITY = 1; //priority of INT1

OSSema task1Go, task2Go;

// Do switch debouncing

unsigned long int0time=0, int1time=0;

// Debouncing function. Returns TRUE if this interrupt was not caused by a bouncing switch

int debounce(unsigned long \*debTimer)

{

unsigned long tmp=\*debTimer;

unsigned long currTime=OSticks();

if((currTime-tmp) > 500)

{

\*debTimer=currTime;

return 1;

}

else

return 0;

}

void task1(void \*param)

{

while(1)

{

//waiting for semaphore from int0ISR

OSTakeSema(&task1Go);

for(int i=0; i<5; i++)

{

digitalWrite(6, HIGH);

OSSleep(250);

digitalWrite(6, LOW);

OSSleep(250);

}

}

}

void task2(void \*param)

{

while(1)

{

// Wait for semaphore from int1ISR

OSTakeSema(&task2Go);

for(int i=0; i<5; i++)

{

digitalWrite(7, HIGH);

OSSleep(250);

digitalWrite(7, LOW);

OSSleep(250);

}

}

}

//ISR for task 1

void int0ISR()

{

if (debounce(&int0time)) {

OSGiveSema(&task1Go);

}

}

//ISR for task 2

void int1ISR()

{

if(debounce(&int1time)) {

OSGiveSema(&task2Go);

}

}

void setup()

{

// Set pins 6 and 7 as output

pinMode(6, OUTPUT);

pinMode(7, OUTPUT);

// Initialize the OS

OSInit(2);

//attach each ISR to corresponding pin

attachInterrupt(INT0, int0ISR, RISING);

attachInterrupt(INT1, int1ISR, RISING);

// Create the binary semaphores

OSCreateSema(&task1Go, 0, 1);

OSCreateSema(&task2Go, 0, 1);

// Add in the tasks

OSCreateTask(INT0\_PRIORITY, task1, NULL);

OSCreateTask(INT1\_PRIORITY, task2, NULL);

// Launch the OS

OSRun();

}

void loop()

{

// Empty

}

// Do not modify

int main()

{

init();

setup();

while(1)

{

loop();

if(serialEventRun)

serialEventRun();

}

}

**Question 8 (4 marks):**

The LED attached to pin6 and connected to INT0 has a higher priority compared to the LED attached to pin7 and connected to INT1. However, both LED seem to blink simultaneously.

Actually, the higher priority LED will be put to sleep by OSSleep every half cycle for 250ms each. Therefore, while the higher LED sleeps, the lower priority LED wakes up and blinks. Thus, each LED changes state while the other is sleeping, which makes it seem like both blink simultaneously.

**Question 9 (4 marks):**

When the INT0 button is pressed after the higher priority LED has blinked twice, the LED at pin6 will blink together with the LED at pin7. When the button is pressed, the LED at pin7 will pre-empt the one at pin6 and start blinking. In the meantime, the LED at pin6 will continue to blink. This is because the lower priority LED changes state only when the higher priority LED is sleeping, hence it appears as if both the LED are blinking simultaneously.

**Question 10 (6 marks):**

The function implemented using ArdOS is similar to the function queue scheduling because the tasks have fixed priorities. It is different because now the tasks can be pre-empted, unlike previously in function queue scheduling implementation where pre-emption is not possible. Using the ArdOS, which contains the sleep function, allows for pre-emption. This is why the way the LED flash using ArdOS is different from otherwise. Besides, ArdOS also supports the use of semaphore to coordinate between tasks while the Function Queue Scheduling is not able to. This will enable tasks to communicate with each other using semaphore which is also a reason why pre-emption is possible as higher priority tasks will block lower ones and only when higher priority tasks finish and give semaphore, lower priority tasks taking that semaphore can be resumed.

**Total Marks: / 45**