**CG2271 Real Time Operating Systems**

**Lab 5**

**Answer Book**

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

Question 1 (6 marks)

It is possible to create two different tasks using the same function as the priority of both the tasks is different. The third parameter is also different. ((void \*) 1, (void \*) 2).

The function is like a blueprint. Inside the setup, we create two instances of this blueprint. Each task (each instance) will have its own context. Therefore, we can create two instances of this function.

Question 2 (3 marks)

We expect to see:

Hello there! I am task # 1

Hello there! I am task # 2

Hello there! I am task # 1

Hello there! I am task # 2

Hello there! I am task # 1

Hello there! I am task # 2

…

…

Question 3 (5 marks)

There is a race condition on the buffer such that more both the tasks are trying to access and change the buffer at the same time, resulting in the serial port is not able to send out all the data in the buffer.

Question 4 (5 marks)

OSCreateQueue function will initialize a FIFO queue. The parameters are, an integer buffer to store the message, length which stores the length of the integer buffer and the pointer to the queue.

OSEnqueue function will add a task number to the back of the queue. The first parameter is the task number and the second parameter is the pointer to the queue.

OSDequeue function will read the task number from the queue and remove the first task from the queue and blocks if the queue is empty. There is only one parameter: the pointer to the queue.

The ArdOS queue is different from the normal queue learnt in the data structure, as we have to declare the size to use the queue in ArdOS and we don’t have to declare the size in the normal queue.

If the ArdOS queue is full, if another task has to be enqueued, there is a chance that it will be lost. Hence, the task calling the dequeue function should have higher priority than the task calling the enqueue function. Therefore, handling the ArdOS queue requires some sort of priority.

Question 5 (3 marks)

The program prints the following outputs:

Task 1

Task 2

Task 1

Task 2

Question 6 (5 marks)

The dequeue function is dequeuing data from the queue at a fast rate. The serialPrint function will send data to the serial port at a fast rate. So the buffer will be overrun and the data will be corrupted. This is why we need the OSSleep(50) to slow down the dequeuing rate.

Question 7 (5 marks)

The appearance of task1 only indicates that only task1 is being queued and dequeued. This is possible when the rate of dequeue is slower than the rate of enqueue. Hence, when the OSSleep function is implemented, it slows down the dequeuing rate and only task1 is enqueued as it has higher priority as compared to task2. The queue ignores the task2 as it concentrates on task1.

Question 8 (8 marks)

void OSCreateBarrier(unsigned int count, struct OSBarrier \*barrier)

{

barrier->count=count;

OSCreateSema(&barrier->sema, 0,1);

}

The function initializes the count to the expected number of tasks going to reach the barrier. The semaphore of the barrier is set to be binary and initialized to zero.

Question 9 (10 marks)

void OSReachBarrier(struct OSBarrier \*barrier)

{

barrier->count--; // Comment 1

if(barrier->count) // Comment 2

{

OSTakeSema(&barrier->sema);

}

OSGiveSema(&barrier->sema);

}

**Comment 1:** Every time a task reaches the barrier, it calls this function and the count is decremented by one, indicating that the number of remaining tasks is reduced.

**Comment 2:** If there is still remaining task, the current task will take the semaphore and is blocked because the semaphore value is 0.

Each task calls the OSReachBarrier function once it reaches the barrier. When the last task that reaches the barrier calls the function, the count value is set to 0. In this case the if condition fails and the OSGiveSema function is called to unblock the rest of the tasks. The last task would be the one that wakes up the other tasks. The task with the highest priority (task 2) will be unblocked first, wake up the next highest priority task (task 3) and cross the barrier. Task 3 will be unblocked, wake up task 1 and then cross the barrier. Finally task 1 will cross the barrier.

Question 10 (5 marks)

Priority of task 2 > task 3 > task 1. After all the tasks have reached the barrier, all the tasks will be executed according to their respective priority. So task 2 will cross before task 3 and task 1, resulting on what we see on the screen.

**Total: \_\_\_\_\_\_\_ / 55**