NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

EXAMINATION FOR
Semester 1 AY2010/2011

CG2271/CS2271 – Real Time Operating Systems

Nov/Dec 2010 Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This examination paper contains **THREE** questions and comprises **SEVENTEEN (17)** printed pages, including this page. The weightage for each question is as indicated, and total 100 marks. Note that the marks distribution is not equal across questions.

2. Answer **ALL** questions within the spaces provided in this booklet. Whatever is written on the back will be ignored of the page or outside the box boundaries will be ignored.

3. This is an Open Book examination.

4. Please write your Matriculation Number below.

MATRICULATION NO: _____

This portion is for examiner's use only

Question	Marks	Remarks
Q1	/30	
Q2	/25	
Q3	/45	
Total	/100	

Question 1 Process Synchronization (30 MARKS)

In this question we will explore setting up synchronization mechanisms on the ARM STR91x microcontroller operating in a single processor environment. We assume no FIQ or WIU is used.

a. Explain, with example, why atomic test-and-set (or atomic exchange) instructions are well suited to implement mutual exclusion. (5 marks)

b. The ARM STR91x microcontroller does not have atomic test-and-set or exchange instructions. There is, however, at least one way to make sections of code atomic in a one-processor system. Explain how and why it works. (5 marks)

c. Using your answer to part b, write the code for the following two functions. The first creates an atomic section of code, and the second leaves the section. (8 marks)

```
void make_atomic()
{









}

void leave_atomic()
{









}
```

d. Using the enter_atomic and leave_atomic functions in part c, implement the following synchronization structures. Assume the existence of a "put_to_blocked()" function that causes the OS to shift the calling task into the blocked queue, and pick another task to run. There is also a "put_to_ready()" function that moves a blocked task to the ready queue. Assume that the OS (somehow) knows which task to move to ready. For simplicity we will use integer mutexes, semaphores and condition variables. (4 marks each, total of 12 marks)

i. Mutex

```
void mutex_pend(int *mutex)
{









}

void mutex_post(int *mutex)
{









}
```

ii. Counting Semaphore

```
void csem_pend(int *sem)
{









}

void csem_post(int *sem)
{









}
```

iii. Condition Variables. These operate in the same way as pthread condition variables.

```
void cond_wait(int *condvar, int *mutex)
{









}

void cond_post(int *condvar)
{









}
```

Question 2 Interrupt Processing (25 MARKS)

a. Explain the causes of interrupt latency, and why is is important in real-time design. (4 marks)

b. On some microprocessors, rather than using an interrupt vector table, all interrupts (also called "exceptions") are vectored to a default address.

   i. Compared with the vectored interrupt approach, what are the relative advantages and disadvantages of this approach? (5 marks)

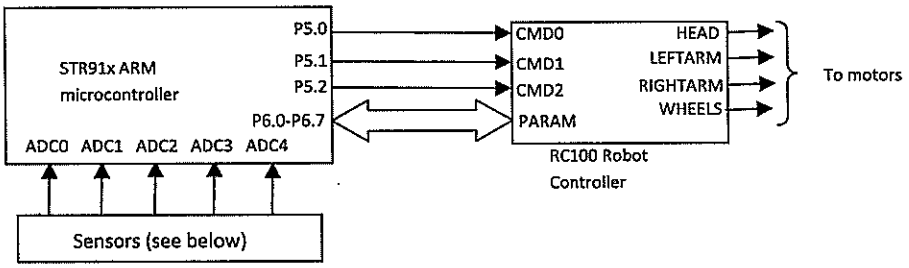
ii. Although all interrupts cause the microprocessor to jump to the same default address, we may still want to know exactly which IRQ line was pulled. Why? (4 marks)

iii. Suggest a scheme to accomplish part ii., and write the pseudocode that uses such a scheme perform appropriate handling, depending on which IRQ line was pulled. If necessary, assume the existence of handlers called "IRQx_handler" where x=0..7. (6 marks)

c. Again on some processors, rather than saving the PC of the instruction that was interrupted onto a stack, the processor saves it in a special register known as the "Exception Program Counter" or EPC. What is the impact of such a design on nested interrupts? How would a system programmer handle nested interrupts on such a processor? (6 marks)

Question 3 (45 MARKS)

We will now consider using the ARM STR91x microcontroller running at 96MHz to control a robot called Ceegee. The architecture of the Ceegee robot is shown below. The ARM STR91x is connected via GPIOs 5 and 6 to a RC100 robot controller chip, which is controlled using a three-bit command, followed by two bytes of parameters.



"P5.0" is shorthand for GPIO port 5, pin 0, etc. (Although Ceegee looks like a walking robot, he actually has wheels in his feet. Don't be fooled!)

Table 1 below lists the commands and parameters (a "motor step" is a unit of movement for a stepper motor). Commands are sent via GPIO5 pins 0, 1 and 2, and parameter 1 is sent via GPIO6 at least 20 microseconds after the command, and parameter 2 is sent at least 20 microseconds after parameter 1, also via GPIO6.

When the RC100 finishes performing an action, it will trigger an interrupt on VIC1.10, sending the current motor angle (ranged from 0 to 0xFF) back over GPIO6. GPIO6 must therefore be reconfigured as INPUT once the parameters 1 and 2 are sent to the RC100. The handler must acknowledge the interrupt by sending over command 0x4 with no parameters.

Action	Command	Parameter 1	Parameter 2
Move	0x0	Number of motor steps. 0x0 to stop, 0xFF to turn the motor continuously	0x0 to move forward, 0x1 to move backwards
Move left arm	0x1	Number of motor steps. 0x0 to stop, 0xFF to turn motor continuosly	0x0 to move arm forward, 0x1 to move arm backwards.
Move right arm	0x2	Number of motor steps. 0x0 to stop, 0xFF to turn motor continuosly	0x0 to move arm forward, 0x1 to move arm backwards.
Turn head	0x3	Angle to turn. 0x0 to stop, 0xFF to turn 360 degrees	0x0 to turn clockwise, 0x1 to turn anti-clockwise
Acknowledge interrupt	0x4	Unused	Unused

In addition the Ceegee robot has the following sensors connected to the ARM through the ports shown:

Sensor	Port
Light	ADC channel 0
Temperature	ADC channel 1
Left microphone (to detect loudness)	ADC channel 2
Right microphone (to detect loudness)	ADC channel 3
Front sonar (detect distance)	ADC channel 4

A task ReadSensors (see later) will be created to read these sensors approximately every 10 ms.

We will run a customized version of uC/OS-II, but will handle interrupts directly instead of going through the OS, to minimize interrupt latency. There is a task called "RobotThinker" whose logic is irrelevant to us, but we send sensor readings through the following globals. RobotThinker will wait on a semaphore called "data_sem" until the variables are all updated, then read the following sensor variables.

Variable	Purpose
CGLight	Light sensor readings
CGTemp	Temperature readings
CGMike1	Left microphone readings
CGMike2	Right microphone readings
CGSonar	Front Sonar

RobotThinker will send commands back using the following command global variables:

Variable	Purpose
CGCommand	Command from RobotThinker. 0x0=Move, 0x1=Move Left Arm, 0x2 = Move Right Arm, 0x3=Turn Head.
CGParam	Parameter for the Command. -255 to -1 to move backwards/anti-clockwise, 1 to 255 to move forward/clockwise.

A task called "CommandRobot" (see later) will pend on a suitably set-up semaphore called "cmd_sem" which RobotThinker will post to when it has updated CGCommand and CGParam (you should note that the ranges for param1 and param2 above are 0 to 255 and make appropriate adjustments).

a. Write a function called "init" which does all the necessary initialization for GPIO5 and 6, and initializes the ADC for one-shot scan-mode no watch-dog ADC, with ECV interrupt. Ensure all SCU and related registers are properly set up for correct ADC operation and that the ADC is in idle mode, and that the relevant VIC lines are properly set up. Assume that the interrupt handler for the ADC is called "ADCHandler", and the handler for the RC100 is called RCHandler. (10 marks)

```
void init()
{
// Set up GPIO 5




// Set up GPIO 6




// Set up VICs
```

Continued on next page.

```
// Set up and configure ADC and put into idle.




}
```

Page is intentionally left blank

b. The function ADCHandler has a modifier "__irq". Explain what this modifier is for and why it is necessary. (3 marks)

c. Write the code for ADCHandler, which copies the contents of channels 0 to 4 to the relevant global variables, and post to data_sem so that RobotThinker can read the variables. (5 marks)

```
__irq void ADCHandler()
{




}
```

d. Write a function called "send_command" which takes in paramater cmd containing a command to be sent to RC100, and param1 and param2 which are parameters to be passed along. The param1 and param2 parameters are irrelevant for cmd=0x4. Remember to switch GPIO6 to input mode so that the RC100 can send back motor position data. (5 marks)

```
void send_command(int cmd, int param1, int param)
{









}
```

e. Write the code for the CommandRobot task, which pends on a semaphore "cmd_sem". When RobotThinker posts to cmd_sem, CommandRobot will send appropriate commands and parameters from CGCommand and CGParam to "send_command". (5 marks)

```
void CommandRobot(void *ptr)
{









}
```

f. Write the code for interrupt handler RCHandler. The data sent back by the RC100 is stored in a global variable called CGReadBack. (5 marks)

```
__irq void RCHandler()
{









}
```

g. Write the code for task "ReadSensors", which start ADC conversion every 10 ms. (5 marks)

```
void ReadSensors(void *ptr)
{









}
```

h. Write the code to declare the semaphores, and for main which will call the "init" function in part a, set up all the semaphores properly, and register the "ReadSensors" and "CommandRobot" and "RobotThinker" tasks with priorities 1, 2 and 3 respectively and start the OS. (7 marks)

END OF PAPER