# NATIONAL UNIVERSITY OF SINGAPORE

**SCHOOL OF COMPUTING**

**SAMPLE ASSESSMENT**
**Semester 2 AY2019/2020**

**CG2271 – Real-Time Operating Systems**

**April/May 2020** **Time Allowed: 2 hours**

**INSTRUCTIONS TO STUDENTS:**

1. This assessment paper contains **SEVEN (7)** questions and comprises **NINETEEN (19)** printed pages, including this page.
2. Students are required to answer **ALL** the questions.
3. Write your answers within the space provided. Answers written on other parts of the answer script will not be graded unless you specify explicitly.
4. You are allowed a single double-sided A4 Reference Sheet.
5. Only Stand-Alone Calculators are allowed.
6. Please write your student number below. Do not write your name.

**Student Number:** ____________________________

This portion is for examiner's use only

Q1. (a) What is meant by unbounded priority inversion? Explain your answer clearly using appropriate diagrams.

(5 marks)

(b) Describe two schemes that can be used by an Operating System to prevent priority inversion.

(4 marks)

(c) Describe how we can prevent deadlocks.

(3 marks)

Q2. (a) Examine the following RTOS code in Figure Q2a.

Assume the following:

- The main function has created these 2 tasks and started the OS
- TaskA was created first, followed by TaskB.
- There are no other tasks or interrupts in the system
- The main function has created a semaphore *rdysem* and initialized it to a value of '1'
- There is a global integer variable *cntr* initialized to a value of '0'
- There is a global integer variable *init* initialized to a value of '1'
- TaskA has a higher priority than TaskB.
- All OS calls are BLOCKING by default.

(i) Which of the printf messages, "Task A" or "Task B" will be printed out? Explain your answer.

(4 marks)

(ii) Use a state-transition timing diagram to clearly show the changes in the states of both TaskA and TaskB. Your diagram must indicate the events that caused the OS to switch between both the tasks.

(6 marks)

(iii) Describe how the value held in the '*cntr*' variable will vary over time.

(4 marks)

```
static void TaskA(void *pdata)
{
     INT8U err;

     while(1)
     {
         xTaskDly(321);
         if(init)
         {
             printf("\nTask A");
             init = 0;
         }

         xSemaphoreTake(rdysem, 0, &err);
         cntr++;

         if(cntr > 6)
         {
             cntr = 10;
         }

         printf("%d ", cntr);
         xSemaphoreGive(rdysem);
    }
}

static void TaskB(void *pdata)
{
     INT8U err;

     while(1)
     {
            if(init)
            {
                printf("\nTask B");
                init = 0;
            }

            xSemaphoreTake(rdysem, 0, &err);

            if(cntr == 5)
            {
                cntr = 0;
            }
            xSemaphoreGive(rdysem);
     }
}
```

Figure Q2a

Q3. (a) The code in Figure Q3a shows 2 tasks written in an RTOS environment.

You can assume the following:

- Task 1 has a higher priority than Task 2
- The button_pressed() function will return a logical '1' when the physical button is pressed.
- A MessageQ called "OSq" has already been created with a buffer size for 10 messages.
- The xQueueSend(), xQueueReceive() and xTaskDly() shown here are the simplified versions of their actual implementation but with the SAME functionality.
- All OS calls are BLOCKING by default.

```
void Task1 (void)
{
     int tx_msg;

      while(1)
      {
          while(button_pressed() == 0) { }
          tx_msg = 1;

          xQueueSend( &OSq, tx_msg);
          xTaskDly(10);
     }
}
void Task2 (void)
{
     int rx_msg;

     while(1)
     {
          rx_msg = xQueueReceive(&OSq);
               if(rx_msg)
                    data_processing_func();
     }
 }
```

Figure Q3a

(i) Explain how the code will execute and the expected outcome.

(6 marks)

(ii) Suggest a means of improving the efficiency of the code to allow the multi-tasking system to fully utilize the CPU's resources. Modify the code to show the improvements you suggested.

(4 marks)

Q4. (a) Assume that ‘a’ is a local integer variable already declared at the start of a function. Consider the following C code line within the function:

**a += 10;**

Is this an atomic operation? Explain your answer.

(3 marks)

(b) Examine the following C function as shown in Figure Q5b.

```
long int compute_square(int num)
{
        return (num * num);
}
```

Figure Q5b

Is the function reentrant? Explain your answer.

(3 marks)

(c) You have decided to use Deadline-driven scheduling to assign priorities for the different tasks in your system. Consider the following two tasks, both initiated at the same time:

Task1: Computation Time = 2 sec, Deadline = 7 sec
Task2: Computation Time = 4 sec, Deadline = 5 sec

Which task should run first in order to ensure that both deadlines are met. Explain your answer clearly using a timing diagram.

(6 marks)

Q5. Suppose that the *p*=calloc(*n*, *b*) returns a pointer *p* to *n* consecutive units of memory each *b* bytes long, in the form of an array. For example, *p*=calloc(5, 6) will return a pointer *p* to a block of memory that is 30 bytes long. free(*p*) frees the memory pointed to by *p*, and sizeof(*x*) returns the size of variable *x* in bytes.
A char variable is 1 byte long, and a double variable is 8 bytes long.
We have a memory system that consists of 256 bytes in total and memory is allocated in units of 8 bytes (e.g., even requesting for 1 byte will still return an 8-bytes of memory). We assume starting at address 0.

Consider the following program fragment:

```
double *A = calloc(10, sizeof(double));
double *B = calloc(10, sizeof(double));
char *C = calloc(12, sizeof(char));
free(B);
double *D = calloc(10, sizeof(double));
```

i. We are using linked-lists to manage free memory. Sketch the linked-lists after each memory operation for a **next fit** policy. In each sketch, you need to indicate a memory region associated with the variable name and the memory size. Replace the variable name with "unused" if a memory region is unused. Indicate FAIL if you are unable to perform the operation shown, and explain why the operation fails. Please show the sketches for the memory operations.

(5 marks)

double *A = calloc(10, sizeof(double));

```
double *B = calloc(10, sizeof(double));
```

```
char *C = calloc(12, sizeof(char));
```

```
free(B);
```

```
double *D = calloc(10, sizeof(double));
```

ii. What is the percentage of memory that is wasted because of internal fragmentation after these memory operations? Express your percentage with respect to the total amount of memory requested by the program.

(2 marks)

Q6. For the following C program snippet, identify where the various data will be located in the memory when the program is loaded into a processor system.

(7 marks)

```
int a = 100;
int b;
char const c[4] = "1234";

typedef struct
{
     int d;
     char e;
} rec;

void main(void)
{
     int x;
     rec *ptr_rec;
     ptr_rec = malloc (sizeof(rec));
     ...
     // Some other code
}

int func1(int arg1, arg2, char arg3, arg4, arg5)
{
     int y;
     static int z = 0;
     ...
     // Some other code
 }
```

*Answer for Q6:*

Q7. You need to design a multi-threaded calculator with the following requirements:

- The software will have **ONLY THREE** Tasks
  - TaskInput: To capture User Input (One at a time, interleaved with operation capture)
  - TaskOp: To capture the intended operation (add, subtract, multiply or divide)
  - TaskResult: To display the Result

- Sample Operation of the Calculator:

```
        Enter First Number: 5
        These are the supported operations.
        1. Add
        2. Subtract
        3. Multiply
        4. Divide
        Enter desired operation: 1
        Enter Second Number: 2
        Result: 7
```

- You **MUST** follow the sample sequence shown above.
  i. Capture the first input
  ii. Get the required operation
  iii. Capture the second input
  iv. Display the final answer.

- Strictly NO Global Variables for holding any data.

Only OS constructs can be declared as Global.

- You are free to use any number of OS constructs such as Mutex, Semaphore, Message, Mailbox, etc. You do not need to know the exact RTOS syntax. A simplified version like CreateSem(rdysem, 1), SemTake(rdysem), MutexLock(shard_var), etc is sufficient. Please provide necessary comments to explain whenever necessary.

- Your implementation only needs to handle integer values. For the Division operation, the Quotient is the answer, and the Remainder can be discarded.

- You can use printf() to display a message and scanf() to capture the user input.

- You can assume that a TaskMain() has already created TaskInput() (Priority: High), TaskOp() (Priority: Medium) and TaskResult() (Priority: Low) with the priorities specified in the brackets. TaskMain() has gone to a permanent BLOCKED state after creating the 3 tasks above.

Provide the code for the **THREE** Tasks with necessary comments.

(20 marks)

**THE END**