CG2271 Real Time Operating Systems
Tutorial 10

Question 1

Suppose that an operating system does not have any way to see in advance if it is safe to read from a file, pipe or device (i.e. to see if a read results in a block), but allows clocks to be set that interrupt blocked system calls. Is it possible to implement a threads package in user space under these circumstances? Discuss.

Question 2

Assume that the following code is the only code in the system that uses the variable `iShareDeviceXData`. The routine `vGetDatafromDeviceX` is an interrupt routine. Now suppose that instead of disabling all interrupts in `vTaskZ` as shown below, we disable only the device X interrupt, allowing all other interrupts. Will this still protect the `iSharedDeviceXData` variable? If not, why not? If so, what are the advantages (if any) and disadvantages (if any) of doing this compared to disabling all interrupts?

```
int iSharedDeviceXData;

void interrupt vGetDataFromDeviceX(void)
{
      iShareDeviceXData = !! Get data from device X hardware
      !! reset hardware
}

void vTaskZ(void) /* Low priority task */
{
      int iTemp;
      while(1)
      {
            ……
            !!disable interrupts
            iTemp = iSharedDeviceXData;
            !! enable interrupts
            !! compute with iTemp
      }
}
```

Question 3

Show, on the two (unrelated) C!! code fragments below running on a pre-emptive RTOS, under what circumstances they might fail, and what the effect of the failure(s) is/are going to be. You should also indicate on the code fragment how to correct the problem, if possible.

C!! Code Fragment 1

```
int stack[MAX_SIZE];
int sp = 0;

void push(int data)
{
      if(sp < MAX_SIZE)
      {
            disable_interrupts();
            stack[sp++] = data;
            enable_interrupts();
      }
}

int pop()
{
      if(sp > 0)
      {
            disable_interrupts();
            return stack[--sp];
            enable_interrupts();
      }
}
```

C!! Code Fragment 2

```
int sharedData1, sharedData2;

/* All of these are initialized
in main, which is omitted here */
OSSemaphore sema1, sema2;
OSMailBox mb1, mb2;

void task1()
{
   int data;
   recvmsg(mb2, &data);
   take_sema(sema2);
   !! Do some processing on sharedData2
   sendmsg(mb2, sharedData2);
   take_sema(sema1);
   !! Do some processing on sharedData
   sendmsg(mb1, sharedData);
```

```
   release_sema(sema1);
   release_sema(sema2);
}

void task2()
{
   take_sema(sema1);
   !! Do some processing on sharedData
   sendmsg(mb2, sharedData1);
   take_sema(sema2);
   !! Do some processing on sharedData2
   recvmsg(mb2, sharedData2);
   recvmsg(mb1, &sharedData1);
   release_sema(sema1);
   release_sema(sema2);
}
```

Question 4

a. Explain the similarities and differences between processes and threads. In particular, explain why threads are more desirable than processes when we need multiple execution streams.

b. Referring to the context saving and restoring macros (portSaveContext and portRestoreContext) in FreeRTOS, are tasks in FreeRTOS treated like threads or processes? I.e. given two tasks in FreeRTOS, are they more similar to threads or processes? Explain your answer.