CMSC 412 Midterm #1 (Fall 2011) - Solutions

1.) (20 points) Define and explain the following terms:

a) Starvation

A steady stream of one process/activity can prevent others from running, caused by priority or synchronization implementation.

b) Interrupt

A hardware event (triggered by HW or SW) that causes a processor to stop doing what it was doing and transfer control to another region of code (i.e. an interrupt handler function).

c) Deadlock

Situation where no process from a group of processes is making forward progress. Indicated by presence of four conditions (mutual exclusion, hold and wait, non-preemption, circular wait).

d) Layering

Construction of a system such that functionality of layer n can only use functionality of later n-1 and export functionality to layer n+1.

2.) (20 points) Synchronization: Give an implementation of the readers/writers problem using semaphores that allows at most 5 readers at once. There can be at most one writer at once. Readers and writers may not be active at the same time. It should use no busy waiting (i.e. blocking on semaphore operations). Full credit requires a starvation free solution.

Semaphore and variable declarations:

```
Semaphore mutex = 1
Semaphore writer = 0
Semaphore reader = 0

int nReader = 0
int nWriter = 0
int wReader = 0
int wWriter = 0
```

Reader():

```
    while (1) {

        P(mutex)
        if (nWriters + wWriter == 0 & nReader < 5) {
            nReaders++;
            V(mutex);
        } else {
            wReaders++;
            V(mutex);
            P(reader);
```

```
                }

                Read operation;

                P(mutex);
                nReaders--;
                if (wWriters > 0 & nReaders == 0) {
                        wWriters--;
                nWriters++;
                        V(writer);
                } else if (wReaders > 0 & wWriters == 0) {
                        nReaders++;
                        wReaders--;
                        V(reader);
                }
        V(mutex);
```

Writer():

```
        while (1) {

                P(mutex);
                if (nReader + wWriter + nWriter == 0) {
                        nWriter++;
                        V(mutex);
                } else {
                        wWriter++;
                        V(mutex);
                        P(writer);
                }

                Write operation;

                P(mutex);
                NWriter = 0;
                If (wReaders > 0) {
                         for i = 1 to min(wReaders,5) {
                                V(readers)
                                nReaders++;
                                wReaders--;
                        }
                else if (wWriters > 0) {
                        wWriters--;
                        nWriters++;
                        V(writer);
                }
                V(mutex);
        }
```

3.) (16 Points) Scheduling

a) Given a round robin scheduler (with a quantum of 1 unit), and the following jobs, indicate when each job finishes:

Arrival Time	Required Time	Completion Time
0	10	**17**
3	2	**5**
6	6	**16**
20	2	**21**

b) Why have scheduling quantum remained relatively the same for the past 30 years (around 3-10 ms) despite many orders of magnitude increase in processor speed?

It's tied to human perception time and not to any technical artifact.

4.) (12 points) Explain the difference between processes and threads. How do kernel only threads (i.e. in the project) differ from user threads scheduled by the kernel?

Processes have their own address space, but threads share an address space (code, heap, globals, but not stack) with one or more other threads. In rare circumstances threads can be managed by user code.

Kernel threads run in the kernel's address space, have ring 0 access, and lack a user context struct. Both kernel managed user threads and kernel threads have a process id, and are scheduled by the scheduler.

5.) (12 points) Can disabling interrupts be used to provide mutual exclusion in a kernel? If so, under what conditions?

Yes, but only for systems with a single core/CPU. On multi-processors, several threads can be running at once, so even if you disabled all interrupts globally, mutual exclusion is not achieved.

6.) (20 points) Project

a) In project #2, setup Frame copies the interrupt state onto the user stack even though there is a copy on the kernel stack. Why does it do that?

To allow restoring execution to the normal user context after running the signal handlers.

b) When implementing PS, what is the purpose of the len parameter passed from user space and how should it be used in the kernel?

To indicate the size of the buffer that is available for the kernel to fill in with process info. Without this field, the kernel could overwrite the user buffer.

c) Given that all memory in the system is mapped into the kernel's address space, why is there a function in the kernel called Copy_From_User?

User memory starts a virtual address of 0 and the function adds the offset to create a kernel address. Also, to check that the range of member being copied is valid and doesn't exceed the limit (size of user's address space).