CMSC 412 Midterm #1 (Spring 2017) - Solution

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

a) Ring 3 on the x86 processor

The level at which user programs run. Some instructions and registers are disabled compared to ring 0 (kernel).

b) Round robin scheduler

A preemptive scheduling policy that runs each process in turn for a quantum. Tends to favor CPU bound jobs over IO bound ones.

c) Policy vs. Mechanism

Policy defines what to do and mechanism is how to do it. File access groups are a mechanism, only cmsc412 students are in the group to get access to class projects is a policy.

d) System call

A software trap into the operating system to allow a user program to request a service that is not directly available such as reading a file, creating another process, or sending traffic on the network.

2.) (20 points) - Synchronization Given a system that provides binary semaphores (semaphores whose values are either 0 or 1). Show the code to implement counting semaphores using binary semaphores.

CreateCountingSemaphore(int initialValue):

```
Sem mutex = 1
Sem wait = 0
count = initialValue

P:
      P(mutex)
      Count—;
      If (count >= 0)
            V(mutex)
      Else
            V(mutex)
            P(wait)

V:
      P(mutex)
      count = count + 1
      If (count <= 0)
            V(wait)
      V(mutex)
```

3.) (16 Points) Deadlock

a) (6 points) Why is mutual exclusion a necessary condition for deadlock?

If there is no mutual exclusion then there is no need to block processes, and therefore there would be no deadlock.

b) (10 points) Consider a system with the resources shown. Process P0 is requesting 3 units of resource A, can the system allow this request and avoid deadlock? If so show a safe sequence for it, if not show the set of processes involved in the deadlock.

Three resources: A, B, C (10, 5, 7 instances each). The snapshot of the system:

	Alloc	Max	Avail	Need
	A B C	A B C	A B C	A B C
P0	0 1 0	7 5 3	3 3 2	7 4 3
P1	2 0 0	3 2 2		1 2 2
P2	3 0 2	9 0 2		6 0 0
P3	2 1 1	2 2 2		0 1 1
P4	0 0 2	4 3 3		4 3 1

Yes, this is safe. The safe sequences are:

P3 P1 P0 P2 P4

P3 P1 P0 P4 P2

P3 P1 P4 P0 P2

P3 P1 P4 P2 P0

4.) (12 points) Safe User Instructions. Which of the following situations are safe to allow user processes to perform directly (i.e. don't create security holes that allow a user process to compromise another user process or the kernel. Crashing the user process running the code is ok):

Safe	**Unsafe**	Write values on the kernel stack
Safe	**Unsafe**	Disable interrupts
Safe	Unsafe	Overwrite an arbitrary value on the user stack
Safe	**Unsafe**	Read a block of data from the disk
Safe	**Unsafe**	Draw a pixel on the screen
Safe	Unsafe	Change the program counter (EIP) to an arbitrary value

5.) (18 points) Signals in GeekOS:

a) (6 points) In GeekOS (and most operating systems), signal handlers remain installed at the completion of a signal hander. Consider a system where signal handlers are reset to SIG_DFL by the kernel just before calling the signal handler. How can a user ensure that sig alarms are still handled without missing an alarm in such a system?

Have the signal hander reset the signal before it exists.

b) (6 points) In project 2, why did you have to copy the interrupt state from the kernel stack to the user stack as part of handling a signal?

The kernel stack is cleared on return of a system call, so returning into user space to call a signal handler will clear the kernel stack, and re-entering the kernel at the end of the signal handler will overwrite this stack. Thus we need to copy the state somewhere (such as the user stack) so that it can be restored after the signal hander completes.

c) (6 points) When a signal handler completes, is the resumed user process guaranteed to run on the same core as the signal handler ran? Explain your answer.

No, the return from system call path includes a call to the scheduler so a different process to be picked.

6.) (14 points) - Project

a) (8 points) A critical function in the kernel is CopyToUser. Write the code for this function. If you can't recall specific global variables or fields in data structures, don't worry if the purpose is clear from the name you will get full credit.

Int CopyToUser(void *user , void *kernel, int size)

```
If (user + size < userContex->size && user < userContect->size)

      memcpy(kernel, user + userContext->start, size);
```

b) (6 points) If you wanted project #0 to limit the number of system calls to a specific number per process per minute rather than per process (over its lifetime), explain how you would implement this?

Use the same alarm mechanism in the kernel that was used to create signal alarms in project #2. The callback function would reset the limit counter for each process back to zero.