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

a) OS Kernel
The most protected part of the Operating system. Generally runs at ring 0 and includes handling scheduling and interrupts.

b) Multi-level feedback queue scheduler
A scheduling approach that uses different priority queues to manage jobs. Based on job behavior (such as not using a full scheduling quantum or waiting time in a queue) jobs are moved between queues.

c) Spawn system call (compared to fork)
Creates a new process and specifies the new program to run. It differs from fork which simply creates a copy of the current running process (and thus the same program).

d) Dispatcher
Also called the shortterm scheduler. Responsible for selecting a job to run and switching the state of running processes to make the new process run (involves saving/restoring registers and other machine state).

2.) (20 points) - Synchronization

Given an implementation of general (counting) semaphores, implement bounded counting semaphores where each semaphore is declared with initial values, but also a maximum value. A V operation on a bounded counting semaphore that is at its maximum value should return immediately and not change the state of the system. P works the same as a general semaphore.

CreateBoundedSemaphore(int max, int initialValue):

Shared int s.max = max
Shared int s.curr = initialValue
Semaphore s.mutex = 1;
Semaphore s.wait = initialValue;

P_{bounded}

P(s.mutex)
 s.curr--
V(s.mutex)
P(s.wait)

V_{bounded}

P(s.mutex)
If (s.curr < s.max)
 V(s.wait)
  s.curr++
 V(s.mutex)
3.) (16 Points) Deadlock

a) (7 points) Why is circular wait only a necessary and not a sufficient condition for deadlock?

There could be another process (not involved in the circular wait) that could release a second instance of one of the resources involved in the circular wait and thus permit one of the waiting processes to proceed.

b) (9 points) Is this system in a safe state? If so show a safe sequence for it.

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

<table>
<thead>
<tr>
<th>Alloc</th>
<th>Max</th>
<th>Avail</th>
<th>Need</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A B C</td>
<td>A B C</td>
<td>A B C</td>
</tr>
<tr>
<td>P0</td>
<td>0 1 0</td>
<td>7 5 3</td>
<td>3 3 2</td>
</tr>
<tr>
<td>P1</td>
<td>2 0 0</td>
<td>3 2 2</td>
<td>1 2 2</td>
</tr>
<tr>
<td>P2</td>
<td>3 0 2</td>
<td>9 0 2</td>
<td>6 0 0</td>
</tr>
<tr>
<td>P3</td>
<td>2 1 1</td>
<td>2 2 2</td>
<td>0 1 1</td>
</tr>
<tr>
<td>P4</td>
<td>0 0 2</td>
<td>4 3 3</td>
<td>4 3 1</td>
</tr>
</tbody>
</table>

Unsafe/impossible Sequences: (from class: P1 P3 P0 P2 P4)
(24) P0 *
(06) P1 P0 *
(06) P1 P2 *
(02) P1 P4 P0 *
(02) P1 P4 P2 *
(24) P2 *
(06) P3 P0 *
(06) P3 P2 *
(02) P3 P4 P0 *
(02) P3 P4 P2 *
(24) P4 *

Save Sequences (from class: P1 P3 P0 P2 P4)
(06) P1 P3 *
(02) P1 P4 P3 *
(06) P3 P1 *
(02) P3 P4 P1 *

Full credit for YES, and then any of the 16 safe sequences.

4.) (12 points) Policy vs. Mechanism: Circle if the following are policies or mechanisms.

Policy     Mechanism
Users must change their passwords every 60 days
An operating system uses a timer to reclaim the cores from user processes
Processes owned by root have higher priority than normal user processes
User’s files are readable only by that user and their professor
An operating system includes semaphore system calls for synchronization
A list of runnable processes is stored in a heap

5.) (14 points) Process Manipulation

a) In GeekOS, the kill system call could not call Exit directly. However, the setup_Frame code for handling a SIGKILL could call Exit. Why?

Exit terminates the currently running process. Setup_Frame is running in the context of the target process, but the kill system call is not necessarily running in the target context.

b) In GeekOS, why is turning off interrupts (i.e. calling Disable_Interrupts) not enough to ensure atomic access to a critical section?

In a multi-core operating system, activities are still happening on the other cores even if interrupts are disabled.
6.) (20 points) - project

a) In project #2, Complete_Handler only needed to POP the signal number and not the address of the "signal trampoline" (supplied by Sys_RegDeliver) even though Setup_Frame pushed them both. Why?

The address of the signal trampoline was popped by the return instruction of the signal handler (which invoked the signal trampoline function)

b) In projects 0 through 2, even if you didn’t care about memory protection for user processes, why is it necessary to have the base and limit registers in the project for user processes?

The base register also allows the address of user program to start at 0 and thus permits them to be loaded into any memory location available.

c) In GeekOS, you want to add a system call that returns the current core a process is running on. How would your system call figure this out?

Look in g_currentThread. Each element of the array is the currently running process on a specific core (index 0 is core 0, ...). A process can determine it’s current thread struct from the macro CURRENT_THREAD).