CSC 360-W1

1a) The main differences between Kernel and User mode:

- When the computer system is executing on behalf of a user application, it is in user mode
- When the computer system is executing on behalf of the operating system (via system call), the system transitions from user to kernel mode.
- Only privileged instructions are executed in kernel mode.
- When control is given to a user application, the mode is set to user mode, eventually switching back to Kernel mode (the operating system) through a trap, interrupt, or system call.
- When computer in User mode, the mode bit is a 0
- When computer in Kernel mode, the mode bit is a 1

1b) From the viewpoint of operating systems, why are they needed?

- Operating systems give the ability to run multiple programs, keeping the CPU or the I/O devices busy at all times
- Operating systems allow for Multiprogramming, which increase CPU utilization, as well as organizing programs so the CPU always has one to process to execute
- Multitasking, an extension of multiprogramming, allows for switches in the CPU constantly, providing the user with a fast response time

1c) What are the main differences between the two modes?

- When a system call or interrupt shifts the CPU from user mode to kernel mode, and shifts back to user mode when the system call/interrupt is finished, it's known as a Mode Switch
- A Context Switch is when the CPU saves the state of a process and restores the state of another lined up process executing the next one. It is often implemented with a queue.

1d) What are the pros and cons of micro-kernel structures in operating systems?

- Pros:
  - It takes less time to perform an operation
  - It is easier to troubleshoot
  - Kernel needs less modifications because all new services are added to the user space
  - Easier to port from one hardware design to another
- Cons:
  - There is a lot of system function overhead with microkernels, affecting performance of computer systems with a microkernel

2a) please write down all possible outputs when running this program

0
1
2

0
2
1

2b. It would look something like this..

```
if (fork()){
        wait(NULL);
        i+= 2; OUTPUT;
{
```

So, you would put the wait right after the fork is executed so that you're telling the parent to wait and the child to execute first.

3. Processes have three major states: running, blocked (also known as waiting), and ready. For each of the following possible state transitions, explain whether it is feasible: if feasible, give an example; if not, give reason

A. Feasible-it takes just one I/O event or wait to get from a running state to a blocked state. Like for example, when a process needs a user input

B. Infeasible- It takes an I/O event or completion and a scheduler dispatch to get to a running state from a waiting state, which makes it infeasible because it's more than one operation.

C. Feasible-from blocked to ready, it takes just one operation, an I/O event or completion, which makes it feasible. For example, if a process that requires user input have already received user input it becomes ready to run and go to the ready state

D. Infeasible- it takes two operations, a scheduler dispatch and an I/O or event wait to get from ready to blocked, which makes it infeasible because it's more than one operation

E. Feasible- it takes one operation to get from ready to running, a scheduler dispatch. Many processes can be in a ready state, and the selector dispatch would select one process to be running on the processor core at any instant.

F. Feasible- it takes one operation to get from running to ready, and it's an interrupt, making it feasible. When this interrupt occurs, it gets shifted to ready until the interrupt is resolved