Quiz 1 Grading

Scheduling

Under Shortest Job First, if the following jobs (with indicated processing times) arrived at time zero:

A (10)	B (3)	C (20)	D (2)	E (5)

Answer the following questions:

Question 1 (5 points)

What is the throughput of these jobs?

This should be autograded, but you can double check that the regex worked.

Question 2 (5 points)

What is the average turnaround time for these jobs?

This should be autograded, but you can double check that the regex worked.

Microkernel/Monolithic Kernel

List an advantage and a disadvantage of both a Microkernel and a Monolithic Kernel architecture.

Question 3 (5 points)

What is one advantage of a microkernel architecture over a monolithic one? Be specific and explain why it is an advantage.

Microkernels are more modular. This could be identified in many ways:

- smaller code in kernel mode, lesser attack surface, fewer bugs, easier to get correct
- Micro/exo kernels can recover from servers failing by restarting them (just a userspace process)
- Micro/exo kernels can load different servers with different algorithms without rebooting (or because of an error)

Question 4 (5 points)

What is one advantage of a monolithic architecture over a microkernel one? Be specific and explain why it is an advantage. Do not just negate your answer to the previous question.

The biggest advantage of a monolithic kernel is that it has fewer context switches than a micro/exo kernel, and thus is likely to be faster. The micro/exo kernel probably has 4 switches (to kernel, to server,

back to kernel, back to application) whereas the monolithic kernel probably just has the 2 into and out of the server.

Question 5 (10 points)

If you have several concurrent tasks, when would you use threads and when would you use processes? What is the difference?

(2 points) Threads are parallelizable tasks whose code and data live within the same address space

(2 points) Processes are isolated parallelizable tasks, managed by the OS, and are given distinct address spaces (cannot share information without OS-mediated communication)

(3 points) We use threads when we wish to express concurrent tasks that cooperate in some way: probably sharing information through variables in the address space. They might also be used because they are "lighter weight" in that the context switch between threads needs to save/restore fewer variables than when switching between processes

(3 points) We use processes to express a task that should be isolated and competes for resources on the system with other processes. We make no assumptions about who wrote these tasks or if they're supposed to be cooperative. This gives better security through the address space abstraction.

Question 6 (4 points)

List two things that would be part of a process table entry (PTE) but not a thread table entry

Probably will see two of:

- Address space
- Open files
- Child processes
- Signals & handlers
- Accounting info
- *Global variables*

Question 7 (6 points)

How did we establish the operating system as an un-bypassable layer and give it the authority to manage resources?

(3 points) The CPU partitioned the instruction set into at least two categories: the unprivileged/protected/user mode instructions and the privileged/kernel mode instructions. The kernel mode instructions can be used to manage the system and are restricted to code designated as the kernel. The CPU knows what mode it is in.

(3 points) The work necessary to be done by the OS on a process's behalf is invoked by a special system call instruction that switches the mode. It generates an event that causes OS code to run, which, when returning from this interrupt, lowers privilege back to user mode.