Homework 1
CSCE 313-504

Derek Heidtke

September 28, 2015

1. Which of the following instructions should be privileged?

- **Set time-of-day clock.**
- **Read time-of-day clock.**
- **Clear memory.**
- **Disable interrupts.**
- **Change the memory map.**

Set time-of-day clock, clear memory, disable interrupts, and change memory map. Because all these instructions could be used maliciously or inadvertently to affect the execution of other applications or the kernel.

2. What is the main advantage of multiprogramming?

Before multiprogramming, user's had to submit their applications to batch systems where programs ran one at a time. Multiprogramming promotes better system utilization by allowing applications to stop execution (for various reasons: preemption, waiting for I/O, etc.) and give control over to other applications. This also allowed for the development of the operating system, because the idea of an operating system doesn't really make sense for batch systems.

3. When a user program makes a system call to read or write a disk file, it provides indication of which file it wants, a pointer to the data buffer, and the count. Control is then transferred to the OS, which calls the appropriate driver. Suppose that the driver starts the disk and then terminates until an interrupt occurs. In the case of reading from the disk, obviously the caller will have to be blocked (because there is no data for it). What about the case of writing to the disk? Need the caller be blocking awaiting completion of the disk transfer?

No, the caller can place the data into an output buffer. At that point, the application doesn't need the data anymore and can continue executing, and the I/O device will get to the buffer when it's ready.

4. Why are the locations of interrupt handlers generally not stored in a linked list?

Interrupt handling needs to be very fast. If handler locations are stored in a predefined vector (interrupt vector table), access time is O(1); whereas if they are stored in a linked list, access time falls to O(n), where n is the number of links that must be traversed to find the correct interrupt handler.

5. What are two differences between user level threads and kernel level threads? Under what circumstances is one type better than the other?

User-threads can run exclusively without triggering context switch (due to needing to access the kernel). Kernel-threads can run privileged instructions without having to trigger a trap/trap handler like a user-thread would have to do.

When a process solely needs to do a lot of computation, a user-thread will be faster; but, when privileged instructions are involved kernel-threads are the only ones that can execute them.

6. Describe the difference of degree to which the following scheduling algorithms discriminate in favor of short processes:

- **First-Come-First-Serve**

This algorithm is the least efficient for short processes because they will always have to wait after a long process. Even after the process gets to have some execution time, it will likely end up behind another long process (convoying). This causes many small processes to repeatedly pile up behind a long process.

- **Round-Robin**

This algorithm greatly favors short processes, specifically processes that are shorter than the preemption time quantum. If a process cannot finish in that amount of time, it will have to wait for it's next turn to come around.

- **Multilevel Feedback Queues**

This algorithm is a balance between the previous two; whether this scheme favors a short process or not depends almost entirely on its priority. If it has low priority, it must wait in the low tiers until it can run or until it is aged to a higher tier. If it is high priority, it will likely get to execute very quickly.

That being said, there is a slight favor towards short processes in this system: when a short process has low priority, it has a greater benefit of aging than does a long process (the longer the process, the more likely it is to be demoted). Also, if a short process has high priority, it is more likely to finish in the very short time quantum levels (and not suffer demotion) than a long process.