**Chapter 8**

Multitasking

1) What is the kernel's most basic task?

Implement multitasking capabilities, including interrupts

2) When an interrupt occurs, what part of the hardware state is saved by hardware?

The program counter, which specifies the place where the kernel should resume at when the time comes

3) What is the difference between an interrupt and a context switch?

An interrupt operates outside the process queue and is analogous to a pause button, whereas a context switch is a switch between the queued processes and is more like change the channel.

4) Give an example of an event that might cause a process to move from the blocked to the ready state.

A process is blocked while it waits for specific data to be accessed and read. Once that data is found, it can be unblocked and is queued to be ran (i.e. ready)

5) Why might a scheduler want to give higher priority to an I/O bound process?

An I/O process typically involves external environment interaction (via sensors or user input), which means that the process should be more flexible and responsive to any state changes.

When I make French toast, I usually make a batch of 12 slices.  But my griddle only has room for 8 slices.  Each piece of toast has to cook for 5 minutes on each side.  How can I schedule 12 slices onto 8 "cores" to minimize the elapsed time to cook all 12 slices?  (Note: this question is not hypothetical; this is really how I make French toast.)

Ideally, the first 8 slices can began simultaneously, and in 10 mins, the last 4 begin simultaneously. Assuming no human delays, this will take 20 mins. But the problem is simplified compared to the work of a real scheduler. These slices of toast have no priority, each slice is the same and can be treated the same as the next. Schedulers handle a lot more details on priorities and exceptions.