CS 33211 # Homework #1 **OS**

Due in class at 12:30pm on Monday 24 September 2007
typed answers preferred

---

1. **How does the use of interrupts permit more efficient usage of the CPU during I/O than software polling?**
   (*Note that I am <u>not</u> asking for a complete description of "interrupts" and "software polling".*)

   With software polling, the CPU has to continuously check to see if the I/O has finished, which prevents it from doing useful work during the I/O operation. With interrupts, the CPU can start an I/O operation and then do something useful until the I/O operation finishes.

2. **What is the relationship between high-level language statements such as "printf( )", APIs, and system calls?**
   (*Note that I am <u>not</u> asking for a complete description of "APIs" and "system calls".*)

   A high-level language statement may be implemented by multiple lower-level API functions, each of which may be implemented by multiple lower-level system calls. Further, a high-level language statement should work on any operating system or CPU, while an API function is limited to a specific class of operating systems (e.g., a POSIX API function supported by most versions of UNIX), and a system call is limited to a very specific operating system (e.g., Windows XP). A programmer needs to know the high-level language statements available, may not need to know the API functions available, and rarely needs to know the system calls available.

3. **In what ways has the definition of OS "kernel" changed over time?**
   In what ways has the definition remained the same?
   (*Note that simply <u>defining</u> "OS kernel" for two or three different systems does not answer the questions asked.*)

   The OS "kernel" has generally referred to the software above the hardware level that runs in kernel / protected mode.

   When kernels grew too large to be easily managed, a lot of the functionality that did not need to directly access the hardware, such as the CPU scheduler and file system implementation, was moved into user-mode processes, resulting in a smaller "micro"-kernel.

4. **In the UNIX process model, there are two "asleep" states and two "ready to run" states. How might it be useful to make this distinction?**

   In both of these cases, one of the states corresponds to a process in memory, and one corresponds to a process that has been swapped out to disk. Processes swapped out have to

be moved back into memory before they can be scheduled onto the CPU. A large number of ready to run processes swapped out indicates a lack of memory that can hamper performance.

5. What is the relationship between blocking send and receive operations and the five-state process model?

When a blocking send or receive operation blocks, the process moves from the running state to the blocked / waiting state. In the case of a send operation, it stays in that state until the data is received. In the case of a receive operation, it stays in that state until some data is received. Once that awaited event occurs, the process moves to the ready state.