

Homework 1
(Due Date: Check eCampus)

1. [1, 3] Which of the following instructions should be privileged?
 - (a) Set time-of-day clock.
 - (b) Read time-of-day clock.
 - (c) Clear memory.
 - (d) Disable interrupts.
 - (e) Change the memory map.
2. [1] What is the main advantage of multiprogramming?
3. [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?
4. [2] Why are the locations of interrupt handles generally not stored in a linked list?
5. [1] What are two differences between user-level threads and kernel-level threads? Under what circumstances is one type better than the other?
6. [1] Describe the difference of degree to which the following scheduling algorithms discriminate in favor of short processes:
 - (a) First-Come-First-Serve.
 - (b) Round-Robin.
 - (c) Multilevel feedback queues.
7. (Programming Assignment: `fork` gymnastics) Write a C/C++ program (call it `string_invert`) that takes a string argument from the command line and outputs the string in reversed order. You have two constraints:

Constraint 1: Each process can output at most one character. If you want to output more than a single character, you must fork off one or more processes in order to do that, and each of the forked processes in turn outputs a single character.

Constraint 2: Each process can fork-off at most one other process.

After the call to program `string_invert` with the command line argument, the output should appear, and no more processes should be running, in addition to the shell. Test your program on any UNIX/LINUX machine, and turn in the source code as part of the written assignment. (The source code should be at most a few lines long.)

Note: Do not create processes inside a loop (you are not supposed to start more than one process anyway). Check your logic to create processes and make sure that only a finite number of processes get created. If too many processes get created, the system will crash or not allow you to log in anymore. Periodically check your processes with the `ps -a` command or variations of it. If it shows that you have lots of processes running, abort them with the `kill -9 <pid>` command, where `<pid>` is the process id listed by the `ps` command.

8. **[BONUS; OPTIONAL; SLIGHTLY TRICKY]** (Programming Assignment: advanced `fork` gymnastics) Write a C/C++ program (call it `fork_fac`) that takes an integer argument from the command line and outputs the factorial of the argument, that is, the product of all integers from 1 to the given argument value. You have two constraints:

Constraint 1: Each process can do at most one multiplication. If you want to multiply more than two numbers, you must fork off one or more processes in order to do that, and each of the forked processes in turn can perform at most one multiplication.

Constraint 2: Each process can fork-off at most one other process.

After the call to program `fork_fac` with the command line argument, the output should appear, and no more processes should be running, in addition to the shell. Test your program on any UNIX/LINUX machine, and turn in the source code as part of the written assignment. (The source code should be at most a few lines long.) j

References

- [1] A. Silberschatz, P. Galvin, and G. Gagne, *Applied Operating Systems Concepts*, John Wiley & Sons, Inc., New York, NY, 2000.
- [2] Deitel, Deitel, and Choffnes, *Operating Systems*, Pearson / Prentice Hall, 2004.
- [3] A. S. Tanenbaum, *Modern Operating Systems*, Pearson / Prentice Hall, 2008.