CSCI 340 Midterm
Spring 2013

Each problem is worth 5 points. Submit your hardcopy solution to me **by 3 p.m., Friday, March 22**.

1. Suppose a computer system and all of its applications are completely bug free. Suppose further that everyone in the world is completely honest and trustworthy. In other words, we do not need to consider fault isolation.
   (a) How should the operating system allocate time on the processor? Should it give all of the processor to each application until it no longer needs it? If there are multiple tasks ready to go at the same time, should it schedule the task with the least amount of work to do or the one with the most? Justify your answer.
   (b) How should the operating system allocate physical memory between applications? What should happen if the set of applications do not all fit in memory at the same time?
   (c) How should the operating system allocate its disk space? Should the first user to ask be able to grab all of the free space? What would the likely outcome be for that policy?
2. Now suppose the computer system needs to support fault isolation. What hardware and/or operating support do you think would be needed to accomplish this goal?
   (a) For protecting an application's data structures in memory from being corrupted by other applications?
   (b) For protecting one user's disk files from being accessed or corrupted by another user?
   (c) For protecting the network from a virus trying to use your computer to send spam?
3. For the computer you are currently using, how should the operating system designers prioritize among reliability, security, portability, performance, and adoption? Explain why.
4. For each of the three mechanisms for supporting dual mode operations — privileged instructions, memory protection, and timer interrupts — explain what might go wrong without that mechanism, assuming the system still had the other two.
5. Define three styles of switching from user-mode to kernel-mode, and four styles of switching from kernel-mode to user-mode.
6. Which of the following components is responsible for loading the initial value in the program counter for an application program before it starts running: the compiler, the linker, the kernel, or the boot ROM?
7. Explain the steps that an operating system goes through when the CPU receives an interrupt.
8. Can UNIX fork return an error? Why or why not?
9. Can UNIX exec return an error? Why or why not?
10. What happens if we run the following program on UNIX?

```
main() {
   while (fork() >= 0)
      ;
}
```

11. Explain what must happen for UNIX wait to return (successfully and) immediately.

12. What happens if you run "exec csh" in the UNIX shell? Why?

13. What happens if you run "exec ls" in the UNIX shell? Why?

14. Consider the following program:

    ```
    main(int argc, char** argv) {
       forkthem(5);
    }

    void forkthem(int n) {
       if (n > 0) {
          fork();
          forkthem(n-1);
       }
    }
    ```

    How many processes are created if the above piece of code is run?

15. Consider the following program:

    ```
    main(int argc, char** argv) {
       int child = fork();
       int x = 5;

       if (child == 0) {
          x += 5;
       } else {
          child = fork();
          x += 10;
          if (child) {
             x += 5;
          }
       }
    }
    ```

    How many different copies of the variable x are there? What are their values when their process finishes?

16. Download this example code at the following URL:

    ```
    http://www.cs.utexas.edu/users/dahlin/osbook/code/threadHello.c
    http://www.cs.utexas.edu/users/dahlin/osbook/code/sthread.c
    http://www.cs.utexas.edu/users/dahlin/osbook/code/sthread.h
    ```

    Compile the code and run it several times. What output do you get if you run it? Do you get the sasme thing if you run it multiple times? What if you are also running some other demanding processes (e.g. compiling a big program, playing a Flash game on the Web, or watching streaming video) when you run this program?

17. For the threadHello program, what is the maximum number of threads that could exist while the program is running?

18. For the threadHello program, suppose that we delete the second for loop so that the main routine simply creates NTHREADS threads and then prints “Main thread done.”. What are the possible outputs of the program now? **Hint:** Fewer than NTHREADS+1 lines may be printed in some runs. Why?

19. For the threadHello program, the procedure *go()* has the parameter *np* and the local variable *n*. Are these variables *per thread* or *shared* state? Where does the compiler store these variables' states?

20. For the threadHello program, the procedure *main()* has local variables such as *ii*, *err*, and *status*. Are these variables *per thread* or *shared* state? Where does the compiler store these variables' states?