CSCI 3323 (Principles of Operating Systems), Fall 2020

Homework 2a

Credit: 55 points.

1 Reading

Be sure you have read, or at least skimmed, Chapter 2, sections 2.1 through 2.3.

2 Problems

Answer the following questions. You may write out your answers by hand and scan them, or you may use a word processor or other program, but please submit a PDF or plain text via e-mail to my TMail address. (No links to shared files on Google Drive please.) Please use a subject line that mentions the course and the assignment (e.g., "csci 3323 hw 2a" or "O/S hw 2a").

1. (10 points) Consider two systems:

   - System A supports processes only (no threads).
   - System B supports both processes and threads, with threads contained in processes, and every process having at least one thread (i.e., actual code is executed by threads; processes exist only to group threads).

   If you were designing data structures for a process table for System A and process and thread tables for System B, for each of the following say whether you would include it in the process table for System A, the process table for System B, the threads table for System B, or some combination of those (e.g., both process tables but not the threads table for B). Also briefly explain way.

   - A place to save CPU registers.
   - A place to save information about what memory is owned by the process or thread.

2. (5 points) When a computer is being designed, it is common to first simulate it using a program that runs one (simulated) instruction at a time. Even computers with more than one processor can be simulated strictly sequentially like this. Is it possible for a race condition to occur when, as in this situation, there are no truly simultaneous events? Why or why not?

3. (10 points) In class we discussed a proposed solution to the mutual-exclusion problem based on disabling interrupts, and rejected it because it doesn't work for systems with more than one CPU. For a system with a single CPU, however, this could be an acceptable solution if critical regions are short. Write pseudocode for an implementation of semaphores for a single-CPU system that might not have a TSL instruction but does have library functions `enable_int()` and `disable_int()` to enable and disable interrupts respectively. (I.e., say what variables you would need for each semaphore, and give pseudocode for `up()` and `down()`.)

4. (15 points) Restrooms are usually designated as men-only or women-only, but this requires having two restrooms if everyone is to be accommodated. A less expensive approach consistent with traditional cultural norms in the U.S. [1] would be to have one restroom with a sign on the door that indicates its current state — empty, in use by at least one woman, or in use by at least one man. If it is empty, either a man or a women may enter; if it is occupied, a person of the same sex may enter, but a person of the opposite sex must wait until it is empty. Write pseudocode for four functions to implement this approach: `woman_enter`, `man_enter`, `woman_leave`, and `man_leave`, to be used by the following pseudocode:

```
/* woman process */
while (TRUE) {
    woman_enter();
    use_restroom();
    woman_leave();
    do_other_stuff();
}
/* man process */
while (TRUE) {
    man_enter();
    use_restroom();
    man_leave();
    do_other_stuff();
}
```

   You can use any of the synchronization mechanisms we have talked about (shared variables, semaphores, monitors, or even message passing). *Hint:* The key issue in solving this problem is making processes interact in the desired way. Shared variables are helpful, but unlikely to work as desired unless you combine them with one of the synchronization mechanisms we discussed.

5. (15 points) Solve the dining philosophers problem with monitors rather than semaphores. (I'm looking for pseudocode here, similar to what I showed in class for the bounded-buffer problem.) (Despite what I say in class/notes about reading the literature, for this assignment do *not* look for a solution online or in another book; this is a problem you can and should try to solve just based on what we've done in this class.)

3 Pledge

Include the Honor Code pledge or just the word "pledged", plus *at least one of the following* about collaboration and help (as many as apply).[2] Text *in italics* is explanatory or something for you to fill in. For programming assignments, this should go in the body of the e-mail or in a plain-text file `pledge.txt` (no word-processor files please).

- This assignment is entirely my own work. *(Here, "entirely my own work" means that it's your own work except for anything you got from the assignment itself — some programming assignments include "starter code", for example — or from the course Web site. In particular,*

[1] Yes, this norm is changing! But I think this is an interesting problem and can't think how to reframe it ...

[2] Credit where credit is due: I based the wording of this list on a posting to a SIGCSE mailing list. SIGCSE is the ACM's Special Interest Group on CS Education.

for programming assignments you can copy freely from anything on the "sample programs page".)

- I worked with *names of other students* on this assignment.
- I got help with this assignment from *source of help — ACM tutoring, another student in the course, the instructor, etc. (Here, "help" means significant help, beyond a little assistance with tools or compiler errors.)*
- I got help from *outside source — a book other than the textbook (give title and author), a Web site (give its URL), etc.. (Here too, you only need to mention significant help — you don't need to tell me that you looked up an error message on the Web, but if you found an algorithm or a code sketch, tell me about that.)*
- I provided help to *names of students* on this assignment. *(And here too, you only need to tell me about significant help.)*

4 Essay

Include a brief essay (a sentence or two is fine, though you can write as much as you like) telling me what if anything you think you learned from the assignment, and what if anything you found found interesting, difficult, or otherwise noteworthy. For programming assignments, it should go in the body of the e-mail or in a plain-text file `essay.txt` (no word-processor files please).