CMSI 587

Operating Systems (Graduate Level)

Spring 2006

Assignment 0314

With no class on March 7 due to spring break, we have a deeper assignment than usual, with a mix of reading, thinking, and coding, in order to close to overall subject of process management.

Not for Submission

1. Read Chapters 6 and 7 in SGG; if you would like supplementary notes on Section 6.9 and some of Chapter 7, see my "Concurrency Control" handout for CMSI 486.
2. Read the various "classic" papers on concurrent programming and synchronization. A useful exercise is to look at how those papers express their ideas, and to see how those ideas have been translated to the text and other more recent material, such as Wikipedia, documents on current operating systems and APIs, and others.

For Submission 1: Exercises

Do the following exercises from SGG; submit your responses in hardcopy only. In case your edition is different from the 7th, I've added a note on the type of question involved; if you can't find the equivalent in your edition, then let me know and I can pass a copy to you.

1. SGG Exercise 6.1 — Prove that Dekker's critical-section solution for two processes (the first known correct software solution, ca. 1965) satisfies all three requirements for the critical-section problem.
2. SGG Exercise 6.17 — Show how the monitor implementation in the text would change if the *signal()* statement can only appear as the last statement in a monitor procedure (i.e., it behaves like a conventional *return* statement, in addition to doing its other activities).
3. SGG Exercise 7.2 — Discuss how the four necessary conditions for deadlock are present in the case of dining philosophers where philosophers obtain chopsticks one at a time, and how deadlocks can be avoided by eliminating any one of the four conditions.

For Submission 2: Programming

The bounded buffer sample code, as well as API specifications for the problems (*dp-api.txt* and *sb-api.txt*), are on the course Web site.

1. Implement a solution to the dining philosophers problem using the POSIX API. Use the given bounded buffer solution as a pattern, and conform to the API in *dp-api.txt*. Also, make sure to provide well-placed *print* statements to report the work done by the program and the state of things at any given time (as some of you know, I typically don't like this approach, but we do it here because critical-section solutions are hard to implement within a traditional unit test fixture).
2. Implement a solution to the sleeping barber problem (SGG Exercise 6.11) using the POSIX API, also following the pattern in the bounded buffer sample while using the API in *sb-api.txt*. As with dining philosophers, use *print* statements to confirm correct behavior.

Commit your work to your Keck lab CVS repository under the paths *homework/cmsi587/hw0314/dp* and *homework/cmsi587/hw0314/sb* respectively. Place source code in a *src* subdirectory in those locations; if you have any other files, such as notes, documentation, configuration, or build files, commit those at the top-level directories. As usual, make sure that your *.cvs* subdirectory (the physical location of your Keck CVS repository) is accessible to users other than yourself.

Extra Credit

There's quite a bit on your plate now, so if you get this far, this truly is extra credit: implement a *graphical*, thread-safe version of dining philosophers using Java and Swing. Commit your code to *homework/cmsi587/hw0314/dp-gui*.