Lab 2: The dining philosophers

Magnus Johansson

April 22, 2005

1 The assignment

You are to write an implementation of the dining philosophers problem. See page 209 (6th edition) or 207 (7th edition) in the textbook for a description of the problem. Your program should take the number of philosophers as an argument and it should be deadlock free. See section 8.4 (6th edition) or 7.4 (7th edition) in the textbook for more information on how to prevent deadlocks. As before you are free to write it in any programming language you wish, as long as you can do POSIX system calls from it. The assignment has been prepared for Java so you will get some code for free if you choose to do it in Java.

The main point of this assignment is to make you comfortable with semaphores. You should read up on semaphores in the textbook before doing the lab. There are two different interfaces to semaphores in the POSIX standard: POSIX 1003.1b semaphores and SystemV semaphores. SystemV is the older of the two, and very difficult to learn. 1003.1b has a cleaner interface. The 1003.1b interface is the natural choice for learning semaphores. However, since the jtux interface is broken for 1003.1b semaphores, this lab will use SystemV semaphores. To compensate for the very difficult interface of SystemV, there is a simplified interface included in the lab package that is similar to, but not identical to the 1003.1b interface. See Semaphore.java for details. It contains the normal create, destroy, wait, and signal operations.

If you choose to do the lab in C you should use the 1003.1b interface to make your life a bit easier. See the man page for `sem_wait` and `sem_init`, among others.

The lab package contains four java files:

- `Semaphore.java` contains the simplified interface to SystemV semaphores.
- `Rice.java` contains the abstraction for the rice bowl that the philosophers will eat from. You will probably not have to use this yourself, since the skeleton already does that.
- `Philosopher.java` contains the code that a single philosopher will run. It is here the philosopher will pick up the chopsticks, eat, and put them down.
- `Philosophers.java` contains the main program that will create the semaphores and the philosophers and set things up. The philosophers will be children to this process.

For this lab you will need to modify Philosophers.java and Philosopher.java.
Here's a suggested implementation order:

1. Familiarize yourself with the given skeleton. Read through all the files and make sure you understand the general idea. Then start by modifying Philosophers.java. You should create the necessary semaphores and fork children to run the philosopher process. Don't forget to wait for the children and to destroy the semaphores afterwards. The source code contains more information. Take a look at the first lab for more information on forking and waiting.

   Run the program by typing `./philosophers.sh 5` if you want five philosophers. If you run your program at this point the philosophers will eat, but they will do so without using the chopsticks. When the program is running, take some time to look at the semaphores from the command prompt. The command to use is `ipcs`, which stands for Inter Process Communication Status. Semaphores are a part of IPC system. You should see your semaphores there, including one for the rice bowl. If your semaphores remain after your program ends you are doing something wrong. Make sure you destroy your semaphores when you are done with them. You can manually clean

up any remaining semaphores with the command `ipcrm`. See the man pages for ipcs and ipcrm for details (type `man ipcs` and `man ipcrm`, respectively).

2. Modify Philosopher.java. In this file you need to grab and put down the chopsticks. See the file for details. When you are done with this step you should have a deadlock free implementation of the dining philosophers.

Some hints:

- Make your program very verbal. It it much easier to see what is going on if your program does a lot of output.
- Slow down your program to encourage deadlocks and to make it easier to see what happens. There is a line in the skeleton that shows how to pause for a random amount of time.

This is the first time this particular lab is used. I want your feedback!!! Please tell me what's good and what's not.

2 How to hand in

Send an email to `magnusj@it.uu.se` with the source code attached.