Washington State University Vancouver

Systems Programming - CS 360

Assignment 6 - Due: 11:09AM March 25

Instructor:
Ben McCamish

March 4, 2019

Overall Assignment - 100 points

Write a program (in C) called `philo.c` targeted at the Linux platform. This program must fork 5 child processes, each representing one philosopher. Using shared memory and/or semaphores, have each philosopher repeat a cycle of eating and thinking until all philosophers have eaten for at least 100 seconds (at which point a philosopher will "leave the table"... terminate). Use the IPC system calls described in the lecture slides.

There are 5 chopsticks at the table (one between each pair of adjacent philosophers). In order to eat, a philosopher must acquire both the chopsticks adjacent to them. If they cannot, they are made to wait until they can. The wait time does not count as thinking time. Once they have a pair of chopsticks, they eat for the expected period of time and then replace (release) the chopsticks onto the table and begin their period of thinking. Each philosopher repeats this process until they have eaten for a total of 100 seconds or more.

There is a routine named "randomGaussian" available to you on the website that you will use to simulate the random amounts of time that a philosopher eats and thinks. The mean eating and thinking times shall be 9 and 11 seconds, respectively, with standard deviations of 3 and 7 seconds, respectively. Obviously, if randomGaussian returns a negative time value, treat it as zero. Simulate the eating or thinking time by calling sleep() for the requisite number of seconds. You should embed `randomGaussian()` into your code.

You can either represent the chopsticks in a shared IPC memory segment and use a semaphore to "authorize" mutually exclusive access to the shared memory segment, or you can define an array of IPC semaphores, each representing one chopstick.

You can execute a implementation of this program in `/remotehomes/b.mccamish/philo/philo` (use -d on the command line to turn on more output). You do not need to match its output exactly, but print enough information about what the philosophers are doing to demonstrate your implementation is functioning correctly. Note, that does not prevent you from installing as much debugging output as you think you need.

Specifications and Restrictions

- (Required) Makefile containing at least 3 rules:
  1. `all:` (compiles everything together and produces an executable)
  2. `clean:` (removes all object and temporary files)
  3. `run:` (command for running your executable that works with the submitted code)
- (80 points) Program must work on the lab machines, including the specifications above.
- (20 points) Must be robust, including error catching. You must catch errors and print out an appropriate error message containing the `errno` and the message produced by that error. This means you will need to use `errno.h` and `string.h`, libraries at least.
- (Note) If you make use of shared memory, use the `shmget()` and `shmat()` system calls. `shmctl()` can be used to remove the shared memory segment you create.
- (Note) To create and manipulate semaphores, use `semget()` and `semop()`. `semctl()` is used to remove a semaphore set.
- (Required) Clean up the IPC artifacts you create so that you don't leave shared memory and semaphore IDs cluttering up the lab systems.

What to turn in (in a zip on Blackboard):

- philo.c (no header files)
- Makefile