CS39002 Operating Systems Laboratory
Spring 2014

Assignment 5
Due on 26–Mar–2014, 1:00pm

In this assignment, you implement the *dining philosophers' problem*, and demonstrate how deadlocks can be dealt with or completely avoided. Assume that there is a round table with five dining philosophers, each having a plate of spaghetti in front of him. There is also a fork between any two adjacent philosophers. Each philosopher thinks for a random amount of time and then feels hungry. In order to feed, a hungry philosopher requires the forks on two sides of him. If he can grab those, he eats for some time, and then leaves back the forks from where he has taken those, and restarts thinking. This may lead to deadlock (think about the situation when all the philosophers have grabbed their left forks, and are waiting for their right neighbors to release the right forks). If this deadlock happens, some philosopher(s) need to be preempted (that is, forced to return its/their left forks) so that all the philosophers do not starve indefinitely. Another option is to avoid deadlocks altogether. In this assignment, you implement both these versions. In each case, a process forks five child processes which simulate the behavior of the philosophers. Each child process runs an infinite loop of thinking and feeding. Generate random periods for thinking and feeding, and simulate these operations by appropriate sleep calls.

Part 1

In this part, you write a C code to implement the deadlock-free version as taught in the class. More precisely, each philosopher attempts to grab both the forks (left and right) *simultaneously*. If it can, it eats. If not, it waits. This means that if a philosopher sees that only one of the forks is available, he does not grab it. Use semaphores to synchronize the think-and-eat loops run by the five philosophers.

Part 2

In this part, each philosopher grabs the two forks one by one – first the left fork, and then after some waiting the right fork. This version may lead to deadlocks. The parent process checks at regular intervals whether a deadlock has occurred. If so, it chooses a philosopher randomly and releases the fork (the left one actually) grabbed by him. This allows the left philosopher to get hold of his right fork and complete eating. Maintain a resource graph using shared memory. The parent process periodically checks for a deadlock (cycle) in the shared resource graph. Use semaphores for synchronization and mutual exclusion.

In both the programs, print suitable diagnostic messages, like the following:

> *Philosopher 3 starts thinking*
> *Philosopher 2 starts eating*
> *Philosopher 0 grabs fork 0 (left)*
> *Philosopher 4 ends eating and releases forks 4 (left) and 0 (right)*
> *Parent detects deadlock, going to initiate recovery*
> *Parent preempts Philosopher 1*

Submit two C source files: *nodeadlock.c* and *withdeadlock.c*.