Operating Systems - Monsoon 2021

Sambuddho Chakravarty, Arani Bhattacharya

December 17, 2021

Bonus Assignment (Total points: 20).

Due: (Hard deadline: Jan. 3@2359hrs; no delays accepted)

The dining philosopher problem is a classic problem in deadlock management. The problem can be described briefly as follows. There is a round table around which a total of five philosophers are sitting. There are also a total of five plates in front of the philosophers and five forks, one each beside each plate. A philosopher can only eat if s/he can get two forks lying beside the plate. The philosophers do not have any means of communicating with each other to control access to their forks. A situation where none of the philosophers is able to eat because they are all able to access a single fork (but not both) is called deadlock. The goal is to design a protocol of choosing the forks among the dining philosophers so that no deadlock occurs.

1. Write a C program to solve a modified version of dining philosopher problem involving **five** philosophers, each having a plate and a fork to his/her left and another on the right.
2. Assume that the philosophers need to access only a single fork to eat. In addition to the forks, the philosophers also need have access to any one of the **four** sauce bowls kept in the center of the table and accessible to all. Is a deadlock possible? If so, write a C program that implements the deadlock-free solution. The philosophers could be represented by threads. You need to ensure that your solution is free of deadlocks, i.e. it should be designed such that it avoids/prevents deadlocks.
3. Now assume that the philosophers need both the forks to eat **AND** the sauce bowls to eat. Write another program to implement the deadlock-free solution of this version of the problem.

What to submit/rubric.

1. A fully compiling and functioning program (**with the Makefile**), that deals with the problem of deadlocks that may arise due to synchronization and context switches. To show that the program doesn't deadlock, you run it for long enough (e.g. 10 mins). **[30 points]**. Full compilation and functionally correct program (30 points). Program compiles successfully but has deadlocks (15 points). Program doesn't compile (0 points).
2. A readme describing the logic of the programs with description of the appropriate data structures and synchronization primitives used **[5 points]**.