

Indian Institute of Technology Jodhpur  
Operating Systems Lab (CS330)  
**Assignment 7**

**Dated 19<sup>th</sup> October, 2021**

**Total marks: 30**

In this assignment, you will implement two classical synchronization problems.

1. In this, you will 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 bowl of rice in front of her. 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 her. If she can grab those, she eats for some time, and then leaves back the forks from where she 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 [10 marks]**

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, she does not grab it. Use semaphores to synchronize the think-and-eat loops run by the five philosophers.

**Part 2 [10 marks]**

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 her. This allows the left philosopher to get hold of her 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: *no\_deadlock.c* and *with\_deadlock.c*

2. You have been introduced to First Reader-Writer's problem in the class where a reader process gets higher priority while it places a request to enter the Critical Section, even if another writer process is waiting. In other words, a writer process may starve while waiting to enter the CS just because there is always another reader keeps on coming and keep the CS busy all the time. Give a starvation-free solution to this problem. **[10 marks]**