OS Lab 6

Ridhima Kohli
B19CSE071

How to run files ?

Type

```
gcc with_deadlocks.c -o wd lpthread
./wd

gcc no_deadlocks.c -o nd lpthread
./nd
```

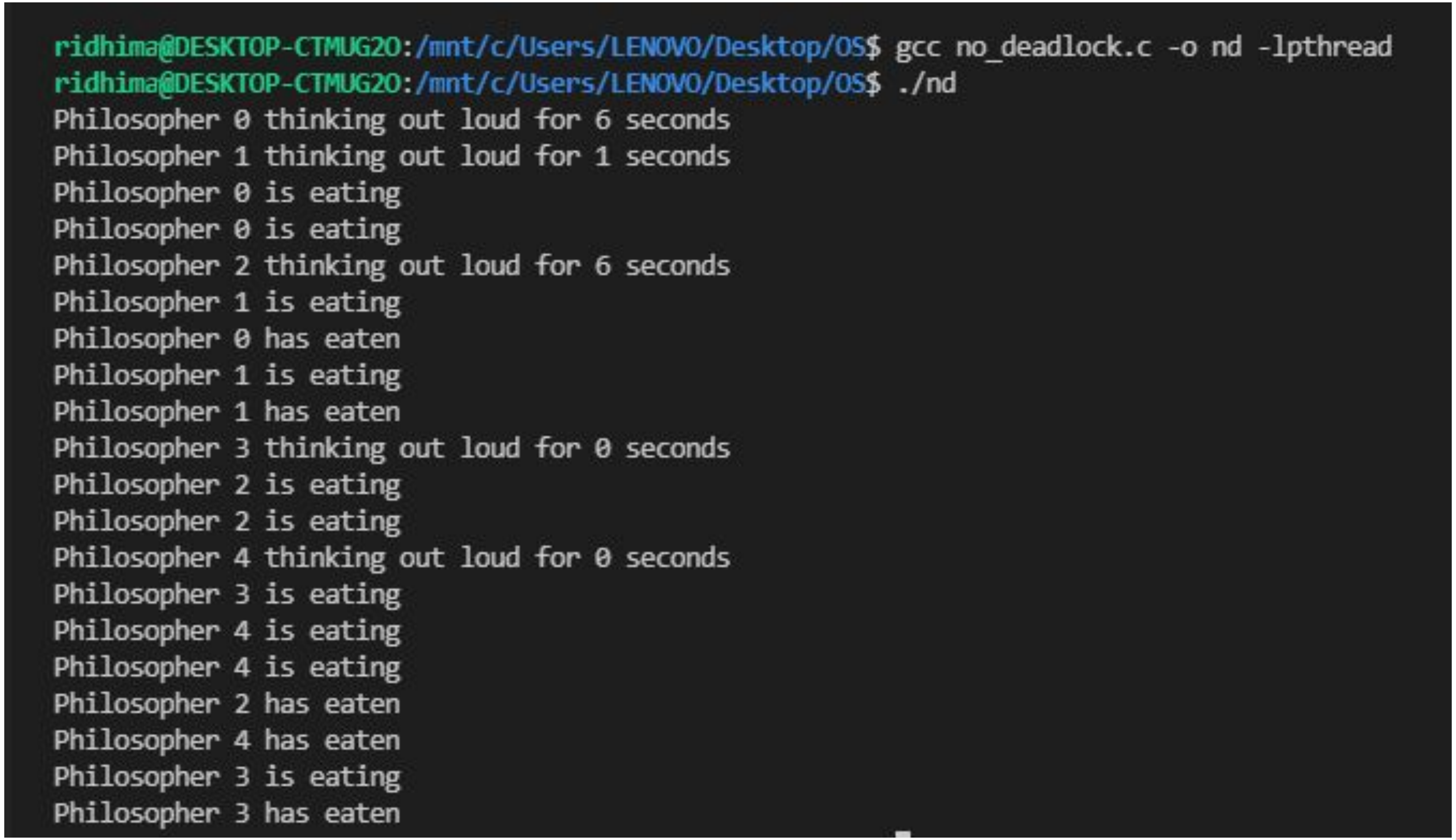
No_deadlocks working

Whenever a philosopher is hungry , a thread is created

Since the threads are independent so it is possible when two philosophers are hungry together and they can eat if their forks are not common easily

Now when a philosopher eats , their two forks are locked by sem_wait() so that if adjacent philosopher gets hungry , they cannot pick the locked forks - this helps in avoiding deadlock

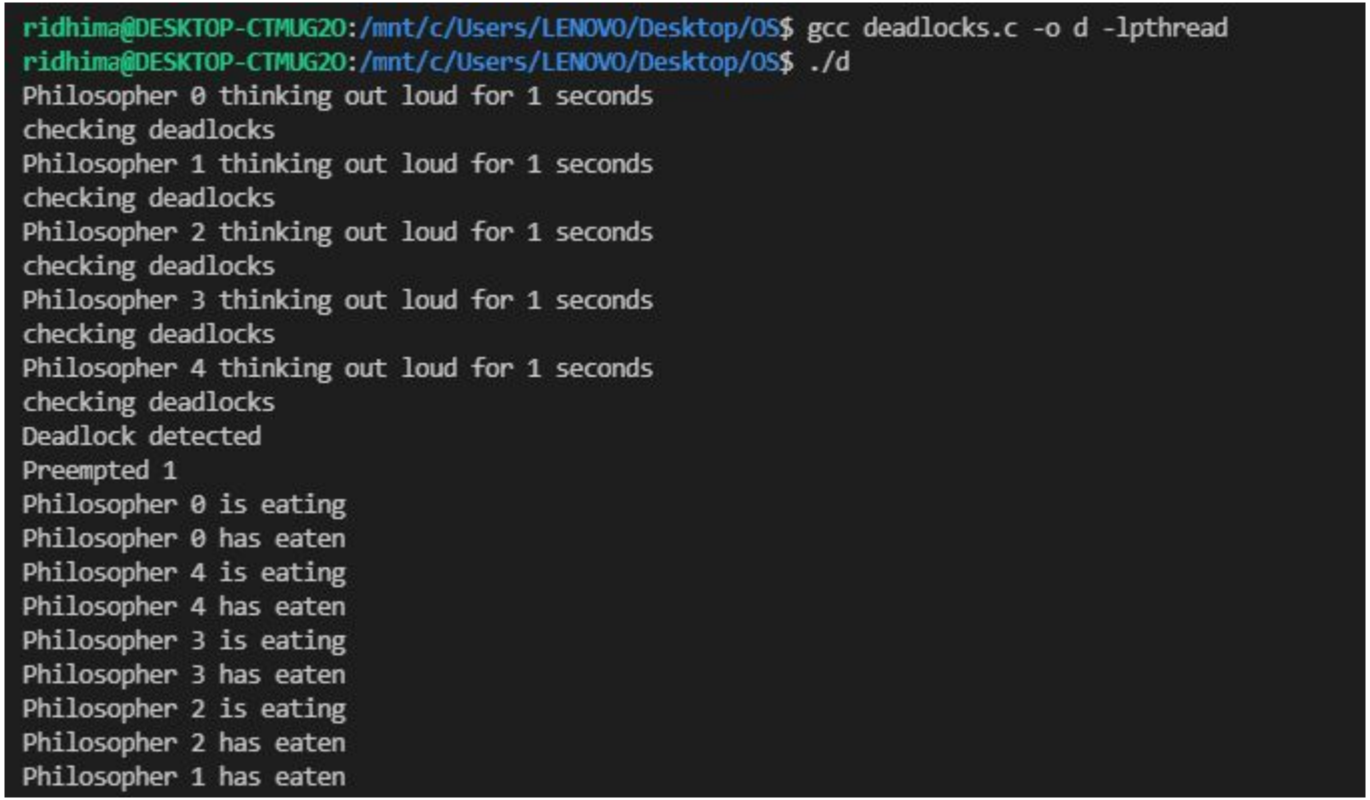
sem_post() unlocks the forks once they are free



```
ridhima@DESKTOP-CTMUG20:/mnt/c/Users/LENOVO/Desktop/OS$ gcc no_deadlock.c -o nd -lpthread
ridhima@DESKTOP-CTMUG20:/mnt/c/Users/LENOVO/Desktop/OS$ ./nd
Philosopher 0 thinking out loud for 6 seconds
Philosopher 1 thinking out loud for 1 seconds
Philosopher 0 is eating
Philosopher 0 is eating
Philosopher 2 thinking out loud for 6 seconds
Philosopher 1 is eating
Philosopher 0 has eaten
Philosopher 1 is eating
Philosopher 1 has eaten
Philosopher 3 thinking out loud for 0 seconds
Philosopher 2 is eating
Philosopher 2 is eating
Philosopher 4 thinking out loud for 0 seconds
Philosopher 3 is eating
Philosopher 4 is eating
Philosopher 4 is eating
Philosopher 2 has eaten
Philosopher 4 has eaten
Philosopher 3 is eating
Philosopher 3 has eaten
```

With deadlocks working

We maintain the array to see which philosopher has picked the left fork , in case of deadlock , its detected and then one philosopher is randomly chosen for preemption
Due to this the remaining philosophers are able to eat and at last the victim phiolospher eats



```
ridhima@DESKTOP-CTMUG20:/mnt/c/Users/LENOVO/Desktop/OS$ gcc deadlocks.c -o d -lpthread
ridhima@DESKTOP-CTMUG20:/mnt/c/Users/LENOVO/Desktop/OS$ ./d
Philosopher 0 thinking out loud for 1 seconds
checking deadlocks
Philosopher 1 thinking out loud for 1 seconds
checking deadlocks
Philosopher 2 thinking out loud for 1 seconds
checking deadlocks
Philosopher 3 thinking out loud for 1 seconds
checking deadlocks
Philosopher 4 thinking out loud for 1 seconds
checking deadlocks
Deadlock detected
Preempted 1
Philosopher 0 is eating
Philosopher 0 has eaten
Philosopher 4 is eating
Philosopher 4 has eaten
Philosopher 3 is eating
Philosopher 3 has eaten
Philosopher 2 is eating
Philosopher 2 has eaten
Philosopher 1 has eaten
```

2. You have been introduced to First Reader-Writer 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.

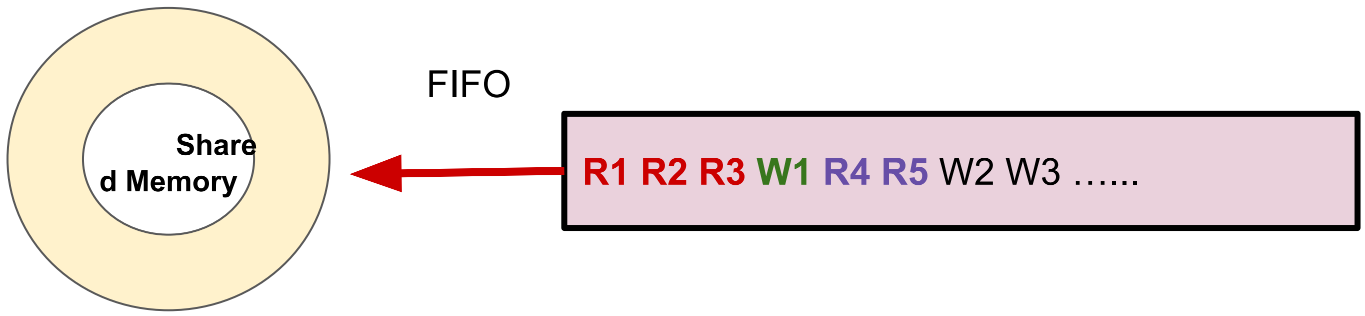
To avoid starvation , we need to make sure that writer request
is addressed and not held for too long or forever
There can be three possible solutions to this problem

- FIFO request address
- Read-Process done based serve
- Time based serve

Answer 2

FIFO

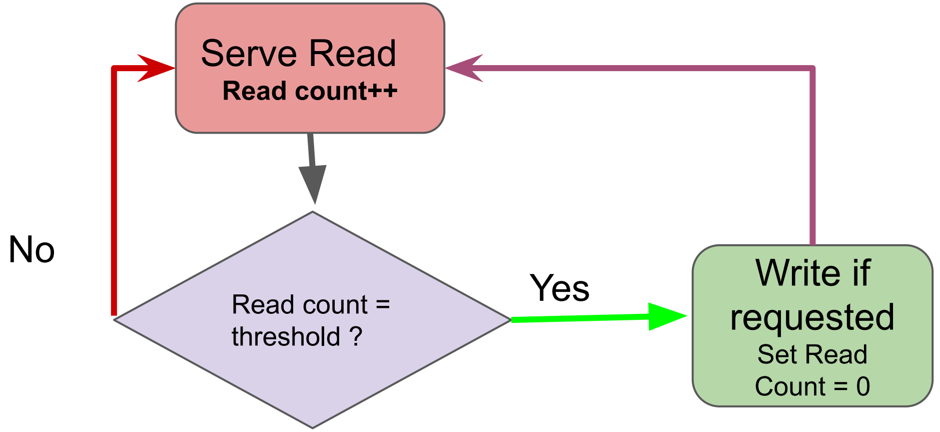
Serve the writer request as they come in the queue after the previous read/write is completed
Readers can work together as they don't cause conflict , when all readers in a session (which comprises continuous readers) are done , if the next is write it can enter



Answer 2

Read-Process done based serve

Keep count of number of reads performed and after a certain threshold , the write in queue is served , at which the counter is set to 0 again



Answer 2

Time based serve

This is similar to previous solution , except that instead of keeping the read count , we can keep count of timestamps between the previous write and current read end time , if it exceeds a threshold then write can be executed

However out of these , FIFO implementation is simpler and better if we go along the logic of execution

Answer 2