CSE231 - Operating Systems
Bonus Assignment

Name: Abhik S Basu
Roll Number: 2020165
Section: A
Branch: CSE

Task: Solving the Producer-Consumer problem in the kernel space

---

I have submitted 2 files namely: dp.c & dp3.c

Approach

In the first question,

The classical dining philosopher problem essentially has the issue of deadlocks. So we solve them by using the concept of semaphores. In our code we will use both counting semaphores as well as binary semaphores.

We use

5 binary semaphores for our 5 forks. Making sure that only one of the two philosophers will ever have access to the fork.
1 counting semaphore for the number of people in our room (in order to avoid all 5 entering in the room and then creating a deadlock by all 5 trying to access the forks to their left. We instantiate it with 4.)

So now that deadlocks have been resolved, we have also made sure that there is no starvation by making sure that sleep is present at various places after eating and entering into thinking states. Further we have also

```
Philosopher 1 is thinking.
Philosopher 2 is thinking.
Philosopher 3 is thinking.
Philosopher 4 is thinking.
Philosopher 5 is thinking.
Philosopher 4 has entered room
Philosopher 4 is eating his meal number 1 with forks 4 and 3
Philosopher 2 has entered room
Philosopher 2 is eating his meal number 1 with forks 2 and 1
Philosopher 3 has entered room
Philosopher 5 has entered room
Philosopher 4 is done with his meal
Philosopher 1 has entered room
Philosopher 2 is done with his meal
Philosopher 3 is eating his meal number 1 with forks 3 and 2
Philosopher 5 is eating his meal number 1 with forks 0 and 4
Philosopher 4 has entered room
Philosopher 3 is done with his meal
Philosopher 1 is eating his meal number 1 with forks 1 and 0
Philosopher 5 is done with his meal
Philosopher 2 has entered room
Philosopher 4 is eating his meal number 2 with forks 4 and 3
Philosopher 3 has entered room
Philosopher 1 is done with his meal
Philosopher 4 is done with his meal
Philosopher 2 is eating his meal number 2 with forks 2 and 1
Philosopher 5 has entered room
Philosopher 5 is eating his meal number 2 with forks 0 and 4
Philosopher 1 has entered room
Philosopher 2 is done with his meal
Philosopher 3 is eating his meal number 2 with forks 3 and 2
Philosopher 5 is done with his meal
Philosopher 4 has entered room
Philosopher 1 is eating his meal number 2 with forks 1 and 0
Philosopher 2 has entered room
```

created a state array to just make sure that starvation does not happen as well as possible deadlocks.

We then create 5 threads and then they work simultaneously at the dining table to get a seat there and eat it. And then we join these threads so as to make the process work.

In useForks() function, we stop the philosopher and make them wait for the sauces and the entry to the room.

In putDownForks() function, we notify that the the fork or room has been released of one philosopher and as result these two functions form the most essential part of our code.

In the second question,

There is no deadlock, since in this case the sauces are central and even one fork needs to be accessed so deadlocks are not possible. So as a result if we apply pigeonhole principle and the worst case analysis, there will still at least always be one philosopher who has had their dinner. So the resources will always get freed up and due to that logic no deadlock is ever created since the program is never truly stopped.

All 5 can sit together and eat.

There will always be one person who has both fork and a sauce, so they will get

In the third question,

There are deadlocks. We can consider two cases for this

1) where each philosopher tries to access the fork to his left so as a result it leads to all 5 accessing one fork and as a result none of them can eat.
2) where one philosopher has two forks, three have one forks and one has no forks. The philosopher's with less than two forks get access to the sauces. So now there is a deadlock created due to the sauces as well.

However our solution of only 4 philosophers to eat at a time will be able to solve this so essentially just

barring the philosophers allowed to eat to 4 is

```
Philosopher 2 is done with his meal
Philosopher 3 is eating his meal number 23 with forks 3 and 2 with sauce
Philosopher 1 has entered room
Philosopher 3 is done with his meal
Philosopher 2 has entered room
Philosopher 4 is eating his meal number 23 with forks 4 and 3 with sauce
Philosopher 4 is done with his meal
Philosopher 3 has entered room
Philosopher 5 is eating his meal number 23 with forks 0 and 4 with sauce
Philosopher 5 is done with his meal
Philosopher 1 is eating his meal number 24 with forks 1 and 0 with sauce
Philosopher 4 has entered room
Philosopher 1 is done with his meal
Philosopher 2 is eating his meal number 24 with forks 2 and 1 with sauce
Philosopher 5 has entered room
Philosopher 2 is done with his meal
Philosopher 3 is eating his meal number 24 with forks 3 and 2 with sauce
Philosopher 1 has entered room
Philosopher 3 is done with his meal
Philosopher 4 is eating his meal number 24 with forks 4 and 3 with sauce
Philosopher 2 has entered room
Philosopher 4 is done with his meal
Philosopher 5 is eating his meal number 24 with forks 0 and 4 with sauce
Philosopher 3 has entered room
Philosopher 5 is done with his meal
Philosopher 1 is eating his meal number 25 with forks 1 and 0 with sauce
Philosopher 4 has entered room
Philosopher 1 is done with his meal
Philosopher 2 is eating his meal number 25 with forks 2 and 1 with sauce
Philosopher 5 has entered room
Philosopher 2 is done with his meal
Philosopher 3 is eating his meal number 25 with forks 3 and 2 with sauce
Philosopher 1 has entered room
Philosopher 3 is done with his meal
Philosopher 4 is eating his meal number 25 with forks 4 and 3 with sauce
```

sufficient to resolve this .

To run the following files we just need to make use of our makefile and run the command `make`

And to run we just write `./dp` or `./dp3`