OS Assignment 4 Gitansh Raj Satija 2019241

For solving the dining philosopher problem we can use semaphore for preventing deadlocks. In my case, I have used the technique where at maximum n-1 philosophers can pick up a fork so that there is no deadlock as at least 1 philosopher can pick up both the forks. Since, additionally each philosopher needs 2 bowls to eat, and there are only two bowls present hence, we implement in such a way that the bowls are treated as 1 unit. We design our my_semaphore in such a way that it contains two elements, 1 pthread_mutex_t for implementing the locks and the other is an integer which acts as signal value. We define 4 primary functions printSignalValue(), wait() , signal(), my_semaphore_init() all of which take the my_semaphore structure as one of the arguments. printSignalValue() function is used for returning the signal value of the semaphore which can be used for debugging. my_semaphore_intit() is used for creation of our own semaphore that we have implemented. Then the wait and signal functions are made to provide the functionality similar to the semaphore structure defined in c. For the signal function we use the blocking system call and increment the value of the signal there. Pthread_mutex_lock and pthread_mutex_unlock functions are used for this process. For the wait signal function, we use the non blocking system call using pthread_mutex_trylock, to return the bowl. Non blocking call is helpful as it is used for releasing the resources. We create 3 semaphores (out of which 1 is a pointer type object used to handle the no. of forks(or philosophers). Hence we have N+2 diff semaphores.try_eat is a counting semaphore. Following is the output attached

```
gitsat@ubuntu:~/Desktop$ gcc -pthread a4.c
gitsat@ubuntu:~/Desktop$ ./a.out
Enter Number of Philosophers:8
Philosopher 7 eats using forks 6 and 7
Philosopher 8 eats using forks 7 and 8
Philosopher 6 eats using forks 5 and 6
Philosopher 5 eats using forks 4 and 5
Philosopher 7 eats using forks 6 and 7
Philosopher 4 eats using forks 3 and 4
Philosopher 6 eats using forks 5 and 6
Philosopher 3 eats using forks 2 and 3
Philosopher 2 eats using forks 1 and 2
Philosopher 1 eats using forks 8 and 1
Philosopher 8 eats using forks 7 and 8
Philosopher 5 eats using forks 4 and 5
Philosopher 7 eats using forks 6 and 7
Philosopher 4 eats using forks 3 and 4
Philosopher 6 eats using forks 5 and 6
Philosopher 3 eats using forks 2 and 3
Philosopher 5 eats using forks 4 and 5
Philosopher 4 eats using forks 3 and 4
Philosopher 2 eats using forks 1 and 2
Philosopher 1 eats using forks 8 and 1
```

As you can see, each philosopher has access to forks in front of him and 1 fork towards his left. Error handling has been done for all possible corner cases and system calls that might lead to some error.