Operating Systems Project - Exercise 2

1. Producer-consumer problem

```
bence@BenceLaptop:~/OPsystems/Exercise 2$ gcc -o producer1 producer1.c -pthread
bence@BenceLaptop:~/OPsystems/Exercise 2$ ./producer1
Producer produced item 83 to 0
Consumer consumed item 83 from 0
Producer produced item 15 to 0
Consumer consumed item 15 from 0
Producer produced item 86 to 0
Producer produced item 49 to 1
Consumer consumed item 49 from 1
Consumer consumed item 86 from 0
Producer produced item 27 to 0
Producer produced item 63 to 1
Producer produced item 40 to 2
Producer produced item 72 to 3
Producer produced item 11 to 4
Consumer consumed item 11 from 4
Producer produced item 29 to 4
Consumer consumed item 29 from 4
Consumer consumed item 72 from 3
Consumer consumed item 40 from 2
Consumer consumed item 63 from 1
Consumer consumed item 27 from 0
bence@BenceLaptop:~/OPsystems/Exercise 2$
```

I made two functions, a producer and a consumer. They have a shared buffer, and both have their own semaphores. I set the producer semaphore value to buffer size and the consumer to zero in the beginning. They both go on until they did 10 actions. When a product is made, producer gets -1 and consumer +1 as to regulate how many products can be made. Each function has its own sem_wait to make sure no products can be made or consumed that would cause an overflow or a product to be taken when there are none. Since the program will do as many producer actions as consumer actions, the buffer will be empty at the end of runtime. I used pthread since each process is in the same file.

2. The dining-philosophers problem

```
bence@BenceLaptop:~/OPsystems/Exercise 2$ ./diningphilosophers
There are 5 philosophers, and the program continues until they ate 7 times.
Philosopher 0 is thinking
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 1 is hungry
Philosopher 1 takes chopsticks 0 and 1
Philosopher 1 is eating
Philosopher 0 is hungry
Philosopher 4 is hungry
Philosopher 4 takes chopsticks 3 and 4
Philosopher 4 is eating
Philosopher 2 is hungry
Philosopher 3 is hungry
Philosopher 1 putting chopsticks 0 and 1 down
Philosopher 1 is thinking
Philosopher 2 takes chopsticks 1 and 2
Philosopher 2 is eating
Philosopher 4 putting chopsticks 3 and 4 down
Philosopher 4 is thinking
Philosopher 0 takes chopsticks 4 and 0
Philosopher 0 is eating
Philosopher 1 is hungry
Philosopher 4 is hungry
Philosopher 2 putting chopsticks 1 and 2 down
Philosopher 2 is thinking
Philosopher 3 takes chopsticks 2 and 3
Philosopher 3 is eating
Philosopher 0 putting chopsticks 4 and 0 down
Philosopher 0 is thinking
Philosopher 1 takes chopsticks 0 and 1
Philosopher 1 is eating
Philosopher 3 putting chopsticks 2 and 3 down
Philosopher 3 is thinking
Philosopher 4 takes chopsticks 3 and 4
Philosopher 4 is eating
Philosophers ate 7 times, stopping the program.
bence@BenceLaptop:~/OPsystems/Exercise 2$
```

I made a solution to the philosophers problem where each philosopher waits until both chopsticks are free before grabbing both. This way, we can ensure that no deadlocks form. The code itself can be modified easily to both have more philosophers and go on until more eatings happen. Each philosopher has 3 states: thinking, hungry and eating. Baseline is thinking, but after a randomized amount of time, the state will change to hungry. When hungry, the philosopher will try to eat and will eat for a randomized amount of time, after which the chopsticks to both of their sides will be released. If we check the code, we can see that it works perfectly, no two forks are used at the same time.