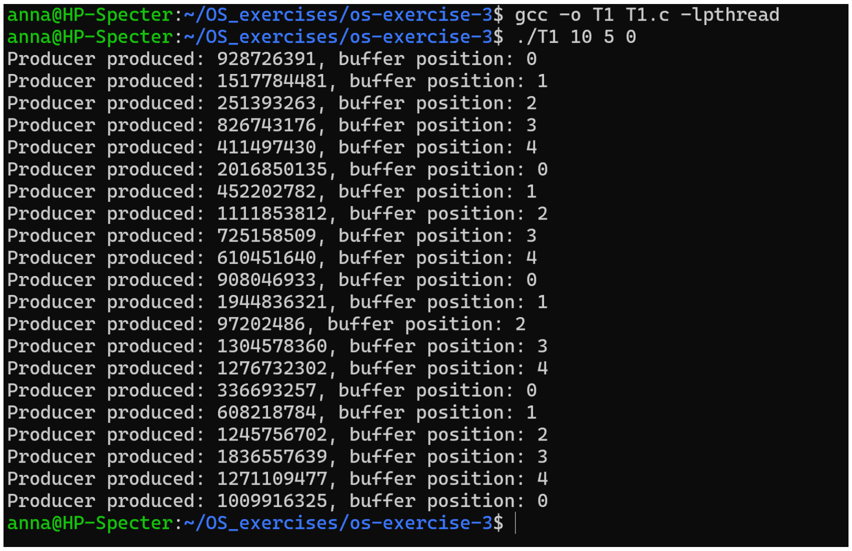
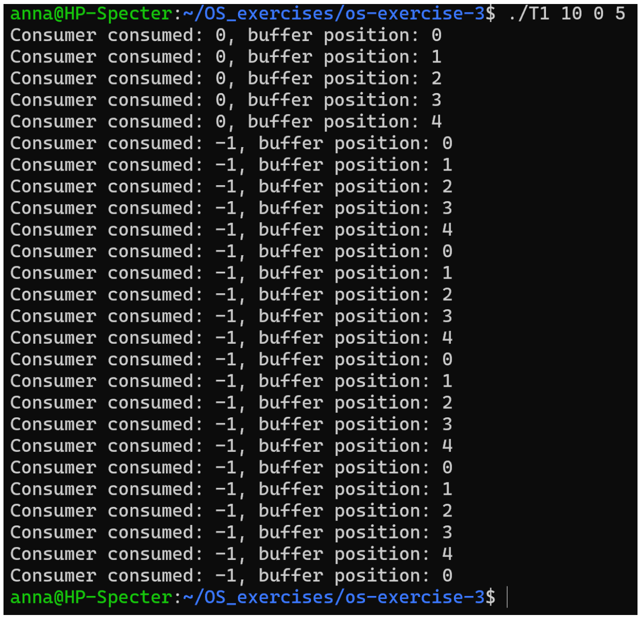
Task 1

When compiling and executing `T1.c` as instructed in task one, the output shown in figure one and two is produced. With 5 producers and no consumer the producers iterates over the buffer and keeps overwritting the existing data with the newly produced one. Similarly with 5 consumers, we iterate over the buffer even though there's nothing to consume.
After filling in the `TODO`s to synchronise the producers and consumers, the producers only add data until the buffer is full and the consumers only read the buffer if there is anything to consume. See figure 3.



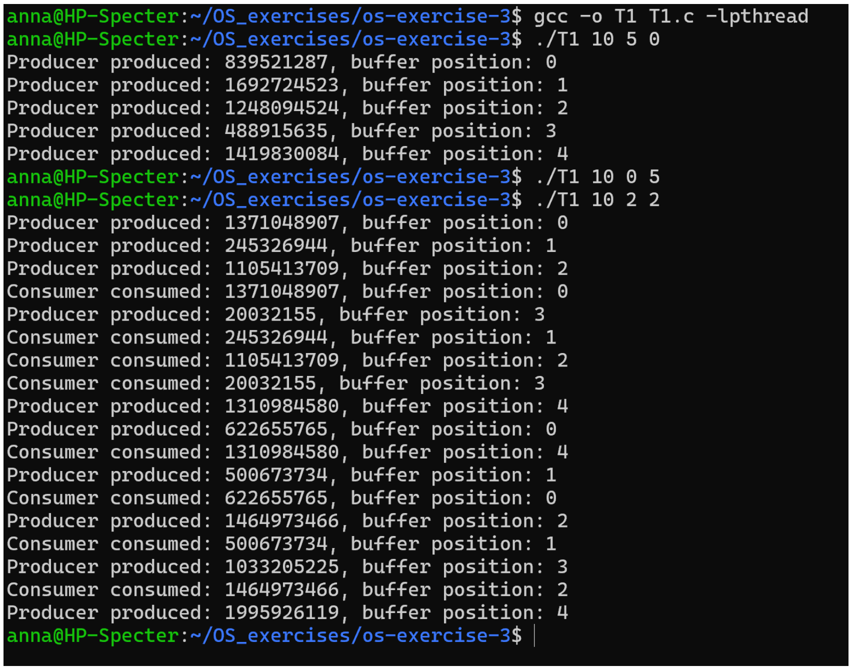
```
anna@HP-Specter:~/OS_exercises/os-exercise-3$ gcc -o T1 T1.c -lpthread
anna@HP-Specter:~/OS_exercises/os-exercise-3$ ./T1 10 5 0
Producer produced: 928726391, buffer position: 0
Producer produced: 1517784481, buffer position: 1
Producer produced: 251393263, buffer position: 2
Producer produced: 826743176, buffer position: 3
Producer produced: 411497430, buffer position: 4
Producer produced: 2016850135, buffer position: 0
Producer produced: 452202782, buffer position: 1
Producer produced: 1111853812, buffer position: 2
Producer produced: 725158509, buffer position: 3
Producer produced: 610451640, buffer position: 4
Producer produced: 908046933, buffer position: 0
Producer produced: 1944836321, buffer position: 1
Producer produced: 97202486, buffer position: 2
Producer produced: 1304578360, buffer position: 3
Producer produced: 1276732302, buffer position: 4
Producer produced: 336693257, buffer position: 0
Producer produced: 608218784, buffer position: 1
Producer produced: 1245756702, buffer position: 2
Producer produced: 1836557639, buffer position: 3
Producer produced: 1271109477, buffer position: 4
Producer produced: 1009916325, buffer position: 0
anna@HP-Specter:~/OS_exercises/os-exercise-3$
```

Figure 1: T1 only producers



```
anna@HP-Specter:~/OS_exercises/os-exercise-3$ ./T1 10 0 5
Consumer consumed: 0, buffer position: 0
Consumer consumed: 0, buffer position: 1
Consumer consumed: 0, buffer position: 2
Consumer consumed: 0, buffer position: 3
Consumer consumed: 0, buffer position: 4
Consumer consumed: -1, buffer position: 0
Consumer consumed: -1, buffer position: 1
Consumer consumed: -1, buffer position: 2
Consumer consumed: -1, buffer position: 3
Consumer consumed: -1, buffer position: 4
Consumer consumed: -1, buffer position: 0
Consumer consumed: -1, buffer position: 1
Consumer consumed: -1, buffer position: 2
Consumer consumed: -1, buffer position: 3
Consumer consumed: -1, buffer position: 4
Consumer consumed: -1, buffer position: 0
Consumer consumed: -1, buffer position: 1
Consumer consumed: -1, buffer position: 2
Consumer consumed: -1, buffer position: 3
Consumer consumed: -1, buffer position: 4
Consumer consumed: -1, buffer position: 0
Consumer consumed: -1, buffer position: 1
Consumer consumed: -1, buffer position: 2
Consumer consumed: -1, buffer position: 3
Consumer consumed: -1, buffer position: 4
Consumer consumed: -1, buffer position: 0
anna@HP-Specter:~/OS_exercises/os-exercise-3$
```

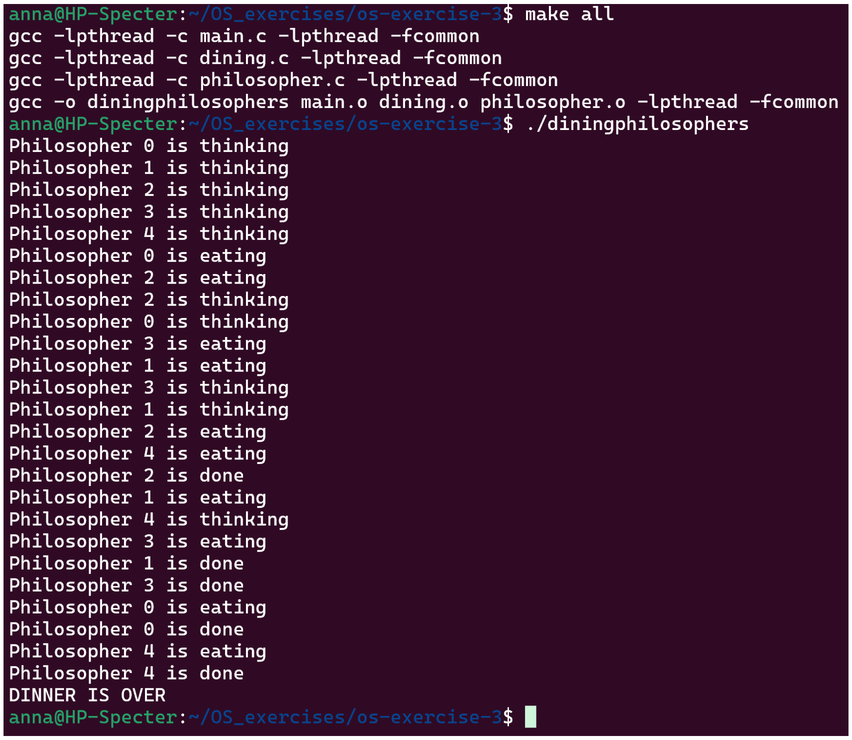
Figure 2: T1 only consumers



```
anna@HP-Specter:~/OS_exercises/os-exercise-3$ gcc -o T1 T1.c -lpthread
anna@HP-Specter:~/OS_exercises/os-exercise-3$ ./T1 10 5 0
Producer produced: 839521287, buffer position: 0
Producer produced: 1692724523, buffer position: 1
Producer produced: 1248094524, buffer position: 2
Producer produced: 488915635, buffer position: 3
Producer produced: 1419830084, buffer position: 4
anna@HP-Specter:~/OS_exercises/os-exercise-3$ ./T1 10 0 5
anna@HP-Specter:~/OS_exercises/os-exercise-3$ ./T1 10 2 2
Producer produced: 1371048907, buffer position: 0
Producer produced: 245326944, buffer position: 1
Producer produced: 1105413709, buffer position: 2
Consumer consumed: 1371048907, buffer position: 0
Producer produced: 20032155, buffer position: 3
Consumer consumed: 245326944, buffer position: 1
Consumer consumed: 1105413709, buffer position: 2
Consumer consumed: 20032155, buffer position: 3
Producer produced: 1310984580, buffer position: 4
Producer produced: 622655765, buffer position: 0
Consumer consumed: 1310984580, buffer position: 4
Producer produced: 500673734, buffer position: 1
Consumer consumed: 622655765, buffer position: 0
Producer produced: 1464973466, buffer position: 2
Consumer consumed: 500673734, buffer position: 1
Producer produced: 1033205225, buffer position: 3
Consumer consumed: 1464973466, buffer position: 2
Producer produced: 1995926119, buffer position: 4
anna@HP-Specter:~/OS_exercises/os-exercise-3$
```

Figure 3: T1 synchronised

Task 2



```
anna@HP-Specter:~/OS_exercises/os-exercise-3$ make all
gcc -lpthread -c main.c -lpthread -fcommon
gcc -lpthread -c dining.c -lpthread -fcommon
gcc -lpthread -c philosopher.c -lpthread -fcommon
gcc -o diningphilosophers main.o dining.o philosopher.o -lpthread -fcommon
anna@HP-Specter:~/OS_exercises/os-exercise-3$ ./diningphilosophers
Philosopher 0 is thinking
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 0 is eating
Philosopher 2 is eating
Philosopher 2 is thinking
Philosopher 0 is thinking
Philosopher 3 is eating
Philosopher 1 is eating
Philosopher 3 is thinking
Philosopher 1 is thinking
Philosopher 2 is eating
Philosopher 4 is eating
Philosopher 2 is done
Philosopher 1 is eating
Philosopher 4 is thinking
Philosopher 3 is eating
Philosopher 1 is done
Philosopher 3 is done
Philosopher 0 is eating
Philosopher 0 is done
Philosopher 4 is eating
Philosopher 4 is done
DINNER IS OVER
anna@HP-Specter:~/OS_exercises/os-exercise-3$
```

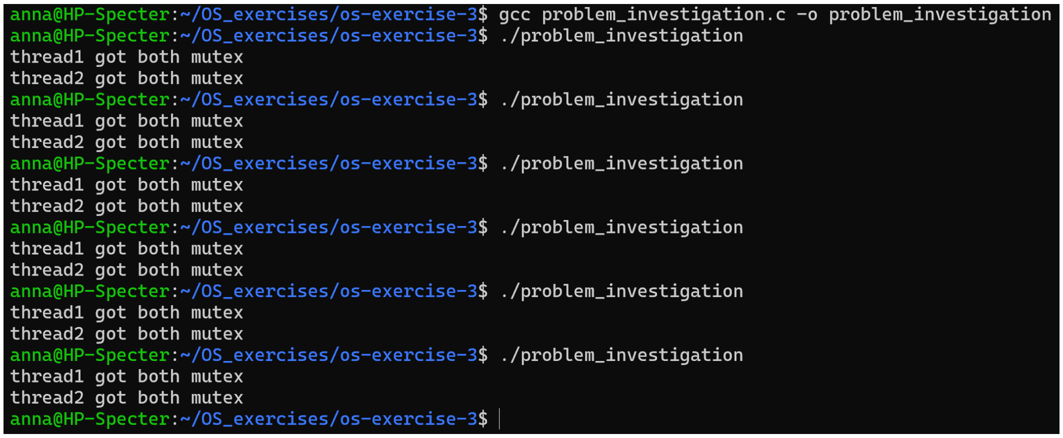
Figure 4: dinig philosphers

Task 3

After adding a main method and the thread infrastructure around the two given methods, we can execute two threads with one of the methods each. Running them a few times did not pose any problems. However when analyzing the code staticly, it appears that there is a possibility to run into a Livelock. This cann occur if both were to start execution at the exact same time:

- thread 1 locks mutex 1 and thread 2 locks mutex 2
- thread 1 one tries to get mutex 2, thread 2 tries to get mutex 2
- thread 1 fails and unlocks mutex 1, thread two fails and unlocks mutex 2
- now both need to lock their first mutex again before they can try to get the second, which, by the time they try to get it, will be locked again already

A possible solution to this, is for both threads to wait a random amount of time, before aquiring any mutex again, giving the other thread more time to get both mutexes and execute their task.



```
anna@HP-Specter:~/OS_exercises/os-exercise-3$ gcc problem_investigation.c -o problem_investigation
anna@HP-Specter:~/OS_exercises/os-exercise-3$ ./problem_investigation
thread1 got both mutex
thread2 got both mutex
anna@HP-Specter:~/OS_exercises/os-exercise-3$ ./problem_investigation
thread1 got both mutex
thread2 got both mutex
anna@HP-Specter:~/OS_exercises/os-exercise-3$ ./problem_investigation
thread1 got both mutex
thread2 got both mutex
anna@HP-Specter:~/OS_exercises/os-exercise-3$ ./problem_investigation
thread1 got both mutex
thread2 got both mutex
anna@HP-Specter:~/OS_exercises/os-exercise-3$ ./problem_investigation
thread1 got both mutex
thread2 got both mutex
anna@HP-Specter:~/OS_exercises/os-exercise-3$ ./problem_investigation
thread1 got both mutex
thread2 got both mutex
anna@HP-Specter:~/OS_exercises/os-exercise-3$ 
```

Figure 5: T3 output

Task 4

- Bounded buffer in Producer / Consumer
- Readers and writer problem
- dining phylosopher Problem

Task 5

A process is in a deadlock if it cannot continue it's execution. It cannot get required ressources and will not be able to make progress.
A process is starving if it is never executed or is indefinitly waiting to be executed. Processes can starve due to a deadlock or due to it having a low priority and never being sheduled for execution.

Task 6