Operating Systems Project 2 by Cecilia Muniz Siqueira – Graduate Student

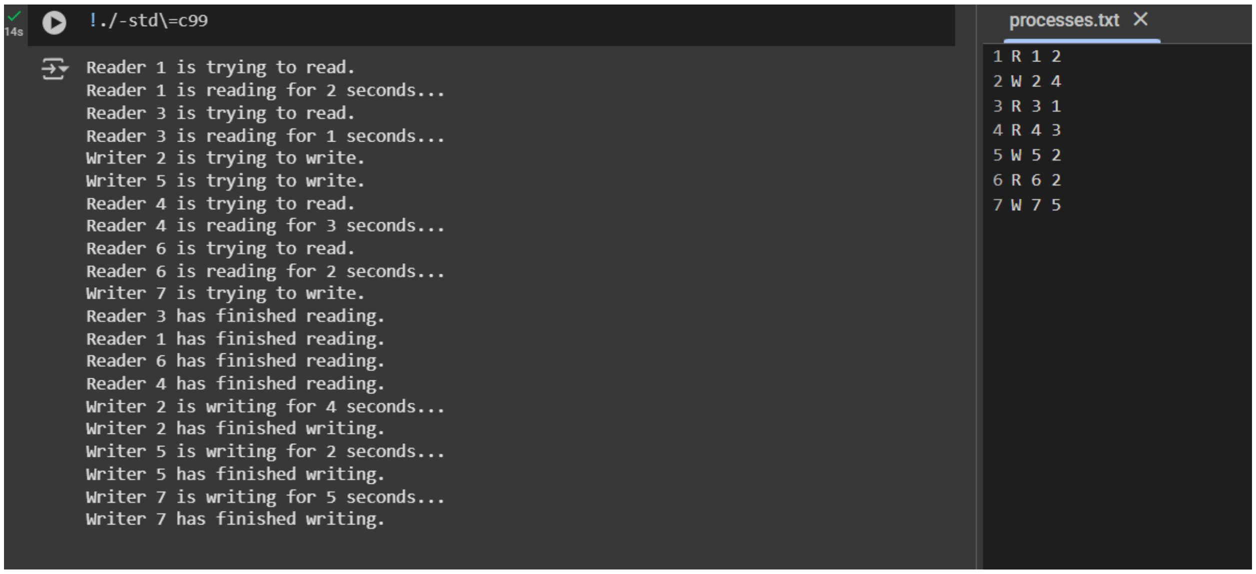
Github link: https://github.com/ceciliamuniz/OS_Project2

1) Which problem I solved

For this project, I chose to solve the Readers-Writers problem, where multiple readers can read simultaneously, and writers need exclusive access. Solving it correctly involves managing thread synchronization to avoid issues like mutual exclusion, deadlocks, or starvation. In my C code, I used POSIX threads (by including pthread.h) for concurrency and semaphores (by including semaphore.h) for synchronization. I tracked the number of active readers with a counter, and used two semaphores: rx_mutex, to control access to shared resources, and mutex, to control access to the read_count variable, managing mutual exclusion.

2) Sample test case and results

Due to multithreading, the output order may vary with each run. Here are two different results, using the same processes.txt file as input:

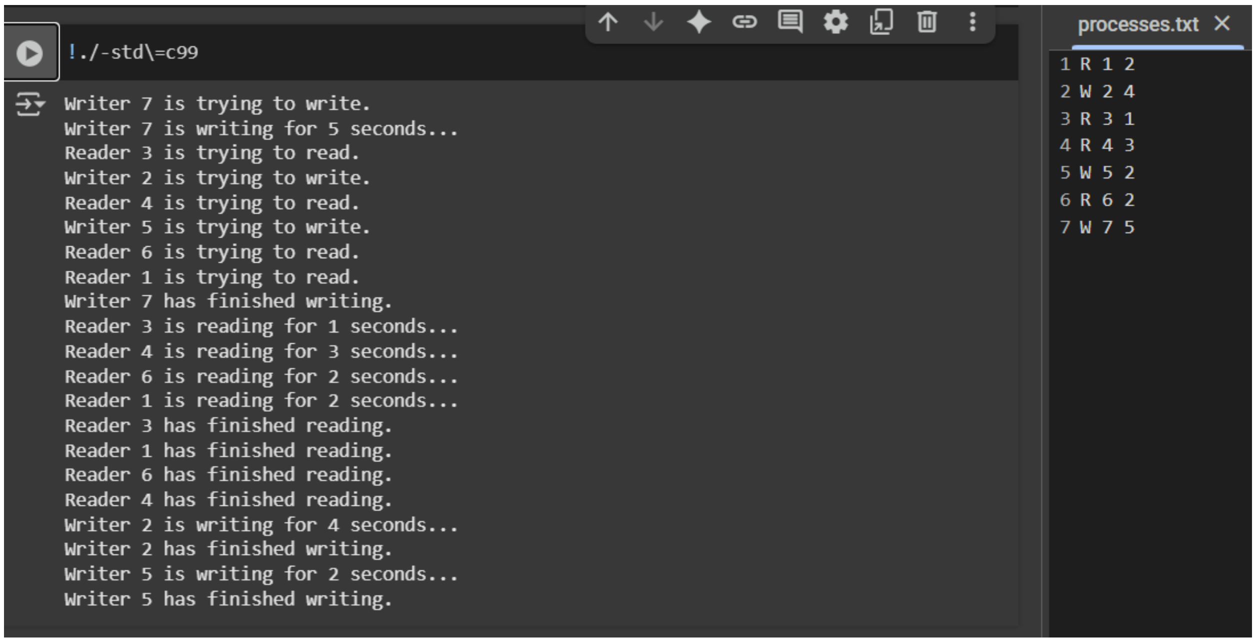


```
!./-std\=c99
```

```
Reader 1 is trying to read.
Reader 1 is reading for 2 seconds...
Reader 3 is trying to read.
Reader 3 is reading for 1 seconds...
Writer 2 is trying to write.
Writer 5 is trying to write.
Reader 4 is trying to read.
Reader 4 is reading for 3 seconds...
Reader 6 is trying to read.
Reader 6 is reading for 2 seconds...
Writer 7 is trying to write.
Reader 3 has finished reading.
Reader 1 has finished reading.
Reader 6 has finished reading.
Reader 4 has finished reading.
Writer 2 is writing for 4 seconds...
Writer 2 has finished writing.
Writer 5 is writing for 2 seconds...
Writer 5 has finished writing.
Writer 7 is writing for 5 seconds...
Writer 7 has finished writing.
```

```
R 1 2
W 2 4
R 3 1
R 4 3
W 5 2
R 6 2
W 7 5
```

Here, Readers 1, 3, 4, and 6 start reading almost simultaneously or with slight overlaps, showing that multiple readers can access the resource at the same time. Writers 2, 5, and 7 try to write but they have to wait until all active readers finish. Once the last reader finishes reading, Writer 2 begins writing, then Writer 5, then Writer 7, each with exclusive access in their turn. No readers or writers overlap during a writer's execution, showing proper synchronization.



```
!./-std\=c99
```

```
Writer 7 is trying to write.
Writer 7 is writing for 5 seconds...
Reader 3 is trying to read.
Writer 2 is trying to write.
Reader 4 is trying to read.
Writer 5 is trying to write.
Reader 6 is trying to read.
Reader 1 is trying to read.
Writer 7 has finished writing.
Reader 3 is reading for 1 seconds...
Reader 4 is reading for 3 seconds...
Reader 6 is reading for 2 seconds...
Reader 1 is reading for 2 seconds...
Reader 3 has finished reading.
Reader 1 has finished reading.
Reader 6 has finished reading.
Reader 4 has finished reading.
Writer 2 is writing for 4 seconds...
Writer 2 has finished writing.
Writer 5 is writing for 2 seconds...
Writer 5 has finished writing.
```

```
1 R 1 2
2 W 2 4
3 R 3 1
4 R 4 3
5 W 5 2
6 R 6 2
7 W 7 5
```

Here, Writer 7 starts and finishes writing before any reader gets access. After Writer 7 finishes, Readers 3, 4, 6, and 1 begin reading concurrently. Writers 2 and 5 must wait until all readers finish, then each gets a writing turn.

3) Any challenges you faced

This project was actually much easier to me than project 1. I didn't face any significant challenges, but I was a bit confused on how the processes.txt file from Project 1 was supposed to be used in this project, until I figured out that yes, I had to completely modify it as the new project requires an input in a completely different format: type (Reader/Writer), PID, and duration.