MTL458 - Operating Systems
Assignment 4: Producer-Consumer and Reader-Writer Locks

October 2024

Objective

This assignment contains two parts:

1. Implement the Single Producer-Single Consumer problem using mutex locks and condition variables in C.
2. Implement reader-writer locks (both reader-preference and writer-preference versions).

Part 1: Producer-Consumer Problem

Problem Description

You are required to simulate a producer-consumer scenario with the following details:

- The shared resource is a queue implemented using **circular buffer** of size 100, holding unsigned integer values.
- A **producer** thread reads integers continuously from an input file (`input-part1.txt`) and adds them to the buffer.
- A **consumer** thread consumes items from the buffer.
- The producer must **wait** if the buffer is full.
- The consumer must **wait** if the buffer is empty.
- Use **mutex locks** and **condition variables**

Input

Read integers continuously from **input-part1.txt**. The input file contains one integer per line, on encountering 0 terminate the producer thread. **There will always be a 0 present at the end of the input file**.

```
input-part1.txt
3
5
2
6
0
```

Output

Write the output to `output-part1.txt`. Each time an item is consumed by the consumer thread, print the consumed integer and the current state of the buffer. The output format is as follows:

```
Consumed:[Element-consumed],Buffer-State:[first-inserted-element in buffer
                                to last-inserted - comma separated]<newline>
```

No space is present
E.g.

```
Consumed:[3],Buffer_State:[5,2]
Consumed:[5],Buffer_State:[2,6]
Consumed:[2],Buffer_State:[6]
Consumed:[6],Buffer_State:[]
```

Testing

Command to compile and execute :

```
gcc prod-cons.c -o prod-cons -lpthread
./prod-cons
```

Part 2: Reader-Writer Locks

Problem Description

In this part, you are required to implement two versions of reader-writer locks:

1. **Reader-preference lock**: Readers are allowed to access the data even when a writer is waiting. This might lead to writer starvation.
2. **Writer-preference lock**: Once a writer is waiting, no new readers are allowed to acquire the reader lock, ensuring writers do not starve.

Task Requirements

- Implement the reader-preference version in `rwlock-reader-pref.c`.
- Implement the writer-preference version in `rwlock-writer-pref.c`.
- Use semaphores

Testing

Command to compile and run

```
gcc rwlock-reader-pref.c -o reader-pref -lpthread
./reader-pref number_of_reader number_of_writer
```

E.g. ./reader-pref 5 2 should spawn 5 reader threads followed by 2 writer threads.

```
gcc rwlock-writer-pref.c -o writer-pref -lpthread
./writer-pref number_of_reader number_of_writer
```

Output

You should use **shared-file.txt** for reading/writing.
Print the given output in **output-reader-pref.txt** and **output-writer-pref.txt** files.

When reading starts print in respective output-files:

`Reading,Number-of-readers-present:[count]<newline>`

Then read the shared-file.txt

When writing starts print in respective output-files:

`Writing,Number-of-readers-present:[count]<newline>`

And append "Hello world!" without quotes in the shared-file.txt on a new line at the end of the file.

Notice there are no spaces in the output

E.g. On command "./reader-pref 2 1"

```
Reading,Number-of-readers-present:[1]
Reading,Number-of-readers-present:[2]
Writing,Number-of-readers-present:[0]
```

Final Submission

Submit a zip file **Entry-Number.zip** which contains exactly three files

- `prod-cons.c`
- `rwlock-reader-pref.c`
- `rwlock-writer-pref.c`
- Follow the output format exactly as mentioned. It will autograded any deviations or printing extra lines will result in a 0 marks without any futher considerations.
- Make sure, your file names, functions signatures, output format should be as it is mentioned in the assignemnt without any deviation.