Operating Systems - Lab Assignment 2

Deadline: Nov. 25, 2023

1 Threads

In the first part of the assignment, you will write a program that creates three threads. These threads access a shared integer, `buffer`, one at a time. The `buffer` will initially be set to 0. Each thread should print its thread ID, process ID, and the `buffer`'s current value in one statement, then increment the `buffer` by one. Use a mutex to ensure this whole process is not interrupted. Have the threads modify the `buffer` 15 times. When each thread is done, it should return the number of times it changed the `buffer` to the main thread. The output would be as follows:

```
$ ./lab2.out
TID: 3077897072, PID: 30656, Buffer: 0
TID: 3069504368, PID: 30656, Buffer: 1
TID: 3059014512, PID: 30656, Buffer: 2
TID: 3077897072, PID: 30656, Buffer: 3
TID: 3069504368, PID: 30656, Buffer: 4
TID: 3077897072, PID: 30656, Buffer: 5
TID: 3059014512, PID: 30656, Buffer: 6
TID: 3069504368, PID: 30656, Buffer: 7
TID: 3077897072, PID: 30656, Buffer: 8
TID: 3059014512, PID: 30656, Buffer: 9
TID: 3069504368, PID: 30656, Buffer: 10
TID: 3077897072, PID: 30656, Buffer: 11
TID: 3069504368, PID: 30656, Buffer: 12
TID: 3059014512, PID: 30656, Buffer: 13
TID: 3069504368, PID: 30656, Buffer: 14
TID 3077897072 worked on the buffer 5 times
TID 3069504368 worked on the buffer 6 times
TID 3059014512 worked on the buffer 4 times
Total buffer accesses: 15
```

2 Reader-Writer problem

In the second part, you will implement the *Reader-Writer problem*. Here, you have two *readers* who read from a shared buffer and one *writer* who updates the buffer. The buffer contains a variable `VAR`, initialized to zero. After each access, the writer reads the content of the buffer, prints it along with its own PID, and increments the content by one. Each reader also periodically reads the content of the buffer and prints the current value along with its own PID. The writer and reader should be implemented as separate processes. Use shared memory to implement the shared buffer and use semaphore(s) to ensure proper synchronization between the reader and writer processes. The program terminates when the writer writes `MAX` to `VAR`.

Notice that both readers can access the shared content together. The reader who first makes a read operation should lock the shared content for reading. When the shared content is locked for reading, the other reader can read it. The last reader, done with reading, releases the lock. The writer can write only when no reader is reading the shared content. Finally, before the writer finishes writing, no reader can read.

You can insert appropriate delays, e.g., `sleep()`, to make the messages visible. Here is a sample run for the writer with PID 123, and the readers with PIDs 124 and 125:

```
The first reader acquires the lock.
The reader (124) reads the value 0
The last reader releases the lock.
The writer acquires the lock.
The writer (123) writes the value 1
The writer releases the lock.
The first reader acquires the lock.
Reader (125) reads the value 1
Reader (124) reads the value 1
The last reader releases the lock.
```