Operating Systems

Lab 03 Exercises – semaphores and conditions

Exercise 1

Write a concurrent C program that allocates two buffers of **16** entries, **normal** and **urgent**, which will be used as circular buffers for communications between two threads: **producer** and **consumer**.

The main thread creates these two threads, and waits for their termination.

The **producer** thread loops **100** times. At every cycle it:

sleeps between **1** and **10** milliseconds

- gets in **ms** the current time in milliseconds (use the function **current_timestamp**)
- randomly selects the **normal** or **urgent** queue with probability **p** and **1-p**, respectively, **p** is given as an argument of the command line
- prints the information including **ms**, and **0** or **1**, respectively if the selected buffer is **normal** or **urgent**
- puts in the selected buffer these values.

The **consumer** thread loops **100** times

- sleeping **10** milliseconds
- waiting that the producer produces at least an information
- getting **an information from the urgent buffer, unless it is empty**, otherwise it gets the information from the **normal** buffer
- printing the information retrieved.

```
long long current_timestamp() {
    struct timeval te;
    gettimeofday(&te, NULL); // get current time
    long long milliseconds = te.tv_sec*1000LL + te.tv_usec/1000;
    return milliseconds;
}
```

Exercise 2

Write a concurrent C program **using only conditions and mutexes**, that implement the **Readers & Writers** protocol, with precedence to the Readers.
In particular, the main threads creates **N** readers and **N** writers (**N** given as an argument of the command line) and waits for their termination.
Each reader/writer

- sleeps for a random time of millisecond (**0-500**)
- **printf("Thread %d trying to read/write at time %d\n", …)**
- when it is able to read, it
  printf("Thread %d reading/writing at time %d1n", …)
- simulates the reading/writing operation sleeping **500** milliseconds
- terminates