CS 314 Operating Systems Lab - Exam

Max Marks -30 Time -1.5 Hrs [2.15PM to 3.45PM]

Instructions:

1. **Please create a folder named as your rollno 18XXXX in the desktop**
2. **Download the skeleton code `master-worker-skeleton.c` from the moodle and rename it as `master-worker.c` and place this file in the folder 18XXXX created in the desktop and implement your solution in `master-worker.c`**
3. **Zip this folder 18XXXX and upload it in the moodle**

You will implement a simple master and worker thread pool, a pattern that occurs in many real life applications. The master threads produce numbers continuously and place them in a buffer, and worker threads will consume them, i.e., print these numbers to the screen. This simple program is an example of a master-worker thread pool pattern that is commonly found in several real-life application servers. For example, in the commonly used multi-threaded architecture for web servers,the server has one or more master threads and a pool of worker threads. When a new connection arrives from a web client, the master accepts the request and hands it over to one of the workers. The worker then reads the web request from the network socket and writes a response back to the client. Your simple master-worker program is similar in structure to such applications, but with much simpler request processing logic at the application layer.

You are given a skeleton program `master-worker-skeleton.c`. This program takes 4 command line arguments: **how many numbers to "produce" (M), the maximum size of the buffer in which the produced numbers should be stored (N), the number of worker threads to consume these numbers(C), and the number of master threads to produce numbers (P).** The skeleton code spawns **P** master threads, that produce the specified number of integers from **0** to **M−1** into a shared buffer array. The main program waits for these threads to join, and then terminates. The skeleton code given to you does not have any logic for synchronization.

You must write your solution in the file `master-worker.c`**.** You must modify this skeleton code in the following ways. You must add code to spawn the required number of worker threads, and write the function to be run by these threads. This function will remove/consume items from the shared buffer and print them to screen. Further, you must add logic to correctly synchronize the producer and consumer threads in such a way that every number is produced and consumed exactly once. Further, producers must not try to produce when the buffer is full, and consumers should not consume from an empty buffer. While you need to ensure that all **C** workers are involved in consuming the integers, it is not necessary to ensure perfect load balancing between the workers. Once all **M** integers (from 0 to M−1) have been produced and consumed, all threads must exit. The main thread must call `pthread_join` on the master and worker threads, and terminate itself once all threads have joined. Your solution must only use `pthreads` condition variables for waiting and signaling: busy waiting is not allowed

Your code can be compiled as shown below.

```
gcc master-worker.c -lpthread
```

If your code is written correctly, every integer from 0 to M−1 will be produced exactly once by the master producer thread, and consumed exactly once by the worker consumer threads. We have provided you with a simple testing script (`test-master-worker.sh` which invokes the script `check.awk`) that checks this above invariant on the output produced by your program. The script relies on the two print functions that must be invoked by the producer and consumer threads: the master thread must call the function `print_produced` when it produces an integer into the buffer, and the worker threads must call the function `print_consumed` when it removes an integer from the buffer to consume. You must invoke these functions suitably in your solution. Please do not modify these print functions, as their output will be parsed by the testing script.