CS3103: Operating Systems (Spring 2019)

Programming Assignment 2
P2: A Simulator for FIFO Queue

1 Goals

This assignment is designed to help you:

1. get familiar with multi-threaded programming with Pthread.
2. get familiar with critical section with Pthread mutex.

You are required to implement your solution in C++ (other languages are not allowed). Your work will be tested on the Linux server `cs3103-01.cs.cityu.edu.hk`.

A good tutorial on pthread could be found at https://computing.llnl.gov/tutorials/pthreads/.

2 Requirements

2.1 Background

First-In-First-Out (FIFO) queue has been broadly used in computer and network systems. In this assignment, you are required to simulate a simple case of FIFO queue. The model includes a `flow`, a `queue`, and a `server`. The `flow` is responsible for generating traffic, measured in terms of `token`, into the `queue`. The `server` tries to empty the `queue` by fetching a certain number of `tokens`, if any.

In the simple FIFO queueing model, the behavior of `flow`, `queue`, and `server` is summarized as follows.

- The `flow` wakes up periodically with an interval time of `flowInterval` seconds, where `flowInterval` is an input parameter. Once waking up, the `flow` generates a random number of tokens, uniformly distributed in $[1, 10]$, and inserts the tokens to the `queue`. Each token is assigned a unique sequence number in the increasing order. We assume that the first token has sequence number 0, and the sequence number will be increased by 1 for each consecutive token. We also assume that the `flow`, after it wakes up, remembers the history of tokens so that the sequence numbers of generated tokens are always increasing.
- The `queue` temporarily stores the token in the FIFO fashion. The maximum size of the queue is set to 50 tokens. If the `queue` is full, the incoming tokens will be dropped.

- The server wakes up periodically with an interval time of 2 seconds. Once waking up, the server fetches a random number of tokens, uniformly distributed in $[1, 20]$. **Note** that if the generated random number is larger than the current number of tokens in the queue, all the tokens in the queue will be fetched.

The simulation stops after MaxToken number of tokens have been served, where MaxToken is an input parameter. A token is **considered to be served if it is fetched from the queue by the server or it is dropped due to buffer overflow.**

2.2 Implementation Requirements

Your program, named by SFIFO, should consists of three threads: the main thread, the flow thread, and the server thread. The main thread is responsible for creating the flow thread and the server thread, and then waits for these threads to return. The flow thread simulates the behavior of the flow. The server thread simulates the behavior of the server.

Your code **must** print out useful information, and the output **must** be formatted as the following table:

```
Flow                                  Queue          Server
Token added  Latest sequence number  Current Length  Token fetched  Total Token fetched

aaa          bbb                     ccc
                                     zzz            xxx            yyy
```

where the header of the table (i.e., Flow, Queue, Server, Token added, Last sequence number, Current Length, Token served, Total Token served) is printed by the main thread before creating the other two threads. The numbers aaa, bbb, and ccc are printed by the flow thread when it wakes up. The numbers xxx, yyy, and zzz are printed by the server thread when it wakes up.

In addition, the number aaa is **not** the accumulative number but the number of tokens at the current round of flow wake-up; the number bbb is the highest sequence number at the current round of flow wake-up; the number ccc is the queue length after the tokens generated at the current round of flow wake-up have been added into the queue. The number xxx is **not** the accumulative number but the number of tokens that will be fetched by the server at the current round of its wake-up; the number yyy is the total number of tokens that have been fetched by the server; the number zzz is the current queue length after the tokens have been fetched by the server at the current round of server wake-up.

Note that at the end of the program, the total number of tokens that have been served is equal to the total number of tokens that have been fetched by the server and the total number of tokens that have been dropped by the queue. As such, **at the end of the program**, you must print out the total number of tokens have been fetched by the server thread, the total number of tokens that have been generated by the flow thread, and the total number of tokens that have been dropped by the queue. The output should be:

```
The total number of tokens that have been fetched by the server is eee.
The total number of tokens that have been generated by the flow is fff.
The total number of tokens that have been dropped by the queue is ggg.
```

where eee, fff, and ggg indicate the corresponding numbers.

Because the simulation stops after MaxToken number of tokens have been served, the sum of eee and ggg must equal MaxToken. Nevertheless, fff could be larger than MaxToken.

The queue is implemented as a data structure, rather than a thread. You need to implement **your own `queue` data structure and its related functions**, which means that **you cannot use the `queue` data structure provided by the C/C++ library**.

Since both the flow thread and the server thread need to access the queue data structure, **the code segment for accessing the queue is critical section and must be protected with the pthread mutex lock.**

2.3 Prompt for user input

Your `SFIFO` needs to accept two integer inputs, `MaxToken` as the first and `flowInterval` as the second, e.g.,

```
$ ./SFIFO 500 2
```

2.4 Code Quality

We cannot specify completely the coding style that we would like to see but it includes the following:

1. Proper decomposition of a program into subroutines (and multiple source code files when necessary)—A 500 line program as a single routine won't suffice.
2. Comment—judiciously, but not profusely. Comments also serve to help a marker, in addition to yourself. To further elaborate:
   (a) Your favorite quote from Star Wars or Douglas Adams' Hitch-hiker's Guide to the Galaxy does not count as comments. In fact, they simply count as anti-comments, and will result in a loss of marks.
   (b) Comment your code in English. It is the official language of this university.
3. Proper variable names—`leia` is not a good variable name, it never was and never will be.
4. Small number of global variables, if any. Most programs need a very small number of global variables, if any. (If you have a global variable named `temp`, think again.)
5. **The return values from all function calls should be checked and all values should be dealt with appropriately.**

3 Marking

Your program will be tested on our CSLab Linux servers (cs3103-01, cs3103-02, cs3103-03).

You should tell TA how to compile and run your code in your `readme` text file.

TAs are not supposed to fix the bugs, either in your source code or in your `Makefile`.

If an executable file cannot be generated and running successfully on our Linux servers, it will be considered as unsuccessful.

Any program does not print the output as required in the section 2.2 will be considered as unsuccessful.

If the program can be run successfully and output is in the correct form, your program will be graded as the following marking scheme.

Table 1: Marking scheme.

Components	Weight
`Create threads`	20%
`Thread join`	10%
`Lock and unlock mutex`	20%
`Exit thread`	10%
`Queue structure & related functions`	20%
`Error handling`	10%
`programming style and in-program comments`	10%

4 Submission

1. This assignment is to be done individually or by a group of two students. You are encouraged to discuss the high-level design of your solution with your classmates but you must implement the program on your own. Academic dishonesty such as copying another student's work or allowing another student to copy your work, is regarded as a serious academic offence.

2. Each submission consists of three files: a source program file (.cpp file), a readme (.txt) file telling us how to compile your code, a text file containing the possible outputs produced by your program (.txt file), and a `Makefile` if applicable.

3. Use your student ID(s) to name your submitted files, such as 5xxxxxxx.cpp, 5xxxxxxx-readme.txt, 5xxxxxxx.txt for individual submission, or 5xxxxxxx_5yyyyyyy.cpp, 5xxxxxxx_5yyyyyyy-readme.txt, 5xxxxxxx_5yyyyyyyy.txt for group submission. Only ONE submission is required for each group.

4. Submit the files to Canvas.

5. The deadline is 10:00 a.m., 14-MAR (Thu). No late submission will be accepted.

Question?
Contact Miss ZHANG, Yifan at `yif.zhang@my.cityu.edu.hk` or your course lecturer.

The End
