EE442 HOMEWORK 1

Bank Branch

Due: May, 2, 2021, 23:55

Kamil SERT – EA405 ksert@metu.edu.tr

Submission

- Send your homework compressed in an archive file with name "**eXXXXXXX_ee442_hw1.tar.gz**", where X's are your **7-digit student ID number**. You will **not** get full credit if you fail to submit your project folder as required.
- Your work will be graded on its correctness, efficiency, clarity and readability as a whole.
- You should insert comments in your source code at appropriate places without including any unnecessary detail.
- Late submissions are welcome, but are penalized according to the following policy:
  - 1 day late submission : HW will be evaluated out of 70.
  - 2 days late submission : HW will be evaluated out of 50.
  - 3 days late submission : HW will be evaluated out of 30.
  - Later submissions : HW will NOT be evaluated.
- The homework must be written in **C** (**not** in C++ or any other language).
- You should **not** call any external programs in your code.
- **Check** what you upload. Do not send corrupted, wrong files or unnecessary files.
- The homework is to be prepared **individually**. Group work is **not** allowed. Your code will be **checked** for cheating.
- The design should be your original work. However, if you partially make use of a code from the Web, give proper **reference** to the related website in your comments. Uncited use is unacceptable.
- Comments will be graded.
- **METU honor code is essential**. Do **not** share your code. Any kind of involvement in cheating will result in a **zero** grade, for **both** providers and receivers.

Introduction

In this homework, you are asked to simulate a bank branch which has multiple pay desks and associated queues. Bank clients will arrive and enter a pay desk's queue. When they are at the front, they will spend some time to do their banking transaction and leave. When they leave, the desk will inform the supervisor about the payment.

Each desk will be represented by a thread. The clients will be generated by the main thread and the supervisor will also be represented by a thread.

In order to change the parameters of the program, command-line options should be used.

Bank Clients

Clients are represented by the following struct:

```
struct BankClient {
      int id; // Client ID
      double duration; // banking transaction duration
}
```

The main thread will generate C clients in total (C is selected with -c option). Between each generation, the main thread should sleep for a random amount of time with an exponential distribution (the rate is selected with -g option standing for generation time). After each generation, the main thread will write to terminal about the client as follows:

```
Client <id> arrived.
```

The client will be inserted to the rear of queue that has the smallest size. If all queues are full, the client will be discarded (they leave). Then, the main thread will sleep and generate as usual.

Each client id must be one larger than the one before. For each client, duration is also random with an exponential distribution (the rate is selected with -d option).

From a continuous uniform distribution *x* between 0 and 1, the exponential distribution can be generated as follows:

$$f(y,\lambda) = -\frac{1}{\lambda}\ln(1-x)$$

where λ is the rate.

Pay Desks

There should be N desk threads and associated queues with maximum size Q (N is selected with -n option and Q is selected with -q option). If the queue is empty, the desk should be **blocked** until a client is inserted to its queue. Each desk thread will sleep according to the duration of the client at the front of the queue and then remove the client from its queue. After the removal, it will update the information represented by the following struct:

```
struct Information {
      int paydeskNo;
      struct BankClient client;
}
```

There must be a single Information variable in the program. All pay desk threads will update this variable.

Supervisor Thread

Every time Information variable is updated, this thread will write to the terminal the following:

```
Desk <paydeskNo> served Client <id> in <duration> seconds.
```

If the information is not updated, the supervisor should be **blocked**.

Queue Structure

Queue is an abstract data type in which the elements are inserted to the rear and are removed from the front (first in, first out). In this homework, it will be implemented as a dynamically allocated array. The array will be used circularly. The following is an example struct:

```
struct ClientQueue {
      client * array;
      int maxSize;
      int currentSize;
      int frontIndex;
}
```

If necessary, you can add additional fields.

Command-line arguments

The program should use five optional arguments to change parameters:

- -c: The total number of clients to be generated (default 20)
- -n: Number of pay desk threads (default 4)
- -q: The maximum size of the queues (default 3)
- -g: The rate of generation time (default 100)
- -d: The rate of duration time (default 10)

Instead of parsing the command-line arguments directly, you may want to use `getopt()`.

Pthread Mutex

An example code to show how pthread mutexes are used for thread synchronization is given below.

```
int count = 0;                /* shared count variable */
pthread_mutex_t mutex;        /* pthread mutex */

int main()
{
      …………
      …………  /* main code */
      …………
}


/* Each thread executes this function. */
void * countFunction(void *arg)
{
      int i;
      for (i = 0; i < 5; i++)
      {
            /* Enter critical section. Acquire mutex lock. */
            Pthread_mutex_lock(&mutex);

            count++;

            /* Exit critical section. Release mutex lock. */
            Pthread_mutex_unlock(&mutex);
      }
      return NULL;
}
```

Hints

- If you have main.c, queue.c and queue.h as source files, you can compile your code with this command:
  ```
  gcc -o office main.c queue.c -pthread
  ```
  If you have only main.c as a source file, you can compile your code with this command:
  ```
  gcc -o office main.c -pthread
  ```
- Some libraries need to be linked explicitly. One example is math.h library where gcc needs -lm option.

Remarks

- There should be **no deadlock or starvation.**
- Synchronization variables should be used **only for critical sections.**
- There should be **no memory leak.**
- You can find information about headers, functions and types here.
- Send only the source code (`queue.h, queue.c, main.c etc.`). Do not send any executable, since your code will be recompiled.

Examples

```
ee442@ee442-VirtualBox: ~/HW1$ ./BankBranch
NUM_CLIENTS       : 20
NUM_DESKS         : 4
QUEUE_SIZE        : 3
DURATION_RATE     : 10.000000
GENERATION_RATE   : 100.000000
Client 0 arrived.
Desk 0 served Client 0 in 0.013490 seconds.
Client 1 arrived.
Client 2 arrived.
Client 3 arrived.
Client 4 arrived.
Desk 2 served Client 3 in 0.062300 seconds.
Client 5 arrived.
Desk 3 served Client 4 in 0.110054 seconds.
Desk 1 served Client 1 in 0.081176 seconds.
Client 6 arrived.
Client 7 arrived.
Client 8 arrived.
Client 9 arrived.
Client 10 arrived.
Client 11 arrived.
Desk 0 served Client 2 in 0.130477 seconds.
Desk 0 served Client 8 in 0.066255 seconds.
Desk 3 served Client 5 in 0.083009 seconds.
Client 12 arrived.
Client 13 arrived.
Client 14 arrived.
Client 15 arrived.
Client 16 arrived.
Client 17 arrived.
Desk 1 served Client 6 in 0.100233 seconds.
Desk 2 served Client 9 in 0.044218 seconds.
Desk 2 served Client 11 in 0.053327 seconds.
Client 18 arrived.
Client 19 arrived.
Desk 0 served Client 18 in 0.091102 seconds.
Desk 3 served Client 13 in 0.063328 seconds.
Desk 1 served Client 14 in 0.015590 seconds.
Desk 0 served Client 7 in 0.048643 seconds.
Desk 2 served Client 10 in 0.103365 seconds.
Desk 1 served Client 12 in 0.029044 seconds.
Desk 3 served Client 16 in 0.077326 seconds.
Desk 1 served Client 15 in 0.086235 seconds.
Desk 2 served Client 19 in 0.065512 seconds.
Desk 0 served Client 17 in 0.030133 seconds.
```

```
ee442@ee442-VirtualBox: ~/HW1$ ./BankBranch -c 7 -n 2 -q 2 -g 25 -d 5
NUM_CLIENTS        : 7
NUM_DESKS          : 2
QUEUE_SIZE         : 2
DURATION_RATE      : 5.000000
GENERATION_RATE    : 25.000000
Client 0 arrived.
Desk 0 served Client 0 in 0.065107 seconds.
Client 1 arrived.
Client 2 arrived.
Client 3 arrived.
Desk 0 served Client 2 in 0.016590 seconds.
Client 4 arrived.
Desk 1 served Client 1 in 0.047872 seconds.
Client 5 arrived.
Desk 1 served Client 5 in 0.022164 seconds.
Desk 0 served Client 3 in 0.029742 seconds.
Client 6 arrived.
Desk 1 served Client 4 in 0.013199 seconds.
Desk 0 served Client 6 in 0.080323 seconds.
```