EE442 HOMEWORK 1

Tax Office

Due: April 1, 2019, 23:55

Aycan Doğa Hakyemez **ARC-201** **hdoga@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.
- 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.
- **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 tax office which has multiple pay desks and associated queues. Tax payers will arrive and enter a desk's queue. When they are at the front, they will spend some time to pay their tax and leave. When they leave, the desk will inform the supervisor about the payment.

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

Command-line options should be used for changing the parameters of the program.

Tax Payers

Tax payers are represented by the following struct:

```
struct TaxPayer {
    int id;                  // tax payer ID
    double duration;         // payment duration
}
```

The main thread will generate `P` tax payers in total (`P` is selected with `-p` 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 tax payer as follows:

```
TaxPayer <id> arrived
```

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

Each tax payer id must be one larger than the one before. For each tax payer, 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 tax payer is inserted to its queue. Each desk thread will sleep according to the duration of the tax payer at the front of the queue and then remove the tax payer from its queue. After the removal, it will update the information represented by the following struct:

```
struct Information {
    int deskNo;
    struct TaxPayer payer;
}
```

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

Supervisor Thread

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

```
Desk <deskNo> served Payer <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 PayerQueue {
    Payer * array;
    int maxSize;
    int currentSize;
    int frontIndex;
}
```

If necessary, you can add additional fields.

Command-line arguments

The program should use four optional arguments to change parameters:

- `-p`: The total number of payers to be generated (default 20)
- `-n`: Number of 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().

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 (`main.c`, `queue.h`, `queue.c`, etc.). Do not send any executable, since your code will be recompiled.

Examples

```
ee442@ee442-VirtualBox:~/HW1$ ./office
NUM_PAYERS      : 20
NUM_DESKS       : 4
QUEUE_SIZE      : 3
DURATION_RATE   : 10.000000
GENERATION_RATE : 100.000000
Payer 0 arrived
Desk 0 served Payer 0 in 0.010919 seconds.
Payer 1 arrived
Payer 2 arrived
Payer 3 arrived
Payer 4 arrived
Payer 5 arrived
Payer 6 arrived
Desk 3 served Payer 4 in 0.028705 seconds.
Payer 7 arrived
Payer 8 arrived
Payer 9 arrived
Payer 10 arrived
Desk 1 served Payer 2 in 0.099867 seconds.
Payer 11 arrived
Payer 12 arrived
Payer 13 arrived
Payer 14 arrived
Payer 15 arrived
Payer 16 arrived
Desk 1 served Payer 6 in 0.065589 seconds.
Desk 2 served Payer 3 in 0.163759 seconds.
Desk 0 served Payer 1 in 0.183701 seconds.
Payer 17 arrived
Payer 18 arrived
Payer 19 arrived
Desk 2 served Payer 8 in 0.031432 seconds.
Desk 2 served Payer 13 in 0.037671 seconds.
Desk 0 served Payer 5 in 0.089310 seconds.
Desk 3 served Payer 7 in 0.217869 seconds.
Desk 1 served Payer 11 in 0.208634 seconds.
Desk 2 served Payer 19 in 0.136107 seconds.
Desk 0 served Payer 10 in 0.193449 seconds.
Desk 0 served Payer 17 in 0.001639 seconds.
Desk 3 served Payer 9 in 0.369479 seconds.
Desk 1 served Payer 12 in 0.299275 seconds.
Desk 3 served Payer 14 in 0.167177 seconds.
Desk 1 served Payer 18 in 0.212812 seconds.
ee442@ee442-VirtualBox:~/HW1$
```

```
ee442@ee442-VirtualBox:~/HW1$ ./office -p 10 -n 2 -q 2 -g 20 -d 5
NUM_PAYERS      : 10
NUM_DESKS       : 2
QUEUE_SIZE      : 2
DURATION_RATE   : 5.000000
GENERATION_RATE : 20.000000
Payer 0 arrived
Desk 0 served Payer 0 in 0.075079 seconds.
Payer 1 arrived
Payer 2 arrived
Payer 3 arrived
Payer 4 arrived
Payer 5 arrived
Desk 0 served Payer 1 in 0.144110 seconds.
Payer 6 arrived
Desk 1 served Payer 2 in 0.185288 seconds.
Desk 1 served Payer 4 in 0.044715 seconds.
Payer 7 arrived
Payer 8 arrived
Payer 9 arrived
Desk 0 served Payer 3 in 0.332861 seconds.
Desk 1 served Payer 7 in 0.246404 seconds.
Desk 1 served Payer 8 in 0.101669 seconds.
Desk 0 served Payer 6 in 0.253777 seconds.
ee442@ee442-VirtualBox:~/HW1$
```

```
ee442@ee442-VirtualBox:~/HW1$ ./office -p 10
NUM_PAYERS      : 10
NUM_DESKS       : 4
QUEUE_SIZE      : 3
DURATION_RATE   : 10.000000
GENERATION_RATE : 100.000000
Payer 0 arrived
Payer 1 arrived
Payer 2 arrived
Desk 0 served Payer 0 in 0.038118 seconds.
Payer 3 arrived
Payer 4 arrived
Payer 5 arrived
Payer 6 arrived
Payer 7 arrived
Payer 8 arrived
Desk 1 served Payer 1 in 0.065267 seconds.
Payer 9 arrived
Desk 2 served Payer 2 in 0.093954 seconds.
Desk 0 served Payer 3 in 0.072866 seconds.
Desk 1 served Payer 6 in 0.062730 seconds.
Desk 0 served Payer 5 in 0.034868 seconds.
Desk 1 served Payer 9 in 0.053005 seconds.
Desk 3 served Payer 4 in 0.199550 seconds.
Desk 2 served Payer 7 in 0.199562 seconds.
Desk 3 served Payer 8 in 0.155543 seconds.
ee442@ee442-VirtualBox:~/HW1$
```