May 6th, 2021

CSE344 – System Programming - Homework #4 – v1
Initiation to POSIX threads

This is your first homework with POSIX threads, so let's take it easy.

Objective

You will simulate a system where one student hires other students to do her homework in her place. Every actor in this system will be represented by a separate thread. Thread h, reads her homeworks one by one from an input file, and puts them in a queue, along with her priority (speed S, cost C or quality Q). There will also be an arbitrary number of students-for-hire, each of course represented as a separate thread. Each of them will possess 3 parameters: a price per homework, speed per homework and quality per homework. The main thread, depending on the requested priority, will give the task to the student-for-hire thread that is most suitable and available (i.e. not busy with another homework).

How

There will be 2 input files.

The first file, will contain the homeworks of h and her priorities, as one character per homework (either C, S or Q; no whitespace). The number of homeworks is arbitrary (there will be at least one); e.g.

`CSQCSQCSQCSCQSCSSCQSCQCQQQSCSCSSCSCSCCSCC` etc

The second file, will contain the names and properties of the students-for-hire, as one student per row in a single space delimited format: name quality (1 to 5, 5 is best) speed (1 to 5, 5 is fastest) and price (flat rate: 100 to 1000 TL); for instance:

```
odtulu 5 3 900         // i.e. quality is 5, speed is 3 and the price is 900TL
bogazicili 4 5 1000
itulu 4 4 800
ytulu 3 4 650
sakaryali 1 2 150
...
```

There will be an arbitrary number of such students (there will be at least one).

The program will simply receive the two file paths as command line arguments as well as an integer representing h's money.

```
./program homeworkFilePath studentsFilePath 10000
```

Thread h

This thread will represent the cheating student h. She will read the homeworks from the file, one by one, and place each of them in a queue. Her money can be kept in a global variable. Thread h will terminate after transferring all the homeworks to the queue or after running out of money. Thread h will be detached.

Sample output

```
H has a new homework C; remaining money is 10000TL
H has a new homework Q; remaining money is 9850TL
...
H has no more money for homeworks, terminating.
```
OR
```
H has no other homeworks, terminating.
```

Main thread

The main thread, will of course create all the other threads and a queue to store the homeworks, plus any synchronization mechanisms that might be necessary. The main thread will read the homework in the queue placed there by thread h, and will select the student-for-hire who is suitable and available. If for instance the homework is C, then that means cost is a priority. In other words, the main thread must select the cheapest student-for-hire. If the homework is Q, it must select the highest-quality student-for-hire, and if the homework is S, it must select the fastest student-for-hire. However, a student-for-hire might be busy with another homework. In that case, the main thread must select the second most suitable student-for-hire.

Example: let's say the homework is C. So we must select the cheapest student-for-hire who in this example is sakaryali. However, if sakaryali is busy, then the main thread must select the next cheapest who is ytulu. If he's busy too, then it must select itulu. If they are all busy, then it must wait until someone becomes available.

Once a student-for-hire thread is selected, then the main thread removes the homework from the queue, notifies the selected student to start solving it, decreases the amount of money h has by the price of the selected student-for-hire and repeats for the next homework. The main thread terminates when either all homeworks have been solved, or when h runs out of money.

Sample output

```
5 students-for-hire threads have been created.
Name Q S C
odtulu 5 3 900
bogazicili 4 5 1000
itulu 4 4 800
ytulu 3 4 650
sakaryali 1 2 150
...
Money is over, closing.
OR
No more homeworks left or coming in, closing.
OR
Termination signal received, closing.

Homeworks solved and money made by the students:
odtulu 3 2700
bogazicili 3 3000
itulu 3 2400
ytulu 2 1300
sakaryali 4 600
Total cost for 15 homeworks 10000TL
Money left at G's account: 0TL
```

Threads of student-for-hire

You will have one thread for each student-for-hire. The student-for-hire will simply wait for the main thread to notify it, that there is a homework for it to solve. Then it will sleep for x seconds where `x = 6 - student's speed` to simulate the homework's solving. During this stage the student-for-hire is assumed unavailable for new homeworks. After which, it will wait once again for new homework assignments.

Evaluation

Your program will be evaluated with arbitrary non-empty input files.

Sample output

```
odtulu is waiting for a homework
odtulu is solving homework Q for 900, H has 9100TL left.
odtulu is waiting for a homework
```

Requirements:
- In case of CTRL-C the process must terminate gracefully.
- You are not allowed to create additional files.
- You are not allowed to use mutexes or condition variables for synchronization. Anything else is fair game. I recommend avoiding signals.

Rules:
1) Compilation error: grade set to 1; if the error is resolved during the demo, then evaluation continues.
2) Compilation warning (with respect to the **-Wall** flag); -1 for every warning until 10. -20 points if there are more than 10 warnings; no chance of correction at demo.
3) No makefile: -20
4) No pdf (other formats are inadmissible) report submitted (or submitted but insufficient, e.g. 3 lines of text or no design explanation, etc): -20.
5) A report prepared via latex (pdf and tex source submitted together): +5
6) If the required command line arguments are missing/invalid, your program must print usage information and exit. Otherwise: -10
8) The program crashes/freezes and/or doesn't produce expected output with normal input: -80
9) Presence of memory leak (regardless of amount – checked with valgrind) -30
10) Zombie process (regardless of number) -30
11) Deadlock of any kind due to poor synchronization -50
12) Use of poor synchronization: busy waiting/trylock/trywait/timedwait/sleep or any other form other than those specified at homework: -80
13) Late submissions will not be accepted
14) In case of an arbitrary error, exit by printing to stderr a nicely formatted informative message. Otherwise: -10

Is my homework submission valid?
If it creates all the required threads, yes.

Submission rules:
- Your source files, your makefile and a report; place them all in a directory with your student number as its name, and zip the directory.
- Your report must contain: how you solved this problem, your design decisions, which requirements you achieved and which you have failed.
- The report **must be in English**.
- Your makefile must only compile the program, not run it!
- Do not submit any binary executable files. The TAs will compile them on their own boxes.
- Any deviation from the submission rules will results in automatic (without content evaluation) failure of the homework.

Bayramınız kutlu olsun.