May 27th, 2020

CSE344 – System Programming - Homework #5 - v4
POSIX threads and synchronization thereof

Objective
The objective of this homework is to develop a multithreaded application using pthreads for flower delivery. Clients will make requests for flowers, a central thread will collect those requests, and then delegate the work to the closest florist that has the kind of flower requested.

How
You are provided with a file containing the list of an arbitrary number of florists and the arbitrary types of flowers that they sell (they have an infinite amount of each).

e.g.
```
Ayse (10,25; 1.5): orchid, rose, violet
```

means the florist Ayse sells the three mentioned types of flowers, and her shop is located at location (10,25) in a cartesian 2D grid with real coordinates and her deliveries are made with an average speed of 1.5 clicks per ms (click = 1 unit of distance in the grid).

The file also contains the requests made by clients:

e.g.
```
client1 (55,76): orchid
```

means that client1 is located at (55,76) and has made a request for a single romantic orchid. The flower requests/orders of clients will always be of a single flower type. The requested flower type can be assumed to be on sale by at least one of the florists.

Your central/main thread will have a pool of one thread for every florist. Then it will start processing the clients' requests one by one. For each request, it will delegate it to the closest florist with respect to the client using the Chebyshev distance.

The florist receiving a request (i.e. a flower order) will prepare it and deliver it, by printing on screen how long it took in total (= time of preparation + time of delivery; use the speed and distance information). The preparation of every order at the florist requires a uniformly random amount of time between 1 to 250 ms (closed interval). You can simulate it by letting the thread sleep for a random duration (equal to the time of preparation + time of delivery).

Important: the central thread must process the requests as fast as possible, that means it should delegate the request to the closest florist even if she/he is busy preparing/delivering a request in the meantime (hint: you could use a request queue for each florist).

Once all requests have been processed, all florist threads should terminate and return their sale statistics (i.e. how much time each florist spent in total for preparing and delivering the requests) to the central thread which will then print on screen that information in a nicely formatted table along with the number of requests handled by each client.

If everything goes well your application should print on screen something similar to the following (the numbers are made up):

```
$floristApp -i data.dat
Florist application initializing from file: data.dat
3 florists have been created
```

```
Processing requests
Florist Fatma has delivered a clove to client2 in 56ms
Florist Murat has delivered a rose to client0 in 145ms
Florist Murat has delivered an orchid to client1 in 77ms
…
All requests processed.
Ayse closing shop.
Murat closing shop
Fatma closing shop.
Sale statistics for today:
-------------------------------------------------
Florist          # of sales       Total time
-------------------------------------------------
Ayse             7                1543ms
Fatma            3                750ms
Murat            6                1343ms
-------------------------------------------------
```

Evaluation

Your code will be evaluated with a different data.dat file.

Rules:

- **Solve any and all synchronization issues using only mutexes and/or condition variables.**
- If the command line arguments are missing/invalid your program must print usage information and exit.
- You can assume the file is not empty and that its contents have the proper expected format.
- Each thread should free allocated resources explicitly.
- Never allow zombie processes!
- In case of CTRL-C make sure your program (and all its threads) shuts down gracefully, by deallocating all resources and printing an informative message.
- Don't use busy waiting of any kind, don't use timed waiting, or trylock.
- You will provide a demonstration of your program to the course assistants. They will provide the details.
- No late submissions will be allowed.

Submission:

- your source files, your makefile and a report on how you solved the issues in this homework.

Grading:

- Details are available at moodle.
- Deviating from the homework's goals/rules will be penalized.

Good luck.