Advanced Programming 1 - 89-210 - Ex6

January 25, 2017

Due date: 30.1.17

Intro

As integration process - in our case it will be between the code logic and the GUI interface (next exercise) - involves making different parts of code and responsibilities work together, it is extremely important to write massive testing to every part and make sure they work on their own. Another change to this part is to handle errors - something we didn't do until now, as we assumed correctness of the input. The last thing in this exercise is to handle thread-usage in a more efficient way - using the thread-pool pattern.

Implementation

- Error handling - for everything that has any client input involvement. Further explained in next section
- Thread-Pool Pattern. In previous exercise, we created a new thread for every new trip we got. This is quite wasteful as we could get lots of trips to calculate. Furthermore, it's something we receive dynamically and constantly during the entire process. The thread-pool pattern fits well to this scenario. Implement this pattern and for every new task - a new trip to calculate - assign it to the thread-pool. Initialize the pool with 5 threads.
  - Pay attention - the code in the presentation is not enough. It doesn't handle the termination of the thread-pool class and the shared resources. Make changes to those to handle it correctly.
- Unit tests - won't be tested, but is extremely recommended.

Error Handling Notes

- Expect errors, for example:
  - the map size, a trip with coordinates outside of the map.
  - map with zero or negative dimensions.

        * Map size above 1000*1000 isn't considered an error, but won't be given as input.
    - negatives values of id, age, experience, trip times (zero is also erroneous), tariff.
    - non-included types (of cabs, manufacturer, color etc)
    - unmatched inputs (like more or less variables needed, for example for a car: "0,1,H,G,2")
    - the ongoing input of the next command (other than the ones which were given)
    - achievable path for every trip - not certain anymore

- For any error, print '-1' if the error was on the server, or exit the whole program, if it was on the client.
    - Make sure that any connection to the server from the client is being done only after the input verification. So there won't occur any situation where, for example, the server opens a thread to the client, and will be active as the client was shut down.
    - If the path isn't reachable, do not print '-1'
- You can assume that the ip and port will be correct.
- You can still assume that the number of vehicles will be the same as the number of drivers (and each driver gets a vehicle).

Submission:

A zip containing several items:

- All your code in hierarchical way. The makefile should be outside the directory so we could easily compile your code from your submission folder.
- Details file.

Notes

- Don't be late.
- Submit your work via the submit system, with your personal user account. ONLY ONE SUBMISSION PER GROUP. A duplicated submission will be penalized. Same goes for a submission without 'details.txt' file (or deprecated) [in every submission].
- The makefile should be able to generate two executable files: 'server.out' and 'client.out' so I can run them as following:
    - user@user$ ./server.out 40000
    - user@user$ ./client.out localhost 40000
    - user@user$ ./client.out localhost 40000
    - ... (as many as desired)

  - in different shells of course

- As last exercise we spoke about a time limit of 5 seconds, and this time you can get bigger grids, which could take more time to calculate, the new limit (the total run time of the program) is 60 seconds. But, it won't really need all of it (it's the upper bound).
- Input/Output files will be uploaded later.

Grading Notes

- Automatic tests (*40%*)
- Thread-Pool Pattern (*30%*)
- Releasing all allocated memory (on both server and client) (*10%*)
- C++ correct code writing (*20%*)