Client-server user guide

by Sivashchenko Pavel

2019

Contents

1 Overview

1.1 Introduction

This client-server architecture is based on UNIX-sockets. Server is terminating on machine and client connects to it by IPv4(server's PC ip in LAN). Client forms and initializes graph and sends it to server, then waits for answer. Server receives data, does evaluations and sends result to client.

Highlights

- Secure data input on client's side
- User-friendly CLI
- Supporting multiple connections
- User can choose between algorithms
- Logging system on server's side

1.2 Architecture

Server and client both written in `C\C++`

Server terminates and starts to listen to client. Client forms graph (CLI interface), automatically sends number of nodes, source and destination node to server. Then it forms graph edges and sends them to server. Server receives that information, forms graph on it's side and starts to solve task. User can choose between two algorithms. First one is finding the shortest path with topological pre-sorting of the graph Problem solution consist of two large parts: topological sorting graph using Kahn's topological sort algorithm(taken with some changes from techiedelight.com) for graphs without circles[1]. After that we find the shortes

[1] for graphs with circles server will send corresponding message

path between source and destination nodes with the help of modified Dijkstra algorithm and send result (in string format) to client. More information you can gain throw reading sources. During the work application you will notice some latency between sending and receiving messages but it is done purposely to ensure a stable connection in different quality and speed connection types.

1.3 Interface

1.3.1 Server

Server provides CLI. It automatically terminates and waits for clients to connect. You can write in console anything you want, it will just be interpreted as wrong input, except strings "exit","term" or "Exit" which will terminate server and disconnect clients from it. Basically server supports four simultaneously connected clients[2].Besides, some useful information will be written in console and in logging journal(file is generated automatically and is called log.txt). It shows actual information on server: activating,termination time e.t.c

1.3.2 Client

Client provides CLI too[3].

1. Firstly you should enter the ID of algorithm you want server to use on it's side:"0" means Bellman-Ford algorithm and "1" means Topological Sorting Algorithm
2. Enter number of nodes in graph
3. Enter source node which you want to calculate the shortest path from
4. Enter destination node node which you want to calculate the shortest path to
5. Then you must define graph: enter "From","To" nodes and "Weight" between these nodes
6. When you want to finish input just input "exit" or "Exit" opposite "From"

Just in a second you will receive the answer is string format.

[2]This number is defined as a constant in source file and can be easily changed

[3]Error checking is implemented of course

1.4 Deployment on play-with-docker

1.4.1 Client

Deploying client on play-with-docker.com

- Pull client docker image from my public repository `maxkile/cpp-server` using:

  ```
  docker pull maxkile/cpp-server:client
  ```

- Launch client immidiately using comand:

  ```
  docker run -it maxkile/cpp-server:client ./client <port> <server-ip>
  ```

  Where `<port>` is port you are connecting to, and `<server-ip>` is ip-adress of server you are connecting to(`-it` is key that means that we want to use client in interactive)

- Or you can also run base image(Ubuntu) using:

  ```
  docker run -it maxkile/cpp-server:client
  ```

  And then simply launch client using: `./client <port> <server-ip>`

- After all is done or some kind of error client terminates automatically

1.4.2 Server

Deploying server on play-with-docker.com

- Pull server docker image from my public repository `maxkile/cpp-server` using:

  ```
  docker pull maxkile/cpp-server:server
  ```

- Launch server immidiately using comand:

  ```
  docker run -it -p <port1>:<port2> maxkile/cpp-server:server ./server <port3>
  ```

  Where `<port3>` is port you are listening to and `<port1>` is number of port which is transferring to the port `<port2>`(`-p` is appropriate key)

- Or you can also run base image(Ubuntu) using:

  `docker run -it -p <port1>:<port2> maxkile/cpp-server:server`
- And then launch server using:

  `./server <port3>`
- After terminating from server you can see journal file `"log.txt"`:

  `more log.txt`