Introduction to Network Programming

1st Exam

Date: 2020/11/2 Time: 16:00-18:30

1) (50%) UDP file downloader

Implement a UDP Server and UDP Client. There are some files storing on the server side. Your client should be able to get multiple files it needs from the server. Your server can only bind to an assigned port and the server will wait for the client's request on this port and send the requested files to the client.

Command format:

- Server
  - **./server {port}**

  Example:

  ```
  ./server 1234
  ```

  Server can only receive the client's request using port 1234 (1234 is just for example. It can be any valid port)

- Client: connect to the server.
  - **./client {server-ip} {server-port}**

  Example:

  ```
  ./client 127.0.0.1 1234
  ```

After connecting to the server, client should be able to do the following functions:

- **get-file-list**

Server will return a list of all filenames existing on the server and
client will print out the result:

Files: {file-name1} {file-name2} {file-name3} …

(Hints: you can use <dirent.h> in man page with C, <filesystem> with C++17 and <os> with python to make this work.)

- **get-file {file-name1} {file-name2} {file-name3}…**

Client can get variable number of files from the server. When the client receives the file, the received file name must be the same as the original file name.

- **exit**

Client will close the connection to the server, but server will still running and wait for another connection.

Notes:

- The specified files will locate on the same directory as the server program.
- Use “% “ as the command line prompt. Notice that there is only one space after the prompt.
- You can assume the requested files always exist on the server.
- We will ONLY test **one** client at the same time.

2) (50%) Implement a TCP server. The server does the following functions:

- Server will give client a name by connection order. For example, the first client connects to the server, it will be named user1, the second client will be user2, the third client will be user3. **When user2 disconnect and connect to the server again, it will be named user4.** When client connects to the server, client will receive its user number from the server:

  Welcome, you are {user#}.

- When client connect/disconnect to the server, **server** should output message: **New connection from {ip}:{port} {user#} / {user#} {ip}:{port} disconnected**

The client does the following functions:

- The service accepts the following commands and at least 10 clients:

  Remember to handle incomplete client input.

Command	**Description**	**Output**
list-users	List all online users.	1. Success: user1 user2...
get-ip	Show client's IP address and port number.	1. Success: IP: {ip}:{port}
exit	Disconnect from the server.	1. Success: Bye, {user#}.

Set up:

Download **np_1st_exam.zip** from new E3. After extracting this file, you will see the following directory structure:

```
np_1st_exam/
└── setup.sh --- This script is used to set up directories of your submission.
```

Submission:

Change directory to **np_1st_exam**. Type **./setup.sh <your_student_id>** to set up the directories. Then, you will see the following directories.

```
<your_student_id>/
 ├── P1/ --- This directory is for storing your answer to the problem 1.
 │    ├── server/
 │    │      Put your server codes and a readme file that describes how to compile your
 │    │      program (Makefile is better) here.
 │    └── client/
 │           Put your client codes and a readme file that describes how to compile your
 │           program (Makefile is better) here.
 └── P2/ --- This directory is for storing your answer to the problem 2.
             Put your codes here and a readme file that describes how to compile your program
             (Makefile is better) here.
```

Type **zip –r <your_student_id> <your_student_id>** and upload the **<your_student_id>.zip** to new E3.