Introduction to Network Programming

1st Exam

Date: 2022/11/3 Time: 10:10-12:00

NOTICE: using the hw1 docker image to code midterm exam!

PART1:(50%) UDP file downloader

Implement a UDP Server and UDP Client. There are some files stored 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.

(20%)Communication command format:

- server
  - ./server {port}
- client
  - ./server {server-ip} {server-port}

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

- (10%)get-file-list :

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

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

```
% get-file-list
File:   . .. f1.txt f2.txt makefile server server.cpp
```

(Hints: you can use <dirent.h> in man page with C, <filesystem> with C++17.)

- (10%)get-file {file-name1} {file-name2} {file-name3}…

  The client can get various numbers of files from the server. When the client receives the file, the received file name must be the same as the original file name.

- (10%)exit

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

```
% get-file f1.txt f2.txt
% exit
root@e57d49eba73c:/home/npdemo/310551124/P1/client# ls
client  client.cpp  f1.txt  f2.txt  makefile
```

Note:

- The specified files will be located in 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.

PART2: (50%) Implement a TCP server. The server does the following functions:

- Server will give the 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 disconnects and connects 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#}

  ```
  Welcome, you are user3
  ```

- When client connect/disconnect to the server, server should output message:

  New connection from {ip}:{port} {user#} / {user#} {ip}:{port} disconnected

  ```
  New connection from 2.0.134.68:34372 user1
  New connection from 2.0.134.70:34374 user2
  user2 2.0.134.70:34374 disconnected
  ```

The client does the following functions:

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

Command	Description	Output

(10%) list-users	List all online users.	user1
		user2 …..
		`% list-users`
		`user1`
		`user2`

(10%) get-ip	Show client IP address and port number.	IP: {ip}:{port}
		`% get-ip`
		`IP: 2.0.134.68:34372`

(10%) exit	Disconnect from the server.	Bye {user#}.
		`% exit`
		`Bye user2.`

Set up:

Download np_1st_exam.zip from the 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.

```
./setup.sh XXXXXXXX
```

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>/
├── Part1/ --- This directory is for storing your answer to the problem 1.
│ ├── server/
│ │ ├── server.cpp
│ │ └── makefile (compile server.cpp to **./server**)
│ └── client/
│ ├── client.cpp
│ └── makefile (compile client.cpp to **./client**)
└── Part2/ --- This directory is for storing your answer to the problem 2.
├── server.cpp
├── client.cpp
└── makefile (compile server.cpp to **./server** and client.cpp to **./client**)

upload your <your_student_id>.zip file to the new E3.