

## Programming Assignment 2

Due Nov 7, 2021 at 5:00 pm

**Late penalty: 30% for up to 24hr late; no credit afterwards**

---

This assignment will give you more practice with using transport layer sockets and developing a simple protocol for transferring files between end machines.

### About cooperation with other students:

This assignment is meant to be done alone. Absolutely no cooperation is allowed. If there are questions, ask the course staff; they are there to help you. For this assignment, you must do all thinking, research, and coding by yourself. Do not even discuss with anyone how far you are with this assignment

**ANY HELP (EXCEPT FROM COURSE STAFF) IS PROHIBITED!!!**

Individuals found guilty of violating above policy will be subjected to disciplinary action. (Warning: We may use software to measure the code similarity)

### More Important Note:

Once again, we are giving you the responsibility to REPORT all incidences of cooperation. You MUST report to the instructor (or the TAs) such incidences. If you do not, we will consider you as guilty as those who you witnessed cooperating. **All coding, debugging, web search for functions, etc. must be done by individual students!**

### Preamble:

You will use the **Linux programming environment** for this assignment.

You will need to submit all the required files zipped in a zip file whose name is derived from your student ID. See submission instructions (at the end) for more details.

### The Assignment:

The assignment consists of **four** parts.

### **VERY IMPORTANT NOTE:**

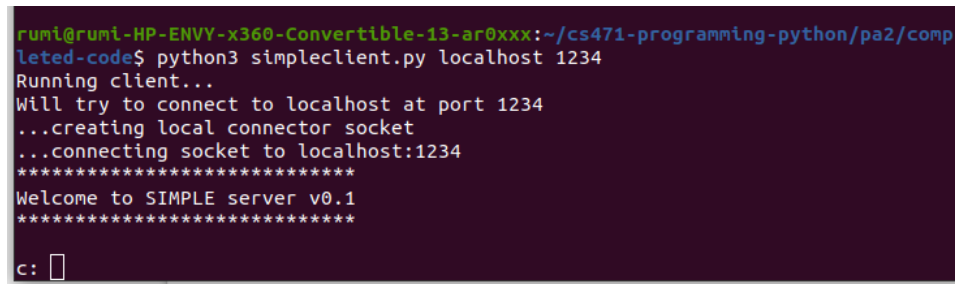
**For each part, you must submit a SEPARATE set of client/server files, as required, even if it means repeating some code from previous parts.**

***You are responsible for providing code for the client as well as for the server SEPARATELY for each part.***

### Part-I:

In this part you have to create a simple server program which listens on a certain port. The hostname and the port number are passed as command-line arguments. *For example:* `python3 simpleserver.py 2090` where '2090' is the port number. Along with the server, you need to create a simple client program which just connects to the server listening on a certain port. *For example:* `python3 simpleclient.py localhost 2090` where '2090' is the port number where the server is running.

When the client connects to the server, the server should send a **welcome message** to the client (See image provided)

A terminal window with a dark purple background. The prompt is 'rumi@rumi-HP-ENVY-x360-Convertible-13-ar0xxx:~/cs471-programming-python/pa2/comp'. The user enters 'python3 simpleclient.py localhost 1234'. The output shows the client running, attempting to connect to localhost at port 1234, creating a local connector socket, and successfully connecting. The server then responds with 'Welcome to SIMPLE server v0.1' between two lines of asterisks. The prompt 'c: ' is visible at the bottom.

```
rumi@rumi-HP-ENVY-x360-Convertible-13-ar0xxx:~/cs471-programming-python/pa2/comp
leted-code$ python3 simpleclient.py localhost 1234
Running client...
Will try to connect to localhost at port 1234
...creating local connector socket
...connecting socket to localhost:1234
*****
Welcome to SIMPLE server v0.1
*****
c: 
```

On the client side if the user types 'EXIT', the client application should close the connection with the server and terminate. A message should pop up at the server side that the client has disconnected and then it should **start listening on the port again** for any other client to connect again.

Also, you need to do error handling in case the user input is incorrect. For example, if a user types `simpleclient.py 12` a message can pop up, "Usage: simpleclient.py <hostname> <port number between 1024 and 65535>". You need to do this error handling for both the client and the server.

By the end of this part, you'll have a working server and client program.

**Your program should do error checking to handle common error cases.** These clients and server will be the starting point for what you will write in Part-II. **You do not need to handle the case where the server unexpectedly ends.**

From Part- II to Part-IV, the client can send requests to the server to send, receive and create files. For simplicity, the file system can be assumed to be flat (i.e. no directories). You do have to handle file sizes of up to `BUF_LEN` bytes. (Although if programmed correctly the system can easily work for file sizes greater than `BUF_LEN` bytes.)

**For the remaining parts, it is important to make sure the client and server programs are in separate directories in your file system.**

### **Part-II:**

In this part, you will start to add some functionality for your client. The HELP command and HOROSCOPE command have already been written out for you on the server side. Complete the getInfoFromServer function in `simpleclient.py` and display the message the server sends it back. Make sure to decode the message before displaying.

Now add the following functionality:

- TIME (sends the time back to client)
- MAC (sends the server's MAC address to client)
- IP (sends the server's IP address to client)
- LIST (sends the filenames present in the server's directory. NOT THE FILES, only the fileNAMES)
- OS (sends the server's operating system information to the client. The platform module may be helpful here)
- USERS (sends the list of users present on the server to the client. Same result as when you type w in the terminal)

It is advised to keep the getInfoFromServer function as general as possible. You will also need to cater to the individual inputs in the analyzeInput function. Note that the first word of the commands the client writes should be case insensitive and hence are converted to upper case when analyzing the input.

Note that the aforementioned commands are inputted on the client's side and sent to the server. **The server should fetch the data requested and send it back to the client.**

You are free to import any python packages for this. **Just make sure that your program can run in a Linux environment.** Test it using your account on the course server where we will grade all the programming assignments.

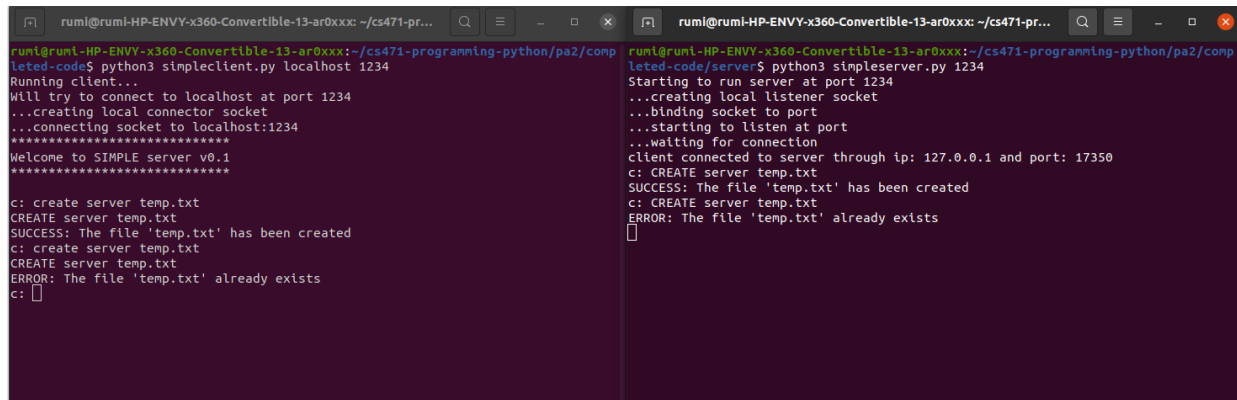
### **Part-III:**

In this part you will start to add the functionality to transfer files between the client and the server. For this functionality you'll first need to add **CREATE** commands.

#### **1. CREATE server:**

When the client sends the 'CREATE server <filename>' command to the server, the server should create an empty file with the filename that the client has specified. After the server creates the file the server should send back a message. If the file already exists on the server side, the server should send back an error message.

Example:



```
ruml@ruml-HP-ENVY-x360-Convertible-13-ar0xxx: ~/cs471-programming-python/pa2/completed-code$ python3 simpleclient.py localhost 1234
Running client...
Will try to connect to localhost at port 1234
...creating local connector socket
...connecting socket to localhost:1234
*****
Welcome to SIMPLE server v0.1
*****
c: create server temp.txt
CREATE server temp.txt
SUCCESS: The file 'temp.txt' has been created
c: create server temp.txt
CREATE server temp.txt
ERROR: The file 'temp.txt' already exists
c: 
```

```
ruml@ruml-HP-ENVY-x360-Convertible-13-ar0xxx: ~/cs471-programming-python/pa2/completed-code/server$ python3 simpleserver.py 1234
Starting to run server at port 1234
...creating local listener socket
...binding socket to port
...starting to listen at port
...waiting for connection
client connected to server through ip: 127.0.0.1 and port: 17350
c: CREATE server temp.txt
SUCCESS: The file 'temp.txt' has been created
c: CREATE server temp.txt
ERROR: The file 'temp.txt' already exists

```

## 2. CREATE client

Similar to the previous case, when the client types CREATE client <filename>, a file should be created with the inputted name in the client's directory. The server should also be informed and should display the success/error message (same message as above).

**For the CREATE command any type of file can be asked to be created: txt file, mp4 files etc. Also, you can assume that no illegal file name would be provided.**

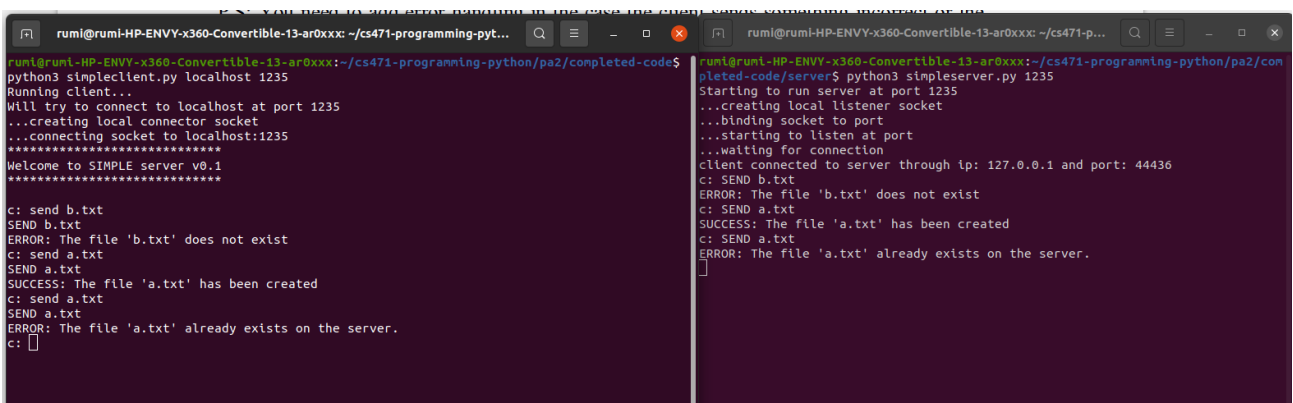
P.S: You need to add error handling in the case the client sends something incorrect or the commands have an incorrect number of arguments etc. The client / server program should not unexpectedly end at any point.

## Part-IV:

This is an extension of Part-III and now you will be adding the 'SEND', 'RECEIVE' and 'DELETE' commands.

1. **SEND:** When the client sends the 'SEND <filename>' command to the server the specific file present in the client's directory should be sent to the server (you see, why it is important to have the server running in a different directory). If the file does not exist on the client side, an error message should pop up (also at the client end). If the file is already present at the server then the server should send back an error message. (In this part you should be able to handle text, audio video files under BUF\_LEN)

Example:

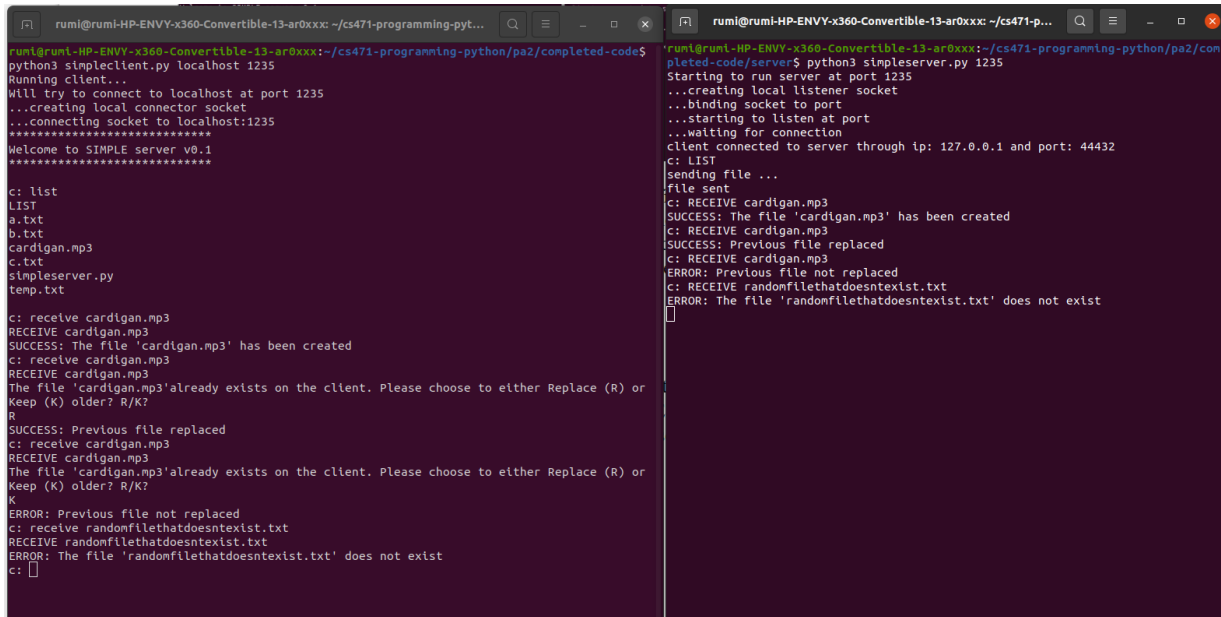


```
ruml@ruml-HP-ENVY-x360-Convertible-13-ar0xxx: ~/cs471-programming-python/pa2/completed-code$ python3 simpleclient.py localhost 1235
Running client...
Will try to connect to localhost at port 1235
...creating local connector socket
...connecting socket to localhost:1235
*****
Welcome to SIMPLE server v0.1
*****
c: send b.txt
SEND b.txt
ERROR: The file 'b.txt' does not exist
c: send a.txt
SEND a.txt
SUCCESS: The file 'a.txt' has been created
c: send a.txt
SEND a.txt
ERROR: The file 'a.txt' already exists on the server.
c: 
```

```
ruml@ruml-HP-ENVY-x360-Convertible-13-ar0xxx: ~/cs471-programming-python/pa2/completed-code/server$ python3 simpleserver.py 1235
Starting to run server at port 1235
...creating local listener socket
...binding socket to port
...starting to listen at port
...waiting for connection
client connected to server through ip: 127.0.0.1 and port: 44436
c: SEND b.txt
ERROR: The file 'b.txt' does not exist
c: SEND a.txt
SUCCESS: The file 'a.txt' has been created
c: SEND a.txt
ERROR: The file 'a.txt' already exists on the server.

```

2. **RECEIVE:** When the client sends the 'RECEIVE <filename>' command to the server, the server should send back the specific file. If the file does not exist at the server, then the server should send back an error message (which the client should display). If the file already exists at the client, then the client program should ask the user if the file should be replaced or the older file should be kept. Example:



```
rumi@rumi-HP-ENVY-x360-Convertible-13-ar0xxx: ~/cs471-programming-pyt...  
python3 simpleclient.py localhost 1235  
Running client...  
Will try to connect to localhost at port 1235  
...creating local connector socket  
...connecting socket to localhost:1235  
*****  
Welcome to SIMPLE server v0.1  
*****  
  
c: list  
LIST  
a.txt  
b.txt  
cardigan.mp3  
c.txt  
simpleserver.py  
temp.txt  
  
c: receive cardigan.mp3  
RECEIVE cardigan.mp3  
SUCCESS: The file 'cardigan.mp3' has been created  
c: receive cardigan.mp3  
RECEIVE cardigan.mp3  
The file 'cardigan.mp3' already exists on the client. Please choose to either Replace (R) or  
Keep (K) older? R/K?  
R  
SUCCESS: Previous file replaced  
c: receive cardigan.mp3  
RECEIVE cardigan.mp3  
The file 'cardigan.mp3' already exists on the client. Please choose to either Replace (R) or  
Keep (K) older? R/K?  
K  
ERROR: Previous file not replaced  
c: receive randomfilethatdoesntexist.txt  
RECEIVE randomfilethatdoesntexist.txt  
ERROR: The file 'randomfilethatdoesntexist.txt' does not exist  
c:   
  
rumi@rumi-HP-ENVY-x360-Convertible-13-ar0xxx: ~/cs471-p...  
python3 simpleserver.py 1235  
Starting to run server at port 1235  
...creating local listener socket  
...binding socket to port  
...starting to listen at port  
...waiting for connection  
client connected to server through ip: 127.0.0.1 and port: 44432  
c: LIST  
sending file ...  
file sent  
c: RECEIVE cardigan.mp3  
SUCCESS: The file 'cardigan.mp3' has been created  
c: RECEIVE cardigan.mp3  
SUCCESS: Previous file replaced  
c: RECEIVE cardigan.mp3  
ERROR: Previous file not replaced  
c: RECEIVE randomfilethatdoesntexist.txt  
ERROR: The file 'randomfilethatdoesntexist.txt' does not exist  
c:   

```

3. **DELETE:** This command should delete a file either at the server or the client end (Depends on the argument.) When the user types "DELETE client <filename>" the file should be deleted at the client end and if the user types "DELETE server <filename>" the file should be deleted at the server end. Again, if the file does not exist it should give an error message. The server should also be informed and should display the success/error message ('deleted' instead of 'created' in the success message).

Try to start this assignment early. because although this may look easy, this could take a lot of time. We have also uploaded sample program for each task so you know what you have to make. **You're allowed to use any python libraries in this assignment.** Try to use functions because a lot of work in these parts are similar. So you don't want to retype or copy paste code.

### Important Programming Notes:

- What ports to use? You **MUST** use only **ONE** port in your server which is XYabc where 20XY-??-0abc is your roll number. Example, if your roll no is 14100055 (2014-10-0055), your required port number is 14055. At the client side, let the OS/program choose a port for you (randomly).
- Make sure you close every socket that you use in your program. If you abort your program, the socket may still hang around and the next time you try and bind a new socket to the port ID you previously used (but never closed), you may get an error. The un-closed port would become available again after a little time, so in such a situation just use a different port in the range 8000-9000 only temporarily for a short time.
- Should error checking (for bad commands) be done on the client or server side? You want it on **BOTH** sides. The server never knows if the client is correctly written, so it needs to check the client input. Similarly, the client can't know if the user is going to input the correct information, so it should check; client should also check return error message from the server (in case there is a problem there as well).

### Where do I start and where do I get help?

Make sure you have read the Beej's guide (or another) to Unix Socket Programming. It **IS** useful. Or talk to the TAs or the instructor if you have any questions.

### What is the BIGGEST piece of advice?

**Start early.** This assignment will require you to debug a lot of client/server code and will take considerable amount of time. Keep the entire weekend(s) for the assignment and add another several days here and there! The only other advice is to get help from the course staff. Learn to debug programs using *print* as well.

### What and where to submit?

1. Very importantly, strictly stick to the following submission guidelines.
2. Use the names of the files specified in this handout.
3. **We need `simpleserver.py` and `simpleclient.py` files for each part in separate directories.**
4. Zip all these files together in one file named `<your_student_num>.zip` where `<your_student_num>` is an 8-digit number. Example, if your roll no is 14100055, your submission file will be 14100055.zip. Be very careful with this, our script might throw away zip files that do not follow this convention. No contest will be allowed in that case.
5. Do NOT email your assignment to us.
6. We need just one submission of your zip file through LMS system (use the *Assignment link* in your portfolio, and NOT the Dropbox).