CPSC 526 Winter 2025

Assignment 2

Due date: posted on D2L
Weight: 17% of your final grade.

Group work is allowed but not required. Max. group size is 2 students.

In this assignment you are going to write a client-server application, allowing smart transfer of files between two computers. You may implement your application in C, C++ or Python, although I strongly recommend Python. The code you submit must run on the Linux lab computers.

The server

The server will listen for connections on a port specified on the command line. When a client connects, the client and server will perform a simple handshake, during which the server will verify that the client knows a shared secret. The secret should be provided to the server and client via the environment variable `SECRET526`.

If the handshake is unsuccessful, the server will disconnect the client and then resume listening for new connections. If the handshake is successful, the server will accept requests from the client. After receiving a request, it will perform any required actions and send relevant results back to the client. The server will keep processing requests from the client until the client disconnects, at which point the server will resume listening for new connections.

The server only needs to accept and service a single client at a time.

Handshake

The server will generate a random alphanumeric string of at least 8 characters and send it to the client. The client will append this challenge to its secret, and compute SHA256 digest on it. It will then send the computed digest back to the server. The server will compute the same digest using its own secret and the challenge and compare it to the received digest. The challenge is considered successful if the two digests match.

If the challenge is successful, the server will send the client a message indicating success. If the challenge was unsuccessful, the server will send the client some type of error message, and immediately disconnect.

The client

The client will connect to the server using a hostname and a port specified on command line and using a secret from the environment variable `SECRET526`. If the handshake is unsuccessful, the client will quit immediately.

If the handshake is successful, the client will wait for the user to type in commands on standard input. After each command, the client will interpret the command, if appropriate send relevant request(s) to the server, wait for replies, and report relevant feedback to the user. The client will keep processing commands until the user terminates the client, e.g. by pressing `CTRL-C`.

Commands

The client should support the following commands:

Command	Description
`pwd`	Displays the current working directory of the server on the client. This command is already implemented in the starter code.
`cd <dirname>`	Changes the server's current working directory to `<dirname>`. If successful, displays the new current working directory.
`ls [options]`	List the contents of the current working directory on the server – must be implemented by executing `ls` on the server and displaying its output to the client. If options are provided, they must be passed to the server. Please note that the starter code partially implements this command.
`cat <filename>`	Displays the contents of the file `<filename>` from the server on the client.

`sha256 <filename>`	Computes SHA256 digest of a file `<filename>` on the server and then displays it on the client.
`download <filename>`	Implements smart downloading of a file from the server to the client. Your application will first compute and compare the digests of the remote and local file `<filename>`. If they are the same, the command does nothing. If they are different, the file is downloaded from the server onto the client's computer.
`upload <filename>`	Implements smart uploading of a file from the client to the server's filesystem. If the digest of the local file is the same as the digest of the remote file, no uploading takes place. If the digests differ, the client uploads the local file to the server.

Each command must provide appropriate feedback to the user, whether the command succeeds or not. If any of the commands cannot be completed due to some error that occurs on the server side or on the client side, an appropriate error message must be displayed to the user on the client side. For example, an upload of a file could fail due to file not existing locally, or file not having right permissions to be read, or destination not having correct write access. In other words, check for error return values and report them.

It should be possible to specify any file name and directory name in the above commands, including unusual cases, where names contain non-alphanumeric characters. You do not need to implement wildcard support.

Feel free to add additional commands to the client, for example, to help you with debugging. These will not be graded.

Error handling

Both the client and server must handle all errors, including disconnects on both ends, gracefully. If a client disconnects at any point, the server should continue running and accept new clients. If a server disconnects, the client should report an appropriate message and quit. If the user types in a bad command, or the command cannot be executed for some reason (i.e. non-existent file), the client must show an appropriate error message and then continue to run as normal.

Starter code

Please download the starter code for this assignment using git:

```
$ git clone https://gitlab.com/cpsc526w24/a2-file-transfer.git
$ cd a2-file-transfer
```

The starter code contains unfinished implementations of the `client.py` and `server.py`. Feel free to use any of the starter code in your solution, but keep in mind the code is incomplete and even wrong in some places. The purpose of the code is to give you a starting point, so that you do not have to start from scratch. The client successfully connects to the server, accepts commands from the user, and is even able to execute some of them (`pwd` works correctly, `ls` only works partially).

The starter code even executes a handshake between the server and the client, although it is an incorrect implementation – it does not compute a digest at all. It is only good enough to prevent students from accidentally connecting to another student's server.

Do not share the starter code, or any parts of it, with anyone outside of this class. For example, do not post any of the starter code in public git repositories. If you are reusing some of the starter code in your solution and want to use GitHub or similar, make your repositories private.

Running the starter code

To start the server from the starter code, do the following:

```
$ cd testDir
$ SECRET526=111111 ../server.py -d 22222
```

Replace `111111` with a secret of your own choosing (any string) and replace `22222` with a random port number [1024..65535]. The reason you want to run the server in the `testDir/` directory is to reduce the chance of accidentally overwriting any files in the source directory during testing. The `-d` option turns on extra debugging output.

Now you can start the client:

```
$ cd testDir
$ SECRET526=111111 ../client.py -d localhost 22222
```

If your client runs on a different computer than the server, replace `localhost` with the hostname of the server's computer.

Both the server and the client use the environment variable `SECRET526` to configure the secret. This is more secure than using command line arguments, as command line arguments are visible to other users on shared Linux systems, but environment variables are not. Another way to specify the secret is to hardcode it inside the `common.py` file (near the top), or by creating a file `.secret526` with secret in the same directory where the `server.py` file resides.

Communication protocol

You can design any protocol for communication between your client and server, including a binary protocol. However, I strongly recommend that you stick to the text-based protocol I implemented in the starter code, and extend it. A text-based protocol will allow you to test your server and client independently of each other, for example, using the `netcat` command line tool.

Let's say you start the server like this:

```
$ SECRET526=111111 ../server.py -d 22222
```

Then you can use `netcat` to interact with it like this (highlighted text represents your input):

```
$ nc csx1 22222
7febb32220
1111117febb32220
OK
ls -lQ
total 11
-rw------- 1 pfederl profs   50 Jan 27 22:26 "code.cpp"
-rw------- 1 pfederl profs    0 Jan 27 22:32 "empty.dat"
-rw------- 1 pfederl profs 2192 Jan 28 18:11 "penguin.jpg"
-rw------- 1 pfederl profs   60 Jan 28 18:30 "weird2.txt "
-rw------- 1 pfederl profs   67 Jan 28 18:31 " weird 3. txt "
-rw------- 1 pfederl profs   59 Jan 28 18:30 "weird .txt"
---
^C
```

Similarly, you could use netcat in listening mode to pretend it is a server, e.g. starting it like this:

```
$ nc -lk 22222
```

And connect to it using your client:

```
$ SECRET526=123 ./client.py localhost 22222
```

Example session with a client

```
$ SECRET526=123 ./client.py csx2 55543
connected
> pwd
/home/profs/pfederl/cpsc526w24/a2
> cd /tmp
/tmp
> ls -l
-rw------- 1 pfederl profs  50 Jan 27 22:26 code.cpp
-rw------- 1 pfederl profs   0 Jan 27 22:32 empty.dat
-rw------- 1 pfederl profs 406 Jan 27 22:22 image1.png
> upload code.cpp
no such local file
> download code.cpp
downloaded 50 bytes
> download code.cpp
download skipped - local file matches remote file
> upload code.cpp
upload skipped - local file matches remote file
> upload empty.dat
uploaded 10 bytes
^C
```

Allowed 3rd party code and libraries

You may use any library/package installed on the Linux lab computers.

You may use any code given to you by your instructor or your TA.

You may not call any external programs, except the following: `ls`, `sha256sum` and `openssl`.

Other than the code listed above, all code your implementation uses must be written by you.

Group work

You are allowed to work on this assignment with another student (max. group size is 2). Just beware that during the demo you will be asked to demonstrate your familiarity with all code. If you do decide to group up, make sure that both of you understand your code. Your group partner can be from different tutorial and/or lecture section.

Demo

You are required to demo your assignments individually. The time for your demo will be arranged by your TAs.

During the demo you will be asked to run your server code on one computer, and the client on another, and demonstrate their functionality. You may be asked to adjust some functionality of your code to show you are familiar with the code you submitted.

Make sure your code works on the Linux lab machines.

Submissions

All group members must submit the assignment on D2L:

`client.[py|c|cpp]`	Implementation of your client
`server.[py|c|cpp]`	Implementation of your server
`common.py`	If you modified it.
`???`	Any other files needed to compile & run your application. For example, if you submit C/C++ code, include Makefile and common header files.
`Readme.[txt|pdf|docx]`	Include names of all group members at the top. List all features you implemented. Include any instructions on how to compile & run your code.

Marking

Category	Marks
Correct handshake implementation, using SHA256.	10
`cd` – works with existing/accessible directories `cd` – works with non-existent/non-accessible directories	5 5
`pwd`	5
`ls` – works with unusual files/directories present	10
`cat` – works with typical filenames `cat` – works with any custom crafted content during grading, including filenames containing dashes, and content ending with dashes	10 5
`upload` – works for at least one upload of a non-empty text file, regardless of digest `upload` – works for at least one upload of a non-empty binary file, regardless of digest `upload` – works correctly for all binary/text files & considers digests	10 5 5
`download` – works for at least one upload of a non-empty text file, regardless of digest `download` – works for at least one upload of a non-empty binary file, regardless of digest `download` – works correctly for all binary/text files & considers digests	10 5 5
At least one command handles unusual filenames/dirnames All commands handle unusual filenames and dirnames * ‘unusual’ refers to names with non-alphanumeric characters, such as spaces and quotes	5 5

Penalties	
Non-functioning code	-100
Not following specifications e.g. not accepting secret using environment variables	Up to -100
Incorrect error handling	-5/per instance, up to -30
Occasional crashes	-5/per instance, up to -20
Missing or non-working client, but working server (must work using netcat)	Up to -50
Ugly or unreadable code	Up to -15
Missing/inadequate documentation	Up to -15
Missing readme.txt or missing information	Up to -20
Code not uploaded to D2L	-100

General information about all assignments:

- All assignments are due on the date listed on D2L. Late submissions will not be marked.
- Extensions may be granted only by the course instructor.
- After you submit your work to D2L, verify your submission by re-downloading it.
- You can submit many times before the due date. D2L will simply overwrite previous submissions with newer ones. It is better to submit incomplete work for a chance of getting partial marks, than not to submit anything. Please bear in mind that you cannot re-submit a single file if you have already submitted other files. Your new submission would delete the previous files you submitted. So please keep a copy of all files you intend to submit and resubmit all of them every time.
- Assignments will be marked by your TAs. If you have questions about assignment marking, contact your TA first. If you still have questions after you have talked to your TA, then you can contact your instructor.
- All programs you submit must run on the Linux workstations in your lab. If your TA is unable to run your code on the Linux machines, you will receive 0 marks.
- Assignments must reflect your own individual work, or the work of the group that belong to. Here are some examples of what you are not allowed to do for assignments: you are not allowed to copy code or written answers (in part, or in whole) from anyone else; you are not allowed to collaborate with anyone; you are not allowed to share your solutions (including code or pseudocode) with anyone; you are not allowed to sell or purchase a solution; you are not allowed to make your code available publicly. This list is not exclusive. For further information on plagiarism, cheating and other academic misconduct, check the information at this link: http://www.ucalgary.ca/pubs/calendar/current/k.html .
- We may use automated similarity detection software to check for plagiarism. Your submission will be compared to other students (current and previous), as well as to any known online sources. Any cases of detected plagiarism or any other academic misconduct will be investigated and reported.

Appendix – sample of possible interaction with the server using netcat

```
$ nc csx1 14456
890b40d3e6
5891b5b522d5df086d0ff0b110fbd9d21bb4fc7163af34d08286a2e846f6be03
OK
ls -l
total 4
-rw------- 1 pfederl profs  50 Jan 27 22:26 code.cpp
-rw------- 1 pfederl profs   0 Jan 27 22:32 empty.dat
-rw------- 1 pfederl profs 406 Jan 27 22:22 image1.png
/###/
cd ..
OK
cd /a/b/c/d/does-not-exist
no such directory
ls
client.py
common.py
```

```
server.py
testDir
/###/
cat testDir/code.cpp
int main() {
    return 7 * 8;

}

/*end of file*/
/###/
cat does-not-exist.cpp
no such file
/###/
xyz
unknown command xyz
download image1.png
iVBORw0KGgoAAAANSUhEUgAAAMgAAADIAQMAAACXljzdAAAABlBMVEX///8AAABVwtN+AAAACXBI
WXMAAA7EAAAOxAGVKw4bAAABNklEQVRYw+2WvY3DMAyFaahQqRE0ijdz5M08ikZwqcIQ75G6ixMn
9ZGFCdiA+DUEfx/Rbbf9r7HYRmmjyCXKo7shB760AU7wxYrX4ockPtImnHKJ9cHVGwk79ShROyTI
qEei1Qbs0omffWBJxpQoqcuX+bEkz7xKJ37ZMZaE95nEPXOjRQrOxQ2B76A57DOjB9+qbU+ClhsQ
RQZZKHc35OQjo9c+MCWJg1wznloGXtsZtWOCHsWyRh+QzvZ67kR7Ajf2ziDjDPsh2Iak84OMFu3R
4oaQ3Dk5dZ2yjlIlP+S35jRy+WgvG8majDvHshNVH7xMiTkZ+iA8o75qZVOipRYdr25uF61sT3BL
oA9WWX6VnBGNGnXOb9vFnAw9ir8Ivjq1U2uZk78poa5RF/qYHzty22329gMxn66/G4dT/AAAAABJ
RU5ErkJggg==
/###/ OK
download non-existant.file
/###/ ERROR cannot open file
upload test1.txt
SGVsbG8=
#
OK
cd /
OK
upload test1.txt
SGVsbG8=
#
ERROR permission denied
^C
```