SFTP

[CSED331] Term Project - SFTP

이성환 (20130486)
생명과학과

Introduction

SFTP(SSH File Transfer Protocol / Secure File Transfer Protocol, SFTP)은 신뢰할 수 있는 데이터 스트림을 통해 파일 접근, 파일 전송, 파일 관리를 제공하는 네트워크 프로토콜입니다. SSH 프로토콜을 통해 실행되며 SSH의 보안 및 인증 기능을 지원합니다. SFTP는 또한 Password Sniffing 및 Man-in-the-Middle Attack 공격으로부터 보호합니다. 암호화 및 암호화 해시 기능을 사용하여 데이터의 무결성을 보호하고 서버와 사용자를 모두 인증합니다.

이번 프로젝트에선 Linux Ubuntu 환경에서 동작하고 SFTP의 동일한 기능을 지원하는 FTP를 구현합니다.

구현에 요구되는 사항은 아래와 같습니다.

1. Experimental setup

- In server side, we execute `python ftp_server.py` to construct server.
- In client slide, we execute `python ftp_client.py` to execute client.
  Then, the program will immediately require authentication information.

2. Requirements - Correctness

- A client has to authenticate their identity.
- A client can traverse the server directory structure.
- A client can traverse the client directory structure.
- A file needs to be uploaded from client to server
- A file needs to be downloaded from server to client.
- Multiple file could be uploaded from client to server.
- Multiple file could be downloaded from server to client.
- A server can handle more than 100 clients at the same time.

3. Requirements - Performance

- The upload speed will be similar to that of existing implementation.
- The download speed will be similar to that of existing implementation.

4. Clients Interface

- `cd path` : Change remote directory to 'path'
- `exit` : Quit sftp
- `get [-afPpRr] remote [local]` : Download file
- `lcd path` : Change local directory to 'path'
- `lls [ls-options] [path]` : Display local directory listing
- `lpwd` : Prifnt local working directory
- `ls [-1afhlnrSt] [path]` : Display remote directory listing
- `put [-afPpRr] local [remote]` : Upload file
- `pwd` : Display remote working directory

Execute

1. Server 실행

`ftp_server.py`에서 아래 코드 부분을 수정합니다.

```
#TODO: Set your INFO
BIND_PORT = 2200
CUSTOM_USER_NAME = 'root'
CUSTOM_PASSWD = '1234'
key_filename = "id_rsa"
```

아래 명령어로 SFTP 서버를 실행합니다. `private_key_file`에는 RSA private key file의 path를 입력합니다.

```
$ python ftp_server.py private_key_file
Listening for connection ...
```

`Listening for connection ...` 메시지가 뜬다면 클라이언트의 접속이 가능한 상태입니다.

2. Client 실행

`ftp_server.py`에서 아래 코드 부분을 수정합니다. Server와 동일한 RSA로 설정합니다.

```
#TODO : Set yout RSA KEY
key_file = "./id_rsa"
```

아래 명령어로 SFTP 클라이언트를 실행합니다. `user`에는 접속 username, `host`는 서버 IP 혹은 url, `port`는 접속 포트를 입력합니다. 이후 패스워드 입력 창에 패스워드를 입력합니다.

```
$ python ftp_client.py user@host -p port
Password:
```

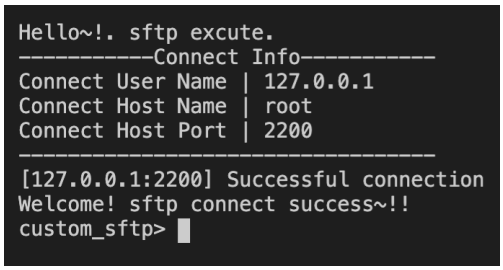

figure1. SFTP Client 접속 화면

3. Local 명령어

- `lpwd` : 현재 로컬 디렉토리 위치를 알려줍니다.

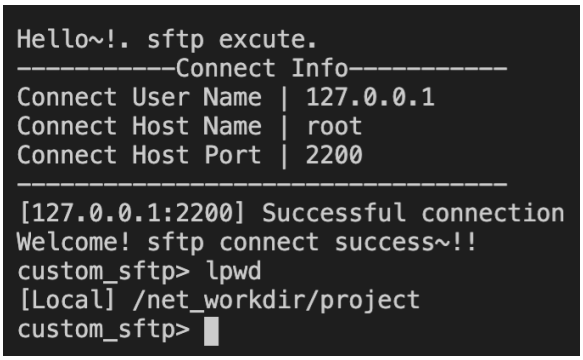

figure2. SFTP Client lpwd 명령어 실행 화면

- `lls [ls-options] [path]]` : 현재 로컬 디렉토리에 위치한 파일 리스트를 보여줍니다.

figure3. SFTP Client lls 명령어 실행 화면

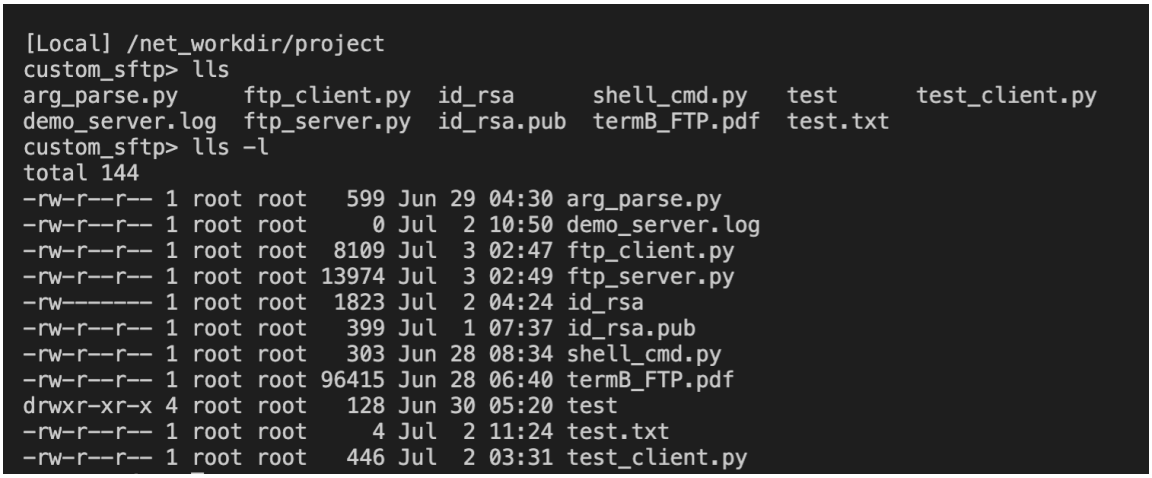

figure4. SFTP Client lls -l 명령어 실행 화면

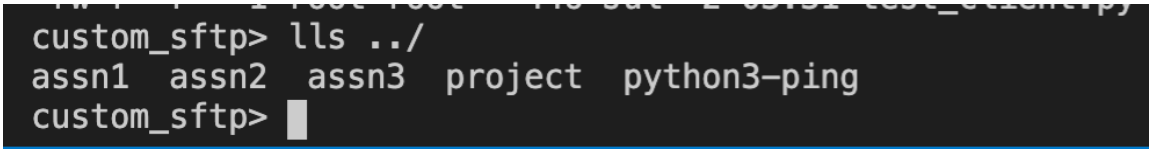

figure5. SFTP Client lls ../ 명령어 실행 화면

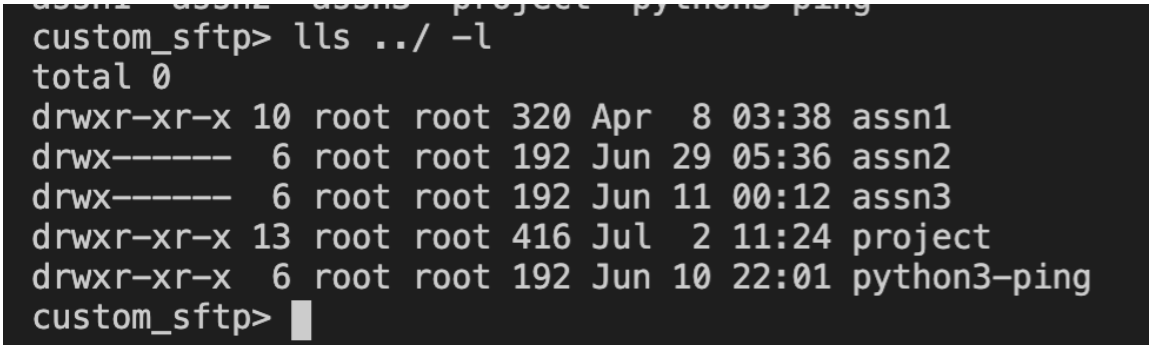

figure6. SFTP Client lls ../ -l 명령어 실행 화면

- `lcd path` : 로컬 디렉토리 위치를 변경합니다.

figure7. SFTP Client lcd .. 명령어 실행 화면

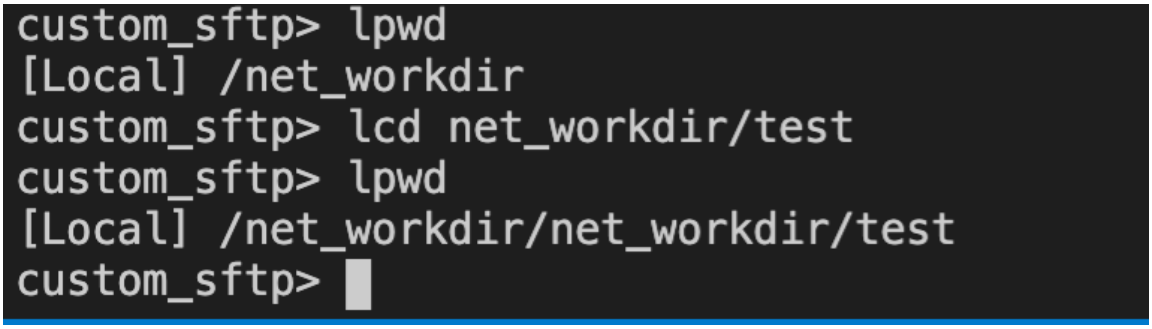

figure8. SFTP Client lcd net_workdir/test 명령어 실행 화면

4. Remote 명령어

- `pwd` : 현재 원격 디렉토리 위치를 알려줍니다.

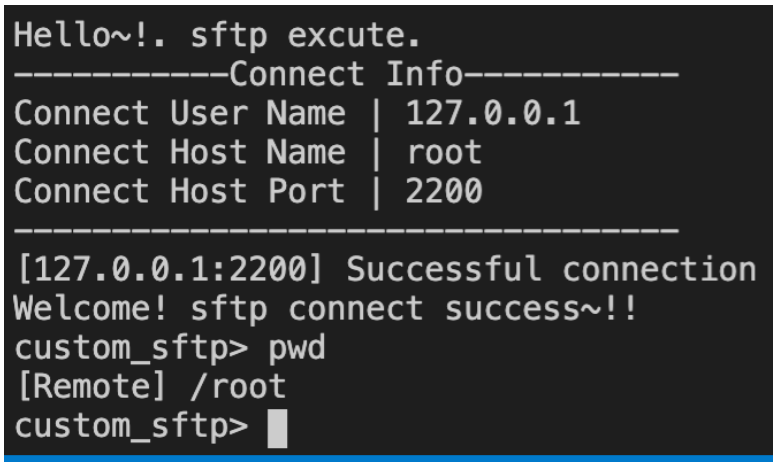

figure9. SFTP Client pwd 명령어 실행 화면

- `ls [ls-options] [path]` : 현재 원격 디렉토리에 위치한 파일 리스트를 보여줍니다.

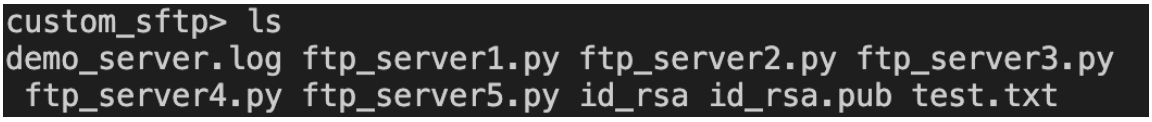

figure10. SFTP Client ls 명령어 실행 화면

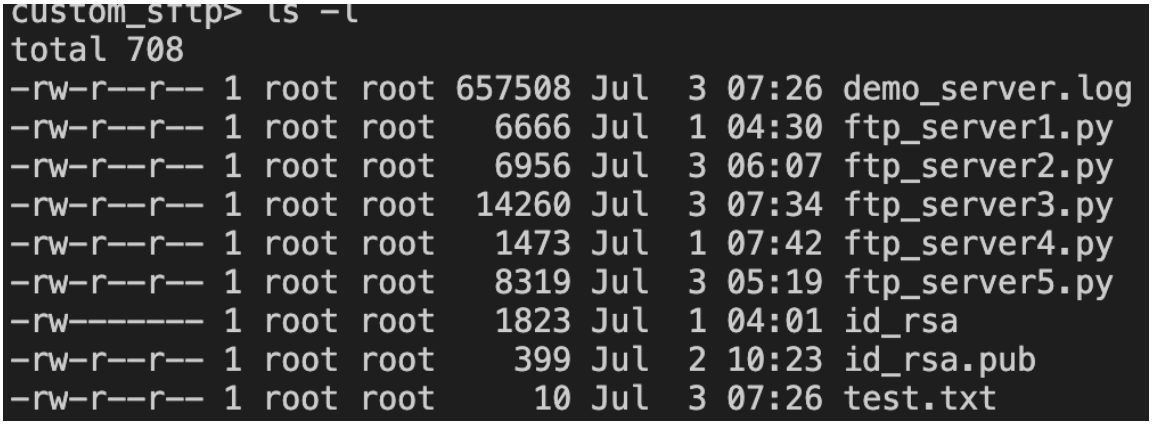

figure11. SFTP Client ls -l 명령어 실행 화면

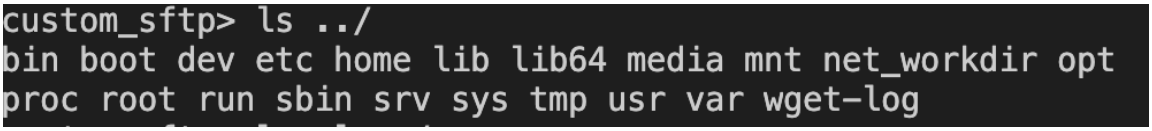

figure12. SFTP Client ls ../ 명령어 실행 화면

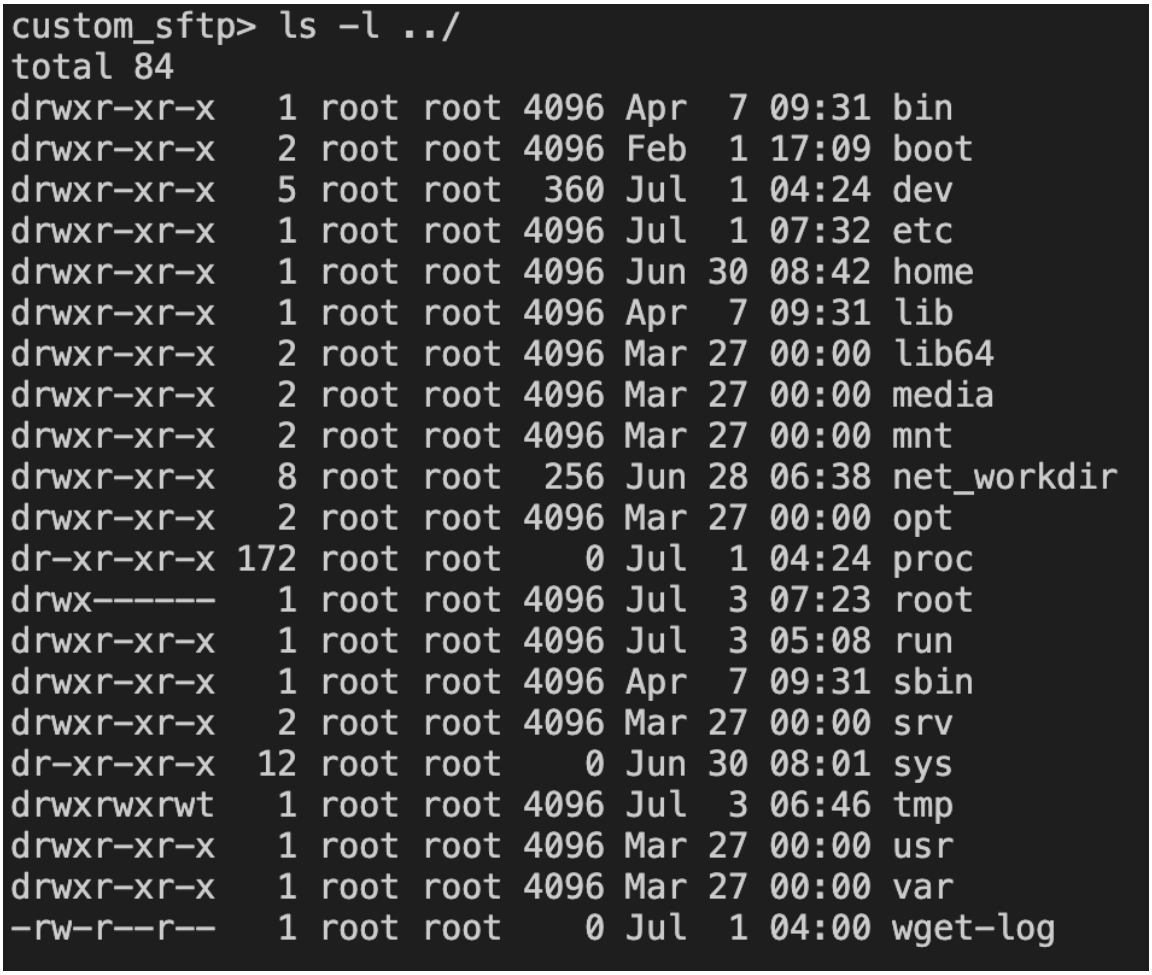

figure13. SFTP Client ls -l ../ 명령어 실행 화면

- `cd path` : 원격 디렉토리의 위치를 변경합니다.

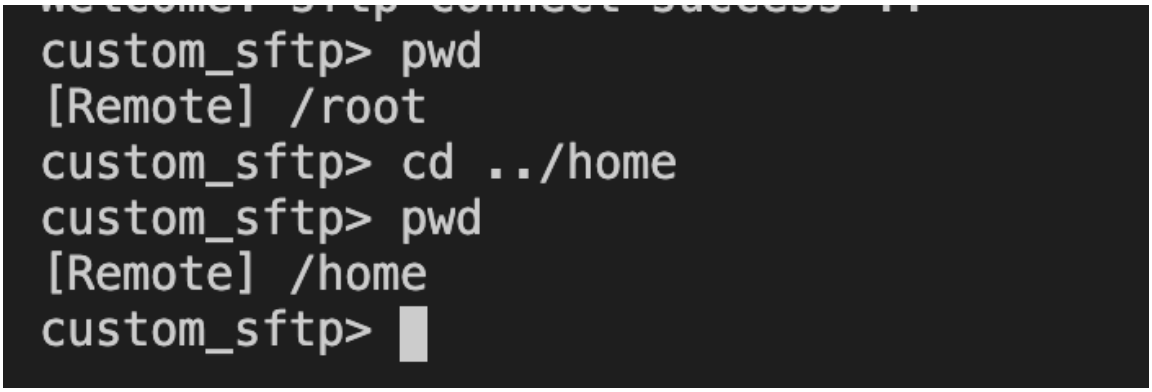

figure13. SFTP Client cd ../home 명령어 실행 화면

5. 파일 전송 명령어

- `put [-afPpRr] local [remote]` : 로컬 파일을 원격으로 업로드합니다.

```
Hello~!. sftp excute.
-----------Connect Info-----------
Connect User Name | 127.0.0.1
Connect Host Name | root
Connect Host Port | 2200
----------------------------------
[127.0.0.1:2200] Successful connection
Welcome! sftp connect success~!!
custom_sftp> ls
bin demo_server.log ftp_server.py id_rsa id_rsa.pub
custom_sftp> lls
ftp_client.py  ftp_server.py  id_rsa  id_rsa.pub  local.txt  test
custom_sftp> put local.txt
local.txt
/net_workdir/project/local.txt
/root/local.txt
[Local] /net_workdir/project/local.txt -> [Remote] ./local.txt
custom_sftp> ls
bin demo_server.log ftp_server.py id_rsa id_rsa.pub local.txt
custom_sftp> lls
ftp_client.py  ftp_server.py  id_rsa  id_rsa.pub  local.txt  test
custom_sftp>
```

figure14. SFTP Client put local.txt 명령어 실행 화면

```
custom_sftp> lls
ftp_client.py  ftp_server.py  id_rsa  id_rsa.pub  local.txt  test
custom_sftp> ls
bin demo_server.log ftp_server.py id_rsa id_rsa.pub local.txt remote_folder
custom_sftp> ls remote_folder

custom_sftp> put local.txt remote_folder
/root/remote_folder/local.txt
[Local] /net_workdir/project/local.txt -> [Remote] ./remote_folder/local.txt
custom_sftp> ls remote_folder
local.txt
custom_sftp>
```

figure15. SFTP Client put local.txt remote_folder 명령어 실행 화면

- `get [-afPpRr] remote [local]` : 원격 파일을 로컬로 다운로드합니다.

figure16. SFTP Client get remote.txt 명령어 실행 화면

figure17. SFTP Client get remote.txt local_folder 명령어 실행 화면

- `exit` : sftp 접속을 종료합니다.

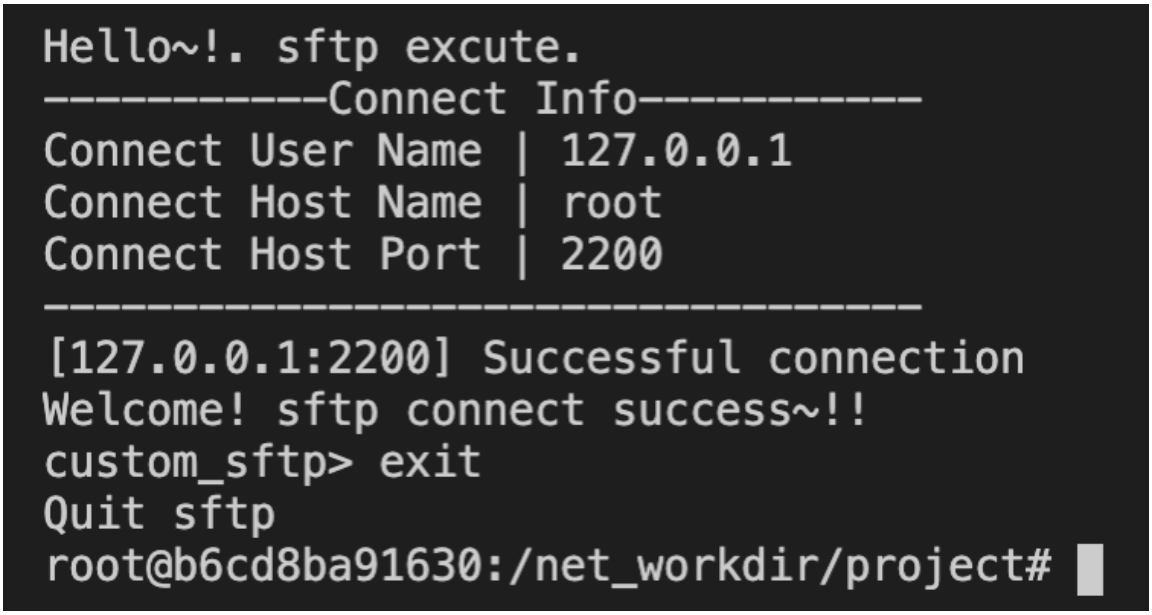

figure18. SFTP Client exit 명령어 실행 화면

figure19. SFTP Server 접속이 끊어진 로그 화면

6. 클라이언트 다중 접속

최대 200개의 다중 접속을 허용하도록 설정했습니다.

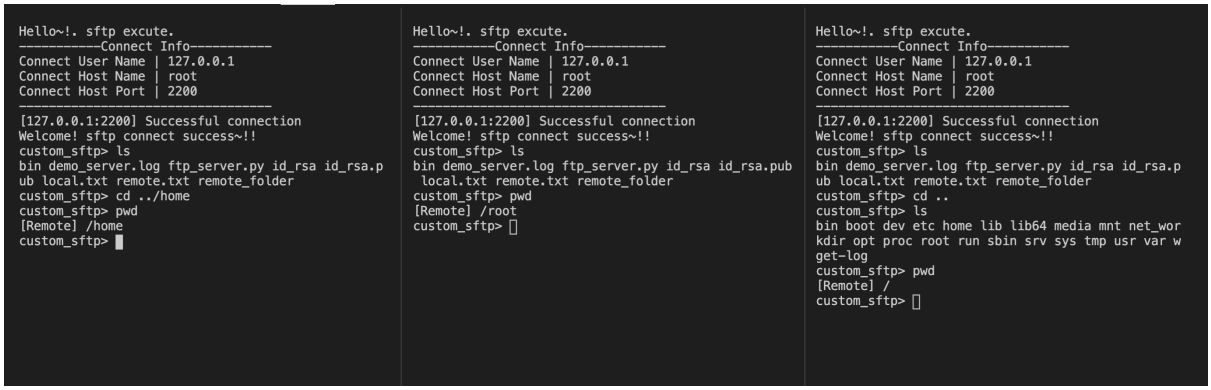

figure18.SFTP Client 다중 접속 화면

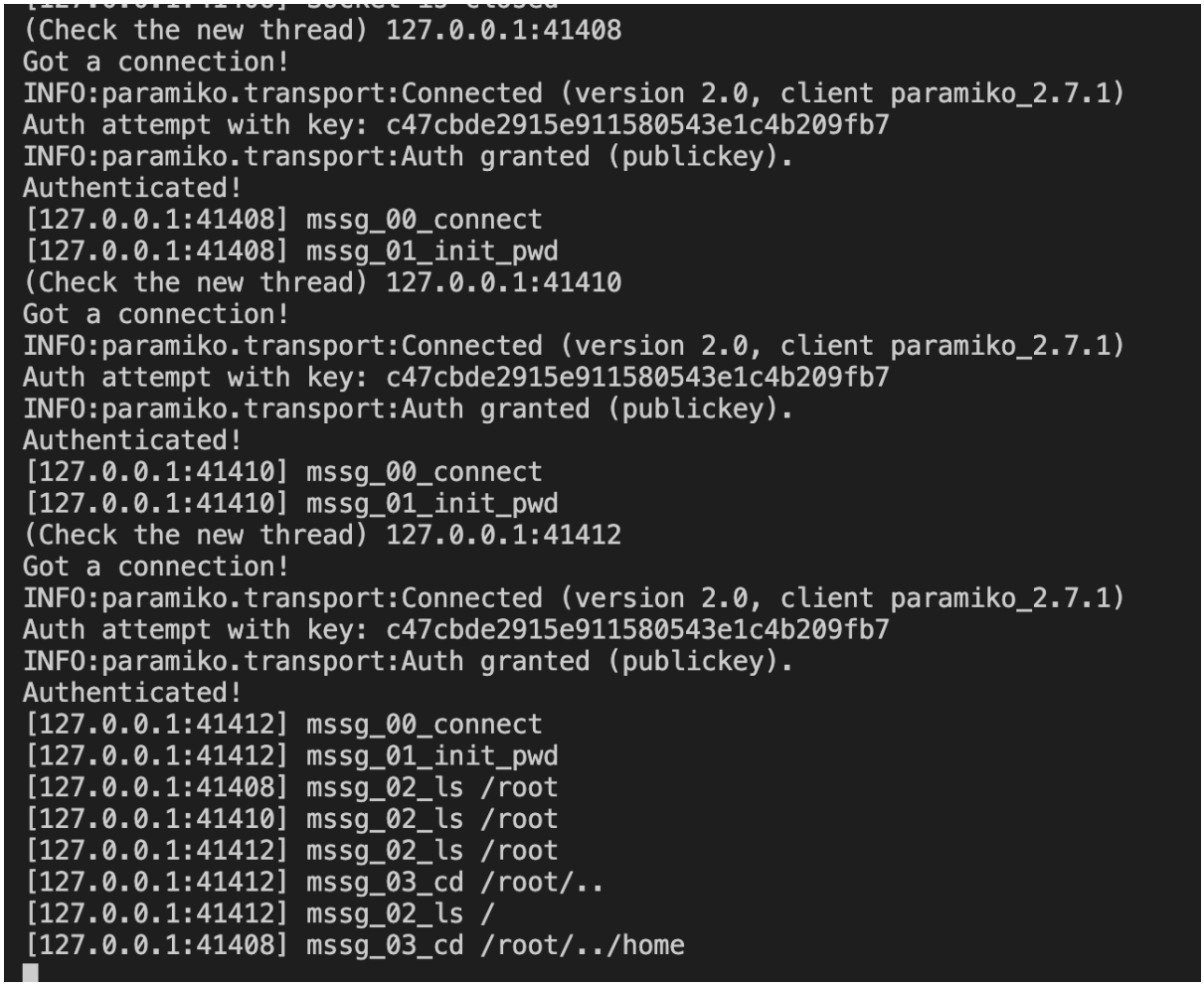

figure19.SFTP Server 다중 접속 로그 화면 (로컬에서 실험 진행, 클라이언트 별로 포트가 다름)

Discussion &Conclusion

이번 프로젝트를 통해 SSH와 SFTP의 기본 원리를 이해할 수 있었습니다. 특히나 SSH 연결은 기존의 소켓 통신보다 보안을 강화한 프로토콜로 과제로 구현해야 했던 소켓 프로그램보다 구현이 조금 더 까다로웠습니다. 또 서버는 다중 접속을 허용해야 하기 때문에 멀티 스레드를 이용하였습니다. SSH 통신을 통해 원격 디렉토리를 검색하고 이를 로컬로 통신을 하였습니다. 파일 전송을 시도하면 해당 클라이언트 스레드에서 SFTP를 위한 소켓을 서버와 새로 연결하고 해당 연결을 통해 파일 전송을 진행하며 파일 전송을 완료하면 SFTP 소켓은 종료되고 다시 SSH 연결을 위한 소켓을 열어 통신을 합니다. 이를 통해 지속적인 연결을 유지하여 파일 전송과 원격 디렉토리 검색을 이후에도 시도할 수 있습니다.

이번 프로젝트를 통해 SFTP 서버와 클라이언트를 구현 하면서 학기 중 수업을 통해 배웠던 지식을 정리 할 수 있었습니다.