시스템 프로그래밍 실습

FTP 3-3

Class : D

Professor : 최상호 교수님

Student ID : 2020202090

Name : 최민석

Introduction

이번 Assignment3-3 에서는 지금까지 구현한 FTP 명령어들과 추가로 USER, PASS, TYPE, RETR, STOR 를 구현하고, fork 를 통한 concurrent 서버 시스템, 화이트리스트를 통한 엑세스 컨트롤 및 로그인 시퀀스, 명령어와 리스폰스를 주고받는 컨트롤 커넥션과 FTP 처리 결과를 클라이언트로 전송하는 데이터 커넥션을 분리한 형태로 작동하는 코드를 구현하고 테스트해볼 것이다.

엑세스 컨트롤을 위해 access.txt 파일에 허용할 IP 주소를 명시하고 passwd 파일에서 로그인 가능한 사용자 이름과 비밀번호를 저장하고 화이트리스트로 사용한다. 서버에 접속하면 사용자는 로그인 시퀀스를 진행하며 로그인을 최대 3 회 시도할 수 있다.

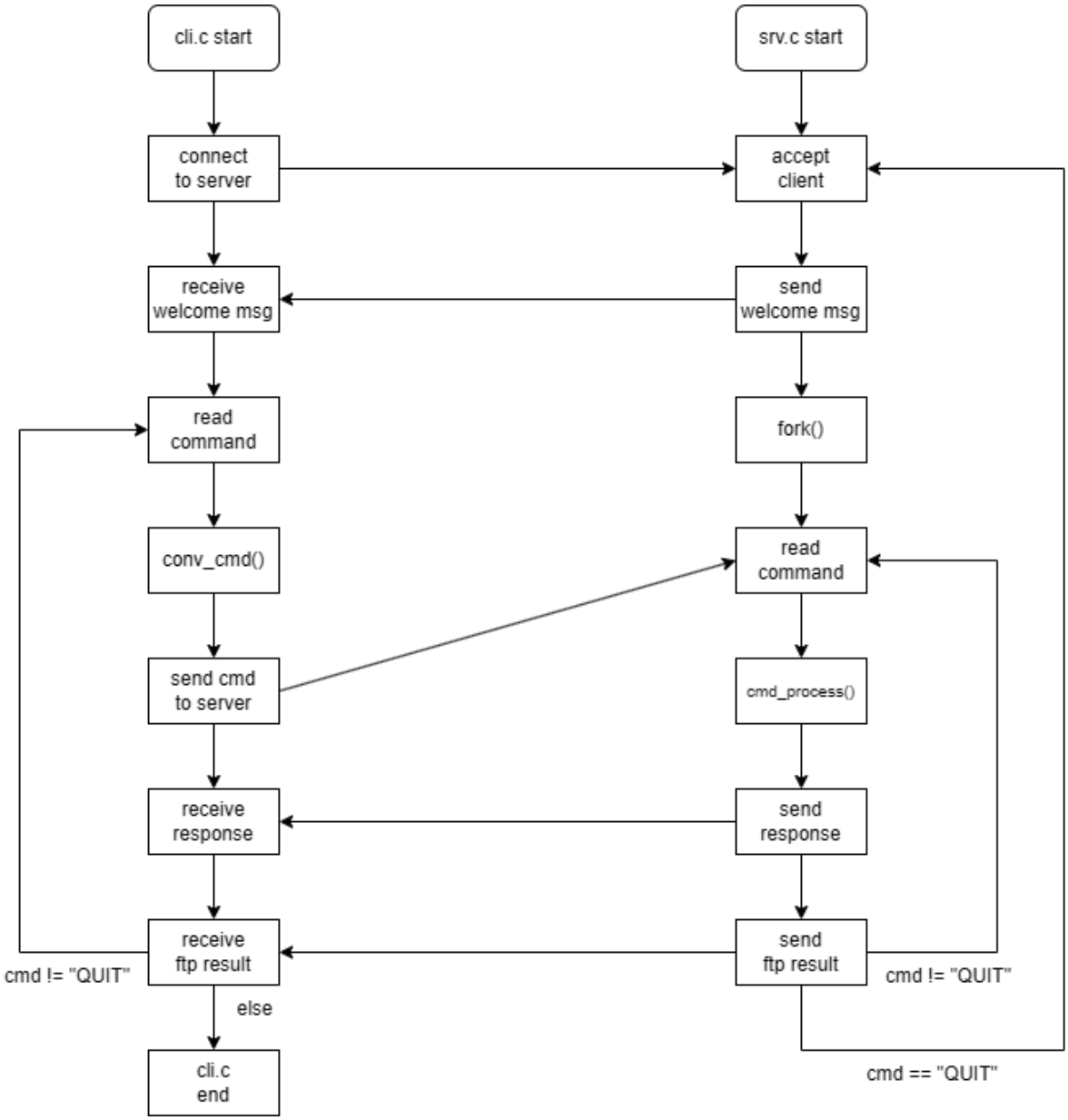
PORT 명령어는 데이터 커넥션을 개설하는 명령어로 직접 사용하지는 않고 사용자가 데이터 커넥션을 사용하는 ls, get, put 명령어를 사용하면 자동적으로 선행 명령어로 전송되어 데이터 커넥션을 사용할 수 있는 상태로 만든다.

get (RETR) 명령어는 서버 디렉토리에 존재하는 파일을 클라이언트로 가져온다. argument 로 파일의 이름을 전송해 서버에서 파일의 내용을 읽고 데이터 커넥션으로 내용을 전송하고, 클라이언트에서는 동일한 이름과 내용을 가진 파일을 생성한다.

put (STOR) 명령어는 클라이언트 디렉토리에 존재하는 파일을 서버로 전송한다. 클라이언트는 데이터 커넥션으로 파일의 이름과 내용을 전송하고, 서버에서는 동일한 이름과 내용을 가진 파일을 생성한다.

bin (TYPE I), ascii (TYPE A) 명령어는 서버의 파일 입출력 상태를 변경한다. bin 모드에서는 서버가 파일을 바이너리 모드로 처리하도록 설정하고, ascii 모드에서는 서버가 파일을 텍스트 모드로 처리하도록 설정한다.

Flow chart


cli.c start
srv.c start
connect to server
accept client
receive welcome msg
send welcome msg
read command
fork()
conv_cmd()
read command
send cmd to server
cmd_process()
receive response
send response
receive ftp result
send ftp result
cmd != "QUIT"
cmd != "QUIT"
else
cli.c end
cmd == "QUIT"

Pseudo code

cli.c

```
define addrstruct ctraddr, dataddr, tempaddr
define string buf, cmd, rcv, rsp
connect(ctraddr)
login()
read(ctraddr, welcome)
print(welcome)
loop {
   read(user, buf);
   conv_cmd(buf, cmd);
   write(ctraddr, cmd)
   cmd_sensitive_communication(cmd, rsp, rcv)
}

function cmd_sensitive_communication(string cmd, string rsp, string rcv) {
   if (cmd == "NLST" || cmd == "LIST" ) {
      accept(tempaddr, dataddr)
      read(ctraddr, rsp)
      print(rsp)
      read(ctraddr, rsp)
      print(rsp)
      read(tempaddr, rcv)
      print(rcv)
      read(ctraddr, rsp)
      print(rsp)
      close(tempaddr)
   }
   else if (cmd == "RETR") {
      accept(tempaddr, dataddr)
      read(ctraddr, rsp)
      print(rsp)
      read(ctraddr, rsp)
```

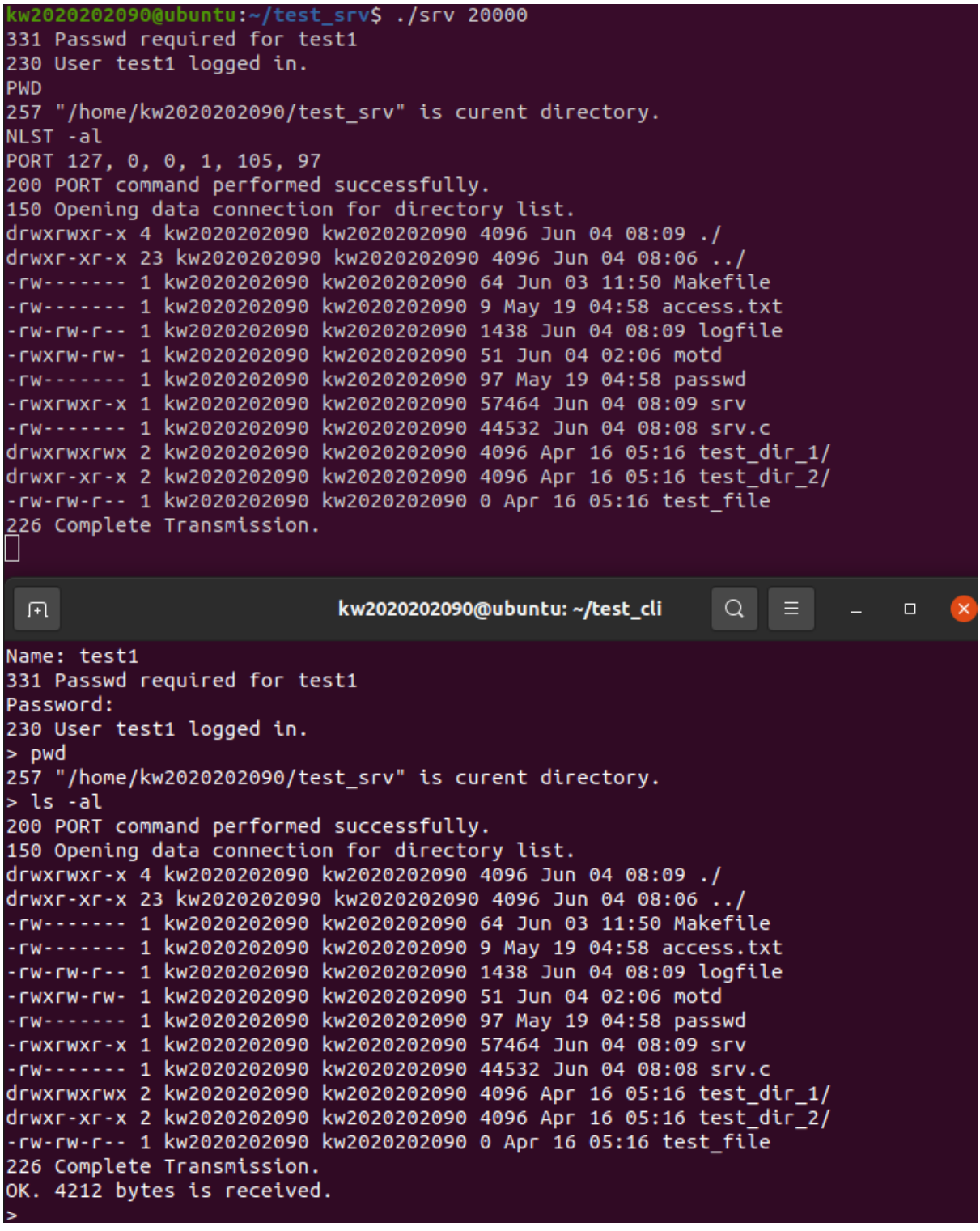
```
        print(rsp)
        read(tempaddr, rcv)
        print("file content is received")
        read(ctraddr, rsp)
        print(rsp)
        file = fopen(arg)
        fwrite(rcv, file)
        fclose(file)
        close(tempaddr)
    }
    else if (cmd == "STOR") {
        accept(tempaddr, dataddr)
        read(ctraddr, rsp)
        print(rsp)
        read(ctraddr, rsp)
        print(rsp)
        file = fopen(arg)
        fread(rcv, arg)
        write(tempaddr, rcv)
        print("file content is sent")
        read(ctraddr, rsp)
        print(rsp)
        fclose(file)
    }
    else {
        read(ctraddr, rst)
        print(rst)
    }
}
```

srv.c

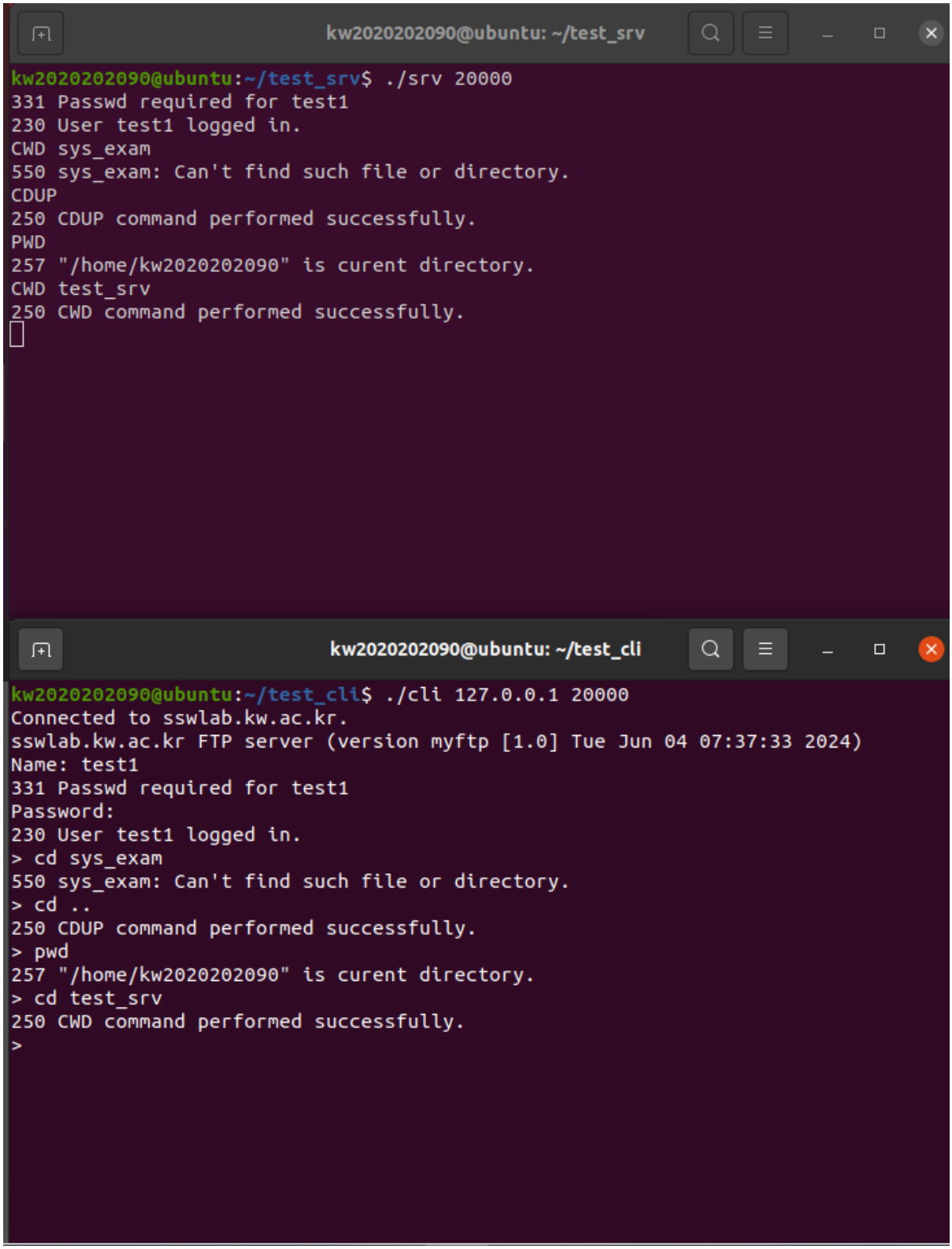
```
define addrstruct ctraddr, dataddr, cliaddr
define string buf, cmd, rsp, rst
bind(ctraddr)
listen(ctraddr)
loop {
  accept(ctraddr, cliaddr)
  logauth()
  write(cliaddr, welcome)
  pid = fork()
  if (pid == 0) {
    loop {
      read(cliaddr, cmd)
      cmd_process(cmd, rst)
      cmd_sensitive_communication(cmd, rsp, rst)
    }
  }
function cmd_sensitive_communication(string cmd, string rsp, string rst) {
  if (cmd == "NLST" || cmd == "LIST") {
    connect(dataddr)
    write(ctraddr, rsp)
    write(ctraddr, rsp)
    write(dataddr, rst)
    write(ctraddr, rsp)
    close(dataddr)
  }
  else if (cmd == "RETR") {
    write(ctraddr, rsp)
    write(ctraddr, rsp)
    file = fopen(arg)
    fread(rst, file)
    write(dataddr, rst)
    write(ctraddr, rsp)
```

```
        fclose(file)
        close(dataddr)
    }
    else if (cmd == "STOR") {
        write(ctraddr, rsp)
        write(ctraddr, rsp)
        read(dataddr, rst)
        file = fopen(arg)
        fwrite(rst, file)
        write(ctraddr, rsp)
        fclose(file)
        close(dataddr)
    }
    else {
        write(ctraddr, rst)
    }
}
```

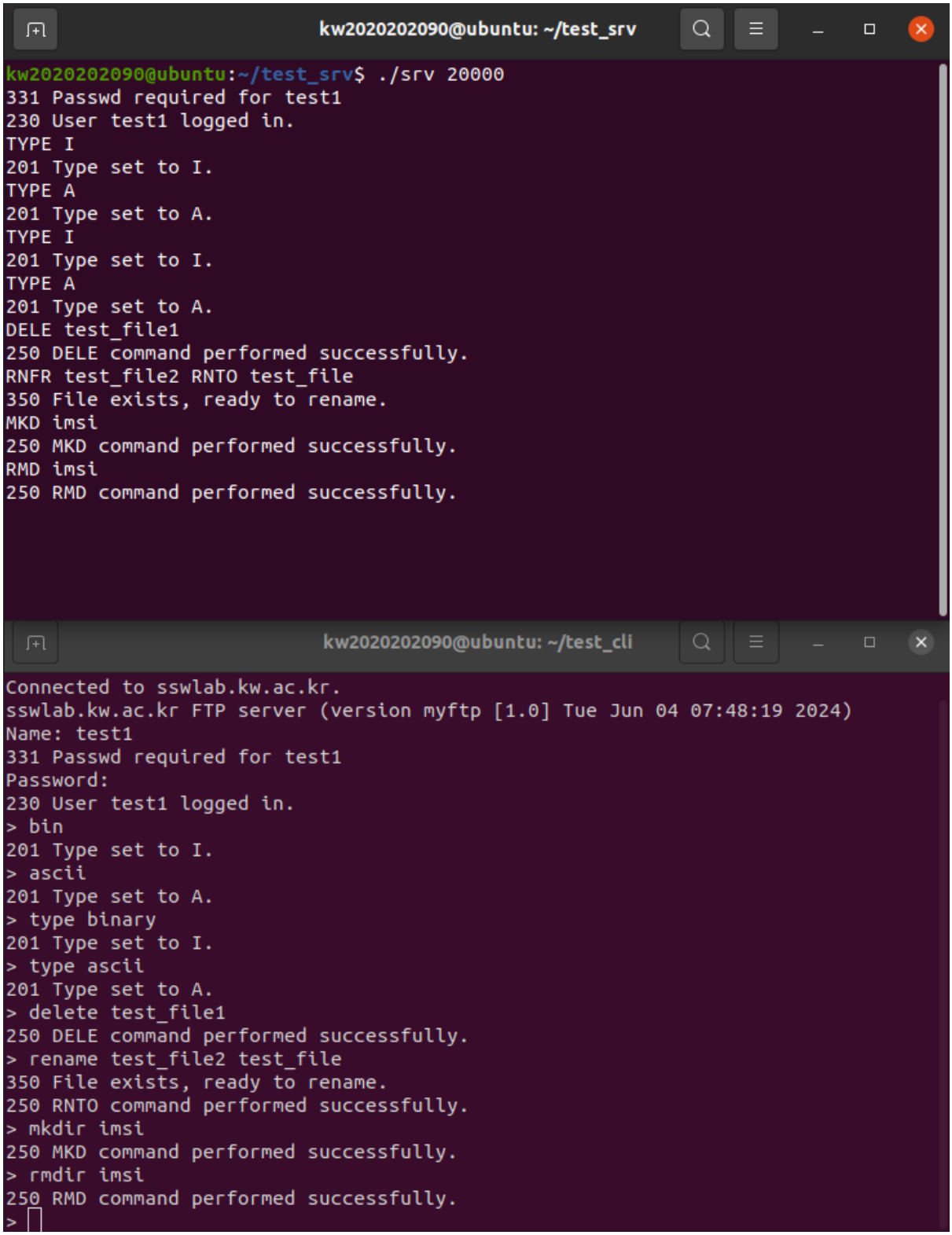
결과화면



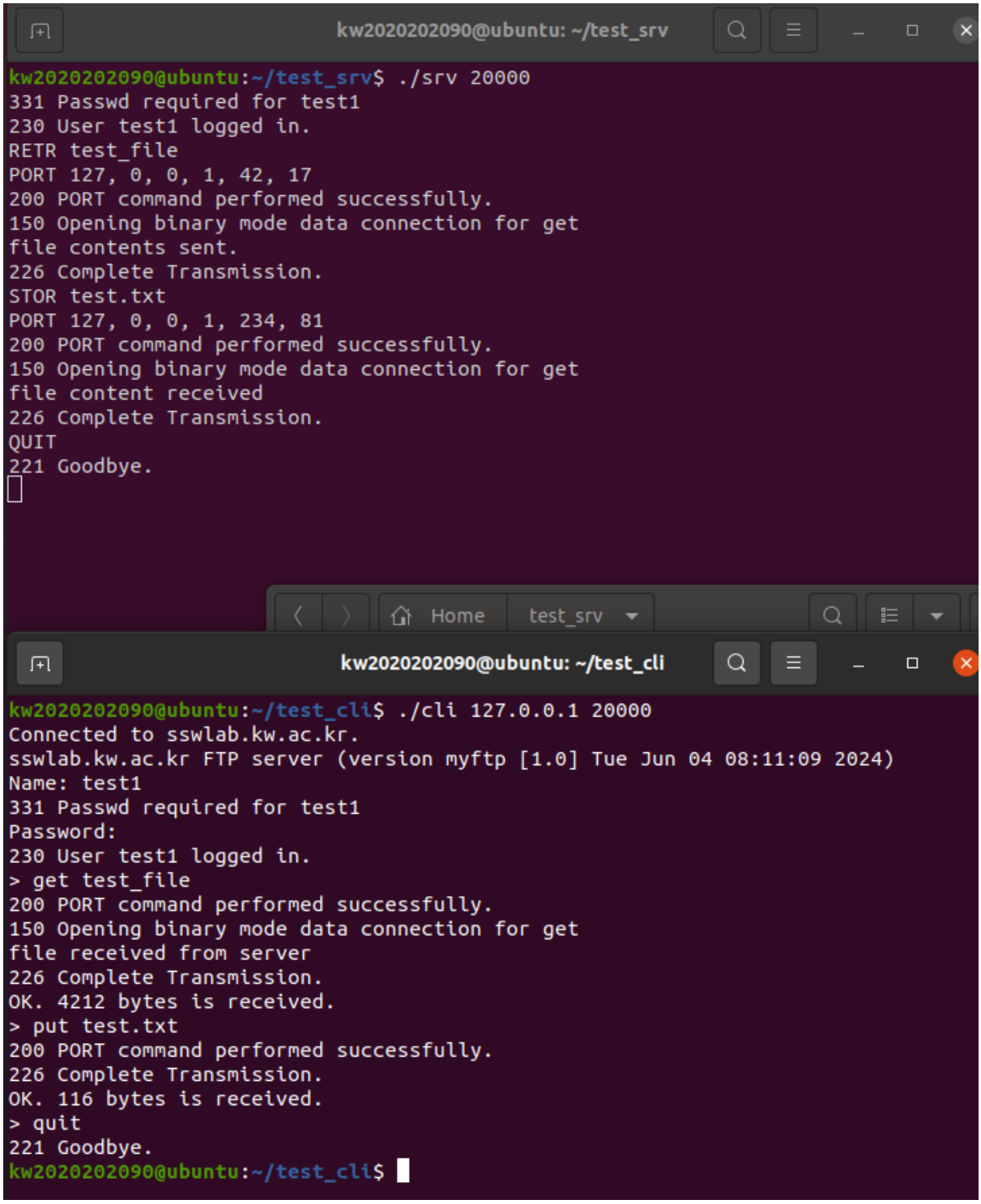
로그인 이후 pwd 명령어와 ls -al 명령어를 수행하였다.



cd, pwd, cd .. 명령어를 수행하였다.



파일 입출력 모드를 변경하고 delete, rename, mkdir, rmdir 명령어를 수행하였다.



get, put 명령어를 수행하였다.

고찰

이번 Assignment 3-3 프로젝트에서 겪은 세 가지 문제는 다음과 같다.

1. 입출력 순서 문제

   서버와 클라이언트 간 read, write 를 할 때 순서가 어긋나면 서로 read 할 때까지 무한 대기하는 문제가 자주 발생하였다. 특히 데이터 커넥션을 사용하는 ls, get, put 명령어 등에서 두드러지게 발생하였다. 또한 read 만 연속으로 받거나 write 만 연속으로 보내면 파이프가 깨지거나 전송에 실패하는 등의 문제도 발생하였다. 이를 해결하기 위해 한 번 read 하면 다음은 무조건 write 를 수행하고, 각 입출력에 따른 시퀀스를 확실하게 점검하여 해결하였다.

2. read, write 순서 문제

   이 문제는 서버와 클라이언트 간 입출력이 아닌 printf, write 를 동시에 사용할 때 발생한 문제로 둘 다 시스템 콜 write 를 이용한 표준출력 함수이기 때문에 수행 속도에 따라 시퀀셜하게 수행되지 않을 수도 있었다. 따라서 read, write 가 잦은 부분에서는 write 만 사용하는 방식 등으로 해결하려 하였지만 완벽하게 해결하지는 못하였다.

3. 기존 코드 리빌딩 문제

   문제라기보단 아쉬웠던 점이다. 이번 3-3 에서는 1 차에서 구현한 명령어의 모음집과, 2 차에서 구현한 fork 를 이용한 concurrent 서버 시스템, 3 차에서 구현한 엑세스 컨트롤과 로그인 시퀀스, 컨트롤 커넥션과 데이터 커넥션의 분리를 모두 집대성하였지만 기존에 구현한 코드를 거의 활용하지 못하였다. 예제의 출력 조건이 기존과 매우 다르게 변경되어 거의 대부분의 코드를 재작업하여 버그 수정에 많은 시간이 필요했다.

작업은 기존 코드를 통합하고 과제 제안서의 내용대로 출력을 수정하고, 3-3 에 필요한 로그 파일 작성, get, put, type 등의 새로운 명령어 구현, 컨트롤 커넥션과 데이터 커넥션 분리, 버그 수정, fork 적용 순서대로 작업하였다. fork 를 사용하면 프로세스가 분리되어 gdb 를 활용한 디버깅이 불가능하기에 제일 마지막에 적용하였다.

Reference

- 강의자료만을 참고하였음.