

시스템 프로그래밍 실습

FTP 3-1

Class : D
Professor : 최상호 교수님
Student ID : 2020202090
Name : 최민석

Introduction

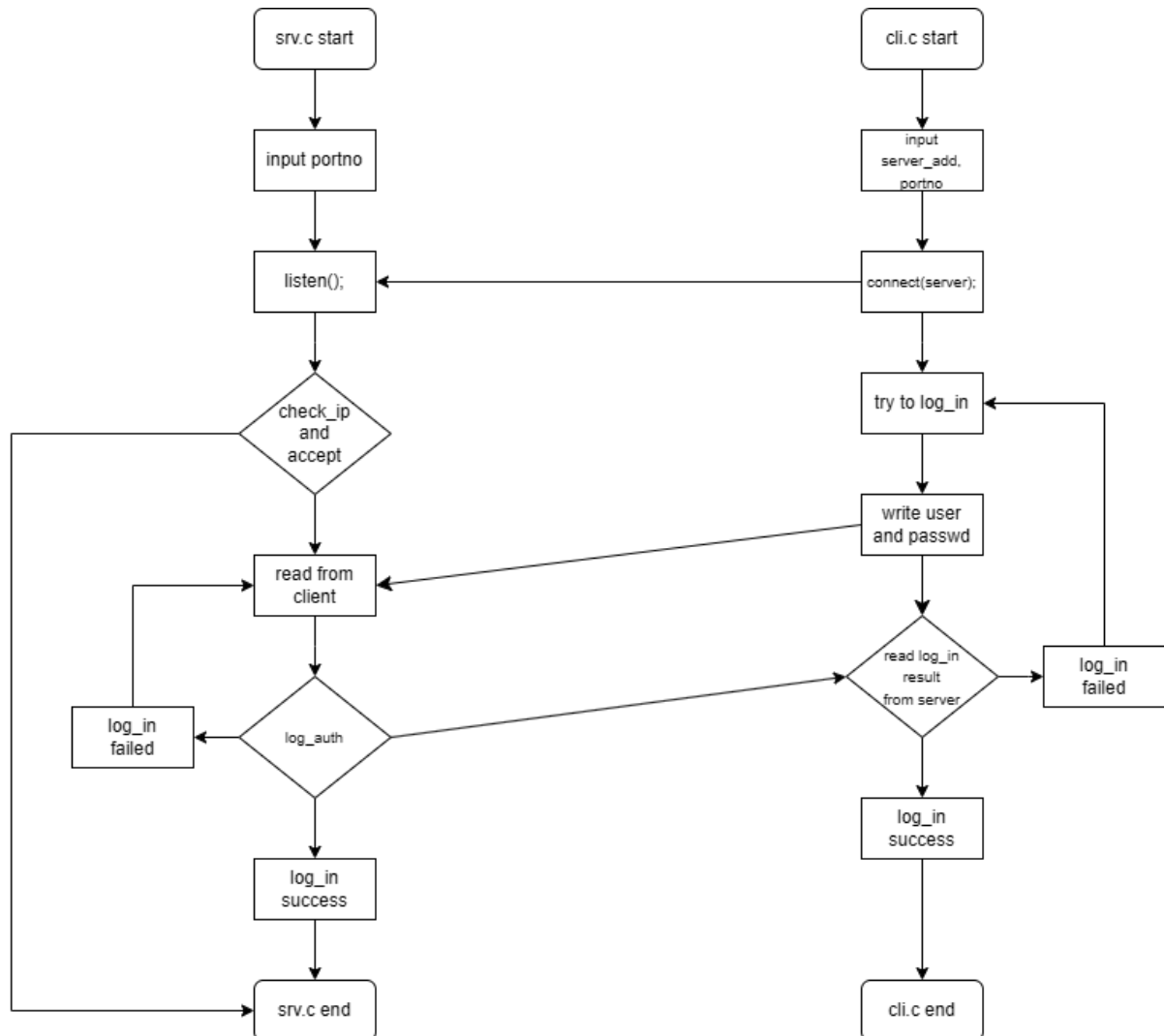
이번 Assignment 3-1에서는 FTP 서버의 사용자 로그인 서비스를 구현해보는 것이 목표이다. 사용자는 올바른 포트번호로 서버에 접속을 시도하면 서버에선 해당 클라이언트의 ip 를 화이트리스트와 비교해보고 (check_ip) 리스트 내에 있으면 접속을 허용한다. 이후 로그인 시퀀스를 진행하며 사용자가 입력한 정보가 passwd 파일 내에 있을 경우 로그인에 성공하고, 총 3 회 로그인에 실패할 시 서버로부터 연결이 종료된다.

클라이언트에선 log_in 함수를 통해 서버의 디스크립터로 유저 이름과 비밀번호를 전송하며 이 때 터미널에 정보가 남지 않도록 getpass 함수를 사용해서 입력을 받는다. 이후 서버로부터 응답에 따라 로그인 횟수를 기록하며 3 회 실패할 경우 서버와 연결이 종료되었음을 알린다.

서버에선 접속하려는 클라이언트의 ip 를 access.txt 파일에서 찾고, 있다면 접속을 허용하고 없다면 접속을 거부한다. 이 때 check_ip 함수를 사용하여 화이트리스트 파일을 검색하고 와일드 카드 여부를 적용하여 해당 ip 의 접속 가능 여부를 확인한다. 접속이 가능하다면 로그인 시퀀스를 진행하며 유저로부터 받은 유저 이름과 비밀번호를 passwd 파일에서 검색해 비교한다. 만약 로그인을 3 회 실패했다면 클라이언트로 연결을 종료한다는 메시지를 전송한다. 로그인에 성공했으면 성공했다는 메시지를 출력한다.

간단한 로그인 및 인증 시퀀스를 수행할 수 있는 시스템을 부분적으로 구현해보고 테스트해보는 것이 이번 Assignment 3-1 의 목표이다.

Flow chart



클라이언트에서 로그인을 시도하면 서버에서 로그인 결과를 전송한다. 총 3 회 로그인에 실패하면 서버는 클라이언트와의 연결을 종료한다.

Pseudo code

cli.c

define sockfd

set servaddr

connect(sockfd, servaddr)

log_in(sockfd)

log_in(int srvfd) {

define string user, passwd, buf

loop {

user = getpass("Input user : ")

passwd = getpass("Input passwd : ")

write(server, user)

write(server, passwd)

read(server, buf)

if(buf == "OK") print("*** Success to log-in **") return

else if(buf == "FAIL") print("Log-in failed **") continue

else if(buf == "DISCONNECTION") print("*** Connection closed **") return

}

srv.c

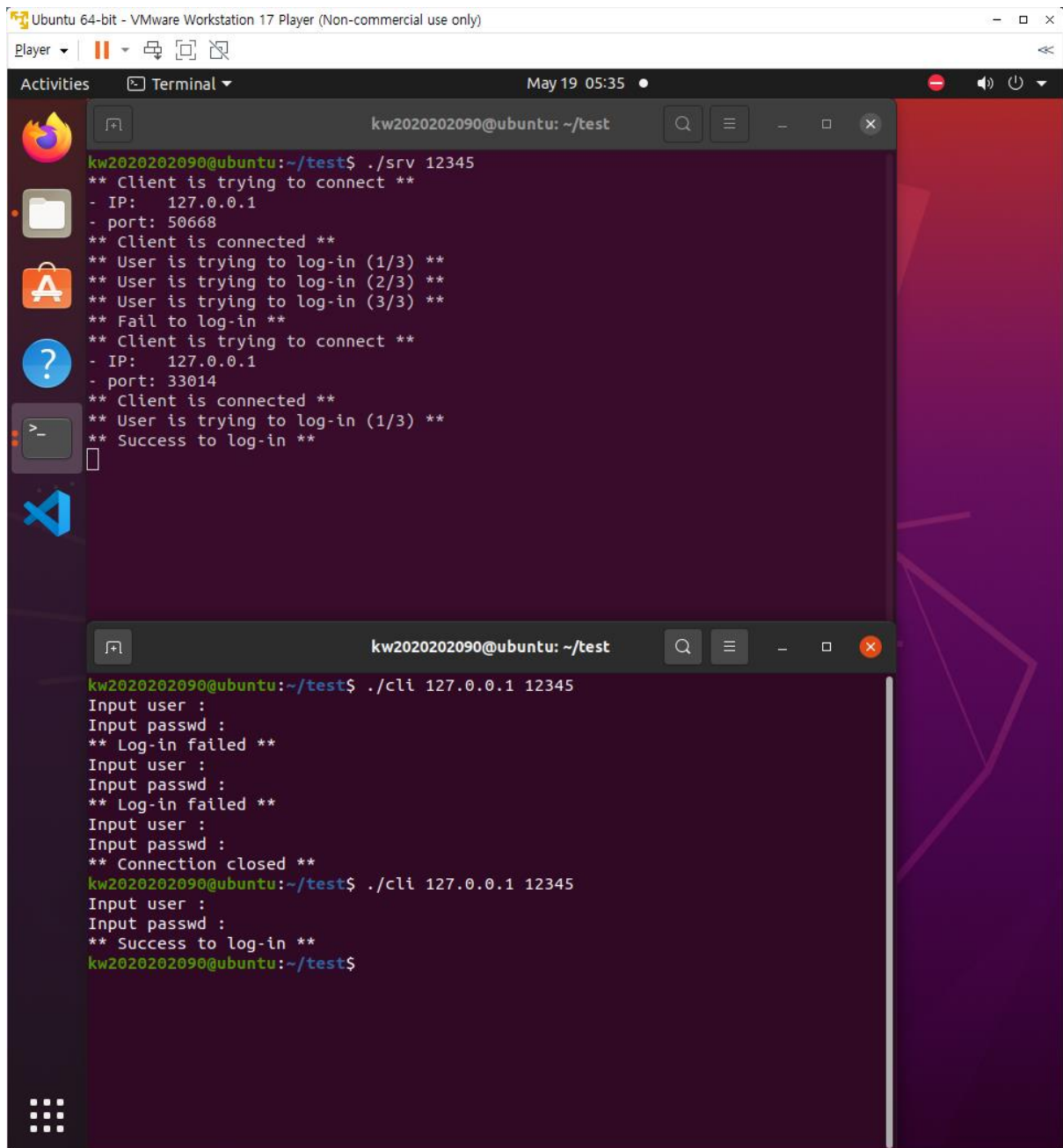
```
define listenfd, connfd
set cliaddr, servaddr
bind(listenfd, cliaddr)
loop {
    connfd = accept(listenfd)
    check_ip(cliaddr)
    log_auth(connfd)
    if(log_in_succesess) print("*** Success to log-in ***")
    else print("*** Fail to log-in ***")
}

function check_ip(sockaddr_in cliaddr) {
    open access.txt file
    print client info
    define string use
    loop {
        getline(access.txt, whitelist_ip)
        if(cliaddr.ip == whitelist_ip || cliaddr.ip == "") return true
        else return false
    }
}

function log_auth(int clifd) {
    define string user, passwd
    define int count = 1
    loop {
        print("** User is trying to log-in (%d/3) **", count)
        read(clifd, user)
        read(clifd, passwd)
        if(user_match(user, passwd) == true) write(clifd, "OK")
        else {
            if(count >= 3) write(clifd, "DISCONNECT"), return
            else write(clifd, "FAIL"), continue
        }
    }
}
```

```
function user_match(string user, string passwd) {  
    open passwd file  
    define string line  
    loop {  
        getline(passwd, line)  
        parse fuser, fpasswd from line  
        if(fuser == user && fpasswd == passwd) return true;  
        else return false;  
    }  
    close passwd file  
}
```

결과화면



```
kw2020202090@ubuntu: ~/test
kw2020202090@ubuntu:~/test$ ./srv 12345
** Client is trying to connect **
- IP: 127.0.0.1
- port: 50668
** Client is connected **
** User is trying to log-in (1/3) **
** User is trying to log-in (2/3) **
** User is trying to log-in (3/3) **
** Fail to log-in **
** Client is trying to connect **
- IP: 127.0.0.1
- port: 33014
** Client is connected **
** User is trying to log-in (1/3) **
** Success to log-in **

kw2020202090@ubuntu:~/test$ ./cli 127.0.0.1 12345
Input user :
Input passwd :
** Log-in failed **
Input user :
Input passwd :
** Log-in failed **
Input user :
Input passwd :
** Connection closed **
kw2020202090@ubuntu:~/test$ ./cli 127.0.0.1 12345
Input user :
Input passwd :
** Success to log-in **
kw2020202090@ubuntu:~/test$
```

첫번째에선 로그인에 3 회 실패하여 서버와의 연결이 종료되었다. 두번째에선 로그인에 성공하였다.

고찰

이번에 로그인 시퀀스를 구현할 때 `getpass` 함수를 이용하여 입력한 정보를 터미널에 남기지 않도록 하였다. 그런데 이 함수는 반환할 때 문자열이 아닌 포인터를 반환했기에 `strcpy` 함수를 이용해서 로그인 함수 내의 문자열에 옮겨주었다.

`passwd` 파일을 읽는 전용 함수가 있지만 패스워드 파일의 구성을 보고 파싱하기 쉬울 것 같아서 직접 파싱해서 사용하였다. `:`로 구분 되어있어 `split` 하기 쉬웠다.

아이피를 확인할 때 와일드카드 구현을 용이하게 하기 위해 전체적으로 아이피를 한번 비교하고, 아니라면 도트로 구분된 4 개의 부분을 `.`으로 `split` 하여 4 번 비교하고 모두 통과하면 1 을 반환하고, 한번이라도 `dismatch` 되었다면 0 을 반환하도록 구현하였다. 부분적인 와일드 카드는 구현에서 제외되었으므로 이 방식을 사용하면 아이피가 `access.txt` 에 있는지 간단하게 확인할 수 있었다.

제공된 스켈레톤 코드는 로그인을 시도하면 서버에서 먼저 OK 를 보내고 그 이후에 또 OK 를 보내면 로그인이 성공하는 것처럼 적혀있었지만, 이를 구분하기 위해 로직이 복잡해질 것 같아서 처음에 OK 를 보내는 것은 제외하고 먼저 서버에 유저 이름과 비밀번호를 입력한 뒤 OK 를 받아 로그인에 성공하는 것으로 구현하였다. 이후 스켈레톤 코드를 기반으로 `socket`, `connect`, `bind`, `accept` 의 예외처리를 구현하였고 필요한 `log_auth`, `user_match`, `check_ip` 모두 반환값으로 오류 여부를 알 수 있도록 하여 `main` 함수 및 서브루틴에서 오류가 발생하면 터미널로 전송할 수 있도록 하였다.

이전에 구현한 소켓+fork 기반 FTP 서버 코드에 이번 Assignment3-1 에서 구현한 로그인 인증 시퀀스, 앞으로 로그 작성 등을 모두 병합하면 완전한 FTP 서버를 구현할 수 있을 것이라고 생각한다.

Reference

- 강의자료만을 참고하였음