

시스템 프로그래밍 실습

FTP 2-1

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

Introduction

이번 Assignment 2-1에선 클라이언트와 서버를 별개의 프로세스로 실행하고 소켓 디스크립터를 이용해 클라이언트에서 FTP 명령어를 서버로 전송하고, 서버에서 처리한 FTP 명령어의 결과를 클라이언트로 전달해 출력할 수 있는 시스템을 구현하고 테스트해볼 것이다.

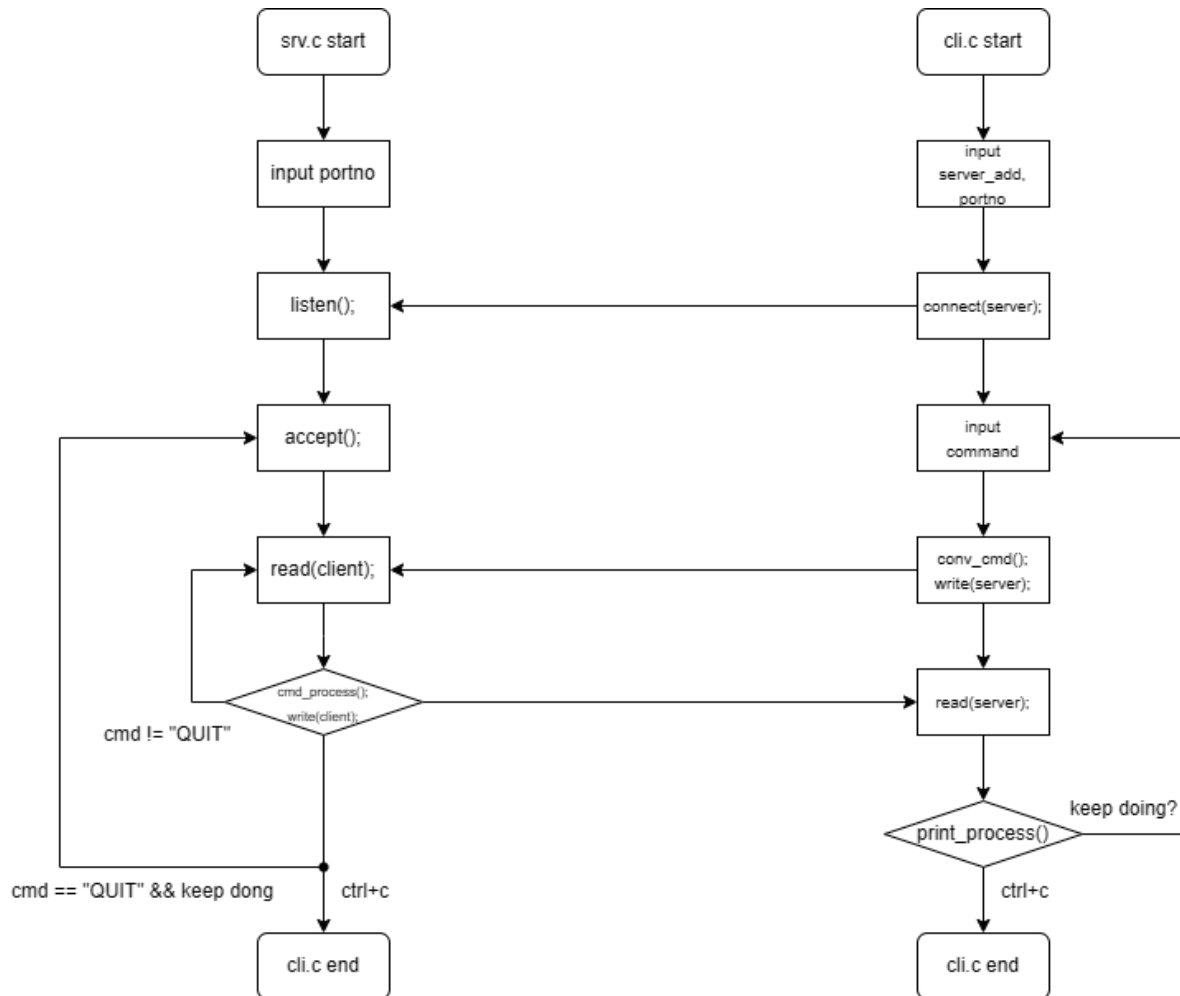
이전에 구현한 ls 명령어를 srv.c 에서 실행하고 결과를 생성한 뒤 이를 cli.c 로 전송하고, 클라이언트에서 수신한 정보를 출력한다. a, l 옵션을 포함한 ls 명령어와 프로세스를 종료하는 quit 명령어까지 2 개의 명령어를 서버 파일에서 구현한다.

클라이언트에서는 socket(), connect(), write() 시스템 콜을 사용해서 서버와 연결하고 메시지를 송수신한다. 유저에게 입력 받는 명령어를 적절한 FTP 명령어로 변환한 뒤 이를 서버로 전송하고 (write), 서버에서 생성한 결과를 출력한다.

서버에서는 socket(), bind(), listen() 시스템 콜을 사용해서 클라이언트와 연결하고 메시지를 송수신한다. 수신한 FTP 명령어에 따른 동작을 수행한 후 결과를 클라이언트로 전송한다. 변환된 명령어를 서버 프로세스에서 출력해야 하며 연결된 클라이언트의 IP, 포트 정보를 출력한다. 여기서 주소 정보는 시스템과 네트워크의 바이트 오더링 방식을 고려하여 inet_ntoa(), ntohs() 등을 사용하여 알맞게 변환한다.

위 기능을 독립적인 프로세스에서 수행할 수 있는 시스템을 구성하고 테스트하는 것이 이번 assignment2-1 의 목표이다.

Flow chart



서버를 실행하면 해당 포트에 클라이언트의 접속을 기다린다. (listen) 클라이언트는 서버의 주소와 포트를 입력해 실행하면 서버와 연결을 시도한다. (connect) 서버는 이 연결 시도를 받아들이고 연결한다. (accept) 이후 클라이언트에서 입력한 명령어는 ftp 명령어로 변환되어 서버에 전달되고 이에 상응하는 동작을 수행한다. (cmd_process) 그리고 이 결과를 클라이언트로 전송하면 클라이언트에서 이를 출력한다. (print_process) 이 과정을 서버는 QUIT 가 입력될 때 까지, 클라이언트는 프로세스를 종료할 때 까지 반복한다.

Pseudo code

- cli.c

```
define string buff, cmd_buff, rcv_buff
input(server_addr, port_number)
connect(server_addr, port_number)
loop {
    input(buff) // command
    conv_cmd(buff, cmd_buff)
    write(server, cmd_buff)
    read(server, rcv_buff)
    print_process_result(rcv_buff)

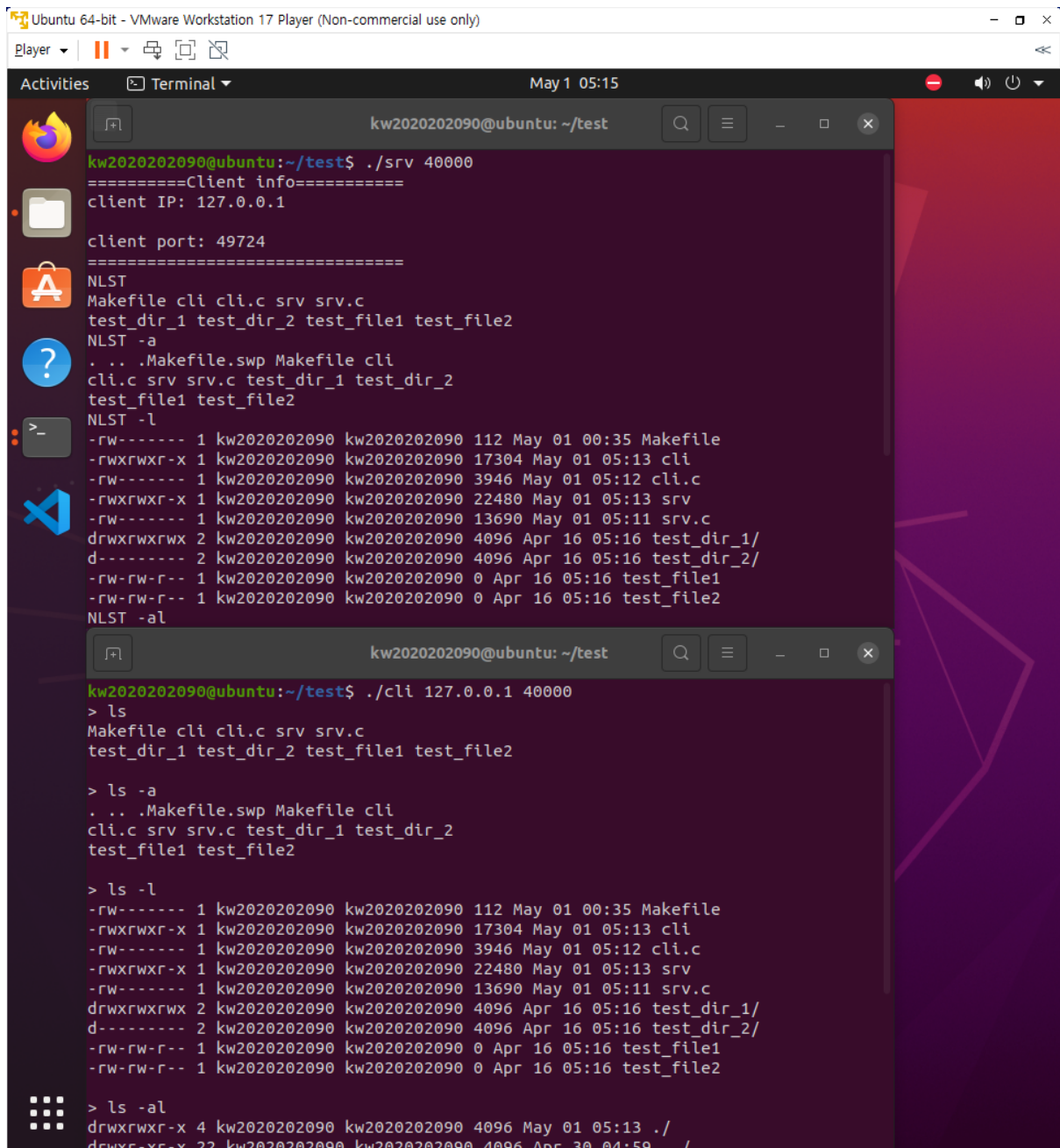
    clear(buff, cmd_buff, rcv_buff)
}
```

- srv.c

```
define string buff, result_buff
input(port_number)
bind(server_addr)
listen(5)

loop {
    accept(client)
    loop {
        read(client, buff)
        print(buff)
        cmd_process(buff, result_buff)
        if(result_buff == "QUIT")
            exit(0)
        clear(buff, result_buff)
    }
}
```

결과화면



```
kw2020202090@ubuntu: ~/test
kw2020202090@ubuntu:~/test$ ./srv 40000
=====Client info=====
client IP: 127.0.0.1

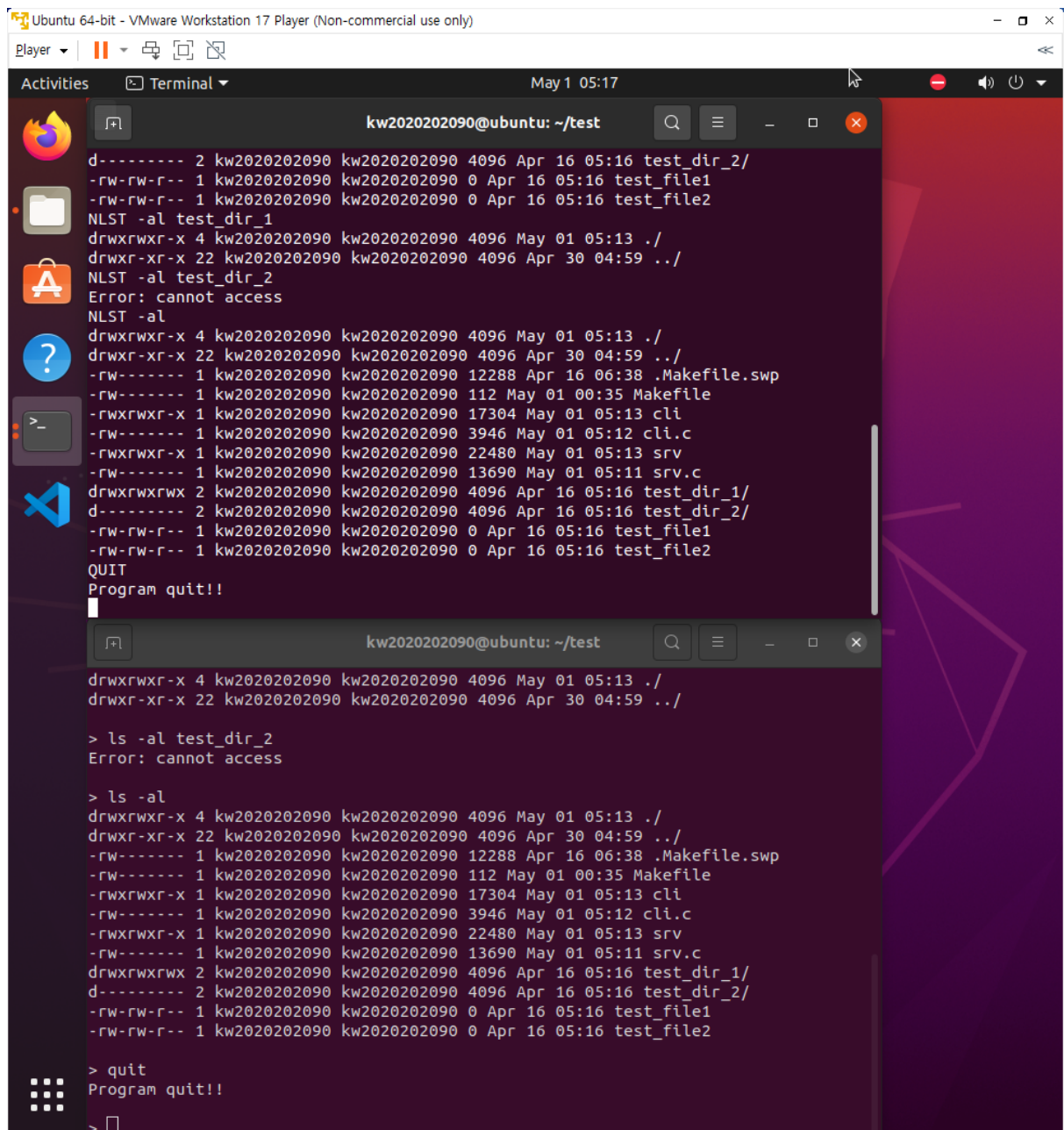
client port: 49724
=====
NLST
Makefile cli cli.c srv srv.c
test_dir_1 test_dir_2 test_file1 test_file2
NLST -a
. . . .Makefile.swp Makefile cli
cli.c srv srv.c test_dir_1 test_dir_2
test_file1 test_file2
NLST -l
-rw----- 1 kw2020202090 kw2020202090 112 May 01 00:35 Makefile
-rwxrwxr-x 1 kw2020202090 kw2020202090 17304 May 01 05:13 cli
-rw----- 1 kw2020202090 kw2020202090 3946 May 01 05:12 cli.c
-rwxrwxr-x 1 kw2020202090 kw2020202090 22480 May 01 05:13 srv
-rw----- 1 kw2020202090 kw2020202090 13690 May 01 05:11 srv.c
drwxrwxrwx 2 kw2020202090 kw2020202090 4096 Apr 16 05:16 test_dir_1/
d----- 2 kw2020202090 kw2020202090 4096 Apr 16 05:16 test_dir_2/
-rw-rw-r-- 1 kw2020202090 kw2020202090 0 Apr 16 05:16 test_file1
-rw-rw-r-- 1 kw2020202090 kw2020202090 0 Apr 16 05:16 test_file2
NLST -al
kw2020202090@ubuntu:~/test$ ./cli 127.0.0.1 40000
> ls
Makefile cli cli.c srv srv.c
test_dir_1 test_dir_2 test_file1 test_file2

> ls -a
. . . .Makefile.swp Makefile cli
cli.c srv srv.c test_dir_1 test_dir_2
test_file1 test_file2

> ls -l
-rw----- 1 kw2020202090 kw2020202090 112 May 01 00:35 Makefile
-rwxrwxr-x 1 kw2020202090 kw2020202090 17304 May 01 05:13 cli
-rw----- 1 kw2020202090 kw2020202090 3946 May 01 05:12 cli.c
-rwxrwxr-x 1 kw2020202090 kw2020202090 22480 May 01 05:13 srv
-rw----- 1 kw2020202090 kw2020202090 13690 May 01 05:11 srv.c
drwxrwxrwx 2 kw2020202090 kw2020202090 4096 Apr 16 05:16 test_dir_1/
d----- 2 kw2020202090 kw2020202090 4096 Apr 16 05:16 test_dir_2/
-rw-rw-r-- 1 kw2020202090 kw2020202090 0 Apr 16 05:16 test_file1
-rw-rw-r-- 1 kw2020202090 kw2020202090 0 Apr 16 05:16 test_file2

> ls -al
drwxrwxr-x 4 kw2020202090 kw2020202090 4096 May 01 05:13 ./
drwxr-xr-x 22 kw2020202090 kw2020202090 4096 Apr 30 04:59 ../
```

서버 및 클라이언트 실행, 서버 연결 및 ls, ls -a, ls -l 명령어를 실행하였다. 서버에서 처리한 결과는 디버그를 위해 출력하였고 실제로는 버퍼를 통해 두 프로세스 간 데이터 전송이 이루어진다.



```
kw2020202090@ubuntu: ~/.test
d----- 2 kw2020202090 kw2020202090 4096 Apr 16 05:16 test_dir_2/
-rw-rw-r-- 1 kw2020202090 kw2020202090 0 Apr 16 05:16 test_file1
-rw-rw-r-- 1 kw2020202090 kw2020202090 0 Apr 16 05:16 test_file2
NLST -al test_dir_1
drwxrwxr-x 4 kw2020202090 kw2020202090 4096 May 01 05:13 ./
drwxr-xr-x 22 kw2020202090 kw2020202090 4096 Apr 30 04:59 ../
NLST -al test_dir_2
Error: cannot access
NLST -al
drwxrwxr-x 4 kw2020202090 kw2020202090 4096 May 01 05:13 ./
drwxr-xr-x 22 kw2020202090 kw2020202090 4096 Apr 30 04:59 ../
-rw----- 1 kw2020202090 kw2020202090 12288 Apr 16 06:38 .Makefile.swp
-rw----- 1 kw2020202090 kw2020202090 112 May 01 00:35 Makefile
-rwxrwxr-x 1 kw2020202090 kw2020202090 17304 May 01 05:13 cli
-rw----- 1 kw2020202090 kw2020202090 3946 May 01 05:12 cli.c
-rwxrwxr-x 1 kw2020202090 kw2020202090 22480 May 01 05:13 srv
-rw----- 1 kw2020202090 kw2020202090 13690 May 01 05:11 srv.c
drwxrwxrwx 2 kw2020202090 kw2020202090 4096 Apr 16 05:16 test_dir_1/
d----- 2 kw2020202090 kw2020202090 4096 Apr 16 05:16 test_dir_2/
-rw-rw-r-- 1 kw2020202090 kw2020202090 0 Apr 16 05:16 test_file1
-rw-rw-r-- 1 kw2020202090 kw2020202090 0 Apr 16 05:16 test_file2
QUIT
Program quit!!

kw2020202090@ubuntu: ~/.test
drwxrwxr-x 4 kw2020202090 kw2020202090 4096 May 01 05:13 ./
drwxr-xr-x 22 kw2020202090 kw2020202090 4096 Apr 30 04:59 ../

> ls -al test_dir_2
Error: cannot access

> ls -al
drwxrwxr-x 4 kw2020202090 kw2020202090 4096 May 01 05:13 ./
drwxr-xr-x 22 kw2020202090 kw2020202090 4096 Apr 30 04:59 ../
-rw----- 1 kw2020202090 kw2020202090 12288 Apr 16 06:38 .Makefile.swp
-rw----- 1 kw2020202090 kw2020202090 112 May 01 00:35 Makefile
-rwxrwxr-x 1 kw2020202090 kw2020202090 17304 May 01 05:13 cli
-rw----- 1 kw2020202090 kw2020202090 3946 May 01 05:12 cli.c
-rwxrwxr-x 1 kw2020202090 kw2020202090 22480 May 01 05:13 srv
-rw----- 1 kw2020202090 kw2020202090 13690 May 01 05:11 srv.c
drwxrwxrwx 2 kw2020202090 kw2020202090 4096 Apr 16 05:16 test_dir_1/
d----- 2 kw2020202090 kw2020202090 4096 Apr 16 05:16 test_dir_2/
-rw-rw-r-- 1 kw2020202090 kw2020202090 0 Apr 16 05:16 test_file1
-rw-rw-r-- 1 kw2020202090 kw2020202090 0 Apr 16 05:16 test_file2

> quit
Program quit!!

>
```

추가로 ls의 기능을 더 테스트하기 위해 접근 권한이 없는 디렉토리에 ls 명령어를 시도해봤고, 이후 quit를 입력하여 서버와의 연결을 종료하였다.

고찰

이번 과제에서 입력과 출력은 모두 버퍼를 통해서 이루어지기 때문에 버퍼의 정크 데이터가 출력되지 않도록 매 송수신마다 이를 매번 비워주는 것이 중요하다. `bzero` 함수를 사용하여 바이트 단위로 0 으로 초기화해주었다.

프로세스에서 메모리를 잘못 참조할 때 발생하는 `segmentation fault` 에러를 해결하기 위해 `gdb` 를 이용하여 디버깅을 수행하였다. 주로 발생한 원인은 문자열을 함수의 매개변수로 줬을 때 `NULL` 인 문자열 포인터를 참조하여 발생하는 것이었다.

그리고 코드를 구현하고 수정하면서 겪었던 문제가 또 있는데, 이미 죽은 프로세스의 서버 주소 점유 문제였다. `segfault` 에러가 나면 프로세스가 강제 종료되는데, 이 동안 `srv` 의 프로세스는 좀비 상태가 되어 사용할 순 없지만 테스트를 위해 사용하는 주소를 점유하여 일정 시간동안 해당 주소를 사용할 수 없게 되는 문제점이 있었다.

물론 OS 의 프로세스 클리너가 주소를 점유하는 죽은 `srv` 프로세스를 자동으로 청소하여 1 분 내로 주소가 다시 사용 가능한 상태가 되었지만 그래도 수정한 코드를 바로 실행해볼 수 없다는 문제점이 있었다. `segmentation fault` 가 발생하여 프로세스가 종료되었을 때 이를 찾거나 찾아서 제거할 수 있는 방법을 찾는다면 수정 및 디버그 과정을 더 빠르게 수행할 수 있을 것 같다.

이번에 구현한 서버-클라이언트 시스템에선 서버 프로세스가 하나뿐이라 한 번에 최대한 개의 클라이언트에게 데이터 전송을 제공할 수 있었는데, `fork()` 시스템 콜을 사용해 서버의 프로세스 자체를 복제하여 처리할 수 있도록 하면 복수의 클라이언트로 데이터 전송을 서비스 할 수 있을 것이라고 생각한다.

Reference

- 강의자료만을 참고하였음.