

시스템 프로그래밍 실습

# FTP 2-3

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

# Introduction

이번 Assignment 2-3에서는 FTP 2-2에서 구현한 `fork()`를 통한 클라이언트 별 프로세스 처리, FTP 1-3에서 구현한 FTP 명령어들을 결합하여 완전한 FTP 프로세스를 수행할 수 있는 서버를 `fork()`를 이용하여 다수의 클라이언트를 대상으로 FTP 서비스를 제공할 수 있는 서버-클라이언트 시스템을 구현하고 테스트해볼 것이다.

클라이언트에선 이전에 구현한 `conv_cmd`를 사용하여 명령어와 필요한 `argument`들을 버퍼에 담아 서버로 전송할 수 있도록 한다. `quit`를 입력하거나, `Ctrl+C`를 입력하여 인터럽트 신호인 `SIGINT`를 발생시켰을 경우 서버로 클라이언트가 연결을 종료하도록 메시지를 전송한다. 이는 `SIGINT`에 대한 핸들러 함수를 구현하고 할당하여 사용한다.w

서버에선 이전에 구현한 `cmd_proces`를 사용하여 전송받은 명령어에 대한 FTP 명령어 동작을 수행하며 결과를 클라이언트로 전송하는 것은 동일하나, 각 클라이언트에 대해 `fork()`를 수행하여 `child process`에서 FTP 명령어를 수행하고, `parent process`에선 클라이언트들의 정보를 관리하는 형태로 구현한다.

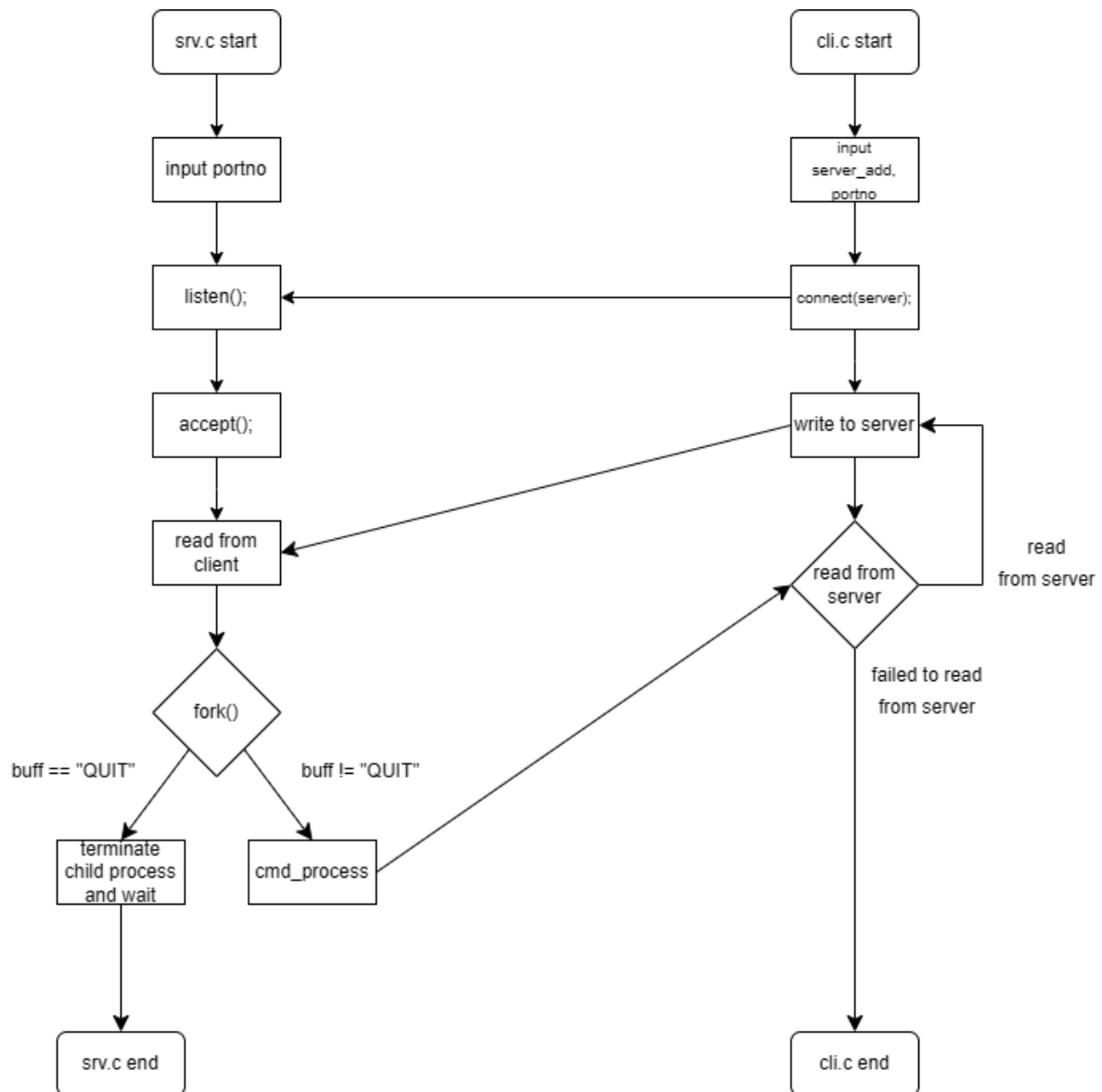
또한 `alarm`을 이용하여 10 초마다 각 클라이언트와 할당된 `child process`의 상태를 출력하도록 한다. `parent process`에서 `alarm`을 발동시킨 후 `SIGALRM`의 핸들러에서 위 내용을 수행할 수 있는 코드를 구현한다.

서버에서 `Ctrl+C`가 입력되어 `SIGINT`가 발생했을 경우 동일하게 핸들러를 사용하여 모든 클라이언트와 연결 해제하고 모든 `child process`를 종료하고 서버를 종료하는 절차를 밟도록 한다.

각 클라이언트 정보를 저장할 수 있는 구조체와, 이를 동적으로 관리할 수 있는 구조체 함수를 구현하여 다수의 클라이언트를 관리할 수 있도록 한다.

위 모든 기능을 합쳐서 동작할 수 있는 서버-클라이언트 구조의 FTP 시스템을 구현하는 것이 이번 Assignment2-3의 목표이다.

## Flow chart



서버를 실행하면 해당 포트로 클라이언트의 접속을 기다린다. 클라이언트는 서버의 주소와 포트를 입력해 실행하면 서버와 연결을 시도한다. 서버는 연결 시도를 받아들이고 연결한다. 이후 `fork()`로 해당 클라이언트를 처리하는 하위 프로세스를 생성하여 `cmd_process`를 수행하는 FTP 서버의 기능을 수행한다. 상위 프로세스에서는 각 클라이언트의 정보를 저장하고, alarm 핸들러를 이용해 10 초마다 현재 접속중인 모든 클라이언트의 정보를 출력할 수 있도록 한다. 서버에서 `SIGINT`가 발생했을 경우 모든 클라이언트와 연결을 해제하고 모든 하위 프로세스를 종료하고 서버를 종료할 수 있도록 한다.

# Pseudo code

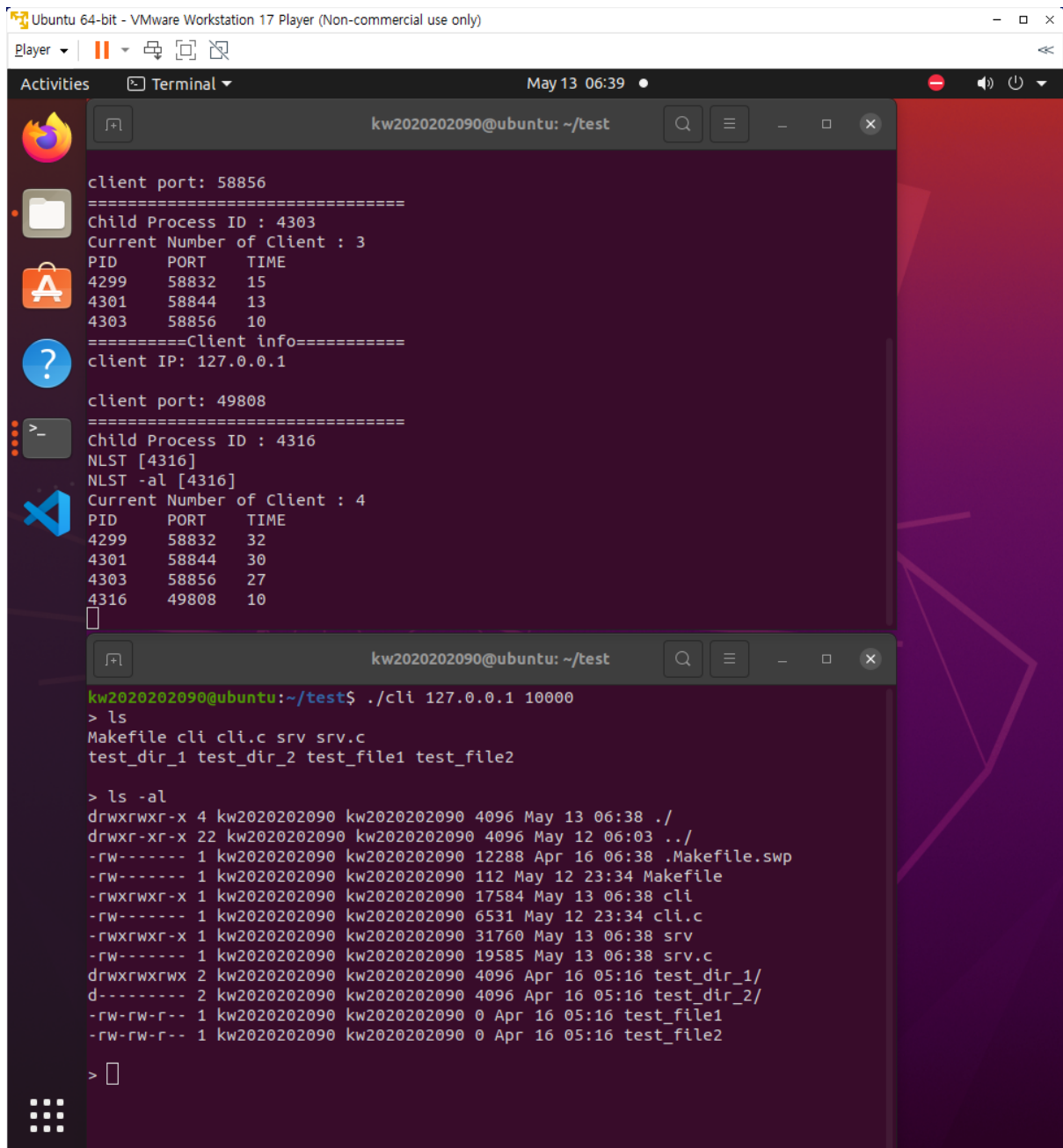
- cli.c

```
define string buff, rcv_buff
input(server_addr, port_number)
connect(server_addr, port_number)
loop {
    input(buff)
    write(server, buff)
    read(server, rcv_buff)
    printf("%s\n", rcv_buff);
    clear(buff)
}
```

- srv.c

```
define string buff, rst_buff
input(port_number)
bind(server_addr)
listen(5)
loop {
    accept(client)
    fork();
    if(child_process) {
        loop {
            client_info(client)
            printf("Child Process ID : %d\n", getpid());
            loop {
                read(client, buff)
                cmd_process(buff, rst_buff)
                write(client, rst_buff)
            }
            close(client)
        }
    }
    else // parent_process {
        cli_list.append(cur_process)
        alarm(10)
    }
}
```

# 결과화면



```
client port: 58856
=====
Child Process ID : 4303
Current Number of Client : 3
PID    PORT    TIME
4299   58832   15
4301   58844   13
4303   58856   10
=====Client info=====
client IP: 127.0.0.1

client port: 49808
=====
Child Process ID : 4316
NLST [4316]
NLST -al [4316]
Current Number of Client : 4
PID    PORT    TIME
4299   58832   32
4301   58844   30
4303   58856   27
4316   49808   10
[]

kw2020202090@ubuntu: ~/test
kw2020202090@ubuntu:~/test$ ./cli 127.0.0.1 10000
> ls
Makefile cli cli.c srv srv.c
test_dir_1 test_dir_2 test_file1 test_file2

> ls -al
drwxrwxr-x 4 kw2020202090 kw2020202090 4096 May 13 06:38 ./
drwxr-xr-x 22 kw2020202090 kw2020202090 4096 May 12 06:03 ../
-rw-r--r-- 1 kw2020202090 kw2020202090 12288 Apr 16 06:38 .Makefile.swp
-rw-r--r-- 1 kw2020202090 kw2020202090 112 May 12 23:34 Makefile
-rwxrwxr-x 1 kw2020202090 kw2020202090 17584 May 13 06:38 cli
-rw-r--r-- 1 kw2020202090 kw2020202090 6531 May 12 23:34 cli.c
-rwxrwxr-x 1 kw2020202090 kw2020202090 31760 May 13 06:38 srv
-rw-r--r-- 1 kw2020202090 kw2020202090 19585 May 13 06:38 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

> []
```

서버에서는 10 초마다 현재 연결된 클라이언트들의 정보와 포트 번호, 활성 시간을 출력한다. 클라이언트에서 명령어를 입력하면 FTP 프로세스를 수행해 다시 클라이언트로 보내 결과를 출력한다.

## 고찰

이번에는 2-2 의 fork()와 1-3 의 완전한 FTP 구현을 결합해 서버에서 다수의 클라이언트를 처리할 수 있는 컨커런트 서버를 구현하였다. alarm 핸들러를 이용해 10 초마다 현재 연결된 모든 클라이언트의 정보를 출력하고, 어떤 명령어가 어떤 프로세스에서 입력되었는지도 출력할 수 있다.

과제 조건으로 NLST 명령어에서 특정 파일에 대한 출력을 시도하는 코드는 삭제되었다. 이제 FTP 서버에서 NLST 명령어는 오직 디렉토리 파일에 대해서만 사용할 수 있다.

이제 서버의 기능을 완성하기 위해 서버에 접속할 수 있는 권한을 정하는 로그인이나 서버의 상태를 기록하고 디버깅에 사용할 수 있는 로그 작성 등을 적용할 수 있을 것 같다.

## Reference