

시스템 프로그래밍 실습

FTP 2-2

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

Introduction

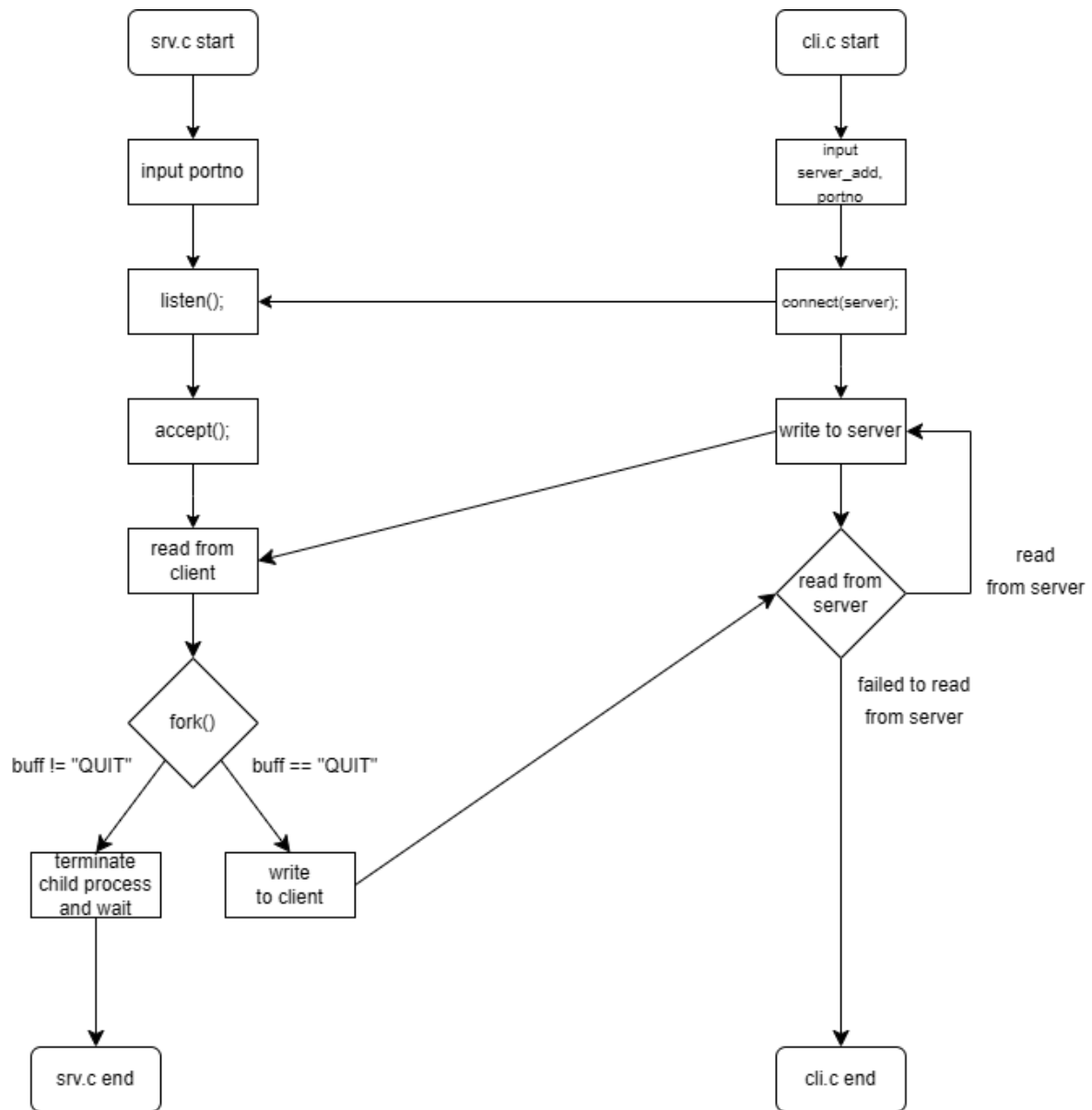
이번 Assignment 2-2에선 클라이언트의 요청을 서버에서 처리할 때 프로세스를 복제하는 `fork()` 시스템 콜을 이용해 다수의 클라이언트에게 서비스를 제공할 수 있는 시스템을 구현하고 테스트해볼 것이다.

이번엔 예제 코드를 활용하여 `srv.c`에서 `fork()`를 활용한 클라이언트의 요청을 처리하는 코드를 작성하였다. `fork()`는 동일한 프로세스를 생성하는데, `getpid()`를 통해 pid 값이 0일 때 child 프로세스의 동작을 수행하는 시퀀스를 작성하여 각 클라이언트의 요청을 처리할 수 있다.

`fork()`로 하위 프로세스를 생성하면 `SIGCHLD`가 발생하는데 이를 처리할 수 있는 핸들러 함수를 지정해주면 시그널이 발생할 때 마다 자동으로 처리할 수 있다. 동일한 방식으로, `SIGALRM`의 신호를 처리할 때 프로세스를 종료하도록 핸들러 함수를 지정해주었다. 하위 프로세스가 좀비 프로세스가 되는 것을 방지하도록 `SIGCHLD` 핸들러 함수에서는 `wait()`를 호출하여 종료 상태를 받을 수 있도록 한다.

실행 예제처럼 `QUIT` 외의 입력이 들어왔을 땐 `echo` 서버처럼 동작하도록 읽은 결과를 다시 클라이언트로 `write` 하고, `QUIT`가 들어왔을 땐 `alarm()`을 호출하여 서버의 하위 프로세스가 종료될 수 있도록 한다. 그렇게 되면 클라이언트에선 서버로부터의 응답이 없게 되므로 자동적으로 메인 루프를 탈출하여 프로그램을 종료하게 된다.

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 클라이언트로의 입력을 다시 출력하는 에코 서버의 기능을 수행한다. 클라이언트에서 QUIT 를 입력하면 알람을 호출한 뒤 시그널 핸들러에 의해 프로세스가 종료되고, 클라이언트는 서버로부터의 응답이 없어 자동으로 종료된다.

Pseudo code

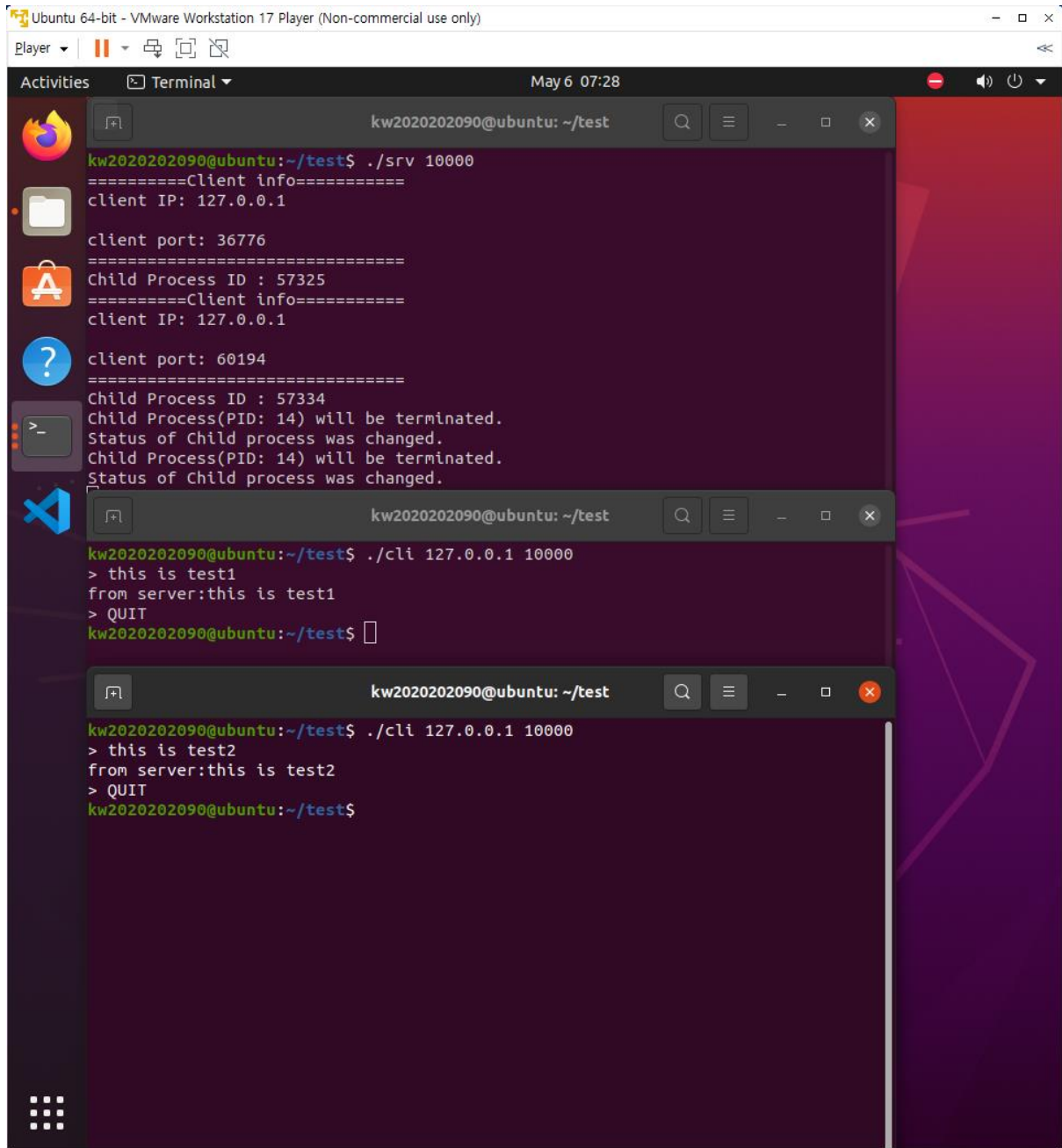
- cli.c

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

- srv.c

```
define string 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)
                if(buff == "QUIT") alarm();
                else write(client, buff); clear(buff)
            }
        }
        close(client)
    }
}
```

결과화면



```
kw2020202090@ubuntu: ~/test
kw2020202090@ubuntu:~/test$ ./srv 10000
=====Client info=====
client IP: 127.0.0.1

client port: 36776
=====
Child Process ID : 57325
=====Client info=====
client IP: 127.0.0.1

client port: 60194
=====
Child Process ID : 57334
Child Process(PID: 14) will be terminated.
Status of Child process was changed.
Child Process(PID: 14) will be terminated.
Status of Child process was changed.

kw2020202090@ubuntu:~/test$ ./cli 127.0.0.1 10000
> this is test1
from server:this is test1
> QUIT
kw2020202090@ubuntu:~/test$

kw2020202090@ubuntu:~/test$ ./cli 127.0.0.1 10000
> this is test2
from server:this is test2
> QUIT
kw2020202090@ubuntu:~/test$
```

2024-1_SPLab_FTP_Assginment2_2_v3 의 예제를 수행하였다. 복수의 클라이언트에게 에코 서버의 역할을 수행할 수 있고, QUIT 가 입력될 경우 해당 클라이언트를 관리하는 하위 프로세스는 종료된다.

고찰

fork()로 프로세스를 복제하였을 때 하위 프로세스의 pid 는 0 이다. 이를 이용해 하위 프로세스와 상위 프로세스가 수행할 시퀀스를 따로 지정해줄 수 있다. 이번 과제에선 if(pid==0) 조건문 안에 하위 프로세스가 수행할 동작들을 지정하고, 상위 프로세스에선 아무 동작도 수행하지 않게 함으로써 클라이언트가 들어올 때 마다 fork()를 통해 각 클라이언트별로 리퀘스트를 처리할 수 있도록 한다.

시그널 핸들러는 프로세스에 대하여 해당 시그널이 발생했을 때 핸들러를 수행하도록 지정하면 공통된 시퀀스에 대해 코드를 줄일 수 있다. 만약 하위 프로세스가 종료될 때 마다 wait 를 하기보다 종료될 때 시그널의 핸들러에서 wait 를 일괄적으로 걸어주는 것이 더 효율적이다.

이번에 작성한 것과 이전 2-1 에서의 코드를 가져오면 fork 를 통해 여러 클라이언트에게 동시에 ftp 서비스를 제공할 수 있는 서버를 구축할 수 있을 것이라고 생각한다.

Reference

- 강의자료만을 참고하였음.