시스템 프로그래밍 실습

[FTP_Assignment 2-2]

Class : **D**

Professor : **최상호 교수님**

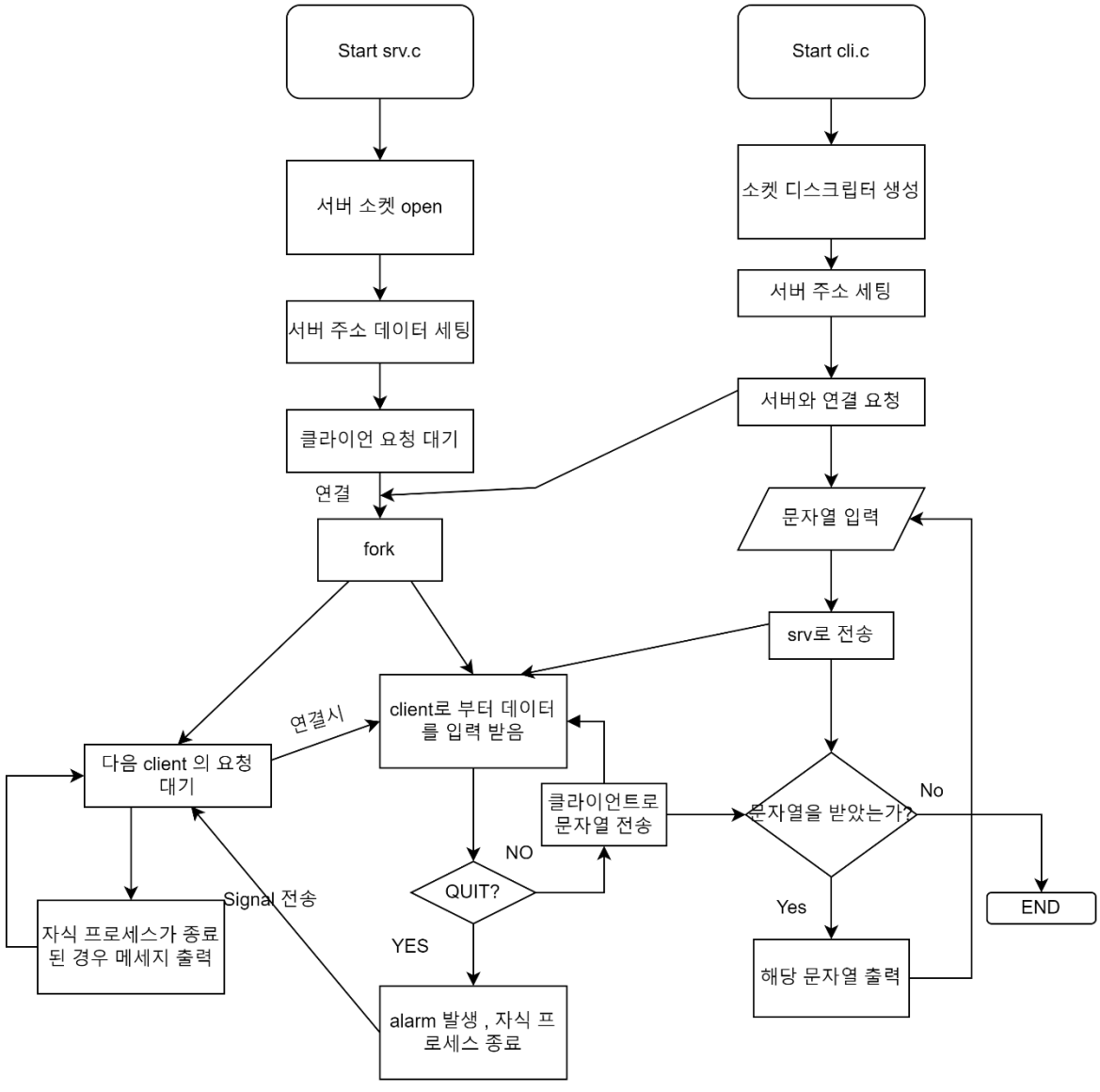
Student ID : **2020202037**

Name : **염정호**

Introduction

본 프로젝트는 fork 함수를 이용해 child process 를 생성하고 해당 process 에서 client 와 연결을 통해 문자열을 주고 받는 프로그램을 구현하는 것을 목표로 한다. 서버는 여러 명의 클라이언트와 문자열을 주고받을 수 있으며 fork 수행시 연결된 child process 의 포트 번호를 출력한다. 이후 자식 프로세스는 클라이언트와 연결되며 문자열을 주고 받는다. 클라이언트가 입력한 문자열이 QUIT 인 경우 해당 프로세스를 종료하며 서버에 해당 프로세스가 종료 됐음을 알린다.

Flow chart



Pseudo code

cli.c

```
int main(int argc, char **argv)
{
    char buff[BUF_SIZE]; // i/o buff
    int sockfd; // Socket descriptor
    struct sockaddr_in serv_addr; // socket address data
    sockfd = 소켓 디스크립터 설정
    socket addr 설정
    connect(서버와 연결 요청)

    while (1)
    {
        buff 초기화
        write(STDOUT_FILENO, "> ", 2);
        SRV 로부터 데이터를 입력

        if (소켓에 데이터 전송) //send input to srv
        {
            if (입력받은 데이터가 0 보다 큰경우)
                printf("from server:%s", buff); // 데이터 출력
            else
             0 보다 작은경우 반복문 종료
                break;
        }
        else
            데이터 전송 실패시 종료
             break;
    }
    close(sockfd); // close descriptor
    return 0;
}
```

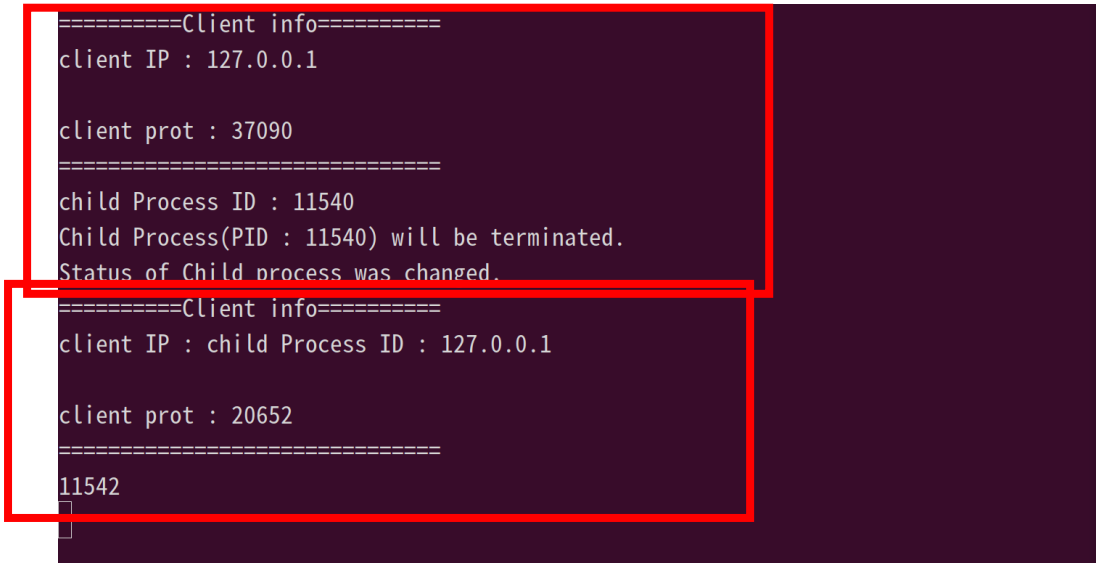
srv.c

```
int main(int argc, char **argv)
{
    void sh_chld(int); 자식 프로세스 종료시 함수 설정
    void sh_alrm(int); 알람 신호 생성시 함수 설정

    char buff[BUF_SIZE]; // i/o buffer
    int n;//
    struct sockaddr_in server_addr, client_addr; // server and client address
    int server_fd, client_fd; // file descriptor
    int len; //size of client address
    int port; // port number

    signal(SIGCHLD, sh_chld); 자식프로세스 종료시 해당 함수 실행
    signal(SIGALRM, sh_alrm); alrm 발생시 해당 함수 실행

    server_fd = 서버 디스크립터 할당

    memset(server_addr 초기화)
    server_addr 값 설정
    bind(디스크립터와 해당 주소를 bind); // bind

    listen(클라이언트의 입력 대기); // get requist

    while (1)
    {
        char PORT_string[10];
        PORT_string[0] = '₩0';
        pid_t pid;
        len = sizeof(client_addr);
        client_fd = 입력을 요청한 서버의 파일 디스크립터 저장

        if ((pid = fork()) < 0)
        {
        }
```

```
        else if (pid == 0)
        {
            sleep(1); 출력형태를 위해 1 초 대기했다가 출력
           write(child 프로세서와 연결시 해당 프로세서의 아이디 출력);
            write(STDOUT_FILENO, "₩n", 1);

            while (1)
            {
                if (클라이언트로부터 받은 데이터를 읽어옴)
                {
                    if 받은데이터가 QUIT 가 아닌 경우)
                        write(해당 문자열을 클라언트로 전송);
                    else
                        alarm(1); // 종료신호 발생
                }
            }
        }
        else

            부모프로세스 에서는 클라이언트의 정보를 출력
        close(client_fd);
    }
    return 0;
}
void sh_chld(int signum)// 자식프로세스가 종료된 경우 알림
{
    printf("Status of Child process was changed.₩n"); //print
    wait(NULL);
}

void sh_alrm(int signum) //print alarm
{// QUIT 가 발생해 알람이 발생한 경우 종료를 알리고 해당 프로세스 종료
    printf("Child Process(PID : %d) will be terminated.₩n", getpid());
    exit(1);
}
```

결과화면

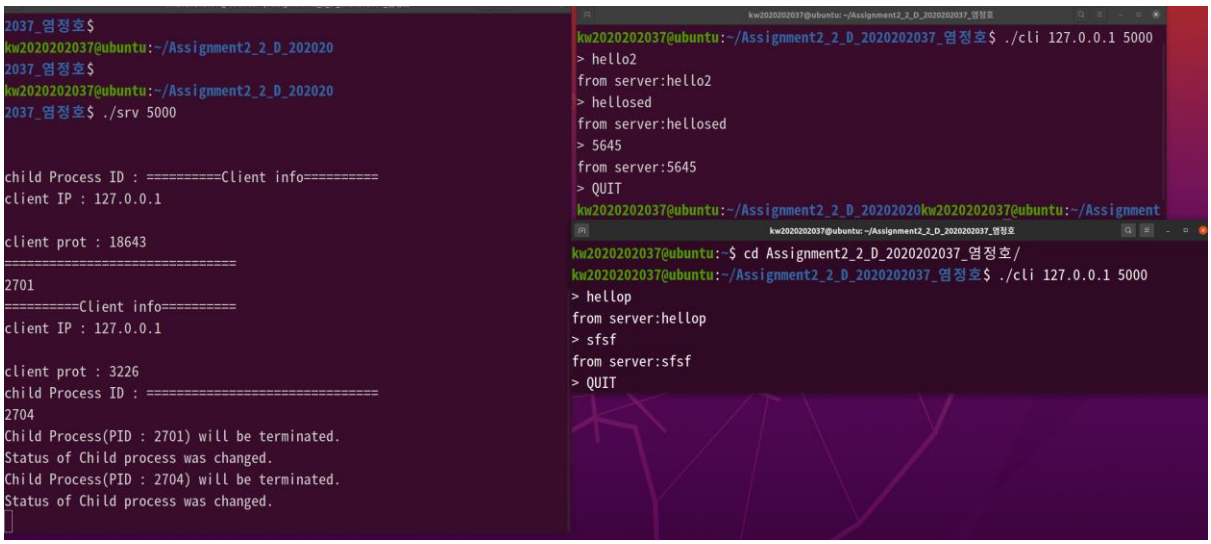
Server



1. 클라이언트와 연결 후 첫번째 자식 프로세스 생성
2. quit 입력시 해당 자식프로세스 종료 메시지 출력
3. 새로운 client 입력 후 자식 프로세스 생성

client

1. 서버와 연결 후 입력한 문자열을 그대로 서버로부터 전달 받음
2. quit 입력 시 종료
3. 같은 서버로 다시 연결



여러 명의 클라이언트와 연결시

고찰

해당 프로젝트를 처음 수행할 때 fork 수행후 부모 프로세서에서는 당연하 wait 를 수행해서 기다려야 한다고 생각했다. 왜냐하면 하나의 프로세서에서 다른 프로세서와 연결할 경우 해당 프로세서의 입력갑이 모든 프로세서로 들어간다고 생각했기 때문이다. 다만 부모 process 에서 wait 를 수행할 경우 다중 사용자 연결이 불가능 하는 딜레마가 발생했다. 이에 대한 해결책은 accept 였다. fork 를 이용해 child 프로세서를 생성하고 해당 프로세서를 개별적으로 if 문 안에서 반복문을 돌린다면 해당 프로세서에서 다른 클라이언트를 받지 못하게 하고 alarm 을 이용해 프로세서를 종료 시킬 수 있었으며 부모 프로세서는 입력 받은 클라이언트 데이터를 출력 후 while 문의 최상단으로 가서 accept 를 수행하면 다른 프로세서의 입력이 들어올 때 까지 기다리는 것이 가능했다.