시스템 프로그래밍 실습

[Assignment2-2]

Class : **[D]**

Professor : **[최상호교수님]**

Student ID : **[2020202060]**

Name : **[홍왕기]**

Introduction

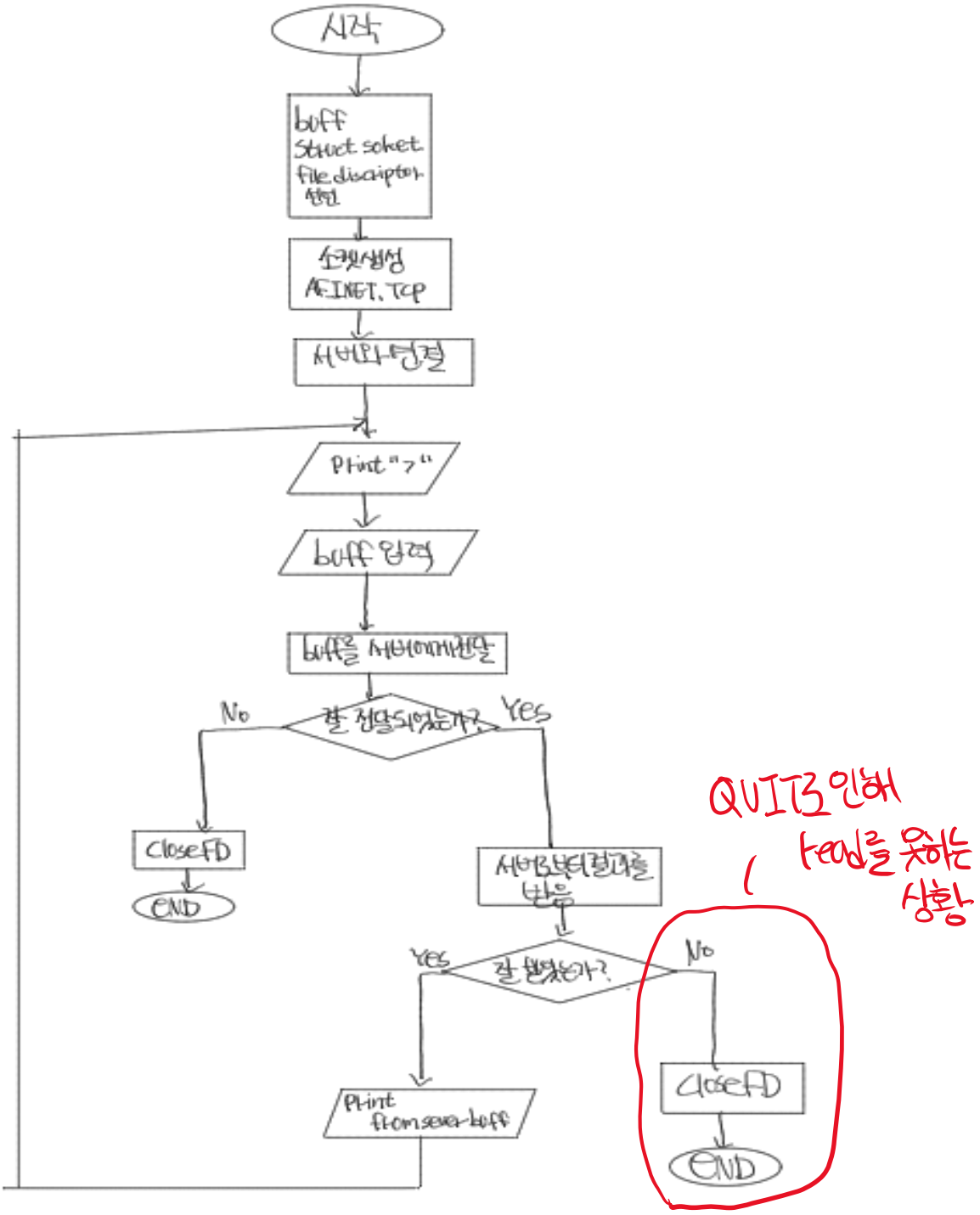
이는 네트워크 통신을 통해 클라이언트와 서버 간의 상호 작용을 용이하게 만들기 위해 소켓을 활용한다. 소켓은 컴퓨터 네트워크 상에서 프로세스 간 통신을 위한 인터페이스로 사용되며, 이를 통해 클라이언트와 서버는 데이터를 주고받을 수 있다.

서버는 여러 클라이언트로부터 동시에 요청을 받을 수 있다. 이러한 다중 클라이언트 요청을 처리하기 위해 서버는 각 클라이언트의 요청을 별도의 프로세스로 처리하여 병렬적으로 작업을 수행할 수 있게 해야 한다. 이를 위해 fork 함수를 사용하여 부모 프로세스에서 자식 프로세스를 생성한다. 각 자식 프로세스는 클라이언트의 요청을 독립적으로 처리하며, 이를 통해 서로 간섭 없이 작업을 수행할 수 있을 것이다.

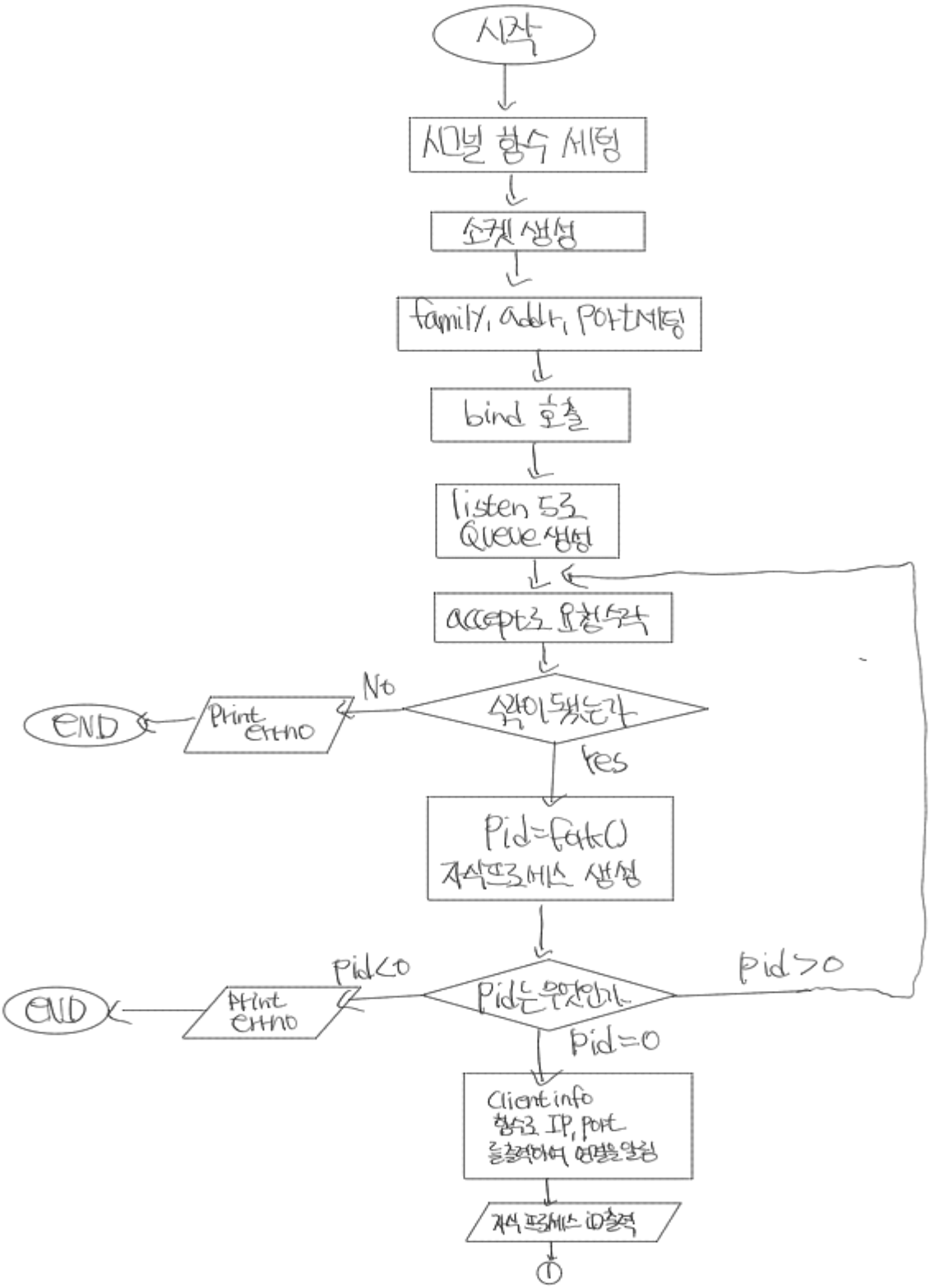
이러한 프로세스 기반의 다중 클라이언트 처리 방식을 통해 서버는 동시에 여러 요청자들의 요청에 대응하고, 효율적으로 작업을 수행할 수 있다. 따라서, fork 함수를 활용하여 여러 요청자를 동시에 처리하여 최종적으로 많은 요청자가 서버와 연결할 수 있게 하는 것이 목표이다.

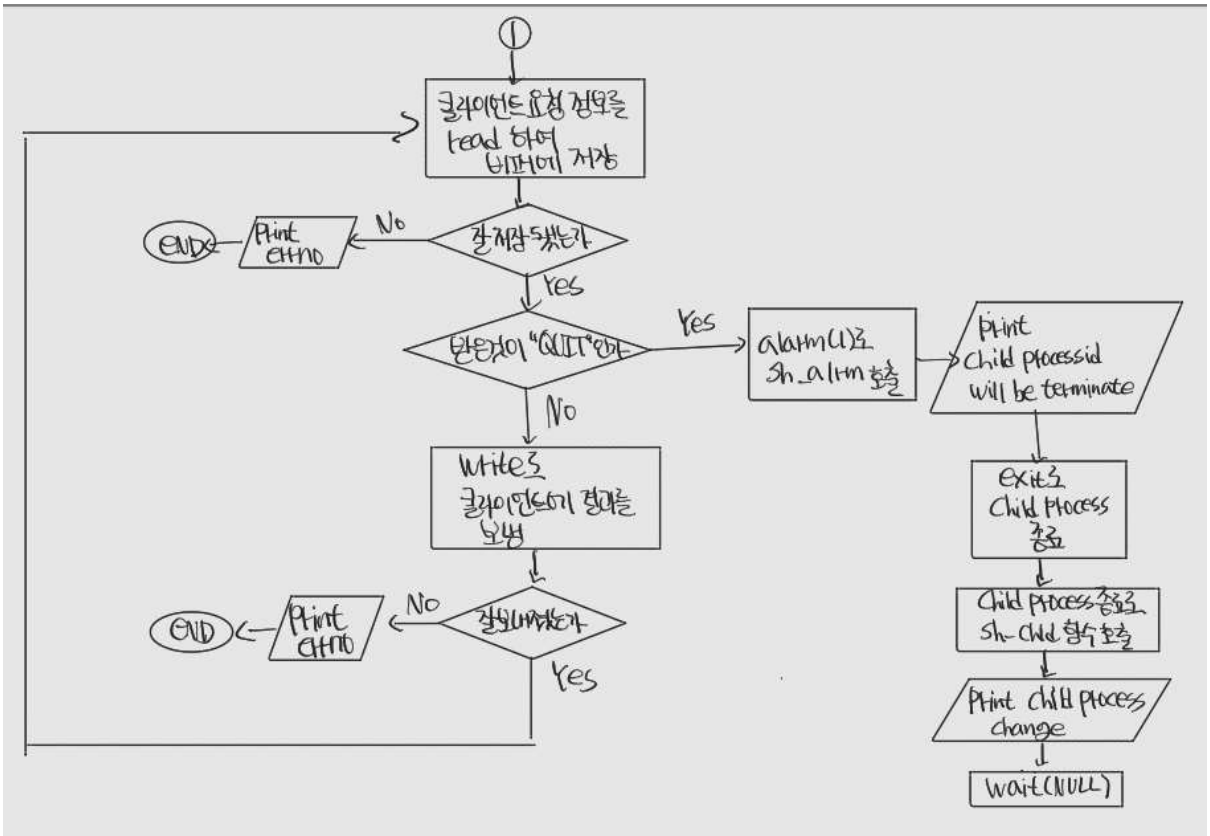
Flow chart

cli.c



Srv.c


시작
시그널 함수 세팅
소켓 생성
family, addr, port세팅
bind 호출
listen 및 Queue생성
accept로 요청수락
성공이 되었는가
No
Print errno
END
Yes
Pid=fork()
자식프로세스 생성
Pid는 양수인가
Pid<0
Print errno
END
Pid>0
Pid=0
Client info
함수로 IP, port
를 출력하여 연결을 알림
자식 프로세스 ID출력
①


클라이언트요청 정보를 read 하여 버퍼에 저장
잘저장되었는가
No
Print errno
END
Yes
받은것이 "QUIT"인가
Yes
alarm(1)로 sh_alrm 호출
Print Child processid will be terminate
No
write로 클라이언트에 정보를 보냄
exit로 Child Process 종료
Child Process 종료로 sh_child 함수 호출
잘보내졌는가
No
Print errno
END
Yes
Print Child process change
wait(NULL)

Pseudo code

Cli.c 필요 헤더파일 선언

```
#define BUF_SIZE 256
int main(int arg, char **argv){
    char buff[BUF_SIZE]

    //소켓이 필요한 요소 선언
    int n
    int sockfd
    struct sockaddr_in serv_addr
    sockfd = socket make this opt(AF_INET, SOCK_STREAM,0) ;

    //세팅
    family=AF_INET;
    addr= make binary addr // inet_addr(argv[1])
    port=host to network prot//thtons(atoi(argv[2]))
    connect Server
    while (1) {
        write ">"
        buffer input //버퍼 입력
        if(send buff to the server sucess){ //서버로부터 성공적으로 보냈다면
            recive buff to the network //서버로부터 결과를 받음
            if (recive buff success){ //결과를 잘 받았다면
                print "from sever buff" //결과 출력
            }
            else break
        }
        else  break
    }
        close socket discriptor
        return
}
```

Srv.c

include 필요한 헤더파일

```
int client_info(struct sockaddr_in *client_addr) {
    ip=inet_ntoa(client_addr->sin_addr)
    prot=ntohs(client_addr->sin_port)
    write(ip and port) -> like this

    ==========Client info===========
    client IP: 127.0.0.1
    client port: ??????
    ================================

    return
}

void sh_chld(int) // signal handler for SIGCHLD
void sh_alrm(int) // signal handler for SIGALRM
```

main

```
int main(int argc, char ** argv){

char buff[BUF_SIZE];
int n;
struct sockaddr_in server_addr, client_addr;
int server_fd, client_fd;
int len;
int port;

if Call the sh_alrm function when the alarm goes off
if Call sh_alrm function at the end of the child process

server_fd = socket make this->(PF_INET, SOCK_STREAM, 0)

   family=AF_INET;
   s_addr=htonl(INADDR_ANY);
   port=htons(atoi(argv[1]));

bind(setting)
listen(server_fd, 5)->Queue make
```

```
while(1){

    pid_t pid;
    len=sizeof(client_addr);
    client_fd= Accepted connection request from server_fd
    if (client_fd < 0) {
        errno and exit
    }

    pid = fork();
    if (pid < 0) {
      errno and exit
    }

   if (pid == child process) {
        Call the client_info function
        if (Call fail print errno and exit)

         print child process ID

        while (1) {
            Save to client request buff
            if (Save to client request buff fail){
                print errno and exit
            }
```

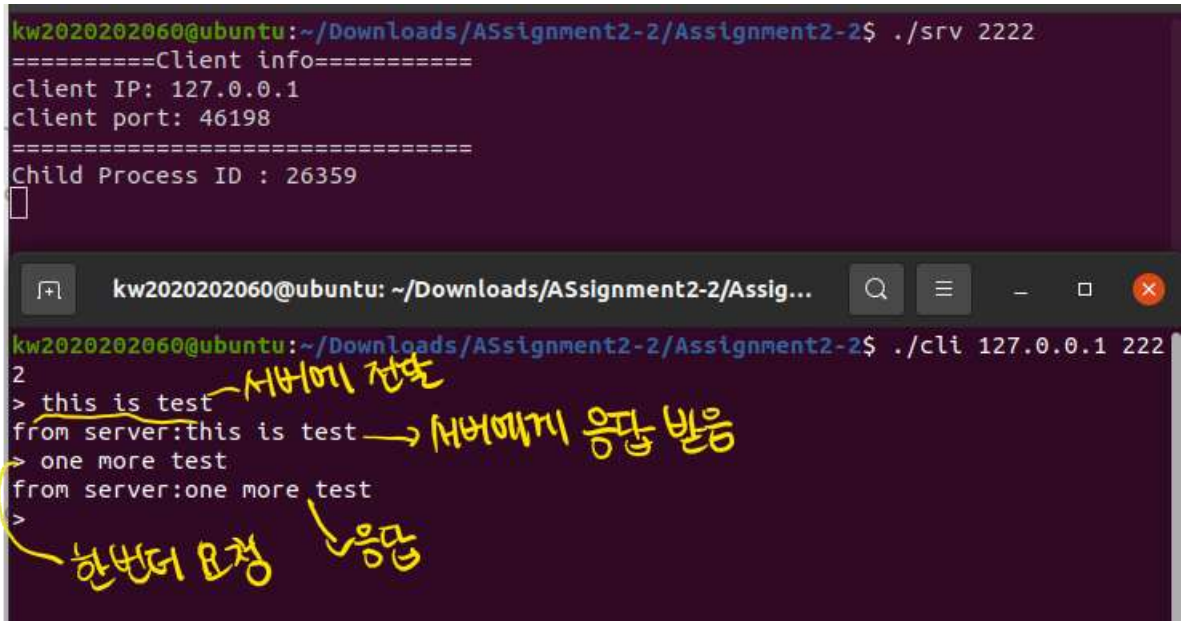
```
            if(buff!="QUIT"){
            Delivering results to the server
                if (Delivering results to the server FAIL) {
                print errno
                close File discriptor
                exit
            }
            }


            if(buff="QUIT"){
                alarm(1)
            }
      }
    }
    else {  // parent process


    }
    close(client_fd)
  }
  return 0
}
```

Main END

```c
void sh_chld(int signum){
    printf "Status of Child process was changed."
    wait(NULL)
}
void sh_alrm(int signum){
    printf "Child Process(PID: PID) will be terminated."
    exit(1);
}
```

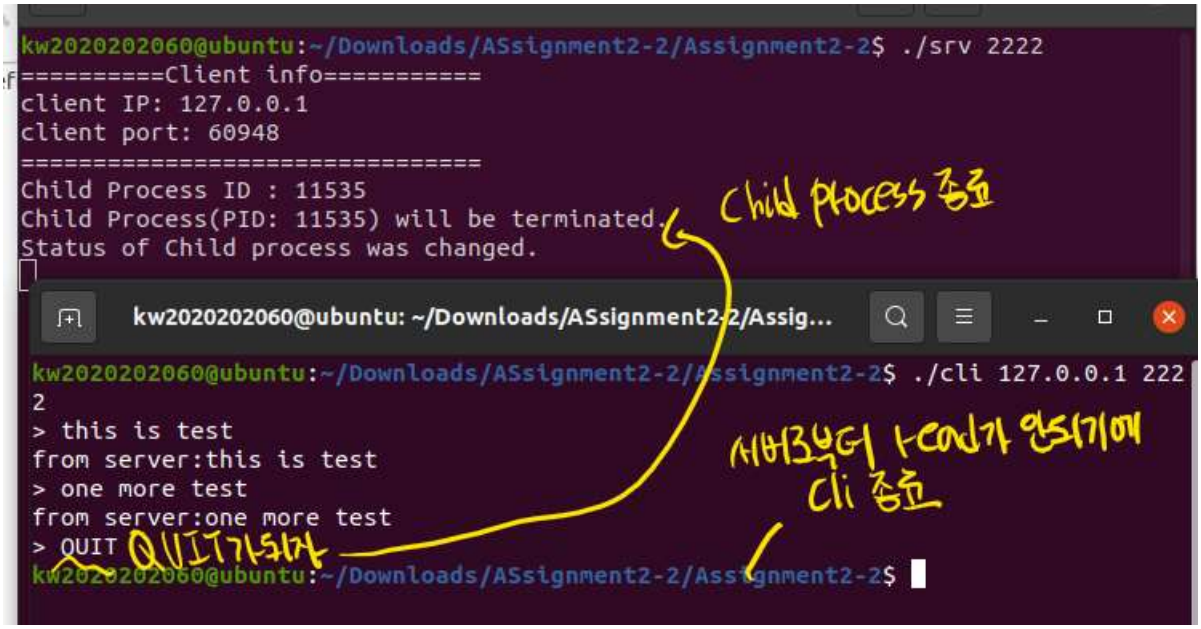
결과화면

위 설명과 함께 그림을 확인한다면 서버와 클라이언트가 잘 연결되었음을 확인할 수 있다. 이때 클라이언트는 명령요청을 준비하고 서버에서는 fork 가 실행됐기에 Child ID 가 출력됨을 확인할 수 있다. 아래는 클라이언트가 요청을 했을 때 서버가 동작을 한 후에 클라이언트로 다시 값을 보내는지 확인하는 검증이다.



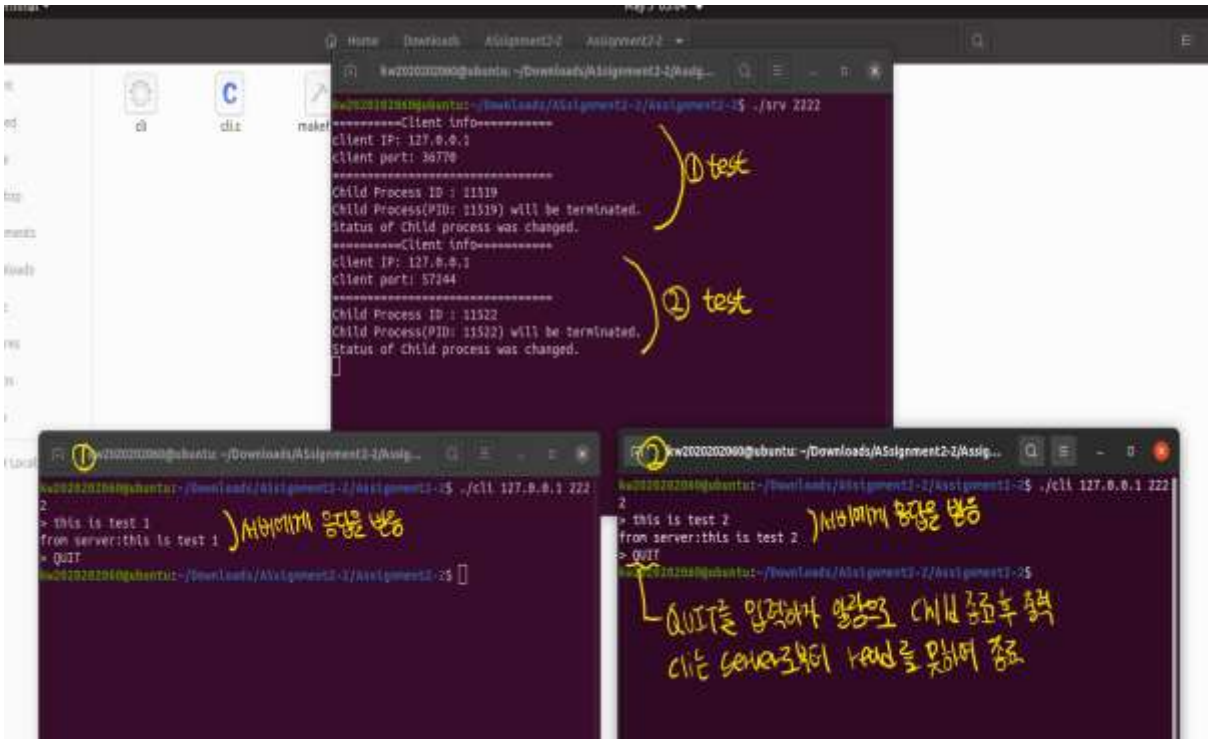
위 결과를 설명과 함께 확인한다면 클라이언트에 요청에 서버가 잘 응답했음을 확인할 수 있다.

다음은 QUIT 를 입력했을 때 Child Process 가 사라지게하고 클라이언트가 요청했을 때 응답해줄 서버가 없을 상황에 대한 검증이다.



위 결과를 설명과 함께 확인한다면 QUIT 를 입력하자 child 프로세서는 사라진 것을 시그널 함수를 통해서 확인할 수 있다. 그 후에 클라이언트는 서버로부터 받은 정보를 read 하지만 자식 프로세스는 종료되었기에 read 를 하지 못하여 종료된 것을 확인할 수 있다.

다음은 다수에 클라이언트가 연결되는 테스트 검증이다.



위를 설명과 함께 확인하면 클라이언트가 요청했을 때 연결 형태 문구와 요청에 대한 답을 잘 받음을 확인할 수 있다. 이때 QUIT 를 입력하면 Child 프로세스가 알람으로 인해 종료되어 서버에

대한 read 를 못하여 종료되는 것을 확인할 수 있다. 이때 다른 터미널에서도 연결을 하니 한 서버가 다수에 클라이언트 요청을 동시에 받을 수 있음을 확인할 수 있다.

고찰

Fork 함수를 공부하고 이를 통해 다중 클라이언트 요청을 처리하는 방법에 대해 직접 구현해본 결과 느낀 점이 많다. Fork 함수의 활용은 서버와 클라이언트 간의 상호작용을 효율적으로 만들어주었다. 클라이언트의 요청이 많을 때, 이를 병렬적으로 처리함으로써 빠른 응답과 서비스의 지속적인 제공하여 Fork 함수에 중요성을 느끼는 계기였다.

또한 Fork 함수를 이용하여 각 요청을 독립적으로 처리함으로써 서버의 안정성을 체감할 수 있었다. 각 클라이언트의 요청은 서로 영향을 주지 않으며, 따라서 한 요청의 오류가 다른 요청에 영향을 미치는 것을 방지하기에 꼭 필요한 요소라고 생각했다. 이로 인해 시스템 프로그래밍 지식에 이해도를 높여 추후 소프트웨어 개발에 큰 영향을 줄 것임을 확신한다.