시스템 프로그래밍 실습

[#2-2: cli & srv with fork]

Class : **[금요일 1, 2 교시]**

Professor : **[최상호 교수님]**

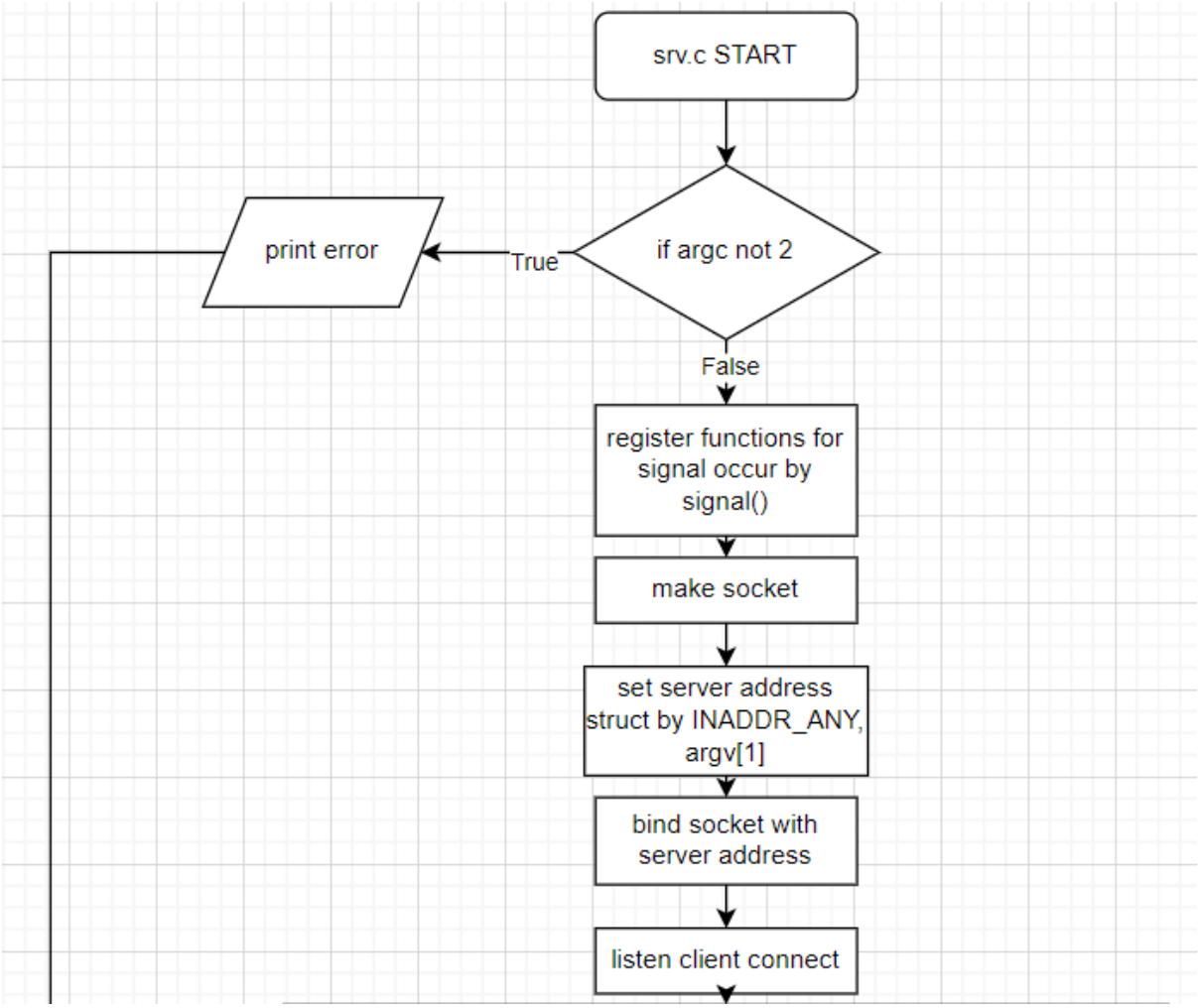
Student ID : **[2020202034]**

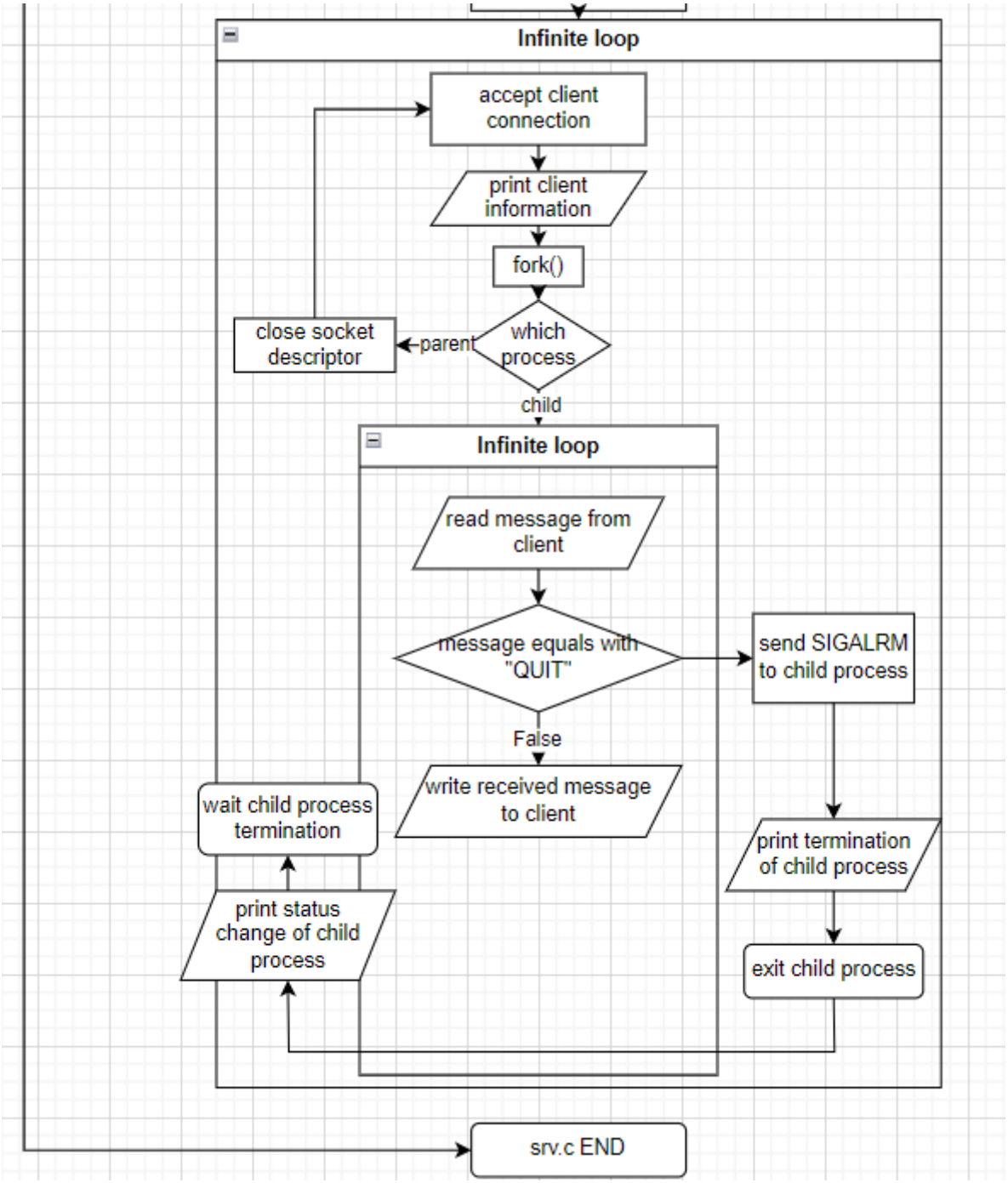
Name : **[김태완]**

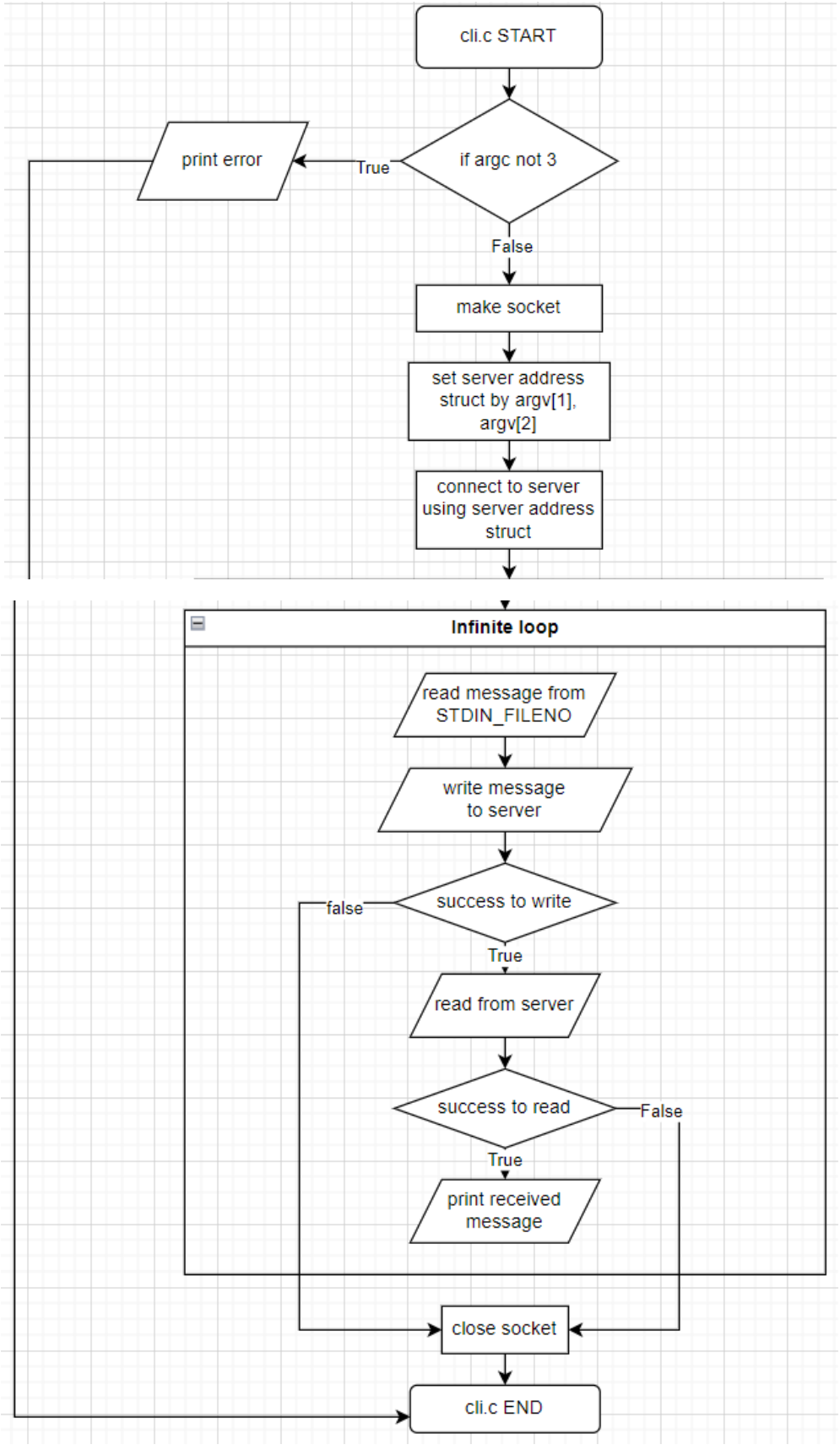
Introduction

이번 과제는 2-1 의 client 와 server 의 socket 통신에 fork() 함수를 추가해 multi-process server 를 만드는 과제입니다. 2-1 에서는 하나의 서버가 하나의 client 와만 통신해 server 의 process 가 하나여도 문제가 없었지만, 보통 서버는 여러 client 가 동시에 접속하여 사용하는 경우가 다반사입니다. 이 경우에 여러 client 의 요청을 처리할 수 있도록 server 에서 client 의 요청이 들어올 때마다 client 을 위한 process 를 할당해 줄 수 있습니다.

Flow chart




Infinite loop
accept client connection
print client information
fork()
which process
parent
close socket descriptor
child
Infinite loop
read message from client
message equals with "QUIT"
send SIGALRM to child process
False
write received message to client
print termination of child process
exit child process
wait child process termination
print status change of child process
srv.c END


cli.c START
if argc not 3
True
print error
False
make socket
set server address struct by argv[1], argv[2]
connect to server using server address struct
Infinite loop
read message from STDIN_FILENO
write message to server
success to write
false
True
read from server
success to read
False
True
print received message
close socket
cli.c END

Pseudo code

srv.c

```
Main(int argc, char **argv)
        if the number of arguments are not 1
                print error
                exit
        register sh_chld() which is called when SIGCHLD occurred
        register sh_alrm() which is called when SIGALRM occurred
        make socket for connection with client
        reset server address structure
        set server address family AF_INET
        set server address INADDR_ANY
        set server port argv[1]
        bind socket with server address structure
        wait client connection using listen()
        start Infinite loop(1)
                accept client connection
                store client address, port to client address structure
                print client information(address, port)
                make child process using fork()
                if child process:
                        print child process ID
                        start Infinite loop(2)
```

```
                                reset buff
                                read message from client
                                if read success:
                                        if message was "QUIT":
                                                send SIGALRM to child process
                                        write message to client
                        end of Infinite loop(2)
                else(== parent process):
                        close socket connected with client
        end of Infinite loop(1)
end of Main(int argc, char **argv)
sh_chld()
        print 'status of child changed'
        wait child's termination
end of sh_chld()
sh_alrm()
        print 'child(PID: xx) terminated'
        exit
end of sh_alrm()
```

cli.c

```
Main(int argc, char **argv)
        if the number of arguments are not 2
                print error
                exit
```

make socket for connection with server

reset server address structure

set server address family AF_INET

set server address argv[1]

set server port argv[2]

connect with server using server address structure

start Infinite loop

reset buffer

write "> "

read message from standard input

write message to server

if write success:

read from server

if read success:

print received message

else: break

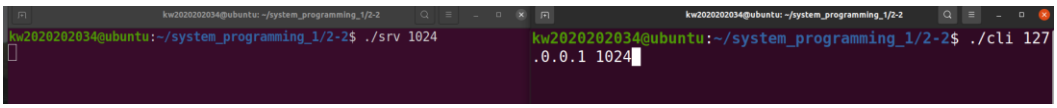
else: break

end of Infinite loop

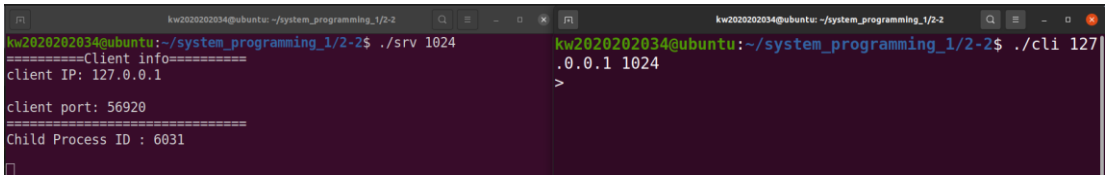
close socket

end of Main(int argc, char **argv)

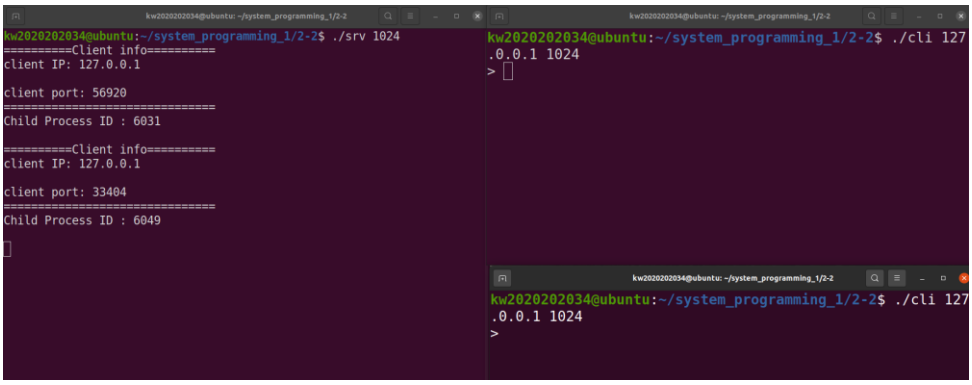
결과화면



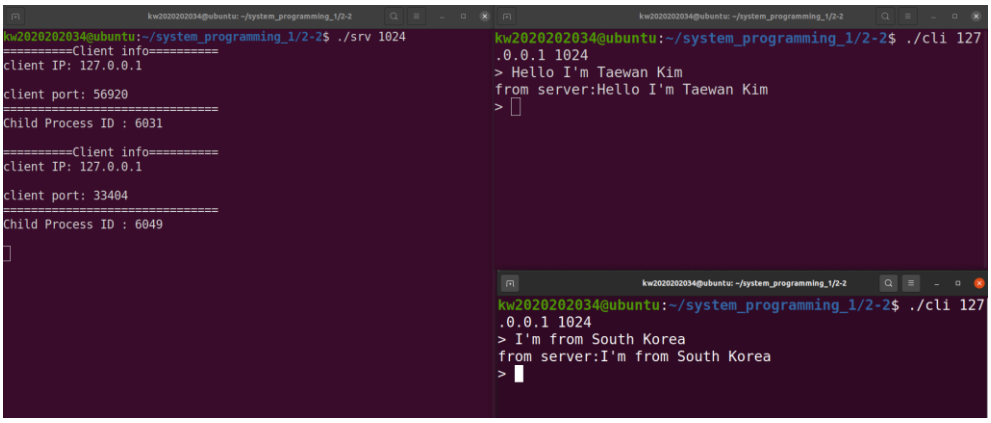
먼저 서버 프로그램에 인자로 포트 번호를 주어 실행시켰습니다. 이때 아직 client 는 connect 요청을 하지 않았기 때문에 서버는 아무 출력도 하지 않습니다.



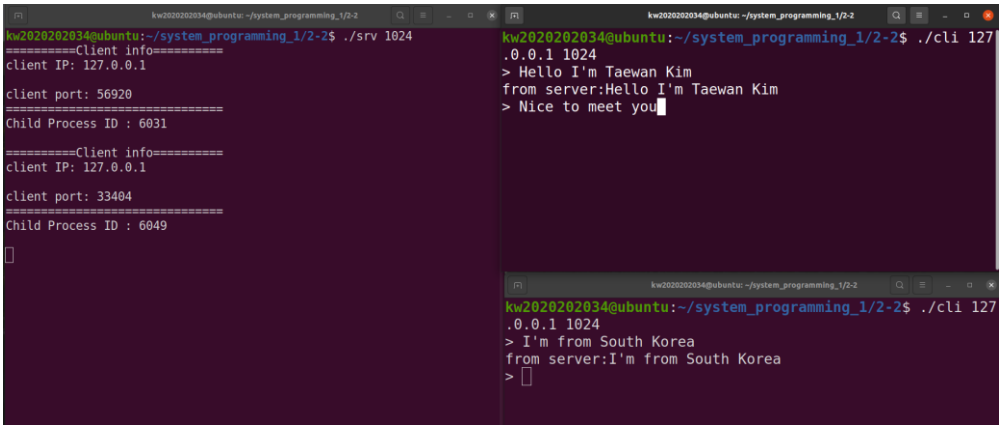
client 에서 connect 요청을 하니 서버와 클라이언트가 연결되고, 클라이언트와의 연결을 위한 child process 가 생성된 것을 볼 수 있습니다.



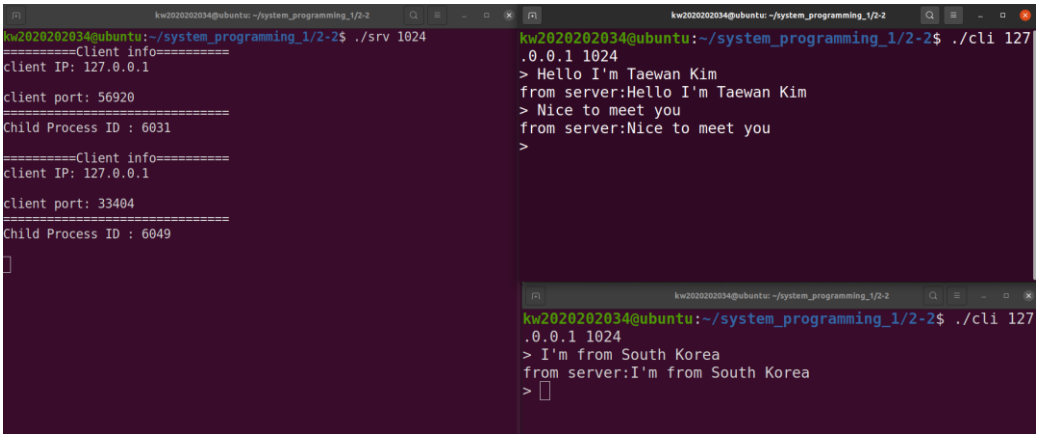
client 를 하나 더 실행해 서버에 connect 요청을 하니 새로운 클라이언트가 서버에 접속했음을 서버 화면을 통해 확인할 수 있습니다.



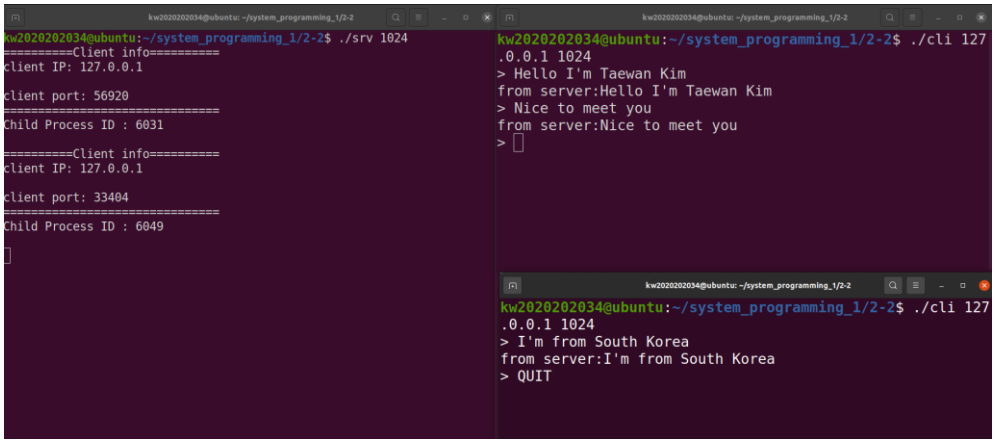
각 클라이언트에서 문자열을 보내니, 보낸 문자열이 다시 클라이언트로 전송되었음을 확인할 수 있습니다.



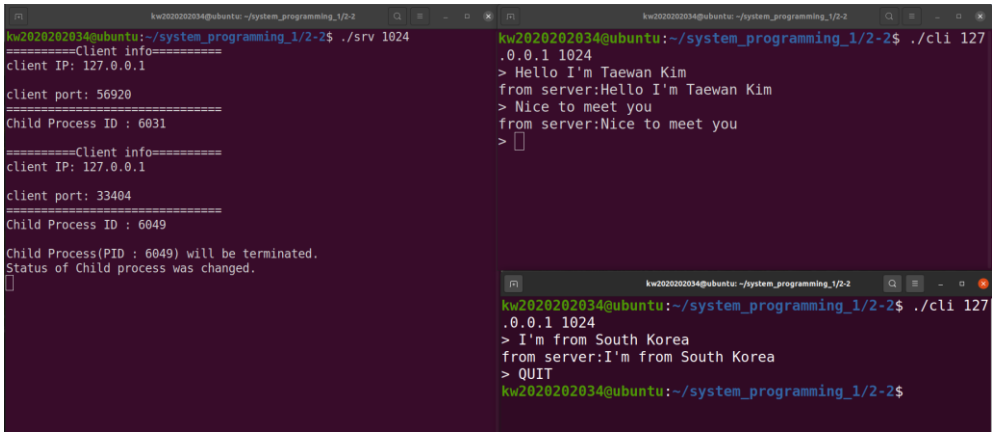
과정을 좀 더 자세히 살펴보면, 클라이언트에서 문자열을 위와 같이 입력하고 엔터를 누르면,



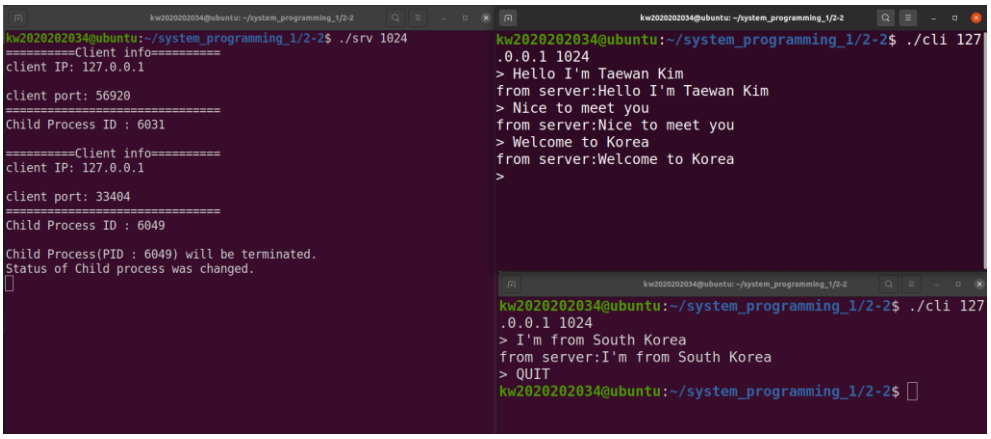
다음 줄에 서버로 전송한 문자열이 그대로 나옴을 볼 수 있습니다.



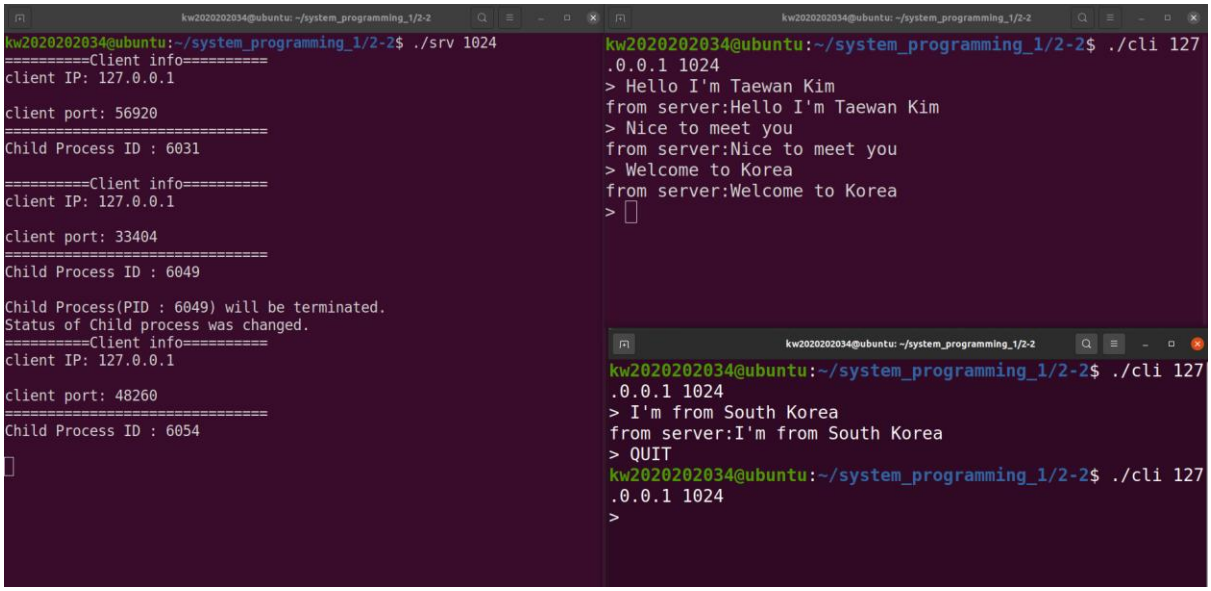
다음은 클라이언트를 종료시키겠습니다. 클라이언트에 QUIT 이라는 문자열을 입력하면,



client 가 종료되고, 서버에는 client 와 연결되었던 child process 가 terminated 되었고, 그로 인해 parent process 가 이를 인지했음을 알 수 있습니다.



한 클라이언트에서 서버와의 연결을 종료하더라도, 이는 해당 클라이언트와 연결된 server 의 child process 만 종료가 되는 것이기에 다른 클라이언트와 서버의 연결은 보시는 것(Welcome to Korea 를 입력하니 그대로 나옴)처럼 유지되고 있음을 확인할 수 있습니다.



또한, 클라이언트를 종료시킨 터미널에서 다시 클라이언트를 서버와 연결시키면, 새로운 client port 가 열려 연결이 되었고, 이 클라이언트와 연결된 child process 도 새로 생김을 확인할 수 있습니다.

고찰

2-1 의 cli, srv 를 거의 그대로 활용하고, fork() 개념만 추가하는 과제라 그리 어렵진 않았습니다. 그러나 처음에 child process 가 죽었음에도 client 의 read()가 0 을 반환하지 않고 여전히 서버의 출력을 기다리고 있어 무엇이 문제인지 생각해보니, 클라이언트와의 연결을 위한 socket descriptor 는 parent process 에서 만들었는데(accept()의 반환은 client 와의 연결을 위한 socket descriptor), 이를 child process 에서만 close()를 해 parent process 와 client 는 여전히 연결이 유지되고 있는 상황이었습니다. 그래서 fork()를 한 후에 바로 부모 프로세스에서는 해당 socket descriptor 를 close()하고, 자식이 terminate 되면 자동으로 socket 이 close()되므로 이때 서버와 클라이언트의 연결이 끊겨 클라이언트의 read()가 0 을 반환하도록 하였습니다.