시스템 프로그래밍 실습

[Assignmen2-2]

Class : **D 반(실습 2 금 56)**

Professor : **최상호 교수님**

Student ID : **2022202104**

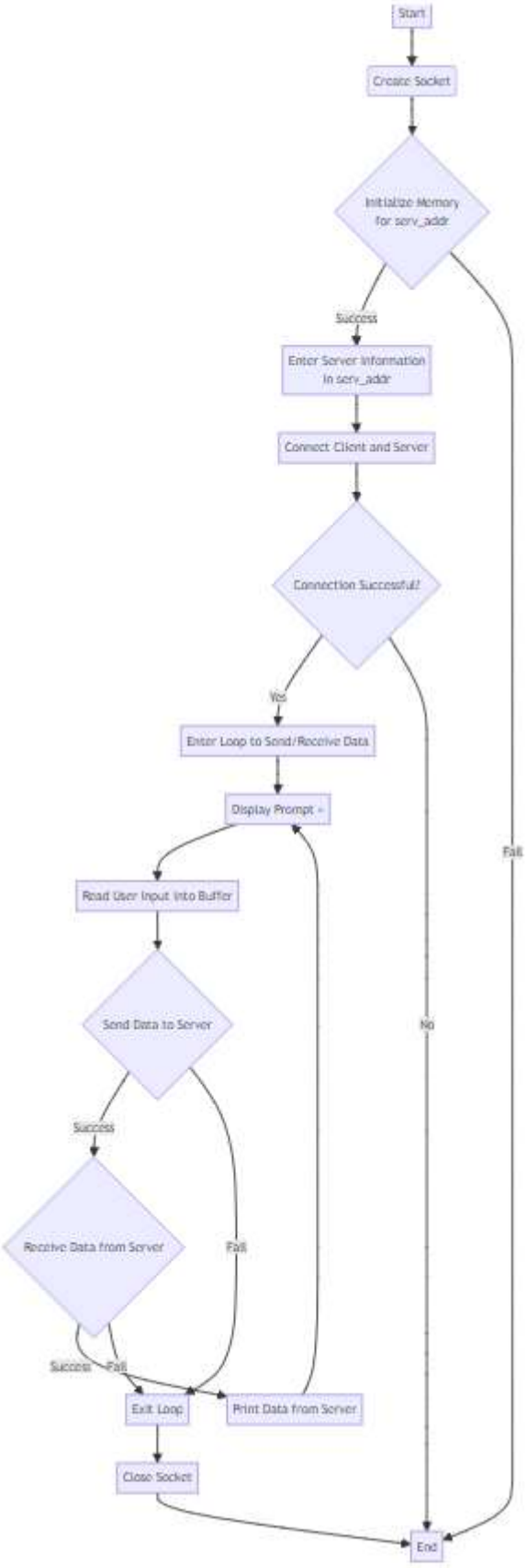
Name : **김유찬**

Introduction

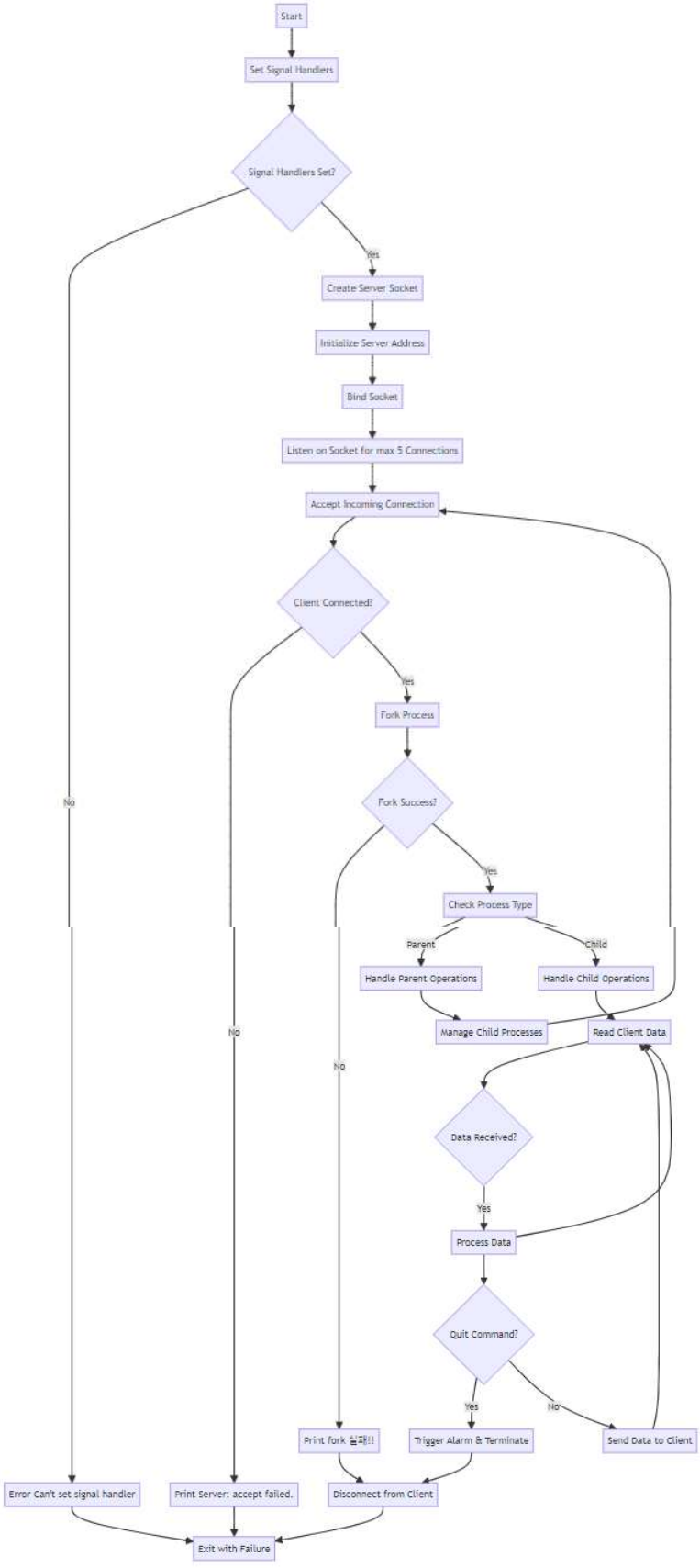
Socket 을 이용해서 서로 다른 두 컴퓨터끼리 연결할 수 있게 했다. 이번엔 system call 중 하나인 fork 를 사용해서 두 프로세스를 만들고 각각 서로 다른 일을 할 수 있도록 할 것이다. fork 를 사용하는 이유는 병렬처리를 하여 동시에 여러 일을 할 수 있기 때문이다. 하지만 코드를 짜는 개발자 입장에서 fork 가 어떻게 작동하고 돌아가는 잘 알아야 한다. 그렇기 때문에 이번 fork 를 이용하여 client 정보를 받는 process 와 client 문자열을 처리하는 process 이렇게 두개를 동시 만들어보며 fork 에 대해 알아보는 시간을 갖을 것이다.

Flow chart

1) cli 의 flow chart


Start
Create Socket
Initialize Memory for serv_addr
Success
Fail
Enter Server Information in serv_addr
Connect Client and Server
Connection Successful?
Yes
No
Enter Loop to Send/Receive Data
Display Prompt >
Read User Input into Buffer
Send Data to Server
Success
Fail
Receive Data from Server
Success
Fail
Exit Loop
Print Data from Server
Close Socket
End

2) srv 의 flow chart



Pseudo code

1) cli 의 pseudo code

```
////////////////////////// Client-side Communication Setup //////////////////////
#define BUF_SIZE 256

int main(int argc, char *argv[])
{
      char buff[BUF_SIZE];
      int n;
      int sockfd;
      struct sockaddr_in serv_addr;

      sockfd = socket(AF_INET, SOCK_STREAM, 0);

      Initializes the memory of serv_addr.
      Enter server information in serv_addr.

      Connect the client and server.
      while(1) {
            write(STDOUT_FILENO, "> ", 2);
            read(STDIN_FILENO, buff, BUF_SIZE);

            if (Send the contents of the buffer to the client and succeed.) {
                  if(Receives content from the client, stores it in the buffer, and succeeds.)
                        printf("from server: %s", buff);
                  else
                        exit loop;
            } else
                   exit loop;
      }
      close endpoint;
      return 0;
}
```

2) srv 의 pseudo code

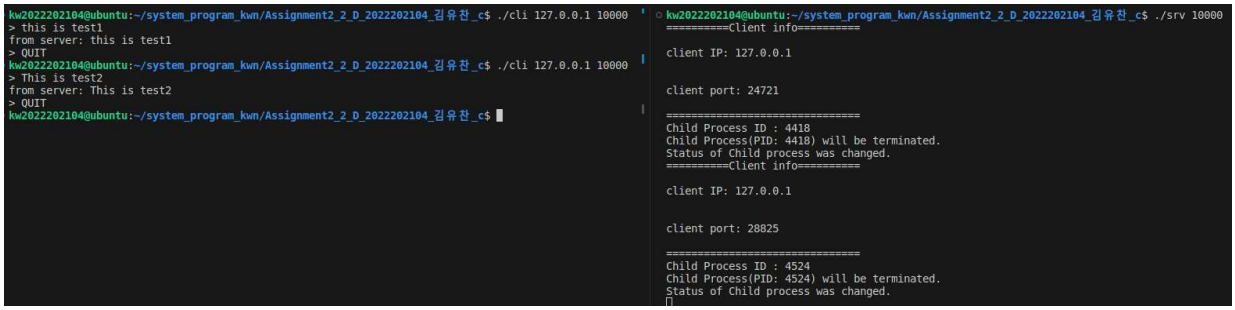
```
////////////////////////////////////
// Server Application Pseudocode //
////////////////////////////////////
#define BUF_SIZE 256
char output[1024];
int length;

int client_info(struct sockaddr_in* cliaddr);
```

```
void sh_chld(int);

void sh_alrm(int);

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 (sh_alrm was called. But if an error occurs) {
        perror("Can't set signal handler");
        return 1;
    }


    if (The child process state changes and the sh_chld function is called. But if there is a problem) {
        perror("Can't set signal handler");
        return 1;
    }

    server_fd = socket(PF_INET, SOCK_STREAM, 0);

    Initializes the memory of serv_addr.
    Enter server information in serv_addr.
    bind(server_fd, (struct sockaddr *)&server_addr, sizeof(server_addr));

    Prepare Socket for Listening to Incoming max 5 Connections

    while(1) {
        pid_t pid;
        len = sizeof(client_addr);
        client_fd = accept(server_fd, (struct sockaddr*)&client_addr, &len);

        if(Can't connect to client.){
            printf("Server: accept failed.₩n");
            return 0;
        }

        Create a new process

        if(If fork doesn't work){
            perror("fork 실패!!");
            Disconnect from the client.
        }

        else if(This is the parent process.){
```

```
                        if (The client contents were printed. But if I couldn't do it)
                                write(STDERR_FILENO, "client_info() err!!₩n", sizeof("client_info() err!!₩n"));

                        length = sprintf(output, "Child Process ID : %d₩n", pid);
                        write(STDOUT_FILENO, output, length);

                        while (Check for any terminated child process without blocking);
        }
        else(){

            while(1){
                if(Successfully receives the contents of the buffer from the client.){
                                        The content received from the client is stored in the buffer.
                }

                    if(!strncmp(buff, "QUIT", strlen("QUIT"))){
                        length = sprintf(output,"Child Process(PID: %d) will be terminated.₩n", getpid());
                                                write(STDOUT_FILENO, output, length);

                        Activates sh_alrm after 1 second.
                        Disconnect from the client.
                        return 0;
                    }
                    else{
                        Send the contents in the buffer to the client.
                    }
                }
            }
        }
        Disconnect from the client.
    }

    return 0;
}

void sh_chld(int signum) {
    printf("Status of Child process was changed.₩n");
    wait(NULL);
}

void sh_alrm(int signum) {
    printf("Child Process (PID : %d) will be terminated.₩n", getpid());
    exit(1);
}


int client_info(struct sockaddr_in* cliaddr){
    if(cliaddr->sin_family is not AF_INET)
```

```
        return -1;
            length = sprintf(output, "==========Client info==========₩n₩n");
            write(STDOUT_FILENO, output, length);
    length = sprintf(output, "client IP: %s₩n₩n₩n", inet_ntoa(cliaddr->sin_addr));
            write(STDOUT_FILENO, output, length);
    length = sprintf(output, "client port: %d₩n₩n", cliaddr->sin_port);
            write(STDOUT_FILENO, output, length);
            length = sprintf(output, "================================₩n");
            write(STDOUT_FILENO, output, length);
    return 1;
}
```

결과화면



1. server 와 Client 와 연결이 되면 server 쪽에서 fork 를 통해 두 프로세스를 만들어 각각 다른 일을 하게 한다.

2. Parent Process 는 연결된 Client 의 IP, Port 를 출력하고 다음 client 와 연결을 준비한다. Child Process 는 Client 에게 문자열을 받으면 Server 쪽에서 문자열을 출력하지 않고 이를 그대로 Client 한테 그대로 전송하고 이를 계속 반복한다.

3. 그림에서 보다시피 Client 와 서버가 연결되면 Parent Process 가 클라이언트 정보를 출력하고 다른 Client 을 받을 준비하고 있다는 것을 알 수 있다. (server 와 client 가 끊겨지고 다시 다른 client 와 연결되었다.)

4. client 쪽에서 server 에 문자열을 보내면 child process 에서 server 쪽에 출력하지 않고 그대로 client 에 보내고 client 는 받은 문자열을 그대로 출력한다.

5. client 쪽에서 QUIT 이라는 문자열을 보내면 해당 Client 는 연결이 종료가 되고 SIGALRM 에 의해 1 초 뒤에 해당 프로세스도 종료가 된다. 그 전에 곧 종료된다고 알려준다. 진짜로 종료가 되면 SIGCHLD 에 의해 Child Process 의 상태가 비꼈다고 알려주면 종료되었다고 알려준다.

고찰

이번 과제에서 System call 인 fork 을 써서 두 프로세스를 병렬 처리할 수 있도록 했다. 그 동시에 자원을 공유할 수 있도록 했기 때문에 fork 는 속도, 메모리 두 측면에서 매우 효율적인 코드라고 할 수 있었다. 하지만 fork 를 통해 child process 가 종료될 때 parent process 가 wait 를 통해 받을 수 있게 하는 등 신경을 많이 써야 했다.

child process 의 상태가 변할 때마다 작동되는 SIGCHLD, 몇 초 후에 작동되는 SIGALRM 등 유용한 SIGINAL 을 써봤는데 다른 코드를 짤 때도 많이 필요할 것 같다는 생각이 들었고 추가로 다른 SIGINAL 도 찾아볼 것이다.

Reference

시스템프로그래밍 실습 강의자료