

# 시스템 프로그래밍

## 2차 과제 – Socket Programing



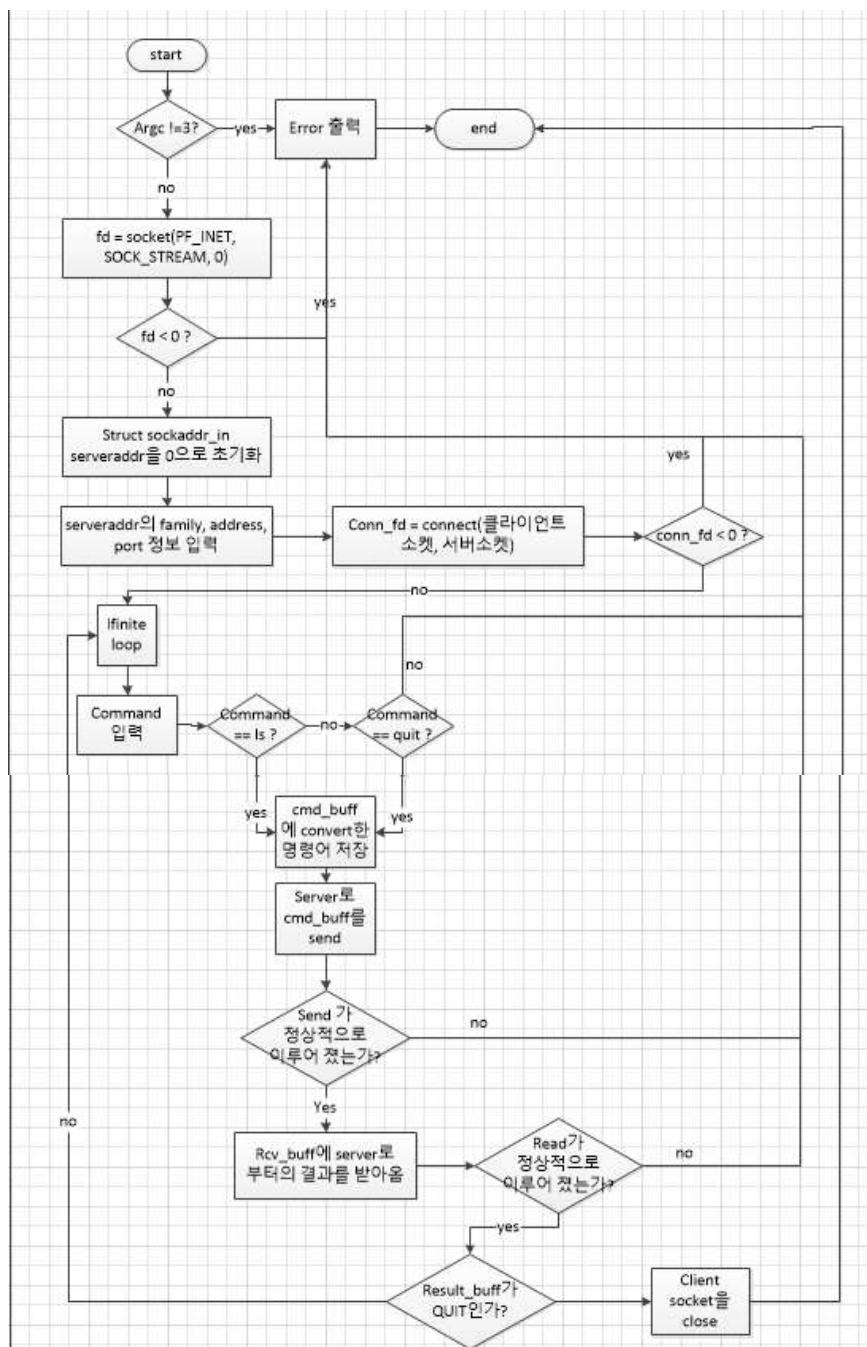
실습 일시 : 화 1,2  
담당 교수님 : 김태석 교수님  
학번 : 2013722095  
이름 : 최재은  
실습 번호 : practice #2-1

## ■ Introduction

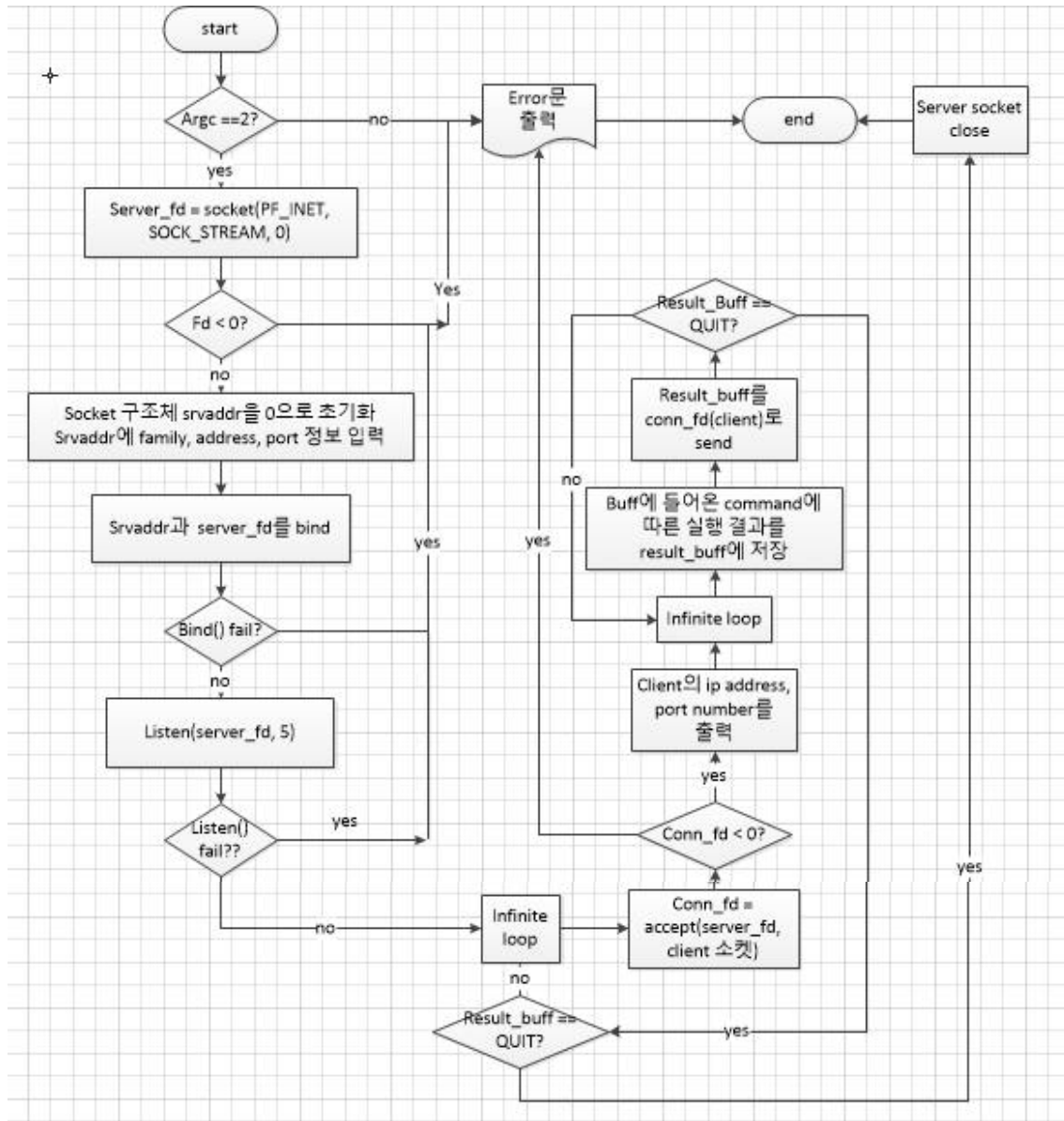
기존의 클라이언트 / 서버 통신 방식에서와 다르게 직접 소켓을 생성하고 주소와 bind하는 등 실제 네트워크 프로토콜에서 동작하는 방식을 구현해 봄으로써 ftp의 원리를 심도 있게 이해한다.

## ■ Flow Chart

### 1. client



## 2. server



## ■ Source Code

### 1. server.c

```
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <dirent.h>
#define MAX_BUFF 500
#define SEND_BUFF 500

int client_info(struct sockaddr_in* clientaddr)
{
    char temp[200];
    char ip[20] ;
    int port;
    if(inet_ntoa(clientaddr->sin_addr) < 0) return -1;
    else strcpy(ip, inet_ntoa(clientaddr->sin_addr));
    if(ntohs(clientaddr->sin_port) < 0) return -1;
    else port = ntohs(clientaddr->sin_port);

    sprintf(temp, "====Client info====\nIP address : %s\nPort # : %d\n", ip, port);
    write(STDOUT_FILENO, temp, strlen(temp)); // print input info
    return 0;
}

int cmd_process(char* buff, char* result_buff) // excute inserted command
{
    DIR* dp; // for open directory
    struct dirent *dirp;

    if((dp = opendir(".")) < 0) // open current directory
        return -1;
    write(STDOUT_FILENO, buff, strlen(buff)); // print passed command
    write(STDOUT_FILENO, "\n", strlen("\n"));
    if(!strcmp(buff, "NLST")) // if input command is NLST
    {
        while(dirp = readdir(dp))
        {
            // print files in the current directory
            strcat(result_buff, dirp->d_name);
            strcat(result_buff, "\n");
        }
    }
}
```

```

        else if(!strcmp(buff, "QUIT")) // if input command is QUIT
        {
            strcpy(result_buff,"QUIT");
        }
        else // About other command,
return -1
            return -1;

        return 0;
    }

int main(int argc, char **argv)
{
    char buff[MAX_BUFF], result_buff[SEND_BUFF]; // array for 'converted command(input)', 'result
(output)'
    int n;
    int server_fd, conn_fd; // socket
discriptor for server and client
    int clien; / /
client struct's size
    struct sockaddr_in srvaddr, cliaddr; // socket address struct

    if(argc != 2) // if wrong input case
    {
        write(STDOUT_FILENO, "Server error : Check Port Numver\n", strlen("Server error : Check
Port Numver\n"));
        return -1;
    }

    /* open socket */
    if((server_fd = socket(PF_INET, SOCK_STREAM,0)) < 0)
    {
        write(STDERR_FILENO, "Server : socket() err!!\n", strlen("Server : socket() err!!\n"));
        return -1;
    }

    /* initialize server socket struct */
    memset(&srvaddr, 0, sizeof(srvaddr));
    srvaddr.sin_family= PF_INET;
    srvaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    srvaddr.sin_port = htons(atoi(argv[1]));

    //bind socket with address
    if(bind(server_fd, (struct sockaddr*)&srvaddr, sizeof(srvaddr)) < 0)
    {
        write(STDERR_FILENO, "Server : bind() err!!\n", strlen("Server : bind() err!!\n"));
        return -1;//exit(1);
    }

```

```

// listen (5 time)
if(listen(server_fd, 5) < 0)
{
    write(STDERR_FILENO, "Server : listen() err!!\n", strlen("Server : listen() err!!\n"));
    return -1;
}

for(;;)
{
    clilen = sizeof(cliaddr);
    /* accept client's connection */
    conn_fd = accept(server_fd, (struct sockaddr* ) &cliaddr, &clilen);
    if(conn_fd < 0) // connection failed case
    {
        write(STDERR_FILENO, "Server : accept() err!!\n", strlen("Server : accept()
err!!\n"));
        return -1;
    }
    /*display client ip and port*/
    if(client_info(&cliaddr) < 0)
    {
        write(STDERR_FILENO,"Server : client_info() err!!\n", strlen("Server : client_info()
err!!\n"));
        return -1;
    }

    while(1)
    {
        n = read(conn_fd, buff, MAX_BUFF);
        buff[n] = '\0';

        if(cmd_process(buff, result_buff) < 0)
            /*command execute and result*/
            {
                write(STDERR_FILENO, "Server : cmd_process() err!!\n", strlen("Server :
cmd_process() err!!\n"));
                break;
            }
        /* send result to client */
        write(conn_fd, result_buff, strlen(result_buff));

        /* if converted command is QUIT*/
        if(!strcmp(result_buff, "QUIT"))
        {
            write(STDOUT_FILENO, "Server Quit!!\n", strlen("Server Quit!!\n"));

```

```
                break;
            }
        }
        if(!strcmp(result_buff, "QUIT")) // break loop
            break;
    }
    close(server_fd); // close server socket
    return 0;
}
```

## 2. client.c

```
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#define MAX_BUFF 500
#define RCV_BUFF 500

int conv_cmd(char* buff, char* cmd_buff) // command converter
{
    if(!strcmp(buff, "ls")) // convert 'ls' to 'NLST'
        strcpy(cmd_buff, "NLST");
    else if(!strcmp(buff, "quit")) // convert 'quit' to 'QUIT'
        strcpy(cmd_buff, "QUIT");
    else return -1; // About other commands, return -1

    return 0;
}

void process_result(char* rcv_buff) // print recieved result
{
    write(STDOUT_FILENO, rcv_buff, strlen(rcv_buff));
}

int main (int argc, char **argv)
{
    char buff[MAX_BUFF], cmd_buff[MAX_BUFF], rcv_buff[RCV_BUFF]; // array for 'user command',
    'converted command', 'recieve result'
    int n;
    char temp[100];
    int client_fd, conn_fd; // client socket discripiter /
server socket discripiter
    char* IP_addr;
    char* port_num;
    struct sockaddr_in srvaddr; // server socket sturct

    if(argc != 3) // uncorrect input case
    {
        write(STDERR_FILENO, "Client : Need IP address / Port Number\n", strlen("Client : Need IP address /
Port Number\n"));
        return -1;
    }

    /* open socket */
    if((client_fd = socket(PF_INET, SOCK_STREAM, 0)) < 0)
    {
        write(STDERR_FILENO, "Client : socket() error!!\n", strlen("Client : socket() error!!\n"));
        return -1;
    }
}
```



```

/* connect to server */
port_num = argv[2];
IP_addr = argv[1];

// initialize srvaddr
memset(&srvaddr, 0, sizeof(srvaddr));
srvaddr.sin_family = PF_INET;
srvaddr.sin_addr.s_addr = inet_addr(IP_addr);
srvaddr.sin_port = htons(atoi(port_num));

// if client failed to connect with server
if((conn_fd = connect(client_fd, (struct sockaddr*)&srvaddr, sizeof(srvaddr))) < 0)
{
    write(STDERR_FILENO, "Client : connect() error!!\n", strlen("Client : connect() error!!\n"));
    return -1;
}

for(;;)
{
    write(STDOUT_FILENO, "CMD>> ", strlen("CMD>> "));
    read(STDIN_FILENO, buff, MAX_BUFF);
    n = strlen(buff);
    buff[n-1] = '\0';

    if(conv_cmd(buff, cmd_buff) < 0) // convert command
    {
        write(STDERR_FILENO, "Client : conv_cmd() error!!\n", strlen("Client : conv_cmd()
error!!\n"));
        return -1;
    }

    n = strlen(cmd_buff); // get converted command's length

    // send converted command to server
    if(write(client_fd, cmd_buff, n) != n)
    {
        write(STDERR_FILENO, "Client : write() error!!\n", strlen("Client : write()
error!!\n"));
        return -1;
    }

    // read result from server
    if((n= read(client_fd, rcv_buff, RCV_BUFF-1)) < 0)
    {
        write(STDERR_FILENO, "Client : read() error!!\n", strlen("Client : read()
error!!\n"));
        return -1;
    }
}

```

```
// add null into result array
rcv_buff[n] = '\0';

if(!strcmp(rcv_buff, "QUIT")) // if recieved command was quit
{
    write(STDOUT_FILENO, "Program quit!!\n", strlen("Program quit!!\n"));
    close(client_fd);    // close socket
    return 0;
}

process_result(rcv_buff); // display ls command result
}
return 0;
}
```

## ■ Result

```
jaeen1113@ubuntu:~/SP/prac21$ ./cli 127.0.0.1 1000 Client : connect() error!!
jaeen1113@ubuntu:~/SP/prac21$ ./cli 127.0.0.1 1200
CMD>> ls
srv
srv.c
Makefile
cli.c
..
.
cli
CMD>> quit
Program quit!!
jaeen1113@ubuntu:~/SP/prac21$

jaeen1113@ubuntu:~/SP/prac21$ ./srv
Server error : Check Port Numver
jaeen1113@ubuntu:~/SP/prac21$ ./srv 1200
=====Client info=====
IP address : 127.0.0.1

Port # : 42
=====
NLST
QUIT
Server Quit!!
jaeen1113@ubuntu:~/SP/prac21$
```

- 처음에 서버를 열지 않고 클라이언트에서 연결을 시도하자 에러가 뜨는 것을 확인할 수 있다.
- 서버 실행 시 포트 넘버를 주지 않으면 에러가 뜬다.
- 서버를 열고 클라이언트에서 연결을 하자 서버 측에 클라이언트의 info가 뜨는 것을 확인 할 수 있다.
- ls 명령어 입력 시, 서버 측으로 NLST 명령어가 넘어가고 이에 대한 결과가 클라이언트로 넘어와서 출력되는 것을 볼 수 있다.
- quit 명령어 입력 시, 서버 측으로 QUIT 명령어가 넘어가고 서버 소켓이 close되고 서버 종료, 이후 클라이언트 쪽 소켓도 닫고 클라이언트 종료