시스템 프로그래밍 실습

9차 과제

실습 일시 : 화 1,2
담당 교수님 : 김태석 교수님
학번 : 2013722095
이름 : 최재은
실습 번호 : practice #3-1

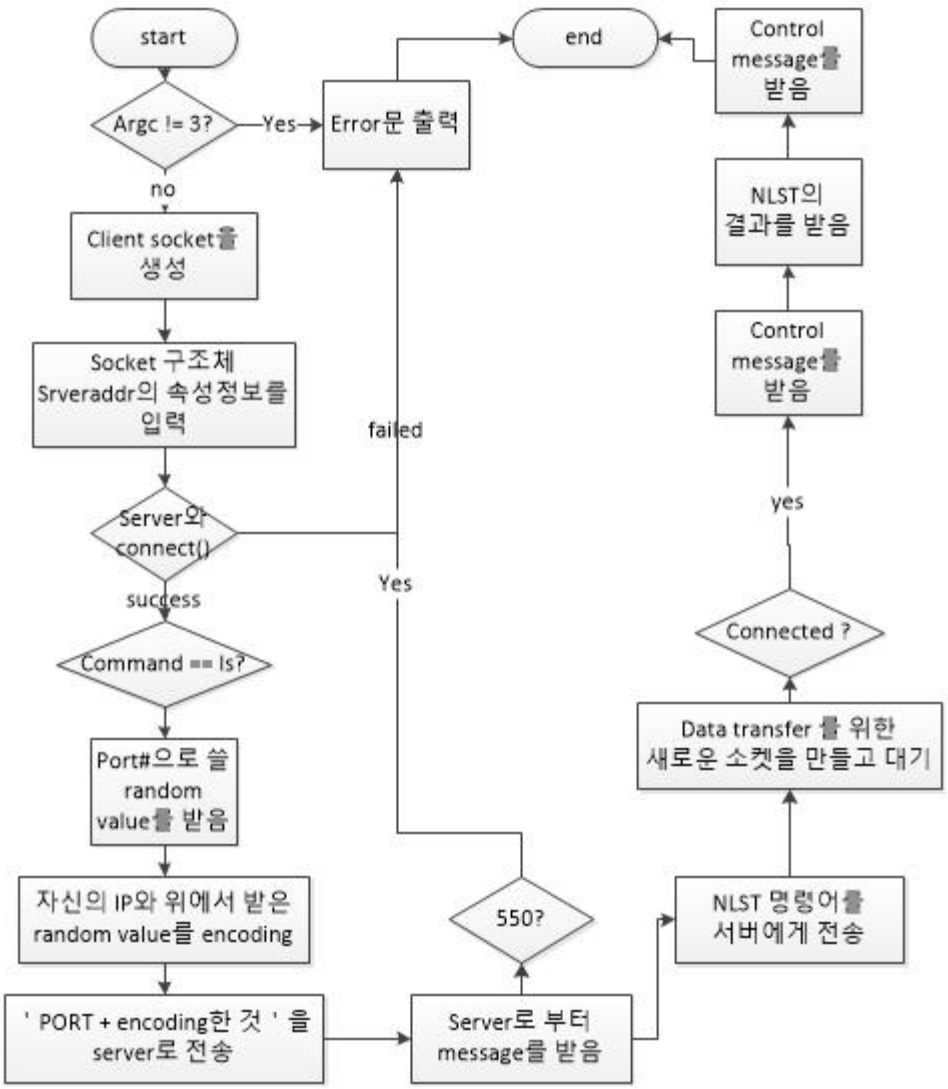
■ Introduction

기존의 client/server 통신 방식에서는 하나의 소켓을 통해 시그널과 Data를 주고 받는 구조로 동작하였다. 이번에는 기존의 link말고 새로운, Data를 주고 받기 위한 Data Transfer 전용의 소켓을 만들고 client/server가 바뀐 역할을 수행하도록 구현해본다.

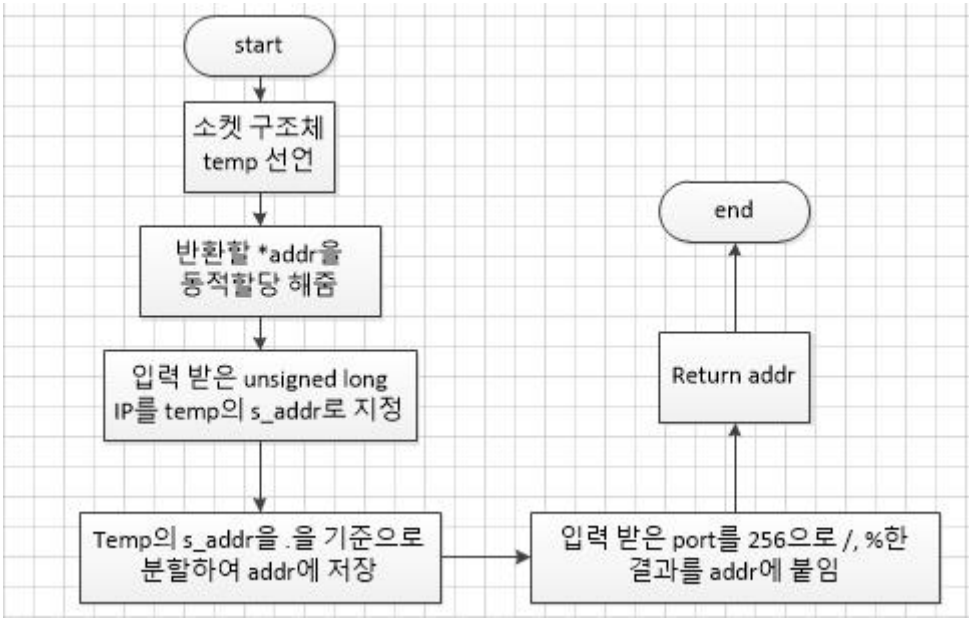
■ Flow Chart

1. cli.c

1) main

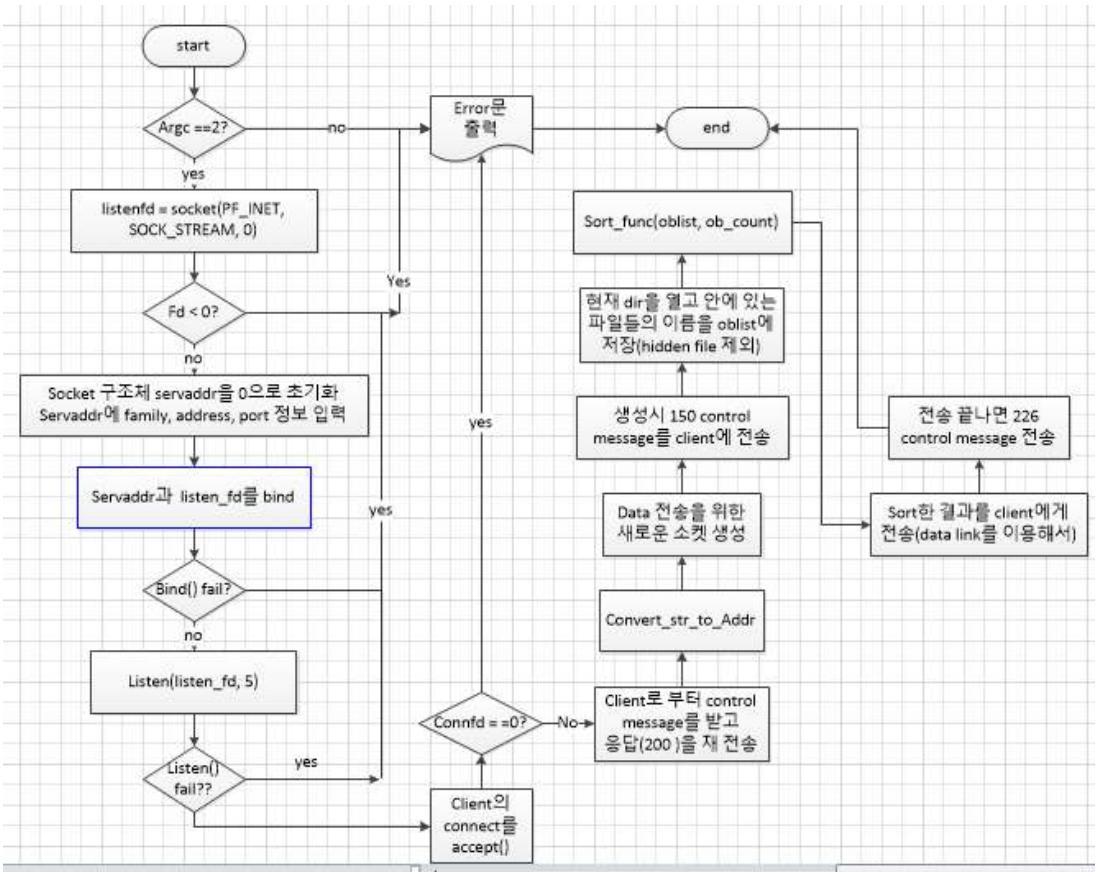


2) convert_addr_to_str

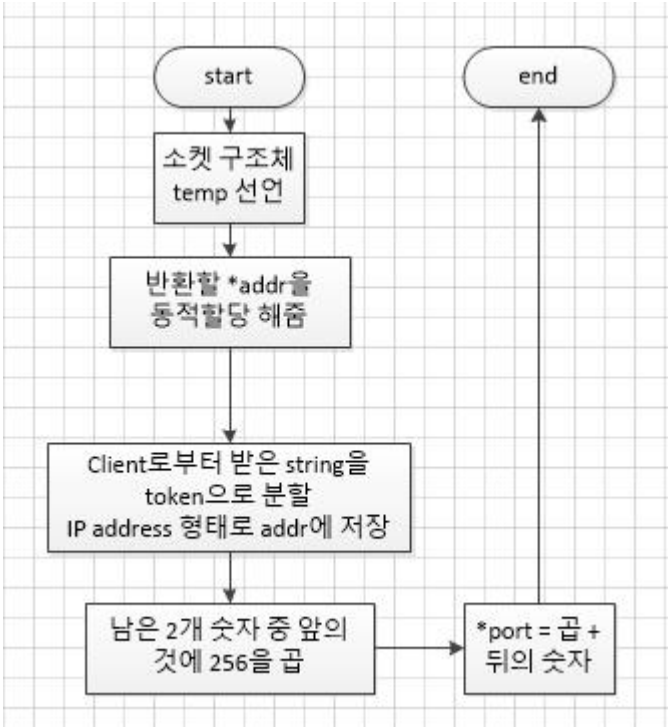


2. srv.c

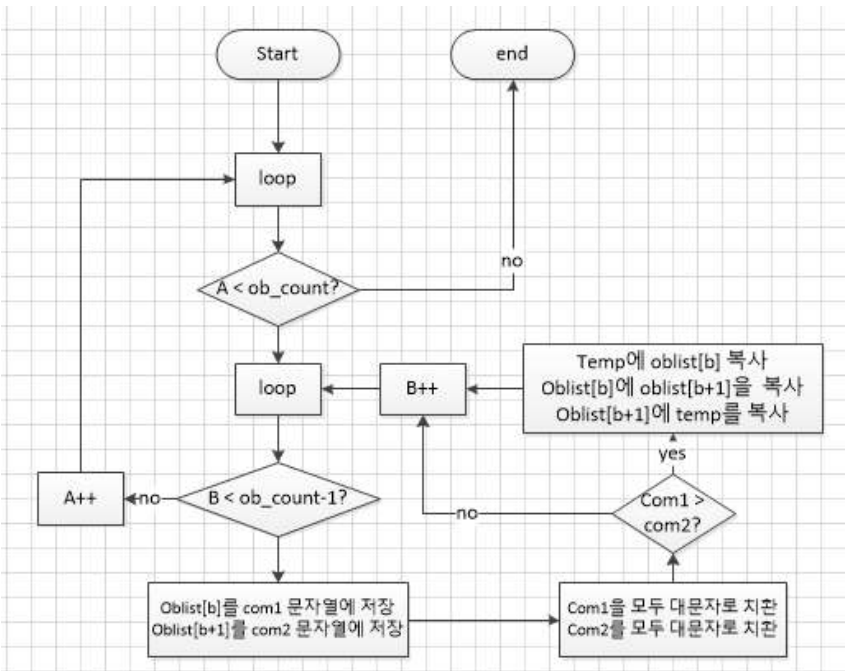
1) main



2) convert_str_to_addr



3) sort_func



■ Source Code

1. srv.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <dirent.h>

#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <sys/wait.h>

#include <arpa/inet.h>
#include <netinet/in.h>

#define MAX 1024
void sort_func(char(*)[50], int);
char* convert_str_to_addr(char * , unsigned int *);
void main(int argc, char **argv)
{
        char *host_ip;
        char *token1, *token2, *token3, *token4;
        char temp[MAX];
        char ip[50];
        char port[50];
        char result[MAX];
        char arr[30];
        char oblist[50][50];
        int ob_count = 0;
        int decoded_port = 0;
        unsigned int port_num;

        DIR *dp;
        struct dirent *dirp;

        int listenfd, connfd, connfd_data;
        int len, a = 0;
```

```
	struct sockaddr_in servaddr, cliaddr, cliaddr_data;

	if(argc != 2)
	{ printf("error : confrim argument₩n"); exit(1);}

	/* create socket(for signal, command) */
	if((listenfd = socket(PF_INET, SOCK_STREAM, 0)) < 0)
	{
		write(STDOUT_FILENO, "error : socket() is failed!₩n", strlen("error :
socket() is failed!₩n"));
		exit(1);
	}

	/* set server socket's information */
	memset(&servaddr, 0 , sizeof(servaddr));
	servaddr.sin_family = AF_INET;
	servaddr.sin_addr.s_addr = htons(INADDR_ANY);
	servaddr.sin_port = htons(atoi(argv[1]));

	/* bind server socket(for signal, command) with address */
	if(bind(listenfd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0)
	{
		write(STDOUT_FILENO, "error : bind() is failed!₩n", strlen("error : bind()
is failed!₩n"));
		exit(1);
	}

	/* prepare connection */
	if(listen(listenfd, 5) < 0)
	{
		write(STDOUT_FILENO, "error : listen() is failed!₩n", strlen("error : listen()
is failed!₩n"));
		exit(1);
	}


	memset(ip, 0 , sizeof(ip));
	memset(port, 0, sizeof(port));
	memset(temp, 0, sizeof(temp));
	memset(result, 0, sizeof(result));
```

```
	len = sizeof(cliaddr);
	/*connect with client*/
	connfd = accept(listenfd, (struct sockaddr*)&cliaddr, &len); // connect server with client (for signal, command)

	if(connfd == 0)
	{printf("error : failed to connect with client.₩n"); exit(1);}

	read(connfd, temp, MAX);
	printf("%s₩n", temp);

	write(STDOUT_FILENO, "200 Port command successful₩n", strlen("200 Port command successful₩n"));
	write(connfd, "200 Port command successful₩n", strlen("200 Port command successful₩n"));

	/* decode message from client*/
	host_ip = convert_str_to_addr(temp, (unsigned int*) &port_num);

	memset(temp, 0, sizeof(temp));
	read(connfd, temp, MAX);
	write(STDOUT_FILENO, temp, strlen(temp));
	write(STDOUT_FILENO, "₩n", strlen("₩n"));

	if(strcmp(temp, "NLST") != 0)
	{
		close(connfd);
		exit(1);
	}

	/* make socket for data transfer */
	connfd_data = socket(AF_INET, SOCK_STREAM, 0);
	memset(&cliaddr_data, 0, sizeof(cliaddr_data));
	cliaddr_data.sin_family = AF_INET;
	cliaddr_data.sin_addr.s_addr = inet_addr(host_ip);
	cliaddr_data.sin_port = htons(port_num);

	/* connect with client for data transfer with appointed port */
	if(connect(connfd_data, (struct sockaddr*)&cliaddr_data, sizeof(cliaddr_data)) < 0)
	{
		printf("error : Failed to connect with client(for data)₩n");
```

```
                exit(0);
        }

        write(STDOUT_FILENO, "150 Opening data connection for directory list₩n", strlen("150 Opening data connection for directory list₩n"));
        write(connfd, "150 Opening data connection for directory list₩n", strlen("150 Opening data connection for directory list₩n"));
        sleep(1); // making delay transfer

        /* NLST command excution */
        dp = opendir(".");
        while(dirp = readdir(dp))
        {
                strcpy(arr, dirp->d_name);
                if(arr[0] == '.')
                        continue;
                strcpy(oblist[ob_count], dirp->d_name);
                ob_count++;
        }

        /*sort result */
        sort_func(oblist, ob_count);

        /*concatnate result into array*/
        for(a = 0; a < ob_count; a++)
        {
                strcat(result, oblist[a]);
                strcat(result, "₩n");
        }
        // send NLST result
        write(connfd_data, result, strlen(result));

        // send 226
        write(STDOUT_FILENO, "226 Result is sent successfully.₩n", strlen("226 Result is sent successfully.₩n"));
        write(connfd, "226 Result is sent successfully.₩n", strlen("226 Result is sent successfully.₩n"));
}

char* convert_str_to_addr(char *str, unsigned int *port)
{
```

```
        char *addr;
        int *decoded_port;
        int head,   tail;
        char *token, *token2, *token3, *token4;

        addr = malloc(100*sizeof(char));


        token = strtok(str, " "); // remove 'PORT' word
        // seperate ip address
        token = strtok(NULL, ",");
        token2 = strtok(NULL, ",");
        token3 = strtok(NULL, ",");
        token4 = strtok(NULL, ",");

        sprintf(addr, "%s.%s.%s.%s", token, token2, token3, token4);

        // decode port number
        token3 = strtok(NULL, ",");
        token4 = strtok(NULL, " ");

        head = atoi(token3) * 256;
        tail = atoi(token4);

        *port = head + tail;

        return addr;
}

void sort_func(char oblist[50][50], int ob_count)
{
        char com1[50];
        char com2[50];
        char temp[50];
        int a, b, c;


        for(a = 0; a < ob_count; a++) // bubble sort
    {
        for(b=0; b < ob_count-1; b++)
        {
```

```
                strcpy(com1, oblist[b]);
                strcpy(com2, oblist[b+1]);

                    // capitalizing name to compare
                for(c = 0; c<strlen(com1); c++)
                {if((com1[c]>=65) && (com1[c]<=90)) com1[c] += 32;}
                for(c =0; c< strlen(com2); c++)
                {if((com2[c] >= 65)&&(com2[c] <=90)) com2[c] +=32;}

                    if(strcmp(com1, com2)>0) // if com1 is bigger
                {
                    strcpy(temp, oblist[b]);
                    strcpy(oblist[b], oblist[b+1]);
                    strcpy(oblist[b+1], temp);
                }
            }
        }
}
```

2. cli.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <sys/wait.h>

#include <arpa/inet.h>
#include <netinet/in.h>

#include <time.h>

#define MAX 1024
char* convert_addr_to_str(unsigned long, unsigned int);
void main(int argc, char **argv)
{
        char *hostport;
        char *token1;
        char command[MAX];
        char result[MAX];
        char ip[4][20];
        char note[MAX];
        struct sockaddr_in srvaddr, temp;
        int sockfd, srv_data, sockfd_data, n;
        int random_port, head, tail;
        int len, result_byte = 0;


        if(argc != 3)
        {
                printf("error : check arguments!₩n"); exit(1);
        }


        // make client socket
```

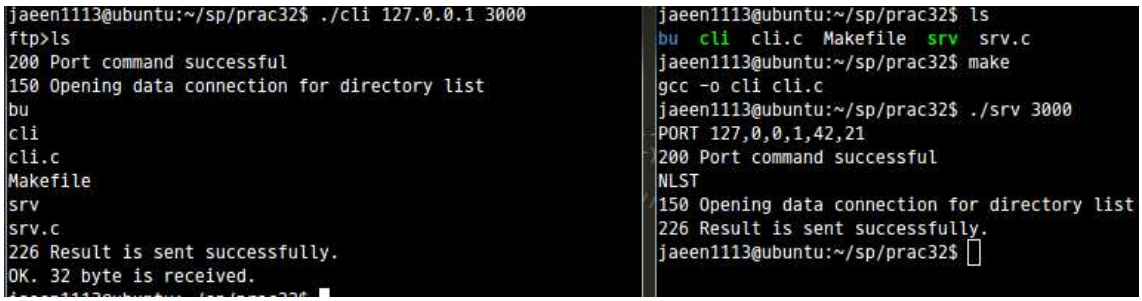
```
sockfd = socket(AF_INET, SOCK_STREAM, 0);

// initialize server socket structure
memset(&srvaddr, 0 , sizeof(srvaddr));
srvaddr.sin_family = AF_INET;
srvaddr.sin_addr.s_addr = inet_addr(argv[1]);
srvaddr.sin_port = htons(atoi(argv[2]));


// try connecting
if(connect(sockfd, (struct sockaddr*)&srvaddr, sizeof(srvaddr)) < 0)
{
        printf("error : Filed to connect with server!₩n");
        exit(1);
}

write(STDOUT_FILENO, "ftp>", strlen("ftp>"));
n = read(STDIN_FILENO, command, sizeof(command));
command[n-1] = '₩0';    // remove enter

if(!strcmp(command, "ls"))
{

        srand(time(NULL)); // make random value

        // get random port number(10001 ~ 30000)
        random_port = (rand()%20000)+10001;

        hostport = convert_addr_to_str(srvaddr.sin_addr.s_addr, random_port);
        // send PORT command to server
        sprintf(note, "PORT %s",hostport);
        write(sockfd, note, strlen(note));

        // receive command result from server
        read(sockfd, result, MAX);
        printf("%s", result);

        //send converted command
        write(sockfd, "NLST ", strlen("NLST"));
```

```
		/* make new link (for data) transfer */
		srv_data = socket(PF_INET, SOCK_STREAM, 0);


		memset(&temp, 0, sizeof(temp));
		temp.sin_family = AF_INET;
		temp.sin_addr.s_addr = htons(INADDR_ANY);
		temp.sin_port = htons(random_port);

		if(bind(srv_data, (struct sockaddr*)&temp, sizeof(temp)) < 0)
		{
			printf("error : Failed to bind(for data)₩n");
			exit(1);
		}

		if(listen(srv_data, 5) < 0)
		{
			printf("error : Failed to listen(for data)₩n");
			exit(1);
		}

		len = sizeof(temp);
		sockfd_data = accept(srv_data,(struct sockaddr*)&temp,&len);
		//receive signal
		memset(result, 0, sizeof(result));
		read(sockfd, result, MAX);
		write(STDOUT_FILENO, result, strlen(result));
		//receive data
		memset(result, 0, sizeof(result));
		read(sockfd_data, result, MAX);
		write(STDOUT_FILENO, result, strlen(result));
		result_byte = strlen(result)*sizeof(char);
		//receive signal
		memset(result, 0, sizeof(result));
		read(sockfd, result, MAX);
		write(STDOUT_FILENO, result, strlen(result));

		printf("OK. %d byte is received.₩n", result_byte);
}
else
{
```

```
                printf("error : Unknown command!₩n");
                exit(1);
        }
        exit(0);

}

char* convert_addr_to_str(unsigned long ip_addr, unsigned int port)
{
        char *addr;
        int head,  tail;
        char ip[20];
        struct sockaddr_in temp;
        char *token, *token2, *token3, *token4;

        temp.sin_addr.s_addr = ip_addr;

        addr = malloc(100*sizeof(char));

        token = strtok(inet_ntoa(temp.sin_addr), ".");
        token2 = strtok(NULL, ".");
        token3 = strtok(NULL, ".");
        token4 = strtok(NULL, " ");

        head = port / 256;
        tail = port % 256;

        sprintf(addr, "%s,%s,%s,%s,%d,%d", token, token2, token3, token4,head,tail);

        return addr;
}
```

■ Result

- 정상적으로 연결시 왼쪽과 같이 ftp 명령어를 입력하도록 동작한다.



- ls 명령어 입력 시 PORT command가 전송되고 server는 그에 대한 응답으로 200 port .....라는 메시지를 보낸다.
- 이후 client는 NLST라는 명령어를 보내고, server는 data 전송을 위한 connection을 연결한다. 이후 150 Opening ....라는 메시지를 전송하고 Data connection을 통해 client가 요청한 명령의 결과를 보내준다.