System Programming

시스템 프로그래밍

(화5, 목6)

Assignment #3-1

김 태 석 교수님

컴퓨터정보공학부

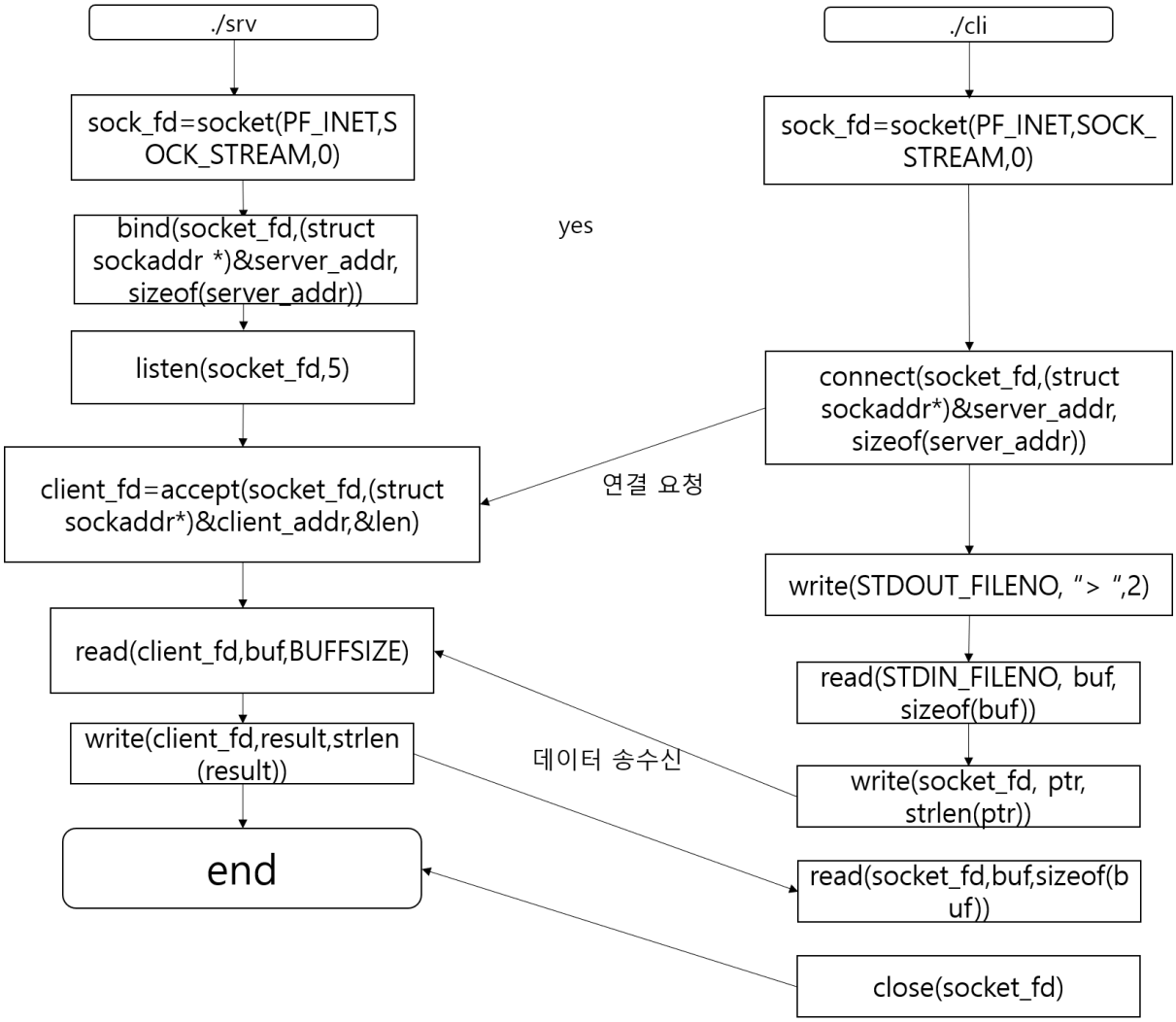
2017202037

오 민 혁

<Introduction>

이번 과제는 Socket을 이용하여 command "ls" implementation와 quit를 구현하는 것이다. 우선 socket file descriptor를 이용하여 Client와 Server를 연결하는 방법을 알아야 한다. Client와 Server의 관계를 이해해야 한다. Client에서 명령어를 입력받아서 write()함수를 이용하여 Server로 보내면 Server에서는 read()함수를 이용하여 명령어를 받는다. 명령어에 해당하는 작동을 Server에서 수행 한 후 결과를 다시 write() 함수를 이용하여 Client로 보낸다. 그럼 Client에서 다시 read()함수를 이용하여 결과를 받고, 이 결과를 커널에 출력한다. 이러한 작동 방식에 대해 정확히 이해해야 하는 과제이다.

<Flow Chart>



<Source Code>

<srv.c>

```
////////////////////////////////////////////////////////////////////
// File Name : srv.c //
// Date : 2020/05/21 //
// Os : Ubuntu 16.04.5 LTS 64bits //
// Author : Oh Min Hyeok //
// Student ID : 2017202037 //
// ------------------------------------------------------------ //
// Title : System Programming Assignment #3-1 ( socket programming ) //
// Description : ... //
////////////////////////////////////////////////////////////////////


#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <dirent.h>
#include <arpa/inet.h>
```

```
#define BUFFSIZE       1024
#define PORTNO                 40000

void client_info(struct sockaddr_in cliaddr){
        printf("==========Client Info==========₩n");
        printf("client IP: %s₩n₩n",inet_ntoa(cliaddr.sin_addr));
        printf("client port: %d₩n",cliaddr.sin_port);
        printf("==============================₩n");
}
char * cmp_process(char *buff,char *result_buff){




        if(strcmp(buff,"ls")==0) { // ls command
                char *cwd=(char*)malloc(sizeof(char)*1024); // dynamic allocate
                DIR * dir =NULL; // directory pointer
                struct dirent * entry =NULL; // directory struct
                getcwd(cwd,1024); // current working directory
                dir=opendir(cwd);

                if(dir==NULL){ // exception handling
                        strcat(result_buff,"error");
```

```
        }

        while((entry=readdir(dir))!=NULL) // read directory
        {

            char *tmp=(char*)malloc(sizeof(char)*1024);
            bzero((char*)&tmp, sizeof(tmp)); // initialize
            tmp=entry->d_name; // directory name
            int length=strlen(tmp);

            strcat(result_buff,tmp);

            strcat(result_buff,"₩n");

        }

        free(cwd); // free dynamic allocate memory
        closedir(dir); // close directory

    }
    else{
```

```
		strcat(result_buff,"input error");
	}



	return result_buff;
}
int main()
{
	struct sockaddr_in server_addr, client_addr; // socket address struct
	int socket_fd, client_fd; // socket file descriptor, client file descriptor
	int len, len_out; // string length
	char buf[BUFFSIZE]; // buf
	char result_buf[BUFFSIZE];
	bzero(result_buf, sizeof(result_buf));
	if((socket_fd=socket(PF_INET,SOCK_STREAM,0))<0){ // socket open
		printf("Server: Can't openstream socket.");
		return 0;
	}

	bzero((char *)&server_addr, sizeof(server_addr)); // server_addr initialize
	//set server address
	server_addr.sin_family = AF_INET;
	server_addr.sin_addr.s_addr=htonl(INADDR_ANY);
```

```
server_addr.sin_port=htons(PORTNO);
// associate an address with a socket
if(bind(socket_fd, (struct sockaddr *)&server_addr, sizeof(server_addr))<0){
        printf("Server: Can't bind local address.₩n"); // exception handling
        return 0;
}

listen(socket_fd,5); // a server announces that it is willing to accept connect requests
while(1){
        len=sizeof(client_addr);
        client_fd=accept(socket_fd,(struct   sockaddr*)&client_addr,&len);   // accept connect request
        if(client_fd<0){ // exception handling
                printf("Server: accept failed.₩n");
                return 0;
        }
        /////////// print Client Info //////////
        client_info(client_addr);

        bzero(buf,sizeof(buf)); // initialize

        // receive command from client file descriptor
```

```
            while((len_out=read(client_fd, buf, BUFFSIZE))>0) {
                write(STDOUT_FILENO, buf, len_out); // print command
                write(STDOUT_FILENO,"₩n",1); // line break

                if(strcmp(buf,"quit")==0){ // quit command
                    write(client_fd,"Program quit!!",14);
                    close(client_fd);
                    close(socket_fd);
                    exit(1);

                }

                write(client_fd,cmp_process(buf,result_buf),sizeof(result_buf));
                bzero(result_buf,sizeof(result_buf));
                bzero(buf,sizeof(buf)); // initialize

            }

        }
        close(socket_fd);
        return 0;
}
```

<cli.c>

```
////////////////////////////////////////////////////////////////////
// File Name : cli.c //
// Date : 2020/05/21 //
// Os : Ubuntu 16.04.5 LTS 64bits //
// Author : Oh Min Hyeok //
// Student ID : 2017202037 //
// ---------------------------------------------------------------- //
// Title : System Programming Assignment #3-1 ( socket programming ) //
// Description : ... //
////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <dirent.h>
#include <stdlib.h>

#define BUFFSIZE	1024
#define PORTNO		40000
```

```
int main()
{
	int socket_fd, len; // socket file descriptor
	struct sockaddr_in server_addr; // socket address struct
	char haddr[]="127.0.0.1"; // address
	char buf[BUFFSIZE]; // buffer



	if((socket_fd=socket(PF_INET, SOCK_STREAM, 0))<0){ // socket open and exception handling
		printf("can't create socket.₩n");
		return -1;
	}
	// set server address
	bzero(buf, sizeof(buf)); // initialize
	bzero((char*)&server_addr, sizeof(server_addr)); // initialize
	server_addr.sin_family=AF_INET;
	server_addr.sin_addr.s_addr=inet_addr(haddr);
	server_addr.sin_port=htons(PORTNO);

	if(connect(socket_fd, (struct sockaddr*)&server_addr, sizeof(server_addr))<0){ // request a connection to server
```

```
        printf("can't connect.₩n"); // exception handling
        return -1;
    }

    write(STDOUT_FILENO, "> ",2);
    while((len=read(STDIN_FILENO, buf, sizeof(buf)))>0){ // input command
        char *ptr=strtok(buf,"₩n"); // erase newline

        if(strcmp(ptr,"quit")==0){ // if command is quit
            write(socket_fd,ptr,strlen(ptr)); // send command to server
            len=read(socket_fd,buf,sizeof(buf)); // receive result from
server
            write(STDOUT_FILENO, buf, strlen(buf)); // print buf
            write(STDOUT_FILENO,"₩n",1); // newline
            exit(1); // program exit

        }


        if(write(socket_fd,ptr, strlen(ptr))>0){ // send command to server

            bzero(buf,sizeof(buf)); // initialize
            if((len=read(socket_fd,buf,sizeof(buf)))>0){ // receive result
```

```
from server
                                write(STDOUT_FILENO, buf, strlen(buf)); // print buf
                                write(STDOUT_FILENO,"₩n",1); // newline



                                bzero(buf,sizeof(buf)); // initialize
                        }
                        bzero(buf,sizeof(buf));// initialize
                }
                write(STDOUT_FILENO, "> ", 2);
        }
        close(socket_fd);
        return 0;
}
```

<Result Screen>