과제보고서 시스템프로그래밍 제출일 : 2020년 5월23일

<Assignment3-1>

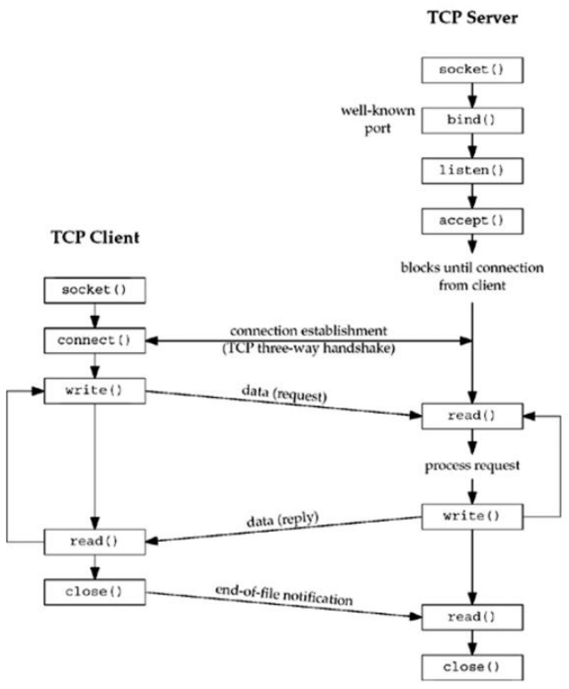
과제명	Assignment3-1 (FTP command “ls” implementation using Socket)			
과목명	시스템프로그래밍(H020-3-0922-01)(화 5 목 6)			
성명	정승훈	**연락처**	**핸드폰**	010–8648-7561
학과	전자통신공학과		**이메일**	tiktaktok116@naver.com
학번	2015707003	**지도교수**	김태석 교수님	
개발기간	2020.05.16 ~ 2020.05.23			
개발 환경	OS	Linux (Ubuntu 18.04 LTS)		
	Language	C		
	Development Tools	Visual Studio Code, gcc compiler		
	Library	sys/types.h, stdio.h, dirent.h, stdlib.h, string.h, unistd.h, sys/socket.h, sys/stat.h		
과제 요구사항 및 구현 내용	1. Client ➔ Implement the client module by using socket(), connect() and write().. - Request the user command (“ls", "quit") - Receive the result - Display the result of processing command 2. Server ➔ Implement the server module by using socket(), bind(), listen()..... - Listen the request of the client - Process return the result to the client - Display command - Display IP and port by using inet_ntoa(), ntohs()			

I. Introduction

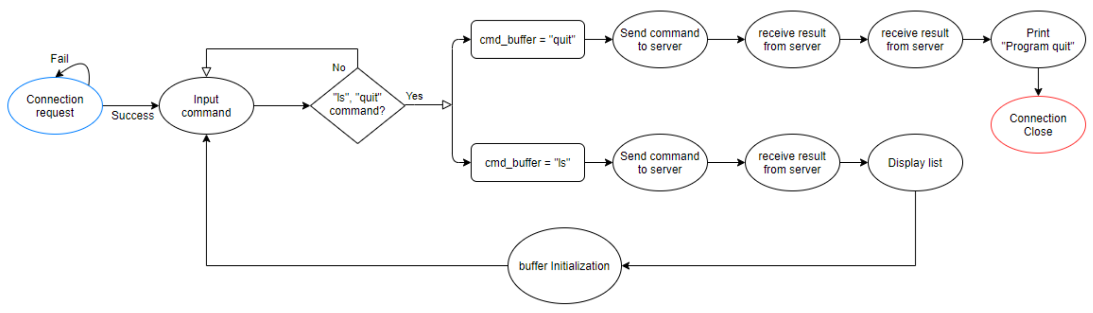
이번 Assignment3-1에서는 FTP “ls” command를 socket programming을 통해 구현하는 것이다. client에서 command 내용을 입력하면 그 정보를 server에 보내서 processing 한 후, 결과값을 다시 client로 가져와 출력하는 것이다. 쉽게 확인하기 위해서 loopback “127.0.0.1”을 사용하였다.

socket(), connect(), write() 함수를 사용해 client를 구현하고, socket(), bind(), listen() 등의 함수를 사용해 server를 구현한다.

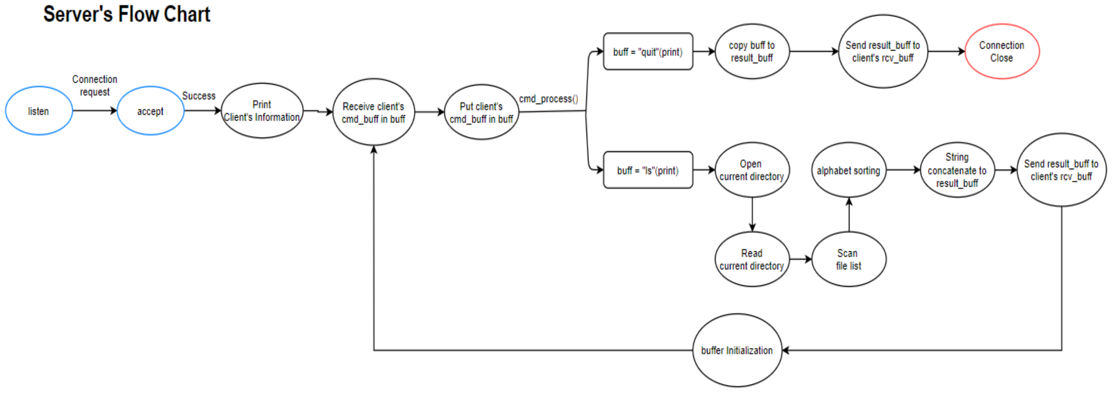
II. Flow Chart



Client's Flow Chart



Server's Flow Chart



III. Source Code

1) Client

```
////////////////////////////////////////////////////////////////////////
// File Name      : cli.c                                    //
// Date           : 2020/05/16 ~ 2020/05/22                  //
// OS             : Ubuntu 18.04.4 LTS                       //
// Student Name   : Seung Hoon Jeong                         //
// Student ID     : 2015707003                               //
// ----------------------------------------------------------------- //
// Title : System Programming Assignment #3-1                //
// Description : FTP command "ls" Implementation using socket (client) //
////////////////////////////////////////////////////////////////////////

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

#define MAX_BUFF 4096
#define RCV_BUFF 2048

//////////////////////////////////////////////////////////////////////
// Function : void process_result(char m_rcv_buff[RCV_BUFF])       //
// ================================================================== //
// Input: result_buff from server                            //
// Output: Print result                                      //
// Purpose: Check final result                               //
//////////////////////////////////////////////////////////////////////
void process_result(char m_rcv_buff[RCV_BUFF])
{
   printf("%s\n",m_rcv_buff);
}

//////////////////////////////////////////////////////////////////////////////////////////////
// Function : int main(int argc, char **argv)                                     //
// ========================================================================================== /
/
// Input: argument on kernel, > ls, > quit                                        //
// Output: 1.ls : File list result                                                //
//        2.quit : Program quit                                                   //
//                                                                       //
// Purpose: Receiving FTP command "ls" and "quit", Implementation and Sending result using socket //
//////////////////////////////////////////////////////////////////////////////////////////////
int main(int argc, char **argv)
{
   char buff[MAX_BUFF], cmd_buff[MAX_BUFF], rcv_buff[RCV_BUFF];
```

```
int n;
///////////////////////////////////open socket and connect to server///////////////////////////////////
int sockfd;
struct sockaddr_in servaddr;

/* argument count exception handling */
if(argc != 3) {
   printf("Usage :  %s [IP_ADDRESS] [PORT_NUMBER]\n", argv[0]);
   exit(0);
}

/* open socket */
if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
   perror("socket error");
   exit(1);
}

memset((char *)&servaddr, '\0', sizeof(servaddr)); // initialize server socket information struct to zero
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(atoi(argv[2])); // short data(port number) to network byte order
inet_pton(AF_INET, argv[1], &servaddr.sin_addr);

/* connect socket */
if(connect(sockfd, (struct sockaddr*)&servaddr, sizeof(servaddr)) < 0) {
   perror("connect");
   close(sockfd);
   exit(1);
}

for(;;) {
   printf("> ");
   if(fgets(cmd_buff,MAX_BUFF,stdin) == NULL) break; // input array string

   /* remove newline of cmd_buff */
   for(int i=0; cmd_buff[i] !=0; i++) {
      if(cmd_buff[i] == '\n') {
         cmd_buff[i] = 0;
         break;
      }
   }

   /* Command Excetion Handling */
   if(!strcmp(cmd_buff, "ls") == 0 && !strcmp(cmd_buff, "quit") == 0) {
      printf("Please use only 'ls' or 'quit'\n");
      bzero(cmd_buff, sizeof(cmd_buff));
      continue;
   }

   n = strlen(cmd_buff);
   /* send socket descriptor to server's buff */
   if(write(sockfd, cmd_buff, n) != n) {
```

```
            write(STDERR_FILENO, "write() error!!\n", sizeof("write() error!!\n"));
            exit(1);
        }

        /* receive from server's result_buff */
        if((n = read(sockfd, rcv_buff, RCV_BUFF-1)) < 0) {
            write(STDERR_FILENO, "read() error\n", sizeof("read() error\n"));
            exit(1);
        }

        rcv_buff[n] = '\0'; // null terminated

        if(!strcmp(rcv_buff, "quit")) {
            write(STDOUT_FILENO, "Program quit!!\n", sizeof("Program quit!!\n"));
            bzero(rcv_buff, sizeof(rcv_buff));
            exit(1);
        }
        process_result(rcv_buff); /*display ls (including options) command result */
        bzero(rcv_buff, sizeof(rcv_buff)); // initialize rcv_buff
        bzero(cmd_buff, sizeof(cmd_buff)); // initialize cmd_buff
    }
    close(sockfd);
    ///////////////////////////////////close socket and disconnect to server///////////////////////////////////
    return 0;
}
```

Colored by Color Scripter

2) Server

```
///////////////////////////////////////////////////////////////////////
// File Name      : srv.c                              //
// Date           : 2020/05/16 ~ 2020/05/22                  //
// OS             : Ubuntu 18.04.4 LTS                   //
// Student Name   : Seung Hoon Jeong                         //
// Student ID     : 2015707003                          //
// ------------------------------------------------------------------ //
// Title : System Programming Assignment #3-1                  //
// Description : FTP command "ls" Implementation using socket (Server) //
///////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <stdlib.h>
#include <string.h> // bzero(), ...
#include <unistd.h> // STDOUT_FILENO, ...
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <sys/stat.h>
#include <dirent.h>
#define MAX_BUFF 4096
#define SEND_BUFF 2048

```

```c
////////////////////////////////////////////////////////////////////
// Function : void *client_info(struct sockaddr_in *clientaddr)      //
// =============================================================== //
// Input: Client address's address                              //
// Output: Print client's Information                            //
// Purpose: Check client's IP ADDRESS and PORT                   //
////////////////////////////////////////////////////////////////////
void *client_info(struct sockaddr_in *clientaddr)
{
   printf("==========Client info==========\n");
   printf("client IP: %s\n", inet_ntoa(clientaddr->sin_addr)); // display IP by using inet_ntoa
   printf("\n");
   printf("client port: %d\n",ntohs(clientaddr->sin_port)); // display Port number by using ntohs()
   printf("================================\n");
}

/////////////////////////////////////////////////////////////////
// Function : static int select_files(const struct dirent *entry) //
// ============================================================//
// Input: file pointer                                   //
// Output: 0 : fail                                     //
//        1 : success                                  //
// Purpose: choose file list name                           //
/////////////////////////////////////////////////////////////////
static int select_files(const struct dirent *entry)
{
   if (entry->d_name[0] == '.')
      return 0;
   else
      return 1;
}

///////////////////////////////////////////////////////////////////////////////
// Function : void *cmd_process(char m_buff[MAX_BUFF], char m_result_buff[SEND_BUFF]) //
// ============================================================================= //
// Input: Client's cmd_buff and Server's result_buff                          //
// Output: cmd process result('ls', 'quit')                                  //
// Purpose: Shell command implementation at Server                            //
///////////////////////////////////////////////////////////////////////////////
void *cmd_process(char m_buff[MAX_BUFF], char m_result_buff[SEND_BUFF])
{
   DIR *dp = NULL;
   struct dirent *dirp = NULL;
   struct dirent **list = NULL;
   struct stat buf;
   char *temp = "\n";
   int n;
   ////////////////////////////////////////insert command to buffer///////////////////////////////////////////////
   if(!strcmp(m_buff, "ls")) { // ls command in buffer
      printf("%s\n",m_buff); // display command
      if((dp=opendir(".")) == NULL) { // open current directory
```

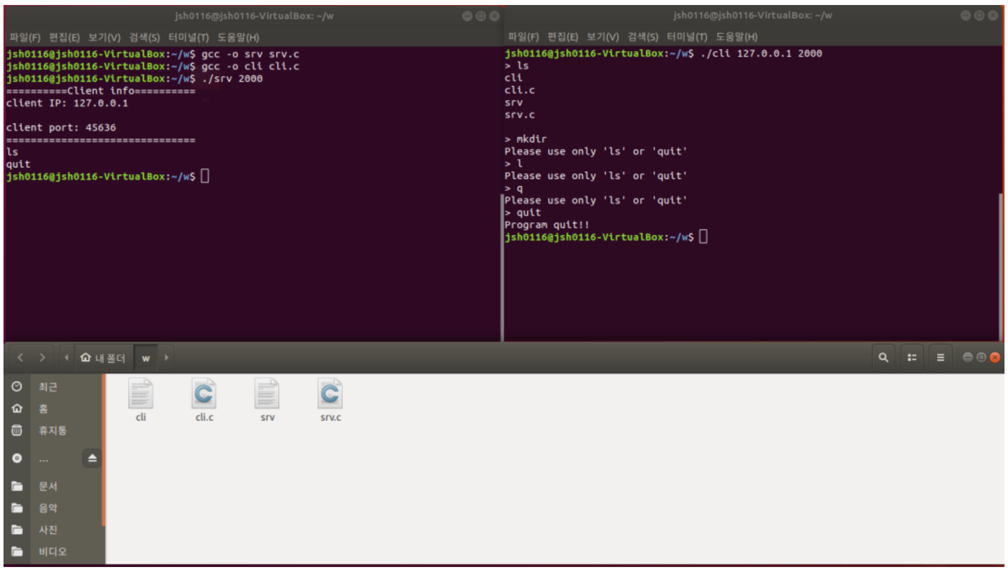
```
            printf("Can't Open this directory\n");
            exit(1);
        }
        else {
            while((dirp = readdir(dp)) != NULL) { // read directory
                if(dirp->d_ino == 0) continue; // skip if it doesn't have a i-nooe information
                if(!strcmp(dirp->d_name, ".") || !strcmp(dirp->d_name,"..")) //except current and parent directory
                    continue;
            }
            /* scandir, alphasort file_list and insert to result_buff  */
            if((n = scandir(".", &list, select_files, alphasort)) < 0) {
                perror("scandir");
                exit(1);
            }
            for(int index=0; index<n; index++) {
                strcat(m_result_buff,list[index]->d_name);
                strcat(m_result_buff,temp);
                free(list[index]);
            }
            free(list);
            closedir(dp);
        }
    }

    else if(!strcmp(m_buff, "quit")) { // quit command in buffer
        strcpy(m_result_buff,m_buff); // buffer string copy to result buffer
        m_result_buff[strlen(m_result_buff)-1] == '\0'; // remove result buffer's newline
    }
    ////////////////////////////////////////End of insert command/////////////////////////////////////////////////
}

//////////////////////////////////////////////////////////////////////////////////////////////
// Function : int main(int argc, char **argv)                              //
// ========================================================================================= /
/
// Input: argument on kernel                                               //
// Output: 1.ls : ls command processing OK                                 //
//        2.quit : quit command processing OK                              //
// Purpose: Receiving FTP command "ls" and "quit", Implementation and Sending result using socket //
//////////////////////////////////////////////////////////////////////////////////////////////
int main(int argc, char **argv)
{
    char buff[MAX_BUFF], result_buff[SEND_BUFF];
    int n;
    /* open socket and listen */
    struct sockaddr_in servaddr, cliaddr;
    int listenfd, connfd, option = 1;
    int clilen = sizeof(cliaddr);

    /* check argument count */
    if(argc !=2) {
```

```
    printf("Usage : %s [PORT_NUMBER]",argv[0]); // fixed argument format
    exit(0);
  }

  /* open socket */
  if((listenfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("socket");
    exit(0);
  }

  /* prevent bind error after server terminated */
  if(setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &option, sizeof(option)) < 0) {
    perror("setsockopt");
    exit(1);
  }

  memset((char *)&servaddr, '\0', sizeof(servaddr)); // initialize server socket information struct to zero
  servaddr.sin_family = AF_INET;
  servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
  servaddr.sin_port = htons(atoi(argv[1]));

  /* bind socket */
  if(bind(listenfd, (struct sockaddr*)&servaddr, sizeof(servaddr)) < 0) {
    perror("bind");
    exit(0);
  }

  /* listen socket */
  if(listen(listenfd,5) < 0) {
    perror("listen");
    exit(0);

  }

  for(;;) {

    /* accept socket descriptor */
    if((connfd = accept(listenfd, (struct sockaddr *)&cliaddr, &clilen)) <0) {
      perror("accept");
      exit(0);
    }

    if(client_info(&cliaddr) < 0) /* display client ip and port */
      write(STDERR_FILENO, "client_info() err!!\n", sizeof("client_info() err!!\n"));

    while(1) {
      n = read(connfd, buff, MAX_BUFF); // read from client's cmd_buff
      buff[n] = '\0'; // null terminated
      bzero(result_buff,sizeof(result_buff)); // garbage terminated

      if(cmd_process(buff, result_buff) < 0) //command execute and result
```

```
            {
                write(STDERR_FILENO, "cmd_process() err!\n",sizeof("cmd_process() err!\n"));
                break;
            }

            write(connfd, result_buff, strlen(result_buff)); // put result_buffer in file descriptor
            result_buff[n] = '\0'; // null terminated

            if(!strcmp(result_buff, "quit")) // "quit" command on result_buff after cmd_process()
            {
                write(STDOUT_FILENO, "quit\n", sizeof("quit\n"));
                close(connfd); // connecting descriptor terminated
                break;
            }
        }

        /* for one more loop break */
        if(!strcmp(result_buff, "quit")) {
            bzero(result_buff, sizeof(result_buff)); // initialize result_buff
            bzero(buff,sizeof(buff)); // initialize buff
            break;
        }
    }
    close(listenfd); // socket terminated
    return 0;
}
```

Colored by Color Scripter

IV. Result


jsh0116@jsh0116-VirtualBox: ~/w
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
내 폴더
w
최근
홈
휴지통
문서
음악
사진
비디오
cli
cli.c
srv
srv.c

```
jsh0116@jsh0116-VirtualBox:~/w$ gcc -o srv srv.c
jsh0116@jsh0116-VirtualBox:~/w$ gcc -o cli cli.c
jsh0116@jsh0116-VirtualBox:~/w$ ./srv 2000
==========Client info==========
client IP: 127.0.0.1

client port: 45636
===============================
ls
quit
jsh0116@jsh0116-VirtualBox:~/w$ 
```

```
jsh0116@jsh0116-VirtualBox:~/w$ ./cli 127.0.0.1 2000
> ls
cli
cli.c
srv
srv.c

> mkdir
Please use only 'ls' or 'quit'
> l
Please use only 'ls' or 'quit'
> q
Please use only 'ls' or 'quit'
> quit
Program quit!!
jsh0116@jsh0116-VirtualBox:~/w$ 
```