시스템 프로그래밍 실습

Assignment3 - 2

Class : **D**

Professor : **최상호 교수님**

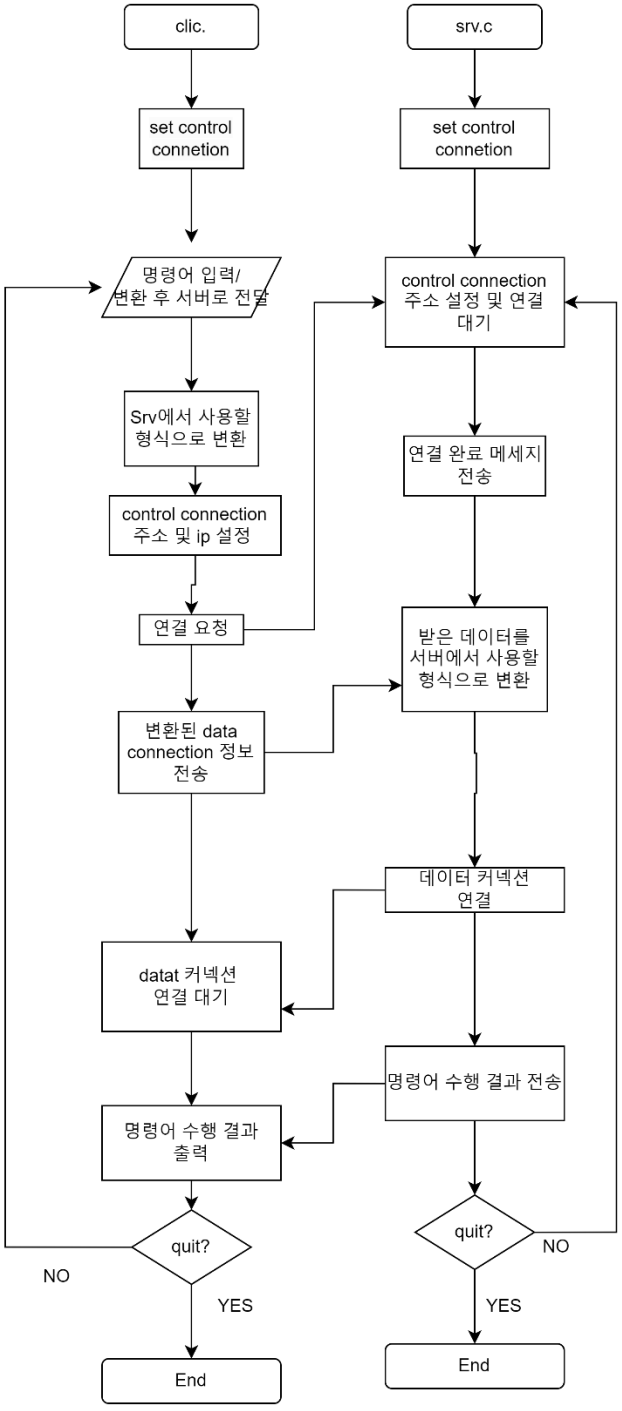
Student ID : **2020202037**

Name : **염정호**

Introduction

본 프로젝트는 c 를 이용한 리눅스 서버를 구현한다. 이전 프로젝트와는 달리 명령어의 처리와 해당 결과를 전송하는 소켓을 달리해 서버를 구현해야 한다. 총 2 개의 소켓을 사용하게 되는데 data connection 과 control connection 이다. 프로그램 시작 시 클라이언트는 서버와의 데이터를 주고받을 데이터 커넥션의 포트 번호로 10000 – 30000 중 하나를 선택해 이용하게 된다. 해당 포트 번호와 ip 주소를 서버에서 활용할 형식으로 변환해 서버로 전달하고 서버는 이를 활용해 클라이언트에게 데이터 커넥션을 요청 할 수 있다. 이때 두 프로그램 간의 통신에 사용되는 것이 컨트롤 커넥션이며 각 명령의 진행 여부를 서버는 클라이언트에게 알려준다.
클라이언트에서 명령어를 입력할 경우 데이터 커넥션 연결을 시도하고 해당 명령어의 수행 결과를 데이터 커넥션을 통해 서버에서 클라이언트로 전송하면 된다. 서버와 클라이언트의 종료를 위해서 quit 명령어를 입력하도록 한다

Flow chart


clic.
srv.c
set control connetion
set control connetion
명령어 입력/ 변환 후 서버로 전달
control connection 주소 설정 및 연결 대기
Srv에서 사용할 형식으로 변환
연결 완료 메세지 전송
control connection 주소 및 ip 설정
연결 요청
받은 데이터를 서버에서 사용할 형식으로 변환
변환된 data connection 정보 전송
데이터 커넥션 연결
datat 커넥션 연결 대기
명령어 수행 결과 전송
명령어 수행 결과 출력
quit?
quit?
NO
NO
YES
YES
End
End

Pseudo code

```
void trim_whitespace(char *str);
char *convert_addr_to_str(unsigned long ip_addr, unsigned int port);
void print_data(const char *print); // Function to print execution results

void main(int argc, char **argv)
{

    control_sockfd = socket(AF_INET, SOCK_STREAM, 0);

    memset(컨트롤 소켓 생성)
    sock.sin= 소켓 주소 할당

    srand(time(0));
    connect(컨트롤 커넥션 연결 요청);

    while (1)
    {

        memset(입력 버퍼 초기화);
        read(명령어 입력);
        user_input[strlen(user_input) - 1] = '₩0'; // Remove newline character
        trim_whitespace(user_input);

        divide = strtok(공백 기준으로 명령어 분리)
        int index_num = 0;

        while (divide != NULL)
        {
            index[index_num] = divide;  // Store the cut string
            divide = strtok(NULL, " "); // Find the next token
            index_num++;                // Move to the next array index
        }

        // Process the 'ls' command
        if (strcmp(입력받은 명령어가 ls인지 확인)
```

```
{
    if (index_num == 1) 명령어만 들어온 경우
    {
        to_srv = (char *)malloc(strlen("NLST") + 1); // Allocate memory and insert data
        strcpy(to_srv, "NLST");
    }
    else if (index_num == 2) // 'ls' + 주소
    {
        to_srv = (char *)malloc(strlen("NLST") + strlen(index[1]) + 2); // Allocate memory and insert data
        strcpy(to_srv, "NLST ");
        strcat(to_srv, index[1]);
    }
}
else if (strcmp(quit 명령이 들어온 경우) == 0)
{
    to_srv = (char *)malloc(strlen("QUIT") + 1); // Allocate memory and insert data
    strcpy(to_srv, "QUIT");
}
else
{
    to_srv = (char *)malloc(strlen("wrong") + 1); // Allocate memory and insert data
    strcpy(to_srv, "wrong");
}
write(서버로 입력받은 데이터를 보냄); // Send the command to the server
free(출력 버퍼 초기화);                                    // Free the allocated memory
to_srv = NULL;                                    // Reset the pointer

int random_port = 랜덤 포트 번호 설정

memset(주소 초기화);

temp.sin = 데이터 소켓 주소 설정
hostport = convert_addr_to_str(서버에게 보낼 형식으로 변환);
if (보내는 명령어가 quit이 아닐경우)
{
    print_data("converting to ");
    print_data(hostport);
```

```
    print_data("₩n");
}
write(포트 번호 전송));


while (1)
{
    n = read(응답 대기);
    client_output[n] = '₩0'; // Ensure null-terminated string

    if (strcmp(client_output, "ready") == 0)
        break;

    print_data(명령어 전달 후 응답 출력);
    print_data("₩n");
    memset(client_output, 0, sizeof(client_output));
    write(control_sockfd, " ", 1); // Send acknowledgment
}

memset(client_output, 0, 4000);

// Create the data socket
data_sockfd = socket(데이터 전송 소켓 설정);
if (data_sockfd < 0)
{
    perror("socket error");
    exit(1);
}

// Set socket option SO_REUSEADDR
int optval = 1;

if (setsockopt(소켓 설정)
{
    perror("setsockopt");
    exit(1);
}
```

```
// Bind the data socket to the temporary address
if (bind(data_sockfd, (struct sockaddr *)&temp, sizeof(temp)) < 0)
{
    perror("bind error");
    exit(1);
}

if (listen(data_sockfd, 5) < 0)
{
    perror("listen error");
    exit(1);
}

int len = sizeof(temp);

write(control_sockfd, " ", 1); // Send acknowledgment

data_srv = accept(데이터 소켓 연결);
if (data_srv < 0)
{
    perror("accept error");
    exit(1);
}

if (!(strcmp(quit명령이 아닐 때 == 0))
{
    read(data_srv, client_output, sizeof(client_output));
    print_data(client_output);
    n = strlen(client_output);

    memset(client_output, 0, 4000);
    write(data_srv, " ", 1); // Send acknowledgment

    read(data_srv, client_output, sizeof(client_output));
    print_data(client_output);
    print_data("₩n");
```

```
            if (atoi(client_output) == quit이 아닌 경우 응답받은 데이터 길이 출력)
            {
                print_data("OK. ");
                sprintf(totalBytesReadStr, "%d", n); // Convert the number of bytes received to a string
                print_data(totalBytesReadStr);
                print_data(" bytes received.₩n");
            }
            close(데이터 소켓 종료); // Close the data connection
        }
        else
        {
            read(data_srv, client_output, sizeof(client_output));
            print_data(client_output);
            break; quit 입력시 종료
        }
        close(data_sockfd);
        close(data_srv); // Close the data connection
        memset(client_output, 0, 4000);
        memset(totalBytesReadStr, 0, 20);
    }
    return;
}

char *convert_addr_to_str(unsigned long ip_addr, unsigned int port)
{
    struct in_addr ip_addr_struct;
    ip_addr_struct.s_addr = ip_addr;

    int random_port = ntohs(port);

    char ip_str[INET_ADDRSTRLEN];
    char *addr;

    if (inet_ntop(AF_INET, &ip_addr_struct, ip_str, INET_ADDRSTRLEN) == NULL)
    {
```

```
        perror("inet_ntop");
        exit(EXIT_FAILURE);
    }

    uint8_t upper_8_bits = 상위 8비트 정수로 변환
    uint8_t lower_8_bits = 하위 8비트 정수로 변환
    addr = 데이터 저장공간 할당

    if (addr == NULL)
    {
        perror("malloc");
        exit(EXIT_FAILURE);
    }
    int ip1, ip2, ip3, ip4;
    sscanf(각 데이터 분리);

    // Format the IP address and port
    snprintf(각 분리한 숫자들을 모아 문자열로 재구성);

    return addr;
}
```

SRV.c

```
 void main(int argc, char **argv)
 {
      control_fd = socket(컨트롤 소켓 설정);

      memset(소켓 정보 초기화)
      control_addr.sin_family = AF_INET;
      control_addr.sin_addr.s_addr = htonl(INADDR_ANY); // Use any available address
      control_addr.sin_port = htons(atoi(argv[1]));       // Use the provided port number

      bind(control_fd, (struct sockaddr *)&control_addr, sizeof(control_addr));
      listen(클라이언트의 연결 대기);
      int len = sizeof(control_addr);
      control_cli = accept(연결요청이 들어온 경우 연결);


        while (quit입력시 까지 반복)
          {

      read(control_cli, temp, 25);
      temp[strlen(temp)] = '₩0';
       if (strcmp(입력받은 명령어가 quit가 아닐경우 )
            {
                print_data(temp);
                print_data("₩n");
            }

      print_data(입력받은 데이터 커넥션 주소/ 포트 출력);
      print_data("₩n");
      host_ip = convert_str_to_addr(주소 설정에 사용할 값으로 변경);
      n = read(클라이언트의 명령어 입력 대기);
      while (입력받은 데이터를 공백을 기준으로 분류)
      {
           s_index[num_of_sindex] = index; // Store the tokenized string
           num_of_sindex++;                    // Move to the next index in the array
           index = strtok(NULL, " ");         // Find the next token
      }

      if (strcmp(s_index[0], "NLST") == 0)
```

```
{
    write_code(성공 메시지 출력); // Send a success code
}
else if (strcmp(s_index[0], "QUIT") == 0)
        {
            write(출력 없음);
        }
else
{
    write_code(잘못된 명령어 임을 알림); // Send an error code
    return;
}
read(control_cli, char_1, sizeof(char_1));
memset(char_1, 0, 100);

if (strcmp(input_form_client, "NLST") == 0)
{
    DIR *dp = NULL;              // Variable for directory traversal
    struct dirent *dirp = 0; // Structure to hold information about files within a directory
    char *dir = NULL;           // Variable to store the directory address to be searched

    char *files[100];    // Array to store the list of files
    int file_count = 0; // Integer to store the number of files in the list

    if (입력받은 인자 개수에 따라 설정 1 개인 경우)
    { // If only the command exists
        print_data("NLST₩n");
        dir = "."; // Use the current directory
    }
    else if (인자가 2 개인 경우) // If two arguments are received
    {
        print_data("NLST ");
        print_data(s_index[1]);
        print_data("₩n");
        dir = s_index[1]; // Use the specified directory
    }

    if (opendir(dir) ==디렉토리 열기)
    {
```

```
    if (에러발생 & 접근  권한이  없는  경우)
        strcpy(concatenate_output, "cannot access: Access denied₩n");
    else
        strcpy(concatenate_output, "No such directory₩n");

    ls_error = 501;
}
else
{
    dp = opendir(dir);
    write_code(150, control_cli);

    read(control_cli, char_1, sizeof(char_1));
    memset(char_1, 0, 100);

    while ((디렉토리  탐색)
    {
        char file_path[300];
        struct stat file_stat;                          // String to store file information
       sprintf(file_path, "%s/%s", dir, dirp->d_name);

        files[file_count] = dirp->d_name; // Store the file name
        stat(file_path, &file_stat);        // Get information about the file
        if (해당  파일이  디렉토리인  경우)
        {
            strcat(files[file_count], "/"); // Append '/' if it's a directory
        }
        file_count++; // Next file
    }
    closedir(dp);
    bubbleSort(파일을  이름  순으로  정렬); // Sort the list of file names
    print_non_option(파일  출력); // Print the sorted file names
    ls_error = 226; 성공 메시지 출력 코드

    else if (strcmp(s_index[0], "QUIT") == 0)
  {
        print_data("QUIT₩n");
         ls_error = 221;
      }
```

```
        }

        write(데이터 커넥션 준비를 대기); // Indicate ready for data transfer
        read(control_cli, char_1, 100);
        // Create a data socket
        data_fd = socket(AF_INET, SOCK_STREAM, 0);

        // Set up the data address struct
        memset(데이터 커네션 주소 설정));
        data_addr.sin_family = AF_INET;
        data_addr.sin_addr.s_addr = inet_addr(host_ip); // Use the host IP address
        data_addr.sin_port = htons(port_num);              // Use the port number

        // Connect to the client for data transfer
        connect(데이터 커넥션 연결 요청)

        // Send the concatenated output to the client
        n = write(결과 전송);

        read(data_fd, char_1, sizeof(char_1));
        write_code(ls_error, data_fd); 성공 or 실패 메시지 전송

        close(data_fd); // 종료
    }

        if (strcmp(s_index[0], "QUIT") == 0)
          {
            close(control_cli);
            break;
          }quit인 경우 종료

}
    return;
 }

 char *convert_str_to_addr(char *str, unsigned int *port)
 {
    char *addr;
    int ip1, ip2, ip3, ip4;
    unsigned int upper_8_bits, lower_8_bits;
```

```
    addr = (char *)malloc(20);
    sscanf(입력받은 데이터의 형태를 변환);
    // Convert the IP address to dot-decimal notation
    snprintf(형태 변환 후 저장.);

    *port = (upper_8_bits << 8) | lower_8_bits;

    return addr;
}
```

결과화면

cli.c

```
kw2020202037@ubuntu:~/Assignment3_2_D_2020202037_염정호$ ./cli 127.0.0.1 5000
ls
converting to 127,0,0,1,78,184
200 Port command performed successful
150 Opening data connection for directory list.
cli             cli.c             dd/             makefile             srv
    srv.c

226 Complete transmission.
OK. 107 bytes received.
ls dd
converting to 127,0,0,1,41,230
200 Port command performed successful
150 Opening data connection for directory list.
dsf (3번째 사본)/             dsf (4번째 사본)/             dsf (5번째 사본)/
       dsf (또 다른 사본)/             dsf (사본)/             dsf/

226 Complete transmission.
OK. 184 bytes received.
quit

221 Goodbye.kw2020202037@ubuntu:~/Assignment3_2_D_2020202037_염정호$
```

srv.c

```
gcc srv.c -o srv
kw2020202037@ubuntu:~/Assignment3_2_D_2020202037_염정호$ ./srv 5000
127,0,0,1,78,184
200 Port command performed successful
NLST
150 Opening data connection for directory list.
226 Complete transmission.
127,0,0,1,41,230
200 Port command performed successful
NLST dd
150 Opening data connection for directory list.
226 Complete transmission.
QUIT
221 Goodbye.
kw2020202037@ubuntu:~/Assignment3_2_D_2020202037_염정호$
```

고찰

프로젝트 수행 중 서버 측에서 두 번의 write를 통해 보낸 내용이 클라이언트 측에서 한 번의 read로 모두 읽히는 상황이 발생했습니다. 이 문제는 매번 프로그램이 실행될 때 발생하는 것은 아니🤔지만, 두 번 중 한 번 정도 빈도로 발생하여 원인 파악이 어려웠습니다. 조사 결과, 연속적인 write 수행 시 데이터가 버퍼에 동시에 저장될 수 있어 read 한 번으로 해당 내용을 모두 읽어온다는 것을 확인했습니다. 이 문제를 해결하기 위해 write 이후 read를 수행하고, 해당 값을 바로 버리는 코드를 추가했습니다. 또한, 클라이언트 측에서도 read 후 공백을 보내는 코드를 추가하여, 서로 번갈아 가며 read와 write를 수행할 수 있도록 수정했습니다. 이를 통해 문제를 해결할 수 있었습니다.