

시스템 프로그래밍 실습

[FTP_Assignment 2-3]

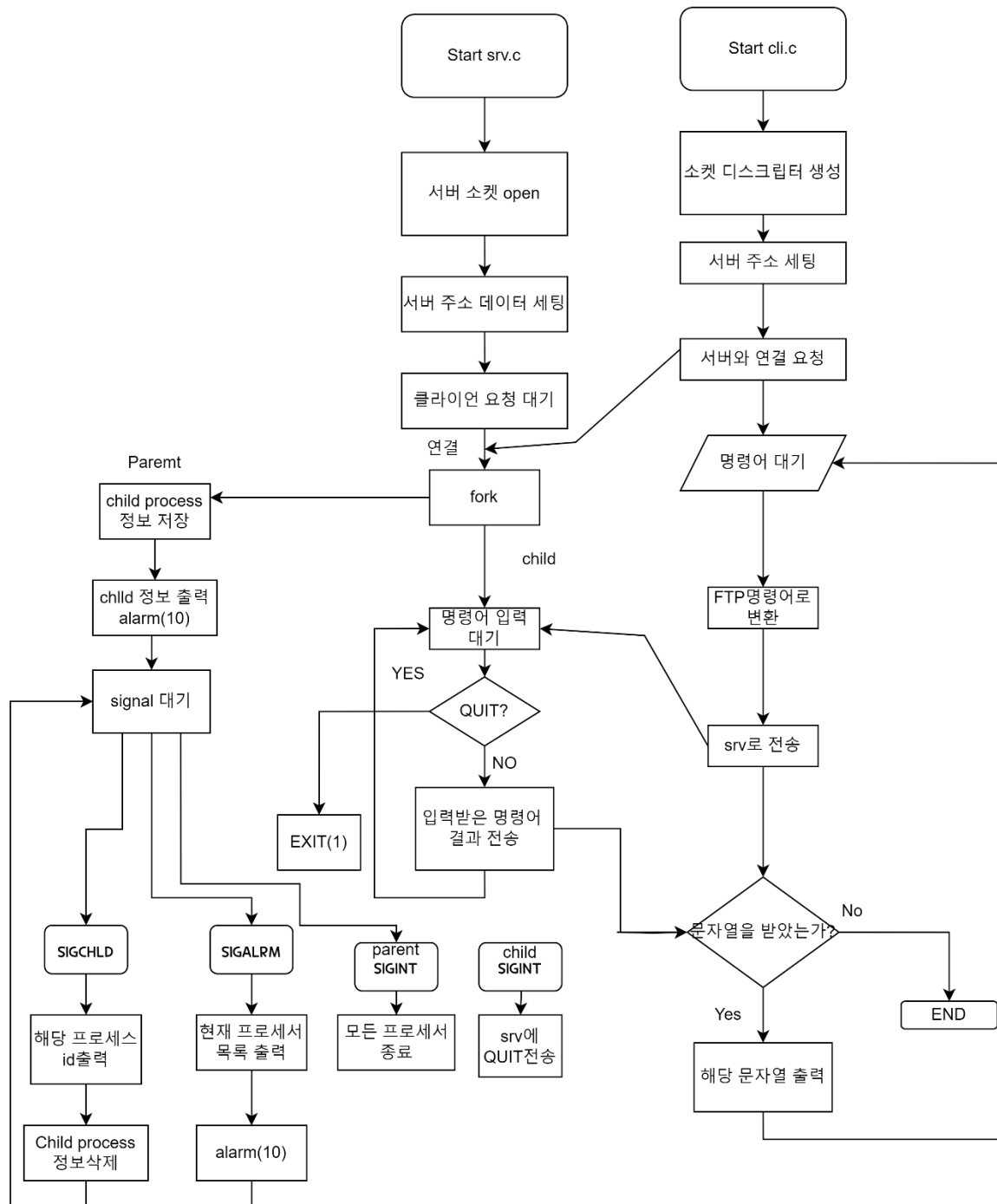
Class : D
Professor : 최상호 교수님
Student ID : 2020202037
Name : 엄정호

Introduction

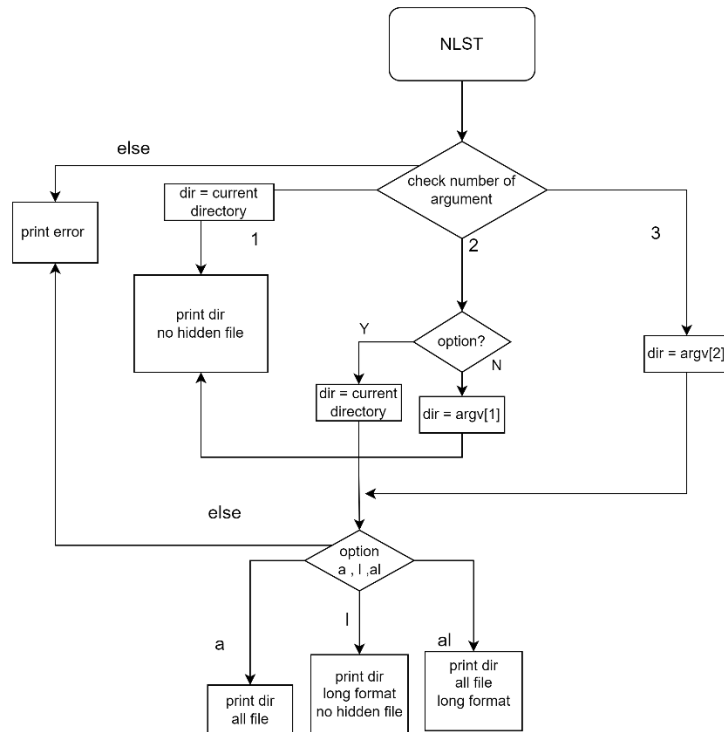
이번 과제에서는 이전에 구현한 FTP 명령어와 FTP 서버 프로그램을 이용해 클라이언트 부분에 사용자가 명령어 입력시 FTP 명령어로 변환후 서버로 해당 명령어를 넘기고 서버에서 해당 명령어 수행 결과를 다시 클라이언트에게 전달하는 프로그램을 구현해야 한다. 구현해야 하는 명령어의 종류는 1-3 과제에서 수행한 것과 같으며 QUIT 명령어 입력시 해당 클라이언트를 종료시킨다. 서버는 여러 사용자의 입력을 받을 수 있으며 이를 위해 클라이언트 마다 자식 프로세스를 생성한다. 이에 대한 확인으로 새로운 자식 프로세스 생성 또는 10 초마다 현재 운영중인 프로세스 정보를 출력하며 서버가 종료되는 경우 모든 자식 프로세스를 종료하며 클라이언트와 연결을 종료한다.

Flow chart

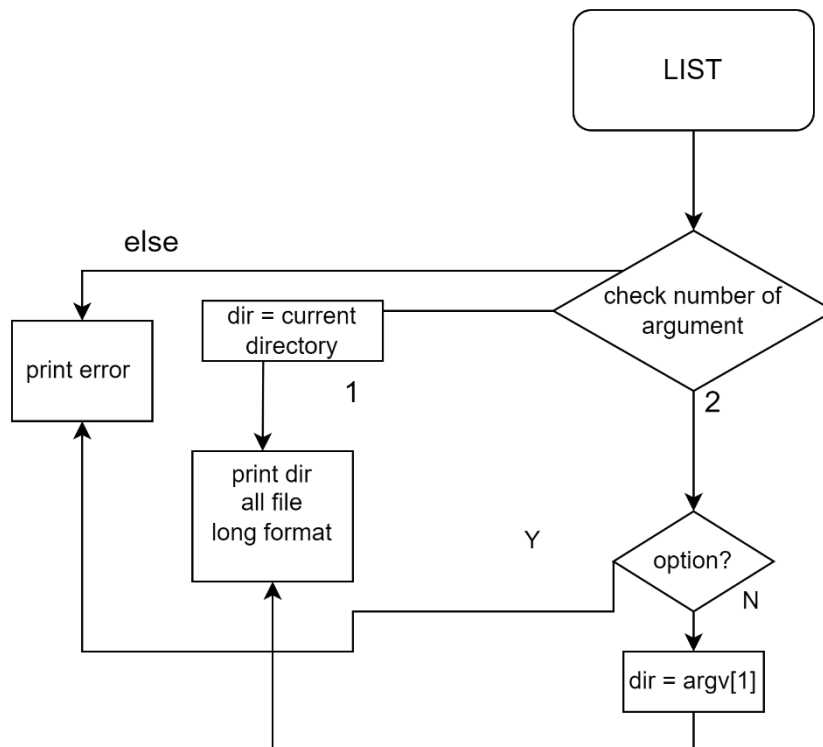
프로그램 전체 흐름도



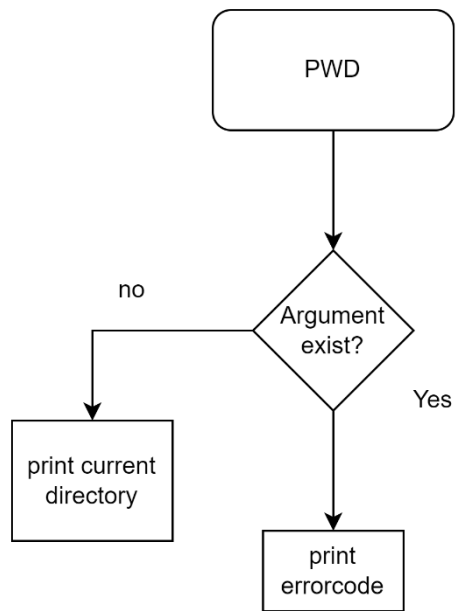
명령어 NSLT



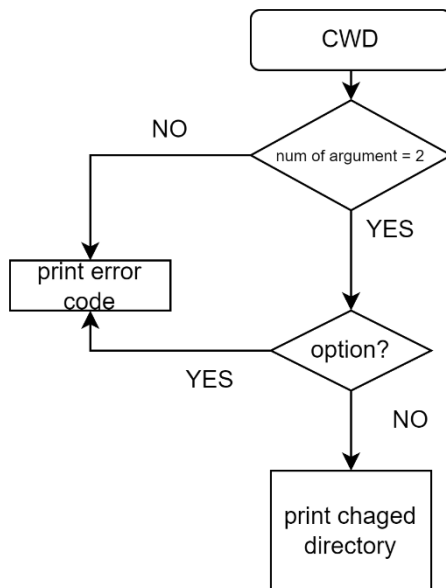
LIST



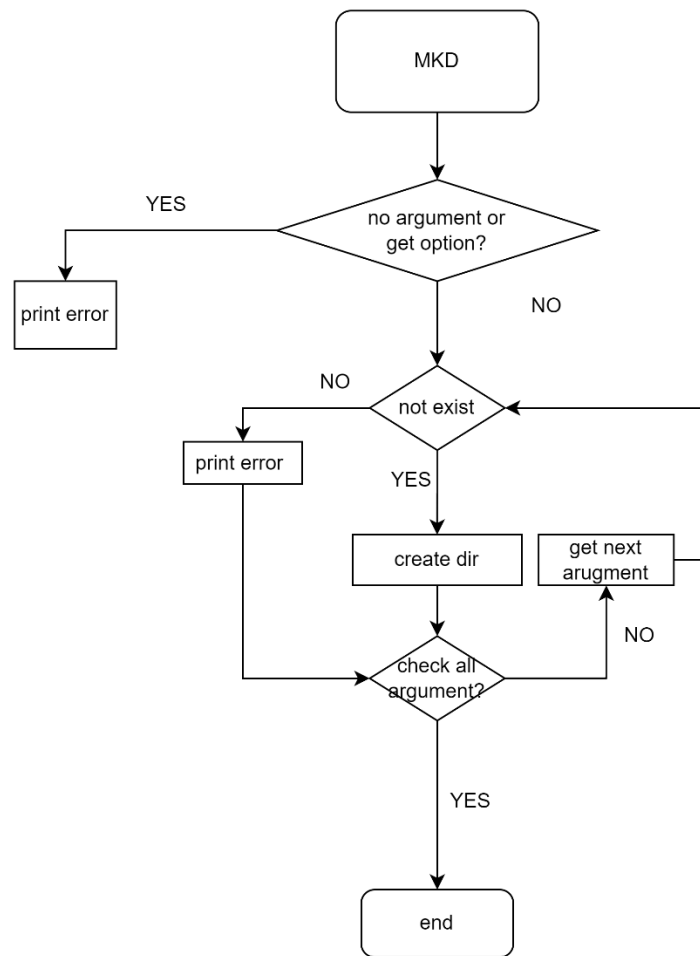
PWD



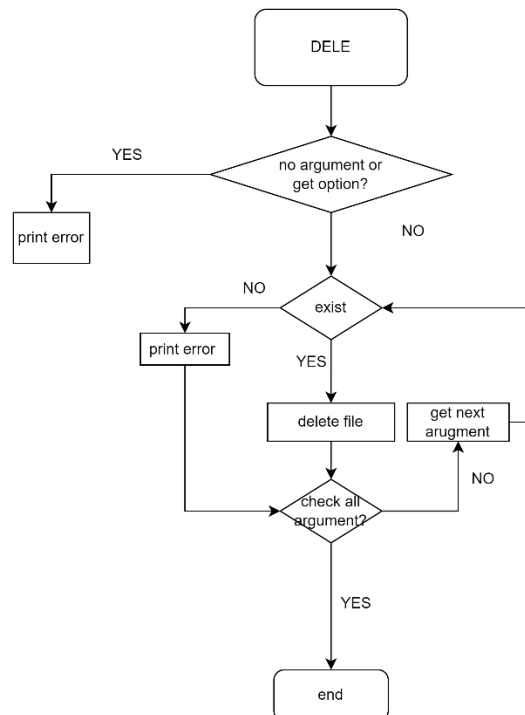
CWD



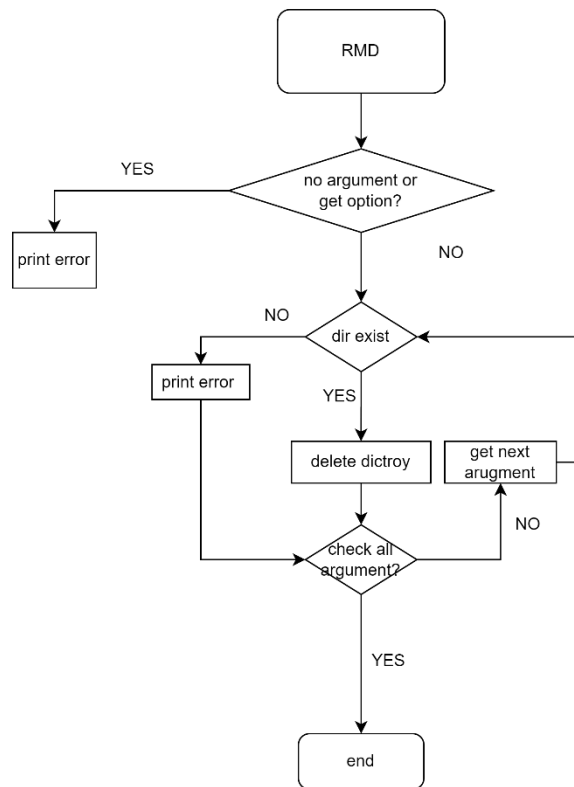
MKD



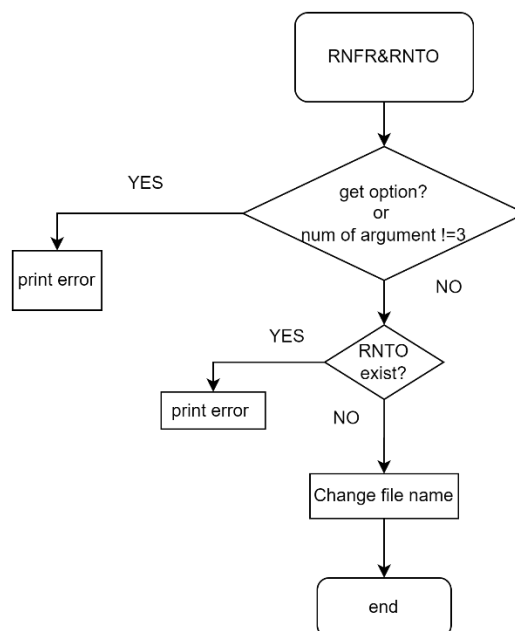
DELE



RMD



RNFR RNT0



각 명령어별 알고리즘은 1-3 과제에서 수행한 것과 동일한 동작 수행

Pseudo code

```
int sockfd; // Socket descriptor
int main(int argc, char **argv)
{
    void sh_int(int); CTRL 발생시

    signal(SIGINT, sh_int); /* Applying signal handler(sh_chld) for SIGINT */
    char
    char *command[] = {"ls", "dir", "pwd", "cd", "mkdir", "delete", "rmdir", "rename", "quit",
NULL}; // List of commands
    char *convert[] = {"NLST", "LIST", "PWD", "CWD", "MKD", "DELE", "RMD", "RNFR&RNTO",
"QUIT", NULL}; // Corresponding commands to be sent to the server

    sockfd = 소켓 생성

    memset 소켓 정보 리셋) // Initialize address structure
        소켓에 포트정보, 주소 할당
    connect(서버에 연결 요청 ); // Connect to server

    while (1)
    {
        입력 버퍼 초기화

        ssize_t bytes_read = read(0, data_in, BUF_SIZE); // Read user input

        if (data_in[0] == '\n')
        {
            write(sockfd, "\n", strlen("\n")); // 개행 문자만 받은 경우 전송
            if (!(read(sockfd, recv_from_srv, 4000) > 0)) // 서버가 연결 됐는지 확인
            {
                close(sockfd);
                return 0; // 서버 종료시 종료
            }
            else /// 서버가 정상 작동하는 경우 그대로 수행
            {
```



```

        memset(recv_from_srv, 0, 4000); // 버퍼를 지우고 다시 입력 받을 준비
        continue;
    }
}

while (divide != NULL)
{
    문자열을 개행문자 기준으로 분리 // Move to next index
}

cmd_num = 0; //
while (command[cmd_num] != NULL)
{
    입력받은 명령어가 명령어 목록에 존재하는지 확인
}

if (command[cmd_num] == NULL)
{
    존재하지 않는 경우 변환하지 않고 그대로 보내 서버에서 에러 출력
}

else if (index_num == 1) // Command without arguments
{
    인자가 하나인 경우 명령어 변환후 전송
}

else if (index_num == 2) // Command with one argument
{
    인자가 2 개인 경우 명령어 변환 후 각 인자들을 합쳐서 전송
}

else if (index_num > 2) // Command with multiple arguments
{
    인자가 2 개 이상이 경우
    to_srv = (char *)malloc(문자열 길이 만큼 공간 할당);
    for (k = 1; k < index_num; k++)

```

```

        strcat(to_srv, convert[cmd_num]); // FTP 명령어 할당

        for (k = 1; k < index_num; k++) // Append arguments
        {
나머지 인자들을 공백 기준으로 붙임
        }
    }

    to_srv[strlen(to_srv)] = '\0'; // 널문자 삽입

    if (write(sockfd, to_srv, strlen(to_srv)) != strlen(to_srv)) 서버로 전송
    {
전송 실패시 에러 출력
        exit(1);
    }

    if (n = read(sockfd, rcv_from_srv, 4000) > 0) // Receive response from server
    {
        printf 서버로부터 받은 데이터 출력); // Print received data
        memset(입력 버퍼를 비움); // Clear receive buffer
    }

    else
        break; 입력이 들어오지 않을 시 반복 종료
    }
    close(sockfd); // 소켓 종료

    return 0;
}

void sh_int(int signum)
{
    write(서버에 QUIT 전송); // Send "QUIT" message to server
    close(sockfd); // Close socket
}

```

```

    exit(1);                                // Exit program
}

void print_data(const char *print) // Function to print execution results
{
    int length = strlen(print);
    write(1, print, length);
    return;
}

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

    void sh_chld(int); // Signal handler for SIGCHLD
    void sh_alrm(int); // Signal handler for SIGALRM
    void sh_int(int);  // Signal handler for SIGINT

    signal(SIGCHLD, sh_chld);
    signal(SIGALRM, sh_alrm);
    signal(SIGINT, sh_int);

    server_fd = socket(서버 디스크립터 새엇ㅇ); // Get server file descriptor

    memset(디스크립터 데이터 초기호);
    서버 주소, 포트번호 도메인 설정

    bind(server_fd, (struct sockaddr *)&server_addr, sizeof(server_addr)); // Bind

    listen(클라이언트 요청 대기); // Listen for incoming connections

    while (1)
    {
        client_fd = accept(클라이언트 에서 연결 요청시 연결)

```

```

if ((pid = fork()) < 0)
{
}
else if (pid == 0) //자식 프로세스
{ // Child process: receive data from the client

    while (1)
    {
        n = read(클라이언트로부터 데이터를 입력받음)
        input_buf[n] = '\0'
        printf(입력한 프로세스 아이디와 받은 데이터 출력);
        index = strtok(input_buf, " ");    공백 기준 분리
        num_of_sindex = 0;

        while (index != NULL) // Repeat until there are no more strings to tokenize
        {
            공백을 기준으로 분리해 배열에 저장
        }

        if (strcmp(s_index[0], "NLST") == 0) // If the command is ls
        {

            DIR *dp = NULL; 파일 디스크립터 생성

            if (num_of_sindex == 1)
            {
                // If only the command exists
                dir = "."; // Output the current directory
            }
            else if (num_of_sindex == 2) // If two arguments are received
            {
                if (옵션이 들어온 경우) // If the second argument is an option
                {
                    dir = "."; // Directory - current directory
                    option_i = check_option(s_index[1]); // Classify the option
                }
            }
        }
    }
}

```

```

options
    if (option_i == -1) // Exception handling for
        print_error(); // Print error
    }
    else// 디렉토리 주소가 들어온 경우
        dir = s_index[1]; // If it's not an option, dir is the given address
}
else if (num_of_sindex == 3)
{
    dir = s_index[2];
    option_i = 존재하는 옵션인지 확인
    if (option_i == 존재하지 않는 옵션인 경우)
    {
        print_error(); // Print error if there's a problem with the option
    }
}
else
{
    strcat(concatenate_output, "too much input\n");
    write(에러 출력);
    memset(concatenate_output, 0, 4000); // recv_from_srv 초기화
    continue;
}
if (디렉토리 파일 탐색)
{
    if (권한 없음)
        strcat(concatenate_output, "cannot access : Access denied \n");
    else
        strcat(concatenate_output, " No such directory\n");
}
else
{
    dp = opendir(dir); // Traverse the directory

    while ((파일을 다 읽을 때 까지 반복)

```

```

{
    파일 목록 저장
}
closedir(dp); // Close the directory

bubbleSort(files, file_count); // 파일 목록 이름순으로 정렬

switch (option_i) // Execute functions according to the option
{
case 0:
    print_non_option(files, file_count, client_fd);
    // If there's no option
    break;
case 1:
    print_a(files, file_count); // If the option is 'a'
    break;
case 2:
    print_l(files, file_count); // If the option is 'l'
    break;
case 3:
    print_al(files, file_count); // If the option is 'al'
    break;
default:
    strcpy(예외 처리)// If an invalid option is provided
    break;
}
n = strlen(concatenate_output);
concatenate_output[n] = '\0';
}
}
else if (strcmp(input_buf, "LIST") == 0)
{
    DIR *dp;
    struct dirent *dirp = 0; // Pointer to structure holding information about
directory entries

```

```

char *dir;                // String to store the directory path

if (num_of_sindex == 1)   명령어만 들어온 경우
    dir = ".";            // Use the current directory
else if (num_of_sindex == 2) // If the command is followed by an
argument
{
    if (s_index[1][0] == '-') 옵션이 들어온 경우 예외처리
    {
        에러 메시지 출력후 반복문 상단을 이동
    }
    else
        dir = s_index[1]; // Use the specified directory
}
else if (2 개 이상의 인자가 들어온 경우)
    에러 출력

// Check if the directory can be opened
if (opendir(dir) == NULL) // If opening the directory fails
{
    if (접근 권한 없음)
        strcpy(concatenate_output, "cannot access : Access denied \n");
    else
        strcpy(concatenate_output, " No such directory\n"); // Print an
}
else // If the directory is successfully opened
{
    dp = opendir(dir); // Open the directory

    while ((모든 디렉토리 요소 탐색)
    {
        배열에 디렉토리 요소 저장
    }
    closedir(dp); // Close the directory stream

    // Sort the files alphabetically

```

```

        bubbleSort(files, file_count);

        // Print the list of files with detailed information (ls -al)
        print_al(files, file_count);
    }
}

else if (strcmp(input_buf, "PWD") == 0) // Command to print the current
working directory
{
    if (인자가 들어온 경우) // If additional arguments are provided
    {
        if (s_index[1][0] == '-') // If the argument is an option
            print_error(); // Print an error message for invalid options
        else
            strcat(concatenate_output, "argument is not required\n"); //
    }
    else // If no additional arguments are provided
    {
        현재 디렉토리 출력
    }
}

else if (strcmp(input_buf, "CWD") == 0)
{

    if (옵션이 들어온 경우) // If an option is passed as an argument
    {
        print_error(); // Print an error message
    }
    else if (인자가 2 개가 아닌 경우)
    {
        strcat(concatenate_output, "wrong number of arguments\n"); //
    }
    else if (chdir(s_index[1]) == -1) // If the directory change operation fails
    {

```



```

        strcat(concatenate_output, "Error: directory not found\n"); //
    }
    else
    {
        디렉토리 이름 변경
    }
}

else if (strcmp(input_buf, "MKD") == 0) // Command to make directories
{
    if (인자가 적게 들어온 경우)
        strcat(concatenate_output, "argument is required\n"); // Indicate
    else if (옵션이 들어온 경우)
    {
        print_error();
    }
    else
    {
        j = 1; // Start checking arguments from index 1
        while (받은 인자만큼 반복) // Loop through all received arguments
        {
            if (디렉토리 생성)
            {
                실패시 에러 메시지 출력
            }
            else
            {
                성공시 메시지 출력
            }
        }
    }
}
else if (strcmp(input_buf, "DELE") == 0) // Command to delete files
{
    if (인자가 없는 경우)
    {
        strcat(concatenate_output, "argument is required\n");
    }
}

```

```

else if (옵션이 들어온 경우)
{
    print_error(); // Print an error message
}
else
{
    j = 1; // Start checking arguments from index 1
    while (인자 수 만큼 반복)// Loop through all received arguments
    {
        if (파일 삭제) // Attempt to delete the file
        {
            존재하지 않는 파일 일 때 에러 출력
        }
        else
        {
            성공 메시지 출력
        }
        j++; // Move to the next argument
    }
}

else if (strcmp(input_buf, "RMD") == 0) // Command to remove directories
{
    if (인자가 없는경우)
    {
        strcat(concatenate_output, "argument is required\n");
    }
    else if (옵션이 들어온 경우)
    {
        print_error(); // Print an error message
    }
    else
    {
        j = 1; // Start checking arguments from index 1
    }
}

```

```

        while (인자 수 만큼 반복)// Loop through all received arguments
        {
            if (파일 삭제) // Attempt to delete the file
            {
존재하지 않는 파일 일 때 에러 출력
            }
            else
            {
                성공 메시지 출력
            }
            j++; // Move to the next argument
        }
    }
else if (strcmp(input_buf, "RNFR&RNTO") == 0)
{
    if (인자가 3 개가 아닌 경우)
        에러 메시지 출력
    else if (옵션이 들어온 경우 에러 출력')
        에러 메시지 출력
    else
    {
        파일 이름 변경
    }
}

else if (strcmp(input_buf, "QUIT") == 0) // Handling the QUIT command
{
    if(인자가 1 개 이상인 경우) // Check if arguments are provided
    {
        if (옵션이 들어온 경우) // If the argument is an option
        else{
            strcat(concatenate_output, "Error: argument is not required\n");
        }
    }
    else
        exit(1); // Exit the program
}

```

```

    }
    else
    {
        잘못된 명령어 입력시
    }

        사용한 문자열들 초기화
    }
}
else 부모 프로세스
{
    프로세스 데이터
    num_of_child++;
        // Increment child process counter

    print_child(child_s, num_of_child); // Print child process details

자식 프로세스 정보 출력
    }

    close(client_fd); // Close client socket
}

return 0;
}

void sh_chld(int signum)
{
    int i = 0;
    pid_t child_id;

    child_id = wait(NULL); // Wait for any child process to terminate
    printf("Client(%d) is Released\n", child_id); // Print message indicating released client

    while (모든 자식 프로세스 탐색)

```

```

    {
        if (child_s[i].pid == child_id)
        {
            배열에 저장된데이터를 하나씩 당김
        }
        i++;
    }
    num_of_child--;자식 수 감소
}

```

```

void sh_alarm(int signum)
{
    print_child(child_s, num_of_child); // Print information of child processes
}

```

```

void print_child(child_data *child, int i)
{
    자식 프로세스 배열 출력
    alarm(10); // Reset the alarm
    return;
}

```

```

void print_data(const char *print) // Function to print execution results
{
    int length = strlen(print);
    write(1, print, length); // Write the given string to stdout
}

```

```

void sh_int(int signum)
{
    exit(1); // Exit the process
}

```

```

int check_option(char *arr) // Function to check options
{

```

```
int num = 0; // Initialize num to 0; it will remain 0 if there are
no options
```

```
for (int i = 1; i < strlen(arr); i++) // Iterate over the characters of the input string,
starting from index 1
```

```
    if (arr[i] == 'a') // If the character is 'a'
        num = num + 1; // Increment num by 1
    else if (arr[i] == 'l') // If the character is 'l'
        num = num + 2; // Increment num by 2
    else
    {
        strcat(concatenate_output, "invalid option\n"); // Append an error message
        indicating an invalid option
        num = num + 100; // Set num to a value greater
        than 2 to indicate an invalid option
    }
    return num; // Return the total value of options found
}
```

```
void print_error() // Function to print error messages
```

```
{
    strcat(concatenate_output, "invalid option\n"); // Append an error message indicating
    an invalid option
}
```

```
void print_non_option(char *arr[], int n, int cd) // When there are no options
```

```
{
    int word = 0;

    for (int i = 0; i < n; i++)
    {

        if (arr[i][0] != '.') // Exclude hidden directories
        {
```

숨긴 파일을 제외한 데이터 목록 저장

```
        if (word > 5) // Change line every 5 words
        {
            5 개당 한번씩 줄바꿈
        }
    }
    strcat(concatenate_output, "\n");
}
```

```
void print_a(char *arr[], int n)
{
    int word = 0;
    for (int i = 0; i < n; i++) // Print all files
    {
        모든 데이터 목록 저장

        if (word > 5) // Change line every 5 words
        {
            데이터 5 개당 줄바꿈
        }
    }
    strcat(concatenate_output, "\n");
}
```

```
void print_al(char *filename[], int n)
{
    int word = 0;
    struct stat file_stat;
    for (int i = 0; i < n; i++)
    {
        모든 파일에 정보를 가져와 출력 버퍼에 저장
    }
}
```

```

void print_l(char *filename[], int n)
{
    int word = 0;
    struct stat file_stat;
    for (int i = 0; i < n; i++)
    {
        if (filename[i][0] != '.') // Exclude hidden files and print the rest. Other variables
serve the same purpose as print_al.
        {
            숨긴 파일을 제외한 디렉토리의 모든 파일 정보를 출력 버퍼에 저장.
        }
    }
}

```

```

void bubbleSort(char *arr[], int n) // Bubble sort used for sorting file names in a directory
{
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (strcmp(arr[j], arr[j + 1]) > 0)
            {
                char *temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

```


결과화면

클라이언트와 서버 연결

```
kw2020202037@ubuntu:~/Assignment2_3_D_2020202037_염정호$ ./cli 127.0.0.1 5000  
>  
> |
```

```

client(52987) is released
kw2020202037@ubuntu:~/Assignment2_3_D_2020202037_염정호$ ./srv 5000
=====Client info=====
client IP : 127.0.0.1

client prot : 25749
=====
Current Number of Clients: 1
  pid    PORT    TIME
  52987   25749    0
child Process ID : 52987

```

2 개의 클라이언트 접속

```

염정호$ ./srv 5000
=====Client info=====
client IP : 127.0.0.1

client prot : 15522
=====
Current Number of Clients: 1
  pid    PORT    TIME
  2986    15522    0
child Process ID : 2986
=====Client info=====
client IP : 127.0.0.1

client prot : 18082
=====
Current Number of Clients: 2
  pid    PORT    TIME
  2986    18082    4
  2988    18082    0
child Process ID : 2988

```

```

=====
Current Number of Clients: 2
  pid    PORT    TIME
  2986    18082    4
  2988    18082    0
child Process ID : 2988
Current Number of Clients: 2
  pid    PORT    TIME
  2986         14
  2988         10
Current Number of Clients: 2
  pid    PORT    TIME
  2986         24
  2988         20
Current Number of Clients: 2
  pid    PORT    TIME
  2986         34
  2988         30
Current Number of Clients: 2
  pid    PORT    TIME
  2986         44
  2988         40
Current Number of Clients: 2
  pid    PORT    TIME
  2986         54
  2988         50

```

10 초마다 child process 정보 출력

명령어 입력시 해당 프로세서의 아이디와 입력받은 명령 출력

```
NLST          [2929]
NLST          [2964]
NLST          [2966]
]

kw2020202037@ubuntu: ~/Assignment2_3_D_2020202037_연정호
YeomJungHo/  cli      cli.c    hello/    mak
efile        srv
srv.c        SSSSSSSSSSSSSS.C

> ]

kw2020202037@ubuntu: ~/Assignment2_3_D_2020202037_연정호
YeomJungHo/  cli      cli.c    hello/    mak
efile        srv
srv.c        SSSSSSSSSSSSSS.C

> ]

kw2020202037@ubuntu: ~/Assignment2_3_D_2020202037_연정호
> ls

YeomJungHo/  cli      cli.c    hello/    mak
efile        srv
srv.c        SSSSSSSSSSSSSS.C

> ]
```

ls 명령어 출력

```
> ls

Jung/      cli      cli.c
hello/     makefile  srv

srv.c      SSSSSSSSSSSSSSS.C

> ls -a
./          ./          Jung/       cl
i           cli.c      hello/
makefile    srv        srv.c
SSSSSSSSSSSSSS.C
```

```
> ls -al
drwxr-xr-x 26 kw202020237 kw202020237 4096 May 1
0 21:48 ./
drwxrwxr-x 4 kw202020237 kw202020237 4096 May 10
21:46 ./
drwxrwxr-x 2 kw202020237 kw202020237 4096 May 10
09:57 Jung/
-rwxrwxr-x 1 kw202020237 kw202020237 17760 May 1
0 21:40 cli
-rwxrwxr-x 1 kw202020237 kw202020237 17760 May 1
0 21:40
drwxrwxr-x 2 kw202020237 kw202020237 4096 May 10
07:58 hello/
drwxrwxr-x 2 kw202020237 kw202020237 4096 May 10
07:58 sh
-rwxrwxr-x 1 kw202020237 kw202020237 31504 May 1
0 21:46 srv
-rw-rw-r-- 1 kw202020237 kw202020237 36819 May 1
0 21:46 srv.c
-rw-rw-r-- 1 kw202020237 kw202020237 5678 May 09
03:56 SSSSSSSSSSSSSSS.C
```

```
> ls -l
drwxrwxr-x 2 kw202020237 kw202020237 4096 May 10
09:57 Jung/
-rwxrwxr-x 1 kw202020237 kw202020237 17760 May 1
0 21:40 cli
-rw-rw-r-- 1 kw202020237 kw202020237 7184 May 10
21:40 cli.c
drwxrwxr-x 2 kw202020237 kw202020237 4096 May 10
07:58 hello/
-rw-rw-r-- 1 kw202020237 kw202020237 74 Apr 10 1
9:49 makefile
-rwxrwxr-x 1 kw202020237 kw202020237 31504 May 1
0 21:46 srv
-rw-rw-r-- 1 kw202020237 kw202020237 36819 May 1
0 21:46 srv.c
-rw-rw-r-- 1 kw202020237 kw202020237 5678 May 09
03:56 SSSSSSSSSSSSSSS.C
```

```
> ls -a
./          ./          Jung/       cl
i           cli.c      hello/
makefile    srv        srv.c
SSSSSSSSSSSSSS.C
```

```
> ls -adfsd
invalid option
> ls -al hule
No such directory

> adfsf
unknown instruction

> 
```

출력결과 및 에러 처리

dir 에러 처리 및 결과 출력

```
> dir ald
No such directory

> dir -al
invalid option

> dir

drwxr-xr-x 26  kw202020237 kw202020237 4096 May 1
0 21:48 ../
drwxrwxr-x 4  kw202020237 kw202020237 4096 May 10
21:46 ./
drwxrwxr-x 2  kw202020237 kw202020237 4096 May 10
09:57 Jung/
-rwxrwxr-x 1  kw202020237 kw202020237 17760 May 1
0 21:40 cli
-rw-rw-r-- 1  kw202020237 kw202020237 7184 May 10
21:40 cli.c
drwxrwxr-x 2  kw202020237 kw202020237 4096 May 10
07:58 hello/
-rw-rw-r-- 1  kw202020237 kw202020237 74 Apr 10 1
9:49 makefile
-rwxrwxr-x 1  kw202020237 kw202020237 31504 May 1
0 21:46 srv
-rw-rw-r-- 1  kw202020237 kw202020237 36819 May 1
0 21:46 srv.c
-rw-rw-r-- 1  kw202020237 kw202020237 5678 May 09
03:56 sssssssssssss.c
```

pwd 현재 디렉토리 출력 및 이자 입력시 에러 출력

```
> pwd

Current working directory: /home/kw202020237/Assignment2_3_D_202020237_염정호

> pwd adf
argument is not required
```

mkdir 새로운 디렉토리 생성 , 이미 있는 디렉토리의 경우 에러 출력

```
> mkdir Yeom jung ho
MKD Yeom
MKD jung
MKD ho

> mkdir Yeom
Error: cannot create directory Yeom
```

cd 현재 디렉토리 변경

```
> cd ..
/home/kw202020237/Assignment2_3_D_202020237_염정호 is the current directory

> cd Jung
/home/kw202020237/Assignment2_3_D_202020237_염정호/Jung is the current directory

> cd ..
/home/kw202020237/Assignment2_3_D_202020237_염정호 is the current directory
```

rmdir 디렉토리 제거

```
> rmdir Jung ho Yeom
RMD Jung
RMD ho
RMD Yeom

> ls

cli                cli.c              hello/             jung/              makefile           srv
srv.c              sssssssssssss.c
```

rename 디렉토리 이름 변경

```
> rename jung YeomJungHo
RNFR: jung
RNT0: YeomJungHo

> ls

YeomJungHo/        cli                cli.c              hello/             makefile           srv
srv.c              sssssssssssss.c
```

delete 파일 삭제

```
YeomJungHo/      cli      cli.c      delete_ME      hello/      makefile
srv      srv.c      sssssssssssss.c

> delete delete_ME
DELE delete_ME

> ls

YeomJungHo/      cli      cli.c      hello/      makefile      srv
srv.c      sssssssssssss.c
```

quit 클라이언트 종료시

```
> quit

kw2020202037@ubuntu:~/Assignment2_3_D_2020202037_염정호$

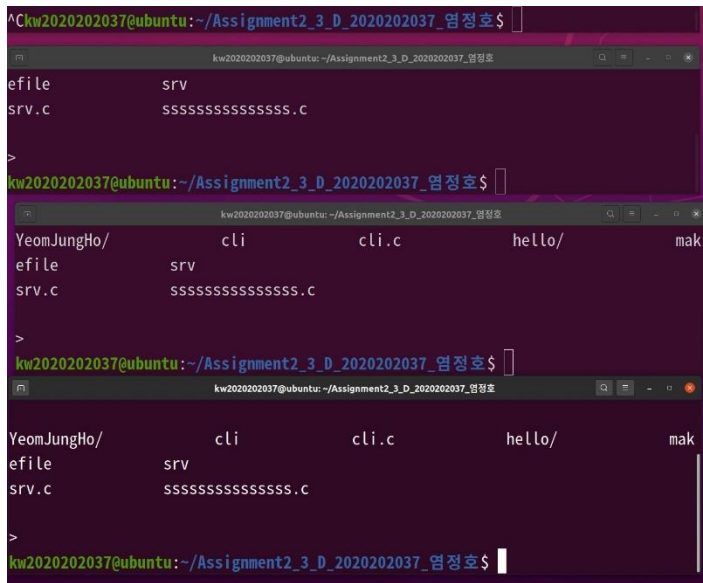
[2986]
QUIT [2986]
Client(2986) is Relased
```

ctrl c 종료

```
kw2020202037@ubuntu:~/Assignment2_3_D_2020202037_염정호$ ./cli 127.0.0.1 5000
> ^Ckw2020202037@ubuntu:~/Assignment2_3_D_2020202037_염정호$

ctrl c PROCESS ID : 3170
QUIT [3170]
Client(3170) is Relased
```

서버 종료



```
kw2020202037@ubuntu: ~/Assignment2_3_D_2020202037_염정호$  
efile      srv  
srv.c      SSSSSSSSSSSSSSS.C  
>  
kw2020202037@ubuntu: ~/Assignment2_3_D_2020202037_염정호$  
YeomJungHo/ cli cli.c hello/ mak  
efile      srv  
srv.c      SSSSSSSSSSSSSSS.C  
>  
kw2020202037@ubuntu: ~/Assignment2_3_D_2020202037_염정호$  
YeomJungHo/ cli cli.c hello/ mak  
efile      srv  
srv.c      SSSSSSSSSSSSSSS.C  
>  
kw2020202037@ubuntu: ~/Assignment2_3_D_2020202037_염정호$
```

고찰

해당 프로젝트에서는 기존에 구현한 FTP 서버를 통합하는 과제를 수행하였다. 이전 과제에서 수행한 코들을 합치는 명령어 구현 부분은 쉽게 진행 할 수 있었으나, 각 프로세스를 관리하는 부분에서 고난을 겪었다. 특히 클라이언트에서 연결을 종료하거나 서버에서 서버를 종료 시켰을 때 SIGINT 를 처리하는 부분에서 자식 프로세서의 종료 여부나 클라이언트의 종료 여부 등 신경 써야 할 부분이 많았다. 이번 프로젝트를 통해 fork 명령어의 동작 순서 제어과정에서 다양한 signal handler 를 이용해 처리하는 법을 알 수 있었다.