FTP#2

학　　과: 컴퓨터 공학과

담당 교수: 김태석 교수님

학　　번: 2013722063

성　　명: 고정원

제출 날짜: 2017 / 05 / 18

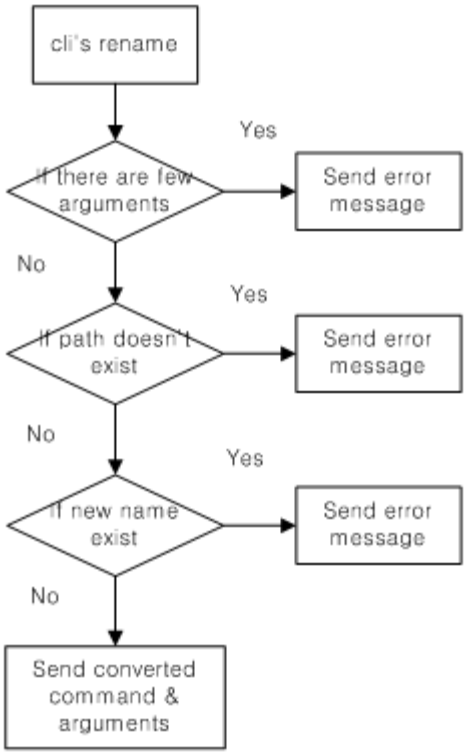
분　　반: 화요일

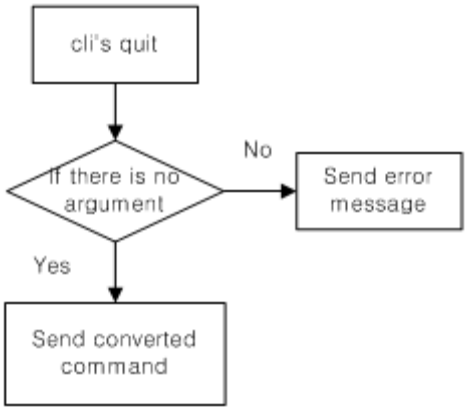
KWANGWOON UNIVERSITY

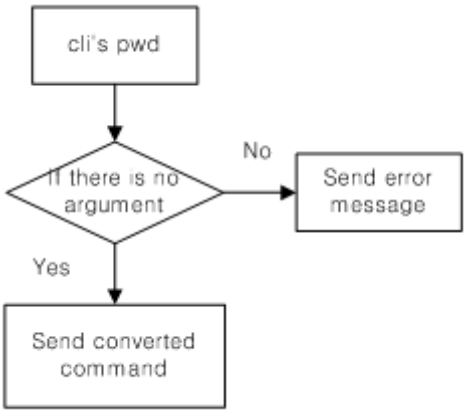
1. Introduction

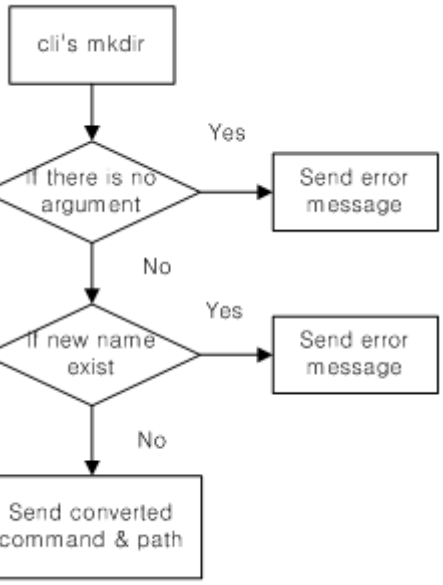
이번 Assignment에서는 앞서 진행해오던 소켓 프로그래밍, 리눅스의 각 명령어 구현, fork()함수를 통한 자식 프로세스의 생성, signal handling을 통한 signal control등을 이용하여 다중접속이 가능한 서버를 만드는 것이 목적이다.

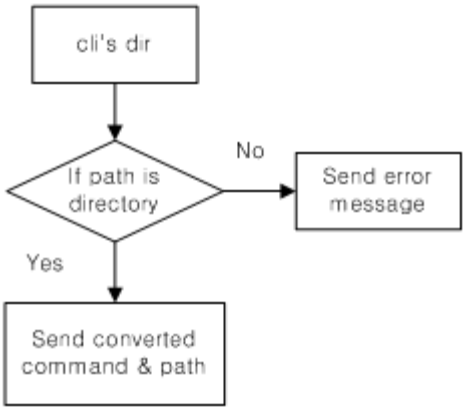
2. Flow Chart

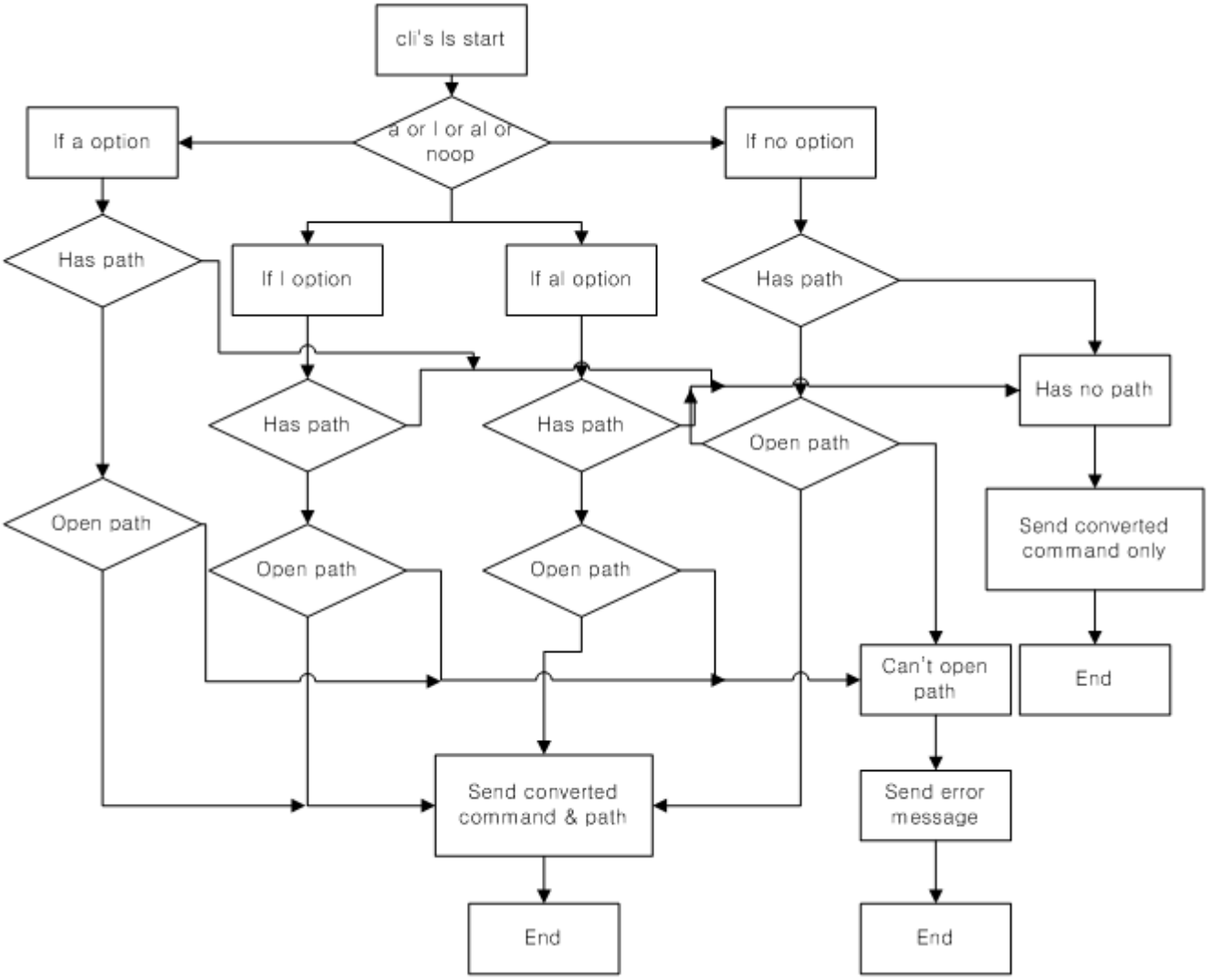


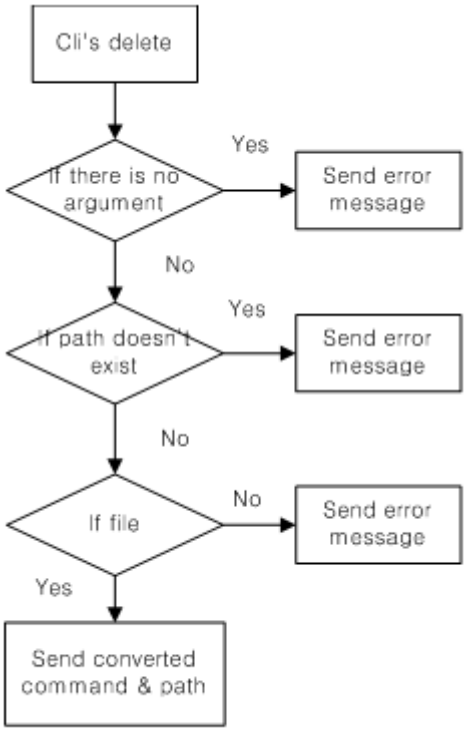


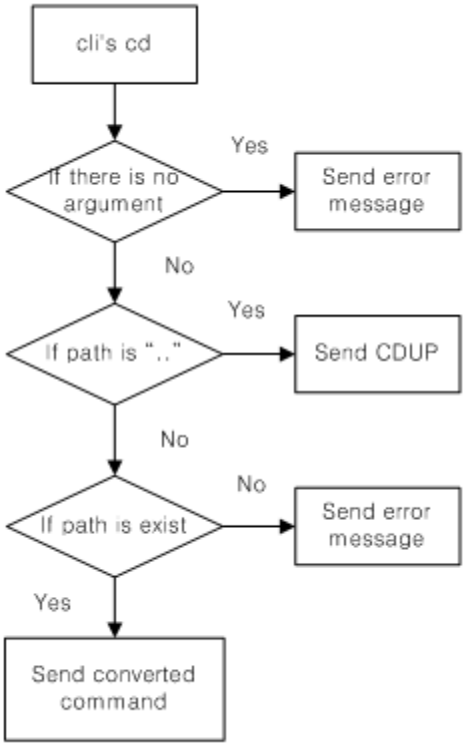


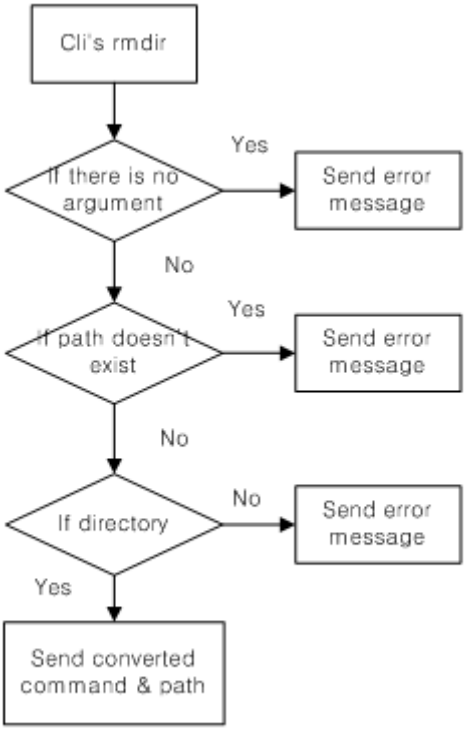

cli's mkdir
If there is no argument
Yes
Send error message
No
If new name exist
Yes
Send error message
No
Send converted command & path

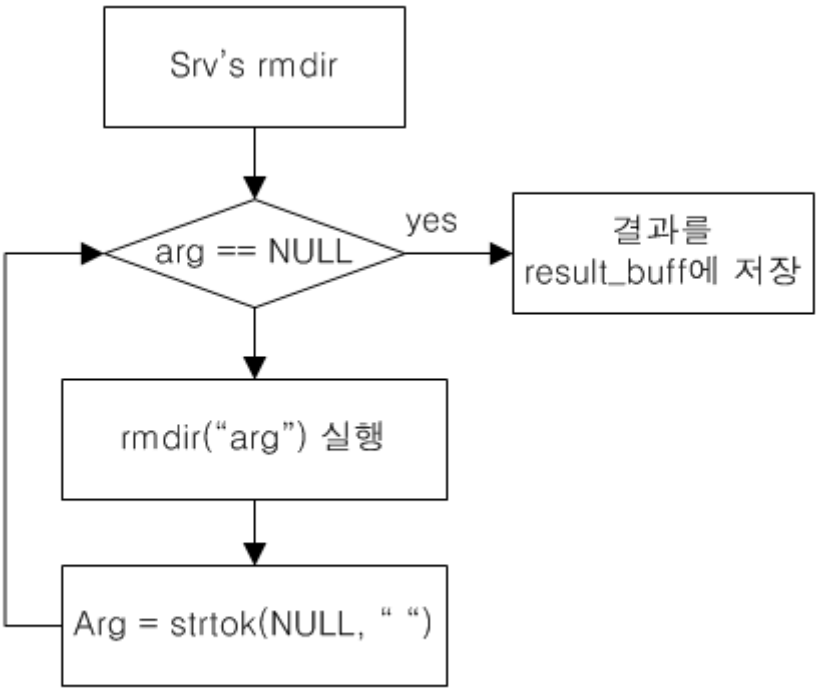

cli's dir
If path is directory
No
Send error message
Yes
Send converted command & path

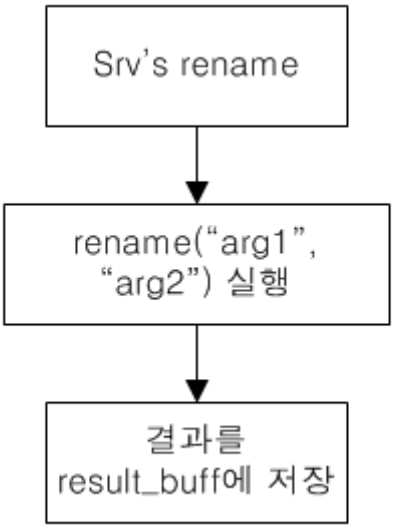

cli's ls start
a or l or al or noop
If a option
If no option
Has path
If l option
If al option
Has path
Has path
Has path
Has path
Has no path
Open path
Open path
Open path
Open path
Send converted command only
Can't open path
End
Send converted command & path
Send error message
End
End

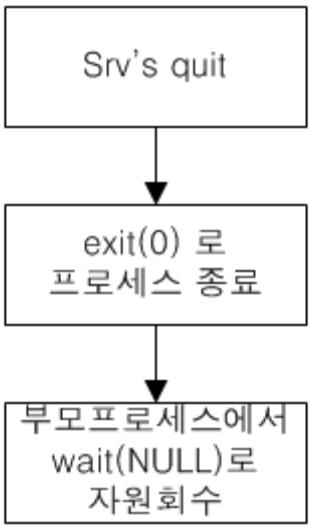

Cli's delete
If there is no argument
Yes
Send error message
No
If path doesn't exist
Yes
Send error message
No
If file
No
Send error message
Yes
Send converted command & path

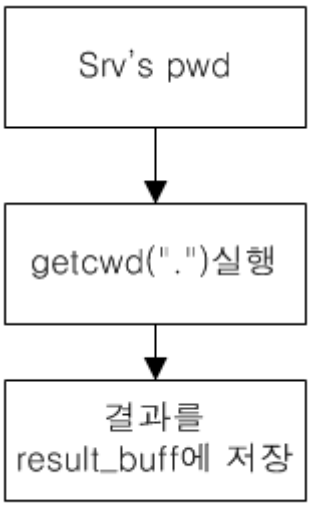

cli's cd
If there is no argument
Yes
Send error message
No
If path is ".."
Yes
Send CDUP
No
If path is exist
No
Send error message
Yes
Send converted command

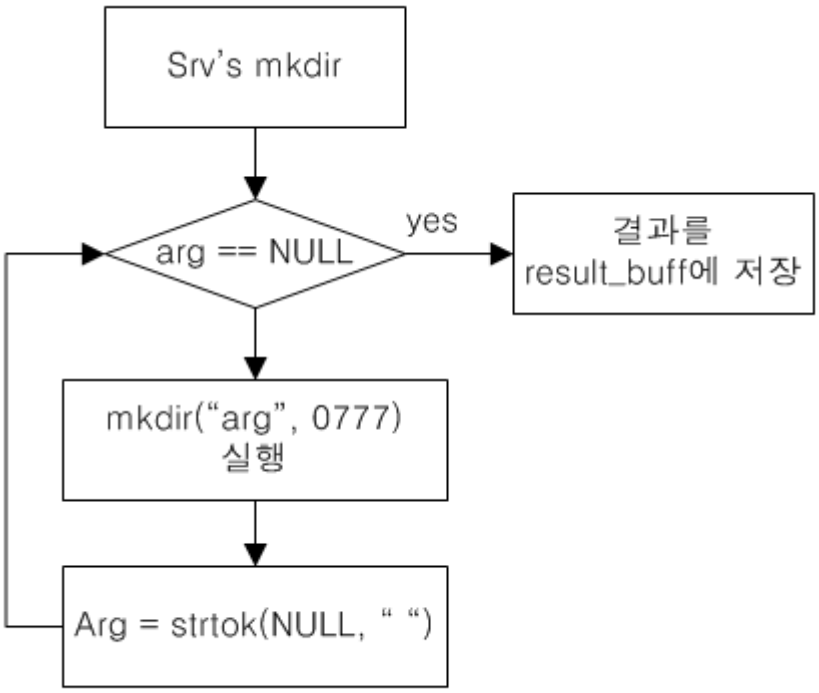

Cli's rmdir
If there is no argument
Yes
Send error message
No
If path doesn't exist
Yes
Send error message
No
If directory
No
Send error message
Yes
Send converted command & path

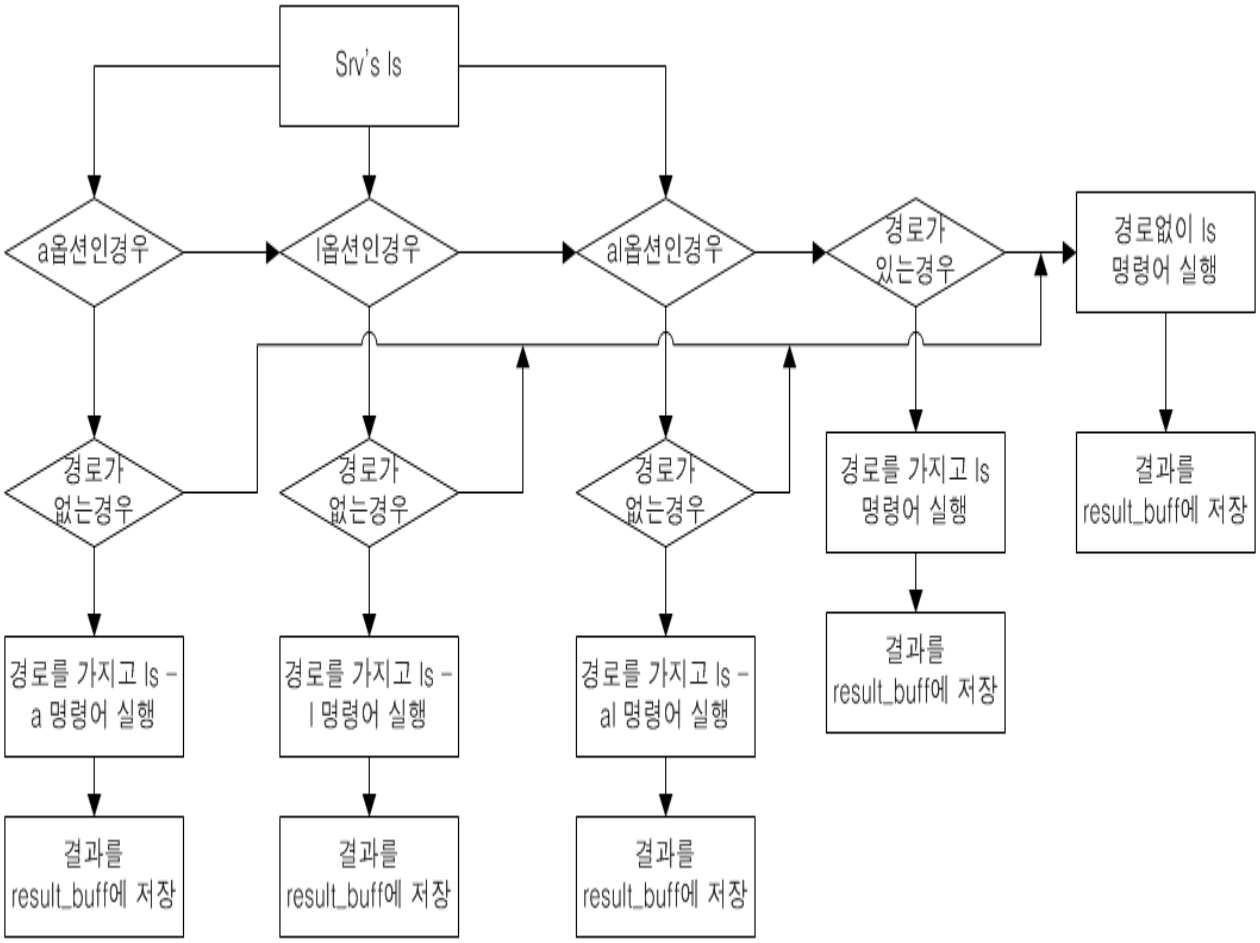

Srv's rmdir
arg == NULL
yes
결과를
result_buff에 저장
rmdir("arg") 실행
Arg = strtok(NULL, " ")

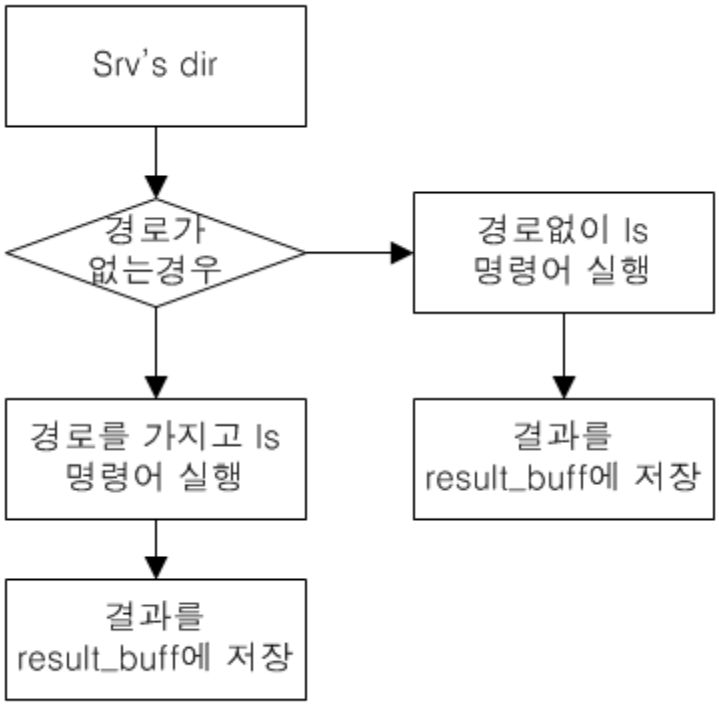

Srv's rename
rename("arg1",
"arg2") 실행
결과를
result_buff에 저장

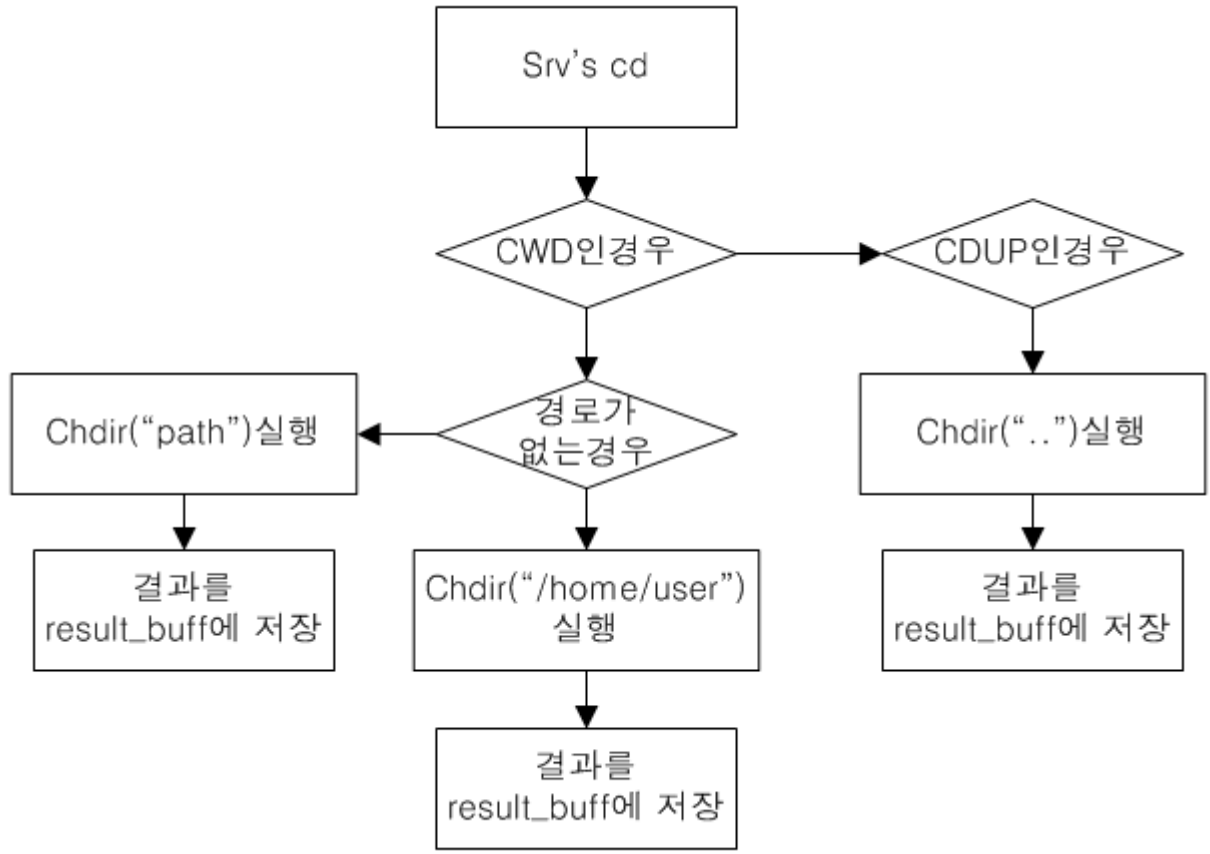

Srv's quit
exit(0) 로
프로세스 종료
부모프로세스에서
wait(NULL)로
자원회수

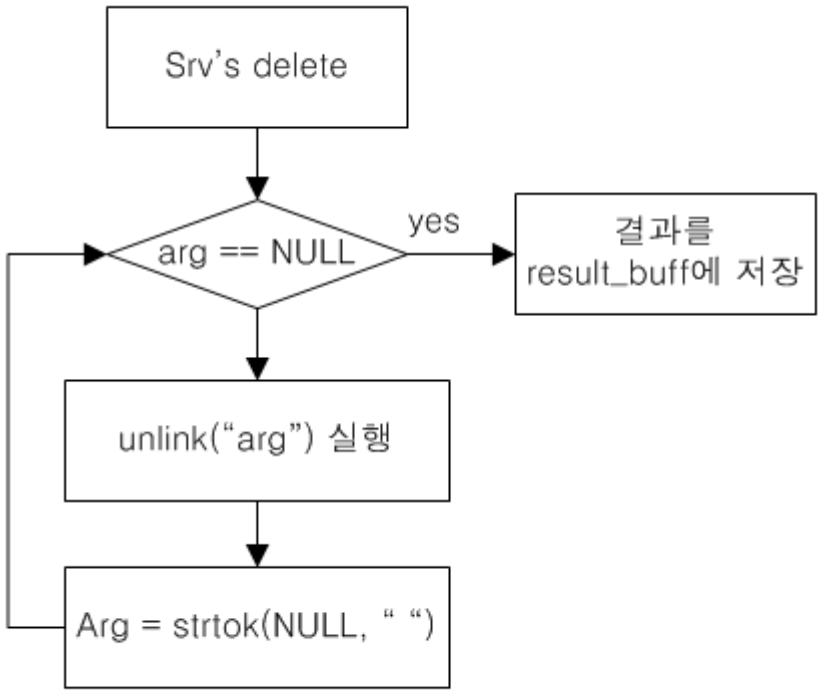

Srv's pwd
getcwd(".")실행
결과를
result_buff에 저장


Srv's mkdir
arg == NULL
yes
결과를
result_buff에 저장
mkdir("arg", 0777)
실행
Arg = strtok(NULL, " ")


Srv's ls
a옵션인경우
l옵션인경우
al옵션인경우
경로가 있는경우
경로없이 ls 명령어 실행
경로가 없는경우
경로가 없는경우
경로가 없는경우
경로를 가지고 ls 명령어 실행
결과를 result_buff에 저장
경로를 가지고 ls -a 명령어 실행
경로를 가지고 ls -l 명령어 실행
경로를 가지고 ls -al 명령어 실행
결과를 result_buff에 저장
결과를 result_buff에 저장
결과를 result_buff에 저장
결과를 result_buff에 저장


Srv's dir
경로가 없는경우
경로없이 ls 명령어 실행
경로를 가지고 ls 명령어 실행
결과를 result_buff에 저장
결과를 result_buff에 저장





3. Pseudo Code

Client Code :

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <arpa/inet.h>

#include <sys/types.h>

#include <dirent.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <sys/wait.h>

#include <signal.h>

#include <sys/stat.h>


#define MAX_BUFF 8192


int conv_cmd(char* buff, char* cmd_buff);


void process_result(char* rcv_buff);


void sig_handler(int signum);


int sockfd;


int main(int argc, char** argv)

{

        pid_t cli_pid;
```

```
signal(SIGINT, sig_handler);

signal(SIGTERM, sig_handler);

signal(SIGUSR1, sig_handler);


if(포트 번호를 입력하지 않은경우)

{

        에러 메시지를 출력하고 프로그램 종료.

}

char buff[MAX_BUFF], cmd_buff[MAX_BUFF], rcv_buff[MAX_BUFF];

int n;

struct sockaddr_in c_addr;

memset(buff, 0, sizeof(buff));


if((소켓을 만들지 못한경우)

{

        에러 메시지를 출력하고 프로그램종료

}


소켓 어드레스를 초기화


if(connect 하지 못한경우)

{

        에러 메시지를 출력하고 프로그램 종료

}


cli_pid = getpid();
```

```
sprintf(buff, "%d", cli_pid);
write(sockfd, buff, strlen(buff));

for(;;)
{
        memset(buff, 0, sizeof(buff));
        memset(cmd_buff, 0, sizeof(cmd_buff));
        memset(rcv_buff, 0, sizeof(rcv_buff));

        write(1, ">> ", strlen(">> "));
        n = read(0, buff, MAX_BUFF);
        buff[n-1] = '₩0';

        if(잘못된 커맨드를 입력한경우)
        {
                에러메시지를 출력하고 for문을 한번 건너뜀
        }

        n = strlen(cmd_buff);

        if(write를 실패한경우)
        {
                에러 메시지를 출력하고 for문을 한번 건너뜀
        }

        if(read를 성공한경우)
```

```
                {
                        읽어온 buff를 터미널에 출력
                }
                else
                        break;

        }
        close(sockfd);
        return 0;

}

int conv_cmd(char* buff, char* cmd_buff)
{
        char tempbuff[MAX_BUFF];
        char multinames[50][MAX_BUFF];
        char* option = NULL; char* arg = NULL; char* temp = NULL;
    DIR *dp;
    struct stat statt;
    int count = 0;
    int num = 0;

    if(buff가 NULL 인경우)
        return -1;

        for(num ; num<50 ; num++)
                2차원 배열인 multinames를 0으로 모두 초기화
```

```
        num = 0;
memset(tempbuff, 0, sizeof(tempbuff));
strcpy(tempbuff, buff);

option = strtok(tempbuff, " ");

if(option이 ls 인경우) {
      option = strtok(NULL, " ");
      if(option이 NULL인경우)
       {
           cmd_buff에 NLST를 저장하고 함수종료
      }
      else if(option이 -a 인경우) {
       arg = strtok(NULL, " ");
           if(path가 존재하지 않는경우) {
                cmd_buff에 NLST -a를 저장하고 함수종료
           }
           else {

                lstat(arg, &statt);
                if(해당 파일 또는 디렉토리가 존재하는경우) {
                     cmd_buff에 NLST -a "path" 를 저장하고 함수종료
                }
                else
                {
                  server에 잘못된 명령어를 보내고 -1을 return
```

```
            }
        }
    }
    else if(option 이 -l 인경우) {
        arg = strtok(NULL, " ");
        if(path가 존재하지 않는경우) {
            cmd_buff에 NLST -l 을 저장하고 함수종료
        }
        else {
            lstat(arg, &statt);
            if(해당 경로가 존재하는경우) {
                cmd_buff에 NLST -l "path"를 저장하고 함수종료
            }
            else
            {
              잘못된 명령어를 server로 보내고 -1을 반환
            }
        }
    }
    else if(option이 -al인경우) {
        arg = strtok(NULL, " ");
        if(경로가 없는경우) {
            cmd_buff에 NLST -al을 저장하고 함수종료
        }
        else {

            lstat(arg, &statt);
```

```
            if(경로가 존재하는경우) {
                cmd_buff에 NLST -al "path"를 저장하고 함수종료
            }
            else
            {
              서버에 잘못된 명령어를 보내고 -1반환
            }
        }
    }
    else if(option 이 -la 인경우) {          //if al option
        arg = strtok(NULL, " ");
        if(경로가 없는경우) {
            cmd_buff에 NLST -al을 저장하고 함수종료
        }
        else {

            lstat(temp, &statt);
            if(해당 경로가 존재하는경우) {
                cmd_buff에 NLST -al "path"를 저장하고 함수종료
            }
            else
            {
              서버에 잘못된 명령어를 보내고 -1반환
            }
        }
    }
    else {
```

```
            lstat(option, &statt);
                if(해당 경로가 존재하는경우) {
                    cmd_buff에 NLST "path"를 저장하고 함수종료
                }
                else
                {
                  잘못된 명령어를 서버에 보내고 -1반환
                }
        }
}
else if(option이 dir인경우) {

        arg = strtok(NULL, " ");
        if(path가 없는경우) {
            cmd_buff에 LIST를 저장하고 함수종료
        }
        else {
            lstat(arg, &statt);
            if(해당 경로가 디렉토리인경우) {
                cmd_buff에 LIST "paht"를 저장하고 함수종료
            }
            else
            {
                잘못된 명령어를 서버에 전송하고 -1반환
            }
        }
}
```

```
else if(option이 pwd인경우) {
        option = strtok(NULL, " ");
       if(추가 argument가 존재하는경우)
       {
        서버에 잘못된명령어를 전송하고 -1반환
       }
       else {
            cmd_buff에 PWD를 저장하고 함수종료
       }
}
else if(option이 cd 인경우) {                //if cd command
        option = strtok(NULL, " ");
        arg = strtok(NULL, " ");
       if(경로가 2개 들어온경우)
       {
        잘못된 명령어를 서버에 전송하고 -1반환
       }
       else if(경로가 존재하지 않는경우)
       {
        cmd_buff에 CWD를 저장하고 함수종료
       }
       else {
            if(해당 경로를 열지 못한경우)
            {
                  서버에 잘못된 명령어를 전송하고 -1반환
            }
            else {
```

```
                lstat(option, &statt);
                if(경로로 ".."이 들어온경우)
                {
                    cmd_buff에 CDUP을 저장하고 함수종료
                }
                else {
                    if(해당 경로가 존재하는경우) {
                        cmd_buff에 CWD "path"를 저장하고 함수종료
                    }
                    else
                    {
                        서버에 잘못된 명령어를 전송하고 -1반환
                    }
                }
            }
        }
    }
    else if(option 이 mkdir인경우) {
        option = strtok(NULL, " ");
        if(옵션이 존재하지 않는경우)
        {
         잘못된 명령어를 서버에 전송하고 -1반환
        }
        else {
        cmd_buff에 MKD 를 저장하고
         multinames배열에 모든 옵션을 저장
         cmd_buff에 multinames의 모든 데이터를 붙이고 함수종료
```

```
        }
    }
    else if(option이 delete인경우) {
        option = strtok(NULL, " ");
        if(옵션이 없는경우)
        {
         서버에 잘못된 명령어를 전송하고 -1반환
        }
        else {
         cmd_buff에 DELE 를 저장하고
         multinames배열에 모든 옵션을 저장
            for(count ; count < num; count++) {
                lstat(multinames[count], &statt);
                if(해당 파일이 디렉토리인경우)
                {
                  잘못된 명령어를 서버에 전송하고 -1반환
                }
                else {
                    cmd_buff에 multinames의 데이터를 덧붙임


                }
            }
            함수종료
        }
    }
    else if(option이 rmdir인경우) {
        option = strtok(NULL, " ");
```

```
        if(옵션이 존재하지 않는경우)
        {
         서버에 에러메시지를 전송하고 -1반환
        }
        else {
         cmd_buff에 RMD 를 저장하고
         multinames배열에 모든 옵션을 저장
         cmd_buff에 multinames의 데이터를 덧붙임
         함수종료
          }
}
else if(option이 rename인경우) {
         option = strtok(NULL, " ");
         arg = strtok(NULL, " ");
         temp = strtok(NULL, " ");
        if(old name과 new name이 존재하지 않는경우)
        {
         서버에 에러 메시지를 전송하고 -1반환
        }
        else {
             lstat(option, &statt);
             if(oldename이 존재하는경우) {
                  if(new name이 존재하지 않는경우) {
                       cmd_buff에 RNFR "old name" "new name" 을 복사하고 함수종료
                  }
                  else {
                    서버에 에러메시지를 전송하고 -1반환
```

```
                    }
                }
                else
                {
                        서버에 에러메시지를 전송하고 -1반환
                }
            }
        }
        else if(option 이 quit인경우) {
             option = strtok(NULL, " ");
            if(option 이 존재하지 않는경우) {
                서버에 "QUIT"전송하고 함수종료
            }
            else {
             서버에 에러메시지를 전송하고 -1반환
            }
        }
        else {
             서버에 에러메시지를 전송하고 -1반환
        }
}

void sig_handler(int signum)
{
             if(signum == SIGINT)
             {
                        서버에 "QUIT"을 전송하고 소켓을 닫고 프로세스 종료
```

```
        }
        else if(signum == SIGTERM)
        {
                소켓을 닫고 프로세스 종료
        }
        else if(signum == SIGUSR1)
        {
                소켓을 닫고 프로세스 종료
        }
}
```

Server Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <dirent.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/wait.h>
#include <signal.h>
#include <sys/stat.h>
#include <time.h>
#include <pwd.h>
#include <grp.h>
```

```
void strlwr(char* str);
void changestr(char* str1, char* str2);
void showlist(char* path, char* result_buff);
void showlist_a(char* path, char* result_buff);
void showlist_l(char* path, char* result_buff);
void showlist_al(char* path, char* result_buff);
void dir_path(char* path, char* result_buff);

int cmd_process(char* buf, char* result_buff);

int client_info(struct sockaddr_in* c_addr);

void sh_alrm(int);
void sh_term(int);
void sh_chld(int);
void sh_int(int);
void sh_usr1(int);

typedef struct Node
{
        pid_t process_ID;
        pid_t cli_pid;
        int port;
        int time;
        int process_count;
        struct Node* p_next;
```

```
}Node;

void insert_node(struct sockaddr_in* c_addr, pid_t cur_pid, pid_t cli_pid);
void delete_node(pid_t pid);
void print_node();

Node* p_head;
int server_fd, client_fd;
pid_t parent_pid;

#define MAX_BUFF 8192

int main(int argc, char** argv)
{
        p_head = NULL;

        char buff[MAX_BUFF], result_buff[MAX_BUFF];
        int n;
        struct sockaddr_in server_addr, client_addr;

        int len;
        int port;

        if(소켓을 만들지 못한경우)
        {
```

```
            터미널에 에러메시지를 출력하고 프로그램 종료
}

memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = PF_INET;
server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
server_addr.sin_port = htons(atoi(argv[1]));

if(bind하지 못한경우)
{
            터미널에 에러 메시지를 출력하고 프로그램 종료
}

listen(server_fd, 5);

while(1)
{
            signal(SIGINT, sh_int);
            signal(SIGALRM, sh_alrm);
            signal(SIGCHLD, sh_chld);
            signal(SIGTERM, sh_term);
            signal(SIGUSR1, sh_usr1);

            pid_t pid;
            int temp;
            len = sizeof(client_addr);
```

```
if(accept하지 못한경우)
{
        터미널에 에러메시지를 출력하고 프로그램 종료
}

memset(buff, 0, sizeof(buff));
read(client_fd, buff, MAX_BUFF);

parent_pid = getpid();

if((pid = fork()) > 0)                    //부모 프로세스
{
        클라이언트의 인포메이션 출력
        노드를 새로 만듬
        리스트 출력
}
else if(pid == 0)                         //자식 프로세스
{
        while(1)
        {
                클라이언트로부터 데이터를 읽어옴

                if(buff가 QUIT인경우)
                {
                        프로세스 종료
                }
                명령어에따라 result_buff에 결과값을저장하고
```

```
                                        클라이언트에 전송
                                    }
                            }
                            close(client_fd);
                    }
                    return 0;
}

int cmd_process(char* buf, char* result_buff)
{
        char temp[MAX_BUFF];
        memset(temp, 0, sizeof(temp));
        strcpy(temp, buf);
        char printpid[10];
        sprintf(printpid, " [%d]₩n", getpid());
        char tempstring[100];
        struct passwd* psd;

        char* arg1; char* arg2; char* arg3; char* arg4;
        arg1 = strtok(buf, " ");

        if(arg1이 NLST인경우) {
                arg2 = strtok(NULL, " ");
                if(path 가 존재하지 않는경우) {
                        해당 프로세스의 pid를 출력하고
                        showlist함수 실행
                }
```

```
else if(옵션이 -a인경우) {
    arg3 = strtok(NULL, " ");
    if(path가 존재하지 않는경우) {
        해당 프로세스의 pid를 출력하고
        showlist_a함수에 인자로"."을넘겨줌
    }
    else {
        해당 프로세스의 pid를 출력하고
        showlist_a함수에 인자로 경로를넘겨줌
    }
}
else if(option이 -l 인경우) {
    arg3 = strtok(NULL, " ");
    if(경로가 없는경우) {
        해당 프로세스의 pid를 출력하고
        showlist_l함수에 인자로 "."을넘겨줌
    }
    else {
        해당 프로세스의 pid를 출력하고
        showlist_l함수에 인자로 경로를넘겨줌
    }
}
else if(option이 al인경우) {
    arg3 = strtok(NULL, " ");
    if(경로가 없는경우) {
        해당 프로세스의 pid를 출력하고
        showlist_al함수에 인자로 "."을넘겨줌
```

```
            }
            else {
                해당 프로세스의 pid를 출력하고
                showlist_al함수에 인자로 경로를넘겨줌
            }
        }
        else {
            해당 프로세스의 pid를 출력하고
            showlist함수에 인자로 경로를넘겨줌
        }
    }
    else if(arg1이 LIST인경우) {
        arg3 = strtok(NULL, " ");
        if(경로가 없는경우) {
                해당 프로세스의 pid를 출력하고
                dir_path함수에 인자로 "."을 넘겨줌
            }
        else
        {
        해당 프로세스의 pid를 출력하고
                dir_path함수에 인자로 경로를 넘겨줌
       }
    }
    else if(arg1이 PWD인경우) {
        result_buff에 현재 pwd를 저장하고 함수종료
    }
    else if(arg1이 cd인경우) {
```

```
        arg2 = strtok(NULL, " ");

        if(경로가 없는경우)
        {

         CWD를 home디렉토리로 변경
        }
         현재 프로세스의 pid출력
        들어온 경로로 CWD를 변경
}
else if(arg1이 CDUP인경우) {
         현재 프로세스의 pid출력
         CWD를 부모 디렉토리로 변경
}
else if(arg1이 MKD인경우) {
         현재 프로세스의 pid출력
        계속 토큰화 하여 디렉토리 생성
}
else if(arg1이 DELE인경우) {
         현재 프로세스의 pid출력
        계속 토큰화 하여 파일 삭제
}
else if(arg1이 RMD인경우) {
         현재 프로세스의 pid출력
        계속 토큰화 하여 디렉토리 삭제
}
```

```
        else if(arg1이 RNFR인경우)
        {

            oldname을 newname으로 변경
        }
        else
        {
            에러메시지를 result_buff에 저장
             함수종료
        }
}

void showlist(char* path, char* result_buff)
{

        DIR *dp;
        struct dirent *dirp;

        int count = 0;
        int count2 = 0;
        int check = 0;
        char sort[100][50];

        for(count ; count < 100 ; count ++)
            memset(sort[count], 0, sizeof(sort[count]));
        count = 0;
```

```
struct stat statt;

if(strcmp(path, ".") == 0) {

    while(dirp = readdir(dp)) {
        디렉토리내 모든 파일을 sort배열에 저장하면서 디렉토리인경우 이름의 마지막에
            /를 같이 저장
        }
    }
}
else {

    lstat(path, &statt);

    if(경로가 파일인경우) {
     파일 이름을 result_buff에 저장하고 함수종료
    }

    dp = opendir(path);

    while(dirp = readdir(dp)) {
        디렉토리내 모든 파일을 sort배열에 저장하면서 디렉토리인경우 이름의 마지막에
            /를 같이 저장
    }
}
```

```
        for(count = 0 ; count < check - 1 ; count++ ) {
            for(count2 = 0 ; count2 < check - count - 1 ; count2 ++) {
             sort배열을 버블정렬을 이용해 sorting
            }
        }

        count = 0;
        count2 = 0;

         for(; count2 < check; ) {
            sort배열을 result_buff에 저장
        }


}

void showlist_a(char* path, char* result_buff)
{
        DIR *dp;
        struct dirent *dirp;

        int count = 0;
        int count2 = 0;
        int check = 0;
```

```
char sort[100][50];


for(count ; count < 100 ; count ++)
    memset(sort[count], 0, sizeof(sort[count]));
count = 0;


struct stat statt;


if(strcmp(path, ".") == 0) {



    while(dirp = readdir(dp)) {
        디렉토리내 모든 파일을 sort배열에 저장하면서 디렉토리인경우 이름의 마지막에
            /를 같이 저장
        }
    }
}
else {


    lstat(path, &statt);


    if(경로가 파일인경우) {
     파일 이름을 result_buff에 저장하고 함수종료
    }


    dp = opendir(path);
```

```
        while(dirp = readdir(dp)) {
            디렉토리내 모든 파일을 sort배열에 저장하면서 디렉토리인경우 이름의 마지막에
                /를 같이 저장
        }
    }



    for(count = 0 ; count < check - 1 ; count++ ) {
        for(count2 = 0 ; count2 < check - count - 1 ; count2 ++) {
         sort배열을 버블정렬을 이용해 sorting
        }
    }

    count = 0;
    count2 = 0;

     for(; count2 < check; ) {
        sort배열을 result_buff에 저장
    }


}


void showlist_l(char* path, char* result_buff)
```

```
{

    DIR *dp;
    struct dirent *dirp;

    int count = 0;
    int count2 = 0;
    int check = 0;
    char sort[100][50];
    char cpath[100];
    char* tok;
    char data[50];
    struct passwd* pd;
    struct group* grp;
    struct tm* ttime;
    char permission[11];
    char time[50];
    char buf[100];

    memset(cpath, 0, sizeof(cpath));

    for(count ; count < 100 ; count ++)
        memset(sort[count], 0, sizeof(sort[count]));
    count = 0;

    struct stat statt;
```

```
    if(현재 경로인 경우) {
        dp = opendir(".");

        while(dirp = readdir(dp)) {
            디렉토리내 모든 파일을 sort배열에 저장하면서 디렉토리인경우 이름의 마지막에
                /를 같이 저장
        }
    }
    else {

        lstat(path, &statt);

        if(경로가 파일인경우) {

            data에 파일 이름을 저장
            permission배열을 초기화해줌
            해당 파일의 information들을 result_buff에 저장
            함수종료
}

        dp = opendir(path);

        while(dirp = readdir(dp)) {
            디렉토리내 모든 파일을 sort배열에 저장하면서 디렉토리인경우 이름의 마지막에
                /를 같이 저장
        }
    }
```

```
for(count = 0 ; count < check - 1 ; count++ ) {
    for(count2 = 0 ; count2 < check - count - 1 ; count2 ++) {

        sort배열을 버블정렬을 이용하여  sorting

        }
    }
}

count = 0;
count2 = 0;


if(strcmp(path, ".") == 0) {
    for(count = 0 ; count < check ; count++) {
        data에 파일 이름을 저장
        permission배열을 초기화해줌
        해당 파일의 information들을 result_buff에 저장
        함수종료
    }
}
else {
    for(count = 0 ; count < check ; count++) {
        data에 파일 이름을 저장
```

```
                permission배열을 초기화해줌
                해당 파일의 information들을 result_buff에 저장
                함수종료
            }
        }

        if(strcmp(result_buff, "") == 0)
            strcpy(result_buff, "₩n");

        closedir(dp);

}


void showlist_al(char* path, char* result_buff)
{
            DIR *dp;
        struct dirent *dirp;

        int count = 0;
        int count2 = 0;
        int check = 0;
        char sort[100][50];
        char cpath[100];
        char* tok;
        char data[50];
        struct passwd* pd;
```

```
struct group* grp;
struct tm* ttime;
char permission[11];
char time[50];
char buf[100];

memset(cpath, 0, sizeof(cpath));

for(count ; count < 100 ; count ++)
    memset(sort[count], 0, sizeof(sort[count]));
count = 0;

struct stat statt;

if(현재 경로인 경우) {
    dp = opendir(".");

    while(dirp = readdir(dp)) {
        디렉토리내 모든 파일을 sort배열에 저장하면서 디렉토리인경우 이름의 마지막에
            /를 같이 저장
    }
}
else {

    lstat(path, &statt);

    if(경로가 파일인경우) {
```

```
                data에 파일 이름을 저장
                permission배열을 초기화해줌
                해당 파일의 information들을 result_buff에 저장
                함수종료
}

        dp = opendir(path);

        while(dirp = readdir(dp)) {
            디렉토리내 모든 파일을 sort배열에 저장하면서 디렉토리인경우 이름의 마지막에
                /를 같이 저장
        }
    }

    for(count = 0 ; count < check - 1 ; count++ ) {
        for(count2 = 0 ; count2 < check - count - 1 ; count2 ++) {

            sort배열을 버블정렬을 이용하여  sorting

            }
        }
    }

    count = 0;
```

```
        count2 = 0;



        if(strcmp(path, ".") == 0) {
            for(count = 0 ; count < check ; count++) {
                data에 파일 이름을 저장
                permission배열을 초기화해줌
                해당 파일의 information들을 result_buff에 저장
                함수종료
            }
        }
        else {
            for(count = 0 ; count < check ; count++) {
                data에 파일 이름을 저장
                permission배열을 초기화해줌
                해당 파일의 information들을 result_buff에 저장
                함수종료
            }
        }

        if(strcmp(result_buff, "") == 0)
            strcpy(result_buff, "₩n");

        closedir(dp);

}
```

```
void dir_path(char* path, char* result_buff)
{
	DIR *dp;
	struct dirent *dirp;

	int count = 0;
	int count2 = 0;
	int check = 0;
	char sort[100][50];
	char cpath[100];
	char* tok;
	char data[50];
	struct passwd* pd;
	struct group* grp;
	struct tm* ttime;
	char permission[11];
	char time[50];
	char buf[100];

	memset(cpath, 0, sizeof(cpath));

	for(count ; count < 100 ; count ++)
		memset(sort[count], 0, sizeof(sort[count]));
	count = 0;

	struct stat statt;
```

```
        if(현재 경로인 경우) {
            dp = opendir(".");

            while(dirp = readdir(dp)) {
                디렉토리내 모든 파일을 sort배열에 저장하면서 디렉토리인경우 이름의 마지막에
                    /를 같이 저장
            }
        }
        else {

            lstat(path, &statt);

            if(경로가 파일인경우) {

                data에 파일 이름을 저장
                permission배열을 초기화해줌
                해당 파일의 information들을 result_buff에 저장
                함수종료
    }

            dp = opendir(path);

            while(dirp = readdir(dp)) {
                디렉토리내 모든 파일을 sort배열에 저장하면서 디렉토리인경우 이름의 마지막에
                    /를 같이 저장
            }
```

```
}



for(count = 0 ; count < check - 1 ; count++ ) {
    for(count2 = 0 ; count2 < check - count - 1 ; count2 ++) {

        sort배열을 버블정렬을 이용하여  sorting


        }
    }
}

count = 0;
count2 = 0;



if(strcmp(path, ".") == 0) {
    for(count = 0 ; count < check ; count++) {
        data에 파일 이름을 저장
        permission배열을 초기화해줌
        해당 파일의 information들을 result_buff에 저장
        함수종료
    }
}
else {
    for(count = 0 ; count < check ; count++) {
```

```
                data에 파일 이름을 저장
                permission배열을 초기화해줌
                해당 파일의 information들을 result_buff에 저장
                함수종료
            }
        }

        if(strcmp(result_buff, "") == 0)
            strcpy(result_buff, "₩n");

        closedir(dp);

}

void changestr(char* str1, char* str2)
{
    str1과 str2의 데이터를 교체
}

void strlwr(char* str)
{
    배열내의 모든 대문자를 소문자로 교체
}

int client_info(struct sockaddr_in* c_addr)
```

```
{

        클라이언트의 information을 출력

}

void sh_int(int signum)
{
        Node* p_temp = p_head;

    write(STDOUT_FILENO, "₩n", 1);


    if(getpid() != parent_pid)
    {
        kill(parent_pid, SIGUSR1);
        return;
    }


        if(p_head == NULL)
        exit(0);

        while(1)
        {
                모든 자식 프로세스를 죽임
        }
```

```
        kill(getpid(), SIGTERM);
}

void sh_alrm(int signum)
{
        print_node();
        return;
}

void sh_term(int signum)
{

    if(자식 프로세스인경우)
    {
        pid를 출력
    }
        //close client
    else
        write(STDOUT_FILENO, "Server Terminated!₩n", sizeof("Server Terminated!₩n"));

        프로세스 종료
}

void sh_chld(int signum)
{
```

```
        pid_t child_id;
        child_id = waitpid(-1, NULL, WNOHANG);

    if(child_id != 0)
        delete_node(child_id);
}

void sh_usr1(int signum)
{
    Node* p_temp = p_head;

    write(STDOUT_FILENO, "₩n", 1);

    if(p_head == NULL)
        exit(0);

    while(1)
    {
        모든 자식프로세스를 죽임
    }

    //kill parent process
    kill(getpid(), SIGTERM);
}
```

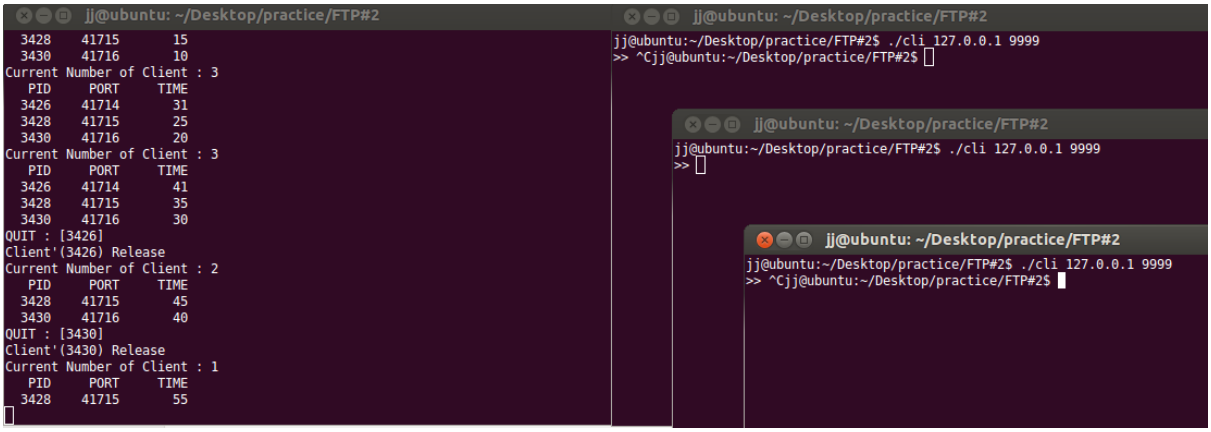
```
void insert_node(struct sockaddr_in* c_addr, pid_t cur_pid, pid_t cli_pid)
{

        struct timeval curtime;
        gettimeofday(&curtime, NULL);

        Node* p_temp = p_head;

        if(헤드가 NULL인경우)
        {
                새로운 노드를 만들어 head로 설정
        }
        else
        {
                가장 마지막 노드의 next노드를 새로운 노드로 설정
        }
}

void delete_node(pid_t pid)
{

        Node* p_temp = p_head;
        Node* p_prev = p_temp;

        while(p_temp->p_next != NULL)
        {
```

```
            pid에 해당하는 노드를 검색
        }

        if(해당 노드가 head인경우)
        {

            if(p_head->p_next == NULL)
            {
                free(p_head);
                p_head = NULL;
            }
            else
            {
                p_head = p_temp->p_next;
                p_head->process_count = p_temp->process_count;
                free(p_temp);
                p_head->process_count -= 1;
            }
            return;
        }

        p_prev->p_next = p_temp->p_next;
        free(p_temp);
        p_head->process_count -= 1;
        return;
}
```

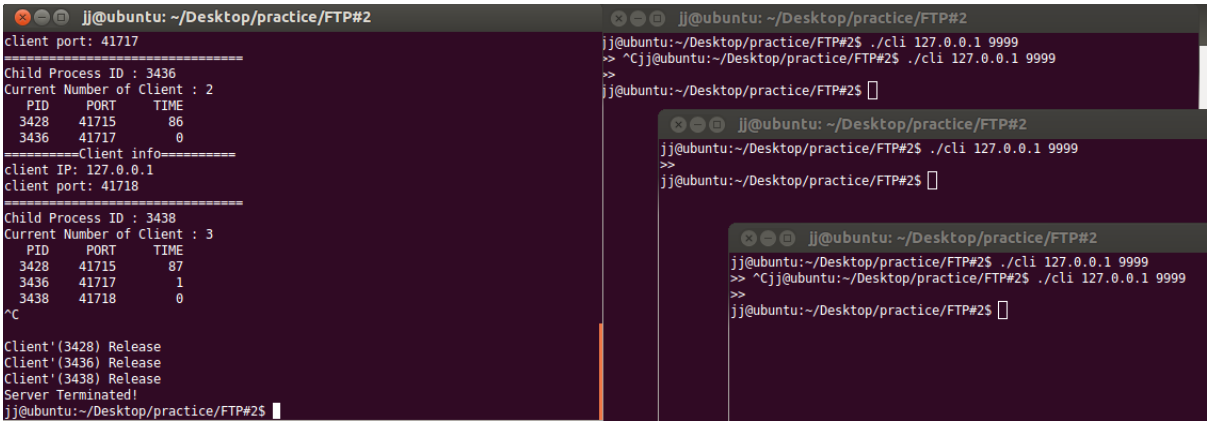
```
void print_node()
{

        if(p_head == NULL)
        {
                알람을 종료하고 함수를 종료
        }

        alarm(10);

        Node* p_temp = p_head;
        struct timeval curtime;
        gettimeofday(&curtime, NULL);
        char string_buff[200];

        if(p_temp->p_next == NULL)
        {
                노드의 정보들을 출력
        }
        else
        {
                모든 노드를 돌면서 정보를 출력
        }
}
```

4. Result Screen

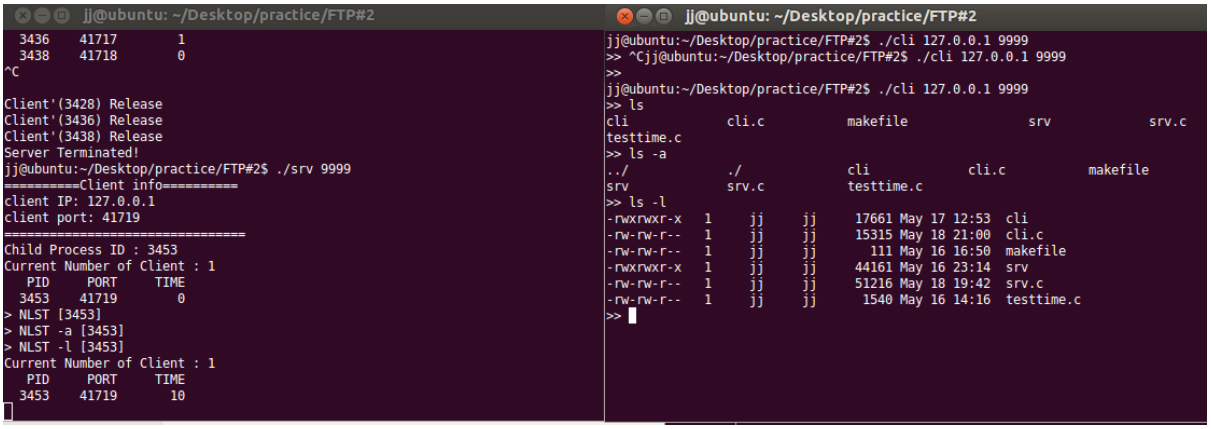
클라이언트의 다중접속을 보여주는 모습



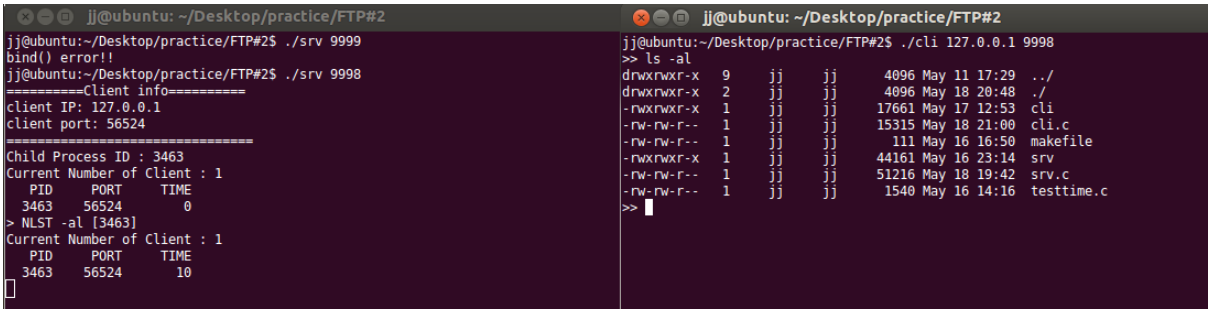
클라이언트의 접속종료를 보여주는 모습



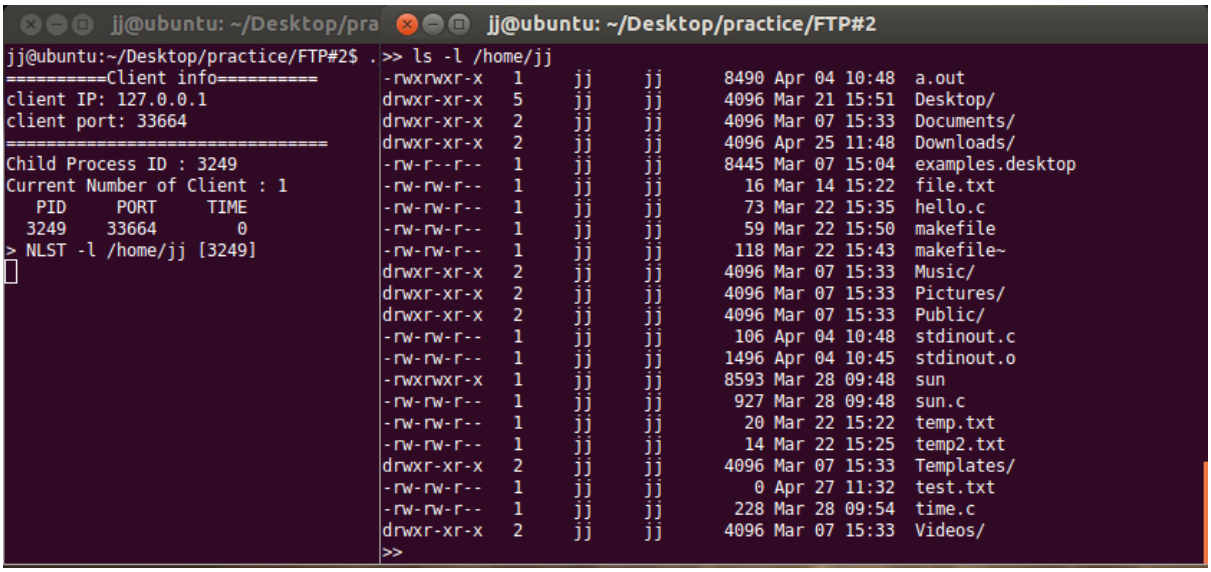
서버에서 ctrl + c로 모든 클라이언트를 종료하고 자신도 종료되는 모습



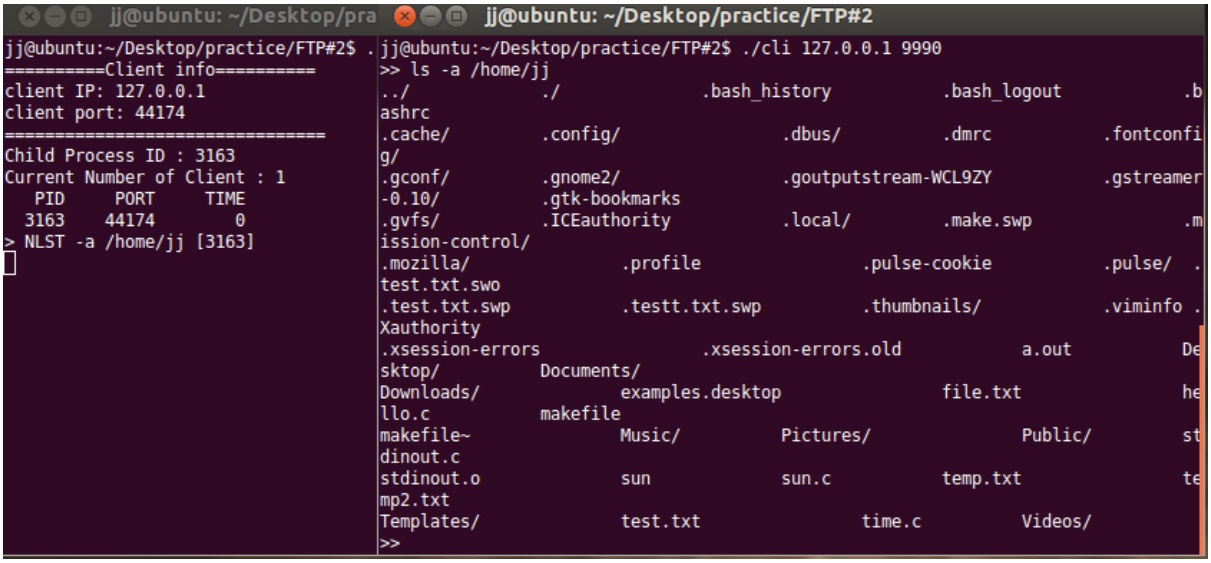
ls, ls -a, ls -l 명령어를 실행하는 모습



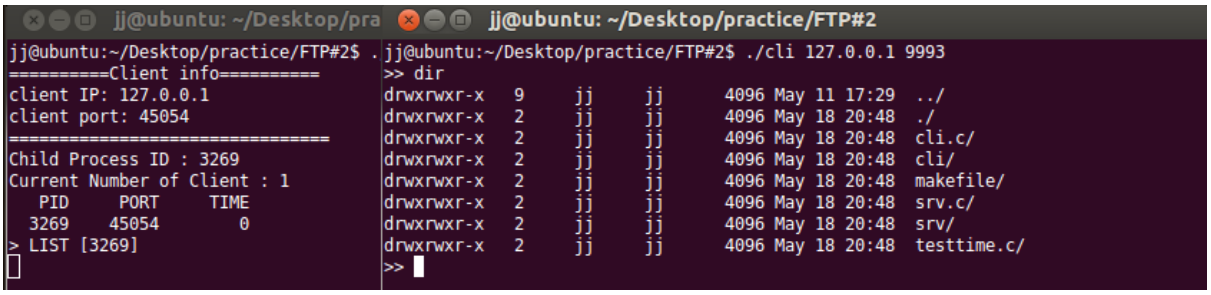
ls -al 명령어를 실행하는 모습



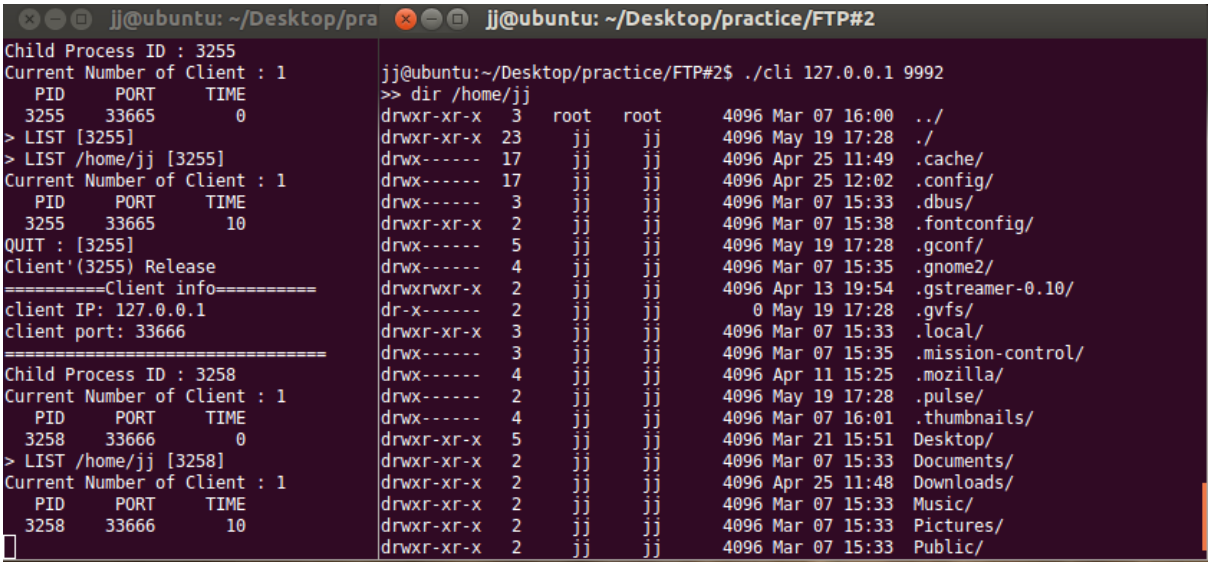
경로를 주고 ls -l 명령어를 사용하는 모습



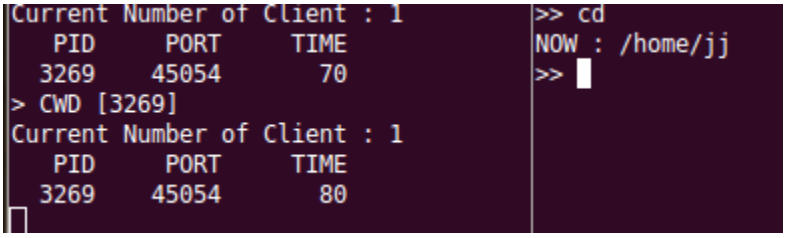
경로를 주고 ls -a 명령어를 사용하는 모습



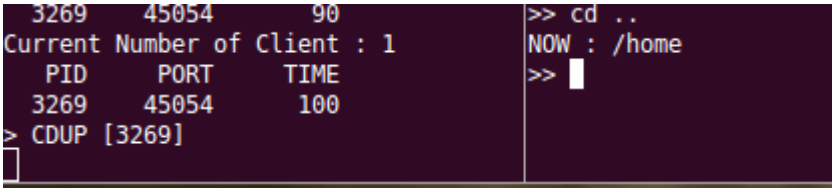
dir 명령어를 사용하는 모습



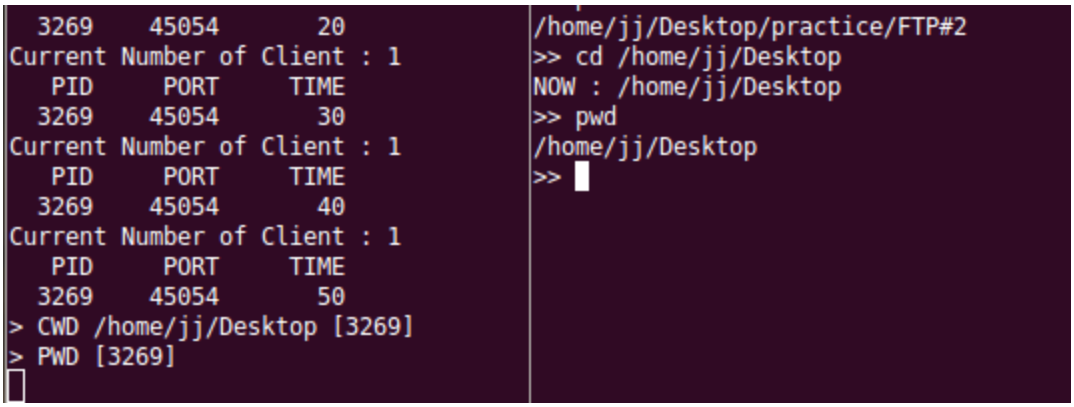
경로를 주고 dir 명령어를 사용하는 모습



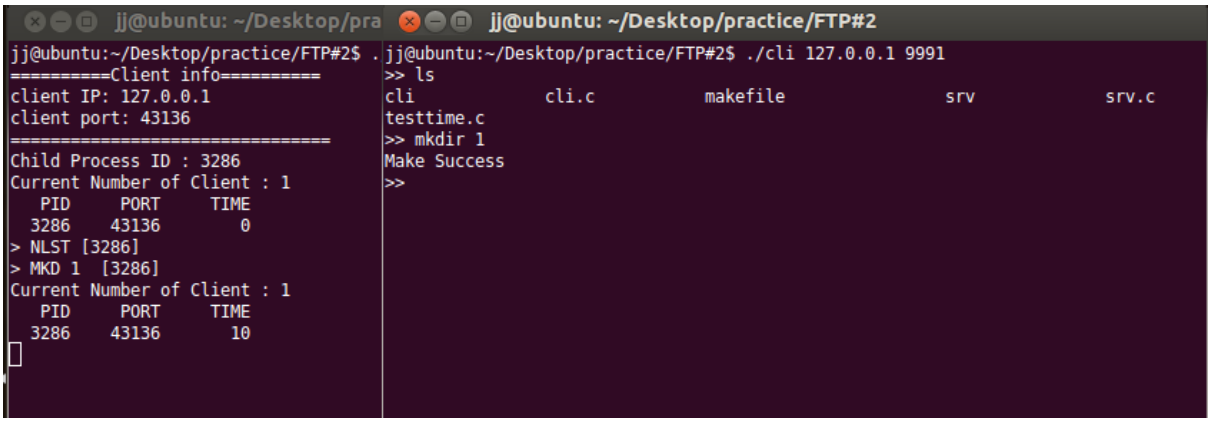
경로없이 cd를 사용한 모습



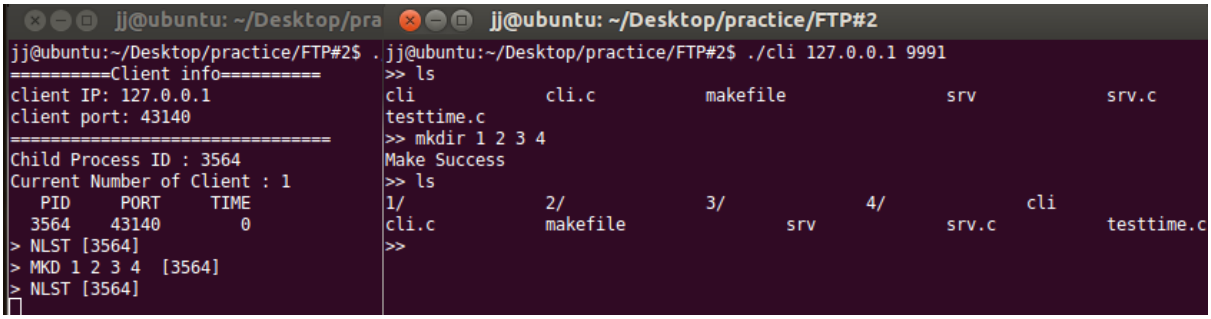
cd .. 을 사용한 모습



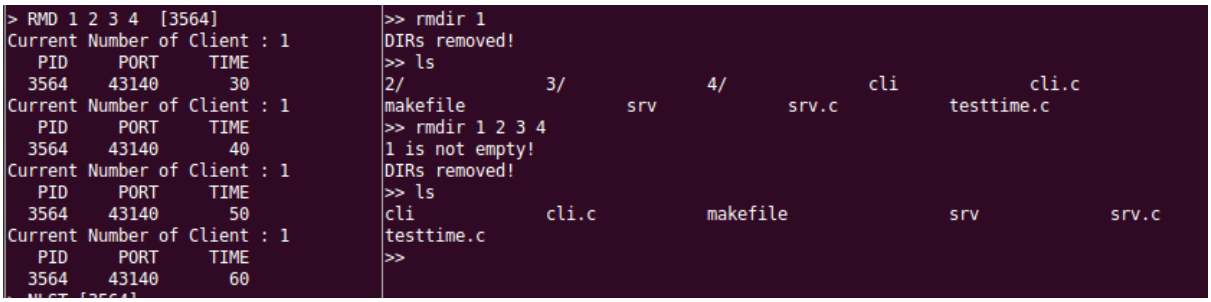
경로를 주고 cd를 사용한 모습



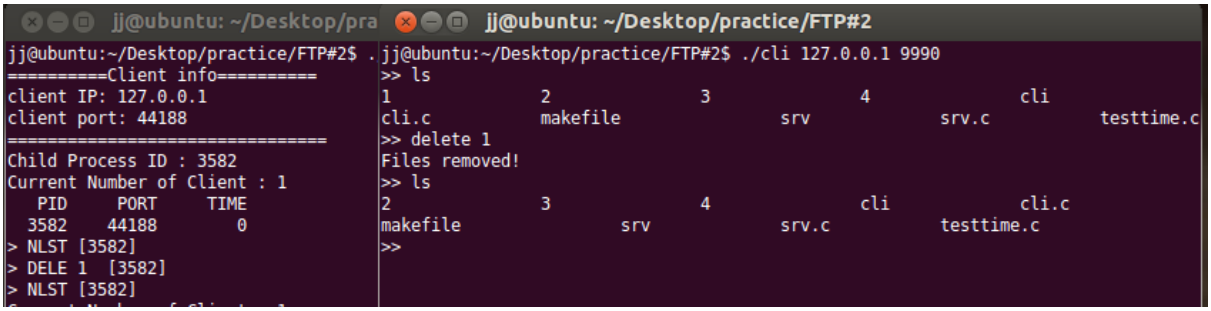
mkdir 명령어를 사용해 1디렉토리를 생성하는 모습



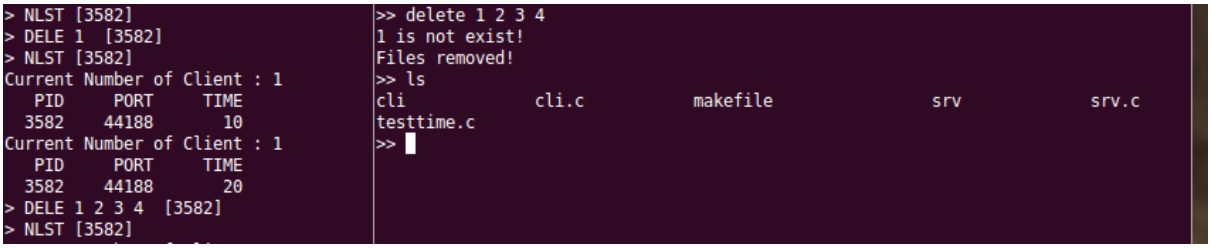
mkdir 1 2 3 4를 통해 여러 개의 디렉토리를 생성하는모습



rmdir 1을 사용하여 1디렉토리를 먼저 삭제하고, 다시 rmdir 1 2 3 4를 진행하는데 1디렉토리가 존재하지않아 에러 메시지를 출력하고 나머지 디렉토리를 삭제하는모습



delete 명령어를 사용해 1을 삭제하는 모습



delete 1 2 3 4를 진행하는데 1이 존재하지않아 에러 메시지를 출력하고 나머지 파일을 삭제하는모습

5. 결론 및 고찰

이번 프로젝트를 진행하면서 signal이 프로세스와 어떻게 영향을 미치는지 잘 알게되었고, fork()명령어를통해 생성된 자식프로세스들과 부모 프로세스들의 연결이 어떻게 이루어지는지 깨닳을 수 있었다.

직전 practice에서 시그널들을 어떻게 다루는지 잠깐 연습해 보았지만 그때는 잘 이해가 가지 않았는데 이번에 부모 프로세스에서 자식 프로세스들에게 signal을 보내 관리를 하게되면서 많은 이해가 필요했다. 또한 time 관련 함수들을 처음 사용해 보았는데, 처음 사용하다보니 익숙하지 않아 연습 파일들을 만들어 필요한 부분들을 연습해 보았다.

이번 프로젝트에서 가장 큰 문제는 서버에서 ctrl + c 로 프로세스를 종료했을때 모든 클라이언트를 종료하고 자신도 종료되는것이었는데, 서버측에서 모든 자식프로세스를 종료하고 부모 프로세스도 종료시키는 것 까지는 잘 진행됐지만, 클라이언트의 터미널이 종료되지 않는 것이 문제였다. 이 부분은 클라이언트가 서버에 접속했을 때 클라이언트 프로세스의 pid를 받아와 서버측에서 저장하고, 후에 서버에서 ctrl + c로 프로세스를 종료시킬때 자식프로세스와 연결된 클라이언트까지 동시에 종료시키는 방법으로 해결하였다.