리눅스 시스템 프로그래밍 설계 과제 2 : ssu\_mntr

20162481 나반

컴퓨터학부 안승훈

1. 개요

- ssu\_mntr 프로그램은 지정된 디렉토리를 모니터링 하며 변경사항을 로그로 남기는 프로그램이다.

- 디몬 프로세스를 이용하여 디렉토리를 백그라운드에서 독립적으로 모니터링 할 수 있도록 한다.

- 모니터링 프로세스 이외에도 프롬프트를 통해, 각종 파일과 디렉토리에 대해 작업할 수 있도록 한다.

- 지정한 디렉토리 내에서 파일의 삭제, 복원, 크기 확인, 디렉토리 구조를 확인할 수 있도록 한다.

2. 설계

- 기본 모니터링

\* 디몬 프로세스를 생성하여, 프롬프트를 동작시키는 프로세스와 별개로 디몬 프로세스가 백그라운드에서 특정디 렉토리의 파일의 변화를 감지할 수 있도록 한다.

\* 디렉토리 내에 파일이 생성, 수정, 삭제될 경우, log.txt 파일에 변경 사항을 추가한다.

\* log.txt 파일에는 생성, 수정, 삭제 시간과 수행내용, 파일 이름을 작성한다.

\* 새로운 변경 사항은 log.txt 끝에 이어서 작성한다.

- 기본 프롬프트

\* 프롬프트는 “학번>” 의 모양을 가져야 한다.

\* delete, size, recover, tree, exit, help 명령어만 수행하고, 그 외의 명령은 모두 help와 동일하게 처리한다.

\* 엔터만 입력될 경우 프롬프트를 재출력한다.

- delete <FILENAME> <END\_TIME> <OPTION>

\* 파일 이름과 시간, 옵션을 입력 받아 지정된 시간에 파일을 삭제한다.

\* 삭제한 파일은 trash 디렉토리로 이동시킨다. trash 디렉토리가 없을 경우 새로 생성한 후 이동한다.

\* trash 디렉토리에 파일을 저장하는 files 디렉토리와 삭제된 파일의 정보를 저장하는 info 파일을 생성한다.

\* info 디렉토리가 2KB를 초과하는 경우 오래된 파일과 파일의 정보를 삭제한다.

\* 파일의 이름은 상대경로, 절대경로 모두 가능해야 하고, 시간이 입력되지 않는 경우 즉시 삭제한다.

\* -i 옵션의 경우 삭제 시 trash 디렉토리로 이동하지 않고 바로 삭제하고, -r 옵션의 경우 삭제 시에 삭제 여 부를 재확인한다.

- size <FILENAME> <OPTION>

\* size는 파일 경로(상대경로)를 입력 받아, 그 파일의 크기를 출력한다.

\* 출력 시에는 문자열 오름차순으로 출력해야 하고, byte 단위로 출력한다.

\* 디렉토리와 파일이 모두 크기를 출력할 수 있어야 하고, 출력은 상대경로로 해야 한다.

\* -d <NUMBER> 옵션이 주어질 경우, 주어진 숫자 만큼의 하위 디렉토리까지 출력한다.

- recover <FILENAME> <OPTION>

\* trash 디렉토리에 있는 파일을 원래의 경로로 복구한다.

\* 동일한 이름의 삭제된 파일이 trash에 여러 개 있을 경우, 이를 파일이름, 삭제 시간, 수정시간을 출력한다.

\* 사용자가 목록을 보고 recover할 파일을 고를 수 있도록 한다.

\* 복구 시, 원래의 파일 위치에 같은 이름의 파일이 있는 경우, 파일의 앞에 “숫자\_” 를 추가한다.

\* -l 옵션이 적용되면, trash 디렉토리 내의 파일과 삭제시간을 시간 오름차순으로 출력하고 명령어를 진행한다.

- tree

\* check 디렉토리의 구조를 tree 형태로 보여준다.

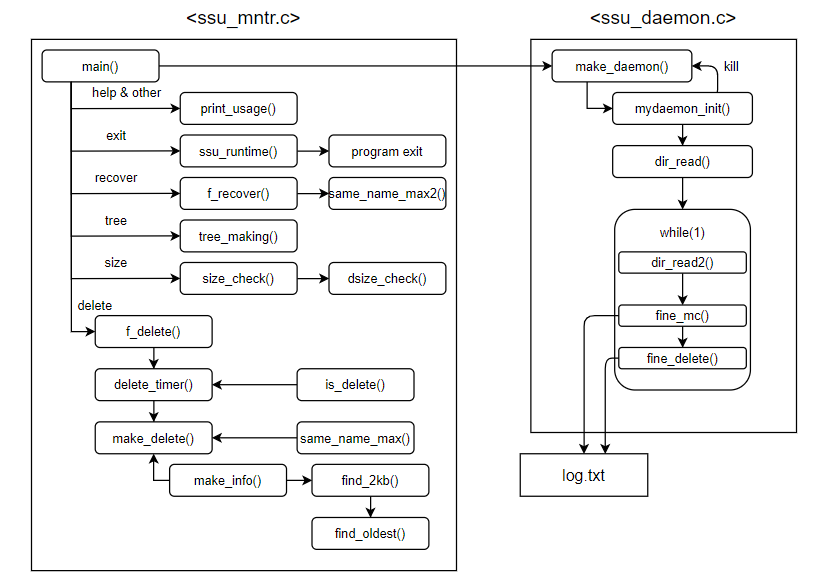
- exit

\* 프로그램을 종료한다.

- help

\* 명령어들의 사용법을 출력한다.

- 위와 같은 작업을 위해 여러 가지 함수를 설계하였다. 이를 순서도로 나타내었다.



- 프롬프트를 생성하기 전, 디몬 프로세스를 생성하는 함수를 호출하여 디몬 프로세스를 생성한다.

- 이후 프롬프트를 통해 입력 받는 명령어를 확인하여, 각 명령에 맞는 작업을 실행하는 함수를 실행한다.

- 종료 시에는 프로그램의 실행 시간을 출력하고 프로그램을 종료한다.

3. 구현

- 함수의 프로토타입을 통해, 프로그램에서 사용한 함수를 설명한다.

<ssu\_mntr.c>

void ssu\_runtime(struct timeval \*begin\_t, struct timeval \*end\_t); //프로그램 실행 시간 측정함수

void print\_usage(void); //도움말을 출력하는 함수

void make\_daemon(void); //디몬 프로세스를 호출하기 위한 함수 -> ssu\_daemon.c에 존재

void tree\_making(char path[]); //트리를 그리기 위한 함수

void size\_check(char input[]); //size를 체크하는 함수

int dsize\_check(char name[]); //디렉토리의 size를 체크하기 위한 함수

void f\_delete(char input[]); //delete 명령어를 처리하는 함수

void delete\_timer(int signo); //delete를 위한 alarm을 받는 핸들러 함수

int is\_delete(void); //delete에 -r 옵션일때 정말로 삭제할 것인지를 묻는 함수

void make\_delete(void); // delete시에 파일 이름을 겹치지 않게 바꿔서 delete 하는 함수

int same\_name\_max(char fname[]); //원 이름이 같은 파일의 갯수를 세는 함수

int dir\_skip(const struct dirent \*info); //scandir의 filter 함수

void make\_info(char fname[]); //delete 시에 info 파일을 만드는 함수

void find\_2kb(void); //info 폴더가 2KB를 넘는지 확인하는 함수

void find\_oldest(void); //info 폴더에서 가장 오래된 파일을 찾아 삭제하는 함수

void f\_recover(char input[]); //recover 명령어를 진행하는 함수

int same\_name\_max2(char fname[], char last\_locdir[]);

//recover 시에 복원될 폴더에 같은 이름을 가진 파일이 있는지 확인하는 함수

<ssu\_daemon.c>

int mydaemon\_init(void); //디몬프로세스를 생성하는 함수

void dir\_read(char path[]); //초기에 디렉토리를 스캔하는 함수

void dir\_read2(char path[]); //while문 내부에서 반복해서 디렉토리를 스캔하는 함수

void find\_mc(void); //modify와 create를 탐색하는 함수

void find\_delete(void); //delete를 탐색하는 함수

void print\_day(struct tm \*t, char buf[]); //날짜를 출력하기 위한 함수

int dir\_check(const struct dirent \*info); //scandir을 위한 필터함수

<ssu\_daemon.c>

**- void make\_daemon(void)**

void make\_daemon(void) {

pid\_t pid;

if((pid = fork()) < 0) {

fprintf(stderr, "fork error\n");

exit(1);

}

else if(pid == 0)

mydaemon\_init();

return;

}

- make\_daemon 함수는 fork를 통해 자식 프로세스가 디몬 프로세스를 생성하게 하는 함수이다.

- 부모 프로세스는 fork이후 종료하여 상위의 프로세스와 연결을 끊음

**- int mydaemon\_init(void)**

int mydaemon\_init(void) {

pid\_t pid;

int fd, maxfd, fd2;

char curpath[1024] = {0, };

char tmppath[1024] = {0, };

char buf[1024] = {0, };

// 디렉토리의 파일들을 저장하기 위한 무언가가 필요해

if((pid = fork()) < 0) {

fprintf(stderr, "fork error\n");

exit(1);

}

else if (pid != 0)

exit(0);

setsid();

signal(SIGTTIN, SIG\_IGN);

signal(SIGTTOU, SIG\_IGN);

signal(SIGTSTP, SIG\_IGN);

maxfd = getdtablesize();

for(fd = 0; fd < maxfd; fd++)

close(fd);

umask(0);

getcwd(curpath, 1024); //현재 작업디렉토리 저장

strcpy(tmppath, curpath); //임시로 현재 디렉토리 추가로 저장

strcat(curpath, "/log.txt");

strcat(tmppath, "/check");

chdir("/");

fd = open("/dev/null", O\_RDWR);

dup(0);

dup(0);

if((logfp = fopen(curpath, "a+")) < 0) {

fprintf(stderr, "fopen error\n");

return 0;

} //curpath는 ./log.txt의 절대경로

setbuf(logfp, NULL); //버퍼를 없앰

chdir(tmppath); // check 폴더로 이동

dir\_read(tmppath); //최초 디렉토리 스캔

while(1) {

check\_size = 0;

dir\_read2(tmppath); //비교를 위해 디렉토리를 읽어옴

find\_mc(); //modify create 확인

find\_delete(); //delete 확인

sleep(1); //1초 멈춤

}

return 0;

}

- 디몬 코딩 규칙을 포함하고 있는 디몬 프로세스 생성 함수이다.

- 디몬 코딩 규칙 후에, log.txt을 생성하고 디렉토리를 스캔하여 비교하는 작업을 반복한다.

**- void find\_delete(void)**

void find\_delete(void) {

struct tm \*t;

time\_t now;

char buf[512];

int idx = 0;

for(int i = 0; i < fs\_size; i++) {

idx = 0;

for(int j = 0; j < check\_size; j++) {

if(!(strcmp(fsarr[i].fpath, checkarr[j].fpath)))

idx = 1; //이름이 일치하는 파일이 있는 경우 idx = 1

}

if(idx == 0) { //이름이 일치하는 경우가 없는 경우

now = time(NULL);

t = localtime(&now);

print\_day(t, buf);

fprintf(logfp, "%s", buf);

fprintf(logfp, "[delete\_");

fprintf(logfp, "%s", strstr(fsarr[i].fpath, "check/")+6);

fprintf(logfp, "]\n");

fsarr[i] = fsarr[fs\_size-1];

strcpy(fsarr[fs\_size - 1].fpath, "\0");

fsarr[fs\_size - 1].c\_time = 0;

fs\_size--;

} // 기존에 스캔한 목록에서 삭제된 파일을 삭제해줌

}

return;

}

- 처음에 스캔한 목록과 나중에 스캔한 내용을 비교하여 삭제된 파일을 알아내는 함수이다.

- 처음 스캔한 목록에는 있지만, 나중에 스캔한 목록에는 없는 경우, 삭제되었다고 판단한다.

**- void find\_mc(void)**

void find\_mc(void) //modify와 create를 판단하는 함수

{

struct tm \*t;

time\_t cur;

char buf[512];

for(int i = 0; i < check\_size; i++) {

int cnt = 0;

for(int j = 0; j < fs\_size; j++) {

if(!strcmp(checkarr[i].fpath, fsarr[j].fpath)) {

cnt= 1;

if(checkarr[i].c\_time != fsarr[j].c\_time) {

t = localtime(&checkarr[i].c\_time);

print\_day(t, buf);

fprintf(logfp, "%s", buf);

fprintf(logfp, "[modify\_");

fprintf(logfp, "%s", strstr(checkarr[i].fpath, "check/")+6);

fprintf(logfp, "]\n");

fsarr[j].c\_time = checkarr[i].c\_time;

break;

}

else

continue;

} //둘을 비교했을때 이름이 같고, 시간이 다르면 modify, 구 목록 업데이트 해줌

}

if(!cnt) {

strcpy(fsarr[fs\_size].fpath, checkarr[i].fpath);

fsarr[fs\_size].c\_time = checkarr[i].c\_time;

fs\_size++;

cur = time(NULL);

t = localtime(&cur);

print\_day(t, buf);

fprintf(logfp, "%s", buf);

fprintf(logfp, "[create\_");

fprintf(logfp, "%s", strstr(checkarr[i].fpath, "check/")+6);

fprintf(logfp, "]\n");

} //이름이 같은 적이 한번도 없었다면 새로 생긴 파일로 판단함

}

return;

}

- 기존의 목록과 새 목록을 비교하여 create와 modify를 판단하는 함수이다.

- 같은 이름의 파일이 존재하고 시간이 바뀌었으면 modify, 기존 목록에 없던 파일이 새 목록에 있는 경우create로 판단한다.

**- void print\_day(struct tm \*t, char buf)**

void print\_day(struct tm \*t, char \*buf)

{

sprintf(buf, "[%4d-%02d-%02d %02d:%02d:%02d]",

1900+t->tm\_year, t->tm\_mon+1, t->tm\_mday, t->tm\_hour, t->tm\_min, t->tm\_sec);

return;

}

- 문자열에 tm 구조체를 시간형식으로 작성하는 함수이다.

**- int dir\_check(const struct dirent \*info)**

int dir\_check(const struct dirent \*info) {

if(!strncmp(info->d\_name, ".", 1))

return 0;

else

return 1;

}

- scandir에서 사용하는 필터 함수이다.

- 파일 이름이 ‘.’으로 시작하는 경우 스캔 목록에 포함시키지 않는다.

**- void dir\_read(char path[])**

void dir\_read(char path[]) { // 초기 파일 목록을 생성하는 함수

struct dirent \*\*namelist;

int count;

char buf[1024] = {"./\0"};

char buf2[1024];

char \*ptr;

memset(buf2, 0, sizeof(buf2));

if((count = scandir(path, &namelist, dir\_check, alphasort)) == -1) {

return;

}

chdir(path);

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

struct stat statbuf;

strcpy(buf, "./\0");

memset(buf2, 0, sizeof(buf2));

if(!(strcmp(namelist[i]->d\_name, ".")) || !(strcmp(namelist[i]->d\_name, "..")))

continue;

stat(namelist[i]->d\_name, &statbuf);

realpath(namelist[i]->d\_name, buf2);

ptr = strstr(buf2, "check");

strcat(buf, ptr);

strcpy(fsarr[fs\_size].fpath, buf);

fsarr[fs\_size].c\_time = statbuf.st\_mtime;

fs\_size++;

if(S\_ISDIR(statbuf.st\_mode))

dir\_read(namelist[i]->d\_name);

} //파일의 경로를 절대경로로 만들어 준 후, check 이후면 저장해서 상대경로로 구조체에 저장

chdir("..");

return;

}

- 초기 파일 목록을 만들어주는 함수이다.

- 디렉토리 내의 모든 파일을 읽어와 구조체 배열에 저장한다. 디렉토리일 경우 재귀적으로 순환하여 탐색한다.

- 구조체 배열의 크기도 저장하여 현재 파일의 총 개수를 판단할 수 있게 한다.

**- void dir\_read2(char path[])**

void dir\_read2(char path[]) { //새 목록을 생성하는 함수

struct dirent \*\*namelist;

int count;

char buf[1024] = {"./"};

char buf2[1024];

char \*ptr;

memset(buf2, 0, sizeof(buf2));

if((count = scandir(path, &namelist, dir\_check, alphasort)) == -1) {

return;

}

chdir(path);

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

struct stat statbuf;

strcpy(buf, "./");

memset(buf2, 0, sizeof(buf2));

if(!(strcmp(namelist[i]->d\_name, ".")) || !(strcmp(namelist[i]->d\_name, "..")))

continue;

stat(namelist[i]->d\_name, &statbuf);

realpath(namelist[i]->d\_name, buf2);

ptr = strstr(buf2, "check");

strcat(buf, ptr);

strcpy(checkarr[check\_size].fpath, buf);

checkarr[check\_size].c\_time = statbuf.st\_mtime;

check\_size++;

if(S\_ISDIR(statbuf.st\_mode))

dir\_read2(namelist[i]->d\_name);

} //파일의 경로를 절대경로로 만들어 준 후, check 이후면 저장해서 상대경로로 구조체에 저장

chdir("..");

return;

}

- 새로 파일의 목록을 읽어서 저장하는 함수이다.

- 디렉토리의 모든 파일을 읽어서 구조체 배열에 저장한다. 디렉토리의 경우 재귀적으로 순환하여 탐색한다.

- 읽어온 파일의 총 개수를 저장하여 확인할 수 있도록 한다.

<ssu\_mntr.c>

**- int main(void)**

int main(void) {

struct timeval begin\_t, end\_t;

gettimeofday(&begin\_t, NULL);

char input[BUF\_LEN]; //입력받은 문장 전체

char const\_input[BUF\_LEN];

char todo[10]; //명령어 분리

memset(input, 0, sizeof(input));

//디몬 부르기

make\_daemon();

//trash 폴더가 없는 경우 만들기

if(access("./trash", F\_OK) == -1)

mkdir("./trash", 0777);

if(access("./trash/files", F\_OK) == -1)

mkdir("./trash/files", 0777);

if(access("./trash/info", F\_OK) == -1)

mkdir("./trash/info", 0777);

getcwd(current\_wd, BUF\_LEN); //현재 작업디렉토리 저장

while(1) {

chdir(current\_wd);

memset(input, 0, sizeof(input));

memset(const\_input, 0, sizeof(const\_input));

printf("20162481>");

fgets(input, BUF\_LEN, stdin);

strcpy(const\_input, input);

memset(check, 0, sizeof(check)); //check 배열 초기화

checksum = 0;

strcpy(todo, strtok(input, " "));

if(!strcmp(todo, "\n")) //엔터만 입력되면 프롬프트 재출력

continue;

if(todo[strlen(todo) - 1] == '\n')

todo[strlen(todo) - 1] = '\0'; //뒤에 있는 개행 문자 제거

for(int i = 0; i < 6; i++) {

if(!strcmp(todo, prompt[i])) {

check[i] = true;

checksum++;

}

} //입력된 명령어와 지정된 명령어를 비교함

if(checksum == 0) { //명령어가 올바르지 않은 경우 도움말 출력

print\_usage();

continue;

}

if(check[0] == true) //delete일때

f\_delete(const\_input);

else if (check[1] == true) {//size 일때

size\_check(const\_input);

}

else if (check[2] == true) //recover일때

f\_recover(const\_input);

else if (check[3] == true) //tree 일때

tree\_making("./check");

else if (check[4] == true) { //exit 일때

printf("Exiting......\n");

gettimeofday(&end\_t, NULL);

ssu\_runtime(&begin\_t, &end\_t);

exit(0);

}

else if (check[5] == true) //help일때

print\_usage();

}

return 0;

}

- 프롬프트를 구성하는 메인 함수이다. 명령어를 입력받아 처리하고 각 함수를 호출하는 역할을 한다.

**- void ssu\_runtime(struct timeval \*begin\_t, struct timeval \*end\_t)**

void ssu\_runtime(struct timeval \*begin\_t, struct timeval \*end\_t)

{

end\_t->tv\_sec -= begin\_t->tv\_sec; //끝난 시간에서 처음시간을 뺌

if(end\_t->tv\_usec < begin\_t->tv\_usec){

end\_t->tv\_sec--;

end\_t->tv\_usec += SECOND\_TO\_MICRO;

} //나중의 us값이 작을경우 1초를 뺴서 us값이 더해줌

end\_t->tv\_usec -= begin\_t->tv\_usec;

printf("Runtime: %ld:%02ld:%06ld(min:sec:usec)\n",

end\_t->tv\_sec / 60, end\_t->tv\_sec % 60, end\_t->tv\_usec);

} //프로그램 수행시간을 분, 초, 마이크로초로 출력

- 프로그램의 수행시간을 측정하여 출력해 주는 함수이다.

**- void f\_recover(char input[])**

void f\_recover(char input[]) { //recover하는 함수

char info\_path[BUF\_LEN] = "./trash/info/";

char file\_path[BUF\_LEN] = "./trash/files/";

char fname[BUF\_LEN]; // 원래의 이름

char dest[BUF\_LEN];

char \*ptr = input;

int count;

struct dirent \*\*namelist1;

struct dirent \*\*namelist2;

char f\_name[BUF\_LEN]; //현재 trash 폴더 안의 이름

char buf[BUF\_LEN];

char buf\_D[BUF\_LEN];

char buf\_M[BUF\_LEN];

char last\_locdir[BUF\_LEN];

int pfx, cnt = 0;

int choose, check;

int idc = 0, frontnum;

char tmp[BUF\_LEN];

FILE \*fp;

memset(fname, 0, sizeof(fname));

memset(dest, 0, sizeof(dest));

memset(f\_name, 0, sizeof(f\_name));

memset(buf, 0, sizeof(buf));

memset(buf\_D, 0, sizeof(buf\_D));

memset(buf\_M, 0, sizeof(buf\_M));

memset(last\_locdir, 0, sizeof(last\_locdir));

strcpy(fname, ptr+8); //명령어를 제외하고 파일 이름만 받아들임

if(fname[strlen(fname) - 1] == '\n')

fname[strlen(fname) - 1] = '\0';

if((count = scandir("./trash/info", &namelist1, dir\_skip, alphasort)) == -1) {

fprintf(stderr, "scandir error in f\_recover function\n");

return;

}

check = 0;

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

strcpy(buf, namelist1[i]->d\_name);

while(1) {

if(buf[cnt] == '\_')

break;

else

cnt++;

}

ptr = buf;

strcpy(f\_name, ptr + cnt + 1);

if(!strcmp(fname, f\_name)) {

check = 1;

memset(buf, 0, sizeof(buf));

strcpy(buf, info\_path);

strcat(buf, namelist1[i]->d\_name);

if((fp = fopen(buf, "r")) < 0) {

fprintf(stderr, "fopen error in recover function\n");

return;

}

memset(buf, 0, sizeof(buf));

fgets(buf, sizeof(buf), fp); //trash info 읽음

memset(buf, 0, sizeof(buf));

fgets(buf, sizeof(buf), fp); //원래 파일 경로 읽음

memset(buf, 0, sizeof(buf));

fgets(buf\_D, sizeof(buf\_D), fp); //삭제 시간 얻어옴

fgets(buf\_M, sizeof(buf\_M), fp); // 최종 수정 시간 얻어옴

if(buf\_D[strlen(buf\_D) - 1] == '\n')

buf\_D[strlen(buf\_D) - 1] = '\0';

if(buf\_M[strlen(buf\_M) - 1] == '\n')

buf\_M[strlen(buf\_M) - 1] = '\0';

//D M 출력 준비

printf("%d. %s %s %s\n", atoi(namelist1[i]->d\_name), f\_name, buf\_D, buf\_M);

fclose(fp);

} //사용자가 입력한 파일이름과 이름이 같은 파일을 모두 출력해줌

}

if(check == 0) {

printf("There is no '%s' in 'trash' directory!\n", fname);

return;

} //이름이 같은 파일이 없는 경우 에러 출력 후 리턴

printf("Choose : ");

scanf("%d", &choose);

getchar();

//사용자가 어떤 파일을 복구 할 것인지 확인함

//숫자 + \_ + 파일 이름 해서 info 파일 열어서 파일 원래 위치 확인

// 그 디렉토리 읽어서 같은 이름 파일 있는 지 확인

//없으면 앞에 숫자 띠고 가는거고 있으면 붙여서 가는거고

memset(f\_name, 0, sizeof(f\_name));

memset(buf, 0, sizeof(buf));

sprintf(f\_name, "%d\_%s", choose, fname); //읽을 파일 확정

strcpy(buf, info\_path);

strcat(buf, f\_name); //info 파일의 위치 만듬

if((fp = fopen(buf, "r")) < 0) {

fprintf(stderr, "fopen error in recover function 2\n");

return;

}

memset(buf, 0, sizeof(buf));

fgets(buf, sizeof(buf), fp); //trash info 읽어서 버림

memset(buf, 0, sizeof(buf));

fgets(buf, sizeof(buf), fp); //파일의 경로 읽어옴

if(buf[strlen(buf) - 1] == '\n')

buf[strlen(buf) - 1] = '\0';

cnt = 0;

for(int i = 0; i < strlen(buf); i++) {

if(buf[i] == '/')

cnt = i;

}

strncpy(last\_locdir, buf, cnt); //원래 파일이 있던 위치를 탐색

if((count = scandir(last\_locdir, &namelist2, dir\_skip, alphasort)) < 0) {

fprintf(stderr, "scandir error in recover function 2\n");

return;

}

ptr = tmp;

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

//memset(tmp, 0, sizeof(tmp));

//strcpy(tmp, namelist2->d\_name);

if(!strncmp(fname, namelist2[i]->d\_name, strlen(namelist2[i]->d\_name))) {

idc = 1;

break;

}

} //같은 이름 파일이 있으면 그만 찾고 break

if(idc == 0) { //이름이 겹치는 파일이 없을떄

strcat(file\_path, f\_name);

rename(file\_path, buf);

strcat(info\_path, f\_name);

remove(info\_path);

return;

}

else { //이름이 겹치는 파일이 있을때

frontnum = same\_name\_max2(fname, last\_locdir); //앞에 붙은 숫자 제일 큰거 구하기

frontnum++;

memset(tmp, 0, sizeof(tmp));

sprintf(tmp, "/%d\_%s", frontnum, fname); //앞에 숫자 붙여서 rename함

strcat(last\_locdir, tmp);

strcat(file\_path, f\_name);

rename(file\_path, last\_locdir);

strcat(info\_path, f\_name);

remove(info\_path); //info파일 지움

return;

}

}

- recover 시에 호출되는 함수이다. 사용자가 입력한 파일의 정보를 보고 trash 디렉토리에서 검색한다.

- 파일을 복원할 때, 파일의 원래 위치에 같은 이름의 파일이 있는 경우 파일 이름 앞에 ‘숫자\_’를 붙인다.

**- int same\_name\_max2(char fname[], char last\_locdir[])**

int same\_name\_max2(char fname[], char last\_locdir[]) { //과거의 폴더에서 같은 이름 파일의 숫자의 최댓값을 계산해주는 함수

struct dirent \*\*namelist;

int count, loc = 0;

char tmp[BUF\_LEN];

int maxnum = 0;

int num = 0;

char \*ptr = tmp;

if((count = scandir(last\_locdir, &namelist, dir\_skip, alphasort)) == -1) {

fprintf(stderr, "last directory scan error in same\_name\_max2\n");

return count;

}

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

ptr = tmp;

memset(tmp, 0, sizeof(tmp));

strcpy(tmp, namelist[i]->d\_name);

for(int j = 0; j < strlen(tmp); j++) {

if(tmp[j] == '\_') {

loc = j;

break;

} // 언더바의 위치를 찾아서 그 앞의 숫자를 제거할 수 있도록 함

}

if(loc != 0)

ptr = ptr + loc + 1;

//이제 파일 이름만 남게 됨

if(!strcmp(fname, ptr)) {

num = atoi(namelist[i]->d\_name); //1\_1.c라면 num은 1이 될꺼임

if(maxnum < num)

maxnum = num; //앞에 붙은 값의 최대를 구함

}

}

for(int i = 0; i <count; i++)

free(namelist[i]);

free(namelist);

return maxnum; //같은 이름을 가진 파일이 없다면 0을 리턴함

}

- recover 시에 파일의 원래 위치에 파일과 같은 이름을 가진 파일이 있는지 확인하는 함수이다.

- 같은 이름의 파일이 있는 경우 파일 이름 앞에 붙일 숫자를 리턴한다.

**- void f\_delete(char input[])**

void f\_delete(char input[]) {

char backup\_input[BUF\_LEN] = {0, };

char fname[BUF\_LEN] = {0, }; //파일 이름

char \*tmp;

char rpath[BUF\_LEN] = {0, }; // 파일 절대경로

char d\_day[BUF\_LEN] = {0, }; // 삭제 날짜

char d\_time[BUF\_LEN] = {0, }; //삭제 시간

struct tm t;

time\_t deltime, curtime;

int timegap = 0;

char option[3]; //delete 할때의 옵션

struct stat statbuf;

struct stat stattmp;

int t\_checksum = 0;

memset(backup\_input, 0, sizeof(backup\_input));

memset(fname, 0, sizeof(fname));

memset(rpath, 0, sizeof(rpath));

memset(d\_day, 0, sizeof(d\_day));

memset(d\_time, 0, sizeof(d\_time));

memset(option, 0, sizeof(option));

signal(SIGALRM, delete\_timer);

input = input+7;

strcpy(backup\_input, input); //명령어 잘라낸 인풋 배열 저장해둠

chdir("./check");

tmp = strtok(input, " ");

if(tmp != NULL)

strcpy(fname, tmp);

if(fname[strlen(fname) - 1] == '\n') // 끝에 개행문자까지 포함된 경우 개행문자 삭제

fname[strlen(fname) - 1] = '\0';

realpath(fname, rpath);

if(lstat(rpath, &statbuf) < 0) { //파일 명이 이상한거 일단 다 걸러짐

fprintf(stderr, "File is not exist!!\n");

chdir(current\_wd);

return;

} // 존재하지 않는 파일을 삭제하려 하는 경우 에러 출력 후 리턴

//printf("hi : %s\n", rpath);

//이 아래부터 파일 명은 멀쩡하다

if(strlen(backup\_input) == strlen(fname) + 1){

strcpy(fname\_delete, rpath);

memset(option\_delete, 0, sizeof(option\_delete));

chdir(current\_wd);

delete\_timer(SIGALRM);

return;

}

else if((strlen(backup\_input) > strlen(fname) + 1) && backup\_input[strlen(fname) + 1] == '-') {

tmp = strtok(NULL, " "); //-가 나온것은 시간이 입력되지 않았다는거

strcpy(option, tmp);

//printf("%s\n", option);

strcpy(option\_delete, option);

strcpy(fname\_delete, rpath);

chdir(current\_wd);

delete\_timer(SIGALRM);

return;

} //시간이 있을때

else if((strlen(backup\_input) > strlen(fname) + 1) && backup\_input[strlen(fname) + 1] >= '0'

&& backup\_input[strlen(fname) + 1] <= '9'){

tmp = strtok(NULL, "-"); //년도 커팅

t.tm\_year = atoi(tmp) - 1900;

if(t.tm\_year < 0)

t\_checksum++;

tmp = strtok(NULL, "-"); //월 커팅

t.tm\_mon = atoi(tmp) - 1;

if(t.tm\_mon < 0)

t\_checksum++;

tmp = strtok(NULL, " "); //일 커팅

t.tm\_mday = atoi(tmp);

if(t.tm\_mday < 1)

t\_checksum++;

tmp = strtok(NULL, ":");

t.tm\_hour = atoi(tmp);

if(t.tm\_hour < 0)

t\_checksum++;

tmp = strtok(NULL, " ");

t.tm\_min = atoi(tmp);

if(t.tm\_min < 0)

t\_checksum++;

t.tm\_sec = 0;

if(t\_checksum > 0) {

fprintf(stderr, "Time input error!!\n");

chdir(current\_wd);

return;

}

if((tmp = strtok(NULL, " ")) != NULL) {

strcpy(option, tmp);

strcpy(option\_delete, option);

}

deltime = mktime(&t);

curtime = time(NULL);

//printf("%ld %ld\n", deltime, curtime);

//현재시간, 삭제 시간 구함

timegap = deltime - curtime;

if(timegap < 0) {

fprintf(stderr, "Delete time error!!\n");

chdir(current\_wd);

return;

}

memset(fname\_delete, 0, sizeof(fname\_delete));

memset(option\_delete, 0, sizeof(option\_delete));

strcpy(fname\_delete, rpath); //시그널 핸들러 함수에서 사용하기 위해 전역변수로 넘겨줌

strcpy(option\_delete, option);

alarm(timegap);

chdir(current\_wd);

return;

}

}

- delete 명령어를 실행하였을 때 호출되는 함수이다.

- 입력된 문장을 읽어들여 파일이름, 시간, 옵션을 파악한다.

**- void delete\_timer(int signo)**

void delete\_timer(int signo) { //타이머가 울리면 파일을 삭제 하는 함수

char rpath[BUF\_LEN] = {0,};

char fname[BUF\_LEN] = {0,};

char \*ptr = fname\_delete;

int option\_i = 0;

int option\_r = 0;

int loc;

memset(rpath, 0, sizeof(rpath));

memset(fname, 0, sizeof(fname));

strcpy(rpath, fname\_delete);

for(int i = 0; i < strlen(fname\_delete); i++)

if(fname\_delete[i] == '/')

loc = i;

strcpy(fname, ptr + loc + 1);

if(!strncmp(option\_delete, "-i", 2))

option\_i = 1;

else if(!strncmp(option\_delete, "-r", 2)) {

option\_r = 1;

}

if(option\_r) {

//삭제할껀지 묻는 함수

//Y는 삭제, N은 취소하고 리턴

if(is\_delete() < 0) //사용자가 n 선택했으면 취소

return;

}

if(option\_i) {

//바로 삭제

struct stat statbuf;

lstat(rpath, &statbuf);

if(!S\_ISDIR(statbuf.st\_mode)) {

remove(rpath);

return;

}

else {

printf("Failed to delete directory : '%s' isn't empty!\n", fname);

return;

}

}

else {

make\_delete();

}

return;

}

- SIGALRM이 울렸을 때 수행하는 시그널 핸들러 함수이다.

- 옵션이 적용되었으면 옵션에 따라 삭제를 진행한다.

- 최종적으로 삭제하는 함수를 호출한다.

**- int dir\_skip(const struct dirent \*info)**

int dir\_skip(const struct dirent \*info) {

if(!strcmp(info->d\_name, ".") || !strcmp(info->d\_name, ".."))

return 0;

else return 1;

} //scandir에서 현재 디렉토리와 상위 디럭토리를 제외하기 위한 함수

- scandir에서 의미 없는 디렉토리를 제외하기 위한 필터함수이다.

**- int same\_name\_max(char fname[])**

int same\_name\_max(char fname[]) { //trash 폴더에서 같은 이름 파일의 숫자의 최댓값을 계산해주는 함수

struct dirent \*\*namelist;

int count, loc = 0;

char tmp[BUF\_LEN];

int maxnum = 0;

int num = 0;

char \*ptr = tmp;

if((count = scandir("./trash/files", &namelist, dir\_skip, alphasort)) == -1) {

fprintf(stderr, "trash directory scan error in same\_name\_max\n");

return count;

}

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

ptr = tmp;

memset(tmp, 0, sizeof(tmp));

strcpy(tmp, namelist[i]->d\_name);

for(int j = 0; j < strlen(tmp); j++) {

if(tmp[j] == '\_') {

loc = j;

break;

}

}

if(loc != 0)

ptr = ptr + loc + 1;

//이제 파일 이름만 남게 됨

if(!strcmp(fname, ptr)) {

num = atoi(namelist[i]->d\_name); //1\_1.c라면 num은 1이 될꺼임

if(maxnum < num)

maxnum = num; //앞에 붙은 값의 최대를 구함

}

}

for(int i = 0; i <count; i++)

free(namelist[i]);

free(namelist);

return maxnum; //같은 이름을 가진 파일이 없다면 0을 리턴함

}

- trash 디렉토리에서 이름이 같은 파일을 확인하는 함수이다.

- 이름이 같은 파일이 있으면 그 파일 앞에 붙은 숫자의 최댓값을 리턴한다.

- 이름이 같은 파일이 없는 경우에는 0을 리턴한다.

**- void name\_delete(void)**

void make\_delete(void) {

char fname[BUF\_LEN]; //순수한 파일 이름

char path\_files[BUF\_LEN] = "trash/files/";

char rpath\_files[BUF\_LEN];

int loc;

char newfname[BUF\_LEN];

char newpath[BUF\_LEN];

memset(fname, 0, sizeof(fname));

memset(newfname, 0, sizeof(newfname));

memset(newpath, 0, sizeof(newpath));

memset(rpath\_files, 0, sizeof(rpath\_files));

realpath(path\_files, rpath\_files);

for(int i = 0; i < strlen(fname\_delete); i++)

if(fname\_delete[i] == '/')

loc = i;

//마지막 슬래시의 위치를 가져옴, 그 다음부터 파일 이름이니까

strcpy(fname, fname\_delete+loc+1); //파일 이름만 건져오기

int pfx = same\_name\_max(fname); //0이면 같은 이름을 가진 파일이 없다는것

// 0 이상이라면 같은 이름을 가진 파일이 count개 있다는것

// count 보다 하나 큰 값으로 파일 앞에 붙이기

pfx++; //pfx보다 1 큰값으로 앞에 붙여야됨

//새 파일 이름 만들기

sprintf(newfname, "/%d\_%s", pfx, fname);

strcpy(newpath, rpath\_files);

strcat(newpath, newfname);

make\_info(newfname); // info 파일 만드는 함수 호출

rename(fname\_delete, newpath);

}

- 함수를 통해 trash 폴더 내에 같은 이름의 파일이 있는 지 확인한 후,

이를 바탕으로 파일의 이름을 수정하여 trash 디렉토리 내로 rename한다.

**- void make\_info(char fname[])**

void make\_info(char fname[]) {

char file\_info[BUF\_LEN] = "./trash/info/";

FILE \*infp;

struct stat statbuf;

struct tm tM, tD;

time\_t M, D;

strcat(file\_info, fname); //info 파일을 만들 경로 + 파일 이름

if((infp = fopen(file\_info, "w+")) < 0) {

fprintf(stderr, "Fail making info file\n");

return;

}

if(lstat(fname\_delete, &statbuf) < 0) {

fprintf(stderr, "lstat error in make\_info\n");

return;

}

M = statbuf.st\_mtime;

D = time(NULL);

localtime\_r(&M, &tM);

localtime\_r(&D, &tD);

//info 파일에 정보를 작성해줌

fprintf(infp, "[Trash Info]\n");

fprintf(infp, "%s\n", fname\_delete);

fprintf(infp, "D : %d-%02d-%02d %02d:%02d:%02d\n", tD.tm\_year+1900, tD.tm\_mon+1,

tD.tm\_mday, tD.tm\_hour, tD.tm\_min, tD.tm\_sec);

fprintf(infp, "M : %d-%02d-%02d %02d:%02d:%02d", tM.tm\_year+1900, tM.tm\_mon+1,

tM.tm\_mday, tM.tm\_hour, tM.tm\_min, tM.tm\_sec);

fclose(infp);

find\_2kb(); //info파일이 2KB가 넘은 경우 가장 오래된 것을 삭제하는 함수

}

- 파일 이름과 동일한 이름의 info 파일을 info 디렉토리 내에 생성하는 함수이다.

- 파일에 대한 정보를 저장한다.

**- void find\_2kb(void)**

void find\_2kb(void) {

char buf[BUF\_LEN];

memset(buf, 0, sizeof(buf));

getcwd(buf, BUF\_LEN);

chdir(current\_wd);

char info\_path[BUF\_LEN] = "./trash/info";

int size;

char fname[BUF\_LEN] = {0,};

while(1) {

size = dsize\_check(info\_path);

if(size < 2048)

break;

else

find\_oldest(); //2KB가 넘었으면, 가장 오래된 파일을 찾아서 삭제함

}

chdir(buf);

return;

}

- info 디렉토리의 크기가 2KB를 넘어가는지 판단하는 함수이다.

- 2KB가 넘을 경우, 2KB 가 넘지 않을때까지 오래된 파일을 삭제한다.

**- void find\_oldest(void)**

void find\_oldest(void) { //가장 오래된 info 파일을 찾는 함수

char info\_path[BUF\_LEN] = "./trash/info";

char buf[BUF\_LEN];

char fname\_todelete[BUF\_LEN];

struct dirent \*\*namelist;

char \*ptr;

struct tm t;

time\_t oldest, file\_time;

int count;

FILE \*fp;

char cwd[BUF\_LEN];

char filepath[BUF\_LEN];

char infopath[BUF\_LEN];

memset(filepath, 0, sizeof(filepath));

memset(infopath, 0, sizeof(filepath));

memset(buf, 0, sizeof(buf));

memset(cwd, 0, sizeof(cwd));

memset(fname\_todelete, 0, sizeof(fname\_todelete));

getcwd(cwd, BUF\_LEN);

if((count = scandir(info\_path, &namelist, dir\_skip, NULL)) == -1) {

fprintf(stderr, "scndir error in find\_oldest\n");

return;

}

chdir(info\_path);

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

if((fp = fopen(namelist[i]->d\_name, "r")) < 0) {

fprintf(stderr, "fopen error in find\_oldest\n");

return;

}

//info 파일의 적혀있는 딜리트 타임을 읽어서 비교함

fgets(buf, BUF\_LEN, fp);

memset(buf, 0, sizeof(buf));

fgets(buf, BUF\_LEN, fp);

memset(buf, 0, sizeof(buf));

fgets(buf, BUF\_LEN, fp); //buf에 삭제 시간 읽어옴

ptr = buf;

ptr = ptr+5;

t.tm\_year = atoi(ptr) - 1900;

t.tm\_mon = atoi(ptr+5) - 1;

t.tm\_mday = atoi(ptr+8);

t.tm\_hour = atoi(ptr+11);

t.tm\_min = atoi(ptr+14);

t.tm\_sec = atoi(ptr+17);

file\_time = mktime(&t);

if(i == 0) {

oldest = file\_time;

strcpy(fname\_todelete, namelist[i]->d\_name);

}

else {

if(oldest > file\_time) {

oldest = file\_time;

memset(fname\_todelete, 0, sizeof(fname\_todelete));

strcpy(fname\_todelete, namelist[i]->d\_name);

}

}

}

chdir(cwd);

strcpy(filepath, "./trash/files/");

strcpy(infopath, "./trash/info/");

strcat(filepath, fname\_todelete);

strcat(infopath, fname\_todelete);

remove(filepath);

remove(infopath);

//원 파일과 info 파일 모두를 삭제함

for(int i = 0; i <count; i++)

free(namelist[i]);

free(namelist);

return;

}

- info파일 중에 가장 오래된 파일을 찾는 함수이다.

- info파일 내부에 적혀있는 삭제시간을 읽어들여 가장 오래된 시간을 가진 파일을 원본 파일과 함께 삭제한다.

**- int is\_delete(void)**

int is\_delete(void) { //i옵션일떄 사용자에게 삭제할지를 묻는 함수

char input;

printf("Delete [y/n]? ");

scanf("%c", &input);

getchar();

if(input == 'y')

return 1;

else

return -1;

}

- delete 시에 r 옵션이 주어졌을 때 사용자에게 재확인을 하기 위한 함수이다.

**- void size\_check(char input[])**

void size\_check(char input[]) { //입력받은 파일 또는 디렉토리의 사이즈를 확인

char name[BUF\_LEN] = {0, };

char rpath[BUF\_LEN] = {0, };

char \*tmp;

struct stat statbuf;

int res = 0;

int maxdepth = 1;

tmp = strtok(input, " "); //명령어 끌어옴

tmp = strtok(NULL, " "); //파일 명을 tmp에 저장함

strcpy(name, tmp);

if(strlen(input) > 5 + strlen(name)) {

tmp = strtok(NULL, " "); //옵션 받아옴

tmp = strtok(NULL, " ");

maxdepth = atoi(tmp);

printf("%d\n", maxdepth);

}

if(name[strlen(name) - 1] == '\n')

name[strlen(name) - 1] = '\0';

if(lstat(name, &statbuf) < 0) {

fprintf(stderr, "stat error in size\_check\n");

return;

}

if(S\_ISREG(statbuf.st\_mode)) {

if(name[0] == '/' || name[0] == '.')

printf("%ld %s\n",statbuf.st\_size, name);

else

printf("%ld ./%s\n",statbuf.st\_size, name);

} //입력된 것이 파일일 경우 사이즈 출력

else if(S\_ISDIR(statbuf.st\_mode)) {

res = dsize\_check(name);

if(name[0] == '/' || name[0] == '.')

printf("%d %s\n", res, name);

else

printf("%d ./%s\n", res, name);

} //디렉토리의 경우 재귀로 내부의 파일들의 합을 구함

return;

}

- 명령어로 size가 입력되었을 때, 호출하는 함수이다.

- 명령어를 보고 파일인지 디렉토리인지 판단한 후,

파일일 경우 바로 출력하고, 디렉토리의 경우 재귀함수를 호출한다.

**- int dsize\_check(char name[])**

int dsize\_check(char name[]) { //디렉토리의 사이즈를 구하는 함수

int res = 0;

struct dirent \*\*namelist;

int count;

struct stat statbuf;

char buf[BUF\_LEN];

memset(buf, 0, sizeof(buf));

getcwd(buf, BUF\_LEN);

if((count = scandir(name, &namelist, NULL, alphasort)) == -1) {

fprintf(stderr, "%s directory scan error in dsize\_check\n", name);

return 0;

}

chdir(name);

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

if((!strcmp(namelist[i]->d\_name, ".")) || (!strcmp(namelist[i]->d\_name, "..")))

continue;

if(lstat(namelist[i]->d\_name, &statbuf) < 0) {

fprintf(stderr, "stat error2\n");

return 0;

}

if(!S\_ISDIR(statbuf.st\_mode)) {

res += statbuf.st\_size;

}

else

res += dsize\_check(namelist[i]->d\_name);

} //디렉토리 내부의 파일의 크기를 더하고, 디렉토리일 경우 재귀적으로 사이즈를 구해서 리턴함

chdir(buf);

for(int i = 0; i <count; i++)

free(namelist[i]);

free(namelist);

return res;

}

- 디렉토리의 크기를 계산하는 함수이다.

- 재귀적으로 디렉토리 내의 모든 파일과, 내부 디렉토리의 내부 파일까지의 크기를 더해준다.

**- void tree\_making(char path[])**

void tree\_making(char path[]) { //tree를 그리는 함수

struct dirent \*\*namelist;

int count;

int idx;

struct stat statbuf;

if((count = scandir(path, &namelist, dir\_skip, alphasort)) == -1) {

fprintf(stderr, "%s directory scan error in tree\_making\n", path);

return;

}

if(depth == 0)

printf("%s\n", path+2);

chdir(path);

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

struct stat statbuf;

if((!strcmp(namelist[i]->d\_name, ".")) || (!strcmp(namelist[i]->d\_name, "..")))

continue;

if((lstat(namelist[i]->d\_name, &statbuf)) < 0) {

fprintf(stderr, "lstat error\n");

return;

}

printf("┃\n");

printf("┃");

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

printf(" ");

if(i < depth-1)

printf("┃");

}

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

printf("┃\n");

}

if(depth != 0)

printf("┃");

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

printf(" ");

if(i < depth-1)

printf("┃");

}

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

printf("━━━━");

}

printf(" %s\n", namelist[i]->d\_name);

if((statbuf.st\_mode & S\_IFDIR) == S\_IFDIR) {

depth++;

tree\_making(namelist[i]->d\_name);

depth--;

} //파일일 경우 출력, 디렉토리일 경우 출력 후 재귀적이르도 디렉토리 내부 검색

}

chdir("..");

for(int i = 0; i <count; i++)

free(namelist[i]);

free(namelist);

return;

}

- tree 명령어가 입력되었을 때, tree를 그려내는 함수이다.

- 내부에 디렉토리가 있는 경우 디렉토리를 바탕으로 재귀적으로 함수를 호출하여 트리를 그린다.

**-void print\_usage(void)**

void print\_usage(void) //사용법 출력

{

printf("Usage : ssu\_mntr\n");

printf(" delete <FILENAME> <END\_TIME> <OPTION>\n");

printf(" option :\n");

printf(" -i delete without moving file & info to 'trash'\n");

printf(" -r At end time, re-check about deleting file\n");

printf(" size <FILENAME> <OPTION>\n");

printf(" option : \n");

printf(" -d <NUMBER> print <NUMBER> level(s) of sub directory\n");

printf(" recover <FILENAME> <OPTION>\n");

printf(" option :\n");

printf(" -l print file & info in 'trash' with time sequence before recover\n");

printf(" tree show 'check' directory structure with tree shape\n");

printf(" exit program exit & kill monitering background program\n");

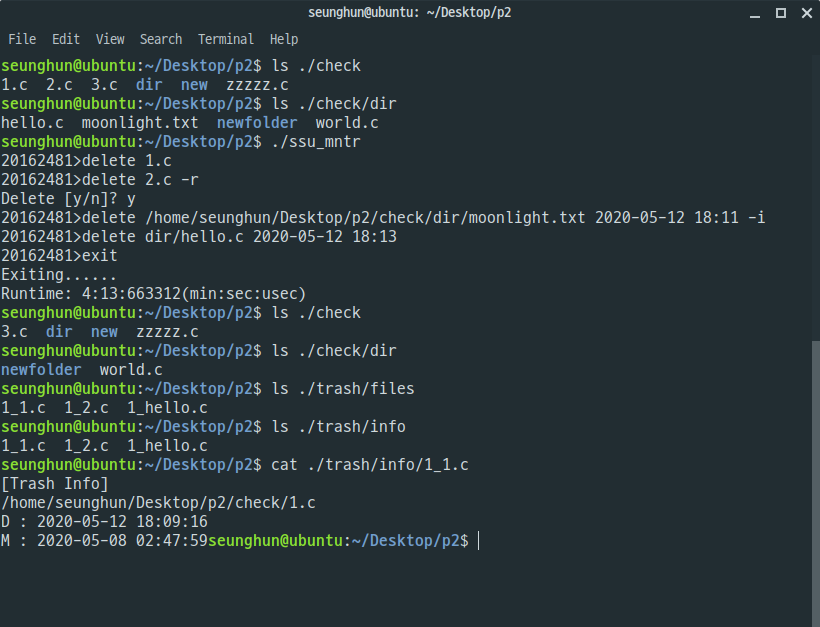
printf(" help print usage\n");

}

- 프로그램의 사용법을 출력하는 함수이다.

4. 테스트 및 결과

- delete 작동 테스트

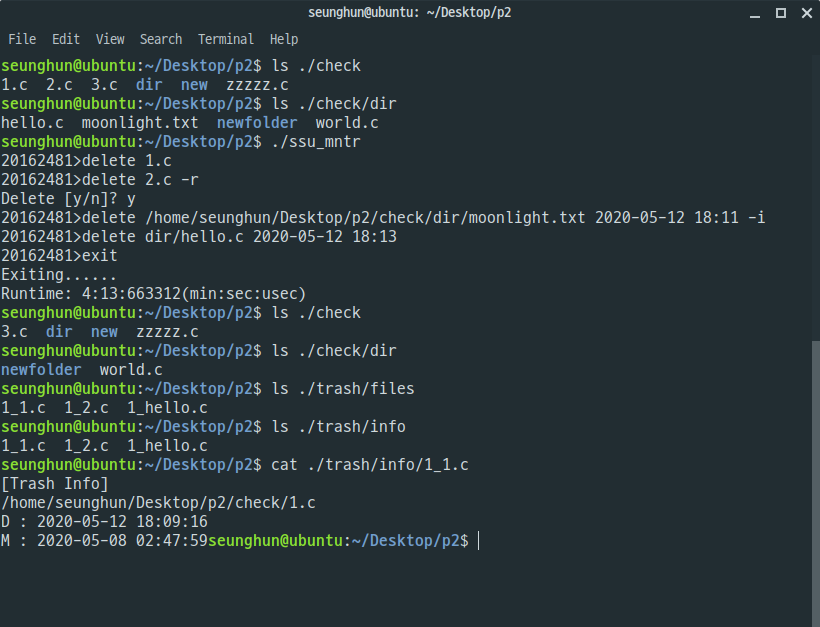


- ls 명령어를 통해 삭제를 진행하기 전의 check 폴더의 내용물을 표시하였다.

- 프로그램 상에서 프롬프트를 이용하여 시간의 유무, 옵션의 유무를 작동시켜 보았다.

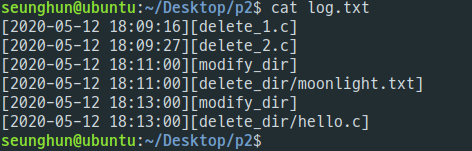
- 이후 다시 ls를 통하며 check 디렉토리와 trash 디렉토리를 확인하였다.

- 상대경로, 절대경로 모두 작동하는 것을 확인할 수 있다.

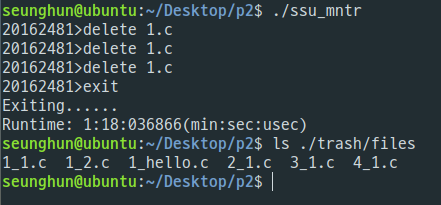


- 앞에서 삭제한 1.c, 2.c, hello.c는 trash 디렉토리로 이동되었고, info파일도 생성되었고, 내용도 작성되어 있는 것을 볼 수 있다.

- i옵션을 적용한 hello.c 파일은 trash 디렉토리로 옮겨지지 않고 바로 삭제되었음을 확인할 수 있다.

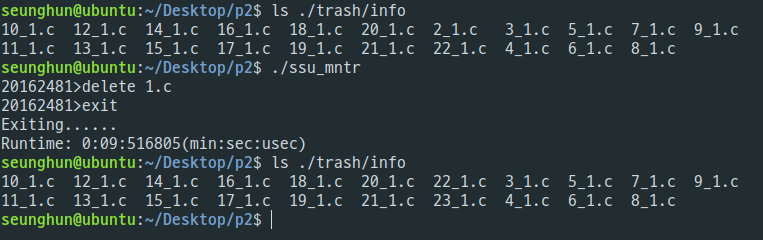


- log.txt를 확인하여 보면, 삭제에 관한 것이 작성되어있는 것을 볼 수 있다.



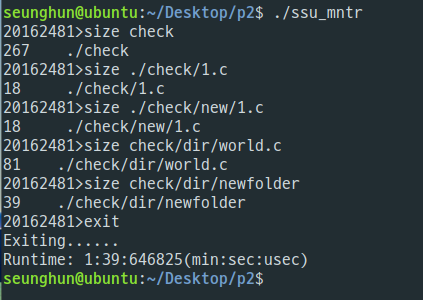
- 같은 이름의 파일을 여러 번 다른 터미널을 통해 생성한 후 delete 명령 하기를 반복하였다.

- 같은 이름의 파일이 늘어날 때 마다 원래의 파일이름 앞에 붙는 ‘숫자\_’의 숫자가 점점 증가한다.



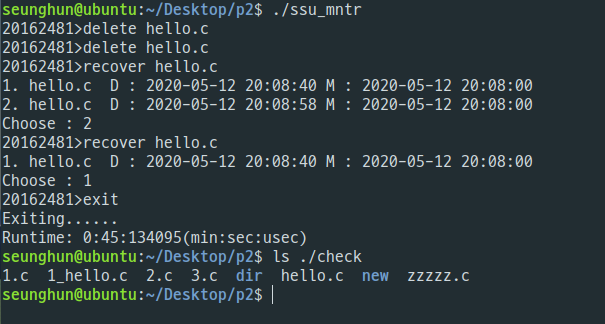
- info 디렉토리가 2KB를 넘기 직전까지 삭제를 진행하여, 2\_1.c 부터 순차적으로 22\_1.c까지 생성하였다.  
 - 이후, 추가적으로 1개를 더 삭제를 진행한 결과, 23\_1.c이 생성되었고, 2KB가 넘게 되자, 가장 오래된 파일인 2\_1.c가 삭제됨으로서 info 디렉토리의 크기를 2KB 이하로 유지 시킴을 확인할 수 있다.

- size 작동 테스트



- size 이후에, 상대경로로 입력한 디렉토리와 파일의 사이즈를 출력하는 것을 볼 수 있다.

- recover 작동 테스트



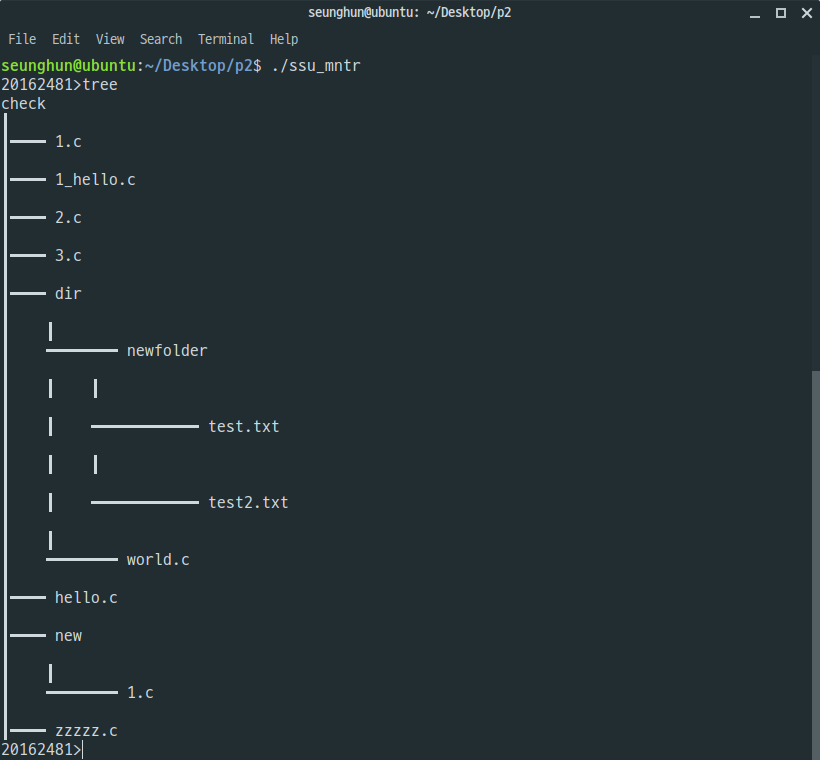
- 이름이 같은 파일을 2번 생성하여 delete 명령어를 통해 trash 디렉토리로 이동하였다.

- 이후, recover를 통해 복구를 진행할 때, 같은 이름의 파일이 2개 있으므로 사용자에게 2개를 출력한다.

- 사용자가 선택한 파일을 복구한다.

- 만약 복구를 진행할 때, 기존의 위치에 이름이 같은 파일이 이미 존재한다면, 파일 이름 앞에 ‘숫자\_’를 붙인 다. 따라서, hello.c 와 1\_hello.c 가 존재하는 것을 볼 수 있다.

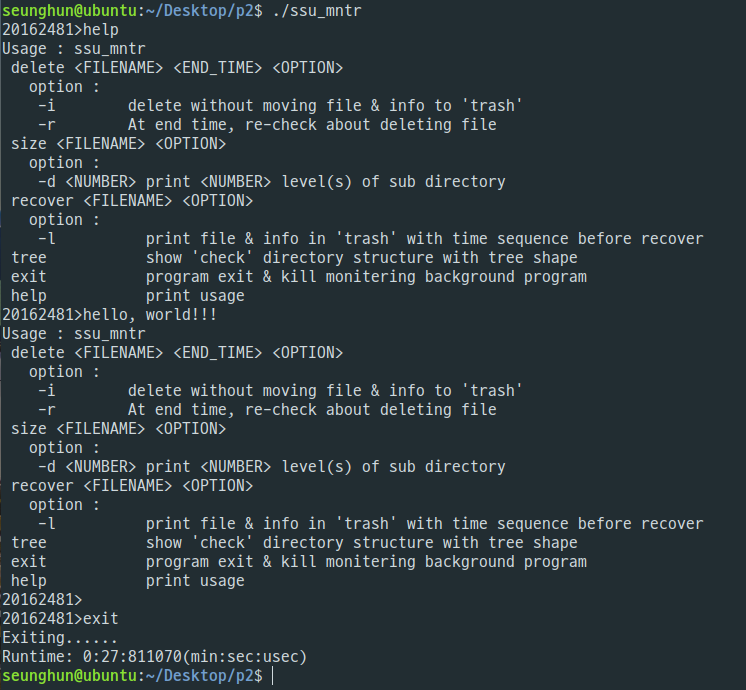
-tree 실행 테스트



- tree 명령어를 실행한 화면이다.

- check 디렉토리 내의 파일과 디렉토리를 순차적으로 보여준다.

- help, exit 및 프롬프트 실행 테스트

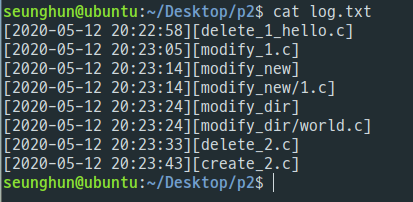
 - help 명령어를 입력한 경우 사용법을 출력한다.

- 명령어가 아닌 것을 입력하는 경우 사용법을 출력한다.

- 엔터(개행문자)를 입력하는 경우, 프롬프트를 재출력한다.

- exit를 실행하면 종료 문구 출력후 프로그램을 종료한다.

- 디몬 프로세스의 log.txt 작성 테스트



- check 디렉토리의 파일이 수정, 생성 삭제가 될 경우, log.txt에 기록되는 것을 볼 수 있다.

5. 소스코드

**<ssu\_daemon.c>**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <signal.h>

#include <syslog.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <limits.h>

#include <string.h>

#include <dirent.h>

#include <time.h>

int mydaemon\_init(void); //디몬프로세스를 생성하는 함수

void dir\_read(char path[]); //초기에 디렉토리를 스캔하는 함수

void dir\_read2(char path[]); //while문 내부에서 반복해서 디렉토리를 스캔하는 함수

void find\_mc(void); //modify와 create를 탐색하는 함수

void find\_delete(void); //delete를 탐색하는 함수

void print\_day(struct tm \*t, char buf[]); //날짜를 출력하기 위한 함수

int dir\_check(const struct dirent \*info); //scandir을 위한 필터함수

typedef struct File\_Stat {

char fpath[1024];

time\_t c\_time;

} FS;

FS fsarr[1024]; //초기에 스캔한 것

FS checkarr[1024]; //반복해서 스캔해서 처음꺼랑 비교하기 위함

int fs\_size = 0; //초기의 스캔한거 갯수

int check\_size = 0; //나중에 스캔한거 갯수

FILE \*logfp;

void make\_daemon(void) {

pid\_t pid;

if((pid = fork()) < 0) {

fprintf(stderr, "fork error\n");

exit(1);

}

else if(pid == 0)

mydaemon\_init();

return;

}

int mydaemon\_init(void) {

pid\_t pid;

int fd, maxfd, fd2;

char curpath[1024] = {0, };

char tmppath[1024] = {0, };

char buf[1024] = {0, };

if((pid = fork()) < 0) {

fprintf(stderr, "fork error\n");

exit(1);

}

else if (pid != 0)

exit(0);

setsid();

signal(SIGTTIN, SIG\_IGN);

signal(SIGTTOU, SIG\_IGN);

signal(SIGTSTP, SIG\_IGN);

maxfd = getdtablesize();

for(fd = 0; fd < maxfd; fd++)

close(fd);

umask(0);

getcwd(curpath, 1024); //현재 작업디렉토리 저장

strcpy(tmppath, curpath); //임시로 현재 디렉토리 추가로 저장

strcat(curpath, "/log.txt");

strcat(tmppath, "/check");

chdir("/");

fd = open("/dev/null", O\_RDWR);

dup(0);

dup(0);

if((logfp = fopen(curpath, "a+")) < 0) {

fprintf(stderr, "fopen error\n");

return 0;

} //curpath는 ./log.txt의 절대경로

setbuf(logfp, NULL); //버퍼를 없앰

chdir(tmppath); // check 폴더로 이동

dir\_read(tmppath); //최초 디렉토리 스캔

while(1) {

check\_size = 0;

dir\_read2(tmppath); //비교를 위해 디렉토리를 읽어옴

find\_mc(); //modify create 확인

find\_delete(); //delete 확인

sleep(1); //1초 멈춤

}

return 0;

}

void find\_delete(void) {

struct tm \*t;

time\_t now;

char buf[512];

int idx = 0;

for(int i = 0; i < fs\_size; i++) {

idx = 0;

for(int j = 0; j < check\_size; j++) {

if(!(strcmp(fsarr[i].fpath, checkarr[j].fpath)))

idx = 1; //이름이 일치하는 파일이 있는 경우 idx = 1

}

if(idx == 0) { //이름이 일치하는 경우가 없는 경우

now = time(NULL);

t = localtime(&now);

print\_day(t, buf);

fprintf(logfp, "%s", buf);

fprintf(logfp, "[delete\_");

fprintf(logfp, "%s", strstr(fsarr[i].fpath, "check/")+6);

fprintf(logfp, "]\n");

fsarr[i] = fsarr[fs\_size-1];

strcpy(fsarr[fs\_size - 1].fpath, "\0");

fsarr[fs\_size - 1].c\_time = 0;

fs\_size--;

}

} //기존목록에는 있지만 새목록에는 없는 경우 삭제되었다고 판단함

return;

}

void find\_mc(void) //modify와 create를 판단하는 함수

{

struct tm \*t;

time\_t cur;

char buf[512];

for(int i = 0; i < check\_size; i++) {

int cnt = 0;

for(int j = 0; j < fs\_size; j++) {

if(!strcmp(checkarr[i].fpath, fsarr[j].fpath)) {

cnt= 1;

if(checkarr[i].c\_time != fsarr[j].c\_time) {

t = localtime(&checkarr[i].c\_time);

print\_day(t, buf);

fprintf(logfp, "%s", buf);

fprintf(logfp, "[modify\_");

fprintf(logfp, "%s", strstr(checkarr[i].fpath, "check/")+6);

fprintf(logfp, "]\n");

fsarr[j].c\_time = checkarr[i].c\_time;

break;

}

else

continue;

} //둘을 비교했을때 이름이 같고, 시간이 다르면 modify, 구 목록 업데이트 해줌

}

if(!cnt) {

strcpy(fsarr[fs\_size].fpath, checkarr[i].fpath);

fsarr[fs\_size].c\_time = checkarr[i].c\_time;

fs\_size++;

cur = time(NULL);

t = localtime(&cur);

print\_day(t, buf);

fprintf(logfp, "%s", buf);

fprintf(logfp, "[create\_");

fprintf(logfp, "%s", strstr(checkarr[i].fpath, "check/")+6);

fprintf(logfp, "]\n");

} //이름이 같은 적이 한번도 없었다면 새로 생긴 파일로 판단함

}

return;

}

void print\_day(struct tm \*t, char \*buf)

{

sprintf(buf, "[%4d-%02d-%02d %02d:%02d:%02d]",

1900+t->tm\_year, t->tm\_mon+1, t->tm\_mday, t->tm\_hour, t->tm\_min, t->tm\_sec);

return;

}

int dir\_check(const struct dirent \*info) {

if(!strncmp(info->d\_name, ".", 1))

return 0;

else

return 1;

}

void dir\_read2(char path[]) { //새 목록을 생성하는 함수

struct dirent \*\*namelist;

int count;

char buf[1024] = {"./"};

char buf2[1024];

char \*ptr;

memset(buf2, 0, sizeof(buf2));

if((count = scandir(path, &namelist, dir\_check, alphasort)) == -1) {

return;

}

chdir(path);

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

struct stat statbuf;

strcpy(buf, "./");

memset(buf2, 0, sizeof(buf2));

if(!(strcmp(namelist[i]->d\_name, ".")) || !(strcmp(namelist[i]->d\_name, "..")))

continue;

stat(namelist[i]->d\_name, &statbuf);

realpath(namelist[i]->d\_name, buf2);

ptr = strstr(buf2, "check");

strcat(buf, ptr);

strcpy(checkarr[check\_size].fpath, buf);

checkarr[check\_size].c\_time = statbuf.st\_mtime;

check\_size++;

if(S\_ISDIR(statbuf.st\_mode))

dir\_read2(namelist[i]->d\_name);

} //파일의 경로를 절대경로로 만들어 준 후, check 이후면 저장해서 상대경로로 구조체에 저장

chdir("..");

return;

}

void dir\_read(char path[]) { // 초기 파일 목록을 생성하는 함수

struct dirent \*\*namelist;

int count;

char buf[1024] = {"./\0"};

char buf2[1024];

char \*ptr;

memset(buf2, 0, sizeof(buf2));

if((count = scandir(path, &namelist, dir\_check, alphasort)) == -1) {

return;

}

chdir(path);

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

struct stat statbuf;

strcpy(buf, "./\0");

memset(buf2, 0, sizeof(buf2));

if(!(strcmp(namelist[i]->d\_name, ".")) || !(strcmp(namelist[i]->d\_name, "..")))

continue;

stat(namelist[i]->d\_name, &statbuf);

realpath(namelist[i]->d\_name, buf2);

ptr = strstr(buf2, "check");

strcat(buf, ptr);

strcpy(fsarr[fs\_size].fpath, buf);

fsarr[fs\_size].c\_time = statbuf.st\_mtime;

fs\_size++;

if(S\_ISDIR(statbuf.st\_mode))

dir\_read(namelist[i]->d\_name);

} //파일의 경로를 절대경로로 만들어 준 후, check 이후면 저장해서 상대경로로 구조체에 저장

chdir("..");

return;

}

<ssu\_mntr.c>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <stdbool.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <signal.h>

#include <syslog.h>

#include <dirent.h>

#include <time.h>

#include <errno.h>

#include <sys/time.h>

#include <termios.h>

#define SECOND\_TO\_MICRO 1000000 // 1초는 백만마이크로초

#define BUF\_LEN 200

void ssu\_runtime(struct timeval \*begin\_t, struct timeval \*end\_t);

//프로그램 실행 시간 측정함수

void print\_usage(void); //도움말을 출력하는 함수

void make\_daemon(void); //디몬 프로세스를 호출하기 위한 함수 -> ssu\_daemon.c에 존재

void tree\_making(char path[]); //트리를 그리기 위한 함수

void size\_check(char input[]); //size를 체크하는 함수

int dsize\_check(char name[]); //디렉토리의 size를 체크하기 위한 함수

void f\_delete(char input[]); //delete 명령어를 처리하는 함수

void delete\_timer(int signo); //delete를 위한 alarm을 받는 핸들러 함수

int is\_delete(void); //delete에 -r 옵션일때 정말로 삭제할 것인지를 묻는 함수

void make\_delete(void); // delete시에 파일 이름을 겹치지 않게 바꿔서 delete 하는 함수

int same\_name\_max(char fname[]); //원 이름이 같은 파일의 갯수를 세는 함수

int dir\_skip(const struct dirent \*info); //scandir의 filter 함수

void make\_info(char fname[]); //delete 시에 info 파일을 만드는 함수

void find\_2kb(void); //info 폴더가 2KB를 넘는지 확인하는 함수

void find\_oldest(void); //info 폴더에서 가장 오래된 파일을 찾아 삭제하는 함수

void f\_recover(char input[]); //recover 명령어를 진행하는 함수

int same\_name\_max2(char fname[], char last\_locdir[]);

//recover 시에 복원될 폴더에 같은 이름을 가진 파일이 있는지 확인하는 함수

char \*prompt[] = {"delete", "size", "recover", "tree", "exit", "help"};

bool check[6] = {0, 0, 0, 0, 0, 0};

int checksum = 0;

int daemon\_pid;

char fname\_delete[BUF\_LEN];

char option\_delete[3];

char current\_wd[BUF\_LEN];

int main(void) {

struct timeval begin\_t, end\_t;

gettimeofday(&begin\_t, NULL);

char input[BUF\_LEN]; //입력받은 문장 전체

char const\_input[BUF\_LEN];

char todo[10]; //명령어 분리

memset(input, 0, sizeof(input));

//디몬 부르기

make\_daemon();

//trash 폴더가 없는 경우 만들기

if(access("./trash", F\_OK) == -1)

mkdir("./trash", 0777);

if(access("./trash/files", F\_OK) == -1)

mkdir("./trash/files", 0777);

if(access("./trash/info", F\_OK) == -1)

mkdir("./trash/info", 0777);

getcwd(current\_wd, BUF\_LEN); //현재 작업디렉토리 저장

while(1) {

chdir(current\_wd);

memset(input, 0, sizeof(input));

memset(const\_input, 0, sizeof(const\_input));

printf("20162481>");

fgets(input, BUF\_LEN, stdin);

strcpy(const\_input, input);

memset(check, 0, sizeof(check)); //check 배열 초기화

checksum = 0;

strcpy(todo, strtok(input, " "));

if(!strcmp(todo, "\n")) //엔터만 입력되면 프롬프트 재출력

continue;

if(todo[strlen(todo) - 1] == '\n')

todo[strlen(todo) - 1] = '\0'; //뒤에 있는 개행 문자 제거

for(int i = 0; i < 6; i++) {

if(!strcmp(todo, prompt[i])) {

check[i] = true;

checksum++;

}

} //입력된 명령어와 지정된 명령어를 비교함

if(checksum == 0) { //명령어가 올바르지 않은 경우 도움말 출력

print\_usage();

continue;

}

if(check[0] == true) //delete일때

f\_delete(const\_input);

else if (check[1] == true) {//size 일때

size\_check(const\_input);

}

else if (check[2] == true) //recover일때

f\_recover(const\_input);

else if (check[3] == true) //tree 일때

tree\_making("./check");

else if (check[4] == true) { //exit 일때

//int id = getpid();

//kill(id+2, SIGKILL);

printf("Exiting......\n");

gettimeofday(&end\_t, NULL);

ssu\_runtime(&begin\_t, &end\_t);

exit(0);

}

else if (check[5] == true) //help일때

print\_usage();

}

return 0;

}

void ssu\_runtime(struct timeval \*begin\_t, struct timeval \*end\_t)

{

end\_t->tv\_sec -= begin\_t->tv\_sec; //끝난 시간에서 처음시간을 뺌

if(end\_t->tv\_usec < begin\_t->tv\_usec){

end\_t->tv\_sec--;

end\_t->tv\_usec += SECOND\_TO\_MICRO;

} //나중의 us값이 작을경우 1초를 뺴서 us값이 더해줌

end\_t->tv\_usec -= begin\_t->tv\_usec;

printf("Runtime: %ld:%02ld:%06ld(min:sec:usec)\n",

end\_t->tv\_sec / 60, end\_t->tv\_sec % 60, end\_t->tv\_usec);

} //프로그램 수행시간을 분, 초, 마이크로초로 출력

//cwd = check 바깥

void f\_recover(char input[]) { //recover하는 함수

char info\_path[BUF\_LEN] = "./trash/info/";

char file\_path[BUF\_LEN] = "./trash/files/";

char fname[BUF\_LEN]; // 원래의 이름

char dest[BUF\_LEN];

char \*ptr = input;

int count;

struct dirent \*\*namelist1;

struct dirent \*\*namelist2;

char f\_name[BUF\_LEN]; //현재 trash 폴더 안의 이름

char buf[BUF\_LEN];

char buf\_D[BUF\_LEN];

char buf\_M[BUF\_LEN];

char last\_locdir[BUF\_LEN];

int pfx, cnt = 0;

int choose, check;

int idc = 0, frontnum;

char tmp[BUF\_LEN];

FILE \*fp;

memset(fname, 0, sizeof(fname));

memset(dest, 0, sizeof(dest));

memset(f\_name, 0, sizeof(f\_name));

memset(buf, 0, sizeof(buf));

memset(buf\_D, 0, sizeof(buf\_D));

memset(buf\_M, 0, sizeof(buf\_M));

memset(last\_locdir, 0, sizeof(last\_locdir));

strcpy(fname, ptr+8); //명령어를 제외하고 파일 이름만 받아들임

//printf("%s\n", fname);

if(fname[strlen(fname) - 1] == '\n')

fname[strlen(fname) - 1] = '\0';

if((count = scandir("./trash/info", &namelist1, dir\_skip, alphasort)) == -1) {

fprintf(stderr, "scandir error in f\_recover function\n");

return;

}

check = 0;

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

strcpy(buf, namelist1[i]->d\_name);

while(1) {

if(buf[cnt] == '\_')

break;

else

cnt++;

}

ptr = buf;

strcpy(f\_name, ptr + cnt + 1);

if(!strcmp(fname, f\_name)) {

check = 1;

memset(buf, 0, sizeof(buf));

strcpy(buf, info\_path);

strcat(buf, namelist1[i]->d\_name);

if((fp = fopen(buf, "r")) < 0) {

fprintf(stderr, "fopen error in recover function\n");

return;

}

memset(buf, 0, sizeof(buf));

fgets(buf, sizeof(buf), fp); //trash info 읽음

memset(buf, 0, sizeof(buf));

fgets(buf, sizeof(buf), fp); //원래 파일 경로 읽음

memset(buf, 0, sizeof(buf));

fgets(buf\_D, sizeof(buf\_D), fp); //삭제 시간 얻어옴

fgets(buf\_M, sizeof(buf\_M), fp); // 최종 수정 시간 얻어옴

if(buf\_D[strlen(buf\_D) - 1] == '\n')

buf\_D[strlen(buf\_D) - 1] = '\0';

if(buf\_M[strlen(buf\_M) - 1] == '\n')

buf\_M[strlen(buf\_M) - 1] = '\0';

//D M 출력 준비

printf("%d. %s %s %s\n", atoi(namelist1[i]->d\_name), f\_name, buf\_D, buf\_M);

fclose(fp);

} //사용자가 입력한 파일이름과 이름이 같은 파일을 모두 출력해줌

}

if(check == 0) {

printf("There is no '%s' in 'trash' directory!\n", fname);

return;

} //이름이 같은 파일이 없는 경우 에러 출력 후 리턴

printf("Choose : ");

scanf("%d", &choose);

getchar();

//사용자가 어떤 파일을 복구 할 것인지 확인함

//숫자 + \_ + 파일 이름 해서 info 파일 열어서 파일 원래 위치 확인

// 그 디렉토리 읽어서 같은 이름 파일 있는 지 확인

//없으면 앞에 숫자 띠고 가는거고 있으면 붙여서 가는거고

memset(f\_name, 0, sizeof(f\_name));

memset(buf, 0, sizeof(buf));

sprintf(f\_name, "%d\_%s", choose, fname); //읽을 파일 확정

strcpy(buf, info\_path);

strcat(buf, f\_name); //info 파일의 위치 만듬

if((fp = fopen(buf, "r")) < 0) {

fprintf(stderr, "fopen error in recover function 2\n");

return;

}

memset(buf, 0, sizeof(buf));

fgets(buf, sizeof(buf), fp); //trash info 읽어서 버림

memset(buf, 0, sizeof(buf));

fgets(buf, sizeof(buf), fp); //파일의 경로 읽어옴

if(buf[strlen(buf) - 1] == '\n')

buf[strlen(buf) - 1] = '\0';

cnt = 0;

for(int i = 0; i < strlen(buf); i++) {

if(buf[i] == '/')

cnt = i;

}

strncpy(last\_locdir, buf, cnt); //원래 파일이 있던 위치를 탐색

if((count = scandir(last\_locdir, &namelist2, dir\_skip, alphasort)) < 0) {

fprintf(stderr, "scandir error in recover function 2\n");

return;

}

ptr = tmp;

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

//memset(tmp, 0, sizeof(tmp));

//strcpy(tmp, namelist2->d\_name);

if(!strncmp(fname, namelist2[i]->d\_name, strlen(namelist2[i]->d\_name))) {

idc = 1;

break;

}

} //같은 이름 파일이 있으면 그만 찾고 break

if(idc == 0) { //이름이 겹치는 파일이 없을떄

strcat(file\_path, f\_name);

rename(file\_path, buf);

strcat(info\_path, f\_name);

remove(info\_path);

return;

}

else { //이름이 겹치는 파일이 있을때

frontnum = same\_name\_max2(fname, last\_locdir); //앞에 붙은 숫자 제일 큰거 구하기

frontnum++;

memset(tmp, 0, sizeof(tmp));

sprintf(tmp, "/%d\_%s", frontnum, fname); //앞에 숫자 붙여서 rename함

strcat(last\_locdir, tmp);

strcat(file\_path, f\_name);

rename(file\_path, last\_locdir);

strcat(info\_path, f\_name);

remove(info\_path); //info파일 지움

return;

}

}

int same\_name\_max2(char fname[], char last\_locdir[]) { //과거의 폴더에서 같은 이름 파일의 숫자의 최댓값을 계산해주는 함수

struct dirent \*\*namelist;

int count, loc = 0;

char tmp[BUF\_LEN];

int maxnum = 0;

int num = 0;

char \*ptr = tmp;

if((count = scandir(last\_locdir, &namelist, dir\_skip, alphasort)) == -1) {

fprintf(stderr, "last directory scan error in same\_name\_max2\n");

return count;

}

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

ptr = tmp;

memset(tmp, 0, sizeof(tmp));

strcpy(tmp, namelist[i]->d\_name);

for(int j = 0; j < strlen(tmp); j++) {

if(tmp[j] == '\_') {

loc = j;

break;

} // 언더바의 위치를 찾아서 그 앞의 숫자를 제거할 수 있도록 함

}

if(loc != 0)

ptr = ptr + loc + 1;

//이제 파일 이름만 남게 됨

if(!strcmp(fname, ptr)) {

num = atoi(namelist[i]->d\_name); //1\_1.c라면 num은 1이 될꺼임

if(maxnum < num)

maxnum = num; //앞에 붙은 값의 최대를 구함

}

}

for(int i = 0; i <count; i++)

free(namelist[i]);

free(namelist);

return maxnum; //같은 이름을 가진 파일이 없다면 0을 리턴함

}

void f\_delete(char input[]) {

char backup\_input[BUF\_LEN] = {0, };

char fname[BUF\_LEN] = {0, }; //파일 이름

char \*tmp;

char rpath[BUF\_LEN] = {0, }; // 파일 절대경로

char d\_day[BUF\_LEN] = {0, }; // 삭제 날짜

char d\_time[BUF\_LEN] = {0, }; //삭제 시간

struct tm t;

time\_t deltime, curtime;

int timegap = 0;

char option[3]; //delete 할때의 옵션

struct stat statbuf;

struct stat stattmp;

int t\_checksum = 0;

memset(backup\_input, 0, sizeof(backup\_input));

memset(fname, 0, sizeof(fname));

memset(rpath, 0, sizeof(rpath));

memset(d\_day, 0, sizeof(d\_day));

memset(d\_time, 0, sizeof(d\_time));

memset(option, 0, sizeof(option));

signal(SIGALRM, delete\_timer);

input = input+7;

strcpy(backup\_input, input); //명령어 잘라낸 인풋 배열 저장해둠

chdir("./check");

tmp = strtok(input, " ");

if(tmp != NULL)

strcpy(fname, tmp);

if(fname[strlen(fname) - 1] == '\n') // 끝에 개행문자까지 포함된 경우 개행문자 삭제

fname[strlen(fname) - 1] = '\0';

realpath(fname, rpath);

if(lstat(rpath, &statbuf) < 0) { //파일 명이 이상한거 일단 다 걸러짐

fprintf(stderr, "File is not exist!!\n");

chdir(current\_wd);

return;

} // 존재하지 않는 파일을 삭제하려 하는 경우 에러 출력 후 리턴

//printf("hi : %s\n", rpath);

//이 아래부터 파일 명은 멀쩡하다

if(strlen(backup\_input) == strlen(fname) + 1){

strcpy(fname\_delete, rpath);

memset(option\_delete, 0, sizeof(option\_delete));

chdir(current\_wd);

delete\_timer(SIGALRM);

return;

}

else if((strlen(backup\_input) > strlen(fname) + 1) && backup\_input[strlen(fname) + 1] == '-') {

tmp = strtok(NULL, " "); //-가 나온것은 시간이 입력되지 않았다는거

strcpy(option, tmp);

//printf("%s\n", option);

strcpy(option\_delete, option);

strcpy(fname\_delete, rpath);

chdir(current\_wd);

delete\_timer(SIGALRM);

return;

} //시간이 있을때

else if((strlen(backup\_input) > strlen(fname) + 1) && backup\_input[strlen(fname) + 1] >= '0'

&& backup\_input[strlen(fname) + 1] <= '9'){

tmp = strtok(NULL, "-"); //년도 커팅

t.tm\_year = atoi(tmp) - 1900;

if(t.tm\_year < 0)

t\_checksum++;

tmp = strtok(NULL, "-"); //월 커팅

t.tm\_mon = atoi(tmp) - 1;

if(t.tm\_mon < 0)

t\_checksum++;

tmp = strtok(NULL, " "); //일 커팅

t.tm\_mday = atoi(tmp);

if(t.tm\_mday < 1)

t\_checksum++;

tmp = strtok(NULL, ":");

t.tm\_hour = atoi(tmp);

if(t.tm\_hour < 0)

t\_checksum++;

tmp = strtok(NULL, " ");

t.tm\_min = atoi(tmp);

if(t.tm\_min < 0)

t\_checksum++;

t.tm\_sec = 0;

if(t\_checksum > 0) {

fprintf(stderr, "Time input error!!\n");

chdir(current\_wd);

return;

}

if((tmp = strtok(NULL, " ")) != NULL) {

strcpy(option, tmp);

strcpy(option\_delete, option);

}

deltime = mktime(&t);

curtime = time(NULL);

//printf("%ld %ld\n", deltime, curtime);

//현재시간, 삭제 시간 구함

timegap = deltime - curtime;

if(timegap < 0) {

fprintf(stderr, "Delete time error!!\n");

chdir(current\_wd);

return;

}

memset(fname\_delete, 0, sizeof(fname\_delete));

memset(option\_delete, 0, sizeof(option\_delete));

strcpy(fname\_delete, rpath); //시그널 핸들러 함수에서 사용하기 위해 전역변수로 넘겨줌

strcpy(option\_delete, option);

alarm(timegap);

chdir(current\_wd);

return;

}

}

void delete\_timer(int signo) { //타이머가 울리면 파일을 삭제 하는 함수

char rpath[BUF\_LEN] = {0,};

char fname[BUF\_LEN] = {0,};

char \*ptr = fname\_delete;

int option\_i = 0;

int option\_r = 0;

int loc;

memset(rpath, 0, sizeof(rpath));

memset(fname, 0, sizeof(fname));

strcpy(rpath, fname\_delete);

for(int i = 0; i < strlen(fname\_delete); i++)

if(fname\_delete[i] == '/')

loc = i;

strcpy(fname, ptr + loc + 1);

if(!strncmp(option\_delete, "-i", 2))

option\_i = 1;

else if(!strncmp(option\_delete, "-r", 2)) {

option\_r = 1;

}

if(option\_r) {

//삭제할껀지 묻는 함수

//Y는 삭제, N은 취소하고 리턴

if(is\_delete() < 0) //사용자가 n 선택했으면 취소

return;

}

if(option\_i) {

//바로 삭제

struct stat statbuf;

lstat(rpath, &statbuf);

if(!S\_ISDIR(statbuf.st\_mode)) {

remove(rpath);

return;

}

else {

printf("Failed to delete directory : '%s' isn't empty!\n", fname);

return;

}

}

else {

make\_delete();

}

return;

}

int dir\_skip(const struct dirent \*info) {

if(!strcmp(info->d\_name, ".") || !strcmp(info->d\_name, ".."))

return 0;

else return 1;

} //scandir에서 현재 디렉토리와 상위 디럭토리를 제외하기 위한 함수

int same\_name\_max(char fname[]) { //trash 폴더에서 같은 이름 파일의 숫자의 최댓값을 계산해주는 함수

struct dirent \*\*namelist;

int count, loc = 0;

char tmp[BUF\_LEN];

int maxnum = 0;

int num = 0;

char \*ptr = tmp;

if((count = scandir("./trash/files", &namelist, dir\_skip, alphasort)) == -1) {

fprintf(stderr, "trash directory scan error in same\_name\_max\n");

return count;

}

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

ptr = tmp;

memset(tmp, 0, sizeof(tmp));

strcpy(tmp, namelist[i]->d\_name);

for(int j = 0; j < strlen(tmp); j++) {

if(tmp[j] == '\_') {

loc = j;

break;

}

}

if(loc != 0)

ptr = ptr + loc + 1;

//이제 파일 이름만 남게 됨

if(!strcmp(fname, ptr)) {

num = atoi(namelist[i]->d\_name); //1\_1.c라면 num은 1이 될꺼임

if(maxnum < num)

maxnum = num; //앞에 붙은 값의 최대를 구함

}

}

for(int i = 0; i <count; i++)

free(namelist[i]);

free(namelist);

return maxnum; //같은 이름을 가진 파일이 없다면 0을 리턴함

}

//현재 작업디렉토리는 check 바깥

void make\_delete(void) {

char fname[BUF\_LEN]; //순수한 파일 이름

char path\_files[BUF\_LEN] = "trash/files/";

char rpath\_files[BUF\_LEN];

int loc;

char newfname[BUF\_LEN];

char newpath[BUF\_LEN];

memset(fname, 0, sizeof(fname));

memset(newfname, 0, sizeof(newfname));

memset(newpath, 0, sizeof(newpath));

memset(rpath\_files, 0, sizeof(rpath\_files));

realpath(path\_files, rpath\_files);

for(int i = 0; i < strlen(fname\_delete); i++)

if(fname\_delete[i] == '/')

loc = i;

//마지막 슬래시의 위치를 가져옴, 그 다음부터 파일 이름이니까

strcpy(fname, fname\_delete+loc+1); //파일 이름만 건져오기

int pfx = same\_name\_max(fname); //0이면 같은 이름을 가진 파일이 없다는것

// 0 이상이라면 같은 이름을 가진 파일이 count개 있다는것

// count 보다 하나 큰 값으로 파일 앞에 붙이기

pfx++; //pfx보다 1 큰값으로 앞에 붙여야됨

//새 파일 이름 만들기

sprintf(newfname, "/%d\_%s", pfx, fname);

strcpy(newpath, rpath\_files);

strcat(newpath, newfname);

make\_info(newfname); // info 파일 만드는 함수 호출

rename(fname\_delete, newpath);

}

void make\_info(char fname[]) {

char file\_info[BUF\_LEN] = "./trash/info/";

FILE \*infp;

struct stat statbuf;

struct tm tM, tD;

time\_t M, D;

strcat(file\_info, fname); //info 파일을 만들 경로 + 파일 이름

if((infp = fopen(file\_info, "w+")) < 0) {

fprintf(stderr, "Fail making info file\n");

return;

}

if(lstat(fname\_delete, &statbuf) < 0) {

fprintf(stderr, "lstat error in make\_info\n");

return;

}

M = statbuf.st\_mtime;

D = time(NULL);

localtime\_r(&M, &tM);

localtime\_r(&D, &tD);

//info 파일에 정보를 작성해줌

fprintf(infp, "[Trash Info]\n");

fprintf(infp, "%s\n", fname\_delete);

fprintf(infp, "D : %d-%02d-%02d %02d:%02d:%02d\n", tD.tm\_year+1900, tD.tm\_mon+1,

tD.tm\_mday, tD.tm\_hour, tD.tm\_min, tD.tm\_sec);

fprintf(infp, "M : %d-%02d-%02d %02d:%02d:%02d\n", tM.tm\_year+1900, tM.tm\_mon+1,

tM.tm\_mday, tM.tm\_hour, tM.tm\_min, tM.tm\_sec);

fclose(infp);

find\_2kb(); //info파일이 2KB가 넘은 경우 가장 오래된 것을 삭제하는 함수

}

void find\_2kb(void) {

char buf[BUF\_LEN];

memset(buf, 0, sizeof(buf));

getcwd(buf, BUF\_LEN);

chdir(current\_wd);

char info\_path[BUF\_LEN] = "./trash/info";

int size;

char fname[BUF\_LEN] = {0,};

while(1) {

size = dsize\_check(info\_path);

if(size < 2048)

break;

else

find\_oldest(); //2KB가 넘었으면, 가장 오래된 파일을 찾아서 삭제함

}

chdir(buf);

return;

}

void find\_oldest(void) { //가장 오래된 info 파일을 찾는 함수

char info\_path[BUF\_LEN] = "./trash/info";

char buf[BUF\_LEN];

char fname\_todelete[BUF\_LEN];

struct dirent \*\*namelist;

char \*ptr;

struct tm t;

time\_t oldest, file\_time;

int count;

FILE \*fp;

char cwd[BUF\_LEN];

char filepath[BUF\_LEN];

char infopath[BUF\_LEN];

memset(filepath, 0, sizeof(filepath));

memset(infopath, 0, sizeof(filepath));

memset(buf, 0, sizeof(buf));

memset(cwd, 0, sizeof(cwd));

memset(fname\_todelete, 0, sizeof(fname\_todelete));

getcwd(cwd, BUF\_LEN);

if((count = scandir(info\_path, &namelist, dir\_skip, NULL)) == -1) {

fprintf(stderr, "scndir error in find\_oldest\n");

return;

}

chdir(info\_path);

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

if((fp = fopen(namelist[i]->d\_name, "r")) < 0) {

fprintf(stderr, "fopen error in find\_oldest\n");

return;

}

//info 파일의 적혀있는 딜리트 타임을 읽어서 비교함

fgets(buf, BUF\_LEN, fp);

memset(buf, 0, sizeof(buf));

fgets(buf, BUF\_LEN, fp);

memset(buf, 0, sizeof(buf));

fgets(buf, BUF\_LEN, fp); //buf에 삭제 시간 읽어옴

ptr = buf;

ptr = ptr+5;

t.tm\_year = atoi(ptr) - 1900;

t.tm\_mon = atoi(ptr+5) - 1;

t.tm\_mday = atoi(ptr+8);

t.tm\_hour = atoi(ptr+11);

t.tm\_min = atoi(ptr+14);

t.tm\_sec = atoi(ptr+17);

file\_time = mktime(&t);

if(i == 0) {

oldest = file\_time;

strcpy(fname\_todelete, namelist[i]->d\_name);

}

else {

if(oldest > file\_time) {

oldest = file\_time;

memset(fname\_todelete, 0, sizeof(fname\_todelete));

strcpy(fname\_todelete, namelist[i]->d\_name);

}

}

}

chdir(cwd);

strcpy(filepath, "./trash/files/");

strcpy(infopath, "./trash/info/");

strcat(filepath, fname\_todelete);

strcat(infopath, fname\_todelete);

remove(filepath);

remove(infopath);

//원 파일과 info 파일 모두를 삭제함

for(int i = 0; i <count; i++)

free(namelist[i]);

free(namelist);

return;

}

int is\_delete(void) { //i옵션일떄 사용자에게 삭제할지를 묻는 함수

char input;

printf("Delete [y/n]? ");

scanf("%c", &input);

getchar();

if(input == 'y')

return 1;

else

return -1;

}

void size\_check(char input[]) { //입력받은 파일 또는 디렉토리의 사이즈를 확인

char name[BUF\_LEN] = {0, };

char rpath[BUF\_LEN] = {0, };

char \*tmp;

struct stat statbuf;

int res = 0;

int maxdepth = 1;

tmp = strtok(input, " "); //명령어 끌어옴

tmp = strtok(NULL, " "); //파일 명을 tmp에 저장함

strcpy(name, tmp);

if(strlen(input) > 5 + strlen(name)) {

tmp = strtok(NULL, " "); //옵션 받아옴

tmp = strtok(NULL, " ");

maxdepth = atoi(tmp);

printf("%d\n", maxdepth);

}

if(name[strlen(name) - 1] == '\n')

name[strlen(name) - 1] = '\0';

if(lstat(name, &statbuf) < 0) {

fprintf(stderr, "stat error in size\_check\n");

return;

}

if(S\_ISREG(statbuf.st\_mode)) {

if(name[0] == '/' || name[0] == '.')

printf("%ld %s\n",statbuf.st\_size, name);

else

printf("%ld ./%s\n",statbuf.st\_size, name);

} //입력된 것이 파일일 경우 사이즈 출력

else if(S\_ISDIR(statbuf.st\_mode)) {

res = dsize\_check(name);

if(name[0] == '/' || name[0] == '.')

printf("%d %s\n", res, name);

else

printf("%d ./%s\n", res, name);

} //디렉토리의 경우 재귀로 내부의 파일들의 합을 구함

return;

}

int dsize\_check(char name[]) { //디렉토리의 사이즈를 구하는 함수

int res = 0;

struct dirent \*\*namelist;

int count;

struct stat statbuf;

char buf[BUF\_LEN];

memset(buf, 0, sizeof(buf));

getcwd(buf, BUF\_LEN);

if((count = scandir(name, &namelist, NULL, alphasort)) == -1) {

fprintf(stderr, "%s directory scan error in dsize\_check\n", name);

return 0;

}

chdir(name);

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

if((!strcmp(namelist[i]->d\_name, ".")) || (!strcmp(namelist[i]->d\_name, "..")))

continue;

if(lstat(namelist[i]->d\_name, &statbuf) < 0) {

fprintf(stderr, "stat error2\n");

return 0;

}

if(!S\_ISDIR(statbuf.st\_mode)) {

res += statbuf.st\_size;

}

else

res += dsize\_check(namelist[i]->d\_name);

} //디렉토리 내부의 파일의 크기를 더하고, 디렉토리일 경우 재귀적으로 사이즈를 구해서 리턴함

chdir(buf);

for(int i = 0; i <count; i++)

free(namelist[i]);

free(namelist);

return res;

}

int depth = 0;

void tree\_making(char path[]) { //tree를 그리는 함수

struct dirent \*\*namelist;

int count;

int idx;

struct stat statbuf;

if((count = scandir(path, &namelist, dir\_skip, alphasort)) == -1) {

fprintf(stderr, "%s directory scan error in tree\_making\n", path);

return;

}

if(depth == 0)

printf("%s\n", path+2);

chdir(path);

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

struct stat statbuf;

if((!strcmp(namelist[i]->d\_name, ".")) || (!strcmp(namelist[i]->d\_name, "..")))

continue;

if((lstat(namelist[i]->d\_name, &statbuf)) < 0) {

fprintf(stderr, "lstat error\n");

return;

}

printf("┃\n");

printf("┃");

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

printf(" ");

if(i < depth-1)

printf("┃");

}

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

printf("┃\n");

}

if(depth != 0)

printf("┃");

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

printf(" ");

if(i < depth-1)

printf("┃");

}

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

printf("━━━━");

}

printf(" %s\n", namelist[i]->d\_name);

if((statbuf.st\_mode & S\_IFDIR) == S\_IFDIR) {

depth++;

tree\_making(namelist[i]->d\_name);

depth--;

} //파일일 경우 출력, 디렉토리일 경우 출력 후 재귀적이르도 디렉토리 내부 검색

}

chdir("..");

for(int i = 0; i <count; i++)

free(namelist[i]);

free(namelist);

return;

}

void print\_usage(void) //사용법 출력

{

printf("Usage : ssu\_mntr\n");

printf(" delete <FILENAME> <END\_TIME> <OPTION>\n");

printf(" option :\n");

printf(" -i delete without moving file & info to 'trash'\n");

printf(" -r At end time, re-check about deleting file\n");

printf(" size <FILENAME> <OPTION>\n");

printf(" option : \n");

printf(" -d <NUMBER> print <NUMBER> level(s) of sub directory\n");

printf(" recover <FILENAME> <OPTION>\n");

printf(" option :\n");

printf(" -l print file & info in 'trash' with time sequence before recover\n");

printf(" tree show 'check' directory structure with tree shape\n");

printf(" exit program exit & kill monitering background program\n");

printf(" help print usage\n");

}