**2022**

**Operating System Project**

< Mini Operating System >

2018112128 박준서

2020112133 송하은

2020112087 우신영

2018112183 정재우

2018112185 한동원



**목차**

**1. 프로젝트 목표 및 방향**

1. 추진목표
2. 연구의 목적
3. 연구과제의 필요성
4. 관련연구
5. **LCRS 구조**
6. **멀티스레딩 구현**
7. **Mini Operating System 구현**
8. 필수 구현 명령어 : cd, mkdir, pwd, ls, rm, cat, cp, find
9. 추가 명령어
10. **자원 소요 계획**

(1) 팀원 역할 소개 및 참여기여도

(2) 프로젝트 추진 단계

(3) 전체 추진 일정

(4) 회의일지

**6. 참고문헌**

1. **프로젝트 목표 및 방향**
2. 추진목표

리눅스의 기본 동작 구조와 체계를 이해하고 자료구조 알고리즘을 바탕으로 C언어를 통해 리눅스와 동일하게 작동하는 파일 탐색기 구조를 구현한다. 명령어의 역할을 이해하고 동작하는 방식을 구현하면서 리눅스의 파일, 디렉토리, 프로세스, 쓰레드에 대한 이해를 높이고 새로운 OS의 조작방법을 익힌다. 또한 AWS 환경을 사용함으로써 클라우드의 동작 구조와 체계를 배우고 동일하게 작동 가능한 시스템을 구현한다.

1. 연구의 목적

이번 프로젝트의 목적은 리눅스 시스템의 이해와 응용에 있다. 본 프로젝트는 C언어를 통해서 리눅스 시스템을 구현함에 있다. 리눅스의 기본 동작 구조와 체계를 이해하고 동일하게 작동 가능한 시스템을 구현한다. 또한 AWS환경을 사용하면서 클라우드 기본 동작 구조와 체계를 이해하고 동일하게 작동 가능한 시스템을 구현한다.

1. 연구과제의 필요성

수업에서 배운 이론을 바탕으로 낯선 리눅스 환경에서 익숙한 C 언어를 통해 명령어를 구현하고, 이를 직접 사용해보면서 리눅스의 조작방법을 보다 수월하게 익히도록 한다. 이번 프로젝트를 통해 리눅스 환경에서 직접 운영체제 프로그램을 만들어보고 명령어와 프로세스, 스레드의 이해를 높이는 것에 중점을 둔다. 또한 이론으로 배운 내용을 실습해보면서 설계에 대한 기본 지식을 쌓는다.

1. 관련연구

운영체제에서 가장 중요한 역할을 하는 파일 시스템 관리는 파일을 생성하고, 읽어 들이거나 지우는 등의 역할을 한다. 가장 익숙한 환경인 window와 같이 리눅스에서도 독자적인 파일 시스템을 가지고 있다. 리눅스 커널이 지원해주는 파일 시스템 관리를 위한 함수는 어떤 것인지, 사용하는 표준 입출력 라이브러리들은 어떤 것인지, 어떤 관계를 통해 작동하는 것인지 c 프로그래밍을 통해 어떤 작업들이 일어나는지 등을 조사하고 이를 프로그램으로 구현한다.

1. **LCRS 구조**

|  |  |
| --- | --- |
| A | |
|  |  |

|  |  |
| --- | --- |
| B | |
|  |  |

|  |  |
| --- | --- |
| D | |
|  |  |

|  |  |
| --- | --- |
| C | |
|  |  |

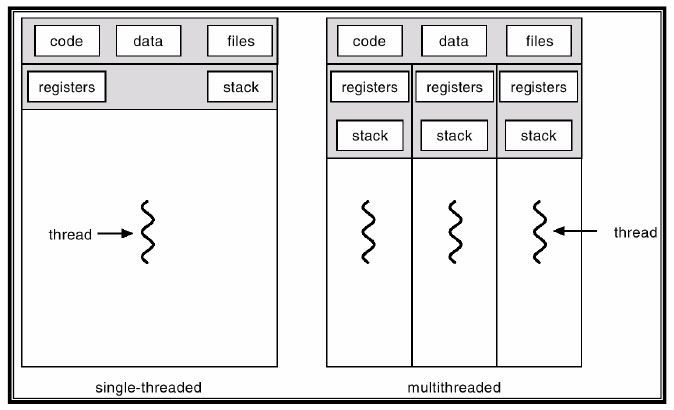
|  |  |
| --- | --- |
| E | |
|  |  |

|  |  |
| --- | --- |
| F | |
|  |  |

LCRS는 Left Child Right Sibling의 약자로 노드의 왼쪽 포인터는 자식 노드를 가리키고 오른쪽 포인터는 형제 노드를 가리키는 Tree 구조이다. 노드를 생성할 때 부모 노드와 자식 노드, 형제 노드를 생성하고 왼쪽에는 자식 노드만이 연결되고, 오른쪽에는 형제 노드만이 연결되는 형태이다. 이 Tree 구조는 하나의 부모 노드에 대해 왼쪽 자식 노드에 대한 포인터를 통해 하위 디렉토리나 파일을 구성 가능하고 오른쪽 형제 노드를 통해 동일한 위치에 여러 디렉토리와 파일을 구성할 수 있기 때문에 포인터만 있다면 모든 노드를 참조할 수 있게 된다. 이러한 자료구조를 이용해서 파일탐색기를 구현하였다.

1. **멀티스레딩 (Multi Threading)**

스레드란 어떠한 프로그램의 프로세스 내에서 실행되는 흐름의 단위를 말한다. 일반적으로 하나의 프로그램은 하나의 스레드를 가지고 작업을 수행하게 된다. 이러한 것을 단일 스레드라고 하고, 이는 한번에 하나의 명령어만을 입력할 수 있고, 그 결과로 하나의 값이 출력된다. 이는 CPU가 수행하는 작업량에 따라 처리하는 데 걸리는 시간이 결정된다. 반면에 멀티 스레드는 하나의 프로세스 내에서 여러 개의, 즉, 둘 이상의 스레드가 동시에 작업을 수행하는 것을 의미한다. 이는 여러 개의 CPU를 사용하면서 여러 개의 프로세스를 동시에 수행하는 것을 뜻한다. 멀티스레드는 하나의 스레드가 작업을 하는 동안 다른 스레드가 다른 작업을 할 수 있기 때문에 사용자와의 응답성이 좋아지고 프로세스의 메모리를 공유하기 때문에 시스템 자원의 낭비가 적다는 장점이 있다.



1. **Mini Operating System 구현**

**(1) 명령어 구현**

* 구조체 선언과 main문

typedef struct tagUserNode{

char name[MAX\_NAME];

char dir[MAX\_DIR];

int UID;

int GID;

int year;

int month;

int wday;

int day;

int hour;

int minute;

int sec;

struct tagUserNode\* LinkNode;

}UserNode;

typedef struct tagUser{

int topUID;

int topGID;

UserNode\* head;

UserNode\* tail;

UserNode\* current;

}UserList;

}

* 원형 큐를 이용한 user 정보의 구조체

typedef struct tagTreeNode{

char name[MAX\_NAME];

char type;

int mode;

int permission[9];

int SIZE;

int UID;

int GID;

int month;

int day;

int hour;

int minute;

struct tagTreeNode\* Parent;

struct tagTreeNode\* LeftChild;

struct tagTreeNode\* RightSibling;

}TreeNode;

typedef struct tagDirectoryTree{

TreeNode\* root;

TreeNode\* current;

}DirectoryTree;

}

* Lchild rsibling tree를 이용한Directory 의 정보들을 담은 구조체

typedef struct tagStackNode{

char name[MAX\_NAME];

struct tagStackNode\* LinkNode;

}StackNode;

typedef struct tagStack{

StackNode\* TopNode;

}Stack;

time\_t ltime;

struct tm \*today;

}

* Linked list stack 구조체

time\_t ltime;

struct tm \*today;

}

**시스템의 시간 반환값**

* Main 구조

int main()

{

char cmd[50];

Linux = LoadDir();

usrList = LoadUserList();

dStack = InitializeStack();

Login(usrList, Linux);

//PrintStart();

SaveUserList(usrList);

while(1){

PrintHead(Linux, dStack);

fgets(cmd, sizeof(cmd), stdin);

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

Instruction(Linux, cmd);

}

return 0;

}

}

메인문 에서는 전역변수들을 초기화 시키고 booting.txt, direc.txt등의 txt파일을 읽어 userList와 저장되어 있던 디렉토리를 불러온다. Direx.txt에 저장된 디렉토리들은 미리 정해진 형식과 순서로 저장되어 있고,

char name[MAX\_NAME];

char type;

int mode;

int permission[9];

int SIZE;

int UID;

int GID;

int month;

int day;

int hour;

int minute;에 맞는 정보를 담고 있다.

로그인에 성공하면 cmd가 실행되어 명령어를 입력 받을 준비를 한다.

Instruction 함수는 main 함수에서 fgets로 받은 명령어를 구분하여 사용자가 입력한 명령어가 무엇인지 확인 후 해당 명령어에 맞는 함수를 실행시킨다. 이때, mkdir, rm 명령어처럼 디렉토리 / 파일이 생성되거나 삭제되어 새로 저장이 필요한 경우 파일입출력을 위해 SAVE\_dir 함수가 실행된다. SAVE\_dir 함수는 변경이 있는 디렉토리를 외부의 direc.txt에 형식에 맞게 속성들을 저장한다. 만약 구현시키지 않은 명령어를 입력하면 오류메시지를 출력하고, quit을 입력하면 프로그램이 종료된다.

* LoadDir()

DirectoryTree\* LoadDir()

{

DirectoryTree\* dirTree = (DirectoryTree\*)malloc(sizeof(DirectoryTree));

char tmp[MAX\_LENGTH];

Dir = fopen("Directory.txt", "r");

while(fgets(tmp, MAX\_LENGTH, Dir) != NULL){

ReadNode(dirTree, tmp);

}

fclose(Dir);

dirTree->current = dirTree->root;

return dirTree;

}

}

LoadDir함수는 load기능을 수행하는 함수이다.

Directory.txt를 읽기권한으로 열어 fgets함수를 통해 한줄씩 ReadNode 함수를 실행시킨다.

* LoadUserList()

UserList\* LoadUserList()

{

UserList\* userList = (UserList\*)malloc(sizeof(UserList));

char tmp[MAX\_LENGTH];

User = fopen("User.txt", "r");

while(fgets(tmp, MAX\_LENGTH, User) != NULL){

ReadUser(userList, tmp);

}

fclose(User);

userList->current = NULL;

return userList;

}

LoadUserList 함수는 User.txt 읽기권한으로 열고 While문의 fgets함수를 사용하여 tmp에 저장해서 그 유저에 대한 모든 정보들을 기반으로 ReadUser함수를 실행한다.

* InitializeStack()

Stack\* InitializeStack()

{

Stack\* returnStack = (Stack\*)malloc(sizeof(Stack));

if(returnStack == NULL){

printf("error occurred, returnStack.\n");

return NULL;

}

//initialize Stack

returnStack->TopNode = NULL;

return returnStack;

}

InitializeStack함수는 stack을 생성하는 함수이다. 생성된 stack을Linked List stack구조체로 반환.

* Login()

void Login(UserList\* userList, DirectoryTree\* dirTree)

{

UserNode\* tmpUser = NULL;

char userName[MAX\_NAME];

char tmp[MAX\_DIR];

tmpUser= userList->head;

printf("Users: ");

while(tmpUser != NULL){

printf("%s ", tmpUser->name);

tmpUser = tmpUser->LinkNode;

}

printf("\n");

while(1){

printf("Login as: ");

fgets(userName, sizeof(userName), stdin);

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

if(strcmp(userName, "exit") == 0){

exit(0);

}

tmpUser = IsExistUser(userList, userName);

if(tmpUser != NULL){

userList->current = tmpUser;

break;

}

printf("'%s' 유저가 존재하지 않습니다\n", userName);

}

strcpy(tmp, userList->current->dir);

MovePath(dirTree, tmp);

}

printf("\n");

while(1){

printf("Login as: ");

fgets(userName, sizeof(userName), stdin);

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

if(strcmp(userName, "exit") == 0){

exit(0);

}

tmpUser = IsExistUser(userList, userName);

if(tmpUser != NULL){

userList->current = tmpUser;

break;

}

printf("'%s' 유저가 존재하지 않습니다\n", userName);

}

strcpy(tmp, userList->current->dir);

MovePath(dirTree, tmp);

}

Login 함수는 우리가 Linux에 접속할 때 ID를 입력함으로서 접속하는것과 유사하게 접근할 수 있도록 만든 함수이다. fget함수를 이용하여키보드에서 입력하여 배열에 UserName을 입력하고 입력받은 UserName이 유저리스트에 있는지 확인한다.

* PrintHead()

void PrintHead(DirectoryTree\* dirTree, Stack\* dirStack)

{

TreeNode\* tmpNode = NULL;

char tmp[MAX\_DIR] = "";

char tmp2[MAX\_DIR] = "";

char usr;

if(usrList->current == usrList->head)

usr = '#';

else

usr = '$';

printf("%s@os-Virtualbox", usrList->current->name);

DEFAULT;

printf(":");

tmpNode = dirTree->current;

if(tmpNode == dirTree->root){

strcpy(tmp, "/");

}

else{

while(tmpNode->Parent != NULL){

Push(dirStack, tmpNode->name);

tmpNode = tmpNode->Parent;

}

while(IsEmpty(dirStack) == 0){

strcat(tmp, "/");

strcat(tmp ,Pop(dirStack));

}

}

strncpy(tmp2, tmp, strlen(usrList->current->dir));

if(usrList->current == usrList->head){

printf("%s", tmp);

}

else if(strcmp(usrList->current->dir, tmp2) != 0){

printf("%s", tmp);

}

else{

tmpNode = dirTree->current;

while(tmpNode->Parent != NULL){

Push(dirStack, tmpNode->name);

tmpNode = tmpNode->Parent;

}

Pop(dirStack);

Pop(dirStack);

printf("~");

while(IsEmpty(dirStack) == 0){

printf("/");

printf("%s",Pop(dirStack));

}

}

DEFAULT;

printf("%c ", usr);

}

while(IsEmpty(dirStack) == 0){

strcat(tmp, "/");

strcat(tmp ,Pop(dirStack));

}

}

strncpy(tmp2, tmp, strlen(usrList->current->dir));

if(usrList->current == usrList->head){

printf("%s", tmp);

}

else if(strcmp(usrList->current->dir, tmp2) != 0){

printf("%s", tmp);

}

else{

tmpNode = dirTree->current;

while(tmpNode->Parent != NULL){

Push(dirStack, tmpNode->name);

tmpNode = tmpNode->Parent;

}

Pop(dirStack);

Pop(dirStack);

printf("~");

while(IsEmpty(dirStack) == 0){

printf("/");

printf("%s",Pop(dirStack));

}

}

DEFAULT;

printf("%c ", usr);

}

PrintHead함수는 우분투 terminal에서와 같이 명령어 입력 전 출력되는 같은 양식을 계속해서 출력해주는 함수이다.

* Instruction ()

void Instruction(DirectoryTree\* dirTree, char\* cmd)

{

char\* str;

char\* str1;

char\* str2;

int val;

if(strcmp(cmd, "") == 0 || cmd[0] == ' '){

return;

}

str = strtok(cmd, " ");

if(strcmp(str, "mkdir") == 0){

str = strtok(NULL, " ");

val = Mkdir(dirTree, str);

if(val == 0){

SaveDir(dirTree, dStack);

}

}

else if(strcmp(str, "cp") == 0){

str = strtok(NULL, " ");

str1 = strtok(NULL, " ");

val=mycp(dirTree,str,str1);

if(val == 0){

SaveDir(dirTree, dStack);

}

}

else if(strcmp(str, "rm") == 0){

str = strtok(NULL, " ");

val = rm(dirTree, str);

if(val == 0){

SaveDir(dirTree, dStack);

}

}

else if(strcmp(str, "cd") == 0){

str = strtok(NULL, " ");

cd(dirTree, str);

}

else if(strcmp(str, "pwd") == 0){

str = strtok(NULL, " ");

pwd(dirTree, dStack, str);

}

else if(strcmp(str, "ls") == 0){

str = strtok(NULL, " ");

if(str==NULL)

ls(dirTree);

else if(strcmp(str,"-a")==0)

ls\_a(dirTree);

else if(strcmp(str,"-l")==0)

ls\_l(dirTree);

else

ls\_al(dirTree);

}

else if(strcmp(str, "cat") == 0){

str = strtok(NULL, " ");

val = cat(dirTree, str);

if(val == 0){

SaveDir(dirTree, dStack);

}

}

else if(strcmp(str, "chmod") == 0){

str = strtok(NULL, " ");

val = chmod\_(dirTree, str);

if(val == 0){

SaveDir(dirTree, dStack);

}

}

else if(strcmp(str, "chown") == 0){

str = strtok(NULL, " ");

val = chown\_(dirTree, str);

if(val == 0){

SaveDir(dirTree, dStack);

}

}

else if(strcmp(str, "find") == 0){

str = strtok(NULL, " ");

find\_(dirTree, str);

}

else if(strcmp(cmd, "exit") == 0){

printf("로그아웃\n");

exit(0);

}

else if(strcmp(str, "grep") == 0){

str = strtok(NULL, " ");

str1 = strtok(NULL, " ");

str2 = strtok(NULL, " ");

if(strcmp(str,"-n")==0)

grep\_n(str1,str2);

else

grep(str,str1);

}

else{

printf("'%s': 명령을 찾을 수 없습니다\n", cmd);

}

return;

}

if(strcmp(str, "mkdir") == 0){

str = strtok(NULL, " ");

val = Mkdir(dirTree, str);

if(val == 0){

SaveDir(dirTree, dStack);

}

}

else if(strcmp(str, "cp") == 0){

str = strtok(NULL, " ");

str1 = strtok(NULL, " ");

val=mycp(dirTree,str,str1);

if(val == 0){

SaveDir(dirTree, dStack);

}

}

else if(strcmp(str, "rm") == 0){

str = strtok(NULL, " ");

val = rm(dirTree, str);

if(val == 0){

SaveDir(dirTree, dStack);

}

}

else if(strcmp(str, "cd") == 0){

str = strtok(NULL, " ");

cd(dirTree, str);

}

else if(strcmp(str, "pwd") == 0){

str = strtok(NULL, " ");

pwd(dirTree, dStack, str);

}

else if(strcmp(str, "ls") == 0){

str = strtok(NULL, " ");

if(str==NULL)

ls(dirTree);

else if(strcmp(str,"-a")==0)

ls\_a(dirTree);

else if(strcmp(str,"-l")==0)

ls\_l(dirTree);

else

ls\_al(dirTree);

}

else if(strcmp(str, "cat") == 0){

str = strtok(NULL, " ");

val = cat(dirTree, str);

if(val == 0){

SaveDir(dirTree, dStack);

}

}

else if(strcmp(str, "chmod") == 0){

str = strtok(NULL, " ");

val = chmod\_(dirTree, str);

if(val == 0){

SaveDir(dirTree, dStack);

}

}

else if(strcmp(str, "chown") == 0){

str = strtok(NULL, " ");

val = chown\_(dirTree, str);

if(val == 0){

SaveDir(dirTree, dStack);

}

}

else if(strcmp(str, "find") == 0){

str = strtok(NULL, " ");

find\_(dirTree, str);

}

else if(strcmp(cmd, "exit") == 0){

printf("로그아웃\n");

exit(0);

}

else if(strcmp(str, "grep") == 0){

str = strtok(NULL, " ");

str1 = strtok(NULL, " ");

str2 = strtok(NULL, " ");

if(strcmp(str,"-n")==0)

grep\_n(str1,str2);

else

grep(str,str1);

}

else{

printf("'%s': 명령을 찾을 수 없습니다\n", cmd);

}

return;

}

else if(strcmp(str, "chown") == 0){

str = strtok(NULL, " ");

val = chown\_(dirTree, str);

if(val == 0){

SaveDir(dirTree, dStack);

}

}

else if(strcmp(str, "find") == 0){

str = strtok(NULL, " ");

find\_(dirTree, str);

}

else if(strcmp(cmd, "exit") == 0){

printf("로그아웃\n");

exit(0);

}

else if(strcmp(str, "grep") == 0){

str = strtok(NULL, " ");

str1 = strtok(NULL, " ");

str2 = strtok(NULL, " ");

if(strcmp(str,"-n")==0)

grep\_n(str1,str2);

else

grep(str,str1);

}

else{

printf("'%s': 명령을 찾을 수 없습니다\n", cmd);

}

return;

}

Instruction 함수는 입력 받은 command를 구분하고 확인하여 . strcmp함수를 통해 구현한 명령어와 비교하여command에 맞는 함수를 실행한다.

* Cd

int cd(DirectoryTree\* dirTree, char\* cmd)

{

TreeNode\* tmpNode = NULL;

char\* str = NULL;

char tmp[MAX\_DIR];

int val;

if(cmd == NULL){

strcpy(tmp, usrList->current->dir);

MovePath(dirTree, tmp);

}

else if(cmd[0] == '-'){

if(strcmp(cmd, "--help") == 0){

printf("사용법: cd 디렉터리...\n");

printf(" Change the shell working directory.\n\n");

printf(" Options:\n");

printf(" --help\t 이 도움말을 표시하고 끝냅니다\n");

return -1;

}

else{

str = strtok(cmd, "-");

if(str == NULL){

printf("cd: 잘못된 연산자\n");

printf("Try 'cd --help' for more information.\n");

return -1;

}

else{

printf("cd: 부적절한 옵션 -- '%s'\n", str);

printf("Try 'cd --help' for more information.\n");

return -1;

}

}

}

else{

tmpNode = IsExistDir(dirTree, cmd, 'd');

if(tmpNode != NULL){

if(HasPermission(tmpNode, 'r') != 0){

printf("-bash: cd: '%s': 허가거부\n", cmd);

return -1;

}

}

tmpNode = IsExistDir(dirTree, cmd, 'f');

if(tmpNode != NULL){

printf("-bash: cd: '%s': 디렉터리가 아닙니다\n", cmd);

return -1;

}

val = MovePath(dirTree, cmd);

if(val != 0)

printf("-bash: cd: '%s': 그런 파일이나 디렉터리가 없습니다\n", cmd);

}

return 0;

}

else if(cmd[0] == '-'){

if(strcmp(cmd, "--help") == 0){

printf("사용법: cd 디렉터리...\n");

printf(" Change the shell working directory.\n\n");

printf(" Options:\n");

printf(" --help\t 이 도움말을 표시하고 끝냅니다\n");

return -1;

}

else{

str = strtok(cmd, "-");

if(str == NULL){

printf("cd: 잘못된 연산자\n");

printf("Try 'cd --help' for more information.\n");

return -1;

}

else{

printf("cd: 부적절한 옵션 -- '%s'\n", str);

printf("Try 'cd --help' for more information.\n");

return -1;

}

}

}

else{

tmpNode = IsExistDir(dirTree, cmd, 'd');

if(tmpNode != NULL){

if(HasPermission(tmpNode, 'r') != 0){

printf("-bash: cd: '%s': 허가거부\n", cmd);

return -1;

}

}

tmpNode = IsExistDir(dirTree, cmd, 'f');

if(tmpNode != NULL){

printf("-bash: cd: '%s': 디렉터리가 아닙니다\n", cmd);

return -1;

}

val = MovePath(dirTree, cmd);

if(val != 0)

printf("-bash: cd: '%s': 그런 파일이나 디렉터리가 없습니다\n", cmd);

}

return 0;

}

cd 명령어는 원하는 디렉터리로 이동하는 명령어다. –-help를 입력하면 간단한 cd 사용법이 출력되도록 하였고, cd – 가 오면 상위 디렉터리로 이동하고, cd NULL 또는 cd – 가 오면 root 위치로 돌아오게 구현하였고 그 이외의 cd – 형태가 오면 부적절한 형태 라는 말이 출력되도록 하였다. 그 이외에 IsExitDir함수를 불러와서 d일 경우, HasPermission에서 r이 아닌 경우 허가거부가 출력되도록 하였고, f일 경우는 디렉터리가 아니므로 디렉터리가 아니라는 문구가 출력되도록 하였다. 현재 위치를 이동시킬 때는 MovePath함수를 이용하여 디렉터리의 이름으로 찾아 해당 디렉터리로 이동시키고, 해당 디렉터리가 없으면 오류 문구가 출력되도록 하였다.

* Mkdir

-initialize tree 함수

DirectoryTree\* InitializeTree()

{

DirectoryTree\* dirTree = (DirectoryTree\*)malloc(sizeof(DirectoryTree));

TreeNode\* NewNode = (TreeNode\*)malloc(sizeof(TreeNode));

time(&ltime);

today = localtime(&ltime);

strncpy(NewNode->name, "/", MAX\_NAME);

NewNode->type ='d';

NewNode->mode = 755;

Mode2Permission(NewNode);

NewNode->UID = usrList->head->UID;

NewNode->GID = usrList->head->GID;

NewNode->SIZE = 4096;

NewNode->month = today->tm\_mon+1;

NewNode->day = today->tm\_mday;

NewNode->hour = today->tm\_hour;

NewNode->minute = today->tm\_min;

NewNode->Parent = NULL;

NewNode->LeftChild = NULL;

NewNode->RightSibling = NULL;

dirTree->root = NewNode;

dirTree->current = dirTree->root;

return dirTree;

}

Tree를 생성해주는 함수로 타입, 모드, UID, GID, SIZE, 시간을 저장하며 연결할 수 있는 부모, 왼쪽자식과, 오른쪽형제를 연결할 수 있는 포인터 부분을 생성한 newnode를 root에 연결해서 tree를 생성한다.

-makedir 함수

int MakeDir(DirectoryTree\* dirTree, char\* dirName, char type)

{

TreeNode\* NewNode = (TreeNode\*)malloc(sizeof(TreeNode));

TreeNode\* tmpNode = NULL;

if(HasPermission(dirTree->current, 'w') != 0){

printf("mkdir: '%s' 디렉터리를 만들 수 없습니다: 허가 거부\n", dirName);

free(NewNode);

return -1;

}

if(strcmp(dirName, ".") == 0 || strcmp(dirName, "..") == 0){

printf("mkdir: '%s' 디렉터리를 만들 수 없습니다\n", dirName);

free(NewNode);

return -1;

}

tmpNode = IsExistDir(dirTree, dirName, type);

if(tmpNode != NULL && tmpNode->type == 'd'){

printf("mkdir: '%s' 디렉터리를 만들 수 없습니다: 파일이 존재합니다\n", dirName);

free(NewNode);

return -1;

}

time(&ltime);

today = localtime(&ltime);

NewNode->LeftChild = NULL;

NewNode->RightSibling = NULL;

strncpy(NewNode->name, dirName, MAX\_NAME);

if(dirName[0] == '.'){

NewNode->type = 'd';

//rwx------

NewNode->mode = 700;

NewNode->SIZE = 4096;

}

else if(type == 'd'){

NewNode->type = 'd';

//rwxr-xr-x

NewNode->mode = 755;

NewNode->SIZE = 4096;

}

else{

NewNode->type = 'f';

//rw-r--r--

NewNode->mode = 644;

NewNode->SIZE = 0;

}

Mode2Permission(NewNode);

NewNode->UID = usrList->current->UID;

NewNode->GID = usrList->current->GID;

NewNode->month = today->tm\_mon + 1;

NewNode->day = today->tm\_mday;

NewNode->hour = today->tm\_hour;

NewNode->minute = today->tm\_min;

NewNode->Parent = dirTree->current;

if(dirTree->current->LeftChild == NULL){

dirTree->current->LeftChild = NewNode;

}

else{

tmpNode = dirTree->current->LeftChild;

while(tmpNode->RightSibling!= NULL){

tmpNode = tmpNode->RightSibling;

}

tmpNode->RightSibling = NewNode;

}

return 0;

}

if(strcmp(dirName, ".") == 0 || strcmp(dirName, "..") == 0){

printf("mkdir: '%s' 디렉터리를 만들 수 없습니다\n", dirName);

free(NewNode);

return -1;

}

tmpNode = IsExistDir(dirTree, dirName, type);

if(tmpNode != NULL && tmpNode->type == 'd'){

printf("mkdir: '%s' 디렉터리를 만들 수 없습니다: 파일이 존재합니다\n", dirName);

free(NewNode);

return -1;

}

time(&ltime);

today = localtime(&ltime);

NewNode->LeftChild = NULL;

NewNode->RightSibling = NULL;

strncpy(NewNode->name, dirName, MAX\_NAME);

if(dirName[0] == '.'){

NewNode->type = 'd';

//rwx------

NewNode->mode = 700;

NewNode->SIZE = 4096;

}

else if(type == 'd'){

NewNode->type = 'd';

//rwxr-xr-x

NewNode->mode = 755;

NewNode->SIZE = 4096;

}

else{

NewNode->type = 'f';

//rw-r--r--

NewNode->mode = 644;

NewNode->SIZE = 0;

}

Mode2Permission(NewNode);

NewNode->UID = usrList->current->UID;

NewNode->GID = usrList->current->GID;

NewNode->month = today->tm\_mon + 1;

NewNode->day = today->tm\_mday;

NewNode->hour = today->tm\_hour;

NewNode->minute = today->tm\_min;

NewNode->Parent = dirTree->current;

if(dirTree->current->LeftChild == NULL){

dirTree->current->LeftChild = NewNode;

}

else{

tmpNode = dirTree->current->LeftChild;

while(tmpNode->RightSibling!= NULL){

tmpNode = tmpNode->RightSibling;

}

tmpNode->RightSibling = NewNode;

}

return 0;

}

else{

tmpNode = dirTree->current->LeftChild;

while(tmpNode->RightSibling!= NULL){

tmpNode = tmpNode->RightSibling;

}

tmpNode->RightSibling = NewNode;

}

return 0;

}

Mkdir 함수는 새로운 디렉터리를 생성하는 명령어로 적절한 명령어가 아니면 오류가 있다는 문구가 출력되도록 하였고, 그렇지 않으면 위의 makedir함수를 사용하여 새로운 디렉터리를 생성한다.

M옵션은 chmod기능처럼 디렉토리의 mode를 설정해주는 기능이다. 입력 받은 문자열의 0,1,2번째 인덱스가 8진수이고, 문자열의 길이가 3일 때 실행 되도록 하였다. 이후 isExistDir함수로 존재유무를 확인하고 존재하지 않으면 changeMode함수로 mode를 변경해준다. 존재하면 ChangeMode, ChangeModeAll 함수를 사용하여 mode를 변경한다.

-mkdir 함수

int Mkdir(DirectoryTree\* dirTree, char\* cmd)

{

TreeNode\* tmpNode = NULL;

char\* str;

char tmp[MAX\_DIR];

char tmp2[MAX\_DIR];

char tmp3[MAX\_DIR];

int val;

int tmpMode;

if(cmd == NULL){

printf("mkdir: 잘못된 연산자\n");

printf("Try 'mkdir --help' for more information.\n");

return -1;

}

tmpNode = dirTree->current;

if(cmd[0] == '-'){

if(strcmp(cmd, "-p") == 0){

str = strtok(NULL, " ");

if(str == NULL){

printf("mkdir: 잘못된 연산자\n");

printf("Try 'mkdir --help' for more information.\n");

return -1;

}

if(strncmp(str, "/", 1) == 0){

dirTree->current = dirTree->root;

}

str = strtok(str, "/");

while(str != NULL){

val = Movecurrent(dirTree, str);

if(val != 0){

MakeDir(dirTree, str, 'd');

Movecurrent(dirTree, str);

}

str = strtok(NULL, "/");

}

dirTree->current = tmpNode;

}

else if(strcmp(cmd, "-m") == 0){

str = strtok(NULL, " ");

if(str == NULL){

printf("mkdir: 잘못된 연산자\n");

printf("Try 'mkdir --help' for more information.\n");

return -1;

}

if(str[0]-'0'<8 && str[1]-'0'<8 && str[2]-'0'<8 && strlen(str)==3){

tmpMode = atoi(str);

str = strtok(NULL, " ");

if(str == NULL){

printf("mkdir: 잘못된 연산자\n");

printf("Try 'mkdir --help' for more information.\n");

return -1;

}

val = MakeDir(dirTree, str, 'd');

if(val == 0){

tmpNode = IsExistDir(dirTree, str, 'd');

tmpNode->mode = tmpMode;

Mode2Permission(tmpNode);

}

}

else{

printf("mkdir: 잘못된 모드: '%s'\n", str);

printf("Try 'mkdir --help' for more information.\n");

return -1;

}

}

else{

str = strtok(cmd, "-");

if(str == NULL){

printf("mkdir: 잘못된 연산자\n");

printf("Try 'mkdir --help' for more information.\n");

return -1;

}

else{

printf("mkdir: 부적절한 옵션 -- '%s'\n", str);

printf("Try 'mkdir --help' for more information.\n");

return -1;

}

}

}

else{

str = strtok(NULL, " ");

if(str == NULL){

strncpy(tmp, cmd, MAX\_DIR);

if(strstr(cmd, "/") == NULL){

MakeDir(dirTree, cmd, 'd');

return 0;

}

else{

strncpy(tmp2, getDir(cmd), MAX\_DIR);

val = MovePath(dirTree, tmp2);

if(val != 0){

printf("mkdir: '%s': 그런 파일이나 디렉터리가 없습니다\n", tmp2);

return -1;

}

str = strtok(tmp, "/");

while(str != NULL){

strncpy(tmp3, str, MAX\_NAME);

str = strtok(NULL, "/");

}

MakeDir(dirTree, tmp3 , 'd');

dirTree->current = tmpNode;

}

}

}

return 0;

}}

if(val != 0){

MakeDir(dirTree, str, 'd');

Movecurrent(dirTree, str);

}

str = strtok(NULL, "/");

}

dirTree->current = tmpNode;

}

else if(strcmp(cmd, "-m") == 0){

str = strtok(NULL, " ");

if(str == NULL){

printf("mkdir: 잘못된 연산자\n");

printf("Try 'mkdir --help' for more information.\n");

return -1;

}

if(str[0]-'0'<8 && str[1]-'0'<8 && str[2]-'0'<8 && strlen(str)==3){

tmpMode = atoi(str);

str = strtok(NULL, " ");

if(str == NULL){

printf("mkdir: 잘못된 연산자\n");

printf("Try 'mkdir --help' for more information.\n");

return -1;

}

val = MakeDir(dirTree, str, 'd');

if(val == 0){

tmpNode = IsExistDir(dirTree, str, 'd');

tmpNode->mode = tmpMode;

Mode2Permission(tmpNode);

}

}

else{

printf("mkdir: 잘못된 모드: '%s'\n", str);

printf("Try 'mkdir --help' for more information.\n");

return -1;

}

}

else{

str = strtok(cmd, "-");

if(str == NULL){

printf("mkdir: 잘못된 연산자\n");

printf("Try 'mkdir --help' for more information.\n");

return -1;

}

else{

printf("mkdir: 부적절한 옵션 -- '%s'\n", str);

printf("Try 'mkdir --help' for more information.\n");

return -1;

}

}

}

else{

str = strtok(NULL, " ");

if(str == NULL){

strncpy(tmp, cmd, MAX\_DIR);

if(strstr(cmd, "/") == NULL){

MakeDir(dirTree, cmd, 'd');

return 0;

}

else{

strncpy(tmp2, getDir(cmd), MAX\_DIR);

val = MovePath(dirTree, tmp2);

if(val != 0){

printf("mkdir: '%s': 그런 파일이나 디렉터리가 없습니다\n", tmp2);

return -1;

}

str = strtok(tmp, "/");

while(str != NULL){

strncpy(tmp3, str, MAX\_NAME);

str = strtok(NULL, "/");

}

MakeDir(dirTree, tmp3 , 'd');

dirTree->current = tmpNode;

}

}

}

return 0;

}}

else{

str = strtok(NULL, " ");

if(str == NULL){

strncpy(tmp, cmd, MAX\_DIR);

if(strstr(cmd, "/") == NULL){

MakeDir(dirTree, cmd, 'd');

return 0;

}

else{

strncpy(tmp2, getDir(cmd), MAX\_DIR);

val = MovePath(dirTree, tmp2);

if(val != 0){

printf("mkdir: '%s': 그런 파일이나 디렉터리가 없습니다\n", tmp2);

return -1;

}

str = strtok(tmp, "/");

while(str != NULL){

strncpy(tmp3, str, MAX\_NAME);

str = strtok(NULL, "/");

}

MakeDir(dirTree, tmp3 , 'd');

dirTree->current = tmpNode;

}

}

}

return 0;

}}

Mkdir 함수는 새로운 디렉터리를 생성하는 명령어로 적절한 명령어가 아니면 오류가 있다는 문구가 출력되도록 하였고, 그렇지 않으면 위의 makedir함수를 사용하여 새로운 디렉터리를 생성한다.

M옵션은 chmod기능처럼 디렉토리의 mode를 설정해주는 기능이다. 입력받은 문자열의 0,1,2번째 인덱스가 8진수이고, 문자열의 길이가 3일 때 실행 되도록 하였다. 이후 isExistDir함수로 존재유무를 확인하고 존재하지않으면 changeMode함수로 mode를 변경해준다. 존재하면 ChangeMode, ChangeModeAll 함수를 사용하여 mode를 변경한다.

* Pwd

int pwd(DirectoryTree\* dirTree, Stack\* dirStack, char\* cmd)

{

char\* str = NULL;

if(cmd == NULL){

PrintPath(dirTree, dirStack);

}

else if(cmd[0] == '-'){

if(strcmp(cmd, "--help") == 0){

printf("How to use pwd : \n");

printf(" Just type in pwd.\n\n");

return -1;

}

else{

printf("wrong command\n");

printf("Try 'pwd --help'.\n");

return -1;

}

}

else {

printf("wrong command\n");

printf("Try 'pwd --help'.\n");

return -1;

}

return 0;

}

}

현재 작업 중인 directory의 위치를 알려주는 명령어이다. Pwd만 입력하면 PrintPath함수가 호출되어 directory의 위치가 출력된다. --help를 입력하면, pwd 사용 방법이 나오고, 그 외의 명령어는 모두 wrong command를 출력한 뒤, --help를 권장한다.

-PrintPath 함수

int PrintPath(DirectoryTree\* dirTree, Stack\* dirStack)

{ TreeNode\* tmpNode = NULL;

tmpNode = dirTree->current;

if(tmpNode == dirTree->root){

printf("/");

}

else{

while(tmpNode->Parent != NULL){

Push(dirStack, tmpNode->name);

tmpNode = tmpNode->Parent;

}

while(IsEmpty(dirStack) == 0){

printf("/");

printf("%s",Pop(dirStack));

}

}

printf("\n");

return 0;

}

}

현재 directory가 root에 있을 때 /를 출력하고 현재 directory가 root가 아닐 때 push를 반복한다. stack이 빌 때까지 pop을 반복하는데, IsEmpty함수를 이용해서 stack이 0이 되면 /을 출력한다.

* ls

void ls(DirectoryTree\* dirTree){

int count = 1;

TreeNode\* tmpNode = dirTree->current;

if(tmpNode->LeftChild==NULL)

printf("directory empty\n");

else{

tmpNode=tmpNode->LeftChild;

while(tmpNode->RightSibling!=NULL){

if(strlen(tmpNode->name)<8)

printf("%s\t\t",tmpNode->name);

else

printf("%s\t",tmpNode->name);

tmpNode=tmpNode->RightSibling;

if(count%5==0)

printf("\n");

count++;

}

printf("%s\t\n",tmpNode->name);

}

}

}

ls는 파일과 디렉터리의 모든 정보를 제공하며 특정 디렉터리와 특정 파일의 내용을 제공하는 명령어이다. Ptr->Ichild 했을때 NULL일 경우 디렉터리가 비었다는 출력문이 뜨도록 하였다. 그 외에 NULL이 아닐경우 디렉터리의 모든 정보가 뜨도록 구성하였다. If(strlen(ptr->name)<8)은 정보에 대한 줄을 출력했을 때 줄이 더 깔끔하게 출력될 수 있도록 하였다. ls에서는 ls함수, ls-l함수, ls-a함수, ls-al함수로 나누어 구성했다.

-ls\_a 함수

void ls\_a(DirectoryTree\* dirTree){

int count = 1;

TreeNode\* tmpNode = dirTree->current;

if(tmpNode->LeftChild==NULL){

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

}

else{

printf(".\t\t..\t\t");

count = count + 2;

tmpNode=tmpNode->LeftChild;

while(tmpNode->RightSibling!=NULL){

if(strlen(tmpNode->name)<8)

printf("%s\t\t",tmpNode->name);

else

printf("%s\t",tmpNode->name);

tmpNode=tmpNode->RightSibling;

if(count%5==0)

printf("\n");

count++;

}

printf("%s\t\n",tmpNode->name);

}

}

}

else

printf("%s\t",tmpNode->name);

tmpNode=tmpNode->RightSibling;

if(count%5==0)

printf("\n");

count++;

}

printf("%s\t\n",tmpNode->name);

}

}

}

ls-a 함수는 모든 디렉토리를 출력한다. 만약 lchild가 없으면 directory empty를 출력하고 만약 RightSibling이 null이 아닐 경우 tmpNode의 모든 디렉토리를 출력한다. 출력할 때 If(strlen(ptr->name)<8)을 통해 깔끔하게 디렉토리가 출력되도록 구성하였다.

-ls\_l 함수

void ls\_l(DirectoryTree\* dirTree){

time\_t timer;

TreeNode\* tmpNode = dirTree->current;

timer = time(NULL);

if(tmpNode->LeftChild==NULL)

printf("directory empty\n");

else{

tmpNode=tmpNode->LeftChild;

while(tmpNode->RightSibling!=NULL){

printf("%c",tmpNode->type);

PrintPermission(tmpNode);

printf(" %d %d %d",tmpNode->SIZE,tmpNode->UID,tmpNode->GID);

printf(" %d(month) %d(day) %d(hour) %d(min) ",tmpNode->month,tmpNode->day,tmpNode->hour,tmpNode->minute);

printf("%s\n",tmpNode->name);

tmpNode=tmpNode->RightSibling;

}

printf("%c",tmpNode->type);

PrintPermission(tmpNode);

printf(" %d %d %d",tmpNode->SIZE,tmpNode->UID,tmpNode->GID);

printf(" %d(month) %d(day) %d(hour) %d(min) ",tmpNode->month,tmpNode->day,tmpNode->hour,tmpNode->minute);

printf("%s\n",tmpNode->name);

}

}

}

ls-l은 -ls에 대한 정보를 출력한다. 만약 lchild가 null일 경우 direct empty를 출력한다. Lchild가 null이 아닐 경우 lchild의 rsibling이 null이 아니면 rsibling으로 이동한다. 이 외에 type,size 등 그에 대한 모든 정보를 출력한다. Rsibling이 null일 경우 rsibling으로 이동하지 않고 현재 노드에서 모든 정보를 출력한다.

-ls\_al 함수

void ls\_al(DirectoryTree\* dirTree){

time\_t timer;

TreeNode\* tmpNode = dirTree->current;

timer = time(NULL);

if(tmpNode->LeftChild==NULL){

//.

printf("%c",dirTree->current->type);

PrintPermission(dirTree->current);

printf(" %d %d %d",dirTree->current->SIZE,dirTree->current->UID,dirTree->current->GID);

printf(" %d(month) %d(day) %d(hour) %d(min) ",dirTree->current->month,dirTree->current->day,dirTree->current->hour,dirTree->current->minute);

printf(".\n");

//..

if(strcmp(dirTree->current->name,"/")==0){

printf("%c",dirTree->current->type);

PrintPermission(dirTree->current);

printf(" %d %d %d",dirTree->current->SIZE,dirTree->current->UID,dirTree->current->GID);

printf(" %d(month) %d(day) %d(hour) %d(min) ",dirTree->current->month,dirTree->current->day,dirTree->current->hour,dirTree->current->minute);

printf("..\n");

}

else{

printf("%c",tmpNode->Parent->type);

PrintPermission(tmpNode->Parent);

printf(" %d %d %d",tmpNode->Parent->SIZE,tmpNode->Parent->UID,tmpNode->Parent->GID);

printf(" %d(month) %d(day) %d(hour) %d(min) ",tmpNode->Parent->month,tmpNode->Parent->day,tmpNode->Parent->hour,tmpNode->Parent->minute);

printf("..\n");

}

}

else{

//.

printf("%c",dirTree->current->type);

PrintPermission(dirTree->current);

printf(" %d %d %d",dirTree->current->SIZE,dirTree->current->UID,dirTree->current->GID);

printf(" %d(month) %d(day) %d(hour) %d(min) ",dirTree->current->month,dirTree->current->day,dirTree->current->hour,dirTree->current->minute);

printf(".\n");

//..

if(strcmp(dirTree->current->name,"/")==0){

printf("%c",dirTree->current->type);

PrintPermission(dirTree->current);

printf(" %d %d %d",dirTree->current->SIZE,dirTree->current->UID,dirTree->current->GID);

printf(" %d(month) %d(day) %d(hour) %d(min) ",dirTree->current->month,dirTree->current->day,dirTree->current->hour,dirTree->current->minute);

printf("..\n");

}

else{

printf("%c",tmpNode->Parent->type);

PrintPermission(tmpNode->Parent);

printf(" %d %d %d",tmpNode->Parent->SIZE,tmpNode->Parent->UID,tmpNode->Parent->GID);

printf(" %d(month) %d(day) %d(hour) %d(min) ",tmpNode->Parent->month,tmpNode->Parent->day,tmpNode->Parent->hour,tmpNode->Parent->minute);

printf("..\n");

}

tmpNode=tmpNode->LeftChild;

while(tmpNode->RightSibling!=NULL){

printf("%c",tmpNode->type);

PrintPermission(tmpNode);

printf(" %d %d %d",tmpNode->SIZE,tmpNode->UID,tmpNode->GID);

printf(" %d(month) %d(day) %d(hour) %d(min) ",tmpNode->month,tmpNode->day,tmpNode->hour,tmpNode->minute);

printf("%s\n",tmpNode->name);

tmpNode=tmpNode->RightSibling;

}

printf("%c",tmpNode->type);

PrintPermission(tmpNode);

printf(" %d %d %d",tmpNode->SIZE,tmpNode->UID,tmpNode->GID);

printf(" %d(month) %d(day) %d(hour) %d(min) ",tmpNode->month,tmpNode->day,tmpNode->hour,tmpNode->minute);

printf("%s\n",tmpNode->name);

}

}

}

UID,dirTree->current->GID);

printf(" %d(month) %d(day) %d(hour) %d(min) ",dirTree->current->month,dirTree->current->day,dirTree->current->hour,dirTree->current->minute);

printf("..\n");

}

else{

printf("%c",tmpNode->Parent->type);

PrintPermission(tmpNode->Parent);

printf(" %d %d %d",tmpNode->Parent->SIZE,tmpNode->Parent->UID,tmpNode->Parent->GID);

printf(" %d(month) %d(day) %d(hour) %d(min) ",tmpNode->Parent->month,tmpNode->Parent->day,tmpNode->Parent->hour,tmpNode->Parent->minute);

printf("..\n");

}

tmpNode=tmpNode->LeftChild;

while(tmpNode->RightSibling!=NULL){

printf("%c",tmpNode->type);

PrintPermission(tmpNode);

printf(" %d %d %d",tmpNode->SIZE,tmpNode->UID,tmpNode->GID);

printf(" %d(month) %d(day) %d(hour) %d(min) ",tmpNode->month,tmpNode->day,tmpNode->hour,tmpNode->minute);

printf("%s\n",tmpNode->name);

tmpNode=tmpNode->RightSibling;

}

printf("%c",tmpNode->type);

PrintPermission(tmpNode);

printf(" %d %d %d",tmpNode->SIZE,tmpNode->UID,tmpNode->GID);

printf(" %d(month) %d(day) %d(hour) %d(min) ",tmpNode->month,tmpNode->day,tmpNode->hour,tmpNode->minute);

printf("%s\n",tmpNode->name);

}

}

}

ls-al은 모든 디렉토리의 정보를 출력한다. 만약 lchild가 null일 경우 directory empty를 출력한다. 만약 lchild가 null이 아닐 경우 rsibling과 lchild의 정보를 모두 출력한다.

. 의 경우 본인의 디렉토리 정보를 읽어와 출력하고, .. 의 경우 상위 디렉토리의 정보를 읽어와 출력한다.

* rm

int rm(DirectoryTree\* dirTree, char\* cmd)

{

TreeNode\* currentNode = NULL;

TreeNode\* tmpNode = NULL;

TreeNode\* tmpNode2 = NULL;

char\* str;

char tmp[MAX\_DIR];

char tmp2[MAX\_DIR];

char tmp3[MAX\_DIR];

int val;

if (cmd == NULL) {

printf("rm: 잘못된 연산자\n");

printf("Try 'rm -- help' for more information\n");

return -1;

}

currentNode = dirTree->current;

if (cmd[0] == '-') {

if (strcmp(cmd, "-r") == 0) {

str = strtok(NULL, " ");

if (str == NULL) {

printf("rm: 잘못된 연산자\n");

printf("Try 'rm -- help' for more information\n");

return -1;

}

strncpy(tmp, str, MAX\_DIR);

if (strstr(str, "/") == NULL) {

tmpNode = lsExistDir(dirTree, str, 'd');

if (tmpNode == NULL) {

printf("rm : '%s'를 지울 수 없음: 그런 파일이나 디렉터리가 없습니다.\n", str);

return -1;

}

else {

if (HasPermission(dirTree->current, 'w') != 0 || HasPermission(tmpNode, 'w') != 0) {

printf("rm : '%s'파일 또는 디렉터리를 지울 수 없습니다.: 허가거부\n", str);

return -1;

}

RemoveDir(dirTree, str);

}

}

else {

strncpy(tmp2, getDir(str), MAX\_DIR);

val = MovePath(dirTree, tmp2);

if (val != 0) {

printf("rm : '%s': 그런 파일이나 디렉터리가 없습니다.\n", tmp2);

return -1;

}

str = strtok(tmp, '/');

while (str != NULL) {

}

return -1;

}

str = strtok(tmp, '/');

while (str != NULL) {

strncpy(tmp3, str, MAX\_NAME);

str = strtok(NULL, "/");

}

tmpNode = IsExistDir(dirTree, tmp3, 'd');

if (tmpNode == NULL) {

printf("rm : '%s'를 지울 수 없음: 그런 파일이나 디렉터리가 없습니다.\n", tmp3);

dirTree->current = currentNode;

return -1;

}

else {

if (HadPermission(dirTree->current, 'w') != 0 || HasPermission(tmpNode, 'w') != 0) {

printf("rm : '%s'파일 또는 디렉터리를 지울 수 없습니다.: 허가거부\n", tmp3);

dirTree->current = currentNode;

return -1;

}

RemoveDir(dirTree, tmp3);

}

dirTree->current = currentNode;

}

}

else if (strcmp(cmd, "-f") == 0) {

str = strtok(NULL, " ");

if (str == NULL) {

return -1;

}

strncpy(tmp, str, MAX\_DIR);

if (strstr(str, "/") == NULL) {

tmpNode = IsExistDir(dirTree, str, 'f');

tmpNode2 = IsExistDir(dirTree, str, 'd');

if (tmpNode2 != NULL) {

return -1;

}

if (tmpNode == NULL) {

return -1;

}

else{

if (HasPermission(dirTree->current, 'w') != 0 || HasPermission(tmpNode, 'w') != 0) {

return -1;

}

RemoveDir(dirTree,str);

}

}

else {

strncpy(tmp2, getDir(str), MAX\_DIR);

val = MovePath(dirTree, tmp2);

if (val != 0) {

return -1;

}

str = strtok(tmp, "/");

while (str != NULL) {

strncpy(tmp3, str, MAX\_NAME);

str = strtok(NULL, "/");

}

tmpNode = IsExistDir(dirTree, tmp3, 'f');

tmpNode2 = IsExistDir(dirTree, tmp3, 'd');

if (tmpNode2 != NULL) {

dirTree->current = currentNode;

return -1;

}

else {

strncpy(tmp2, getDir(str), MAX\_DIR);

val = MovePath(dirTree, tmp2);

if (val != 0) {

return -1;

}

str = strtok(tmp, "/");

while (str != NULL) {

strncpy(tmp3, str, MAX\_NAME);

str = strtok(NULL, "/");

}

tmpNode = IsExistDir(dirTree, tmp3, 'f');

tmpNode2 = IsExistDir(dirTree, tmp3, 'd');

if (tmpNode2 != NULL) {

dirTree->current = currentNode;

return -1;

}

if (tmpNode == NULL) {

dirTree->current = currentNode;

return -1;

}

else {

if (HasPermission(dirTree->current, 'w') != 0 || HasPermission(tmpNode, 'w') != 0) {

dirTree->current = currentNode;

return -1;

}

RemoveDir(dirTree, tmp3);

}

dirTree->current = currentNode;

}

}

else if (strcmp(cmd, "-rf") == 0) {

str = strtok(NULL, " ");

if (str == NULL) {

return -1;

}

strncpy(tmp, str, MAX\_DIR);

if (strstr(str, "/") == NULL) {

tmpNode = IsExistDir(dirTree, str, 'd');

if (tmpNode != NULL) {

return -1;

}

else {

if (HasPermission(dirTree->current, 'w') != 0 || HasPermission(tmpNode, 'w') != 0) {

return -1;

}

RemoveDir(dirTree, str);

}

}

else {

strncpy(tmp2, getDir(str), MAX\_DIR);

val = MovePath(dirTree, tmp2);

if (val != 0) {

return -1;

}

str = strtok(tmp, "/");

while (str != NULL) {

strncpy(tmp3, str, MAX\_NAME);

str = strtok(NULL,"/");

}

tmpNode = IsExistDir(dirTree, tmp3, 'd');

if (tmpNode == NULL) {

dirTree->current = currentNode;

return -1;

}

else {

if (HasPermission(dirTree->current, 'w') != 0 || HasPermission(tmpNode, 'w') != 0) {

dirTree->current = currentNode;

return -1;

}

RemoveDir(dirTree, tmp3);

}

dirTree->current = currentNode;

}

}

else if (strcmp(cmd, "--help") == 0) {

}

RemoveDir(dirTree, str);

}

}

else {

strncpy(tmp2, getDir(str), MAX\_DIR);

val = MovePath(dirTree, tmp2);

if (val != 0) {

return -1;

}

str = strtok(tmp, "/");

while (str != NULL) {

strncpy(tmp3, str, MAX\_NAME);

str = strtok(NULL,"/");

}

tmpNode = IsExistDir(dirTree, tmp3, 'd');

if (tmpNode == NULL) {

dirTree->current = currentNode;

return -1;

}

else {

if (HasPermission(dirTree->current, 'w') != 0 || HasPermission(tmpNode, 'w') != 0) {

dirTree->current = currentNode;

return -1;

}

RemoveDir(dirTree, tmp3);

}

dirTree->current = currentNode;

}

}

else if (strcmp(cmd, "--help") == 0) {

printf("사용법: rm[<옵션>]...[<파일>]...\n");

printf(" Remove (unlink) the FILE(s).\n\n");

printf(" Options:\n");

printf(" -f, --force \t ignore nonexistent files and arguments,never prompt\n");

printf(" -r, --recursive\t remove directories and their contents recursively\n");

printf(" --help\t 이 도움말을 표시하고 끝냅니다.\n");

return -1;

}

else {

str = strtok(cmd, "-");

if (str == NULL) {

printf("rm: 잘못된 연산자\n");

printf("Try 'rm --help' for more information\n");

return -1;

}

else {

printf("rm: 부적절한 옵션 -- '%s'\n", str);

printf("Try 'rm --help' for more information\n");

return -1;

}

}

}

else {

strncpy(tmp, cmd, MAX\_DIR);

if (strstr(cmd, "/") == NULL) {

tmpNode = IsExistDir(dirTree, cmd, 'f');

tmpNode2 = IsExistDir(dirTree, cmd, 'd');

if (tmpNode2 != NULL) {

printf("rm: '%s'를 지울 수 없음: 디렉터리입니다.\n", cmd);

return -1;

}

if (tmpNode == NULL) {

printf("rm: '%s'를 지울 수 없음: 그런 파일이나 디렉터리가 없습니다.\n", cmd);

return -1;

}

else {

if (HasPermission(dirTree->current, 'w') != 0 || HasPermission(tmpNode, 'w') != 0) {

printf("rm : '%s'파일 또는 디렉터리를 지울 수 없습니다.: 허가거부\n", cmd);

return -1;

}

RemoveDir(dirTree, cmd);

}

}

else {

strncpy(tmp2, getDir(cmd), MAX\_DIR);

val = MovePath(dirTree, tmp2);

if (val != 0) {

printf("rm: '%s': 그런 파일이나 디렉터리가 없습니다.\n", tmp2);

return -1;

}

str = strtok(tmp, "/");

while (str != NULL) {

strncpy(tmp3, str, MAX\_NAME);

str = strtok(NULL, "/");

}

tmpNode = IsExistDir(dirTree, tmp3, 'f');

tmpNode2 = IsExistDir(dirTree, tmp3, 'd');

}

printf("Try 'rm --help' for more information\n");

return -1;

}

}

}

else {

strncpy(tmp, cmd, MAX\_DIR);

if (strstr(cmd, "/") == NULL) {

tmpNode = IsExistDir(dirTree, cmd, 'f');

tmpNode2 = IsExistDir(dirTree, cmd, 'd');

if (tmpNode2 != NULL) {

printf("rm: '%s'를 지울 수 없음: 디렉터리입니다.\n", cmd);

return -1;

}

if (tmpNode == NULL) {

printf("rm: '%s'를 지울 수 없음: 그런 파일이나 디렉터리가 없습니다.\n", cmd);

return -1;

}

else {

if (HasPermission(dirTree->current, 'w') != 0 || HasPermission(tmpNode, 'w') != 0) {

printf("rm : '%s'파일 또는 디렉터리를 지울 수 없습니다.: 허가거부\n", cmd);

return -1;

}

RemoveDir(dirTree, cmd);

}

}

else {

strncpy(tmp2, getDir(cmd), MAX\_DIR);

val = MovePath(dirTree, tmp2);

if (val != 0) {

printf("rm: '%s': 그런 파일이나 디렉터리가 없습니다.\n", tmp2);

return -1;

}

str = strtok(tmp, "/");

while (str != NULL) {

strncpy(tmp3, str, MAX\_NAME);

str = strtok(NULL, "/");

}

tmpNode = IsExistDir(dirTree, tmp3, 'f');

tmpNode2 = IsExistDir(dirTree, tmp3, 'd');

if (tmpNode2 != NULL) {

printf("rm: %s'를 지울 수 없음: 디렉터리입니다.\n", tmp3);

dirTree->current = currentNode;

return -1;

}

if (tmpNode == NULL) {

printf("rm: '%s'를 지울 수 없음: 그런 파일이나 디렉터리가 없습니다.\n", tmp3);

dirTree->current = currentNode;

return -1;

}

else {

if (HasPermission(dirTree->current, 'w') != 0 || HasPermission(tmpNode, 'w') != 0) {

printf("rm : '%s'파일 또는 디렉터리를 지울 수 없습니다.: 허가거부\n", tmp3);

dirTree->current = currentNode;

return -1;

}

RemoveDir(dirTree, tmp3);

}

dirTree->current = currentNode;

}

}

return 0;

}

}

if (tmpNode == NULL) {

printf("rm: '%s'를 지울 수 없음: 그런 파일이나 디렉터리가 없습니다.\n", tmp3);

dirTree->current = currentNode;

return -1;

}

else {

if (HasPermission(dirTree->current, 'w') != 0 || HasPermission(tmpNode, 'w') != 0) {

printf("rm : '%s'파일 또는 디렉터리를 지울 수 없습니다.: 허가거부\n", tmp3);

dirTree->current = currentNode;

return -1;

}

RemoveDir(dirTree, tmp3);

}

dirTree->current = currentNode;

}

}

return 0;

}

}

Rm함수는 빈 파일이나 디렉터리를 삭제하는 기능을 가진 함수이다.

명령이 -r일 때, -f일 때, -rf일 때, 아무것도 쓰지 않았을 때로 나누어서 구성했다.

Rm을 치고 아무것도 치지 않으면 “rm: 잘못된 연산자” 라는 메시지를 띄운다.

-r일 때는 하위 파일과 디렉터리를 지울 수 있다. IsExistDir 함수를 사용하여 파일이나 디렉터리가 존재하는지 확인하였고 존재한다면 지울 수 있도록 구현했다.

-f 일 때는 강제 삭제 기능을 가지고 있다. 또한 디렉터리는 지울 수 없어 이를 고려하여 구현했다. 마찬가지로 IsExistDir 함수를 사용하여 존재하는지 확인했고 존재한다면 지웠다. 만약 디렉터리를 지우는 경우엔 “디렉터리입니다”라는 메시지를 띄워 지우지 못하도록 하였다.

-rf 는 -r과 -f 를 합쳐 디렉터리까지 강제 삭제가 가능하도록 구현하였다.

--help인 경우에는 rm의 옵션을 설명해주는 문구가 출력되도록 구성하였다.

-RemoveDir 함수

int RemoveDir(DirectoryTree\* dirTree, char\* cmd) {

TreeNode\* DelNode = NULL;

TreeNode\* tmpNode = NULL;

TreeNode\* prevNode = NULL;

tmpNode = dirTree->current->LeftChild;

if (cmd == NULL) {

printf("rm : '%s'를 지울 수 없음; 파일이나 디렉터리를 확인해주세요\n", dirName);

return -1;

}

if (strcmp(tmpNode->name, dirName) == 0) {

dirTree->current->Leftchild = tmpNode->RightSibling;

DelNode = tmpNode;

if (DelNode->LeftChild != NULL) {

DestroyDir(DelNode->LeftChild);

}

DestroyNode(DelNode);

}

else {

while (tmpNode != NULL) {

if (strcmp(tmpNode->name, dirName) == 0) {

DelNode = tmpNode;

break;

}

prevNode = tmpNode;

tmpNode = tmpNode->RightSibling;

}

if (DelNode != NULL) {

prevNode->RightSibling = DelNode->RightSibling;

if (DelNode->LeftChild != NULL) {

DestroyDir(DelNode->LeftChild);

}

DestroyNode(DelNode);

}

else {

printf("rm : '%s'를 지울 수 없음; 파일이나디렉터리를 확인해주세요.\n", dirname);

return -1;

}

}

return 0;

}

if (strcmp(tmpNode->name, dirName) == 0) {

dirTree->current->Leftchild = tmpNode->RightSibling;

DelNode = tmpNode;

if (DelNode->LeftChild != NULL) {

DestroyDir(DelNode->LeftChild);

}

DestroyNode(DelNode);

}

else {

while (tmpNode != NULL) {

if (strcmp(tmpNode->name, dirName) == 0) {

DelNode = tmpNode;

break;

}

prevNode = tmpNode;

tmpNode = tmpNode->RightSibling;

}

if (DelNode != NULL) {

prevNode->RightSibling = DelNode->RightSibling;

if (DelNode->LeftChild != NULL) {

DestroyDir(DelNode->LeftChild);

}

DestroyNode(DelNode);

}

else {

printf("rm : '%s'를 지울 수 없음; 파일이나디렉터리를 확인해주세요.\n", dirname);

return -1;

}

}

return 0;

}

RemoveDir 함수는 rm을 실행할 때 사용되는 함수이다.

먼저 HasPermisson 함수를 사용하여 쓰기 권한이 있는지 확인하고 없으면 불가 메시지를 띄운다. 하위 파일이나 디렉토리를 삭제하는 것이기 때문에 tempNode를 leftchild로 설정하였고, 만약 leftchild에 파일이 없을 경우, 즉 NULL값이면 “그런 파일이나 디렉터리가 없습니다.”라는 문구가 출력되도록 설정해주었다.

NULL값이 아닐때에는 tempNode를 rightSibling을 통해 LeftChild로 지정한다. 그러면 tempNode는 독립적이기 때문에, tempNode를 DelNode로 설정하고 현재 노드 포함해서 하위 노드들까지 다 삭제한다.

그리고 else문에서 dirName과 tmpNode의 이름이 같지 않을때를 검사한다. 이때는 rightsibling으로 계속 돌려서 dirName과 같으면 break를 통해 탈출한다.

만약 DelNode가 NULL이라면 파일이나 디렉터리가 없는 것이기 때문에 “그런 파일이나 디렉터리가 없습니다.”라는 메시지를 띄우고 NULL이 아닐 경우 중간에 파일이나 디렉터리가 끊어지지 않도록 prevNode->RightSibling = DelNode->RightSibling를 통해 이어주었다.

그 후 DestroyNode 함수와 DestroyDir 함수를 사용하여 삭제했다.

* cat

int cat(DirectoryTree\* dirTree, char\* cmd)

{

TreeNode\* currentNode = NULL;

TreeNode\* tmpNode = NULL;

TreeNode\* tmpNode2 = NULL;

char\* str;

char tmp[MAX\_DIR];

char tmp2[MAX\_DIR];

char tmp3[MAX\_DIR];

int val;

/\*

cat0: write, EOF to save

cat1: read

cat2: read w/ line number

\*/

if(cmd == NULL){

printf("cat: 잘못된 연산자\n");

return -1;

}

currentNode = dirTree->current;

if(strcmp(cmd, ">") == 0){

str = strtok(NULL, " ");

if(str == NULL){

printf("cat: 잘못된 연산자\n");

printf("Try 'cat --help' for more information.\n");

return -1;

}

strncpy(tmp, str, MAX\_DIR);

if(strstr(str, "/") == NULL){

if(HasPermission(dirTree->current, 'w') != 0){

printf("cat: '%s'파일을 만들 수 없음: 권한없음\n", dirTree->current->name);

return -1;

}

tmpNode = IsExistDir(dirTree, str, 'd';

if(tmpNode != NULL){

printf("cat: '%s': 디렉터리입니다\n", str);

return -1;

}

else{

Concatenate(dirTree, str, 0);

}

}

else{

strncpy(tmp2, getDir(str), MAX\_DIR);

val = MovePath(dirTree, tmp2);

if(val != 0){

printf("cat: '%s': 그런 파일이나 디렉터리가 없습니다\n", tmp2);

return -1;

}

str = strtok(tmp, "/");

while(str != NULL){

strncpy(tmp3, str, MAX\_NAME);

str = strtok(NULL, "/");

}

if(HasPermission(dirTree->current, 'w') != 0){

printf("cat: '%s'파일을 만들 수 없음: 허가거부\n", dirTree->current->name);

dirTree->current = currentNode;

return -1;

}

tmpNode = IsExistDir(dirTree, tmp3, 'd');

if(tmpNode != NULL){

printf("cat: '%s': 디렉터리입니다\n", tmp3);

dirTree->current = currentNode;

return -1;

}

else{

Concatenate(dirTree, tmp3, 0);

}

dirTree->current = currentNode;

}

return 0;

}

else if(cmd[0] == '-'){

if(strcmp(cmd, "-n")== 0){

str = strtok(NULL, " ");

strncpy(tmp, str, MAX\_DIR);

if(strstr(str, "/") == NULL){

if(HasPermission(dirTree->current, 'w') != 0){

printf("cat: '%s'파일을 만들 수 없음: 허가거부\n", dirTree->current->name);

return -1;

}

tmpNode = IsExistDir(dirTree, str, 'd');

tmpNode2 = IsExistDir(dirTree, str, 'f');

if(tmpNode == NULL && tmpNode2 == NULL){

printf("cat: '%s': 그런 파일이나 디렉터리가 없습니다\n", str);

return -1;

}

else if(tmpNode != NULL && tmpNode2 == NULL){

printf("cat: '%s': 디렉터리입니다\n", str);

return -1;

}

else if(tmpNode2 != NULL && HasPermission(tmpNode2, 'r') != 0){

printf("cat: '%s'파일을 열 수 없음: 허가거부\n", tmpNode2->name);

return -1;

}

else{

Concatenate(dirTree, str, 2);

}

}

else{

strncpy(tmp2, getDir(str), MAX\_DIR);

val = MovePath(dirTree, tmp2);

else{

Concatenate(dirTree, str, 2);

}

}

else{

strncpy(tmp2, getDir(str), MAX\_DIR);

val = MovePath(dirTree, tmp2);

if(val != 0){

printf("cat: '%s': 그런 파일이나 디렉터리가 없습니다\n", tmp2);

return -1;

}

str = strtok(tmp, "/");

while(str != NULL){

strncpy(tmp3, str, MAX\_NAME);

str = strtok(NULL, "/");

}

tmpNode = IsExistDir(dirTree, tmp3, 'd');

tmpNode2 = IsExistDir(dirTree, tmp3, 'f');

if(tmpNode == NULL && tmpNode2 == NULL){

printf("cat: '%s': 그런 파일이나 디렉터리가 없습니다\n", tmp3);

dirTree->current = currentNode;

return -1;

}

else if(tmpNode != NULL && tmpNode2 == NULL){

printf("cat: '%s': 디렉터리입니다\n", tmp3);

dirTree->current = currentNode;

return -1;

}

else if(tmpNode2 != NULL && HasPermission(tmpNode2, 'r') != 0){

printf("cat: '%s'파일을 열 수 없음: 허가거부\n", tmpNode2->name);

dirTree->current = currentNode;

return -1;

}

else{

Concatenate(dirTree, tmp3, 2);

}

dirTree->current = currentNode;

}

}

else if(strcmp(cmd, "-b")== 0){

str = strtok(NULL, " ");

strncpy(tmp, str, MAX\_DIR);

if(strstr(str, "/") == NULL){

if(HasPermission(dirTree->current, 'w') != 0){

printf("cat: '%s'파일을 만들 수 없음: 허가거부\n", dirTree->current->name);

return -1;

}

tmpNode = IsExistDir(dirTree, str, 'd');

tmpNode2 = IsExistDir(dirTree, str, 'f');

if(tmpNode == NULL && tmpNode2 == NULL){

printf("cat: '%s': 그런 파일이나 디렉터리가 없습니다\n", str);

return -1;

}

tmpNode = IsExistDir(dirTree, str, 'd');

tmpNode2 = IsExistDir(dirTree, str, 'f');

if(tmpNode == NULL && tmpNode2 == NULL){

printf("cat: '%s': 그런 파일이나 디렉터리가 없습니다\n", str);

return -1;

}

else if(tmpNode != NULL && tmpNode2 == NULL){

printf("cat: '%s': 디렉터리입니다\n", str);

return -1;

}

else if(tmpNode2 != NULL && HasPermission(tmpNode2, 'r') != 0){

printf("cat: '%s'파일을 열 수 없음: 허가거부\n", tmpNode2->name);

return -1;

}

else{

Concatenate(dirTree, str, 3);

}

}

else{

strncpy(tmp2, getDir(str), MAX\_DIR);

val = MovePath(dirTree, tmp2);

if(val != 0){

printf("cat: '%s': 그런 파일이나 디렉터리가 없습니다\n", tmp2);

return -1;

}

str = strtok(tmp, "/");

while(str != NULL){

strncpy(tmp3, str, MAX\_NAME);

str = strtok(NULL, "/");

}

tmpNode = IsExistDir(dirTree, tmp3, 'd');

tmpNode2 = IsExistDir(dirTree, tmp3, 'f');

if(tmpNode == NULL && tmpNode2 == NULL){

printf("cat: '%s': 그런 파일이나 디렉터리가 없습니다\n", tmp3);

dirTree->current = currentNode;

return -1;

}

else if(tmpNode != NULL && tmpNode2 == NULL){

printf("cat: '%s': 디렉터리입니다\n", tmp3);

dirTree->current = currentNode;

return -1;

}

else if(tmpNode2 != NULL && HasPermission(tmpNode2, 'r') != 0){

printf("cat: '%s'파일을 열 수 없음: 허가거부\n", tmpNode2->name);

dirTree->current = currentNode;

return -1;

}

else{

Concatenate(dirTree, tmp3, 3);

}

dirTree->current = currentNode;

}

}

else if(strcmp(cmd, "--help") == 0){

printf("사용법: cat [<옵션>]... [<파일>]...\n");

printf(" FILE(들)을 합쳐서 표준 출력으로 보낸다.\n\n");

printf(" Options:\n");

printf(" -n, --number \t number all output line\n");

printf(" -b, --number-nonblank\t number nonempty output line\n");

printf(" --help\t 이 도움말을 표시하고 끝냅니다\n");

return -1;

}

else{

str = strtok(cmd, "-");

if(str == NULL){

printf("cat: 잘못된 연산자\n");

printf("Try 'cat --help' for more information.\n");

return -1;

else{

Concatenate(dirTree, tmp3, 3);

}

dirTree->current = currentNode;

}

}

else if(strcmp(cmd, "--help") == 0){

printf("사용법: cat [<옵션>]... [<파일>]...\n");

printf(" FILE(들)을 합쳐서 표준 출력으로 보낸다.\n\n");

printf(" Options:\n");

printf(" -n, --number \t number all output line\n");

printf(" -b, --number-nonblank\t number nonempty output line\n");

printf(" --help\t 이 도움말을 표시하고 끝냅니다\n");

return -1;

}

else{

str = strtok(cmd, "-");

if(str == NULL){

printf("cat: 잘못된 연산자\n");

printf("Try 'cat --help' for more information.\n");

return -1;

}

else{

printf("cat: 부적절한 옵션 -- '%s'\n", str);

printf("Try 'cat --help' for more information.\n");

return -1;

}

}

}

else{

if(strcmp(cmd, "/etc/passwd") == 0){

Concatenate(dirTree, cmd, 4);

return 0;

}

strncpy(tmp, cmd, MAX\_DIR);

if(strstr(cmd, "/") == NULL){

if(HasPermission(dirTree->current, 'w') != 0){

printf("cat: '%s'파일을 만들 수 없음: 허가거부\n", dirTree->current->name);

return -1;

}

tmpNode = IsExistDir(dirTree, cmd, 'd');

tmpNode2 = IsExistDir(dirTree, cmd, 'f');

if(tmpNode == NULL && tmpNode2 == NULL){

printf("cat: '%s': 그런 파일이나 디렉터리가 없습니다\n", cmd);

return -1;

}

else if(tmpNode != NULL && tmpNode2 == NULL){

printf("cat: '%s': 디렉터리입니다\n", cmd);

return -1;

}

else if(tmpNode2 != NULL && HasPermission(tmpNode2, 'r') != 0){

printf("cat: '%s'파일을 열 수 없음: 허가거부\n", tmpNode2->name);

return -1;

}

else{

Concatenate(dirTree, cmd, 1);

}

}

else if(tmpNode2 != NULL && HasPermission(tmpNode2, 'r') != 0){

printf("cat: '%s'파일을 열 수 없음: 허가거부\n", tmpNode2->name);

return -1;

}

else{

Concatenate(dirTree, cmd, 1);

}

}

else{

strncpy(tmp2, getDir(cmd), MAX\_DIR);

val = MovePath(dirTree, tmp2);

if(val != 0){

printf("cat: '%s': 그런 파일이나 디렉터리가 없습니다\n", tmp2);

return -1;

}

str = strtok(tmp, "/");

while(str != NULL){

strncpy(tmp3, str, MAX\_NAME);

str = strtok(NULL, "/");

}

tmpNode = IsExistDir(dirTree, tmp3, 'd');

tmpNode2 = IsExistDir(dirTree, tmp3, 'f');

if(tmpNode == NULL && tmpNode2 == NULL){

printf("cat: '%s': 그런 파일이나 디렉터리가 없습니다\n", tmp3);

dirTree->current = currentNode;

return -1;

}

else if(tmpNode != NULL && tmpNode2 == NULL){

printf("cat: '%s': 디렉터리입니다\n", tmp3);

dirTree->current = currentNode;

return -1;

}

else if(tmpNode2 != NULL && HasPermission(tmpNode2, 'r') != 0){

printf("cat: '%s'파일을 열 수 없음: 허가거부\n", tmpNode2->name);

dirTree->current = currentNode;

return -1;

}

else{

Concatenate(dirTree, tmp3, 1);

}

dirTree->current = currentNode;

}

}

return 1;

}

Cat 명령어는 작은 파일의 내용을 생성하거나 그 내용을 터미널 창으로 보여주는 함수이다.

cat 명령어 후 아무것도 입력을 하지 않으면 오류 메시지를 출력한다.

cat > ‘filename’ 을 입력하면 ‘filename’의 이름을 가진 파일을 생성한다.

먼저 HasPermission 함수를 통해 쓰기 권한이 있는지 확인한다. 만약 쓰기 권한이 없다면 오류 메시지로 허가거부 메시지를 띄우고 쓰기 권한이 있다면 IsExistDir(dirTree, str, ‘d’)함수를 실행시킨다.

이를 통해 type이 ‘d’라면 디렉토리인지 파일인지 확인한다 IsExistDir의 세번째 인수인 type이 ‘d’이므로 만약 이 값이 존재한다면 이 이름은 디렉토리를 나타낸다.

Cat은 파일의 내용을 다루는 함수이기 때문에 결국 IsExistDir의 값이 NULL인 경우를 찾아야 한다. (찾으려는 이름의 파일 이 있거나 파일이 아예 없는 경우) 값이 NULL이라면 Concatenate(dirTree, str, 0)함수를 실행시킨다. 그 후 겹치는 이름의 파일 혹은 디렉터리가 없다면 텍스트 파일을 생성한다.

앞에 코드 부분이 일반경로에 대해서 파일을 생성한다면, 지금 소개하는 부분인 절대, 상대경로에 대해서 파일을 생성할 수 있도록 구현하였다.

getDir함수는 앞에서 소개하였고 이것을 통해 상대경로에서도 파일을 생성할 수 있다.

그 외로 여러 예외처리를 구현하였다. 여기서 이용된 Concatenate함수는 뒤에서 설명하겠다.

cat –n : 출력에 줄 번호를 덧붙인다. 다른 알고리즘과 동일하게 IsExistDIr()함수를 통해 이미 존재하는 파일 또는 디렉토리인지 확인을 한다. 마지막 else if문에서는 파일이 존재하고 읽기 권한이 없는 경우를 뜻하고 이 경우 ‘허가 거부’라는 메시지를 출력한다. 일반경로에도 적용은 물론 Mkdir과 rm 뿐만 아니라 cat에도 getDir() 함수를 사용함으로써 절대경로/상대경로에도 적용하도록 하였다.

cat –b : 공백부분을 제외하고 출력에 줄 번호를 덧붙인다. 위의 알고리즘과 동일한 원리를 이용하였다.

마지막으로 cat /etc/passwd를 입력한 경우는 이는 사용자 정보와 그룹정보를 알 수 있게 해주는 옵션이다. 마찬가지로 IsExistDir() 함수를 통해 디렉토리인지 파일인지 구분하고, HasPermission()함수를 통해 읽기 권한이 있는지 판별한다.

-Concatenate() 함수

int Concatenate(DirectoryTree\* dirTree, char\* fName, int o)

{

UserNode\* tmpUser = NULL;

TreeNode\* tmpNode = NULL;

FILE\* fp;

char buf[MAX\_BUFFER];

char tmpName[MAX\_NAME];

char\* str;

int tmpSIZE = 0;

int cnt = 1;

if(val != 0){

printf("cat: '%s': 그런 파일이나 디렉터리가 없습니다\n", tmp2);

return -1;

}

str = strtok(tmp, "/");

while(str != NULL){

strncpy(tmp3, str, MAX\_NAME);

str = strtok(NULL, "/");

}

tmpNode = IsExistDir(dirTree, tmp3, 'd');

tmpNode2 = IsExistDir(dirTree, tmp3, 'f');

if(tmpNode == NULL && tmpNode2 == NULL){

printf("cat: '%s': 그런 파일이나 디렉터리가 없습니다\n", tmp3);

dirTree->current = currentNode;

return -1;

}

else if(tmpNode != NULL && tmpNode2 == NULL){

printf("cat: '%s': 디렉터리입니다\n", tmp3);

dirTree->current = currentNode;

return -1;

}

else if(tmpNode2 != NULL && HasPermission(tmpNode2, 'r') != 0){

printf("cat: '%s'파일을 열 수 없음: 허가거부\n", tmpNode2->name);

dirTree->current = currentNode;

return -1;

}

else{

Concatenate(dirTree, tmp3, 1);

}

dirTree->current = currentNode;

}

}

return 1;

}

//file read

if(o != 0){

if(o == 4){

tmpUser = usrList->head;

while(tmpUser != NULL){

printf("%s:x:%d:%d:%s:%s\n", tmpUser->name, tmpUser->UID, tmpUser->GID, tmpUser->name, tmpUser->dir);

tmpUser = tmpUser->LinkNode;

}

return 0;

}

tmpNode = IsExistDir(dirTree,fName, 'f');

if(tmpNode == NULL){

return -1;

}

fp = fopen(fName, "r");

while(feof(fp) == 0){

fgets(buf, sizeof(buf), fp);

if(feof(fp) != 0){

break;

}

if(o == 2){

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

printf(" %d ",cnt);

cnt++;

}

}

else if(o == 3){

if(buf[strlen(buf)-1] == '\n' && buf[0] != '\n'){

printf(" %d ",cnt);

cnt++;

}

}

fputs(buf, stdout);

}

fclose(fp);

}

else{

fp = fopen(fName, "w");

char ch;

int test = 33;

while(1)

{

ch = getchar();

if(ch == test)

{

getchar();

break;

}

buf[tmpSIZE++] = ch;

}

fputs(buf, fp);

fclose(fp);

tmpNode = IsExistDir(dirTree, fName, 'f');

if(tmpNode != NULL){

time(&ltime);

today = localtime(&ltime);

tmpNode->month = today->tm\_mon + 1;

tmpNode->day = today->tm\_mday;

tmpNode->hour = today->tm\_hour;

tmpNode->minute = today->tm\_min;

}

else{

MakeDir(dirTree, fName, 'f');

}

tmpNode = IsExistDir(dirTree, fName, 'f');

tmpNode->SIZE = tmpSIZE;

}

return 0;

}

fopen 함수를 통해 쓰기 권한으로 파일을 불러온 후에 파일에 키보드로 입력 받은 값을 넣는다. 입력을 한 후 입력을 종료하려면 리눅스 상에서 “!”를 입력하여 종료하면 된다. 이때, 느낌표가 포함된 줄은 파일에 입력되지 않는다.

Cat “>” 옵션은 만약 파일이 있다면 기존 파일에 내용을 덧씌우고, 파일이 없다면 새로 만들어 주는 함수이다.

그렇기 때문에 다시 IsExistDir(dirTree, fName, 'f') 함수를 실행시켜 리눅스 상에 같은 이름의 파일이 있는지 확인한다.

만약 있다면 시간 값들을 최신화 시켜주고, 파일이 없다면 MakeDir(dirTree, fName, 'f');를 통해 새로운 파일을 만들어 준다.

“-n” 옵션은 파일을 줄 별로 번호를 메겨 출력하기, 옵션이 없고 텍스트 파일의 이름만 있으면 그냥 텍스트 파일 출력하기를 실행한다.

“-b”옵션은 “-n” 옵션과는 달리 buf[0] != ‘\n’부분을 if문에 넣어 공백부분을 제외하고 줄 번호를 출력하게 하였다.

위 코드에서 else부분은 우리가 궁극적으로 찾고자 하는 부분으로, 읽기 권한이 있어야 하고, 디렉토리가 아닌 파일을 찾아야 한다.

tmpUser 를 유저리스트의 헤드로 설정한다. 그리고 tmpUser의 이름,UID,GID,이름,디렉터리 순으로 출력을 한 후 tmpUser를 linknode를 통해 연결한다.

또한 cat에 다른 옵션에 부여한 것과 마찬가지로 절대경로와 상대경로에도 적용이 될 수 있도록 getDir() 함수를 사용하였다.

* cp

int mycp(DirectoryTree\* dirTree,char\* sName, char\* oName){

//printf("source : %s\n",sName);

//printf("object : %s\n\n",oName);

char buf[1024];

int in, out;

int nread;

if(access(sName,F\_OK) != 0) {

printf("원본 파일이 존재하지 않습니다.\n");

return -1;

}

if(strcmp(sName,oName) == 0) {

printf("원본과 대상이 같습니다.\n");

return -1;

}

in = open(sName,O\_RDONLY); //원본파일

out = open(oName, O\_WRONLY| O\_CREAT, S\_IRUSR| S\_IWUSR);//만들파일

nread = read(in,buf,sizeof(buf)); //읽은만큼 nread가 올라간다

write(out,buf,nread); //read만큼 쓴다.

MakeDir(dirTree, oName, 'f');

return 0;

}

cp 명령어는 원하는 파일을 복사할 수 있도록 하는 명령어이다. 먼저 cp명령어를 통해 복사 할 원본 파일과 그 결과로 얻을 파일을 입력 받은 후 이를 출력하여 확인한다. 이후 원본 파일이 존재하는지 파악하기 위해 access를 통해 원본 파일을 확인하고 strcmp를 통해 원본 파일과 결과로 얻을 파일을 비교하여 같은 파일인지 확인한다. open 함수를 이용해 파일이 없을 경우 새로 생성하고 이 과정이 진행된 후 write 함수를 사용해 buf 인자에 담긴 내용을 파일에 저장하고 nread에 기록된 byte를 저장한다. 이후 생성한 Makedir를 이용해 파일의 형식인 f로 파일을 만든다.

* Find

int find\_(DirectoryTree\* dirTree, char\* cmd)

{

char\* str;

if(cmd == NULL){

FindDir(dirTree, dirTree->current->name, 1);

return 0;

}

else if(cmd[0] == '-'){

if(strcmp(cmd, "-name") == 0){

str = strtok(NULL, " ");

if(str == NULL){

printf("find: Wrong Operator \n");

printf("Try 'find --help' for more information.\n");

return -1;

}

FindDir(dirTree, str, 0);

}

else if(strcmp(cmd, "--help") == 0){

printf("use: find [<option>]... [<file>]...\n");

printf("\n");

printf(" Options:\n");

printf(" -name\t finds file by name\n");

printf(" --help\t Display help and exit.\n");

return -1;

}

else{

str = strtok(cmd, "-");

if(str == NULL){

printf("find: Wrong Operator\n");

printf("Try 'find --help' for more information.\n");

return -1;

}

else{

printf("find: Wrong option -- '%s'\n", str);

printf("Try 'find --help' for more information.\n");

return -1;

}

}

}

else{

FindDir(dirTree, cmd, 1);

}

return 0;

}

-FindDir 함수

void FindDir(DirectoryTree\* dirTree, char\* dirName, int o)

{

char tmp[MAX\_LENGTH];

Dir = fopen("Directory.txt", "r");

while(fgets(tmp, MAX\_LENGTH, Dir) != NULL){

ReadDir(dirTree, tmp, dirName, o);

}

fclose(Dir);

}}

-ReadDir 함수

int ReadDir(DirectoryTree\* dirTree, char\* tmp, char\* dirName, int o)

{

char\* str;

char str2[MAX\_NAME];

if(o == 0){

str = strtok(tmp, " ");

strcpy(str2, str);

for(int i=0;i<10;i++){

str = strtok(NULL, " ");

}

if(str != NULL){

if(strstr(str2, dirName) != NULL){

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

if(strcmp(str, "/") == 0)

printf("/%s\n", str2);

else

printf("%s/%s\n", str, str2);

}

}

}

else{

str = strtok(tmp, " ");

strcpy(str2, str);

for(int i=0;i<10;i++){

str = strtok(NULL, " ");

}

if(str != NULL){

if(strstr(str, dirName) != NULL){

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

if(strcmp(str, "/") == 0)

printf("/%s\n", str2);

else

printf("%s/%s\n", str, str2);

}

}

}

return 0;

}}

Find 명령어는 사용법에 따라 찾고자 하는 파일을 디렉토리에서 검색할 수 있게 해주는 명령어 이다. Find를 사용할 때 옵션으로는 이름을 통해서 파일을 찾도록 했다. 이 명령어를 구현하기 위해 FindDir함수와 ReadDir함수를 구현했는데 먼저 FindDir함수는 Directory.txt 파일을 열어서 줄 단위로 반복하여 ReadDir함수를 호출한다. 호출된 ReadDir함수는 FindDir함수에서 사용했던 Directory.txt파일의 내용을 띄어쓰기 단위로 str변수에 저장한 후 새로운 함수를 만들어 원하는 파일의 경로를 출력하도록 하고, 이 과정을 통해 원하는 파일과 디렉토리를 찾을 수 있다.

**(2) 추가 명령어**

* Grep

void grep(char \*searching\_word,char \*f\_name){

int i=1;

char output\_line[MAX\_LENGTH];

FILE \*fp = fopen(f\_name,"rt");

if(fp==NULL)

printf("cannot read the file\n");

while(1){

if(feof(fp))

break;

else

fgets(output\_line,sizeof(output\_line),fp);

i++;

}

FILE \*fp2=NULL;

fp2=fopen(f\_name,"rt");

for(int j=1;j<i-1;j++){

fgets(output\_line,sizeof(output\_line),fp2);

if(strstr(output\_line,searching\_word)!=NULL)

printf("%s",output\_line);

}

fclose(fp);

}}

-grep\_n 함수

void grep\_n(char \*searching\_word,char \*f\_name){

int i=1;

char output\_line[MAX\_LENGTH];

FILE \*fp = fopen(f\_name,"rt");

if(fp==NULL)

printf("cannot read the file\n");

while(1){

if(feof(fp))

break;

else

fgets(output\_line,sizeof(output\_line),fp);

i++;

}

FILE \*fp2=NULL;

fp2=fopen(f\_name,"rt");

for(int j=1;j<i-1;j++){

fgets(output\_line,sizeof(output\_line),fp2);

if(strstr(output\_line,searching\_word)!=NULL)

printf("%d:%s",j,output\_line);

}

fclose(fp);

}

Grep 명령어는 파일에서 원하는 문자열이 들어간 행을 찾아서 출력하는 역할로 특정 패턴을 입력 받아 지정 파일을 검색한다. 이는 grep과 grep\_n으로 구분하여 구현했는데 grep은 searching과 f\_name을 통해서 찾고자 하는 문자열과 문자열이 속해 있는 행을 찾는 방식으로 진행하도록 했다. FILE \*fp는 파일을 열면 내부에 있는 행을 하나씩 읽어 들여서 찾고자 하는 문자열이 일치하는 행이 있을 때 그 행을 출력하도록 했다. grep\_n에서는 grep과 비슷한 형태로 진행하고 추가로 fgets를 통해 찾고자 하는 문자열이 있는 행의 행번호를 출력한다.

* Chmode

int chmod\_(DirectoryTree\* dirTree, char\* cmd)

{

TreeNode\* tmpNode = NULL;

char\* str;

int tmp;

if(cmd == NULL){

printf("chmod: Wrong Operator\n");

printf("Try 'chmod --help' for more information.\n");

return -1;

}

if(cmd[0] == '-'){

if(strcmp(cmd, "-R") == 0){

str = strtok(NULL, " ");

if(str == NULL){

printf("chmod: Wrong Operator\n");

printf("Try 'chmod --help' for more information.\n");

return -1;

}

if(str[0]-'0'<8 && str[1]-'0'<8 && str[2]-'0'<8 && strlen(str)==3){

tmp = atoi(str);

str = strtok(NULL, " ");

if(str == NULL){

printf("chmod: Wrong Operator\n");

printf("Try 'chmod --help' for more information.\n");

return -1;

}

tmpNode = IsExistDir(dirTree, str, 'd');

if(tmpNode != NULL){

if(tmpNode->LeftChild == NULL)

ChangeMode(dirTree, tmp, str);

else{

ChangeMode(dirTree, tmp, str);

ChangeModeAll(tmpNode->LeftChild, tmp);

}

}

else{

printf("chmod: '%s': No file or directory.\n", str);

return -1;

}

}

else{

printf("chmod: Wrong Mode: '%s'\n", str);

printf("Try 'chmod --help' for more information.\n");

return -1;

}

}

else if(strcmp(cmd, "--help") == 0){

printf("사용법: chmod [옵션]... 8진수-MODE... 디렉터리...\n");

printf(" Change the mode of each FILE to MODE.\n\n");

printf(" Options:\n");

printf(" -R, --recursive\t change files and directories recursively\n");

printf(" --help\t 이 도움말을 표시하고 끝냅니다\n");

return -1;

}

else{

str = strtok(cmd, "-");

if(str == NULL){

printf("chmod: 잘못된 연산자\n");

printf("Try 'chmod --help' for more information.\n");

return -1;

}

else{

printf("chmod: 부적절한 옵션 -- '%s'\n", str);

printf("Try 'chmod --help' for more information.\n");

return -1;

}

}

}

else{

if(cmd[0]-'0'<8 && cmd[1]-'0'<8 && cmd[2]-'0'<8 && strlen(cmd)==3){

tmp = atoi(cmd);

str = strtok(NULL, " ");

if(str == NULL){

printf("chmod: 잘못된 연산자\n");

printf("Try 'chmod --help' for more information.\n");

return -1;

}

ChangeMode(dirTree, tmp, str);

}

else{

printf("chmod: 잘못된 모드: '%s'\n", cmd);

printf("Try 'chmod --help' for more information.\n");

return -1;

}

}

return 0;

}printf(" -R, --recursive\t change files and directories recursively\n");

printf(" --help\t Display help and exit.\n");

return -1;

}

else{

str = strtok(cmd, "-");

if(str == NULL){

printf("chmod: Wrong Operator\n");

printf("Try 'chmod --help' for more information.\n");

return -1;

}

else{

printf("chmod: Wrong option -- '%s'\n", str);

printf("Try 'chmod --help' for more information.\n");

return -1;

}

}

}

else{

if(cmd[0]-'0'<8 && cmd[1]-'0'<8 && cmd[2]-'0'<8 && strlen(cmd)==3){

tmp = atoi(cmd);

str = strtok(NULL, " ");

if(str == NULL){

printf("chmod: Wrong Operator\n");

printf("Try 'chmod --help' for more information.\n");

return -1;

}

ChangeMode(dirTree, tmp, str);

}

else{

printf("chmod: Wrong mode: '%s'\n", cmd);

printf("Try 'chmod --help' for more information.\n");

return -1;

}

}

return 0;

}

else if(strcmp(cmd, "--help") == 0){

printf("how to use: chmod [option]... Octal-MODE... directory...\n");

printf(" Change the mode of each FILE to MODE.\n\n");

printf(" Options:\n");

printf(" -R, --recursive\t change files and directories recursively\n");

printf(" --help\t Display help and exit.\n");

return -1;

}

else{

str = strtok(cmd, "-");

if(str == NULL){

printf("chmod: Wrong Operator\n");

printf("Try 'chmod --help' for more information.\n");

return -1;

}

else{

printf("chmod: Wrong option -- '%s'\n", str);

printf("Try 'chmod --help' for more information.\n");

return -1;

}

}

}

else{

if(cmd[0]-'0'<8 && cmd[1]-'0'<8 && cmd[2]-'0'<8 && strlen(cmd)==3){

tmp = atoi(cmd);

str = strtok(NULL, " ");

if(str == NULL){

printf("chmod: Wrong Operator\n");

printf("Try 'chmod --help' for more information.\n");

return -1;

}

ChangeMode(dirTree, tmp, str);

}

else{

printf("chmod: Wrong Mode: '%s'\n", cmd);

printf("Try 'chmod --help' for more information.\n");

return -1;

}

}

return 0;

}

if(cmd[0]-'0'<8 && cmd[1]-'0'<8 && cmd[2]-'0'<8 && strlen(cmd)==3){

tmp = atoi(cmd);

str = strtok(NULL, " ");

if(str == NULL){

printf("chmod: Wrong Operator\n");

printf("Try 'chmod --help' for more information.\n");

return -1;

}

ChangeMode(dirTree, tmp, str);

}

else{

printf("chmod: Wrong mode: '%s'\n", cmd);

printf("Try 'chmod --help' for more information.\n");

return -1;

}

}

return 0;

}

-ChangeMode함수

int ChangeMode(DirectoryTree\* dirTree, int mode, char\* dirName)

{

TreeNode\* tmpNode = NULL;

TreeNode\* tmpNode2 = NULL;

tmpNode = IsExistDir(dirTree, dirName, 'd');

tmpNode2 = IsExistDir(dirTree, dirName, 'f');

if(tmpNode != NULL){

if(HasPermission(tmpNode, 'w') != 0){

printf("chmod: '%s' Unable to modify file: Permission denied \n", dirName);

return -1;

}

tmpNode->mode = mode;

Mode2Permission(tmpNode);

}

else if(tmpNode2 != NULL){

if(HasPermission(tmpNode2, 'w') != 0){

printf("chmod: '%s' Unable to modify file: Permission denied \n", dirName);

return -1;

}

tmpNode2->mode = mode;

Mode2Permission(tmpNode2);

}

else{

printf("chmod: '%s unable to acess: No file or directory\n", dirName);

return -1;

}

return 0;

}}

-ChangeModeAll함수

void ChangeModeAll(TreeNode\* dirNode, int mode)

{

if(dirNode->RightSibling != NULL){

ChangeModeAll(dirNode->RightSibling, mode);

}

if(dirNode->LeftChild != NULL){

ChangeModeAll(dirNode->LeftChild, mode);

}

dirNode->mode = mode;

Chmod명령어는 파일 및 디렉토리의 접근 권한을 변경시키는 역할로 옵션 -R을 통해 하위 디렉토리 및 파일들의 접근 권한을 전부 변경한다. 먼저 예외처리를 진행한 후에 tmpNode를 통해 디렉토리의 유효성을 판단하고 changeMode와 changeModeAll함수를 이용해 chmode를 수행한다. changeMode함수는 isExitDir를 통해서 디렉토리인지 파일인지를 구분해주고 Mode2permission함수를 통해 값을 숫자로 변경한다. 변경된 숫자는 temp %2 를 이용해 0과1로 나타낸다. changemodeAll함수는 하위에 존재하는 모든 파일의 권한을 변경하고 해당 디렉토리의 하위의 파일이나 디렉토리의 접근 권한을 전부 바꾸는 옵션이 NULL이 될 때까지 재귀함수를 통해 진행한다.

* Chown

int chown\_(DirectoryTree\* dirTree, char\* cmd)

{

TreeNode\* tmpNode = NULL;

UserNode\* tmpUser = NULL;

char\* str;

char tmp[MAX\_NAME];

if(cmd == NULL){

printf("chown: Wrong Operator\n");

printf("Try 'chown --help' for more information.\n");

return -1;

}

if(cmd[0] == '-'){

if(strcmp(cmd, "-R") == 0){

str = strtok(NULL, " ");

if(str == NULL){

printf("chown: Wrong Operator\n");

printf("Try 'chown --help' for more information.\n");

return -1;

}

tmpUser = IsExistUser(usrList, str);

if(tmpUser != NULL){

strncpy(tmp, str, MAX\_NAME);

}

else{

printf("chown: Wrong User: '%s'\n", str);

printf("Try 'chown --help' for more information.\n");

return -1;

}

str = strtok(NULL, " ");

if(str == NULL){

printf("chown: Wrong Operator\n");

printf("Try 'chown --help' for more information.\n");

return -1;

}

tmpNode = IsExistDir(dirTree, str, 'd');

if(tmpNode != NULL){

if(tmpNode->LeftChild == NULL)

ChangeOwner(dirTree, tmp, str);

else{

ChangeOwner(dirTree, tmp, str);

ChangeOwnerAll(tmpNode->LeftChild, tmp);

}

}

else{

printf("chown: '%s': 그런 파일이나 디렉터리가 없습니다\n", str);

return -1;

}

}

else if(strcmp(cmd, "--help") == 0){

printf("사용법: chown [옵션]... [소유자]... 파일...\n");

printf(" Change the owner and/or group of each FILE to OWNER and/or GROUP.\n\n");

printf(" Options:\n");

printf(" -R, --recursive\t change files and directories recursively\n");

printf(" --help\t 이 도움말을 표시하고 끝냅니다\n");

return -1;

}

else{

str = strtok(cmd, "-");

if(str == NULL){

printf("chown: 잘못된 연산자\n");

printf("Try 'chown --help' for more information.\n");

return -1;

}

else{

printf("chown: 부적절한 옵션 -- '%s'\n", str);

printf("Try 'chown --help' for more information.\n");

return -1;

}

}

}

else{

strncpy(tmp, cmd, MAX\_NAME);

str = strtok(NULL, " ");

if(str == NULL){

printf("chown: 잘못된 연산자\n");

printf("Try 'chown --help' for more information.\n");

return -1;

}

else{

ChangeOwner(dirTree, tmp, str);

}

}

return 0;

}

else if(strcmp(cmd, "--help") == 0){

printf("how to use: chown [option]... [user]... file...\n");

printf(" Change the owner and/or group of each FILE to OWNER and/or GROUP.\n\n");

printf(" Options:\n");

printf(" -R, --recursive\t change files and directories recursively\n");

printf(" --help\t Dispaly help and exit.\n");

return -1;

}

else{

str = strtok(cmd, "-");

if(str == NULL){

printf("chown: Wrong Operator\n");

printf("Try 'chown --help' for more information.\n");

return -1;

}

else{

printf("chown: Wrong option -- '%s'\n", str);

printf("Try 'chown --help' for more information.\n");

return -1;

}

}

}

else{

strncpy(tmp, cmd, MAX\_NAME);

str = strtok(NULL, " ");

if(str == NULL){

printf("chown: Wrong Operator\n");

printf("Try 'chown --help' for more information.\n");

return -1;

}

else{

ChangeOwner(dirTree, tmp, str);

}

}

return 0;

}

if(tmpNode->LeftChild == NULL)

ChangeOwner(dirTree, tmp, str);

else{

ChangeOwner(dirTree, tmp, str);

ChangeOwnerAll(tmpNode->LeftChild, tmp);

}

}

else{

printf("chown: '%s': No file or directory.\n", str);

return -1;

}

}

else if(strcmp(cmd, "--help") == 0){

printf("how to use: chown [option]... [user]... file...\n");

printf(" Change the owner and/or group of each FILE to OWNER and/or GROUP.\n\n");

printf(" Options:\n");

printf(" -R, --recursive\t change files and directories recursively\n");

printf(" --help\t Display help and exit.\n");

return -1;

}

else{

str = strtok(cmd, "-");

if(str == NULL){

printf("chown: Wrong Operator\n");

printf("Try 'chown --help' for more information.\n");

return -1;

}

else{

printf("chown: Wrong option-- '%s'\n", str);

printf("Try 'chown --help' for more information.\n");

return -1;

}

}

}

else{

strncpy(tmp, cmd, MAX\_NAME);

str = strtok(NULL, " ");

if(str == NULL){

printf("chown: Wrong Operator\n");

printf("Try 'chown --help' for more information.\n");

return -1;

}

else{

ChangeOwner(dirTree, tmp, str);

}

}

return 0;

}

else if(strcmp(cmd, "--help") == 0){

printf("how to use: chown [option]... [user]... file...\n");

printf(" Change the owner and/or group of each FILE to OWNER and/or GROUP.\n\n");

printf(" Options:\n");

printf(" -R, --recursive\t change files and directories recursively\n");

printf(" --help\t Dispaly help and exit.\n");

return -1;

}

else{

str = strtok(cmd, "-");

if(str == NULL){

printf("chown: Wrong Operator\n");

printf("Try 'chown --help' for more information.\n");

return -1;

}

else{

printf("chown: Wrong option -- '%s'\n", str);

printf("Try 'chown --help' for more information.\n");

return -1;

}

}

}

else{

strncpy(tmp, cmd, MAX\_NAME);

str = strtok(NULL, " ");

if(str == NULL){

printf("chown: Wrong Operator\n");

printf("Try 'chown --help' for more information.\n");

return -1;

}

else{

ChangeOwner(dirTree, tmp, str);

}

}

return 0;

}

-ChangeOwnerAll 함수

void ChangeOwnerAll(TreeNode\* dirNode, char\* userName)

{

UserNode\* tmpUser = NULL;

tmpUser = IsExistUser(usrList, userName);

if(dirNode->RightSibling != NULL){

ChangeOwnerAll(dirNode->RightSibling, userName);

}

if(dirNode->LeftChild != NULL){

ChangeOwnerAll(dirNode->LeftChild, userName);

}

dirNode->UID = tmpUser->UID;

dirNode->GID = tmpUser->GID;

}

-ChangeOwner 함수

int ChangeOwner(DirectoryTree\* dirTree, char\* userName, char\* dirName)

{

TreeNode\* tmpNode = NULL;

TreeNode\* tmpNode2 = NULL;

UserNode\* tmpUser = NULL;

tmpNode = IsExistDir(dirTree, dirName, 'd');

tmpNode2 = IsExistDir(dirTree, dirName, 'f');

if(tmpNode != NULL){

if(HasPermission(tmpNode, 'w') != 0){

printf("chown: '%s' Unable to modify file : Permission denied\n", dirName);

return -1;

}

tmpUser = IsExistUser(usrList, userName);

if(tmpUser != NULL){

tmpNode->UID = tmpUser->UID;

tmpNode->GID = tmpUser->GID;

}

else{

printf("chown: Wrong User: '%s'\n", userName);

printf("Try 'chown --help' for more information.\n");

return -1;

}

}

else if(tmpNode2 != NULL){

if(HasPermission(tmpNode2, 'w') != 0){

printf("chown: '%s' Unable to modify file: Permission denied \n", dirName);

return -1;

}

tmpUser = IsExistUser(usrList, userName);

if(tmpUser != NULL){

tmpNode2->UID = tmpUser->UID;

tmpNode2->GID = tmpUser->GID;

}

else{

printf("chown: 잘못된 사용자: '%s'\n", userName);

printf("Try 'chown --help' for more information.\n");

return -1;

}

}

else{

printf("chown: '%s'에 접근할 수 없습니다: 그런 파일이나 디렉터리가 없습니다\n", dirName);

return -1;

}

return 0;

}

if(tmpUser != NULL){

tmpNode2->UID = tmpUser->UID;

tmpNode2->GID = tmpUser->GID;

}

else{

printf("chown: Wrong User: '%s'\n", userName);

printf("Try 'chown --help' for more information.\n");

return -1;

}

}

else{

printf("chown: '%s'unable to access: No file or directory\n", dirName);

return -1;

}

return 0;

}

Chown 명령어는 파일 및 디렉토리의 소유권 권한을 변경시키는 역할로 옵션은 -R을 통해 하위 디렉토리 및 파일들의 접근 권한을 전부 변경한다. 처음 입력되는 user가 존재할 경우 tmp에 정보를 담아 놓는다. 접근 권한을 바꾸고자 하는 디렉토리에 user가 존재할 경우 changeowner 와 changeownerall 함수를 통해 권한을 변경한다. Changeowner함수는 디렉토리와 파일을 구분하여 저장하고, changownerall함수는 UID와 GID를 복사한다. 디렉토리의 접근 권한을 바꿀 수 없을 경우 파일 수정이 불가하다는 메시지를 띄우고 오류가 발생할 경우 접근이 불가하다는 메시지를 띄운다.

**5. 자원 소요 계획**

(1) 프로젝트 팀원 역할 소개 및 참여기여도

|  |  |  |
| --- | --- | --- |
| 이름 | 역할 | 참여기여도 |
| 박준서 | pwd/ls, Multi threading 구현 | 20% |
| 송하은 | cp/find구현, 추가명령어, 보고서 | 20% |
| 우신영 | 팀장, rm/cat 구현, 보고서, 발표 | 20% |
| 정재우 | cd/mkdir, Multi threading 구현 | 20% |
| 한동원 | main 구현, ppt 작성 | 20% |

(2) 프로젝트 추진 단계

|  |  |
| --- | --- |
| 단계 | 세부사항 |
| 프로젝트 파악, 계획 | 프로젝트의 목적 이해 및 계획 설정 |
| 역할분담 | 구체적인 명령어 구현 역할 분담 |
| 자료조사 | 실습시간 자료 및 외부자료 조사 |
| 프로그램 설계 | C 프로그래밍을 통한 리눅스 명령어 구현 |
| 보고서 및 발표 자료 작성 | 자료 취합 및 ppt 제작 |
| 제출 발표 | 프로그램 발표 및 시연 |

(3) 전체 추진 일정

|  |  |
| --- | --- |
| 기간 | 추진 내용 |
| 1주차 | 프로젝트 계획 및 역할분담 |
| 2주차 | EC2 연결 및 자료 조사, 명령어 구현 |
| 3주차 | 멀티스레딩 구현 및 추가 명령어 구현 |
| 4주차 | 보고서 작성 및 발표 준비 |

(4) 회의일지

|  |  |
| --- | --- |
| 1차 | |
| 일시 | 2022.05.10. 20:00 |
| 장소 | Webex 미팅룸 |
| 참여자 | 박준서, 송하은, 우신영, 정재우, 한동원 |
| 회의 내용 | 1. 팀장 선정 2. 자료 공유 3. 역할 분담: 각자 구현할 명령어 분배 |

|  |  |
| --- | --- |
| 2차 | |
| 일시 | 2022.05.19. 11:00 |
| 장소 | 신공학관 3165 |
| 참여자 | 박준서, 송하은, 우신영, 정재우, 한동원 |
| 회의 내용 | 1. EC2 생성 및 연결 2. 구현 부분 확인 3. 자료 취합 및 선정 |

|  |  |
| --- | --- |
| 3차 | |
| 일시 | 2022.05.26. 11:00 |
| 장소 | 신공학관 3165 |
| 참여자 | 박준서, 송하은, 우신영, 정재우, 한동원 |
| 회의 내용 | 1. 추가 기능 정하기 2. 발표, ppt 역할 정하기 3. 명령어 취합 4. 코드 정리 |

|  |  |
| --- | --- |
| 4차 | |
| 일시 | 2022.05.31 20:00 |
| 장소 | 신공학관 3165 |
| 참여자 | 박준서, 송하은, 우신영, 정재우, 한동원 |
| 회의 내용 |  |

**6. 참고문헌**