컴퓨터학부 20162518 최승서

1. **개요**
   1. **myls.c**

myls.c는 리눅스 시스템 상의 ls명령어와 동일한 기능을 수행하는 파일 목록과 정보를 출력하는 프로그램이다. 기본적으로 myls, myls file\_name, myls dir\_name이 구현이 되어 있으며 -i, -l, -t 옵션이 구현이 되어 있다 추가로 -a 옵션 또한 구현이 되어 있다. 또한 이 4가지 옵션을 중첩하여 원하는 출력결과물을 얻을 수 있다.

* **int main(int argc, int argv[]);**

main함수는 myls 명령 실행 시 사용자로부터 인자들을 입력을 받는 기능을 한다. 사용자로부터 입력 받은 인자를 unistd.h의 getopt함수를 이용하여 옵션을 확인한다. option 구조체의 구현된 i,l,t,a 옵션의 변수를 설정하여 해당 옵션이 입력 되었을 경우 해당 옵션의 변수의 값을 1로 변경한다. 만약 아무 인자도 입력하지 않았을 경우 현재 디렉터리의 파일들을 출력하기 위해 argv[optind]에 현재 디렉터리를 의미하는 “.”을 넣어준다. 다음으로 getFileStat()으로 stat 구조체 포인터, 파일들의 이름을 저장하기 위한 변수 statName, 입력 받은 argv[optind]를 넘겨주고 파일들의 stat구조체 정보를 받아온다. 또한 읽어온 stat 구조체의 개수를 반환하여 준다.

다음으로 받아온 stat 구조체 정보를 기반으로 sortStat() 함수를 호출하여 파일들을 정렬하는데 이때 인자로 opt\_t(t 옵션의 변수)를 넘겨주어 t 옵션이 들어왔을 경우 수정 시간 기준으로 정렬하고, t 옵션이 없을 경우 파일의 이름 순으로 정렬을 한다. 정렬이 끝나면 각 옵션에 맞추어 파일 정보를 출력한다.

* **int getFileStat(strcut stat \*info, char \*\*statName, char \*input);**

getFileStat()은 stat 구조체 포인터, 파일들의 이름을 저장할 char형 \*\*statName, 탐색할 파일 혹은 디렉터리 이름을 main으로부터 넘겨 받아 해당하는 파일 혹은 디렉터리의 정보를 stat 구조체와 char형 \*\* s tatName 변수에 저장을 하는 기능을 수행한다.

먼저 is\_dir()함수를 호출하여 input으로 들어온 파일이 디렉터리인지 일반 파일인지 확인하여 return 값이 1일 경우 디렉터리를 열어서 해당 디렉터리의 모든 파일 엔트리를 읽어온다. 읽어온 각 파일 별로 lstat()함수를 호출하여 해당 파일의 stat 구조체 정보를 읽어오고 statName변수에 파일 이름을 저장한다. 또한 읽어와 -l 옵션이 들어왔을 경우 total을 출력하기 위해 읽어온 stat 구조체에서 파일들의 블럭 개수를 저장한다. 만약 디렉터리가 아니라면 lstat()함수를 1번만 실행하여 해당 파일의 stat구조체 정보를 받아온다. 모든 프로세스가 끝나면 getFileStat()함수는 읽어온 파일 개수를 return한다.

* **int is\_dir(char \*fname);**

is\_dir() 함수는 파일 이름을 입력 받아 해당 파일이 디렉터리인지 확인을 하는 함수이다.

인자로 넘어온 파일의 이름으로 lstat()함수를 호출하여 stat 구조체 정보를 받아와서. stat 구조체 맴버 변수인 st\_mode를 S\_ISDIR()이라는 매크로 함수를 통해서 디렉터리인지 확인하여 값을 리턴한다.

* **void mode\_to\_str(mode\_t mode, char \*buf);**

mode\_to\_str()함수는 stat구조체로부터 받아온 숫자 형태의 파일의 모드 값을 사람이 알아볼 수 있는 string 형태로 바꾸어 주는 함수이다.

먼저 인자로 받아온 mode와 어떠한 파일인지 확인해주는 S\_IFMT와 & 연산을 하여 원하는 bit를 뽑아낸다. 그렇게 뽑아낸 값을 switch문을 이용하여 해당하는 파일 종류를 char형 버퍼의 0번째 인덱스에 담아준다. 그 다음 비트 연산을 하여 User, Group, Other별로 해당 파일의 읽기, 쓰기, 실행 권한이 있는지 확인하여 char형 버퍼에 순서대로 저장한다.

다음으로 mode의 user id, group id, sticky bit을 확인하여 해당 bit값이 활성화 되어 있으면 각각 user, group, other의 x권한을 s, S, t, T의 문자들로 알맞게 바꾸어준다.

* **void printLmode(char \*name, struct stat \*info);**printLmode()는 -l 옵션이 들어왔을 때 파일 목록을 출력해주는 함수이다. 기본적으로 ls 명령에서의 -l을 주었을 때 출력 결과와 동일하게 출력이 된다. 파일 권한, 하드 링크 수, 소유자 이름, 소유자 그룹 이름, 파일 크기, 수정 날짜, 파일 이름이 출력되며 각 파일의 종류 별로 터미널에서 출력되는 색깔이 다르다. 파일 별로 다른 색깔을 출력하는 코드는 check\_color()함수를 호출하여 출력 색깔을 결정한다. 출력하는 파일이 symbolic link 파일일 경우 해당 파일이 링크되어 있는 원본 파일의 경로 또한 같이 출력된다.
* **void printNormal(char \*name, struct stat \*info);**

printNormal은 -l 옵션이 없을 경우 일반적으로 파일 이름들만 출력하는 함수이다. 이 함수에서 또한 파일의 종류별로 출력 색깔을 지정하기 위해 check\_color()함수를 호출하여 색깔을 지정한다.

* **void check\_color(mode\_t mode);**

check\_color함수는 앞선 printtNormal, printLmode함수에서 각 파일 종류 별로 색깔을 지정하기 위해 사용되는 함수이다. 인자로는 파일의 mode값을 받아와서 실행권한 유무, 디렉터리 파일, symbolic link 파일, GID bit, UID bit등의 조건에 따라 초록색, 파란색, 하늘색, 노란색 배경, 빨간색 배경 등의 출력 색깔을 지정한다. 더욱 강력한 조건일 경우 함수 내에서 조건문이 하단에 위치하여 강력한 조건의 색깔로 설정될 수 있게 하였다.

* **void sortStat(struct stat \*info, char \*\*name, int cnt, int opt\_t);**

sortStat()함수는 일반 모드와 -t 옵션이 들어 왔을 때 두가지 경우에 따라 출력할 stat 구조체를 정렬하는데 인자로 들어오는 int opt\_t가 0일 때 파일들의 이름순으로 stat 구조체를 정렬하고 -t 옵션이 들어와 opt\_t의 값이 1일때 stat 구조체로부터 파일의 최종 수정 시간을 읽어와 수정 시간 순으로 파일을 정렬하는 기능을 수행한다.

* **void error(char \*name);**

error()함수는 여러가지 시스템 콜 라이브러리 혹은 프로그램 실행 중 예상치 못한 결과값을 받아 오류 메세지를 출력하기 위해 호출되는 함수이다. 인자로 파일의 이름을 받아 오류 메세지를 출력한다.

* 1. **mychmod.c**

mychmod.c는 리눅스 시스템 상의 chmod 명령어와 동일하게 파일의 권한을 설정하는 프로그램이다. 기본적으로 mychmod 4자리 이하의 숫자 파일이름을 입력했을 때 사용자가 입력한 권한으로 파일의 권한을 변경해주고 권한 주체 u,g,o,a 와 연산자 ‘+’, ’-‘, ‘=’ 그리고 설정할 권한인 ‘r’, ‘w’, ‘x’, ‘s’, ‘t’를 조합하여 파일의 권한을 변경해주는 string 입력 모드로 구현이 되어 있다. 이 2가지 기능은 리눅스 시스템 상의 chmod와 동일하게 작동한다.

먼저 mychmod.c에서는 chmod 명령어의 옵션들의 기능은 구현이 되어 있지 않기 때문에 unistd.h의 getopt함수를 이용하여 혹시라도 옵션 입력이 들어오면 해당 옵션들을 제외하여 프로그램이 작동할 수 있게 한다.

String 입력 모드인지 숫자 입력 모드인지 확인을 위하여 사용자 입력에서 ‘+’, ’-‘, ‘=’가 있는지 확인하여 해당 문자가 존재하면 String 입력 모드로 분기하고 아닐 경우 숫자 입력 모드로 분기를 하게 된다.

String 모드일 경우 우선 strchr을 이용하여 리턴 받은 해당 문자열 포인터의 0번째 인덱스의 값을 char op 변수에 저장하여 추후 연산자를 저장한다. 다음으로 먼저 사용자가 입력한 문자열이 누구에게 어떤 권한을 주는것인지 확인하는 코드를 수행하여 각 u, g, o 별로 char flag[4]에 알맞는 위치에 8진수 숫자를 저장한다. 다음으로 사용자가 uid, gid, sticky bit 설정을 입력하였는지 확인하여 char flag[0] 알맞는 위치(0번째 인덱스)에 8진수 숫자를 저장한다.

위와 같은 확인 작업이 끝나면 char flag[4]를 sscanf()를 이용하여 int형 변수 to\_change에 8진수 형태로 저장을 한다. 다음으로는 위에서 저장한 char op변수를 확인하고 알맞는 연산을 파악해 |=, &= ~, = 연산을 수행하여 mode값을 수정한다. 이때 String 모드는 숫자 입력 모드와 달리 권한을 새로 설정하는 것이 아닌 권한 추가 혹은 제거 연산이 포함되어 있기 때문에 미리 사용자가 입력한 파일의 mode값을 가져와서 |=, &=~ 연산과 같이 추가 혹은 제거 연산을 수행해야 한다.

다음으로 숫자 입력 모드는 단순히 사용자가 입력한 숫자를 sscanf()를 사용하여 int mode 변수에 8진수 형태로 저장한다.

숫자 입력 모드 혹은 String 입력 모드를 통해 int mode 값의 입력이 완료되면 입력된 mode값이 올바른 값인지 확인하는 과정을 거치고 chmod() 시스템 콜 라이브러리 함수를 이용하여 해당 파일의 권한을 변경하면 프로그램이 종료가 된다.

* 1. **mytouch.c**

mytouch.c는 리눅스 시스템 상의 touch 명령어와 동일하게 파일 수정 시간을 바꾸어 주는 명령어이다. 다만 touch 명령어의 옵션들을 구현되어 있지 않다. touch 명령어는 입력한 파일의 수정 시간을 바꾸어 주는 기능도 하는 반면 해당하는 파일이 없을 경우 파일을 새로 생성하는 기능도 수행하게 되는데 이 경우 파일의 권한을 default 권한인 0666에서 실행하는 프로세스의 umask값을 빼서 생성하는 파일의 권한을 설정하게 된다. 예를 들어 현재 터미널 프로세스의 umask 값이 002라고 할때 touch 명령어를 사용하여 파일을 생성하게 되면 해당 파일의 권한은 0664로 설정되어 파일을 생성하게 된다.

가장 먼저 현재 mytouch.c에는 기본 touch 명령어의 옵션들이 구현되어 있지 않기 때문에 getopt()함수를 이용하여 혹시 들어올 수 있는 옵션 입력들을 제거한다.

다음으로 utime()함수를 사용하여 입력 받은 파일의 수정 시간을 변경하게 되는데 만약 해당 파일이 존재하지 않아 음수 값이 리턴 되었을 경우 creat()함수를 호출하여 파일을 생성한다 이때 mode\_t mode 변수를 선언하고 다음으로 mode 변수에 0666-umask(0) 연산을 수행하여 저장하여 생성할 파일의 권한을 얻어 올 수 있다. umask()함수는 사용자가 입력한 mode대로 현재 프로세스의 umask값을 수정하고 리턴 값으로 변경 전에 설정되어 있던 umask값을 리턴하기 때문에 위의 연산을 통하여 현재 프로세스가 부여할 수 있는 권한을 얻어 올 수 있다. 이렇게 부여할 권한을 설정한 후 해당 권한으로 파일을 생성하면 프로그램이 종료가 된다.

1. **소스코드와 주석**

**2-1. myls.c**

#include<stdio.h>

#include<stdlib.h>

#include<string.h> //strcmp()

#include<unistd.h>//getopt()

#include<sys/stat.h> //struct stat, lstat()

#include<pwd.h> //getpwuid()

#include<grp.h> //getgrgid()

#include<time.h> //ctime()

#include<dirent.h> //for read directory

#define MAX\_FILE\_CNT 100

#define MAX\_FILE\_PATH 1024

#define MAX\_FILE\_NAME 50

#define ANSI\_COLOR\_GREEN "\x1b[92m"

#define ANSI\_COLOR\_BLUE "\x1b[94m"

#define ANSI\_COLOR\_SKY "\x1b[96m"

#define ANSI\_COLOR\_BLACK "\x1b[30m"

#define ANSI\_COLOR\_RED\_BACK "\x1b[41m"

#define ANSI\_COLOR\_YEL\_BACK "\x1b[43m"

#define ANSI\_COLOR\_RESET "\x1b[0m"

struct option{

int opt\_i;

int opt\_l;

int opt\_t;

int opt\_a;

}option;

int total\_blk = 0;

int getFileStat(struct stat \*i, char \*\*, char \*);

void mode\_to\_str(mode\_t , char \*);

void printLmode(char \*, struct stat \*);

void printNormal(char \*, struct stat \*);

void sortStat(struct stat \*, char \*\*, int, int);

void check\_color(mode\_t);

int is\_dir(char \*);

void error(char \*);

int main(int argc, char \*argv[]){

struct stat \*info = malloc(sizeof(struct stat)\*MAX\_FILE\_CNT);

char \*statName[MAX\_FILE\_CNT];

char opt;

//list할 파일의 개수를 저장

int cnt = 0;

//입력에 i,l,t옵션이 존재하는 지 확인

//존재하면 해당 옵션변수를 1로 변경

while((opt = getopt(argc, argv, "ilta")) != -1){

switch(opt){

case 'i':

option.opt\_i = 1;

break;

case 'l':

option.opt\_l = 1;

break;

case 't':

option.opt\_t = 1;

break;

case 'a':

option.opt\_a = 1;

}

}

//지정 디렉터리나 파일이 없을 경우 현재 디렉터리를 인자로 넣어준다.

//ls명령어의 기본 현재 디렉터리 확인

if(optind == argc)

argv[argc++] = ".";

cnt = getFileStat(info, statName, argv[optind]);

//option.opt\_t의 값을 확인하여 알파벳순 정렬(default)할지, 수정 시간순으로 정렬할지 인자를 넘겨준다.

//option.opt\_t의 default값은 0이고 t옵션이 입력 되었을 경우 1

sortStat(info, statName, cnt, option.opt\_t);

//ls file이 아니고 -l옵션이 들어왔을때 total block 개수를 출력해준다.

if(cnt > 1 && option.opt\_l)

printf("total %d\n", total\_blk/2);

//getFileStat()을 통해서 가져온 모든 stat 구조체 정보를 출력한다.

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

//-i옵션이 있을 경우 앞에 inode번호를 출력한다.

if(option.opt\_i)

printf("%-7ld ",info[i].st\_ino);

//-l 옵션이 있을 경우 printLmode함수를 호출하여 stat구조체의 모든 맴버의 상세 정보를 출력한다.

if(option.opt\_l){

printLmode(statName[i], info+i);

}else{

//-l 옵션이 아닌 경우 단순 파일 이름 들만 출력한다.

printNormal(statName[i], info+i);

//10번째 파일 항목마다 개행을 수행, 마지막 항목일 경우에도 개행

if((i%9 == 0 && i != 0) || i == (cnt-1))

putchar('\n');

}

}

free(info);

// printf("max : %d\n", max\_file\_len);

return 0;

}

int getFileStat(struct stat \*info, char \*\*statName, char \*input){

int cnt=0;

//ls directory

if(is\_dir(input)){

DIR \*dp;

struct dirent \*dFile;

char fPath[MAX\_FILE\_PATH];

//해당 dir을 open하고 dirent를 하나씩 lstat으로 확인하여 stat구조체를 가져온다.

if((dp = opendir(input)) != NULL){

//디렉터리 엔트리 전체 확인

while(( dFile = readdir(dp)) != NULL){

//절대 경로를 생성해준다.

sprintf(fPath, "%s/%s", input, dFile->d\_name);

//숨김파일의 경우 list하지 않는다.

if(dFile->d\_name[0] == '.' && option.opt\_a != 1)

continue;

//해당 파일의 stat구조체 확득

if(lstat( fPath, info+cnt) < 0)

error(dFile->d\_name);

//statName에 현재 이름을 저장

statName[cnt] = dFile->d\_name;

total\_blk += (info+cnt)->st\_blocks;

cnt++;

}

}

//ls file

}else{

cnt++;

statName[0] = input;

//인자로 들어온 파일 이름의 stat구조체를 획득

if(lstat( statName[0], info) < 0)

error(input);

}

return cnt;

}

void error(char \*name){

//파일 엑세스에 오류가 발생할 경우 출력문을 출력하고 프로그램 종료

printf("myls: can't access file'%s'\n", name);

exit(1);

}

int is\_dir(char\* fname){

//fname이 디렉터리 파일인지 일반 파일인지 확인

struct stat buff;

if(lstat(fname, &buff) != 0)

error(fname);

return S\_ISDIR(buff.st\_mode);

}

void mode\_to\_str(mode\_t mode, char \*buf){

const char chars[] = "-rwxrwxrwx";

//해당 파일이 어떤 종류인지 확인

switch(mode & S\_IFMT){

case S\_IFREG:

buf[0] = '-';

break;

case S\_IFDIR:

buf[0] = 'd';

break;

case S\_IFCHR:

buf[0] = 'c';

break;

case S\_IFBLK:

buf[0] = 'b';

break;

case S\_IFIFO:

buf[0] = 'p';

case S\_IFLNK:

buf[0] = 'l';

break;

case S\_IFSOCK:

buf[0] = 's';

default:

buf[0] = '?';

}

//권한을 확인하여 저장

for(size\_t i =1; i<10; i++)

buf[i] = (mode & (1 << (9-i))) ? chars[i] : '-';

const char ch[2][3] = {{'s', 's', 't'},

{'S', 'S', 'T'}};

for(int i=0; i<3; i++){

if(mode & (S\_ISUID>>i)){

if(buf[(i+1)\*3] == 'x')

buf[(i+1)\*3] = ch[0][i];

else

buf[(i+1)\*3] = ch[1][i];

}

}

buf[10] = '\0';

}

void printLmode(char \*name, struct stat \*info){

//-l 옵션 항목 출력

char str[11];

mode\_to\_str(info->st\_mode, str);

printf("%s ", str);

printf("%2ld", info->st\_nlink);

printf(" %-9s ", getpwuid(info->st\_uid)->pw\_name);

printf("%-9s ", getgrgid(info->st\_gid)->gr\_name);

printf("%5ld ", info->st\_size);

printf("%.12s ", ctime(&info->st\_mtime)+4);

check\_color(info->st\_mode);

printf( "%s" ANSI\_COLOR\_RESET, name);

if(str[0] == 'l'){

char \*path = malloc(sizeof(char)\*MAX\_FILE\_NAME);

memset(path, '\0', MAX\_FILE\_NAME);

printf(" -> ");

if(readlink(name, path, MAX\_FILE\_NAME) < 0)

path = "can't find path";

else{

if(lstat( path, info) < 0)

error(path);

check\_color(info->st\_mode);

}

printf("%s"ANSI\_COLOR\_RESET, path);

}

putchar('\n');

}

void printNormal(char \*name, struct stat \*info){

char str[11];

mode\_to\_str(info->st\_mode, str);

check\_color(info->st\_mode);

printf("%.30s "ANSI\_COLOR\_RESET, name);

//printf("%.30s "ANSI\_COLOR\_RESET, name);

}

void check\_color(mode\_t mode){

for(int i=0; i<3; i++){

//user, group, other 중에 파일 실행 권한이 있을 경우

//초록색

if(mode &(S\_IXUSR >> i\*3))

printf(ANSI\_COLOR\_GREEN);

}

//디렉터리일 경우 파란색 -> 디렉터리의 경우 실행권한이 있어도 파란색으로 출력이 된다.

if(S\_ISDIR(mode))

printf(ANSI\_COLOR\_BLUE);

//symbolic link 파일일 경우 하늘색으로 출력

else if(S\_ISLNK(mode))

printf(ANSI\_COLOR\_SKY);

//GID bit이 설정되어 있을 경우 노란색 배경에 검은색 글씨로 출력

if(mode &(S\_ISGID)){

printf(ANSI\_COLOR\_YEL\_BACK);

printf(ANSI\_COLOR\_BLACK);

}

//uid bit이 설정 되어 있을 경우 빨간색 배경으로 출력

//uid, gid bit이 모두 설정되어 있을 경우 uid bit이 더 상위이다.

if(mode & (S\_ISUID)){

printf(ANSI\_COLOR\_RESET);

printf(ANSI\_COLOR\_RED\_BACK);

}

}

void sortStat(struct stat \*info, char \*\*name, int cnt, int opt\_t){

struct stat tmpStat;

char \*tmpName;

//인자로 들어온 t옵션을 기준으로 어떠한 순서로 정렬할지 선택한다.

for(int i=cnt-1; i>0; i--){

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

//flag == 1일때 st\_mtime비교, flag == 0일때 이름 비교

if((opt\_t == 1 && !(info[j].st\_mtime < info[j+1].st\_mtime)) || (opt\_t== 0 && !(strcmp(name[j],name[j+1]) > 0)))

continue;

tmpStat = info[j];

tmpName = name[j];

info[j]= info[j+1];

name[j] = name[j+1];

info[j+1] = tmpStat;

name[j+1] = tmpName;

}

}

}

**2-2. mychmod.c**

#include<stdio.h>

#include<stdlib.h> //exit()

#include<sys/stat.h> //chmod()

#include<string.h> //strchr()

#include<errno.h> //errno

#include<unistd.h> //getopt)()

int main(int argc , char \*argv[]){

int mode = 0;

char \*ptr;

//옵션이 들어왔을 경우 무시하고 처리

while(getopt(argc, argv, "cfvR") != -1);

if(((ptr =strchr(argv[optind], '+')) != NULL) ||

((ptr =strchr(argv[optind], '-')) != NULL) ||

((ptr =strchr(argv[optind], '=')) != NULL)){

//ptr에 담긴 연산자를 op에 저장

char op = ptr[0];

//string으로 들어온 권한 설정을 확인하기 위해

//미리 확인할 값을 저장

const char perm[2][3] = {{'u', 'g', 'o'},

{'r', 'w', 'x'}};

//+,-, = 기준으로 토큰을 분리

char \*ugo = strtok(argv[optind], &op);

char \*rwx = strtok(NULL, &op);

struct stat info;

char flag[4] = { '0', };

int to\_change;

//입력된 파일의 mode값을 받는다.

if(lstat( argv[optind+1] , &info) < 0){

fprintf(stderr, "%s : chmod error : %s\n", argv[optind+1], strerror(errno));

exit(1);

}

//ugo 전체에 해당하는 a가 들어왔을 때 1번째 토큰에 ugo를 넣어준다.

if(strcmp(ugo, "a") == 0)

ugo = "ugo";

mode = info.st\_mode;

//파일 종류를 확인하는 bit를 없앤다.

mode%=010000;

int check;

//들어온 인자를 확인하여 flag에 저장

for(int i=0, check =0; i<3; i++){

check = 0;

// div = 1;

if(strchr(ugo, perm[0][i]) != NULL){

for(int j=0; j<3; j++){

if(strchr(rwx, perm[1][j]) != NULL){

check += 4/(j+1);

}

//div \*= 2;

}

}

flag[i+1] = check+'0';

}

for(int i=0, check =0; i<3; i++){

if(strchr(ugo, perm[0][i]) != NULL){

char ch = i<2 ? 's' : 't';

if((strchr(rwx, ch) != NULL)){

check += (4/(i+1));

flag[0] = check+'0';

}

}

}

//flag에 저장되어 있는 값을 8진수 형태로 to\_change 변수에 담아준다.

sscanf(flag, "%o", &to\_change);

//op에 들어와 있는 연산자 별로 올바른 연산을 수행한다.

switch(op){

case '+':

mode |= to\_change;

break;

case '-':

mode &= ~to\_change;

break;

case '=':

mode = to\_change;

break;

default:

fprintf(stderr, "%s : chmod error : invalid operator\n", argv[optind+1]);

break;

}

}else{

//숫자 입력 모드일때 해당 값을 8진수 형태로 mode에 담아준다.

sscanf(argv[optind], "%o", &mode);

}

//mode에 저장된 값이 8진수 07777보다 크거나 0보다 작을 경우 미리 오류 처리를 해준다.

if(mode > 07777 || mode < 0){

fprintf(stderr, "%s : chmod error : invalid permission\n", argv[optind+1]);

exit(1);

}

//권한 변경 실행

if(chmod(argv[optind+1], mode) < 0){

fprintf(stderr, "%s : chmod error : %s\n", argv[optind+1], strerror(errno));

}

exit(0);

}

**2-3. mytouch.c**

#include<stdio.h>

#include<stdlib.h> //exit()

#include<fcntl.h> //creat()

#include<utime.h> //utime()

#include<unistd.h> //getopt()

#include<sys/stat.h> //umask()

#include<sys/types.h> //mode\_t

int main(int argc, char \*argv[]){

while(getopt(argc, argv, "acdfhmrt") != -1);

if(utime(argv[optind], NULL) < 0){

mode\_t mode = 0666 - umask(0);

int fd;

if((fd = creat(argv[optind], mode)) < 0){

fprintf(stderr, "creat error for %s\n", argv[optind]);

}

}

exit(0);

}

1. **실행결과**

실습 과제 3의 모든 실행파일은 /bin/ 디렉터리에 심볼릭링크하여 일반 시스템 명령어와 동일하게 동작할 수 있도록 하여 테스트하였다.

**3-1. myls.c**

**텍스트, 점수판이(가) 표시된 사진

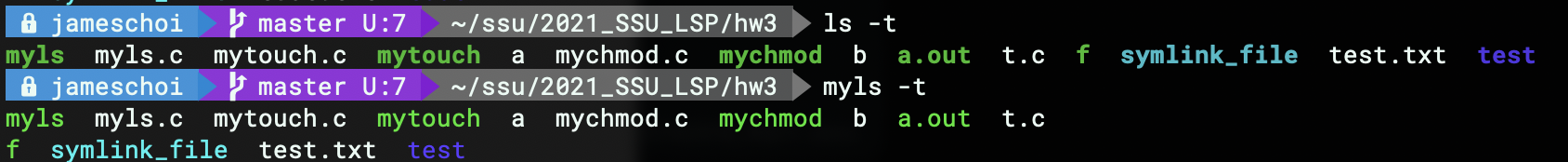
자동 생성된 설명**

**[일반 myls 호출 비교 화면]**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**[myls -I 호출 비교 화면]**

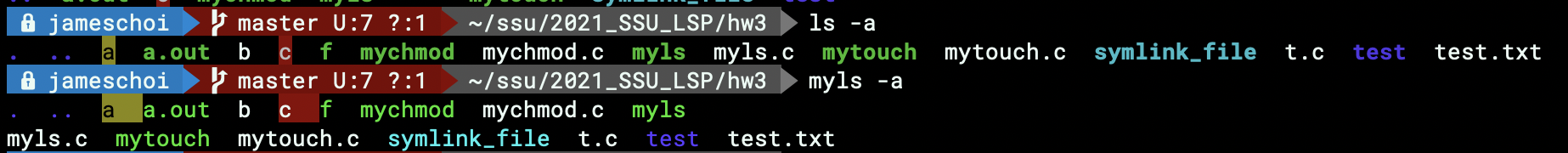
****

**[myls -t 호출 비교 화면]**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**[myls -l 호출 비교화면]**

****

**[myls -a 호출 비교화면]**

****

**[myls -li 옵션 중첩 호출 비교화면]**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**[myls -lita 구현 옵션 전체 호출 비교화면]**

**3-2. mychmod.c**

**mychmod 명령어는 호출 후 myls -l “파일이름” 을 이용하여 결과를 확인했다.**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**[mychmod 0 a 호출 화면]**

**텍스트, 점수판이(가) 표시된 사진

자동 생성된 설명**

**[mychmod 0664 a 호출 화면]**

**텍스트, 점수판, 닫기이(가) 표시된 사진

자동 생성된 설명**

**[mychmod 7664 a 호출 화면]**

**텍스트, 점수판, 닫기이(가) 표시된 사진

자동 생성된 설명**

**[mychmod ug+rwx a 호출 화면]**

**텍스트, 점수판이(가) 표시된 사진

자동 생성된 설명**

**[mychmod g-wx a 호출 화면]**

**텍스트, 점수판이(가) 표시된 사진

자동 생성된 설명**

**[mychmod go+rw 호출화면]**

**텍스트, 점수판이(가) 표시된 사진

자동 생성된 설명**

**[mychmod ugo=r a 호출 화면]**

**텍스트, 점수판이(가) 표시된 사진

자동 생성된 설명**

**[mychmod ug+s a 호출 화면]**

**텍스트, 점수판이(가) 표시된 사진

자동 생성된 설명**

**[mychmod o+t a 호출 화면]**

**3-1. mytouch.c**

**텍스트, 점수판, 닫기이(가) 표시된 사진

자동 생성된 설명**

**[mytouch t.c 호출 화면]**

파일 수정 시간이 바뀌었음을 확인할 수 있다.

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**[mytouch a를 호출하여 파일 생성]**

현재 프로세스의 umask는 0002로 설정되어 있어 파일 생성 시 권한이 0664로 생성됨을 확인할 수 있다.

1. **구현을 통해 배운 점**

먼저 myls.c 구현을 통하여 파일의 정보가 리눅스 시스템 상의 어떠한 형태로 stat 구조체의 담기는지 알 수 있었다. 또 stat 구조체 맴버 변수인 st\_mode에 파일의 종류, uid, gid, sticky bit, 권한이 어떠한 형태로 담겨 있는지 확인하고 그것을 일반적인 사용자가 볼 수 있는 형태로 가공하는 방법을 알 수 있었다. 또한 dirent 구조체를 선언하여 디렉터리를 open하고 open한 디렉터리의 파일 엔트리를 읽어와서 디렉터리를 구성하고 있는 파일들의 정보를 읽으며 디렉터리 정보가 어떠한 형태로 dirent 구조체에 담기는지 알 수 있었다.

ls 명령어와 유사한 myls를 구현함으로서 ls 명령어의 동작 구조와 파일 정보 관리 방법을 간접적으로 이해할 수 있었다.

다음으로 mychmod 명령어 구현을 통하여 앞선 myls.c 구현을 통하여 st\_mode 값을 확인하는 방법을 알 수 있었는데 mychmod 구현을 통하여 사용자 입력을 통하여 올바른 값으로 변환하는 방법을 알 수 있었다. 또한 uid, gid, sticky bit에 대한 이해도를 높일 수 있었다.

마지막으로 mytouch 명령어 구현을 통해서 touch 명령어가 리눅스 시스템 상에서 어떠한 방식으로 파일의 수정 시간을 변경하고 파일 생성 시 권한 설정이 프로세스의 umask값에 의해 결정된다는 사실을 알 수 있었다.