컴퓨터학부

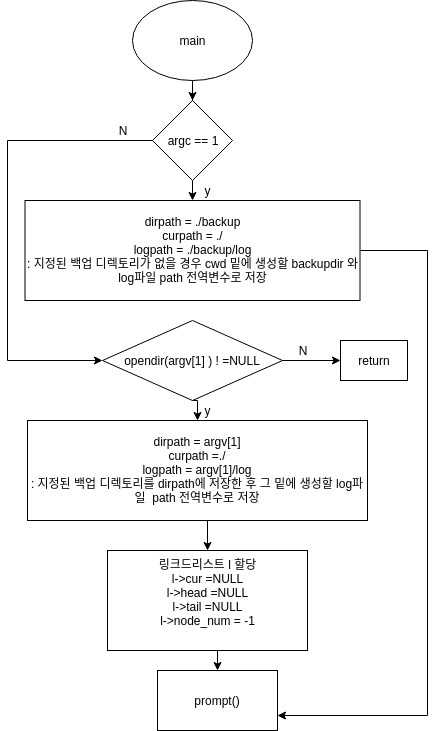
20162467

민지혜

1. 개요

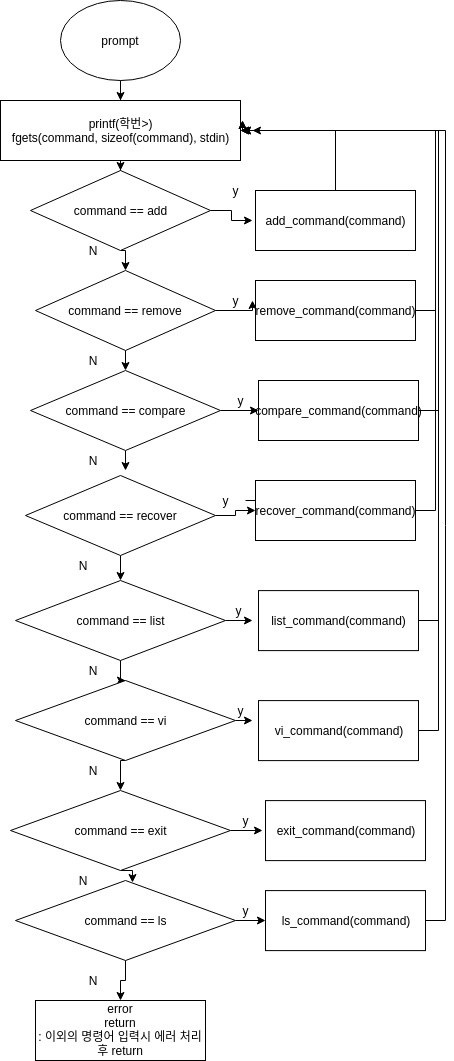
리눅스 시스템 상에서 사용자가 백업을 원하는 파일이나 디렉토리를 옵션에 따라 추가, 삭제하고 백업된 파일을 다시 복구하는 것을 관리하는 프로그램이다. 프로그램에 사용되는 명령어는 add, remove, compare, recover, list, exit, ls, vim 가 있으며 add 명령어의 옵션은 –m, -n, -t, -d가 있으며 각 명령어는 NUMBER ,TIME, DIR 등 추가적으로 입력이 되어야 하며, 각 옵션을 수행하기 필요한 요소가 입력되지 않았을 경우 에러를 출력하는 식으로 예외 처리를 한다. Remove 명령어는 -a 의 옵션을 둘 수 있다. recover의 경우는 -n의 옵션을 둘 수 있고, -n은 NEWFILE의 입력을 필요로 한다. 이 프로그램은 기본적으로 하나의 쓰레드가 한 개 파일의 백업을 담당하고 있다. 프로그램을 실행할 때, 사용자가 백업 디렉토리를 설정 가능하고, 설정하지 않았을 경우, 현재 있는 디렉토리에 backup이라는 이름의 백업 디렉토리를 생성해 백업파일, log파일을 저장한다. log파일은 add, remove, recover 명령어가 수행 됬을 경우, 수행시간과 수행내역을 로그파일에 기록한다. 로그파일은 실행 순서대로 순차적으로 기록이 되도록 한다. 이 이 프로젝트를 통해 새로운 명령어를 시스템 함수를 사용하여 구현함으로써 쉘의 원리를 이해하고, 유닉스/리눅스 시스템에서 제공하는 여러 시스템 자료구조를 이용하여 프로그램을 작성함으로써 시스템 프로그래밍 설계 및 응용 능력을 향상하는 것을 목표로 한다.

1. 상세설계

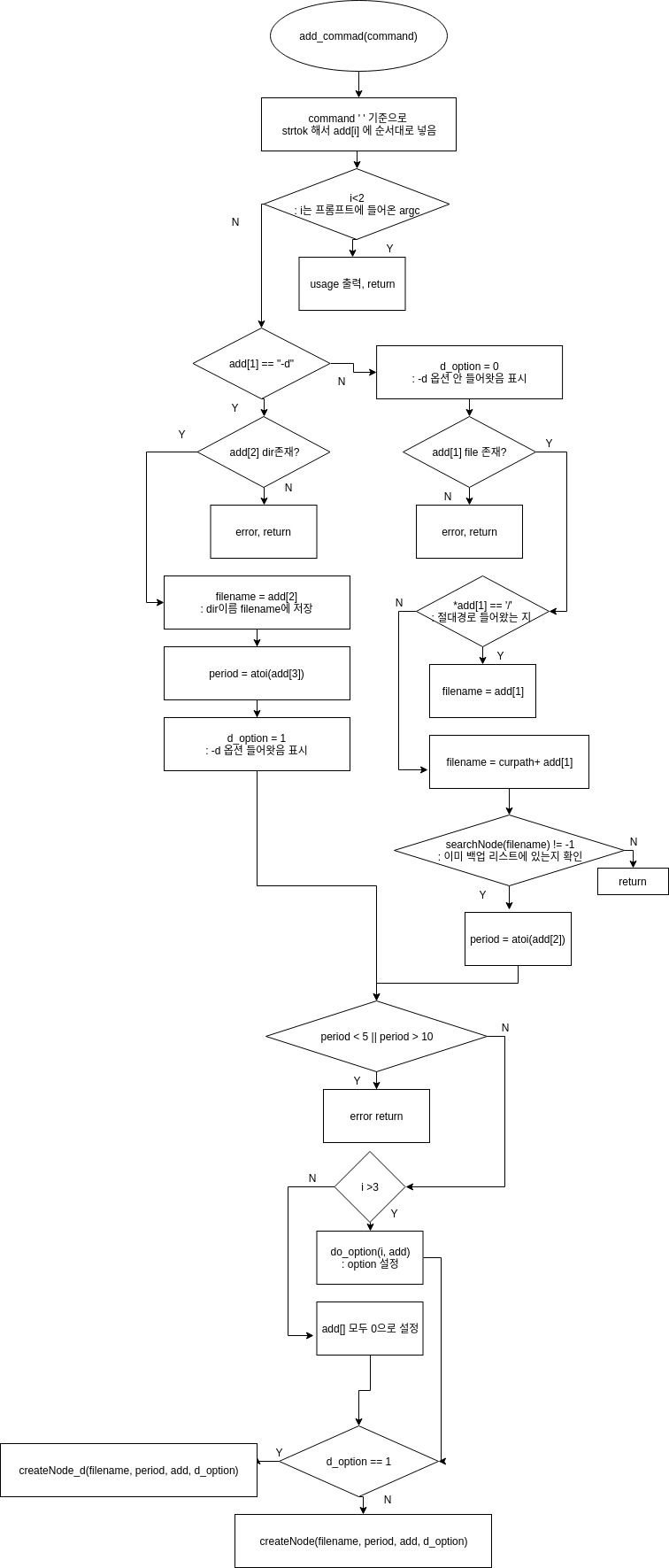


Main()은 argc 의 개수로 백업dir가 사용자에 의해 설정되었는지 판단한다.

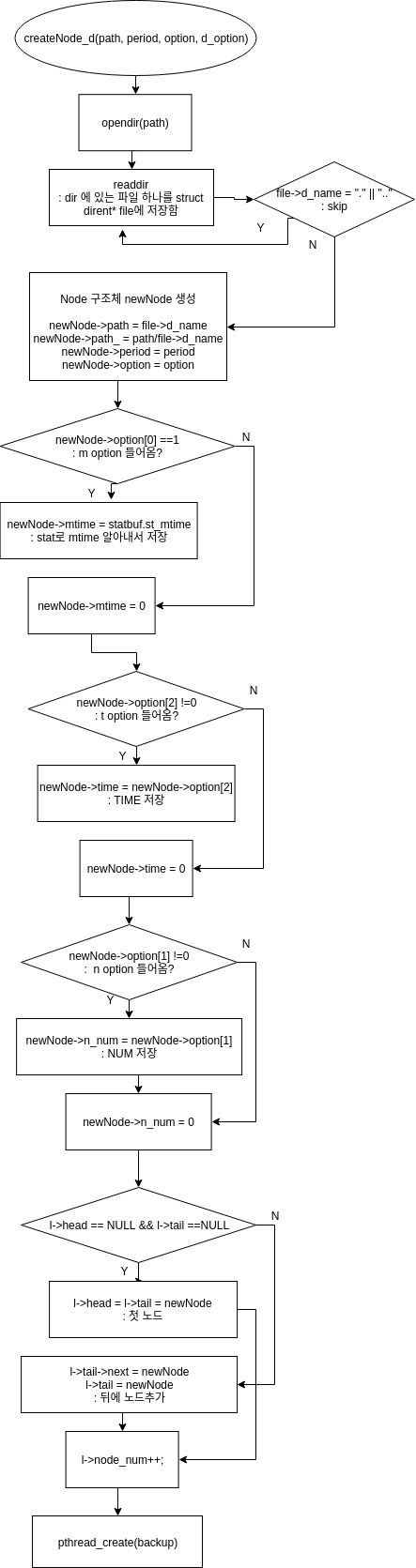
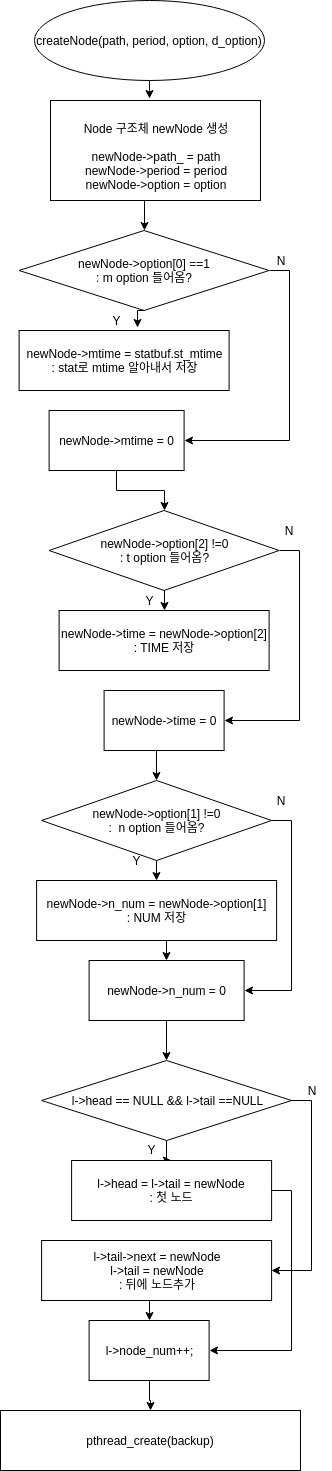
백업 dir가 들어오지 않았을 경우 현재 작업 dir 내에 backup이라는 이름으로 백업 디렉토리를 생성한 후 작업을 실행한다. 그 이후 링크드리스트를 초기설정을 한 후 prompt()를 실행한다.



Prompt()는 학번> 을 반복적으로 출력해 프롬프트 처럼 보이게 한다. 사용자가 입력한 명령어를 분석하여 그 명령어에 맞는 함수를 호출한다. 지정된 이외의 명령어가 입력됬을 경우 에러처리를 하고 다시 새로운 프롬프트를 출력해 새로운 값의 입력을 기다린다.

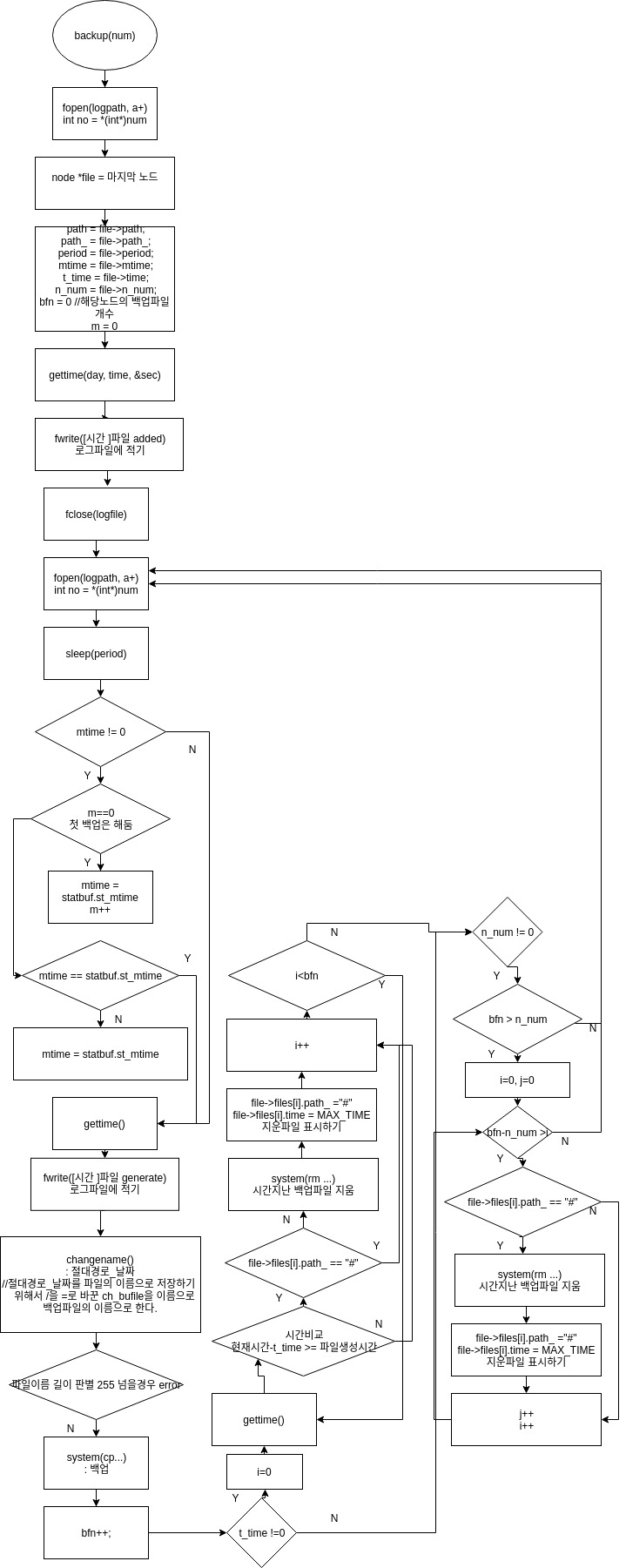


Add\_command()는 사용자가 add 명령어를 입력했을 경우 실행된다. 사용자가 입력한 값 command를 분석하여 각 옵션의 유무를 판단 옵션을 다음 함수에서 이용할 수 있도록 do\_option()을 이용해 option들의 값을 설정한다. 사용자가 입력한 파일의 이름이 절대경로인지 상대경로인지 판단한 후 상대경로일 경우 절대경로로 변환하여 filename 에 저장한다. d옵션이 있을 경우 createNode\_d()로 다음 작업을 진해하고 없을 경우 createNode()로 진행한다.

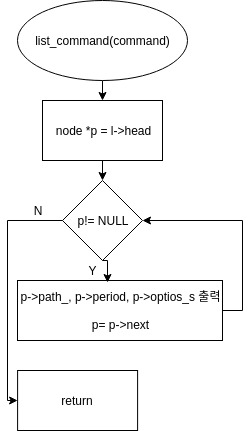
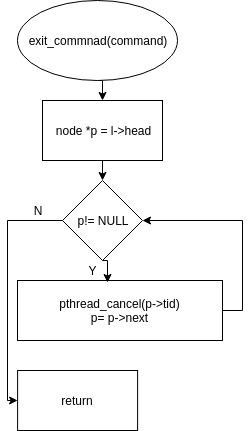


createNode\_d()는 사용자가 백업하고자 하는 dir의 유무를 판단한 후 dir내의 파일의 정보를 하나하나 읽어 Node 구조체에 저장한다. 이렇게 생성된 Node 구조체들은 각 파일에 대한 정보, 백업을 하기 위한 정보를 담고 있다. 각 파일들이 이전에 리스트에 추가되어있는 파일과 동일한지 searchNode()를 통해 검사를 한 후 있을 경우 그 파일은 무시한 후 다음 작업을 진행한다. 그 다음 링크드리스트에 추가하기 위한 작업을 수행하고pthread\_create()를 하여 백업을 하기위한 backup()으로 이동한다.

createNode()는 사용자가 백업하고자 하는 파일의 유무를 판단한 후 파일의 정보를 Node 구조체에 저장한다. 이렇게 생성된 Node 구조체는 파일에 대한 정보, 백업을 하기 위한 정보를 담고 있다. 각 파일이 이전에 리스트에 추가되어있는 파일과 동일한지 searchNode()를 통해 검사를 한 후 있을 경우 그 파일은 무시한 후 다음 작업을 진행한다. 그 다음 링크드리스트에 추가하기 위한 작업을 수행하고pthread\_create()를 하여 백업을 하기위한 backup()으로 이동한다.

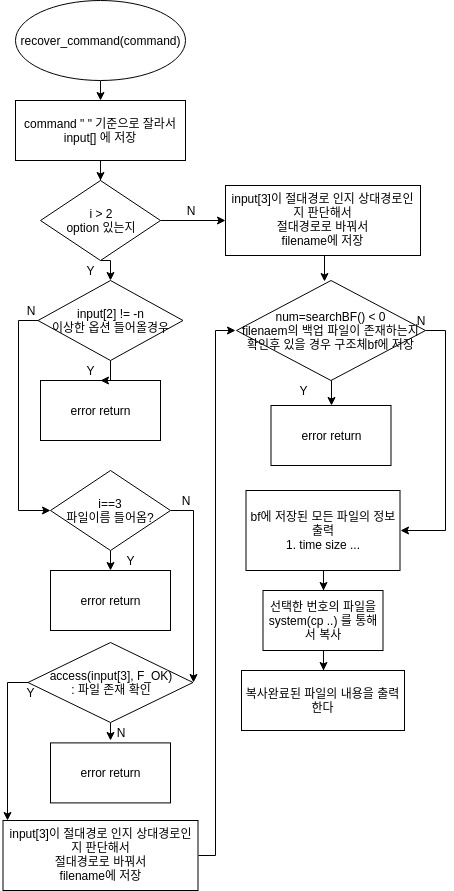


Backup()은 각 옵션의 유무를 판별 한 후, 각 옵션에 알맞게 백업 작업을 수행하기 위한 함수이다. 수행한 내역을 log 파일에 기록한다.

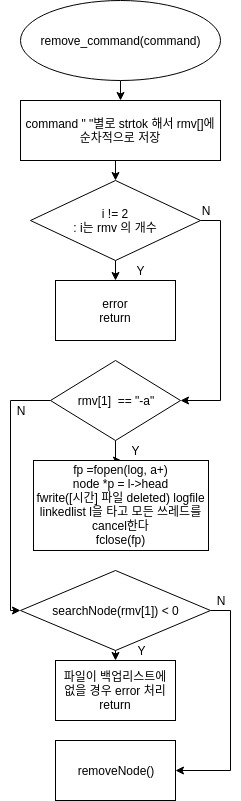


Exit\_command() 는 프롬프트를 종료하기 위한 함수이다. 링크드리스트를 타고 가면서 실행되고 있는 쓰레드를 pthread\_cancel()로 중지시킨다.

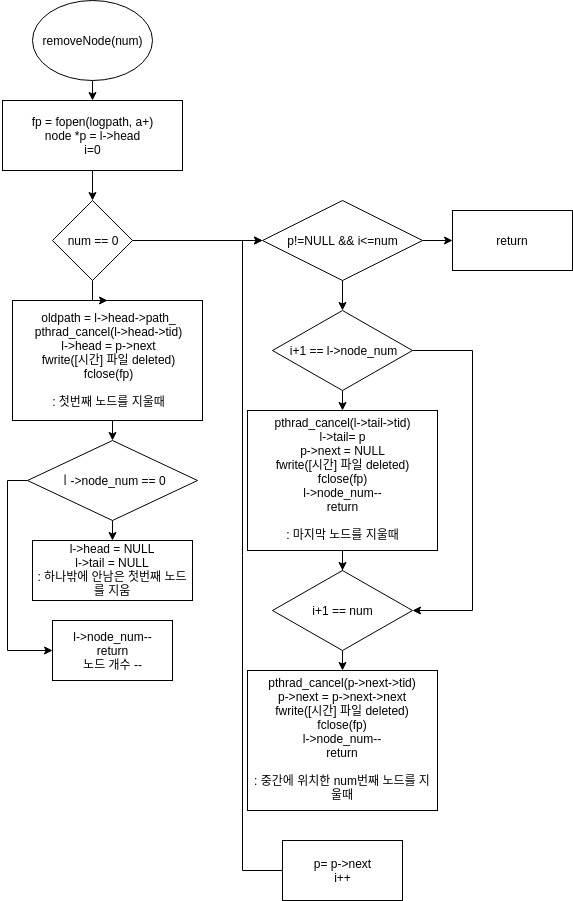
List\_command()는 현재 실행중인 쓰레드들의 백업 하고 있는 원본파일의 절대경로, period, option 의 정보를 출력하기 위한 함수이다.



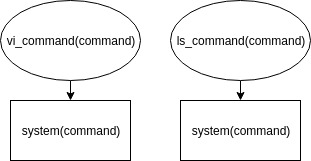
Recover()는 백업파일을 사용하여 현재의 파일을 백업된 파일로 변경하는 함수이다. 백업파일이 존재하는 경우 백업디렉토리에서 복구하고자 하는 파일의 백업파일들의 목록을 읽어와서 그 파일들의 백업시간 파일크기를 출력한 후, 사용자가 어떤 백업파일로 복구하고자 하는 지 선택할 수 있도록 한다. 복구작업은 system()를 사용한다. 사용자는 –n 옵션을 통해 기존의 파일에 복구된 내용을 넣는 대신 복구된 내용이 있는 새로운 파일을 생성할 수 있다. 복구가 성공적이었을 경우 변경된 파일의 내용을 출력한다.



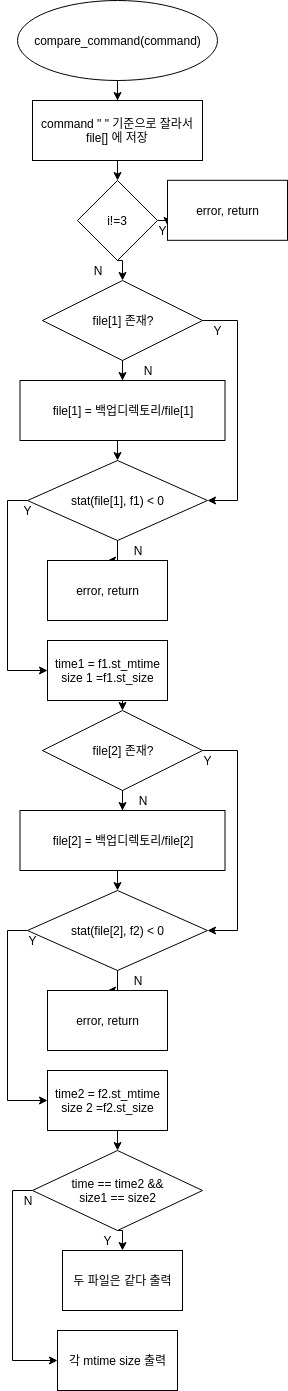
백업 리스트에 존재하는 파일의 백업을 중단하기 위해 백업리스트에서 삭제하는 함수이다. 사용자는 –a 옵션을 통해 모든 백업리스트를 초기화 할 수 있다. 사용자가 중단하고자 하는 파일이 백업리스트에 있는 지 searchNode()로 판별한 후 있을 경우 그 파일의 링크드리스트의 위치를 removerNode()에 전달해 다음 작업을 수행한다.



RemoveNode()는 지우고자 하는 노드가 링크드리스트의 머리 중간 꼬리 어느 부분에 있는지 확인한 후 각 경우에 맞는 작업을 하여 그 노드를 링크드리스트에서 제거한 후 pthread\_cancel()을 이용해 백업 작업을 중단한다.



Vi\_command() 와 ls\_command()는 vi와 ls의 명령어를 수행하기 위한 함수이다.



Compare\_command() 는 사용자가 입력한 filename1 과 filename2의 mtime과 파일크기를 비교 하는 함수이다. 백업한 파일, 백업할 파일 모두 가능하다. 우선 filename1, filename2이 각 현재 dir에 존재하는 지 판별한 후 없다면 백업폴더에서 찾아본다. 찾은 파일들의 정보를 stat()를 이용해 알아내고 각 파일의 mtime과 size를 비교한 후 동일하다면 같은 파일이라고 출력하고 다를경우 각 파일의 mtime 과 size를 출력한다.

구현

typedef struct backup\_list {

char path\_[BUFFER\_SIZE];

int time;

char time\_[BUFFER\_SIZE];

int size;

}backup\_list;

백업 작업으로 생성된 백업파일을 위한 구조체 backup\_list 이다. Path\_는 파일의 절대경로를 저장하고 있고 time은 생성된 시간을 저장한다.

typedef struct node {

char path[BUFFER\_SIZE]; //이름

char path\_[BUFFER\_SIZE]; //절대경로

int period; //기간

int option[BUFFER\_SIZE]; //들어온 옵션 저장

char option\_s[BUFFER\_SIZE]; //들어온 옵션을 문자열로 저장

pthread\_t tid; //thread

time\_t mtime;//mtime

int time; //time for -t option

int n\_num; //number for -n option

char d\_path[MAX\_FILE\_DIR][BUFFER\_SIZE];

struct backup\_list files[MAX\_BACKUP\_FILE];//백업파일들의 정보를 저장한다.

struct node \*next;

}node;

구조체 node는 각 thread를 수행할 파일의 정보를 담고있다.

typedef struct list {

struct node \*cur;

struct node \*head;

struct node \*tail;

int node\_num;

}list;

링크드리스트를 위한 구조체 list 이다. Node\_num은 링크드리스트에 연결된 노드의 개수를 의미한다.

char \*dirpath; //백업디렉토리를 의미한다

char \*curpath; //현재작업디렉토리

char logpath[BUFFER\_SIZE]; //log파일의 절대경로

int period;

char filename[BUFFER\_SIZE];

list \*l; //링크드리스트

void changename(char\* file, char\* newfile); //절대경로\_시간을 백업파일의 이름으로 저장하기 위해 ‘/’을 ‘=’로 변경한다.

void gettime(char\* day, char\* time, int\* sec); //현재 시간을 알기위한

void gettime\_bf(backup\_list \*bf);// 백업파일의 이름에서 시간을 알아오는 방법

void prompt(); //프롬프트창을 보여주고 사용자의 명령을 분석하는 함수

void add\_command(void\* command); //노드를 추가하는 함수

void remove\_command(void\* command); //삭제하는 함수

void compare\_command(void\* command); //비교하는 함수

void recover\_command(void\* command); //복구하는 함수

void list\_command(void\* command); //백업리스트 출력하는 함수

void vi\_command(void\* command); //vi 실행

void ls\_command(void\* command); //ls 실행

void exit\_command(); //프롬프트를 종료

void do\_option(int num, char\* option[]); //옵션을 찾아 정리하는 함수

void createNode(char\* path, int period, char\* option[], int d\_option);//새로운 노드를 생성해서 뒤에 붙이는 함수

void createNode\_d(char\* path, int period, char\* option[], int d\_option); //d옵션이 들어왔을 경우 새로운 노드를 생성해서 뒤에 붙이는 함수

int searchNode(char\* filename); //파일이름으로 노드에 있는지 찾는 함수

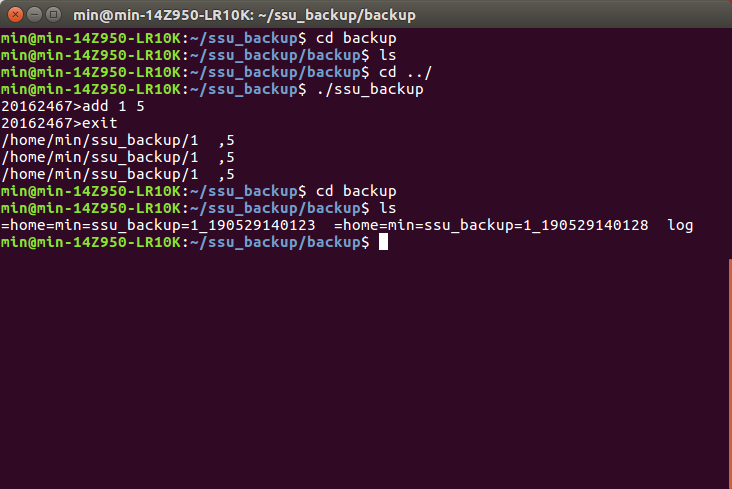
void removeNode(int num); //node를 제거하는 함수

void\* backup(void\* num); //thread를 수행하는 함수

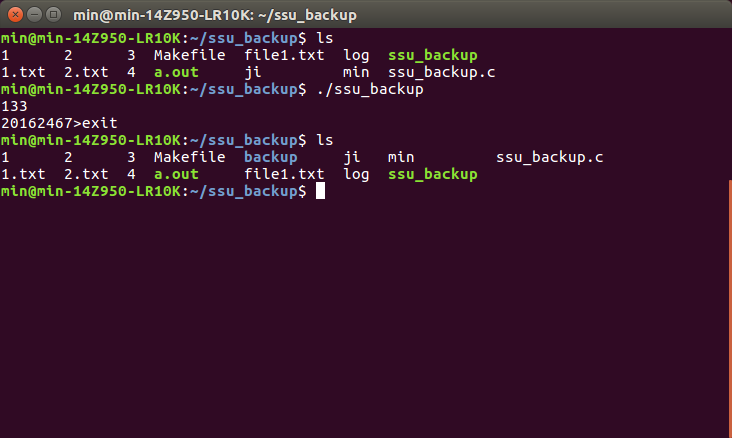
int searchBF(char \*filename, backup\_list \*bf); //원하는 파일을 복구하기 위해 필요한 백업파일이 백업디렉토리에 존재하는 지 판단

void clean\_up(void \*arg); thread정리

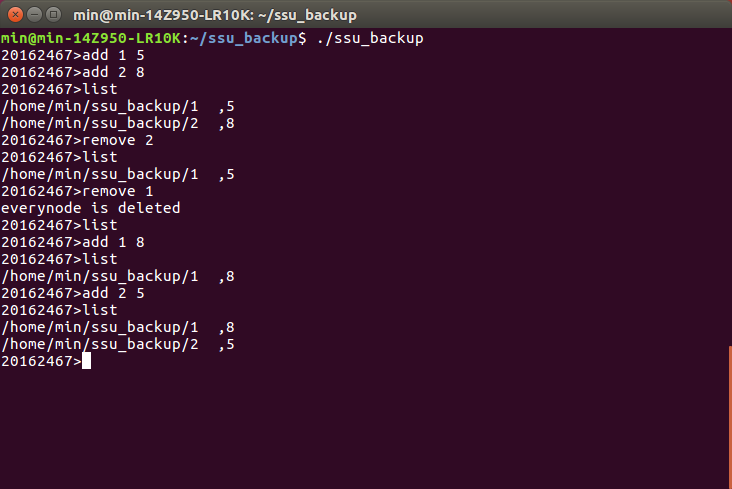
실행결과



Add 명령어

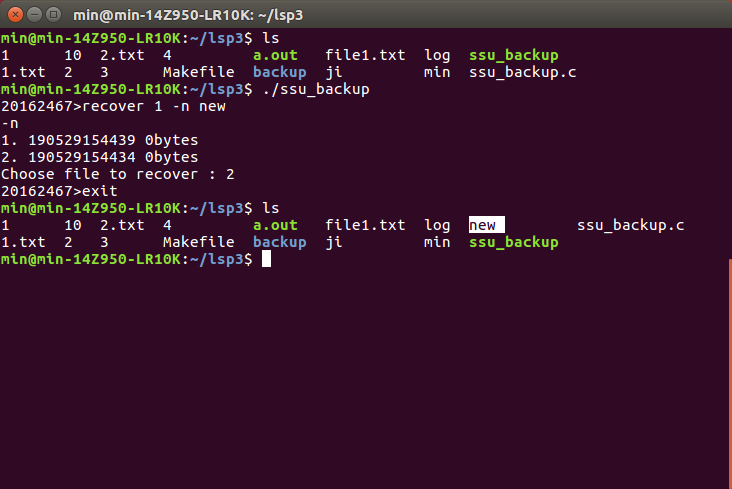


사용자가 백업 디렉토리를 설정하지 않고 실행했을 경우 backup 디렉토리가 현재 디렉토리 내에 생성되어 백업 디렉토리로 사용된다.

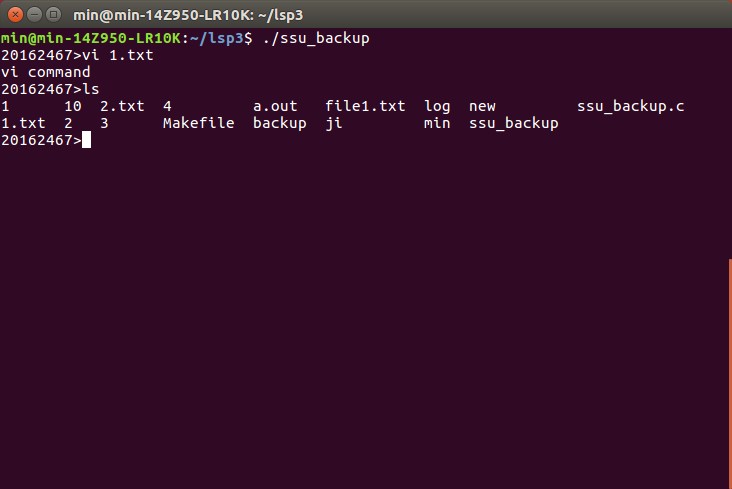


List 명령어 : 백업작업에 수행중인 파일의 절대경로 , period, d옵션이 출력된다.

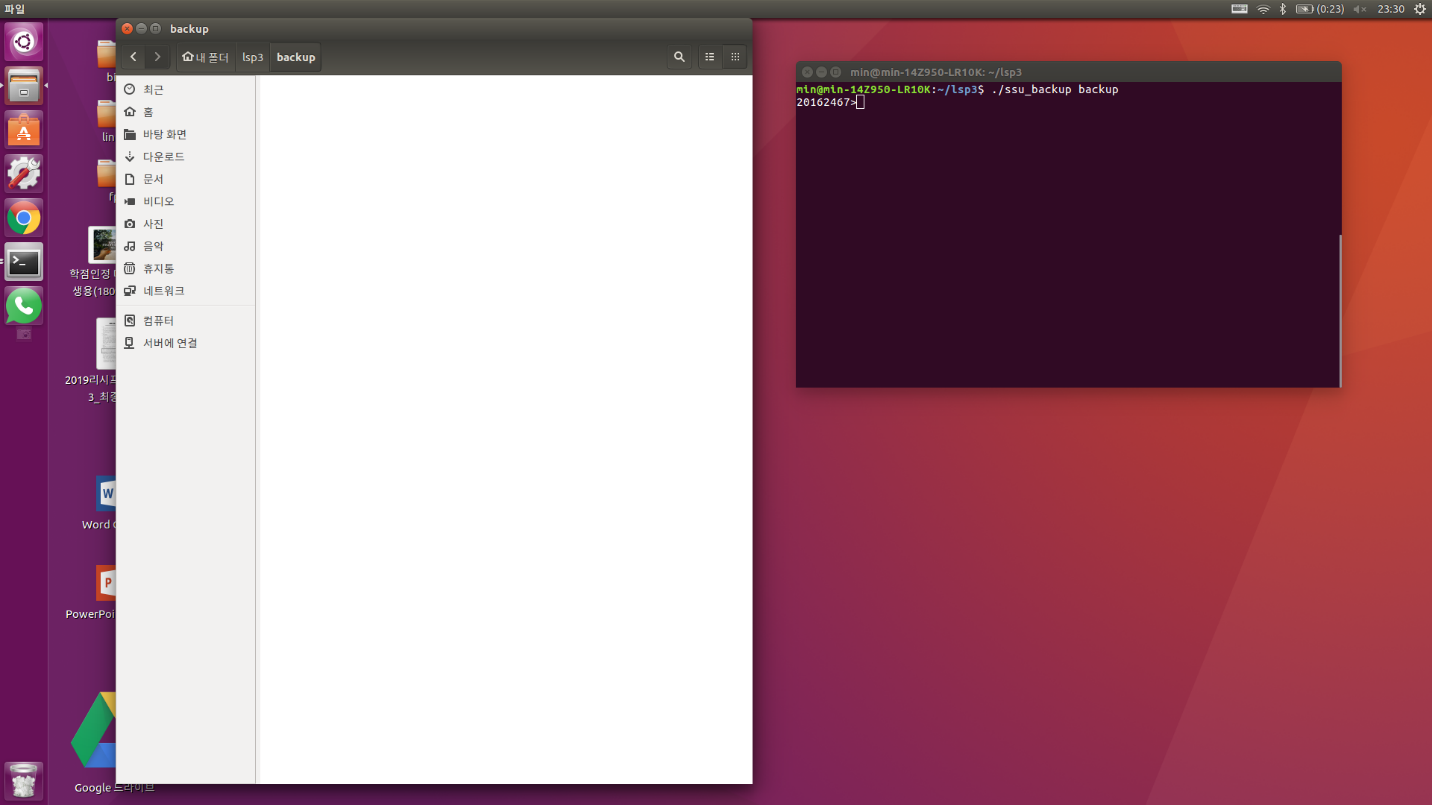
Remove 명령어 : 모든 리스트가 제거된 후에도 새로 추가 가능하다.



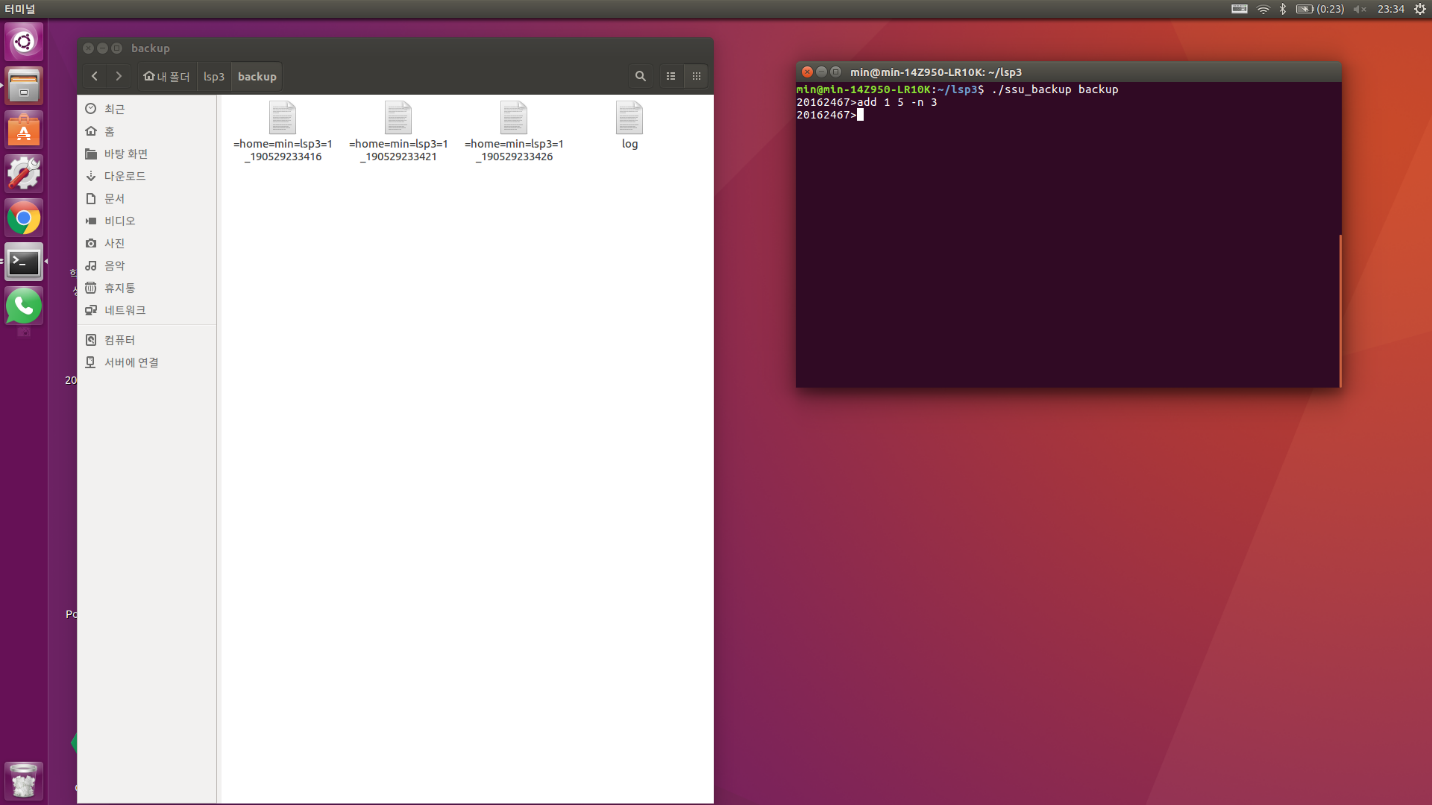
Recover 명령어 –n 옵션을 추가해 new 파일을 생성하여 파일 1 의 내용을 복원한다. 복원된 파일의 내용을 출력한다.



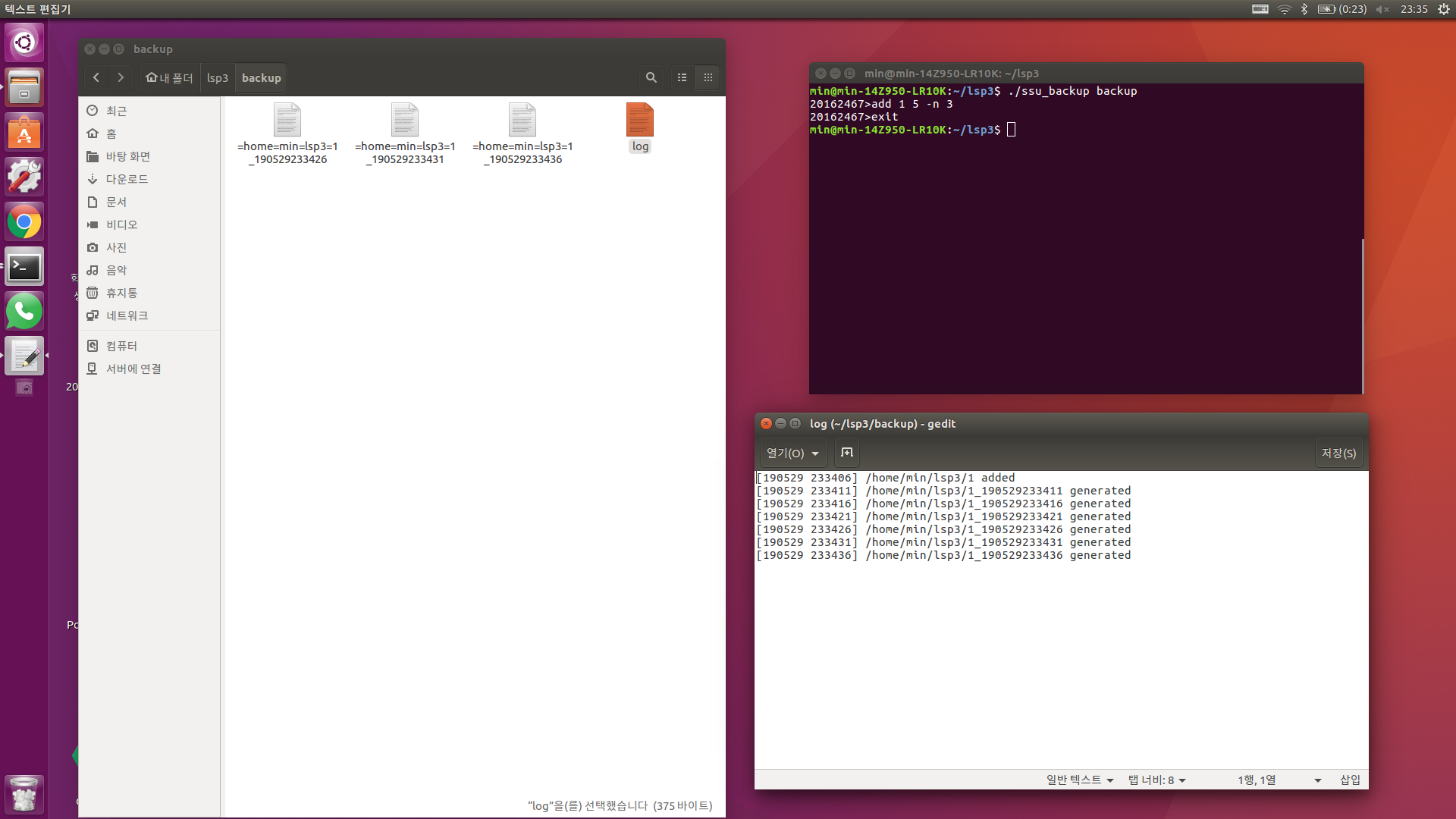
Vi 와 ls 명령어가 수행이 되는 것을 볼 수 있다.



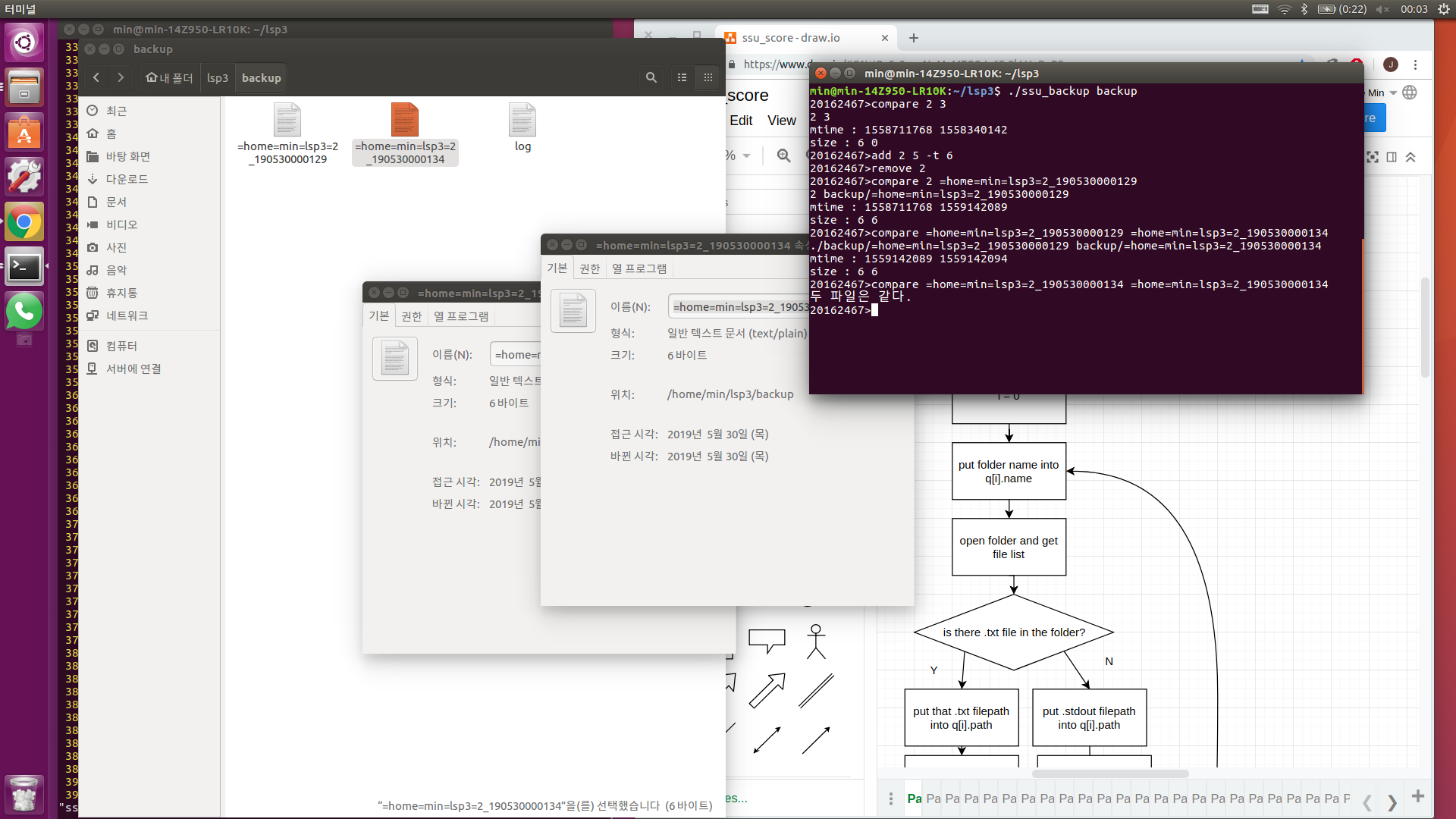
수행전



수행 후 Add –n 옵션 새로 생긴 3개의 파일들만 남긴다.



Log 파일에 기록이 남는다.



Compare 백업할 파일 끼리 비교, 백업된 파일 비교, 같은 백업파일 비교

소스코드

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <dirent.h>

#include <string.h>

#include <pthread.h>

#define BUFFER\_SIZE 5000

#define FILE\_LEN 255

#define MIN\_PERIOD 5

#define MAX\_PERIOD 10

#define MIN\_TIME 60

#define MAX\_TIME 1200

#define MAX\_BACKUP\_FILE (int)(MAX\_TIME / MIN\_PERIOD)

#define MIN\_NUM 1

#define MAX\_NUM 100

#define MAX\_FILE\_DIR 10 //-d 옵션 디렉토리 안 파일 최대 개수

typedef struct backup\_list {

char path\_[BUFFER\_SIZE];

int time;

char time\_[BUFFER\_SIZE];

int size;

}backup\_list;

typedef struct node {

char path[BUFFER\_SIZE]; //파일의 절대경로

char path\_[BUFFER\_SIZE]; //파일의 상대경로

int period; //백업 주기

int option[BUFFER\_SIZE]; //백업 옵션

char option\_s[BUFFER\_SIZE];

pthread\_t tid; //thread

time\_t mtime;//mtime

int time; //time for -t option

int n\_num; //number for -n option

char d\_path[MAX\_FILE\_DIR][BUFFER\_SIZE];

struct backup\_list files[MAX\_BACKUP\_FILE];

struct node \*next;

}node;

typedef struct list {

struct node \*cur;

struct node \*head;

struct node \*tail;

int node\_num;

}list;

char \*dirpath;

char \*curpath;

char logpath[BUFFER\_SIZE];

int period;

char filename[BUFFER\_SIZE];

list \*l;

void changename(char\* file, char\* newfile);

void gettime(char\* day, char\* time, int\* sec);

void gettime\_bf(backup\_list \*bf);

void prompt();

void add\_command(void\* command);

void remove\_command(void\* command);

void compare\_command(void\* command);

void recover\_command(void\* command);

void list\_command(void\* command);

void vi\_command(void\* command);

void ls\_command(void\* command);

void exit\_command();

char\* getfilename(char\* command);

int getperiod(char\* command);

int check\_option(char\* command);

void do\_option(int num, char\* option[]);

void logfile();

void createNode(char\* path, int period, char\* option[], int d\_option);

void createNode\_d(char\* path, int period, char\* option[], int d\_option);

void deleteLastNode();

void printNode();

int searchNode(char\* filename);

void removeNode(int num);

char\* optblock\_to\_string();

void\* backup(void\* num);

int searchBF(char \*filename, backup\_list \*bf);

void clean\_up(void \*arg) {

}

pthread\_cond\_t cond = PTHREAD\_COND\_INITIALIZER;

pthread\_mutex\_t lmu = PTHREAD\_MUTEX\_INITIALIZER;

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

DIR\* dir;

if(argc == 1) {

//인자가 없으면 current working dir 밑에 백업 디렉토리 생성

dirpath = getcwd(NULL, BUFFER\_SIZE);

strcat(dirpath, "/backup");

curpath = getcwd(NULL, BUFFER\_SIZE);

strcat(curpath, "/");

sprintf(logpath, "%s/log",dirpath);

mkdir("backup", 0777);

}

else if(argc == 2) {

if(opendir(argv[1]) == NULL) {

//인자로 입력받은 디렉토리를 찾을 수 없으면 usage출력 후 프로그램 종료

//인자로 입력받은 디렉토리가 디렉토리 파일이 아니라면 usage 출력 수 프로그램 종료

//인자로 입력받은 디렉토리의 접근 권한이 없는 경우 usage 출력 후 프로그램 종료

fprintf(stderr, "Usage\n");

exit(1);

}

else {

dirpath = argv[1];

curpath = getcwd(NULL, BUFFER\_SIZE);

strcat(curpath, "/");

sprintf(logpath, "%s/log",argv[1]);

}

}

else if(argc > 2) {

//인자가 2개 이상이면 usage 출력 후 프로그램 종료

fprintf(stderr, "Usage\n");

exit(1);

}

//링크드리스트 할당

l = (list\*)malloc(sizeof(list));

l->cur = NULL;

l->head = NULL;

l->tail = NULL;

l->node\_num = -1;

/////////프롬프트 출력////////

prompt();

}

void prompt() {

char command[BUFFER\_SIZE];

pthread\_t tid1, tid2, tid3, tid4;

int th\_id;

int result;

char\* line\_p;

/\*if(system("export PS1=\"20162467>\"") == 0) {

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

exit(1);

}\*/

//exit 명령이 입력될 때까지 위에서 지정한 실행 가능 명령어를 입력받아 실행

while(strncmp(command, "exit",4)) {

printf("20162467>");

fgets(command, BUFFER\_SIZE, stdin);

if((line\_p = strchr(command, '\n')) != NULL) \* line\_p = '\0';

if(strncmp(command, "vi", 2)==0) {

vi\_command((void\*)command);

}

else if(strncmp(command, "ls", 2)==0) {

ls\_command((void\*)command);

}

else if(strncmp(command, "add", 3)==0) {

add\_command((void\*)command);

}

else if(strstr(command, "exit")!=NULL) {

exit\_command((void\*)command);

exit(1);

}

else if(strncmp(command, "remove", 6)==0) {

remove\_command((void\*)command);

}

else if(strncmp(command, "compare", 7) == 0) {

compare\_command((void\*)command);

}

else if(strncmp(command, "recover", 7) == 0) {

recover\_command((void\*)command);

}

else if(!strncmp(command, "list ", 5)==0) {

list\_command((void\*)command);

}

else {

//이외의 명령어 입력 시 에러 처리 후 프롬프트로 넘어감.

fprintf(stderr, "109 error\n");

exit(0);

}

logfile(); //logfile에 기록

}

for(int i=0; i< BUFFER\_SIZE; i++)

command[i] = '\0';

}

void add\_command(void\* command){

//add <filename> [period] [option]

//add test1.txt\n하면 에러뜸...

char\* cmd = command;

int i = 0;

char\* ptr = strtok(cmd, " ");

char\* add[10];

char\* option[3];

int j;

int d\_option;

while(ptr != NULL) {

add[i] = (char\*)malloc(sizeof(ptr));

strcpy(add[i], ptr);

ptr = strtok(NULL, " ");

i++;

}

if(i< 2) {

fprintf(stderr,"usage: add <filename> [period] [option]\n");

return;

} //add -d dirname period -n 5 -t 6

if(strstr(add[1], "-d") != NULL) {

printf("-d option\n");

if(opendir(add[2]) == NULL) {

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

return;

}

strcpy(filename, add[2]); //dir 이름 filename 저장

period = atoi(add[3]);

d\_option = 1; //-d옵션 들어왓음

}

else {

d\_option = 0;

if(access(add[1], 00) < 0) {

fprintf(stderr,"there is no file name %s\n", add[1]);

return;

}

if(\*add[1] == '/') {

strcpy(filename, add[1]);

}

else {

sprintf(filename, "%s%s", curpath, add[1]);

}

if(searchNode(filename) != -1) {

fprintf(stderr, "there is filename in backup list\n");

return;

}

period = atoi(add[2]);

}

if(period < 5 || period > 10) {

fprintf(stderr, "error : period error\n");

return;

}

if(i>3) {

do\_option(i, add); //do option //add->option설정

}

else {

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

add[j] = "0";

}

if(d\_option == 1)

createNode\_d(filename, period, add, d\_option);

else

createNode(filename, period, add, d\_option);

return;

}

void remove\_command(void\* command){

char\* cmd = command;

int i = 0;

char\* ptr = strtok(cmd, " ");

char\* rmv[2];

int exist;

char day[BUFFER\_SIZE];

char time[BUFFER\_SIZE];

char buf[BUFFER\_SIZE];

int sec;

//node \*p = l->head;

while(ptr != NULL) {

rmv[i] = (char\*)malloc(sizeof(ptr));

strcpy(rmv[i], ptr);

ptr = strtok(NULL, " ");

i++;

}

//if it's not remove -a || remove filename

if(i!=2) {

fprintf(stderr, "error\n");

return;

}

//옵션이 있을 경우

//remove -a

if(!strncmp(rmv[1], "-a", 2)) {

FILE\* fp = fopen(logpath, "a+");

printf("-a option\n");

node\* p = l->head;

while(p!=NULL) {

pthread\_cancel(p->tid);

pthread\_cond\_signal(&cond);

pthread\_join(p->tid, NULL);

gettime(day, time, &sec);

sprintf(buf,"[%s %s] %s deleted\n", day, time, p->path\_);

fwrite(buf, strlen(buf), 1, fp);

p = p->next;

}

l->head = NULL;

fclose(fp);

}

//파일이 백업리스트에 없을 경우

else if((exist = searchNode(rmv[1])) < 0) {

fprintf(stderr," there is no file name %s in backup list\n", rmv[1]);

return;

}

else {

removeNode(exist); //파일 백업에서 삭제

}

return;

}

void compare\_command(void\* command) {

char\* cmd = command;

int i=0;

char file[3][BUFFER\_SIZE];

//int f1, f2;

struct stat f1;

struct stat f2;

off\_t size1, size2;

time\_t time1, time2;

char\* cmp = strtok(cmd, " ");

while(cmp != NULL) {

strcpy(file[i], cmp);

cmp = strtok(NULL, " ");

i++;

}

if(i!=3) {

fprintf(stderr, "input files have to be 2\n");

return;

}

if(access(file[1], F\_OK) < 0) {

char buf[BUFFER\_SIZE];

strcpy(buf, file[1]);

sprintf(file[1], "%s/%s", dirpath, buf);

}

if(stat(file[1], &f1) < 0) {

fprintf(stderr, "%s error\n", file[1]);

return;

}

time1 = f1.st\_mtime;

size1 = f1.st\_size;

if(access(file[2], F\_OK) < 0) {

char buf[BUFFER\_SIZE];

strcpy(buf, file[2]);

sprintf(file[2], "%s/%s", dirpath, buf);

}

if(stat(file[2], &f2) < 0) {

fprintf(stderr, "f2 error\n");

return;

}

time2 = f2.st\_mtime;

size2 = f2.st\_size;

if(time1 == time2 && size1 == size2)

printf("두 파일은 같다.\n");

else {

printf("mtime : %ld %ld\n", time1, time2);

printf("size : %ld %ld\n", size1, size2);

}

}

void recover\_command(void\* command) {

char\* cmd = command;

int i=0;

int num;

char \*input[4];

char filename[BUFFER\_SIZE];

backup\_list bf[100]; //backup file list

char\* ptr = strtok(cmd, " ");

while(ptr != NULL) {

input[i] = (char\*)malloc(sizeof(ptr));

strcpy(input[i], ptr);

// printf("%s\n", input[i]);

ptr = strtok(NULL, " ");

i++;

}

if(i > 2) {

char \*p;

if(( p = strstr(input[2], "-n")) == NULL) {

fprintf(stderr, "there is no option name %s\n", input[2]);

return;

}

if(i==3) {

fprintf(stderr, "there is no newfile name\n");

return;

}

if(!access(input[3], F\_OK)) {

fprintf(stderr, "there is file named %s\n", input[3]);

return;

}

else {

if(\*input[3] == '/') {

// printf("start with filepath\n");

strcpy(filename, input[3]);

}

else {

// printf("start with partial path\n");

sprintf(filename, "%s%s", curpath, input[3]);

}

}

}

else {

if(\*input[1] == '/') {

// printf("start with filepath\n");

strcpy(filename, input[1]);

}

else {

// printf("start with partial path\n");

sprintf(filename, "%s%s", curpath, input[1]);

}

}

if((num = searchBF(input[1], bf)) < 0) {

fprintf(stderr, "there is no file for recover\n");

return;

}

else {

for(int j=0;j<num-1 ; j++) {

printf("%d. %8s %dbytes \n",j+1,bf[j].time\_, bf[j].size);

}

printf("Choose file to recover : ");

char buf[4];

fgets(buf, sizeof(buf), stdin);

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

int num = atoi(buf);

//cp bf[num].path\_ filename

char cp[BUFFER\_SIZE];

sprintf(cp,"cp %s/%s %s",dirpath, bf[num-1].path\_, filename);

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

system(cp);

FILE\* fp = fopen(filename, "r");

while(!feof(fp)) {

fgets(cp, sizeof(cp), fp);

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

}

}

}

int searchBF(char \*filename, backup\_list \*bf) {

DIR\* dir = NULL;

struct dirent\* entry = NULL;

int i=0;

struct stat sb;

char path[BUFFER\_SIZE];

if((dir = opendir(dirpath)) == NULL) {

fprintf(stderr, "open dir error\n");

return -1;

}

while((entry = readdir(dir)) != NULL) {

//printf("507 open %s\n", dirpath);

if(strstr(entry->d\_name, filename)!= NULL) {

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

i--;

printf("skip\n");

continue;

}

// printf("%d %s\n", i, entry->d\_name);

strcpy(bf[i].path\_, entry->d\_name);

sprintf(path, "%s/%s", dirpath, entry->d\_name);

stat(path, &sb);

bf[i].size = sb.st\_size;

gettime\_bf(&bf[i]);

i++;

}

}

closedir(dir);

return i;

}

void gettime\_bf(backup\_list \*bf) {

char name[BUFFER\_SIZE];

strcpy(name, bf->path\_);

char \* ptr;

ptr =strstr(name,"\_");

ptr = strtok(ptr, "\_");

strcpy(bf->time\_, ptr);

}

void list\_command(void\* command) {

node \*p = l->head;

while(p!=NULL) {

printf("%s ,%d %s\n", p->path\_, p->period,p->option\_s);

p= p->next;

}

return;

}

void vi\_command(void\* command) {

system(command);

}

void ls\_command(void\* command) {

system(command);

}

void exit\_command(){

node \*p = l->head;

while(p!=NULL) {

pthread\_cancel(p->tid);

p=p->next;

}

}

char\* getfilename(char\* command) {

char\* filename = strtok(command, " ");

filename = strtok(NULL, " ");

strtok(filename, " ");

printf("154 filename %s\n", filename);

return filename;

}

int getperiod(char\* command) {

int period;

char\* buf = strtok(command, " ");

buf = strtok(NULL, " ");

buf = strtok(NULL, " ");

period = atoi(buf);

return period;

}

int check\_option(char\* command) {}

void do\_option(int argc, char\* option[]) {

char c;

char\* buf\_;

int num;

int time;

int buf[4];

int num\_option = 0;

for(int i=0; i<4; i++)

buf[i] = 0;

while((c = getopt(argc, option, "mn:t:d:")) != -1) {

switch(c) {

case 'm' :

buf[0] = 1;

num\_option++;

break;

case 'n' :

// printf("n option\n");

buf\_ = optarg;

if(buf\_ == NULL) {

fprintf(stderr,"there is no num\n");

return;

}

num = atoi(buf\_);

//printf("583 %d\n", num);

if(num <MIN\_NUM ||num > MAX\_NUM) {

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

return;

}

buf[1] = num;

num\_option++;

break;

case 't' :

buf\_ = optarg;

time = atoi(buf\_);

//time 이 정수형이 아닌 실수형 일경우 에러처리

//입력이 없을 시 에러처리 후 프롬르트로 제어가 넘어감

buf[2] = time;

num\_option++;

break;

case '?' :

printf("error");

break;

/\* default:

printf("default\n");

break;\*/

}

}

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

char tmp[BUFFER\_SIZE];

sprintf(tmp, "%d",buf[i]);

strcpy(option[i],tmp);

}

/\*for(int i=0; i<4; i++)

printf("384 %d %s\n",i, option[i]);\*/

}

void createNode\_d(char\* path, int period, char\* option[], int d\_option) {

int i;

void\* num;

node\* newNode;

DIR \*d = opendir(path);

struct dirent\* file = NULL;

while((d = opendir(path)) == NULL) {

fprintf(stderr, "unavailble to opendir %s\n", path);

return;

}

i=0;

while((file = readdir(d)) != NULL) {

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

continue;

}

else {

newNode = (node\*)malloc(sizeof(node));

strcpy(newNode->path, file->d\_name);

strcpy(newNode->path\_, path);

strcat(newNode->path\_, "/");

strcat(newNode->path\_, newNode->path);

newNode->period = period;

strcpy(newNode->option\_s, "-d ");

for(int i=0; option[i]!= '\0'&& i<4; i++) {

newNode->option[i] = atoi(option[i]);

}

//m option

if(newNode->option[0] == 1) {

strcat(newNode->option\_s, "-m ");

struct stat statbuf;

if((stat(newNode->path\_, &statbuf)) <0) {

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

return;

}

newNode->mtime = statbuf.st\_mtime;

}else

newNode->mtime = 0;

//t option

if(newNode->option[2] !=0) {

strcat(newNode->option\_s, "-t ");

newNode->time = newNode->option[2];

}else

newNode->time = 0;

// n option

if(newNode->option[1] !=0) {

strcat(newNode->option\_s, "-n ");

newNode->n\_num = newNode->option[1];

}else

newNode->n\_num =0;

//첫노드

if(l->head == NULL && l->tail == NULL && l->node\_num == -1)

l->head = l->tail = newNode;

else {//추가

l->tail->next = newNode;

l->tail = newNode;

}

l->node\_num++;

if(searchNode(newNode->path\_) < 0) {

fprintf(stderr,"there is a same name of file in the backup list\n");

return;

}

num = &l->node\_num;

pthread\_create(&newNode->tid, NULL, backup, num);

i++;

}

}

}

void createNode(char\* path, int period, char\* option[], int d\_option) {

int i;

void\* num;

node\* newNode;

newNode = (node\*)malloc(sizeof(node));

//strcpy(newNode->path , path);

strcpy(newNode->path\_, path);

newNode->period = period;

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

newNode->option[i] = atoi(option[i]);

}

strcpy(newNode->option\_s, " ");

if(newNode->option[0] == 1) {

struct stat statbuf;

strcat(newNode->option\_s, "-m ");

if((stat(newNode->path\_, &statbuf)) <0) {

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

exit(1);

}

newNode->mtime = statbuf.st\_mtime;

}else

newNode->mtime = 0;

if(newNode->option[2] !=0) {

strcat(newNode->option\_s, "-t ");

newNode->time = newNode->option[2];

}else

newNode->time = 0;

if(newNode->option[1] !=0) {

strcat(newNode->option\_s, "-n ");

newNode->n\_num = newNode->option[1];

}else

newNode->n\_num =0;

if(l->head == NULL && l->tail == NULL && l->node\_num == -1) {

l->head = l->tail = newNode;

}

else {

l->tail->next = newNode;

l->tail = newNode;

}

l->node\_num++;

//printf("772 %d\n", l->node\_num);

num = &l->node\_num;

if(pthread\_create(&newNode->tid, NULL, backup, num) != 0) {

fprintf(stderr,"why??\n");

}

// return;

}

void deleteLastNode() {

node \*p = l->head;

while(p->next->next != NULL)

p = p->next;

p->next = p->next->next;

l->tail = p;

}

void printNode() {

node\*p = l->head;

while(p!= NULL) {

p= p->next;

}

}

int searchNode(char\* filename) {

int i = 0;

int len;

node \*p = l->head;

while(p!= NULL) {

len = strlen(filename);

// printf("compare %s with %s %d\n", filename, p->path\_, len);

//printf("444 %d\n",strncmp(filename, p->path\_, len));

if(strstr(p->path\_, filename)!=NULL) {

// printf("%d\n", i);

return i;

}

i++;

p= p->next;

}

//printf("%s return -1\n", filename);

return -1;

}

void removeNode(int num) {

FILE\* fp = fopen(logpath, "a+");

int i=0;

char time[BUFFER\_SIZE];

char day[BUFFER\_SIZE];

char buf[BUFFER\_SIZE];

int sec;

node \*p = l->head;

//첫번째 노드 지움

if(num == 0) {

char\* oldhead = l->head->path\_;

pthread\_cancel(l->head->tid);

pthread\_cond\_signal(&cond);

//printf("510\n");

pthread\_join(l->head->tid, NULL);

l->head = p->next;

gettime(day, time, &sec);

sprintf(buf,"[%s %s] %s deleted\n", day, time, oldhead);

fwrite(buf, strlen(buf), 1, fp);

fclose(fp);

//히니빆에 안남은 첫번째 노드를 지움

if(l->node\_num == 0) {

//printf("everynode is deleted\n");

l->head = NULL;

l->tail = NULL;

}

l->node\_num --;

return;

}

while(p!= NULL && i<=num) {

if(i+1 == l->node\_num){

pthread\_cancel(l->tail->tid);

pthread\_cond\_signal(&cond);

pthread\_join(l->tail->tid, NULL);

free(l->tail);

l->tail = p;

p->next = NULL;

gettime(day, time, &sec);

sprintf(buf,"[%s %s] %s deleted\n", day, time, p->path\_);

fwrite(buf, strlen(buf), 1, fp);

fclose(fp);

l->node\_num--;

// printf("860\n%d\n", l->node\_num);

return;

}

if(i+1 == num) {

pthread\_cancel(p->next->tid);

pthread\_cond\_signal(&cond);

pthread\_join(p->next->tid, NULL);

p->next = p->next->next;

l->node\_num--;

gettime(day, time, &sec);

sprintf(buf,"[%s %s] %s deleted\n", day, time, p->path\_);

fwrite(buf, strlen(buf), 1, fp);

fclose(fp);

return;

}

p = p->next;

i++;

}

return;

}

//backup 할 파일을 백업 디렉토리에 추가한다.

void\* backup(void\* num) {/////////////////자원정리 함수 추가//////////

FILE \*fp = fopen(logpath, "a+");

char buf[BUFFER\_SIZE];

char day[BUFFER\_SIZE];

char time[BUFFER\_SIZE];

char bufile[BUFFER\_SIZE];

int sec;

int i = 0;

int j;

int no = \*(int\*)num;

node\* file = l->head;

for(i; i<no; i++) {

file = file->next;

}

char \*path = file->path;

char\* path\_ = file->path\_;

int period = file->period;

time\_t mtime = file->mtime;

int t\_time = file->time;

int n\_num = file->n\_num;

char command[BUFFER\_SIZE];

pthread\_cleanup\_push(clean\_up, (void\*)file);

sprintf(command, "cp %s %s/", path\_, dirpath);

//strcat(command, dirpath);

//strcat(command, " ./backup/");

//strcat(command, path);

gettime(day, time, &sec);

sprintf(buf, "[%s %s] %s added\n",day, time, file->path\_);

fwrite(buf, strlen(buf), 1,fp);

fclose(fp);

int bfn = 0;

int m = 0;

while(1) {

if(fopen(logpath, "a+") < 0) {

fprintf(stderr, "log file error\n");

}

char cmd\_buf[BUFFER\_SIZE];

strcpy(cmd\_buf, command);

sleep(period);

//moption 실행

if(mtime != 0) {

struct stat statbuf;

stat(path\_, &statbuf);

if(m == 0) {

mtime =statbuf.st\_mtime;

m++;

}else if(mtime == statbuf.st\_mtime) {

// printf("there was nothing happend\n");

}

else mtime = statbuf.st\_mtime;

}

gettime(day, time, &sec);

sprintf(bufile, "%s\_%s%s",file->path\_,day, time); //절대경로\_날짜

sprintf(buf, "[%s %s] %s generated\n", day, time, bufile);

char ch\_bufile[BUFFER\_SIZE];

changename(bufile, ch\_bufile); //절대경로\_날짜를 파일의 이름으로 저장하기 위해서 /을 =로 바꾼 ch\_bufile을 이름으로 저장

sprintf(file->files[bfn].path\_, "%s/%s", dirpath, ch\_bufile);

file->files[bfn].time = sec;

//cp file->path\_ ./backup/file->path\_daytime

strcat(cmd\_buf, ch\_bufile);

if(strlen(bufile) > 255) {

fprintf(stderr,"this file name is so long\n");

pthread\_exit(NULL);

}

system(cmd\_buf);

fwrite(buf, strlen(buf), 1,fp);

fclose(fp);

bfn++;

//toption 실행

if(t\_time != 0) {

for(i=0; i<bfn; i++) {

gettime(day, time, &sec);

if(sec - t\_time >= file->files[i].time)

if(strstr(file->files[i].path\_,"#")== NULL) {

// printf("%d - %d >= %d\n",sec, t\_time, file->files[i].time);

char rm[BUFFER\_SIZE];

sprintf(rm,"rm %s", file->files[i].path\_);

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

system(rm);

strcpy(file->files[i].path\_, "#");//tombstone

file->files[i].time = MAX\_TIME;

}

}

}

//noption 실행

if(n\_num != 0) {

if(bfn > n\_num) {

for(i=0,j=0; bfn-n\_num > i; i++) {//bfn 5 ..--0

if(strstr(file->files[i].path\_, "#")== NULL) {

char rm[BUFFER\_SIZE];

sprintf(rm, "rm %s", file->files[i].path\_);

strcpy(file->files[i].path\_ ,"#"); //tombstone;

// printf("723 %s\n", rm);

system(rm);

file->files[i].time = MAX\_TIME;

j++;

//bfn--;

}

}

}

}

}

pthread\_cleanup\_pop(0);

}

void changename(char\* file, char\* newfile) {

strcpy(newfile, file);

char\* pch = strstr(newfile, "/");

while(pch!=NULL) {

strncpy(pch, "=", 1);

pch = strstr(newfile, "/");

}

}

void gettime(char\* dy, char\* tm, int \*s) {

time\_t timer;

struct tm \*t;

timer = time(NULL);

t = localtime(&timer);

char time[BUFFER\_SIZE];

char year[10];

sprintf(year, "%02d",t->tm\_year-100);

char mon[10];

sprintf(mon, "%02d",t->tm\_mon+1);

char day[10];

sprintf(day, "%02d",t->tm\_mday);

char hour[10];

sprintf(hour, "%02d",t->tm\_hour);

char min[10];

sprintf(min, "%02d",t->tm\_min);

char sec[10];

sprintf(sec, "%02d", t->tm\_sec);

sprintf(dy, "%s%s%s",year, mon, day);

sprintf(tm,"%s%s%s", hour, min, sec);

\*s = (int)((t->tm\_min\*60) + t->tm\_sec);

}

char\* optblock\_to\_string() {}

void logfile(){}