컴퓨터학부 20202920 조민혁

1.과제개요

리눅스 시스템 상에서 입력한 파일이나 디렉토리를 모니터링하여 자동으로 백업하는 프로그램입니다. add로 모니터링할 경로를 정해주고 이에 대해서 create, remove, modify를 측정하며 해당하는 결과에 대해서 로그에 작성하여 줍니다. remove를 통해 작성된 로그와 백업파일을 삭제할 수 있으며, list명령어를 통해서 적혀있는 로그들을 트리구조의 형태로 볼 수 있습니다. help명령어를 통해 사용할 수 있는 명령어를 확인하고 exit을 통해 프로그램을 종료할 수 있게 하는 프로그램입니다.

2.기능설명

add : 옵션이 없는 경우에는 파일의 경로를 입력받아서 데몬 프로세스를 생성하여 추적하기 시작합니다. add로 모니터링되는 파일의 경우 처음에 create로 새로운 pid.log파일을 생성하여 작성하여 주면 monitor\_list.log 파일에 pid와 원본 경로를 적어줍니다. mtime과 사이즈, hash함수를 이용하여 modify를 판단하고 modify된 경우에는 백업파일을 생성하는 과정을 진행시켜줍니다. 또한 디렉토리 경로가 들어오면 에러를 처리해줍니다.

-d 옵션 : -d옵션을 사용하면 디렉토리에 대해서 모니터링을 할 수 있게 해줍니다. 해당 디렉토리에 들어있는 모든 파일에 대해서 모니터링을 진행시켜줍니다. 파일들이 들어오는 경우에는 에러처리를 진행해줍니다. 디렉토리 안에 있는 파일이 생길 때는 create, 바꿨을 때는 modify를 적고 삭제된 경우는 remove를 로그에 남겨줍니다. 또한 이에 대한 백업파일들도 계속해서 생성시켜줍니다.

-r 옵션 : -d옵션과 다르게 bfs로 재귀적으로 파일을 돌면서 디렉토리 하부의 모든 파일에 대해서 모니터링을 진행시켜줍니다. -d옵션과 같이 입력된 경우 -r옵션으로 간주하여 수행시켜줍니다.

-t옵션 : -t옵션은 time을 인자로 하나 더 입력받아 해당하는 시간의 간격만큼 모니터링을 하도록 설정하는 것입니다. -t옵션을 사용하지 않는 경우에는 기본적으로 1초 간격으로 모니터링해서 변경사항들을 적어줍니다.

remove: remove명령어는 pid를 인자로 전달받아 수행해주는 함수입니다. 만약에 인자로 pid가 들어오지 않는 경우 에러처리를 해줍니다. 들어온 pid가 monitor\_list.log에 없는 경우에도 에러처리를 진행해줍니다. 정확한 값이 들어온 경우에는 monitor\_list.log에서 해당 pid부분을 지워주고 해당 pid의 백업파일과 pid.log파일을 삭제해줍니다.

list: list명령어는 인자를 받지 않고 쓰는 경우에는 monitor\_list.log에 적힌 값들을 출력하고, 그렇지 않은 경우에는 pid를 인자로 받습니다. Pid를 인자를 받은 경우에는 pid.log파일에 적힌 내용들에 대해서 어떤 행동들이 언제 일어났는지에 대해 트리구조로 출력시켜줍니다.

help: help명령어는 각 명령어의 사용법을 출력해줍니다. 인자가 들어온 경우에는 해당 인자인 명령어에 대해서 출력해줍니다.

exit : exit은 프로그램을 종료시켜주는 명령어입니다.

3.상세설계

1)함수 및 모듈 구성

ssu\_sync 프로그램은 처음 실행하는 경우 init함수를 통해 homePATH,exePATH를 설정해주고, 그 후에 20202920>을 계속해서 띄우면서 명령어를 입력받습니다. 명령어가 아닌 것이 들어오는 경우에는 helpExec함수를 수행시켜서 명령어의 사용법을 알려줍니다. add명령어를 통해서 경로를 입력받으면 해당 경로와 같이 들어온 옵션을 판단해줍니다. 옵션과 경로를 판단하였으면 fork함수를 통해서 exec시켜줍니다. AddCommand로 해당하는 값들을 넘긴 후에 옵션에 따라 다른 ssu\_daemon함수들을 호출해줍니다. 해당 함수가 호출되면 daemon프로세스를 생성하여 해당 경로에 대한 모니터링을 수행합니다. 경로에 대해서 각각을 모니터링할 수 있기 때문에 중복된 경로가 들어오는 경우에는 예외처리를 진행시켜줍니다. remove명령어가 들어온 경우 pid를 monitor\_list.log를 읽어 있는지 확인하고 이에 대해서 해당하는 것이 있으면 kill 명령어로 종료시키며 백업파일과 해당 pid.log, pid를 monitor\_list.log에서 지워줍니다. list명령어는 pid가 주어진 경우에는 pid.log에 적힌 내용을 트리구조로 만들어서 출력하고 list만 입력된 경우에는 monitor\_list.log의 내용들을 출력해줍니다.help와 exit명령어는 각각 명령어에 대한 설명을 출력하고 프로그램을 종료하는 함수입니다.

아래는 주된 구성들에 사용되는 구조체들입니다.

//구조체들입니다.

//exec해줄 때 넘겨주는 인자에 대한 구조체입니다.

typedef struct command\_parameter {

char \*command;

char \*filename;

char \*tmpname;

int commandopt;

char \*argv[10];

} command\_parameter;

//path를 구해줄 때 쓰기 위한 구조체이다.

typedef struct pathList\_ {

struct pathList\_ \*next;

struct pathList\_ \*prev;

char path[NAMEMAX];

} pathList;

//각 로그에 적혀 있는 파일에서 어떤 작업들이 이루어졌는지 알려주기 위해서 생성한 구조체입니다.

typedef struct \_backupNode {

struct \_dirNode \*root\_version\_dir;

struct \_fileNode \*root\_file;

char backup\_time[20];

char action[10];

char origin\_path[PATHMAX];

char backup\_path[PATHMAX];

struct \_backupNode \*prev\_backup;

struct \_backupNode \*next\_backup;

} backupNode;

//list명령어에서 사용하기 위해 필요한 file구조체입니다.

typedef struct \_fileNode {

struct \_dirNode \*root\_dir;

int backup\_cnt;

char backup\_time[21];

char file\_name[FILE\_MAX];

char file\_path[PATHMAX];

char made\_path[PATHMAX];

int alive;

time\_t firstMtime;

backupNode \*backup\_head;

struct \_fileNode \*prev\_file;

struct \_fileNode \*next\_file;

} fileNode;

//list명령어를 사용하기 위해서 필요한 dir구조체입니다.

typedef struct \_dirNode {

struct \_dirNode \*root\_dir;

int file\_cnt;

int subdir\_cnt;

int backup\_cnt;

char backup\_time[21];

char dir\_name[FILE\_MAX];

char dir\_path[PATHMAX];

char parent\_path[PATHMAX];

fileNode \*file\_head;

struct \_dirNode \*subdir\_head;

struct \_dirNode\* prev\_dir;

struct \_dirNode \*next\_dir;

} dirNode;

//queue를 만들기 위한 구조체다.

typedef struct queue{

struct queue \*first;

struct queue \*last;

char path[PATHMAX];

struct queue \*next;

}queue;

위의 구조체를 들을 사용하여 ssu\_struct.c의 함수들을 수행시킵니다.

//로그를 표현하기 위해서 사용하는 action에 대한 링크드리스트를 만들어주는 함수입니다.

int backup\_list\_insert(dirNode\* dirList, char\* backup\_time, char\* path, char\* action) {

char\* ptr;

path = path;

if(ptr = strchr(path, '/')) {

char\* dir\_name = (char \*)malloc(sizeof(char)\*PATHMAX);

dir\_name = substr(path, 0, strlen(path) - strlen(ptr));

dirNode\* curr\_dir = dirNode\_insert(dirList->subdir\_head, dir\_name, dirList->dir\_path,backup\_time);

backup\_list\_insert(curr\_dir, backup\_time, ptr+1, action);

curr\_dir->backup\_cnt++;

} else {

char \*file\_name = (char \*)malloc(sizeof(char)\*FILE\_MAX);

strcpy(file\_name,path);

fileNode\* curr\_file = fileNode\_insert(dirList->file\_head, file\_name, dirList->dir\_path,backup\_time);

backupNode\_insert(curr\_file->backup\_head, backup\_time,action ,file\_name, dirList->dir\_path);

}

return 0;

}

이 함수는 ssu\_struct.c에서 주된 역할을 하는 함수로서 list명령어를 구현할 때 주로 사용되는 함수입니다.

경로를 하나씩 끊어가며 노드를 구성하는 식으로 작동합니다.

다음으로 가장 중요한 함수는

//큐도 초기화해서 만들어서 넣어줘야합니다.

//BFS를 사용하기 위해서 재귀적으로 반복하여 모니터링하는 함수입니다.

int backupRecursive(queue\* Queue,char \*path,dirNode \*dir\_head,char \*backupLog,pid\_t currentPid){

struct stat statbuf,tempbuf;

struct stat currentbuf,backupbuf;

struct dirent \*\*namelist;

FILE\* fp;

int length,fd3,fd4;

char \*forDateBuffer = (char \*)malloc(sizeof(char)\*BUF\_MAX);

char\* tempPath = (char \*)malloc(sizeof(char \*)\*PATHMAX);

char \*hash1 = (char \*)malloc(sizeof(char)\*BUF\_MAX);

char \*hash2 = (char \*)malloc(sizeof(char)\*BUF\_MAX);

char \*forWriteLog22 = (char \*)malloc(sizeof(char)\*PATHMAX\*3);

char \*madePath = (char \*)malloc(sizeof(char)\*PATHMAX);

char \*gotDate = (char \*)malloc(sizeof(char)\*15);

char \*buf = (char \*)malloc(sizeof(char)\*BUF\_MAX);

//새로운 파일링크드리스트를 생성해주는 것입니다.

fileNode \*headFile = (fileNode\*)malloc(sizeof(fileNode));

fileNode\_init(headFile);

fileNode \*forCheckFile;

char \*nextDir;

//비교하기 위한 디렉토리 링크드리스트를 만들 위한 헤드를 만들어줍니다

dirNode \*compareDirHead = (dirNode \*)malloc(sizeof(dirNode));

dirNode\_init(compareDirHead);

int cnt;

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

fprintf(stderr,"ERROR: lstat cuase error\n");

return 1;

}

if((cnt = scandir(path,&namelist,NULL,alphasort)) == -1){

fprintf(stderr,"ERROR: cannot scan dir.\n");

return 1;

}

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

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

if(strstr(namelist[i]->d\_name,".swp")!=NULL)continue;

sprintf(tempPath,"%s/%s",path,namelist[i]->d\_name);

if(lstat(tempPath,&tempbuf)<0){

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

exit(1);

}

if(S\_ISDIR(tempbuf.st\_mode)){

enqueue(Queue,tempPath);

//삭제된 디렉토리가 있는 경우에는 노드에서 제거해줍니다.

//그러기 위해서 비교하기 위한 리스트를 만들어줍니다.

dirNode\_insert\_forR(compareDirHead,NULL,tempPath,path);

} //디렉토리인 경우에는 queue에 추가해줍니다.

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

//파일인 경우에는 파일 링크드리스트에 추가해줍니다.

//파일인 경우에는 create,modified에 대한 처리를 해주어야합니다.

//아래는 create랑 같이 나오면 되는 것이다.

if(dirNode\_get(dir\_head,path)==NULL){

//directory도 없어서 해당 디렉토리 안의 파일들이 처음 존재하는 경우입니다.

fileNode\_insert\_forD(headFile,tempPath);

strcpy(forDateBuffer,makeDate());

fp = fopen(backupLog,"a+");

fseek(fp,0,SEEK\_END);

sprintf(forWriteLog22,"[%s][create][%s]\n",forDateBuffer,tempPath);

fwrite(forWriteLog22,strlen(forWriteLog22),1,fp);

sscanf(forDateBuffer, "%4s-%2s-%2s %2s:%2s:%2s",gotDate, gotDate + 4, gotDate + 6,gotDate + 8, gotDate + 10, gotDate + 12);

sprintf(madePath,"%s/%d/%s\_%s",backupPATH,currentPid,GetFileName(tempPath),gotDate);

fileNode\* hiNode = fileNode\_get(headFile,tempPath);

strcpy(hiNode->made\_path,madePath);

fd3 = open(tempPath,O\_RDONLY);

fd4 = open(madePath,O\_RDWR|O\_CREAT,0777);

//파일을 복구하여 줍니다.

while((length = read(fd3,buf,BUF\_MAX))>0){

write(fd4,buf,length);

}

close(fd3);

close(fd4);

fclose(fp);

}else{

dirNode \*currencyNode;

if((currencyNode=dirNode\_get(dir\_head,path))!=NULL){

if(fileNode\_get(currencyNode->file\_head,tempPath)==NULL){

//여기서 create를 적어주는 것입니다.

fileNode\_insert\_forD(headFile,tempPath);

fileNode\_insert\_forD(currencyNode->file\_head,tempPath);

strcpy(forDateBuffer,makeDate());

fp = fopen(backupLog,"a+");

fseek(fp,0,SEEK\_END);

sprintf(forWriteLog22,"[%s][create][%s]\n",forDateBuffer,tempPath);

fwrite(forWriteLog22,strlen(forWriteLog22),1,fp);

sscanf(forDateBuffer, "%4s-%2s-%2s %2s:%2s:%2s",gotDate, gotDate + 4, gotDate + 6,gotDate + 8, gotDate + 10, gotDate + 12);

sprintf(madePath,"%s/%d/%s\_%s",backupPATH,currentPid,GetFileName(tempPath),gotDate);

fileNode\* hiNode = fileNode\_get(currencyNode->file\_head,tempPath);

strcpy(hiNode->made\_path,madePath);

fd3 = open(tempPath,O\_RDONLY);

fd4 = open(madePath,O\_RDWR|O\_CREAT,0777);

//파일을 복구하여 줍니다.

while((length = read(fd3,buf,BUF\_MAX))>0){

write(fd4,buf,length);

}

close(fd3);

close(fd4);

fclose(fp);

}else{

//여기서 modified판단하는 것입니다.

fileNode \*currencyFile;

if((currencyFile=fileNode\_get(currencyNode->file\_head,tempPath))!=NULL){

stat(currencyFile->made\_path,&backupbuf);

stat(currencyFile->file\_path,&currentbuf);

if(backupbuf.st\_size == currentbuf.st\_size){

if(currencyFile->firstMtime == currentbuf.st\_mtime){

md5(currencyFile->made\_path,hash1);

md5(currencyFile->file\_path,hash2);

if(strcmp(hash1,hash2)==0){

//두 파일이 변한 것 없이 아예 같은 경우이기 때문에 무시해주면 됩니다.

}

else{

//해쉬값이 다른 경우입니다.

fp = fopen(backupLog,"a+");

currencyFile->firstMtime = currentbuf.st\_mtime;

strcpy(forDateBuffer,makeDate());

sscanf(forDateBuffer, "%4s-%2s-%2s %2s:%2s:%2s",gotDate, gotDate + 4, gotDate + 6,gotDate + 8, gotDate + 10, gotDate + 12);

sprintf(madePath,"%s/%d/%s\_%s",backupPATH,currentPid,currencyFile->file\_name,gotDate);

strcpy(currencyFile->made\_path,madePath);

//파일을 복구하여 줍니다.

fd3 = open(currencyFile->file\_path,O\_RDONLY);

fd4 = open(madePath,O\_RDWR|O\_CREAT,0777);

while((length = read(fd3,buf,BUF\_MAX))>0){

write(fd4,buf,length);

}

close(fd3);

close(fd4);

fseek(fp,0,SEEK\_END);

sprintf(forWriteLog22,"[%s][modify][%s]\n",forDateBuffer,currencyFile->file\_path);

fwrite(forWriteLog22,strlen(forWriteLog22),1,fp);

fclose(fp);

}

}else{

//mtime이 다른 경우입니다.

fp = fopen(backupLog,"a+");

currencyFile->firstMtime = currentbuf.st\_mtime;

strcpy(forDateBuffer,makeDate());

sscanf(forDateBuffer, "%4s-%2s-%2s %2s:%2s:%2s",gotDate, gotDate + 4, gotDate + 6,gotDate + 8, gotDate + 10, gotDate + 12);

sprintf(madePath,"%s/%d/%s\_%s",backupPATH,currentPid,currencyFile->file\_name,gotDate);

strcpy(currencyFile->made\_path,madePath);

//파일을 복구하여 줍니다.

fd3 = open(currencyFile->file\_path,O\_RDONLY);

fd4 = open(madePath,O\_RDWR|O\_CREAT,0777);

while((length = read(fd3,buf,BUF\_MAX))>0){

write(fd4,buf,length);

}

close(fd3);

close(fd4);

fseek(fp,0,SEEK\_END);

sprintf(forWriteLog22,"[%s][modify][%s]\n",forDateBuffer,currencyFile->file\_path);

fwrite(forWriteLog22,strlen(forWriteLog22),1,fp);

fclose(fp);

}

}else{

//size가 다른 경우입니다.

fp = fopen(backupLog,"a+");

currencyFile->firstMtime = currentbuf.st\_mtime;

strcpy(forDateBuffer,makeDate());

sscanf(forDateBuffer, "%4s-%2s-%2s %2s:%2s:%2s",gotDate, gotDate + 4, gotDate + 6,gotDate + 8, gotDate + 10, gotDate + 12);

sprintf(madePath,"%s/%d/%s\_%s",backupPATH,currentPid,currencyFile->file\_name,gotDate);

strcpy(currencyFile->made\_path,madePath);

//파일을 복구하여 줍니다.

fd3 = open(currencyFile->file\_path,O\_RDONLY);

fd4 = open(madePath,O\_RDWR|O\_CREAT,0777);

while((length = read(fd3,buf,BUF\_MAX))>0){

write(fd4,buf,length);

}

close(fd3);

close(fd4);

fseek(fp,0,SEEK\_END);

sprintf(forWriteLog22,"[%s][modify][%s]\n",forDateBuffer,currencyFile->file\_path);

fwrite(forWriteLog22,strlen(forWriteLog22),1,fp);

fclose(fp);

}

}

}

}

}

}

}

//이 함수는 이미 존재하는 경우에는 추가하지 않습니다.

dirNode\_insert\_forR(dir\_head,headFile,path,getParentPath(path));

//parent가 같은 것 중에서 비교하면 remove를 찾을 수 있습니다.

dirNode \*currentDir;

//해당디렉토리안의 파일이 지워진경우

if((currentDir=dirNode\_get(dir\_head,path))!=NULL){

fileNode \*forFile;

forFile = currentDir->file\_head;

forFile = forFile->next\_file;

while(forFile!=NULL){

printf("list : %s\n",forFile->file\_path);

if(stat(forFile->file\_path,&statbuf)<0){

fp = fopen(backupLog,"a+");

fseek(fp,0,SEEK\_END);

sprintf(forWriteLog22,"[%s][remove][%s]\n",makeDate(),forFile->file\_path);

fwrite(forWriteLog22,strlen(forWriteLog22),1,fp);

fileNode \*eraseNode = forFile;

forFile = forFile->next\_file;

//노드를 링크드리스트에서 제거해줍니다.

fileNode\_remove(eraseNode);

fclose(fp);

}

if(forFile==NULL){

continue;

}

forFile = forFile->next\_file;

}

}

// 디렉토리 자체가 삭제된 경우

currentDir = dir\_head->next\_dir;

dirNode \*compareDir;

while(currentDir!=NULL){

compareDir = compareDirHead->next\_dir;

while(compareDir!=NULL){

if(strcmp(currentDir->parent\_path,compareDir->parent\_path)==0){

if(dirNode\_get(compareDirHead,currentDir->dir\_path)==NULL){

//remove된 것입니다.

//dir자체가 지워진 상태입니다.

fileNode \*loopFile;

loopFile=currentDir->file\_head;

loopFile = loopFile->next\_file;

while(loopFile!=NULL){

fp = fopen(backupLog,"a+");

fseek(fp,0,SEEK\_END);

sprintf(forWriteLog22,"[%s][remove][%s]\n",makeDate(),loopFile->file\_path);

fwrite(forWriteLog22,strlen(forWriteLog22),1,fp);

fileNode \*eraseNode = loopFile;

loopFile = loopFile->next\_file;

//노드를 링크드리스트에서 제거해줍니다.

fileNode\_remove(eraseNode);

fclose(fp);

}

dirNode \*eraseDir = currentDir;

currentDir = currentDir->next\_dir;

dirNode\_remove(eraseDir);

break;

}

}else if (currentDir->next\_dir==NULL){

if(stat(currentDir->dir\_path,&statbuf)<0){

fileNode \*loopFile;

loopFile=currentDir->file\_head;

loopFile = loopFile->next\_file;

while(loopFile!=NULL){

fp = fopen(backupLog,"a+");

fseek(fp,0,SEEK\_END);

sprintf(forWriteLog22,"[%s][remove][%s]\n",makeDate(),loopFile->file\_path);

fwrite(forWriteLog22,strlen(forWriteLog22),1,fp);

fileNode \*eraseNode = loopFile;

loopFile = loopFile->next\_file;

//노드를 링크드리스트에서 제거해줍니다.

fileNode\_remove(eraseNode);

fclose(fp);

}

dirNode \*eraseDir = currentDir;

currentDir = currentDir->next\_dir;

dirNode\_remove(eraseDir);

break;

}

}

compareDir = compareDir->next\_dir;

}

if(currentDir ==NULL){

continue;

}

currentDir = currentDir->next\_dir;

}

if(Queue->first ==NULL){

return 0;

}

nextDir=dequeue(Queue);

backupRecursive(Queue,nextDir,dir\_head,backupLog,currentPid);

}

입니다. 이는 -r로 bfs를 통해서 모든 파일을 모니터링을 할 때 사용하는 함수로서 scandir을 통해서 디렉토리와 파일들을 판단하고 이에 대한 링크드리스트와 디렉토리에 대한 큐를 생성하여 bfs진행해 나갑니다. 각 수행시마다 dir링크드리스트를 생성하여 삭제된 디렉토리를 찾아서 해당 파일들에 대해 remove해줄 수 있으며, 해당 경로안의 모든 파일과 하위 디렉토리의 파일들에 대해서 time으로 입력받은 시간만큼의 간격으로 이를 수행해줍니다. 또한 전체 dir링크드리스트는 함수를 수행하는 중간에 계속 공유함으로 이를 통해서 dir가 삭제되었는지 판단할 수 있게 됩니다. 제가 만든 3가지의 데몬함수는 위의 데몬함수와 비슷하게 동작하여 하나만 첨부하였습니다.

2)함수 프로토타입

//help.c에 있는 함수원형들 입니다.

void printUsage();

void printAdd();

void printRemove();

void printList();

void printHelp();

void printExit();

void printAll();

직관적인 이름 그대로 쓰여있는 것들을 출력합니다.

//ssu\_header.c에 있는 함수원형들입니다.

int realpath\_2\_relative(char \*realpath, char \*returnPath);

절대경로를 상대경로로 바꿔주는 함수입니다.

int ConvertPath(char\* origin, char\* resolved);

상대경로가 들어왔을 때 절대경로로 바꿔주는 함수입니다.

char \*\*GetSubstring(char \*str, int \*cnt, char \*del);

문자열을 원하는 del을 기준으로 쪼개서 문자열 배열의 형태로 리턴하는 함수입니다.

char \*Tokenize(char \*str, char \*del);

마찬가지로 문자열을 쪼개주는 함수입니다.

int md5(char \*target\_path, char \*hash\_result);

내용의 변화를 확인하기 위해서 md5로 해싱하기 위해서 필요한 함수입니다.

int ssu\_daemon\_init(char\*, char\*,unsigned int);

add명령어에서 일반적인 파일 경로가 주어졌을 때 daemone프로세스를 생성하는 함수입니다. 내부에서 해당 경로에 대한 변화를 인자로 받은 time만큼 확인하면서 변화의 내용을 로그에 적어주고 백업파일을 만들어주는 역할을 하는 함수입니다.

char \*GetFileName(char file\_path[]);

경로가 주어졌을 때 해당 경로에서 파일의 이름을 알아내는 역할을 하는 함수입니다.

char \*makeDate();

로그에 작성되기 편한 형태로 시간을 리턴해주는 함수입니다.

char \*dateForm(char\*);

날짜 데이터를 14자리의 문자열로 바꿔주는 함수입니다.

void sigusr\_handler(int signum);

SIGUSR1이 kill로 보내졌을 때 해당 데몬 프로세스를 종료하기 위한 함수입니다.

int deleteDirectory(const char\* dirname);

디렉토리를 순회하면서 디렉토리안에 있는 파일과 디렉토리와 하위 디렉토리안의 파일들까지 전부 삭제시켜주는 함수입니다.

char \*substr(char \*str, int beg, int end);

문자열을 원하는 시작점부터 끝나는 지점까지를 잘라서 리턴시켜주는 함수입니다.

int ssu\_daemon\_init\_dir(char\* currentPath,char \*dirName,unsigned int time);

-d옵션이 들어왔을 때 해당 디렉토리안에 있는 파일들에 대해 모니터링하는 함수입니다.

char \*getParentPath(char \*path);

parent경로를 구해서 리턴시켜줍니다.

void enqueue(queue\* Queue, char\* dirPath);

queue를 연결해주는 함수입니다.

char\* dequeue(queue\* Queue);

queue에서 하나르 제거해주는 함수입니다.

int ssu\_daemon\_init\_dirR(char\* currentPath,char \*dirName,unsigned int time);

-r옵션이 들어온 경우에 데몬프로세스를 생성시켜주는 함수입니다. 해당 함수를 실행하면 bfs로 디렉토리를 순회하며 해당 경로의 하위 디렉토리의 파일들까지 전부 모니터링하여 로그에 작성하고 백업해주는 함수입니다.

//ssu\_struct.c에 있는 함수원형들입니다.

int init\_backup\_list(int log\_fd,char\* getPath);

list명령어를 수행하기 위해서 pid.log에 적힌 값들을 분리하여 링크드리스트에 넣어주는 역할을 합니다.

void print\_list(dirNode\* dirList, int depth, int last\_bit);

list명령어를 수행하였을 때 사용하여 로그에 적힌 값들을 트리구조로 함수입니다.

void print\_depth(int depth, int is\_last\_bit);

트리구조를 만들어주는 함수입니다.

dirNode \*dirNode\_insert(dirNode\* dir\_head, char\* dir\_name, char\* dir\_path,char\*backup\_time);

list를 위해 필요한 dirNode를 만들어주는 함수입니다.

void dirNode\_init(dirNode \*dir\_node);

dirNode를 초기화시켜주는 함수입니다.

fileNode \*fileNode\_insert(fileNode \*file\_head, char \*file\_name, char \*dir\_path,char \*backup\_time);

list를 위해 필요한 fileNode를 만들어주는 함수입니다.

int backup\_list\_insert(dirNode\* dirList, char\* backup\_time, char\* path, char\* backup\_path);

list를 위해 필요한 backup\_list에서 사용되는 함수입니다.

void fileNode\_init(fileNode \*file\_node);

fileNode를 초기화해주는 함수입니다.

void fileNode\_remove(fileNode \*file\_node);

노드를 인자로 넣으면 해당 파일노드를 링크드리스트에서 삭제시키는 함수입니다.

fileNode \*fileNode\_insert\_forD(fileNode \*file\_head, char \*file\_path);

D옵션에서 사용되는 filenode를 추가하는 함수입니다

fileNode \*fileNode\_get(fileNode \*file\_head, char \*file\_name);

원하는 경로를 입력하여 링크드리스트에서 해당하는 파일노드를 찾아서 리턴시키는 함수입니다.

dirNode \*dirNode\_get(dirNode\* dir\_head, char\* dir\_path);

원하는 경로를 입력하여 링크드리스트에서 해당하는 디렉토리노드를 찾아서 리턴시키는 함수입니다.

void dirNode\_remove(dirNode \*dir\_node);

디렉토리 노드를 입력받아 해당 디렉토리 노드를 삭제하는 함수입니다.

int backupRecursive(queue\* Queue,char \*path,dirNode \*dir\_head,char \*backupLog,pid\_t currentPid);

BFS를 사용하기 위해서 재귀적으로 반복하여 모니터링하는 함수입니다.

dirNode \*dirNode\_insert\_forR(dirNode\* dir\_head,fileNode \*fileHead, char\* dir\_path,char \*parentPath);

-r옵션에서 dirNode를 추가하기 위해서 사용하는 옵션입니다.

//ssu\_sync.c에서 사용되는 함수들입니다.

int ListCommand(command\_parameter \*parameter)

parameter값을 입력받아 해당값을 통해 list명령어를 수행시켜줍니다.

int RemoveCommand(command\_parameter \*parameter)

pid를 입력받아서 임시파일을 만들어 삭제할 pid를 제거할 로그를 만들고 다시 넘겨주는 역할을 하는 함수입니다.

int AddCommand(command\_parameter \*parameter)

add명령어를 수행하기 위해서 옵션에 따라 다른 데몬프로세스를 실행시켜주는 함수이며 이 함수에서 경로가 파일인지 디렉토리인지 판단하여 줍니다.

int HelpCommand(command\_parameter \*parameter)

help명령어를 수행해주는 함수입니다.

void HelpExec()

잘못된 값이 입력되었을 때 전체 명령어에 대한 설명을 출력해주는 함수입니다.

void CommandFun(char \*\*arglist)

fork()되었을 때 어떤 함수를 실행할지 설정해주는 함수입니다.

void CommandExec(command\_parameter parameter)

parameter값들을 벡터형태로 저장해서 넘겨주고 fork시켜주는 함수입니다.

int ParameterProcessing(int argcnt, char \*\*arglist, int command, command\_parameter \*parameter)

인자로 받은 것에 대한 처리를 진행해주는 함수입니다.

void ParameterInit(command\_parameter \*parameter)

parameter를 초기화해주는 함수입니다.

3)순서도

도표, 평면도, 개략도, 기술 도면이(가) 표시된 사진

자동 생성된 설명

4.실행결과

1)add 명령어를 실행한 경우

: add명령어만을 수행했을 경우입니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

-d옵션

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

-t옵션

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

-r옵션텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

2)remove명령어를 수행한 경우

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

3)list명령어를 입력한 경우

스크린샷, 텍스트, 패턴이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

4)help명령어를 수행한 경우

잘못된 값을 입력한 경우

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

5)exit명령어를 수행한 경우

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명