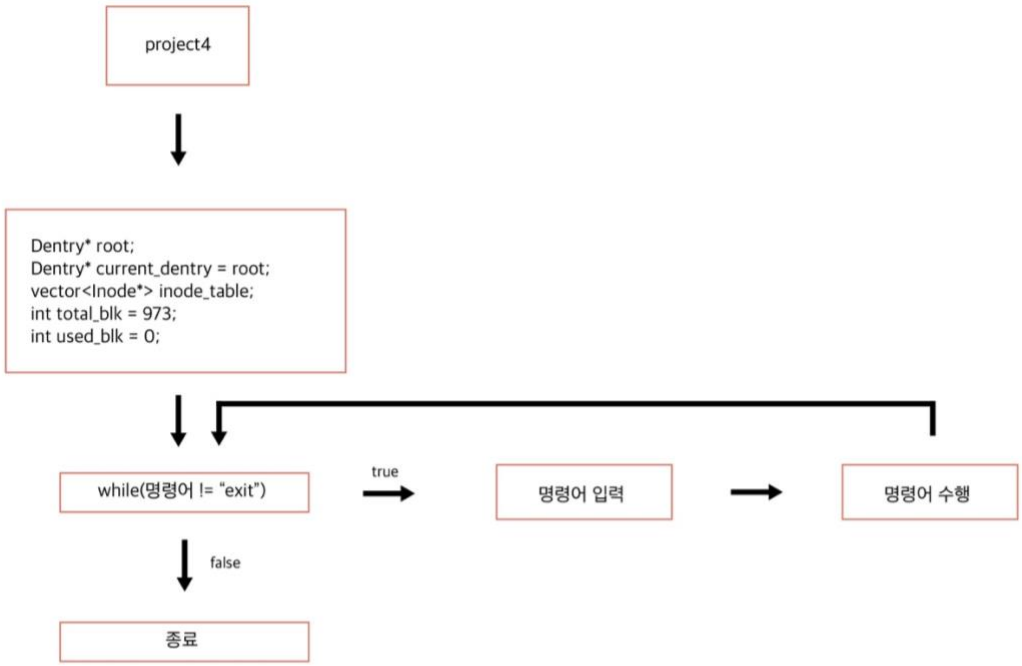
Assignment 4

2015198005
허진욱

1. 작성한 프로그램의 동작 과정과 구현 방법

a) main

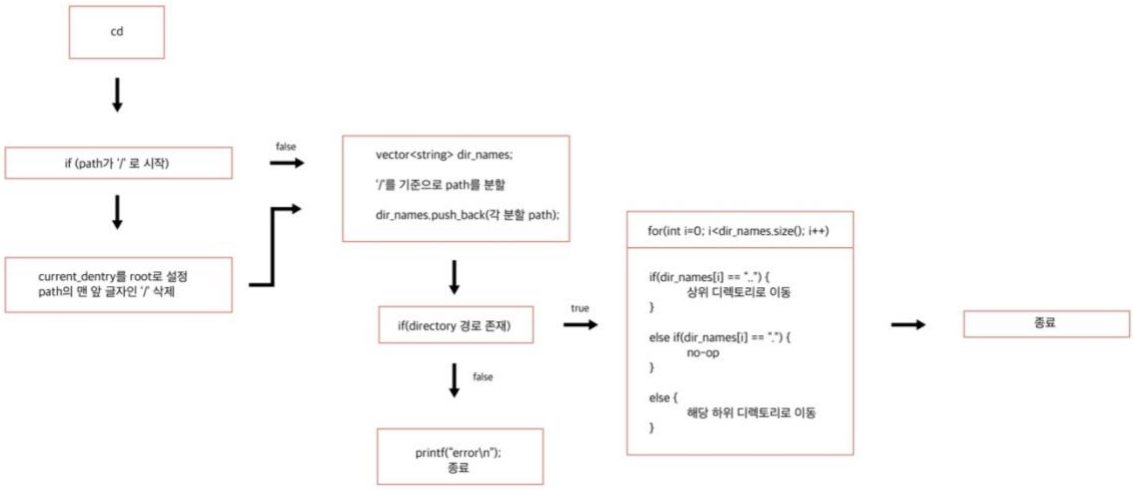


우선 프로그램이 시작하면 root directory, inode table, 전체 disk block 수, 현재 사용된 disk block 수에 대한 변수를 정의한다. 이후 while 문을 돌면서 "exit" 명령어가 나오기 전까지 인풋으로 주어진 명령을 수행한다.

b) ls

current_dentry 에 속한 모든 하위 디렉토리를 출력한다. 이후 하위 파일들도 출력을 해야하는데 이는 id 가 작은 순서로 출력해야한다. 이를 수행하기 위해 파일들의 id 만을 담은 priority_queue 를 생성한 후, 그 순서대로 id 를 하위 파일들에서 찾아 출력한다.

c) cd

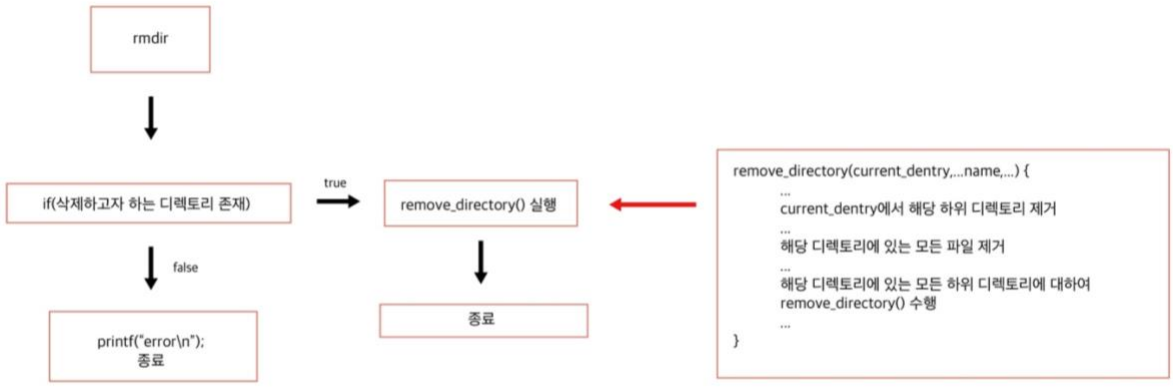


cd 명령이 수행되면 우선 인풋으로 들어온 디렉토리 경로가 '/'로 시작하는지 확인한다. 만약 true 면 경로탐색 시작을 root 로 설정한다. 이후 경로를 '/'를 기준으로 분할을 한 후, 각각의 string 을 dir_names 라는 vector 에 넣는다. 인풋으로 들어온 경로가 존재하지 않으면 에러와 함께 명령을 종료한다. 만약 경로가 존재한다면, for 문을 이용해 각각의 경로 string 에 대하여 처리를 해준다. '.'면 아무 동작을 하지 않고, '..'면 상위 디렉토리로 이동한다. 만약 string 이 하위 디렉토리 이름이면 해당 디렉토리로 이동한다. 모든 경로 string 에 대한 처리가 완료되면 명령을 종료한다.

d) mkdir

mkdir 명령이 들어오면 우선 디렉토리 이름을 인풋으로 받은 후 Dentry 객체를 생성한다. 이후 이 객체를 current_dentry 의 하위 디렉토리에 추가한다.

e) rmdir



우선 삭제하고자하는 디렉토리가 current_dentry 에 있는지 확인한다. 만약 없다면 에러를 출력한 후 명령 수행을 종료한다. 만약 존재한다면 remove_directory() 함수를 실행한다. 이 함수는 우선 current_dentry 에서 해당 디렉토리를 삭제한다. 이후 해당 디렉토리의 파일들을 모두 삭제한 후, 모든 하위 디렉토리에 대하여 recursive 하게 remove_directory()함수를 콜 한다.

f) mkfile

생성하고자 하는 파일의 이름과 크기를 인풋으로 받는다. 만약 파일의 크기가 너무 커 남아있는 disk block 으로 부족하거나, 현재 생성된 파일의 수가 128 이라면, 에러를 출력한 후 명령을 종료한다. 그렇지 않다면 Inode 객체를 생성한 후 current_dentry 에 해당 객체를 추가한다. 이후 used_blk 를 해당 파일 크기만큼 증가 시킨다.

g) rmfile

삭제하고자 하는 파일의 이름을 인풋으로 받은 후, 해당 이름을 current_dentry 에서 찾는다. 만약 존재하지 않는다면 에러를 출력한 후 종료한다. 존재한다면 해당 파일을 삭제한 후 파일이 차지했던 disk block 만큼 used_blk 를 감소시킨다. 해당 과정은 이전 rmdir 에서 각 디렉토리의 하위 파일들을 삭제할 때 동일하게 적용된다.

h) inode

인풋으로 들어온 파일의 이름을 current_dentry 에서 찾는다. 존재한다면 해당 파일의 정보를 출력한다. 존재하지 않는다면 에러를 출력한 후 종료한다.

i) exit

프로그램을 종료한다.

2. 개발 환경 명시

i) uname -a 실행 결과

```
jinwook@jinwook-ubuntu:~$ uname -a
Linux jinwook-ubuntu 5.4.33-2015198005 #1 SMP Tue Apr 21 03:29:54 KST 2020 x86_6
4 x86_64 x86_64 GNU/Linux
```

ii) 사용한 컴파일러 버전

```
gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)
```

iii) CPU 코어 수

```
jinwook@jinwook-ubuntu:~$ grep -c processor /proc/cpuinfo
4
```

iv) 운영체제 버전

```
jinwook@jinwook-ubuntu:~$ cat /etc/issue
Ubuntu 18.04.4 LTS \n \l
```

v) 메모리 정보

```
jinwook@jinwook-ubuntu:~$ cat /proc/meminfo
MemTotal:        2034964 kB
MemFree:          126004 kB
MemAvailable:     836692 kB
Buffers:          152484 kB
Cached:           635900 kB
SwapCached:         1212 kB
Active:           925428 kB
Inactive:         706352 kB
Active(anon):     461028 kB
Inactive(anon):   397584 kB
Active(file):     464400 kB
Inactive(file):   308768 kB
Unevictable:          16 kB
Mlocked:              16 kB
SwapTotal:       1942896 kB
SwapFree:        1918400 kB
Dirty:                 0 kB
Writeback:             0 kB
AnonPages:        842864 kB
Mapped:           149988 kB
Shmem:             15216 kB
KReclaimable:     125416 kB
Slab:             171024 kB
SReclaimable:     125416 kB
SUnreclaim:        45608 kB
KernelStack:        9056 kB
PageTables:        38736 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:     2960376 kB
Committed_AS:    4529480 kB
VmallocTotal:   34359738367 kB
VmallocUsed:       45980 kB
VmallocChunk:          0 kB
Percpu:             1296 kB
HardwareCorrupted:     0 kB
AnonHugePages:         0 kB
ShmemHugePages:        0 kB
ShmemPmdMapped:        0 kB
FileHugePages:         0 kB
FilePmdMapped:         0 kB
CmaTotal:              0 kB
CmaFree:               0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
Hugetlb:               0 kB
DirectMap4k:      163776 kB
DirectMap2M:     1933312 kB
```

3. 과제 수행 중 발생한 애로사항 및 해결방법

i) cd 명령어 구현

cd 명령어와 함께 입력 되는 path 의 형태가 매우 다양하여 구현하는 과정에서 매우 복잡했다. 입력된 path 를 / 단위로 나누어, 하나의 디렉토리를 하나의 명령어로 취급하여 구현하였다. /로 나누었을 때 각각 만들어지는 string 의 종류는 '.', '..', '디렉토리 이름' 세가지이다.

```
stringstream d(dir);
vector<string> dir_names;
string dirs;
char cut = '/';
while(getline( &: d,  &: dirs, cut))
    dir_names.push_back(dirs);  //split directory command
```

```
for(int i=0; i<dir_names.size(); i++) {
    if(dir_names[i] == ".") {
    }
    else if(dir_names[i] == "..") {
        current_dentry = current_dentry->parent;
    }
    else {
        for(int j=0; j<current_dentry->d_dentry.size(); j++) {
            if(dir_names[i] == current_dentry->d_dentry[j]->name) {
                current_dentry = current_dentry->d_dentry[j];
                break;
            }
        }
    }
}
```

ii) rmdir 의 구현

rmfile 과는 달리 rmdir 은 단순히 디렉토리를 없애는 것이 아닌 하위 파일들과 디렉토리, 그리고 그보다 하위에 있는 모든 객체들을 삭제해야 한다. 디렉토라마다 하위 디렉토리의 개수는 모두 다르기 때문에 이는 recursion 을 이용하여 구현해야한다.

```
for(int i=0; i<dentry_size; i++) {
    string dir_name = dentry_names[i];
    remove_directory(p,  &: inode_table, dir_name,  &: used_blk);
}
```

모든 파일을 지운 후, 마지막 부분에 for 문을 통해 모든 하위 디렉토리에 대하여 remove-directory 함수를 다시 부른다.