OS Assignment #4

2015147531 서기원

1. 개발 환경

```
giwon@giwon-VirtualBox:~$ uname -a
Linux giwon-VirtualBox 5.4.28-2015147531 #1 SMP Sun Mar 29 02:17:02 KST 2020 x86_64 x86_64 x86_64 GNU/Linux
```

```
giwon@giwon-VirtualBox:~$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 60
model name      : Intel(R) Core(TM) i5-4210M CPU @ 2.60GHz
stepping        : 3
cpu MHz         : 2594.004
cache size      : 3072 KB
physical id     : 0
siblings        : 2
core id         : 0
cpu cores       : 2
apicid          : 0
initial apicid  : 0
```

```
giwon@giwon-VirtualBox:~$ free -h
              total        used        free      shared  buff/cache   available
Mem:           3.8G        791M        2.0G         27M        1.1G        2.8G
스왑:          2.0G          0B        2.0G
```

2. 프로그램의 기본 구성

- 구조체

```
struct dentry {
    string name;
    int d_num = 0;
    int i_num = 0;

    struct dentry *parent;
    struct dentry *d_dentry[64];
    struct inode *d_inode[128];
};
```

File System 내의 디렉토리를 나타내는 구조체입니다.

name : 디렉토리의 이름

d_num : 해당 디렉토리 내의 디렉토리의 개수
I_num : 해당 디렉토리 내의 파일의 개수
*parent : 상위 디렉토리를 가리키는 포인터
*d_dentry : 하위 디렉토리들을 가르키는 포인터 집합
*d_inode : 해당 디렉토리 내의 파일 inode를 가리키는 포인터 집합

```
struct inode {
    int id;
    string name;
    int size;

    int direct_block;
    int single_indirect;
    int double_indirect;
};
```

id : 해당 inode의 id를 저장하는 변수
name : 해당 inode가 담당하는 파일의 이름
size : 해당 inode가 담당하는 파일의 크기
direct_block : direct block에 할당된 블록의 개수
single_indirect : single indirect block에 할당된 블록의 개수
double_indirect : double indirect block에 할당된 블록의 개수
(single_indrect, double_indirect의 경우, 실제 파일이 저장된 블록의 포인터들을 저장하는 블록도 개수에 포함하였습니다.)

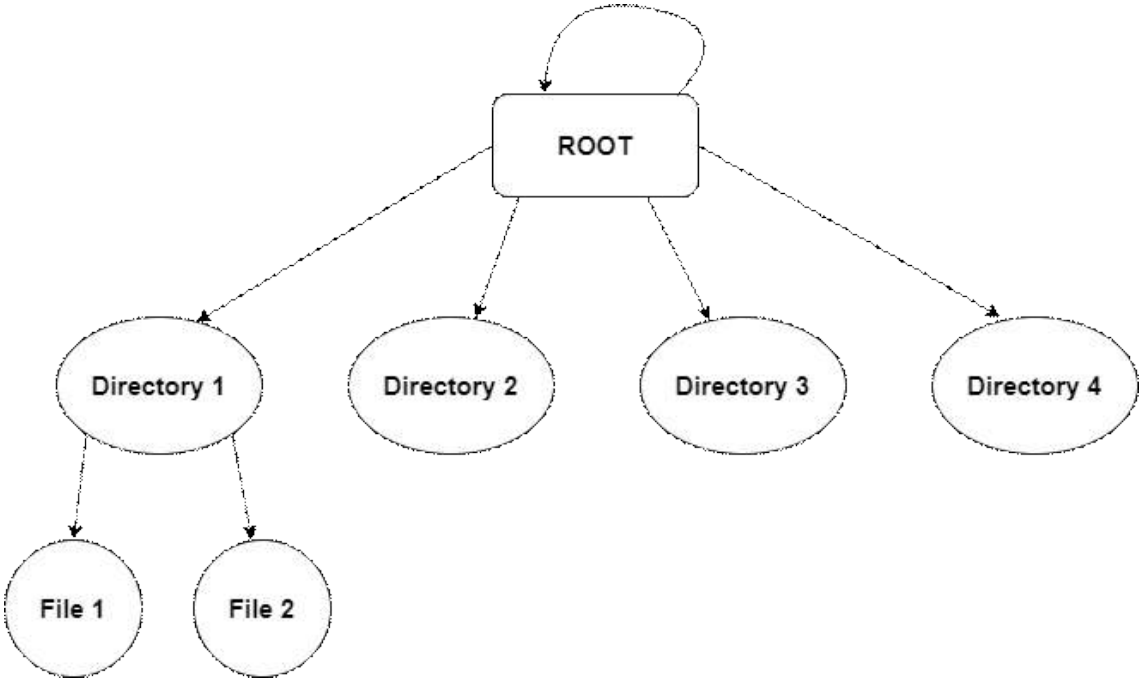
3. 프로그램의 동작 과정과 구현 방법

```
struct dentry* curr_dir;
```

시스템 내에서 현재 참조하고 있는 디렉토리를 알기 위해 curr_dir을 선언했습니다.

```
curr_dir = &root;
```

root 디렉토리로 초기화하여 초기 시스템을 세팅하였습니다.
이후 트리 형식으로 구조체 내에 있는 포인터들로 상위 디렉토리, 하위 디렉토리와 디렉토리 내에 존재하는 파일들에 접근할 수 있도록 하였습니다.



[ls]

(Error-handling)

- ls 명령어의 인자값이 존재할 경우

해당 디렉토리의 d_num과 i_num을 이용하여 모든 하위 항목들을 참조하여 출력합니다.
하위 디렉토리의 경우, 삭제시 포인터들의 배열의 인덱스를 한 칸씩 조정하였기 때문에 순서대로 출력하면 되었습니다.
파일의 경우, 생성된 순서가 아닌 ID의 순서대로 출력을 해야하기 때문에 전역변수인 inode_table에서 유효한 ID들에 대하여 순차적으로 접근하여 해당 디렉토리의 파일들의 ID와 일치하는 것들부터 출력하였습니다.

[cd]

(Error-handling)

- cd 명령어의 인자값으로 0개이거나 2개 이상이 주었을 경우
- 현재 디렉토리에서 참조할수 없는 경로로의 접근이 주어졌을 경우

```
root.parent = &root;
```

root 디렉토리의 경우, 상위 디렉토리를 자기 자신으로 설정함으로써 상위 디렉토리로 이동하려는 시도를 할 시, 자기 자신을 리턴하게끔 하였습니다.

```
vector<string> directories;
```

cd 명령어의 인자값을 '/'로 나누어 directories vector에 추가하였습니다. 이후 인자가 '/', '.', '..' 인지를 분류하여 처리하였습니다. 그 외의 경우는 알맞은 하위 디렉토리를 current_path에 추가하고 curr_dir의 값을 해당 디렉토리로 바꾸었습니다.

[mkdir]

(Error-handling)

- mkdir 명령어의 인자값이 없는 경우
- 생성하려는 디렉토리 이름과 같은 디렉토리가 현재 디렉토리에 존재할 경우

하위 디렉토리 항목들 중 같은 이름을 가진 디렉토리가 존재하는 지 여부를 판단합니다. 존재하지 않는다면, 새로운 dentry를 만들어 parent 항목을 현재 디렉토리와 일치시켜 상하 관계를 유지할 수 있도록 합니다.

[rmdir]

(Error-handling)

- mkdir 명령어의 인자값이 없는 경우
- 생성하려는 디렉토리 이름과 같은 디렉토리가 현재 디렉토리에 없는 경우

mkdir과 마찬가지로 제거하려는 디렉토리가 현재 디렉토리에 존재하는 지 여부를 검사하고 명령어를 실행합니다.

```
void RM(dentry* dentry) {
    // remove files
    for (int i=0; i<dentry->i_num; i++) {
        inode_table[dentry->d_inode[i]->id] = 0;
        int b_size = find_block_size(dentry->d_inode[i]->size);
        block_size += b_size;
        dentry->d_inode[i] = NULL;
    }
    dentry->i_num = 0;
    // remove directories
    for (int i=0; i<dentry->d_num; i++) {
        RM(dentry->d_dentry[i]);
        dentry->d_dentry[i] = NULL;
    }
    dentry->d_num = 0;
}
```

해당 디렉토리 내의 파일들을 지우고 하위 디렉토리를 지우는 재귀함수를 구현하였습니다. 디렉토리들을 삭제한 후, d_num값을 0으로 바꾸어 줍니다.

[mkfile]

(Error-handling)

- mkdir 명령어의 인자값이 없는 경우
- 생성하려는 파일의 크기 값으로 숫자를 입력하지 않은 경우

- 생성하려는 파일의 크기 값으로 음수를 입력한 경우
- 생성하려는 파일이 현재 디렉토리에 같은 이름으로 존재하는 경우
- 생성하려는 파일의 크기가 현재 남은 메모리의 크기보다 큰 경우

```
int find_block_size(unsigned int size) {
    // calculate the number of blocks needed
    int direct_b = 0, single_id = 0, double_id = 0;
    int num = (size / 1024);
    if (size%1024 != 0) num++;
    if (num < 13) {
        direct_b = num;
    }
    else {
        direct_b = 12;
        single_id = num-12;
        single_id++;
        if (single_id > 257) {
            double_id = single_id - 257;
            single_id = 257;
            double_id+=2;
            if (double_id > 258) double_id++;
        }
    }
    db = direct_b; si = single_id; di = double_id;
    return (direct_b + single_id + double_id);
}
```

위 함수를 통해 생성하려는 파일의 크기를 inode 내의 direct block과 indirect block들에게 잘 분배합니다.

[rmfile]

(Error-handling)

- mkdir 명령어의 인자값이 없는 경우
- 제거하려는 파일 이름과 같은 파일이 현재 디렉토리에 없는 경우

제거하고자 하는 파일의 유무를 확인하고, 해당 파일의 inode를 삭제합니다. 그 과정에서 inode내의 블록 총 개수를 반환하고 디렉토리의 inode 포인터 배열의 index를 한 칸씩 조정합니다.

[inode]

(Error-handling)

- mkdir 명령어의 인자값이 없는 경우
- 제거하려는 파일 이름과 같은 파일이 현재 디렉토리에 없는 경우

해당 파일이 현재 디렉토리에 존재하는지 확인 후, inode 구조체에 저장되어 있는 정보들을 하나씩 출력합니다.

[exit]

```
cout<<ID<<":"<<current_path<<"$ ";
getline(cin, input);
if (input == "exit") break;
```

input을 입력받은 직후 “exit”과 일치할 경우 while 문을 탈출하여 즉시 프로그램을 종료합니다.

4. 결과

```
PS D:\a4> .\a.exe
2015147531:/$ mkdir a b c d
2015147531:/$ ls
a b c d
2015147531:/$ cd a
2015147531:/a$ mkfile f1 32768
Now you have ...
940 / 973 (blocks)
2015147531:/a$ mkfile f2 65536
Now you have ...
875 / 973 (blocks)
2015147531:/a$ ls
f1 f2
2015147531:/a$ mkdir d1 d2
2015147531:/a$ ls
d1 d2 f1 f2
```

```
2015147531:/a$ inode f2
ID: 1
Name: f2
Size: 65536 (bytes)
Direct blocks: 12
Single indirect blocks: 53
Double indirect blocks: 0
2015147531:/a$ rmfile f2
Now you have ...
940 / 973 (blocks)
2015147531:/a$ cd ..
2015147531:/$ ls
a b c d
2015147531:/$ rmdir a
Now you have ...
973 / 973 (blocks)
2015147531:/$ ls
b c d
2015147531:/$ exit
PS D:\a4>
```

왼쪽에서부터 오른쪽으로 가는 결과입니다.

ls, mkdir, cd, mkfile, inode, rmfile, rmdir, exit의 명령어들을 실행하여 얻은 결과값입니다.

5. 과제 수행 시 겪었던 어려움과 해결 방법

✓ rmdir 명령어를 구현하는 방법 중 하나인 재귀함수를 구현하는 데 어려움이 있었습니다.

✓ inode에서 파일의 크기만큼 블록을 할당하는 것을 메모리까지 구현해야 하는 문제로 착각하였습니다.

[해결] : 블록의 개수가 문제의 핵심인 것을 깨닫고 inode 내의 원소들을 int 타입으로 변경하였습니다.

✓ cd 명령어를 구현하면서 실행창에 띄워지는 current_path의 '/'유무를 처리하기 어려웠습니다.

✓ 파일을 삭제 후 해당 inode id를 가지는 새로운 파일을 생성할 수 있는 부분을 구현하는 과정

6. Reference

✓ Operating System Class Lecture 10 「File System」