OS Assignment4 결과 보고서

컴퓨터과학과
2015147533 유현석

1. 작성한 프로그램의 동작 과정과 구현 방법

1-1 구조체 선언(INODE , DENTRY , SUPER_BLOCK)

프로그램의 실질적인 main 함수를 들어가기 전에 사용될 3개의 구조체를 선언해주었습니다. INODE, DENTRY, SUPER_BLOCK으로 각각 이루어져 있고, 각각의 역할은 다음과 같습니다.

1. INODE -> 파일과 관련된 정보를 저장
2. DENTRY -> 각각의 폴더(dentry)와 관련된 정보를 저장
3. SUPER_BLOCK -> 처음이자 최초로 생성되는 폴더(dentry)를 나타내고 있습니다.

<구조체 선언>

```
struct INODE{
    int id;                 // inode ID(0~127)
    char name[64];          // inode name (file name)
    int size;               // file size


    int direct_block;
    int single_indirect;
    int double_indirect;

    int totalblock;
};

struct DENTRY{
    char name[64];      //dentry name(directory name)
    char pathname[100];

    DENTRY* parent;         //parent dentry
    DENTRY* d_dentry;       //a list of children dentries
    INODE* d_inode;         //a list of inodes

    int dentrynum;          //size of have dentry
    int inodenum;           //size of have inode
};


struct SUPER_BLOCK{
    struct DENTRY s_root;       //parent dentry
};
```

각각의 구조체에 포함된 변수의 의미는 옆에 나타나져 있는 주석을 참고하시면 될 거 같습니다. 이 중, 핵심 구조체는 DENTRY로 하나의 DENTRY는 포인터로 새로운 DENTRY와 INODE 배열의 정보를 포함하고 있습니다. 이것으로 하나의 DENTRY 안에 포함되어 있는 또다른 DENTRY들의 정보와 file들의 정보를 알 수 있습니다.

1-2 함수 선언(deletedir)

main 함수가 시작하기 전에 사용한 또 다른 것은 이번 과제에서 만들어서 사용한 유일한 함수 deletedir(DENTRY A)입니다. 이 함수의 역할은 삭제하고자 하는 DENTRY를 지칭해주면 그 DENTRY안에 있는 모든 INODE의 크기 더하여 삭제 후의 전체 block의 크기를 알 수 있게 합니다.

<deletedir 함수 선언>

```
void deletedir(DENTRY delcurrent){


    for(int m=0;m<delcurrent.inodenum;m++){
        leftblock = leftblock + delcurrent.d_inode[m].totalblock;
        checkpid[delcurrent.d_inode[m].id] = 0;
    }

    for(int n=0;n<delcurrent.dentrynum;n++){
        deletedir(delcurrent.d_dentry[n]);
    }

}
```

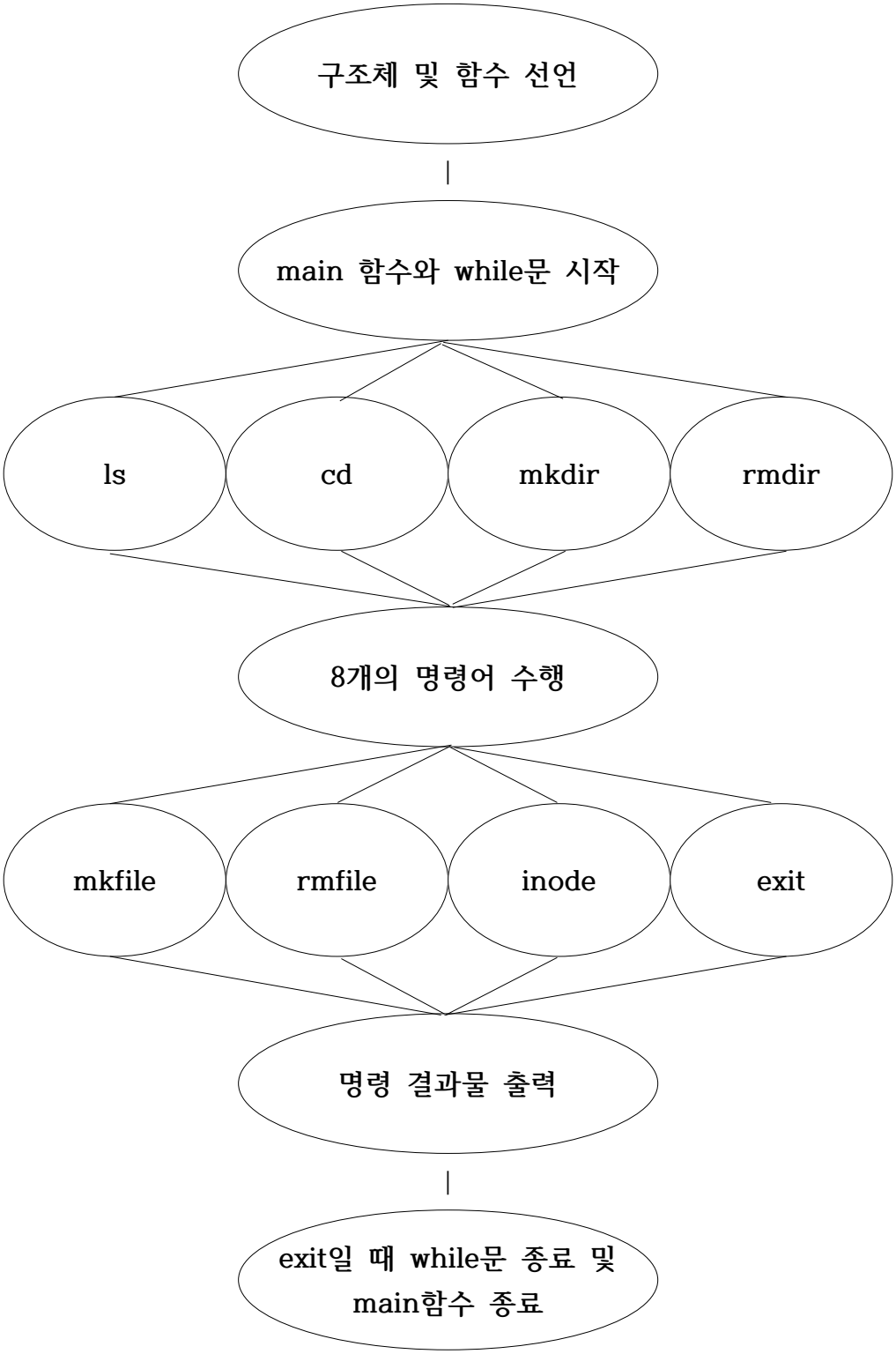
이 함수의 구조는 위와 같습니다. 안에 있는 모든 DENTRY 내의 INODE를 구하기 위해 재귀함수의 구조를 사용하였습니다.

1-3 메인 함수 선언

메인 함수에선 후에 사용될 while문 전에 몇가지 필요한 정보들을 초기화 및 동적할당 시켜주었습니다. 앞선 구조체에서 선언한 d_dentry와 d_inode 포인터를 각각 64개, 128개의 크기로 설정해주었고 parent 즉 전 단계의 dentry를 저장하는 포인터를 1의 크기로 설정해주었습니다. 그 외에 최초의 dentry의 정보와 현재의 dentry를 나타내는 dentry를 만들어주었습니다.

<main 함수 시작>

```
int main(){

    char sinput[100];               //reading the string
    SUPER_BLOCK firstp;         //frist root firstparent = firstp

    DENTRY starting;            // firstp = starting dentry


//initalize the all content
    string temp="/";
    strcpy(starting.pathname, temp.c_str());
    strcpy(starting.name, temp.c_str());
    starting.parent = new DENTRY[1];
    starting.d_dentry = new DENTRY[64];
    starting.d_inode = new INODE[128];
    starting.dentrynum=0;
    starting.inodenum=0;
    starting.parent[0] = starting;

    firstp.s_root = starting; //we can know the start

    DENTRY current;
    current = starting;         // the directory where we are now

    for(int b=0;b<128;b++){
        checkpid[b]=0;
    }                           //checking the pid is use or not
```



전체 알고리즘의 대략적인 형태의 순서도는 위와 같습니다. 이제부터 아래에서는 각각의 8개의 명령어에 대한 수행함수의 알고리즘 및 방식을 설명하도록 하겠습니다.

1-4 메인 함수 내 while문 선언

while문 내에서는 명령어를 입력받고 그 입력 받은 명령어를 빈칸으로 쪼개서 해당하는 명령어 부분을 실행합니다.

<main 함수 시작>

```
while(1){
    cout<<"2015147533:"<<current.pathname<<"$ ";    //print order
    cin.getline(sinput,100);                        //read line

    char *ptr = strtok(sinput," ");                 //split with " "
/================================================================================
```

1-5 while문 내 해당 명령어 수행

① 명령어 ls

현재 위치의 dentry에서 내부에 존재하는 dentry 개수와 inode의 개수만큼의 해당 이름을 출력해줍니다. 결과적으로 내부에 존재하는 모든 파일과 폴더를 출력합니다.

```
else if(strcmp(sinput,"ls")==0){
    for(int i=0;i<current.dentrynum;i++){
        cout<<current.d_dentry[i].name<<" ";
    }
    for(int i=0;i<current.inodenum;i++){
        cout<<current.d_inode[i].name<<" ";
    }
    cout<<endl;

}
--------------------------------- org = ls-------
```

② 명령어 cd

cd 명령어를 수행하기 위해서는 먼저 cd 명령어를 크게 4개로 나누어 줬습니다.

1. cd ..

-> 상위의 DENTRY로 이동하는 명령어 (parent[0]로 이동)

2. cd /

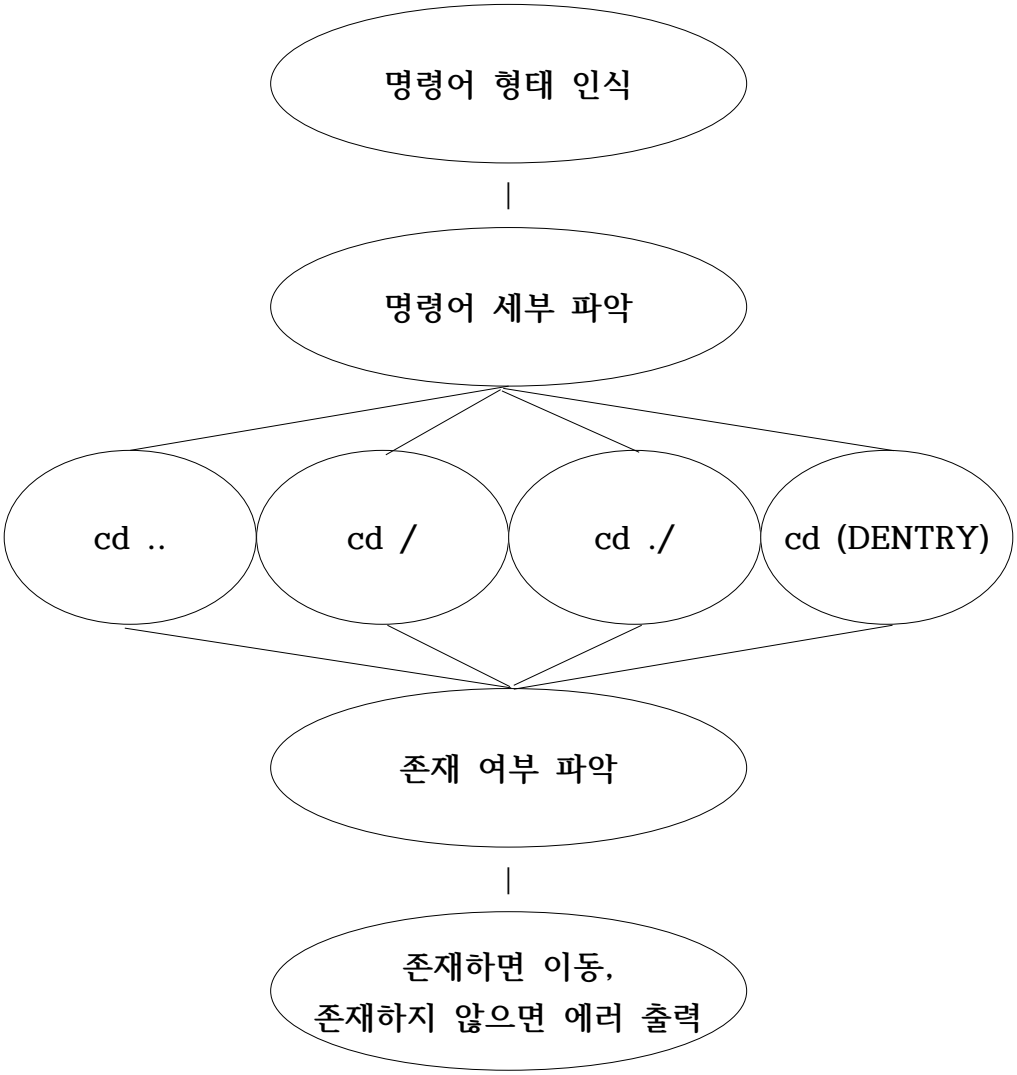
-> root 위치를 기준으로 특정 위치를 지칭하는 명령어(super_block을 이용)

3. cd ./

-> 현재의 위치를 기준으로 특정 위치를 지칭하는 명령어(current을 기준으로 이용)

4. cd (DENTRY)

-> 현재의 내부에 있는 DENTRY로 이동



명령어의 세부 파악을 위하여 먼저 빈칸을 기준으로 나누어준 후 (cd ..) 명령어 인지 판단하고 아닐시 '/'을 기준으로 다시 나누어줬습니다. 그 이유로는 모든 DENTRY는 /을 기준으로 구분되기 때문입니다. 그 후 경로를 따라가면서 모든 경로가 맞을 때만 이동을 하고 아닐 시에는 에러를 출력하였습니다.

```
if(boolfind==1){
    if(strcmp(current.parent[0].pathname,current.pathname)==0){
        current = tempcurrent;
    }
    else{
        for(int v=0; v<current.parent[0].dentrynum;v++){
            if(strcmp(current.parent[0].d_dentry[v].name,current.name)==0){
                current.parent[0].d_dentry[v]=current;
                current = tempcurrent;
                break;
            }
        }
    }
}
```

위의 코드는 모든 경로를 확인한 후 경로가 맞을 때 실행되는 코드입니다. 현재의 위치가 root인 여부를 파악 후 현재 위치를 해당 경로로 바꾸어줍니다.

③ 명령어 mkdir

새로 만들어질 DENTRY의 정보를 newdentry에 저장한 후, 그 newdentry를 현재 위치의 내부 DENTRY 포인터에 저장해줍니다. 해당 파일의 경로 역시 '/'를 더하여 만들어줍니다. 새로 DENTRY가 만들어질 때마다 기존에 만들어진 DENTRY의 parent dentry의 정보를 업데이트 시켜줍니다.

```
else if(strcmp(sinput,"mkdir")==0){
    ptr=strtok(NULL," ");
    if(ptr == NULL){
        cout<<"mkdir NULL ERROR"<<endl;
    }
    else{
        DENTRY newdentry;
        newdentry.dentrynum=0;
        newdentry.inodenum=0;
        newdentry.parent = new DENTRY[1];
        newdentry.d_dentry = new DENTRY[64];
        newdentry.d_inode = new INODE[128];
        newdentry.parent[0]=current;                        //make new dentry and put the new info
        string tem2(current.pathname);
        string tem = ptr;
        strcpy(newdentry.name,tem.c_str());                 // update the name
        if(tem2=="/"){
            tem = tem2 + tem;
            strcpy(newdentry.pathname,tem.c_str());             //update the new path with adding /
        }
        else{
            tem = tem2 + "/" +tem;
            strcpy(newdentry.pathname,tem.c_str());             //update the new path with adding /
        }
        current.d_dentry[current.dentrynum] = newdentry;
        current.dentrynum++;
        for(int v=0;v<current.dentrynum;v++){
            current.d_dentry[v].parent[0].dentrynum=current.dentrynum;
        }
    }
}
--------------------------------- org = mkdir-------------------------
```

④ 명령어 rmdir

삭제하고자 하는 이름을 가진 내부 DENTRY를 찾은 후 배열을 하나씩 앞으로 이동시켜줍니다. 이 때 삭제될 DENTRY 내부에 있는 모든 INODE를 확인하기 위해서 앞서 만들었던 deletedir 함수를 이용합니다.

```
else if(strcmp(sinput,"rmdir")==0){
    ptr=strtok(NULL," ");
    if(ptr == NULL){
        cout<<"rmdif NULL ERROR"<<endl;
    }
    else{
        string tem1 =ptr;
        for(int v=0;v<current.dentrynum;v++){
            if(strcmp(current.d_dentry[v].name,tem1.c_str())==0){
                DENTRY delcurrent;
                delcurrent = current.d_dentry[v];
                deletedir(delcurrent);

                for(int z=v;z<current.dentrynum-1;z++){
                    current.d_dentry[z]=current.d_dentry[z+1];
                }
                break;
            }
        }
        current.dentrynum--;
        for(int v=0;v<current.dentrynum;v++){
            current.d_dentry[v].parent[0].dentrynum=current.dentrynum;
        }
        cout<<"Now you have ..."<<endl;
        cout<<leftblock<<" / 973 (blocks)"<<endl;
    }
}
--------------------------------- org = rmdir-------------------------
```

⑤ 명령어 mkfile

파일을 만들어줘 그 정보를 inode에 저장을 해줍니다. 이때 파일의 크기에 해당하여 차지하는 block의 개수 역시 계산 해주어야 합니다. 아래의 사진은 이를 계산하는 알고리즘입니다. 전체 크기를 1024로 나누어준 것이 총 들어가야 하는 block의 숫자입니다. 이를 토대로 3가지 범위로 나누어줬습니다.

```
int total = stoi(tem1);
newinode.size = total;                              //update the size

int tblock = total/1024;

if(tblock<13){
    newinode.direct_block=tblock;
    newinode.single_indirect=0;
    newinode.double_indirect=0;
    newinode.totalblock =tblock;
    leftblock=leftblock - newinode.totalblock;
}
else if(tblock<269){
    newinode.direct_block=12;
    newinode.single_indirect=(tblock-12)+1;
    newinode.double_indirect=0;
    newinode.totalblock = (tblock+1);
    leftblock =leftblock - newinode.totalblock;
}
else{
    newinode.direct_block=12;
    newinode.single_indirect=256+1;
    newinode.double_indirect=(tblock-268);
    int S = (newinode.double_indirect/256)+1;
    int R = newinode.double_indirect%256;
    newinode.double_indirect=(1+S+((S-1)*256)+R);
    newinode.totalblock = (270+S+((S-1)*256)+R);
    leftblock =leftblock - newinode.totalblock;
}

                                                    //update the block;
for(int c=0;c<128;c++){
    if(checkpid[c]==0){
        checkpid[c]=1;
        newinode.id = c;
        break;
    }
}                                       //check unuse pidnum and update the pid;
current.d_inode[current.inodenum] = newinode;
current.inodenum++;

cout<<"Now you have ..."<<endl;
cout<<leftblock<<" / 973 (blocks)"<<endl;
}
```

1. 총 block이 13개보다 작은 경우

-> direct_block만을 사용합니다.

전체사이즈 : N

차지하는 블록의 숫자 = N

2. 총 block이 269개보다 작은 경우

-> direct_block과 single_indirect block만을 사용합니다.

전체사이즈 : N

차지하는 블록의 숫자 = N + 1

3. 총 block이 269개보다 큰 경우

-> double_indirect block까지 사용합니다.

전체사이즈 : N

계산식 : S = ((전체 사이즈 N - 268) / 256) + 1

R = ((전체 사이즈 N - 268) % 256)

차지하는 블록의 숫자 = 270 + S + (S-1) *256 + R

⑥ 명령어 rmfile

삭제하고자 하는 이름을 가진 내부 INODE를 찾은 후 배열을 하나씩 앞으로 이동시켜줍니다. 이 때 삭제될 INODE가 차지하는 block 수를 다시 원상 복귀 시켜줍니다.

```
    else if(strcmp(sinput,"rmfile")==0){
        cout<<sinput<<endl;
        ptr=strtok(NULL," ");
        if(ptr == NULL){
            cout<<"rmfile NULL ERROR"<<endl;
        }
        else{
            string tem1 =ptr;
            for(int v=0;v<current.inodenum;v++){
                if(strcmp(current.d_inode[v].name,tem1.c_str())==0){
                    leftblock = leftblock + current.d_inode[v].totalblock
                    for(int z=v;z<current.inodenum-1;z++){
                        current.d_inode[z]=current.d_inode[z+1];
                    }
                    break;
                }
            }
            current.inodenum--;
            cout<<"Now you have ..."<<endl;
            cout<<leftblock<<" / 973 (blocks)"<<endl;
        }
    }
---------------------------------- org = rmfile------------------------
```

⑦ 명령어 inode

해당 이름을 가진 inode의 정보를 모두 출력해줍니다.

```
else if(strcmp(sinput,"inode")==0){
    ptr=strtok(NULL," ");
    if(ptr == NULL){
        cout<<"inode NULL ERROR1"<<endl;
    }
    else{
        string tem1 =ptr;
        for(int v=0;v<current.inodenum;v++){
            if(strcmp(current.d_inode[v].name,tem1.c_str())==0){
                cout<<"ID: "<<current.d_inode[v].id<<endl;
                cout<<"Name: "<<current.d_inode[v].name<<endl;
                cout<<"Size: "<<current.d_inode[v].size<<" (bytes)"<<endl;
                cout<<"Direct blocks: "<<current.d_inode[v].direct_block<<endl;
                cout<<"Single indirect blocks: "<<current.d_inode[v].single_indirect<<endl;
                cout<<"Double indirect blocks: "<<current.d_inode[v].double_indirect<<endl;
                break;
            }
        }
    }
}
-------------------------------- org = inode-------------------------
```

⑧ 명령어 exit

반복되었던 while문을 끝내줍니다.

```
================================================
 if(strcmp(sinput,"exit")==0){
     break;
 }
---------------------------------- org = exit---
```

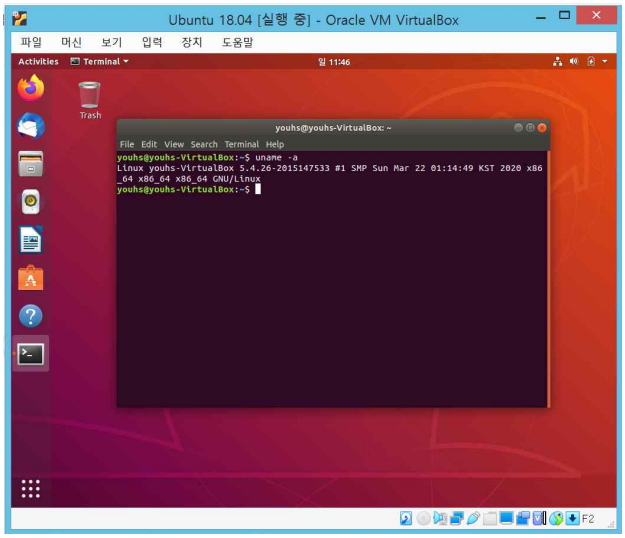
⑨ 명령어 그 외

위에 나열된 명령어 외의 값이 들어오면 잘못된 input을 넣었다는 에러 메시지를 출력해줍니다.

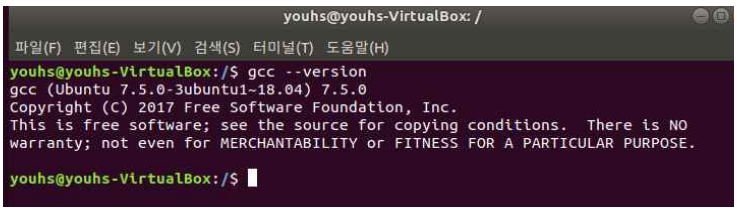
```
        else{
            cout<<"Wrong Input"<<endl;

        }
//------------------------------------ org = error-
    }
```

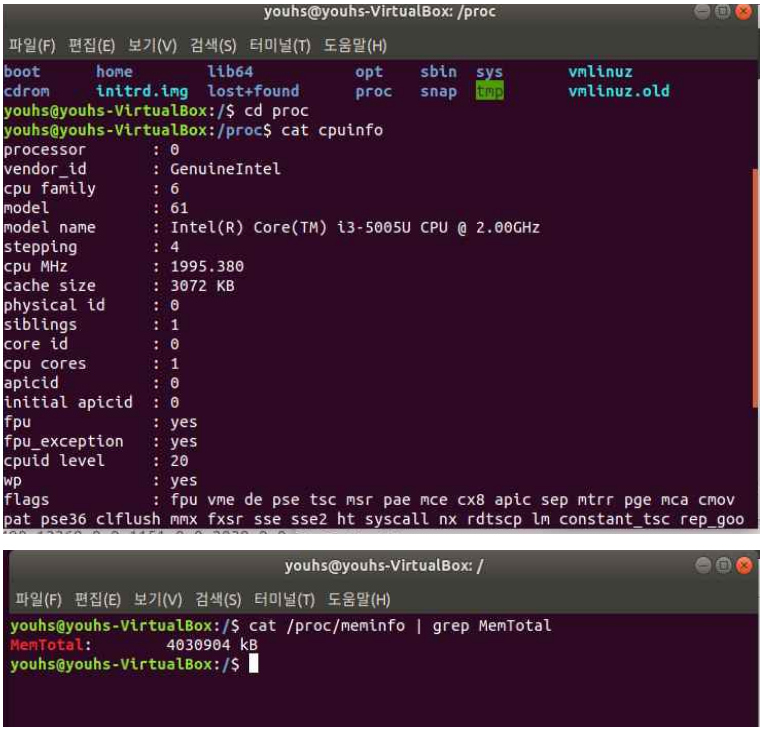
2. 개발 환경 명시

2-1 uname -a 실행 결과

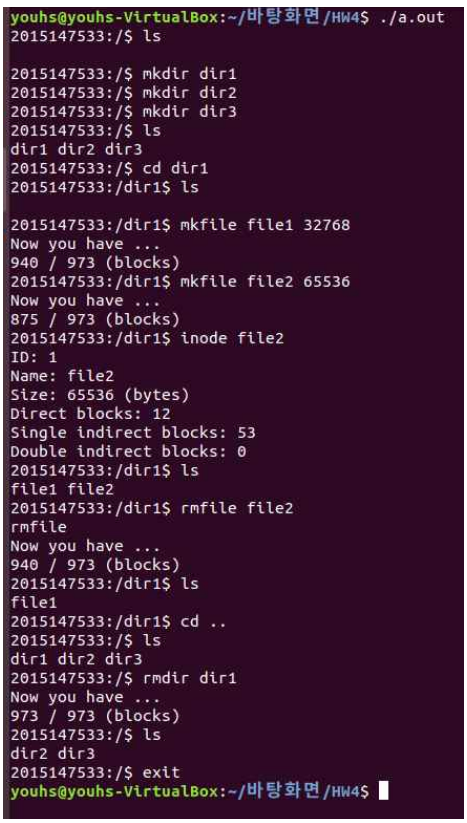


2-2 사용한 컴파일 버전, CPU, 메모리정보




youhs@youhs-VirtualBox: /proc
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
boot home lib64 opt sbin sys vmlinuz
cdrom initrd.img lost+found proc snap tmp vmlinuz.old
youhs@youhs-VirtualBox:/$ cd proc
youhs@youhs-VirtualBox:/proc$ cat cpuinfo
processor : 0
vendor_id : GenuineIntel
cpu family : 6
model : 61
model name : Intel(R) Core(TM) i3-5005U CPU @ 2.00GHz
stepping : 4
cpu MHz : 1995.380
cache size : 3072 KB
physical id : 0
siblings : 1
core id : 0
cpu cores : 1
apicid : 0
initial apicid : 0
fpu : yes
fpu_exception : yes
cpuid level : 20
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ht syscall nx rdtscp lm constant_tsc rep_goo
youhs@youhs-VirtualBox: /
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
youhs@youhs-VirtualBox:/$ cat /proc/meminfo | grep MemTotal
MemTotal: 4030904 kB
youhs@youhs-VirtualBox:/$

3. 결과 화면


youhs@youhs-VirtualBox:~/바탕화면/HW4$ ./a.out
2015147533:/$ ls
2015147533:/$ mkdir dir1
2015147533:/$ mkdir dir2
2015147533:/$ mkdir dir3
2015147533:/$ ls
dir1 dir2 dir3
2015147533:/$ cd dir1
2015147533:/dir1$ ls
2015147533:/dir1$ mkfile file1 32768
Now you have ...
940 / 973 (blocks)
2015147533:/dir1$ mkfile file2 65536
Now you have ...
875 / 973 (blocks)
2015147533:/dir1$ inode file2
ID: 1
Name: file2
Size: 65536 (bytes)
Direct blocks: 12
Single indirect blocks: 53
Double indirect blocks: 0
2015147533:/dir1$ ls
file1 file2
2015147533:/dir1$ rmfile file2
rmfile
Now you have ...
940 / 973 (blocks)
2015147533:/dir1$ ls
file1
2015147533:/dir1$ cd ..
2015147533:/$ ls
dir1 dir2 dir3
2015147533:/$ rmdir dir1
Now you have ...
973 / 973 (blocks)
2015147533:/$ ls
dir2 dir3
2015147533:/$ exit
youhs@youhs-VirtualBox:~/바탕화면/HW4$

4. 과제 수행 시 겪었던 어려움과 해결 방법

1. DENTRY가 변화된 후 이동하는 과정에서 저장을 해주지 않는 경우

-> cd 명령어로 이동해야 하는 경우가 생길 때 현재의 위치인 current를 이동해야 하는 DENTRY로 바꿔 줘야 합니다. 이때 지금 current dentry가 가지고 있는 정보를 업데이트시켜야만 정보의 손실이 일어나지 않습니다.

이를 해결하기 위해서 current의 parent dentry의 내부 dentry중 current에 해당하는 dentry의 정보를 current로 저장해주는 과정을 진행하여 해결하였습니다.

2. 새로운 내부 DENTRY를 만들고 그전에 있던 내부 DENTRY의 parent 정보를 업데이트 하지 않은 경우

-> 하나의 dentry에서 내부의 새로운 dentry를 mkdir로 만드는 경우가 존재합니다. 이때 새로 만들어진 dentry를 포함하는 정보로 기존에 내부에 존재하던 dentry의 parent dentry를 업데이트해줘야 합니다. 그렇지 않으면 기존에 존재하는 dentry의 위치에서 parent dentry를 호출하게 되면 손실된 정보를 가진 parent dentry가 호출됩니다.

이를 해결하기 위해서 mkdir 명령어가 실행될 때마다 기존의 내부에 존재하던 모든 dentry의 parent dentry의 정보를 현재의 current dentry의 정보로 업데이트 시켜 주었습니다.

3. ROOT의 DENTRY를 업데이트 안 한 경우

-> root dentry를 main에서 선언할 때만 정보를 입력하고 그 후 업데이트를 해주지 않으면 cd / 명령어 즉, root의 위치에서 경로를 따라갈 때 정보의 손실로 인해 올바른 위치로 이동할 수 없게 됩니다.

이를 해결하기 위해서 cd / 명령어가 호출되면 기존의 current 위치의 dentry부터 root dentry까지의 정보를 모두 업데이트시켜 저장해줍니다. root 위치에서 이동이 일어날 때 역시 root dentry의 정보를 업데이트시켜 저장합니다.