**Pintos Project 5: Filesystem**

담당 교수 : 박성용 교수님

학번 / 이름 : 20201635 전찬

개발 기간 : 2022.12.08 – 2022.12.19

1. **개발 목표**

이번 pintos 프로젝트 5는 실제 상용화된 file system과 비슷한 형태를 구현하는 목표를 갖는다.

첫 번째로, 기존의 pintos file은 disk 상의 공간에서 linear하게 저장되어 external fragmentation 문제를 갖는다. 따라서 file이 disk 상에서 block 단위로 나누어 저장될 수 있도록 바꾸어야 한다.

두 번째로, 현재 pintos는 file growth을 수행할 수 없다. 이를 수행할 수 있도록 구현해야 한다.

세 번째로, 현재 pintos는 subdirectory 없이 root directory에 모든 file을 저장하는 형태이다. 이를 변경하여 실제 윈도우, 혹은 맥 OS와 같이 directory 내의 directory 구조를 가능하도록 구현해야 한다.

추가로 기존의 비효율적인 disk의 입출력 구조에서 벗어나, buffer cache을 사용하는 구조 또한 구현해야 한다.

1. **개발 범위 및 내용**
   1. **개발 범위**
2. Extensible file & file growth

위 구조를 구현하여 파일의 크기가 고정되지 않으며, external fragmentation을 줄이는 방법으로 하나의 file을 block 단위로 나누어 저장하여 disk 관리를 수행할 수 있다. 또한 해당 file을 doubly linked list 형태로 구현해 큰 file size도 저장할 수 있도록 구현해야 한다.

1. Subdirectory

subdirectory을 통해 directory 내의 directory을 가능하게 하며, 대부분의 OS에서 사용하는 계층적인 구조의 디렉토리를 구현해야 한다. 또한 해당 subdirectory 내에서도 파일을 생성할 수 있도록 구현해야 한다.

* 1. **개발 내용**
* Extensible file & file growth
  + Index structure와 management에 대해서 기술

해당 inode\_disk 는 disk 상의 inode 구조체이다. 따라서 해당 inode\_disk 구조체를 disk에서 읽어올 때, disk read / write size인, 512 Byte 씩 읽어오게 된다. 따라서 inode\_disk size를 512 Byte로 맞추어 주어야 하며, 이에 따라 direct = 123 / indirect = 1 / double indirect = 1개의 pointer을 가지게 된다.(이는 inode\_disk structure에 directorty인지 판단하는 변수가 있을 때의 기준이다.) 이를 바탕으로 해당 주소 offset이 0~123 \* 512 사이이면 direct 접근, 123 \* 512 ~ 123 \* 512 + 128 \* 512 사이라면 indirect 접근, 그 이상이라면 double indirect 접근으로 취급하여 fextensible file과 file growth 등을 처리할 수 있다.

* Subdirectory
  + Directory entry 관리 방법

각 실제 linux / window는 계층적 구조라고 할 수 있다. pintos 또한 이 구조를 갖기 위해 각 directory entry가 해당하는 이름의 file의 inode, 그리고 해당하는 inode에 대한 directory을 point할 수 있도록 구현해야 한다. 또한 “.”, “..” 등의 실제 리눅스 상에 존재하는 자기 자신에 대한 참조 등과 함께 절대 경로 / 상대 경로 처리를 수행해 주어야 한다.

* Buffer cache
  + Buffer cache eviction 방식

buffer cache가 실제 file system 보다 상대적으로 작기 때문에, eviction 방식을 수행해야 한다. 여러 가지 eviction 중 쉽게 사용할 수 있는 방식은 pintos 프로젝트 4 에서 구현해야 했던 second chance 방법을 사용할 수 있다.

* + Buffer cache flush 방식

해당 buffer에 write가 수행되었다면, 이 buffer은 disk 상에도 새롭게 저장해 주어야 한다. 이와 달리, 해당 buffer가 write 을 수행하지 않은 buffer라면, 해당 buffer을 buffer cache에서 제거해 주기만 해도 문제가 없는 형태이다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

12.07 – 12.08 : system call prototype 구현

12.09 – 12.12 : buffer cache 구현 시도

12.13 – 12.19 : buffer cache 제외 및 extensible file 형태 구현 시도.

* 1. **제작 내용**

첫 번째로, 필요한 system call에 대해, userprog/syscall.c 에 해당하는 system call 각각의 prototype을 구현해 주었다.

/\* project 5의 추가 구현해야 할 system call \*/

static bool

chdir (char \*path)

{

//return filesys\_change\_dir(path);

return false;

}

static bool

mkdir (const char \*dir)

{

//return filesys\_create\_dir(dir);

return false;

}

static bool

readdir (int fd, char \*name)

{

/\*

bool result = true;

struct file \*file = thread\_current()->fd[fd];

struct inode \*inode = file\_get\_inode(file);

if (inode == NULL)

{

return false;

}

if (!inode\_is\_dir(inode))

{

return false;

}

...

\*/

return false;

}

static bool

isdir (int fd)

{

//return inode\_is\_dir(file\_get\_inode(thread\_current()->fd[fd]));

return false;

}

static int

inumber (int fd)

{

//inode\_get\_inumber(file\_get\_inode(thread\_current()->fd[fd]));

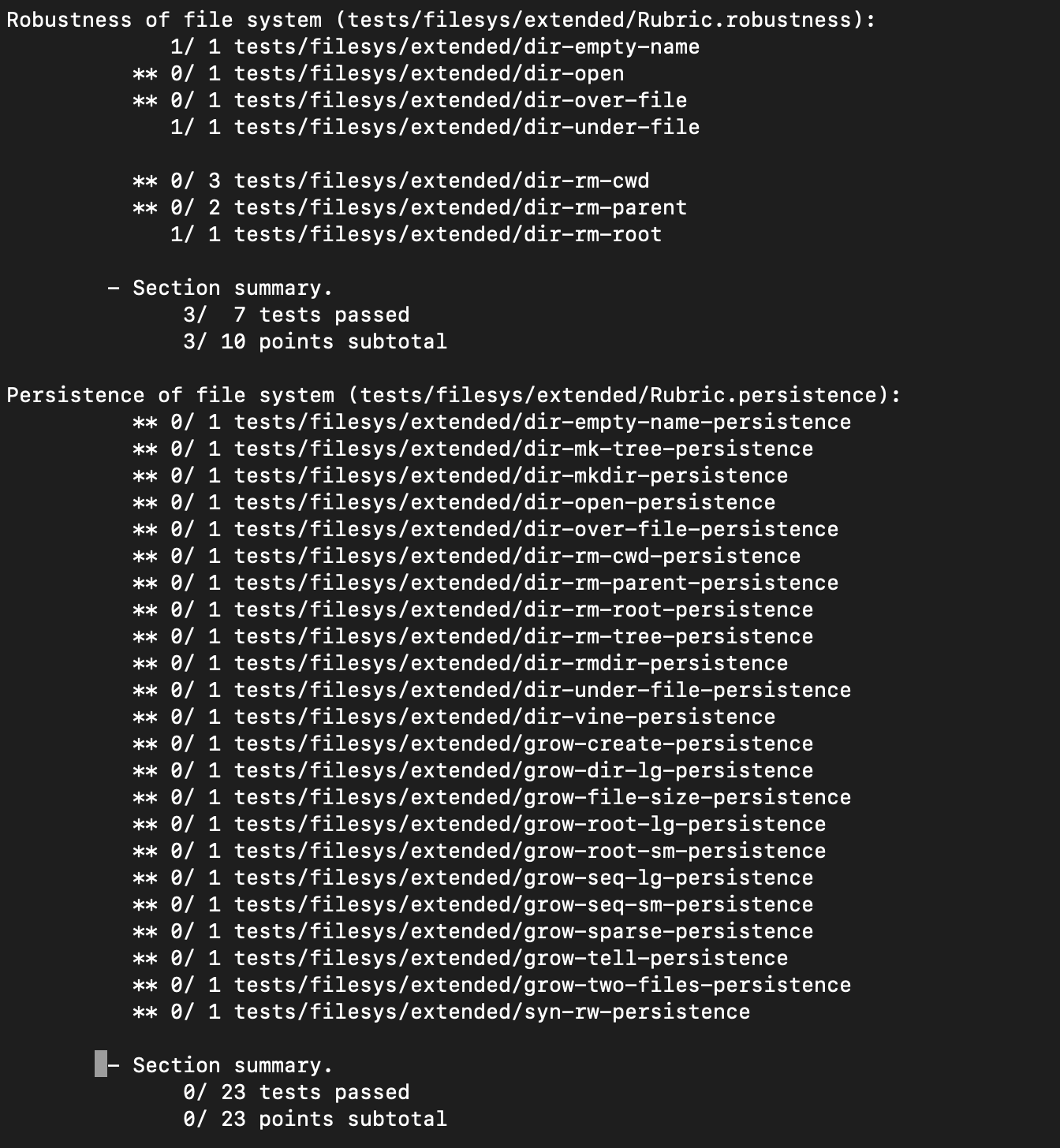
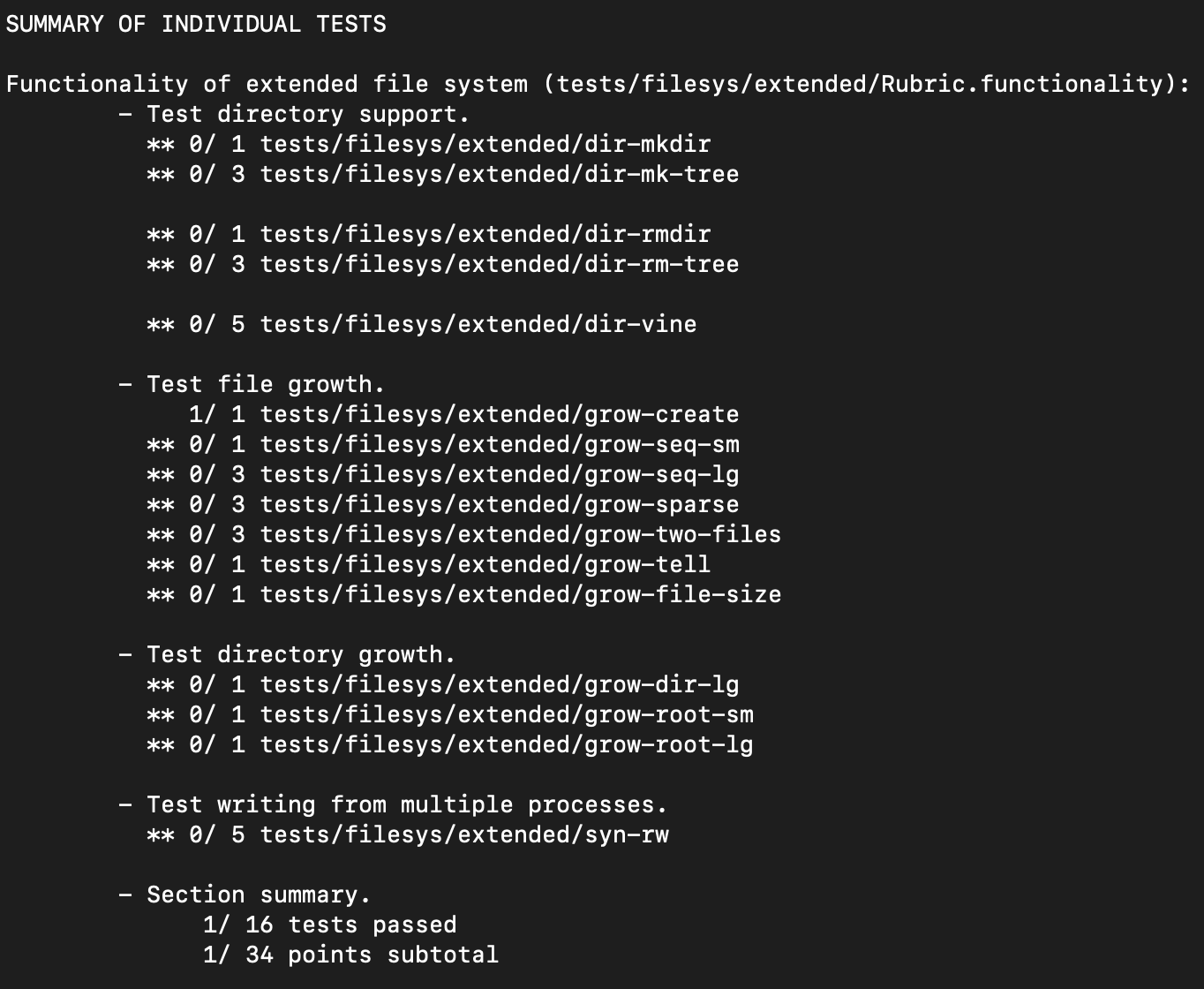
return -1;

}

<userprog/syscall.c에 구현한 prototype>

이를 추가해준 이후에, buffer 구현을 먼저 시도했고, 복잡한 link, error로 인해 buffer 구현, extensible file 구현을 성공할 수 없었다. 개인적으로는 inode와 disk 상의 상호작용해서 문제가 발생한 것 같은데, 이는 방학에 개인적으로 더 해 보아야 알 것 같으며, 마지막 프로젝트를 여기에서 마무리 하였다.

* 1. **시험 및 평가 내용**

****