**Pintos Project 5: Filesystem**

담당 교수 :김 영 재

조 / 조원 :이 호 성

개발 기간 :2021.12.1 ~2021.12.5

1. **개발 목표**

* **해당 프로젝트에서 구현할 내용을 간략히 서술.**

이번 프로젝트에서는 핀토스의 file system을 조금 더 발전시킬 예정이다. 현재는 기본적인 file system의 구조로서 external fragmentation이 심하며 file size 역시 file 이 create 될 때 확정된다. 또한, 현재는 추가적인 디렉토리 없이 root directory에 모든 파일이 존재하며 buffer cache 없이 모든 I/O request를 disk에 직접 read/write를 하고 있는데 이는 매우 비효율적이다. 그러므로 Multi-level indexing을 이용하여 external fragmentation과 file growth에 관한 문제를 해결할 예정이며, subdirectory를 구현하여 root directory 안에 또 다른 directory를 만들어 파일관리를 용이하게 할 예정이다. 또한, buffer cache를 구현하여 효율적으로 I/O request를 수행할 수 있게 만들 예정이다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Extensible file & file growth

현재 핀토스의 filesystem은 기본 contiguous 한 allocation을 사용하여 external fragmentation에 취약하며, file size역시 create했을때의 크기를 따를 수밖에 없다. 이를 multi-level indexing의 방식으로 구현을 하여 효율적인 inode structure의 관리를 통해 external fragmentation을 줄이는 것은 물론 file size 가 자유롭게 조절 가능하도록 구현할 것이다.

1. Subdirectory

현재 핀토스는 오직 root directory만 존재하며 생성하는 모든 파일들은 이 root directory 하위에 일괄적으로 존재하게 된다. 이 root directory의 entry를 file 뿐만 아니라 다른 directory 도 가리킬 수 있게 만들어 추가적인 subdirectory를 사용할 수 있도록 구현할 것이다.

1. Buffer cache

현재 핀토스는 모든 I/O에 대해 read/write를 직접 disk에다 하고 있다. 이는 매우 비 효율적이며 buffer cache를 구현하여 block에 대한 read/write request 가 들어왔을 때 buffer cache를 우선적으로 확인하여 cache 안에 data 가 있다면 disk에 가지 않고 cache에 있는 데이터를 활용하고 없다면 그제서야 disk에 접근하도록 구현할 것이다. 이때, cache에 존재하는 data 들은 eviction 알고리즘을 활용하여 교체할 예정이다.

* 1. **개발 내용**
* **아래 항목의 내용만 서술 (기타 내용은 서술하지 않아도 됨.)**
* Extensible file & file growth
  + Index structure와 management에 대해서 기술

Index structure를 활용하는 indexed allocation은 inode에 direct pointer를 활용하여 이 pointer들이 data를 가리키도록 구현하는 것이다. 하지만 이 방법은 file size가 (inode block의 크기 \* pointer의 개수)를 넘어버리면 파일을 표현할 수 없게 된다. 이번 프로젝트에서 구현할 multi-level indexing은 앞서 말한 indexed allocation의 한계를 극복하도록 inode가 data를 가리키는 것에 더불어 indirect pointer를 사용하는 indirect block이라는 개념을 도입한다. indirect block의 indirect pointer는 data를 포함하는 block을 가리키는 것이 아니라 또 다른 pointer들로 구성된 block을 가리켜 이 pointer 들이 data block들을 가리키게 만드는 것이다. 이를 통해 단순히 direct block 들을 이용하였을 때보다 더 많은 file 들을 가리킬 수 있게 된다.

* Subdirectory
  + Directory entry 관리 방법

원래는 root directory 아래에 모두 존재했던 파일들을 subdirectory를 만들어 하위 디렉토리에 보관할 수 있도록 한다. 그렇게 하려면 우선 기존에 file만 가리키던 directory entry를 또 다른 directory를 가리킬 수 있게 만들어야 한다. 또한, 기존에는 없던 디렉토리를 생성/삭제하는 함수도 작성하여야 하며 이 외 파일만 다루던 함수들 역시 path handling을 통해 적절한 주소에 접근할 수 있도록 작성하여야 한다.

* Buffer cache
  + Buffer cache eviction 방식

pseudo-lru algorithm 인 second chance algorithm을 활용한다. cache 가 모두 사용 중일 때 eviction을 진행해야 하는데 우선 access bit 가 true면 이를 false 로 바꾼 후 다음 캐시로 넘어간다. 만약 access bit가 false면 해당 원소를 victim으로 선정하여 evict 한다.

* + Buffer cache flush 방식

만약 해당 원소가 cache에 있을 때 내용에 변화가 있었다면 dirty bit 가 true 일 것이다. 그렇다는 것은 cache에서 바로 free를 해주면 안되고 반드시 disk에 변경사항을 저장해야 한다. 그렇기 때문에 cache eviction을 할 때나 buffer cache를 close 할 때 반드시 변경사항이 있는지 dirty bit를 확인한 후 있다면 disk에 저장하는 과정을 거쳐야 한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* **II. A.의 개발 범위를 포함하여 구현 내용에 대한 일정 작성**

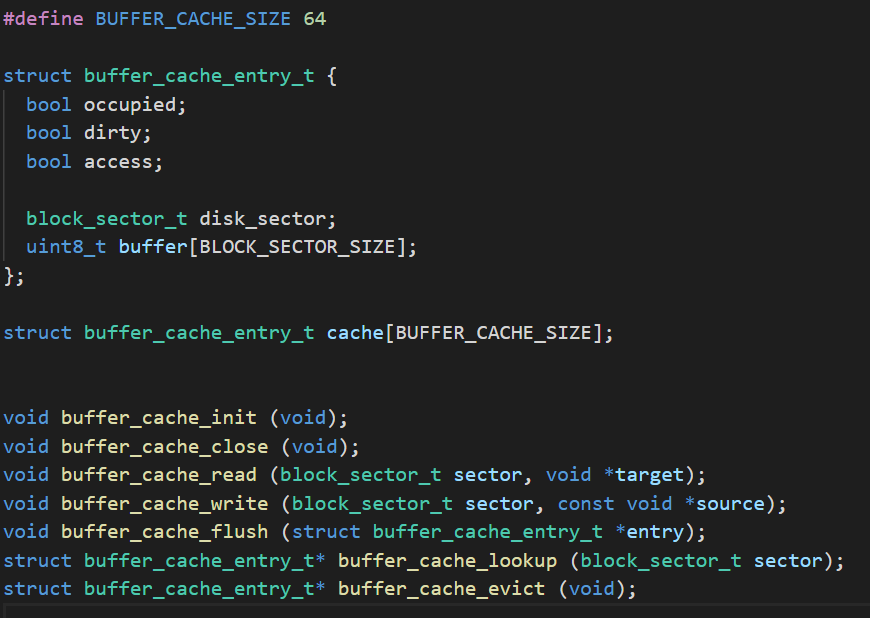
12.1~12.2: buffer cache 구현

12.3~12.4: Extensible file & file growth 구현

12.5: subdirectory 구현

* 1. **개발 방법**
* **II. B.의 개발 내용을 구현하기 위해 어느 소스코드에 어떤 요소를 추가 또는 수정할 것인지 설명. (함수, 구조체 등의 구현이나 수정을 서술)**

**cache.c/cache.h 파일 생성**



**directory.c/directory.h**

[추가함수]

split\_path\_filename(); dir\_open\_path(); dir\_is\_empty();

[변형 함수]

dir\_create(); dir\_get\_inode; lookup(); dir\_lookup(); dir\_add(); dir\_remove;

**inode.c / inode.h**

[추가함수]

index\_to\_sector(); inode\_is\_directory(); inode\_is\_removed(); inode\_allocate();

inode\_reserve\_indirect(); inode\_reserve(); inode\_deallocate\_indirect();

inode\_deallocate();

[변형함수]

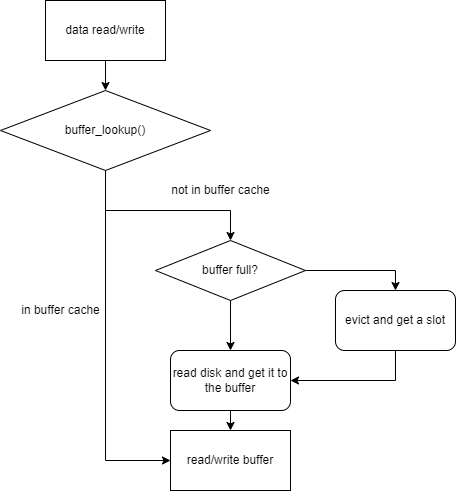
disk를 read write 하는 부분을 buffer cache를 사용하도록 변경하였다.

**filesys.c / filesys.h**

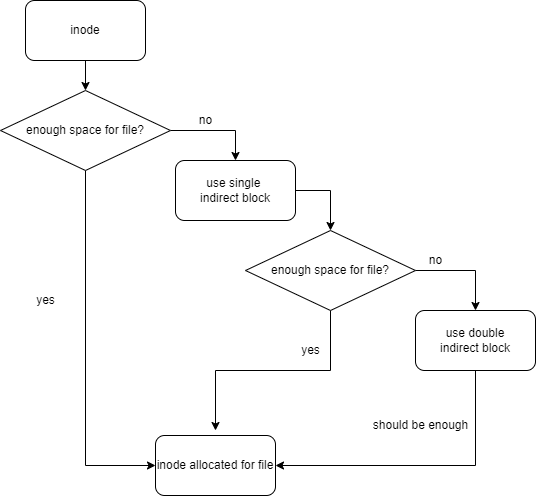
모든 함수에 직접 구현한 buffer cache 와 subdirectory의 개념을 추가하였다.

1. **연구 결과**
   1. **Flow Chart**

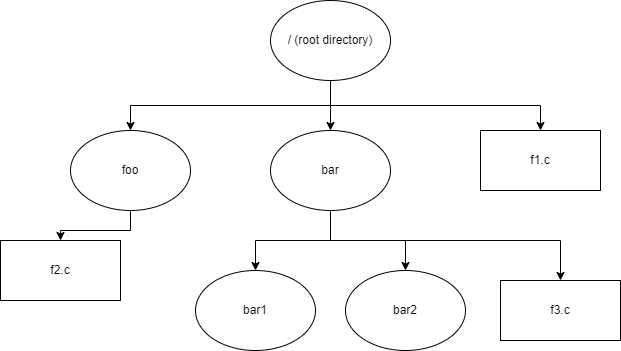
* **II. B. 개발 내용에 대한 Flow Chart를 작성**
* Buffer cache



* Extensible file & file growth



* Subdirectory



* 1. **제작 내용**
* **II. B. 개발 내용의 실질적인 구현에 대해 코드 관점에서 작성.**
* **구현에 있어 Pintos에 내장된 라이브러리나 자체 제작한 함수를 사용한 경우 이에 대해서도 설명.**
* **개발상 발생한 문제나 이슈가 있으면 이를 간략히 설명하고 해결책에 대해 설명.**
* Buffer cache

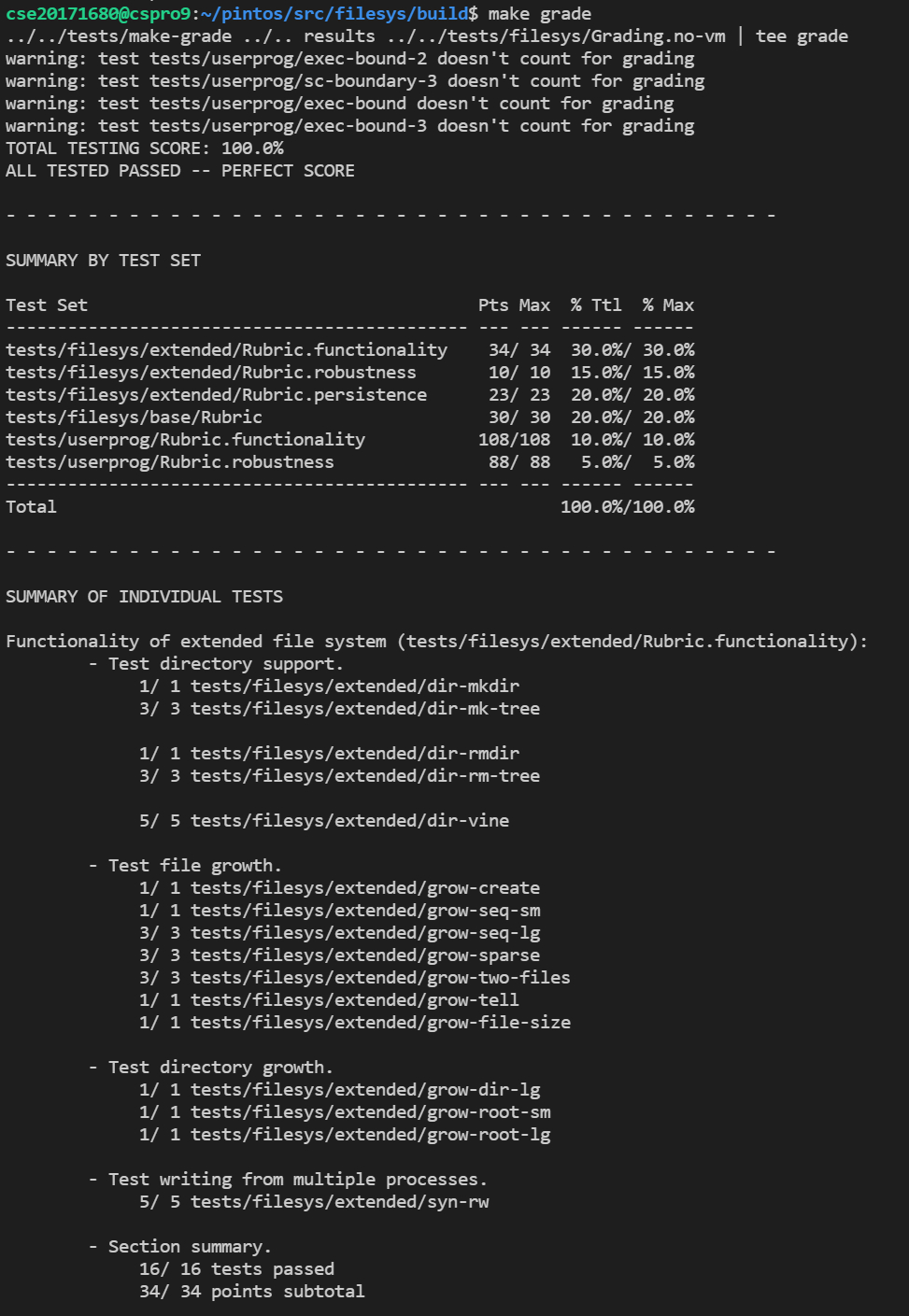
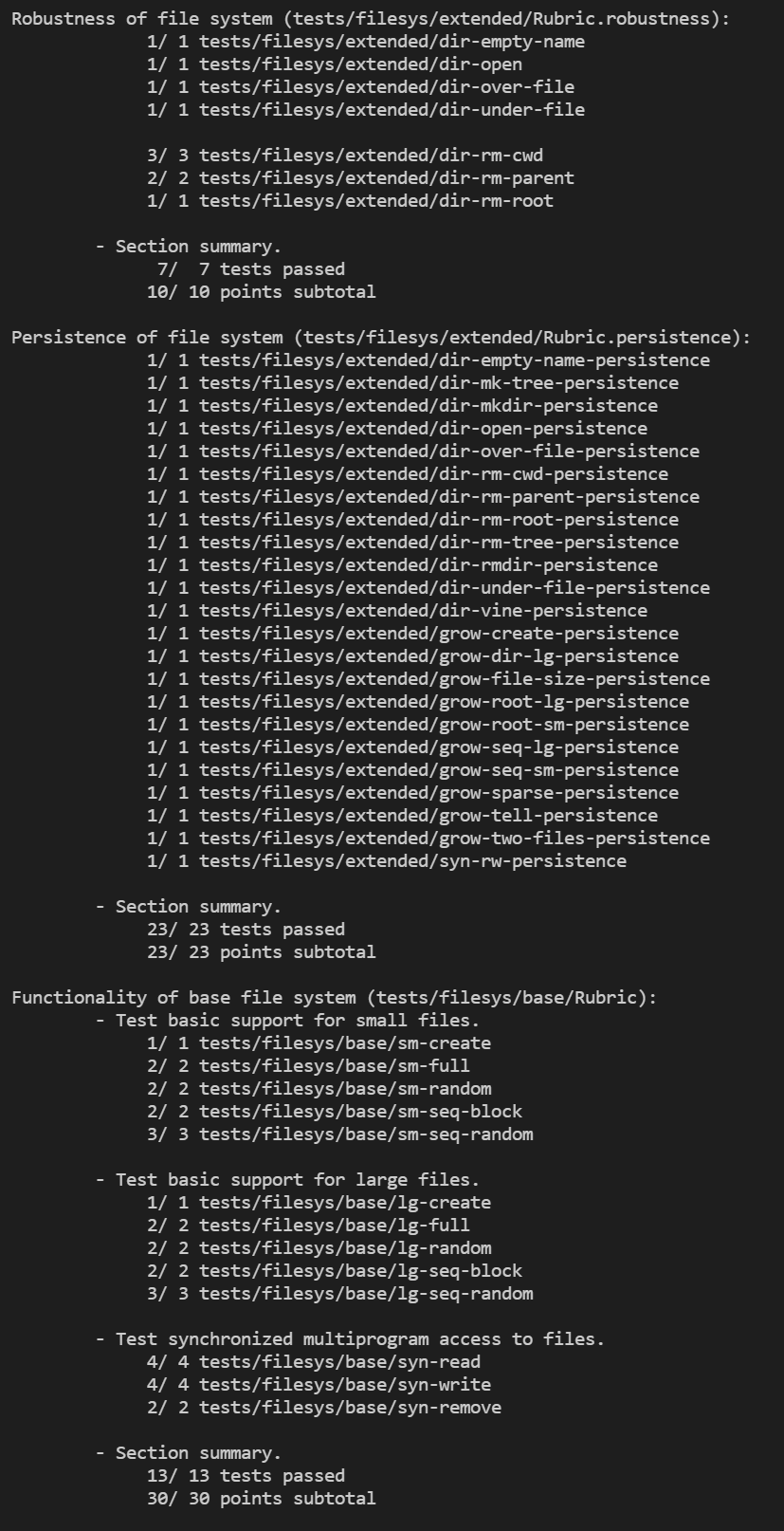
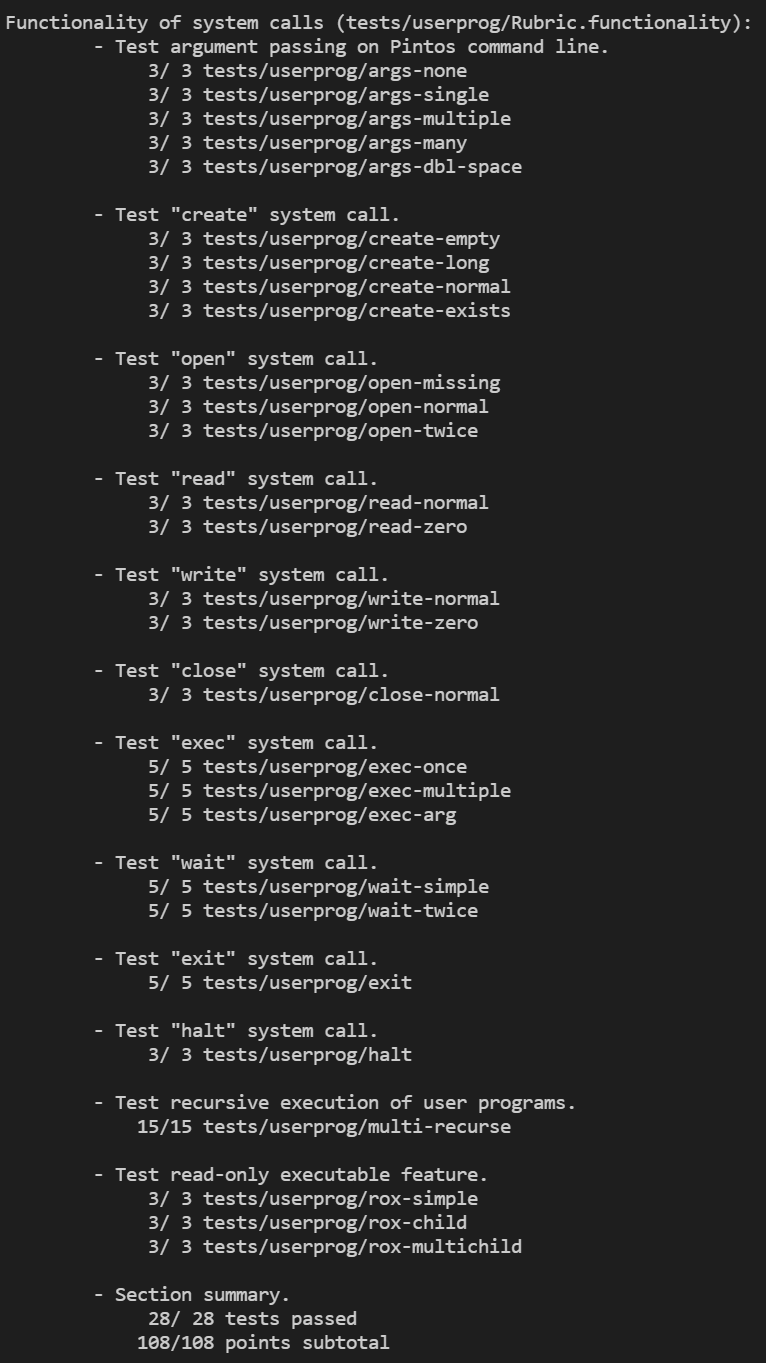
기존에 disk에 직접 read/write를 하던 것을 buffer cache를 거쳐 작동되도록 하였다. buffer\_cache\_read(), buffer\_cache\_write()의 경우 비어 있는 slot을 찾아보고 없다면 evict를 진행하여 빈 cache를 얻어낸다. 그 다음 reference\_bit()를 바꾸고 write의 경우 dirty\_bit까지 바꾸어 주도록 한다. 여기서 eviction은 second chance alogoithm을 활용하여 구현하였다. 그리고 unmount 나 OS를 종료할 때 buffer\_cache에 있는 내용을 disk로 flush를 해야 하기 때문에 buffer\_cache\_close() 가 호출된다면 buffer\_cache\_flush()를 하여 buffer cahce에 있는 data 들이 disk로 정상적으로 저장될 수 있게 하였다. 위 함수들을 사용하여 기존 disk block에 직접 접근 하는 filesys.\* 와 inode.\* 부분들 모두 buffer cache를 거치도록 변경하였다.

* Extensible file & file growth

기존 continuous allocation을 활용했던 pinots의 filesystem을 multi-level-indexing의 방식을 적용시키기 위해. inode\_disk 구조체에 direct, indirect, doubly indirect block들을 추가하였으며 indirect block의 표현을 위한 inode\_indirect\_block\_sector 구조체와 directory 여부를 확인하기 위한 is\_dir를 선언하였다. 또한, inode를 create 할 때 inode를 allocate하는 부분과 close 할 때 deallocate하는 부분을 함수를 작성하여 추가하였고 그리고는 index\_to\_sector() 함수를 새롭게 추가하여 multi-level indexing의 방식을 구현하였다. 인자로 들어온 index에 따라 direct block 영역인지 indirect 인지 doubly indirect인지 확인을 한 다음 적절한 영역의 block을 반환한다. 이를 활용하여 원래 byte\_to\_sector() 함수에서 기존의 방식으로 block을 구하는 부분을 작성한 함수로 교체하여 multi-level indexing이 구현되도록 하였다. byte\_to\_sector()를 통해 inode\_read\_at() 과 inode\_write\_at()이 작동되었는데 이는 기존에 파일을 확장하지 못해 확장을 다루는 부분이 없어 파일사이즈를 늘려주는 부분을 inode\_reserve(), inode\_reserve\_indirect()를 추가하였다.

* Subdirectory

우선 directory.c.에서는 기존에 root directory밖에 없었기 때문에 새로운 directory를 handle할 수 있도록 하여야 한다. dir\_create() 에서는 원래 directory 가 없었기 때문에 바로 inode를 만들었다면, 이 경우 sector의 첫 dir entry는 parent directory여야 하기 때문에 이를 추가하였다. dir\_open(), lookup(), dir\_add()의 경우 dir의 시작 offset이 0 이던 것을 이제 parent directory의 정보가 있기 때문에 이 다음부터 시작하도록 sizeof(struct dir)로 바꾸었으며 실제 parent directory에 관한 정보를 추가하는 부분은 dir\_add()에 추가하였다. dir\_lookup()의 경우 기존 inode만 확인하면 되었는데 이제는 자기자신(.) 과 부모(..)의 경우도 추가해 주었다. 또한, directory를 remove 할 때 이제는 안에 내용물이 있는 directory는 제거하면 안되기 때문에 이 조건을 확인하는 부분을 dir\_remove()에 추가하였다. 이에 추가로 작성한 함수들로는 filesys에서 path name handling을 하기 위해 split\_path\_filename()함수를 작성하였는데 이 함수는 path 가 주어지면 directory와 filename 인자에 각각의 값을 반환한다. 또 다른 함수로는 dir\_open\_path()가 있는데 이 함수는 기존에는 root 만 열었기 때문에 dir\_open\_root()를 활용하였는데 이제는 directory가 생겼기 때문에 dir를 열 수 있도록 함수를 작성하였다. 위에서 설명한 함수들을 이용하여 filesys.c의 경우 filesys\_create()와 filesys\_open(), filesys\_remove()는 무조건 root를 참조하는 방식에서 split\_path\_filename() 과 dir\_open\_path()를 활용하여 directory를 열도록 변경하였다. 이에 추가로 dir을 바꾸는 경우의 함수도 thread에 현재 directory를 가리키는 변수 cwd를 추가하여 file\_sys\_chdir()를 구현하였다.

* 1. **시험 및 평가 내용**
* **Src/filesys make grade 수행결과를 캡처 하여 첨부.**
* 
* 
* 
* 