**Pintos Project 5: Filesystem**

담당 교수 : 박성용 교수님

조 / 조원 : 20181641 백승훈

개발 기간 : 2022.12.21 ~ 2022.12.26

1. **개발 목표**

* **해당 프로젝트에서 구현할 내용을 간략히 서술.**

1. file extends를 구현하여 파일의 크기를 자유롭게 변경 가능하도록 한다.
2. subdirectory system을 구현하여 root이외의 다른 디렉터리를 사용 가능하도록 한다.
3. buffer cache를 구현하여 file 접근 시 성능을 향상시킨다.
4. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Extensible file & file growth

파일의 크기를 자유롭게 변경 가능할 수 있고, index block을 통해 block의 크기보다 더 많은 sector를 pointing할 수 있게 되어 더 큰 크기의 파일을 다룰 수 있다.

1. Subdirectory

root 디렉터리 이외의 다른 디렉터리들을 사용할 수 있게 된다

1. Buffer cache

block\_read / write 시 바로 file에 쓰지 않고 중간에 버퍼를 둠으로써 파일 r/w의 성능이 향상된다.

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

기존 핀토스는 따로 indexing을 구현하지 않고, 처음 inode 생성시 시작 sector idx부터 블록들을 연속적으로 할당하고 있었다. 그러다보니 external fragmentation의 문제가 있었다. 이를 해결하기 위해 index structure를 도입하였다. index structure에서는 inode에 해당 file이 사용하는 block들의 sector index를 저장한 배열인 index block을 사용한다.

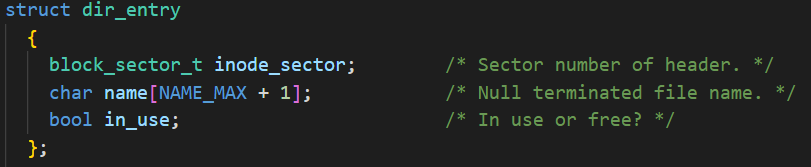
index block을 사용하여 indexing 할 때 direct / indirect / double indirect index을 사용하였다. 우선 inode\_disk 구조체에 남는 공간을 사용하여 direct indexing을 위한 배열을 선언하였고, 또한 indirect / double indirect를 위해 만드는 index block을 가리키기 위핸 sector index도 추가해주었다.

이후 inode\_create를 수정하였다. 현재 inode\_create는 필요한 sector수 만큼의 sector를 연속적으로 할당하려고 하는데, 이를 한 sector 할당 후 indexing하는 방식으로 수정하였다.

이후 inode\_write / inode\_read 에서 해당 파일에서 접근하고자 하는 offset을 주면, 해당 offset이 접근하려 하는 sector를 계산하는 함수를 사용하여 inode\_disk 내부의 인덱스 블록들에 저장된 실제 block sector index 값을 반환하게 하여 block에 접근해 i/o를 구현하였다.

* Subdirectory
  + Directory entry 관리 방법

핀토스는 directroy entry를 dir\_entry라는 20바이트짜리 구조체를 이용하여 관리한다.



이는 directory file에 할당된 block에 저장되고, directory에서 파일을 관리하는 일이 생길 때 해당 블록에서 20바이트씩 entry를 읽어오면서 처리한다.

디렉터리에 파일이 추가된 경우 dir\_add를 통해 디렉터리의 블록에 접근한 후, 해당 파일의 inode sector index와 file name을 블록에 append한다.

결국 서브디렉터리를 만들기 위해서는 inode에 이것이 디렉터리인지 확인할 수 있는 구조체를 추가하고, filesys에서 file을 생성하고 읽을 때 입력받은 file name을 분석하여 디렉터리와 파일을 구분하여 r/w하도록 구현하면 된다.

inode가 우선 directory인지 확인할 수 있게 inode\_disk 구조체에 is\_dir를 추가하였다. 각 inode를 생성하는 부분에서 상황에 맞게 is\_dir 값을 정할 수 있게 수정해주었다.

그 다음 filesys와 directory에서 입력받은 name에 따라 디렉터리 / 파일을 구분하여 파일 입출력을 처리할 수 있도록 수정하였다. 그리고 디렉터리와 관련된 system call (rmdir, mkdir, chdir, isdir, inumber)를 구현하여 파일시스템 내부에 폴더를 생성하고 수정할 수 있도록 구현하였다.

* Buffer cache
  + Buffer cache eviction 방식
  + Buffer cache flush 방식

Project4에서 page를 replace할 때 사용했던 방식과 동일한 방식인 second chance algorithm을 사용하였다. Buffer Cache를 관리하는 구조체에 reference bit를 추가한다.

이후 buffer가 가득 차서 evict해야 하는 경우에는 각 buffer cache의 reference bit을 확인하며 reference bit이 0인 buffer를 찾는다, 만약 이번에 검사한 buffer의 reference bit이 1인 경우는 bit을 0으로 바꿔준다.

buffer\_read / write시 reference bit을 1로 바꿔, 최근에 접근한 buffer들이 더 오래 buffer에 있을 수 있도록 한다.

한 buffer를 evict할 경우에는 해당 buffer를 block에 새로 써야 하는지 결정해야 한다. 이를 위해 dirty bit을 추가하고, buffer\_write시에는 dirty bit을 1로 변경해준다. 만약 buffer evict 시 dirty bit이 1이라면 block\_write를 호출하여 블록의 내용을 갱신해준다.

또한 버퍼는 메모리에 있지만 실제로는 파일에 저장되어야 하는 값이므로, 프로그램 종료 시 반드시 flush 해주어야 한다. file system terminate시 buffer flush를 구현하여 dirty bit이 1인 buffer들을 file에 기록하도록 한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

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

2022/12/21 ~ 2022/12/21: Buffer Cache 구현

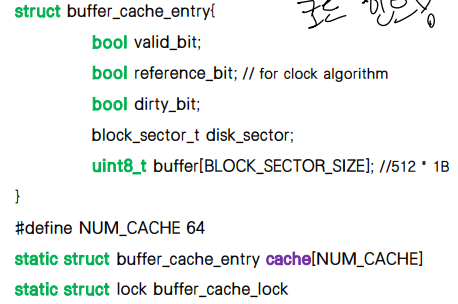
2022/12/22 ~ 2022/12/25: File Extension 구현

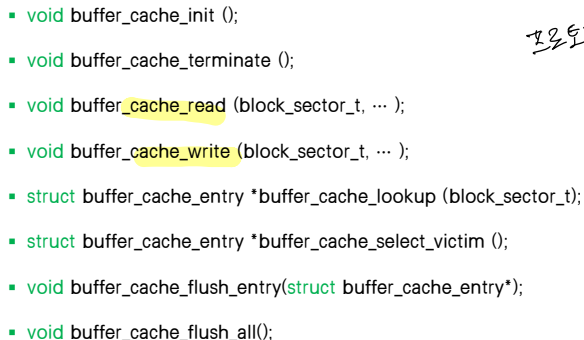
2022/12/26: Subdirectory 구현

* 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)**

1. Buffer Cache

filesys 내에 cache.c / cache.h 파일을 구현해야 한다. 명세서의 내용과 동일하게 구하였으며, 내용은 아래와 같다.





위의 내용을 구현한 후 inode\_write\_at과 inode\_read\_at에서 block\_read / write를 사용할 때 해당 함수 대신 buffer\_cache\_write와 buffer\_cache\_read를 사용하도록 구현하였다.

또 pinotos init시 buffer\_cache\_init을 호출하여 버퍼캐시를 사용 가능하도록 구현하였고, 프로그램 종료 시 flush\_all을 호출하여 모든 내용을 디스크에 저장하도록 구현하였다.

1. File Extension

우선 inode.c의 inode\_disk 구조체를 수정해야 한다. 기존 시작 섹터와 길이를 제거하고 index block을 사용할 수 있도록 direct block 배열과 indirect, double indirect index block pointer를 추가한다. 이 때 inode\_disk의 크기가 한 블록 크기보다 넘지 않도록 설정해야 한다.

이후 offset에 따른 index block 결정을 위한 계산 함수를 추가해야 한다. 또 inode\_create및 inode\_write에서 파일 크기가 변경되어 새로운 블록을 할당해야 할 때 사용할 block\_allocate 함수를 추가해야 한다. 해당 함수에서 블록을 할당한 후 offest에 따라 index block에 이번에 할당한 sector idx를 기록해준다. 또 inode\_close시 생성한 block을 해지할 수 있도록 모든 인덱스 블록을 순회하며 bitmap을 정리하는 함수를 추가한다.

위 함수들을 사용하여 inode\_create에서 inode 생성 시 새로운 블록을 할당할 때 block\_allocate를 호출하여 수정할 수 있게 하고, inode\_write에서 file의 크기가 변경되어 새로운 블록이 필요한 경우에도 호출하도록 한다. inode\_read\_at에서는 offset에 맞는 index를 찾아 접근할 수 있도록 offset을 통해 실제 sector idx를 찾아주는 함수를 사용하도록 수정한다.

1. Subdirectory

우선 inode\_disk 구조체에 디렉터리인지 여부를 확인해주는 is\_dir flag를 추가한다. 그 다음 file에서 directory를 다룰 수 있도록 경로를 분석해주는 make\_path 함수를 추가한다. 함수에서는 상대 경로와 절대 경로를 모두 분석하여 현재 파일이 위치한 directory와 파일의 실제 이름을 반환해야 한다. (pintos/a/b/c일 경우 디렉터리는 b고 파일 이름은 c이다)

또한 mkdir system call에서 사용할 filesys\_create\_dir 함수를 추가해야 한다.

이후 filesys create / open / remove 에서 해당 path함수를 사용하여 기존 무조건 루트 디렉터리를 열어 파일을 처리하던 로직을 현재 디렉터리를 열어 처리할 수 있도록 함수를 수정해야 한다.

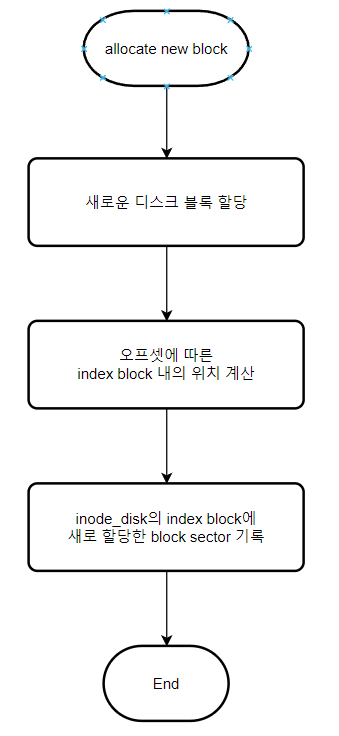
이후 dir\_create / remove를 수정하여 서브디렉터리 내부에서 파일을 수정/제거할 수 있도록 수정해야 한다.

또한 서브디렉터리 시스템에서는 각 폴더에 상대 경로를 위한 “.” 과 “..”을 디렉터리 엔트리에 추가해야 한다. 루트 디렉터리부터 서브디렉터리까지 위 엔트리들을 사용할 수 있도록 루트디렉터리 및 서브디렉터리 생성시 추가해준다.

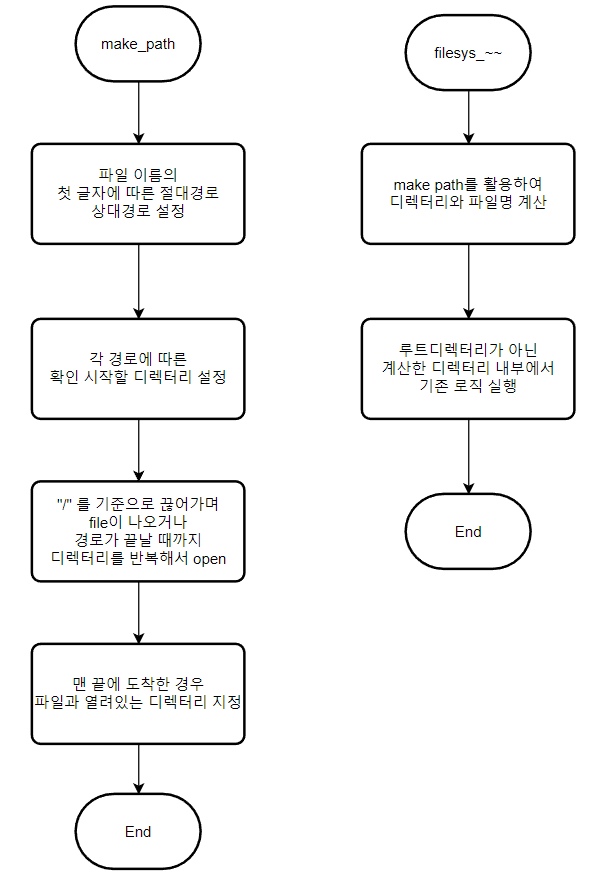
마지막으로 file system과 관련한 미구현 system call들을 구현해야 한다. isdir, chdir, mkdir, readdir, inumber system call을 구현한다. 또한 chdir 및 상대 경로를 사용하기 위해, 현재 쓰레드에서 접근중인 디렉터리에 대한 정보가 필요하다. 그러므로 thread 구조체에 현재 디렉터리 정보를 나타내기 위한 값을 추가해야 한다.

1. **연구 결과**
   1. **Flow Chart**

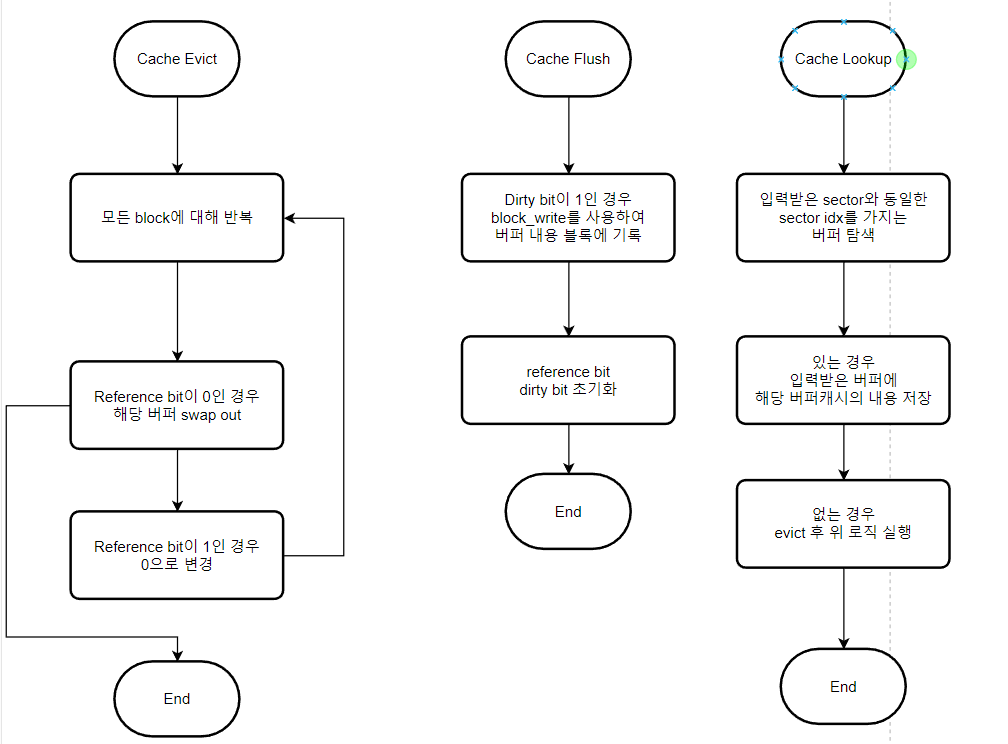
* **II. B. 개발 내용에 대한 Flow Chart를 작성**
* Extensible file & file growth



* Subdirectory

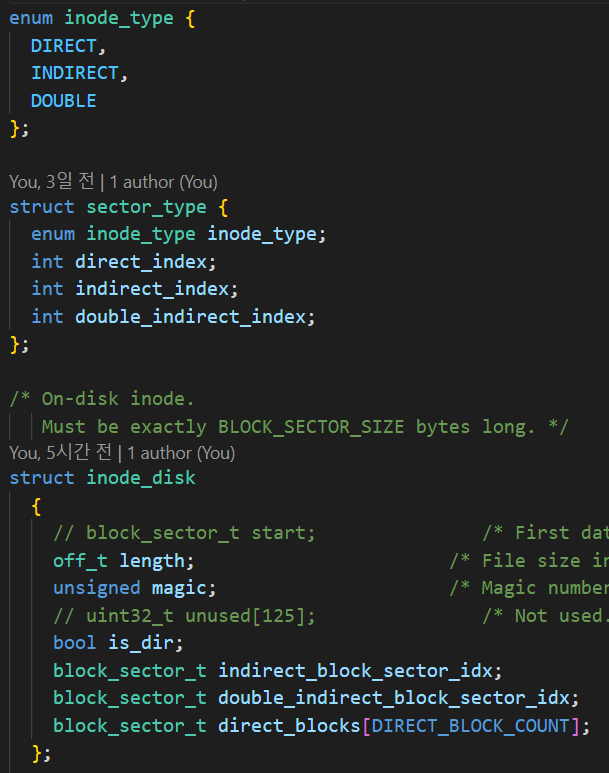


* Buffer cache  
  **구현한 경우만 작성**

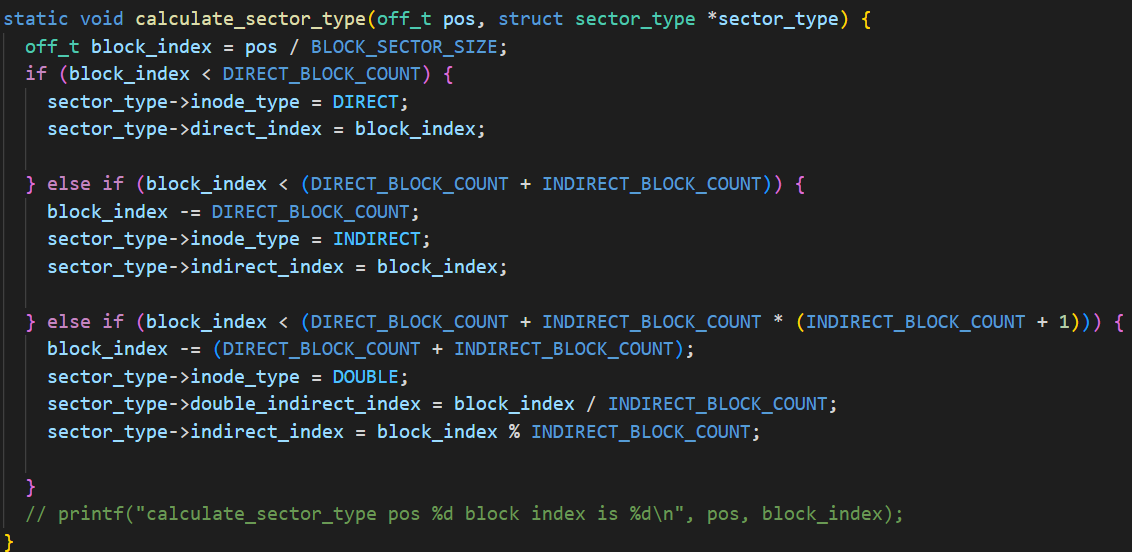
****

* 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.**
* Extensible file & file growth

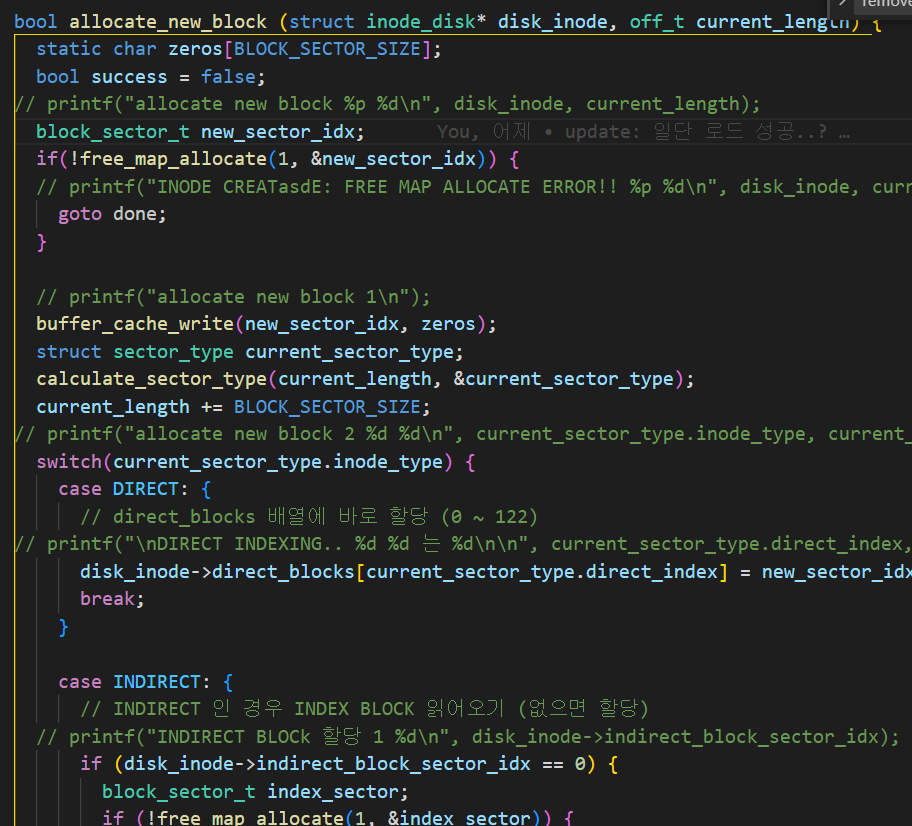
우선 inode\_disk 구조체를 수정하고 필요한 구조체들을 추가하였다.



이후 offest에 따라 지정될 index block의 index를 계산해주는 함수를 구현하였다.

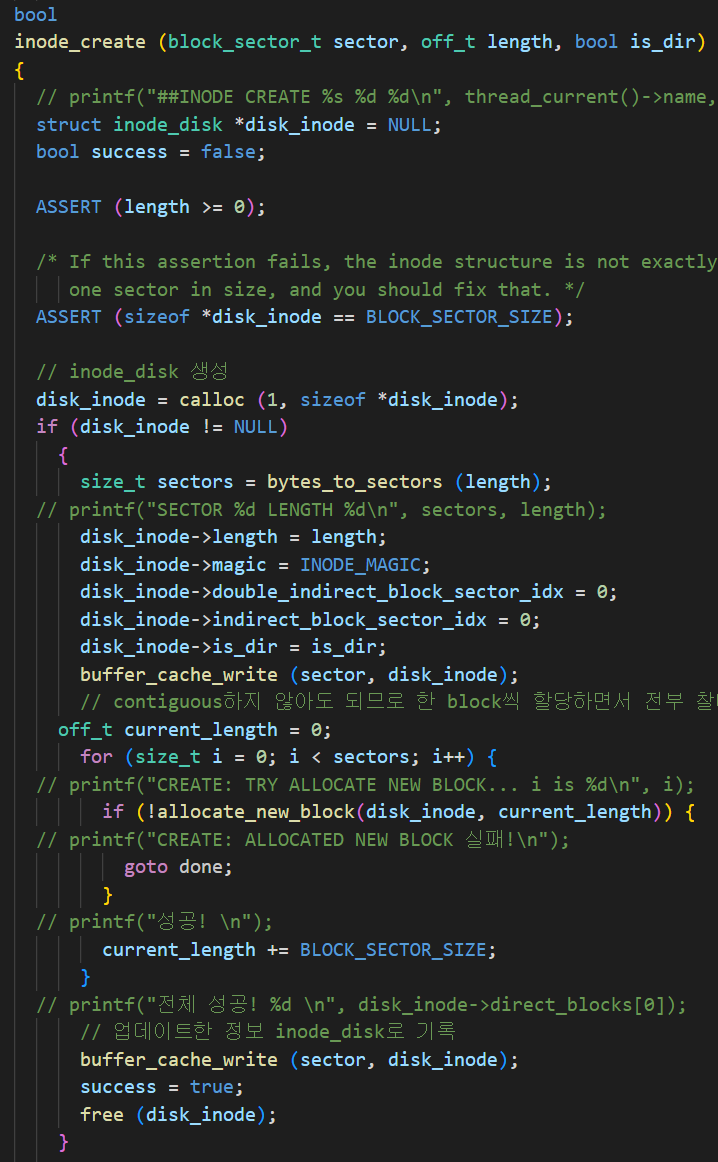


위 함수를 사용하여 block을 연속적으로 할당하지 않고, 하나씩 할당할 수 있도록 allocate\_new\_block 함수를 구현하였다.



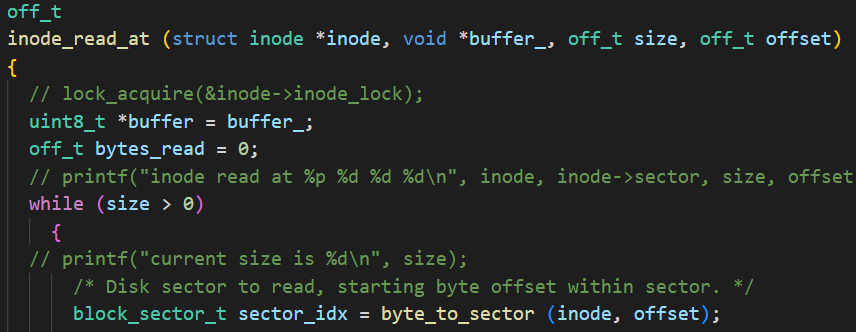
calculate\_sector\_type 함수를 사용하여 인덱스를 정한 후 새로 생성한 블록 인덱스를 할당해준다.

위 함수를 사용하여 블록을 할당하도록 inode\_create와 inode\_read를 수정하였다.





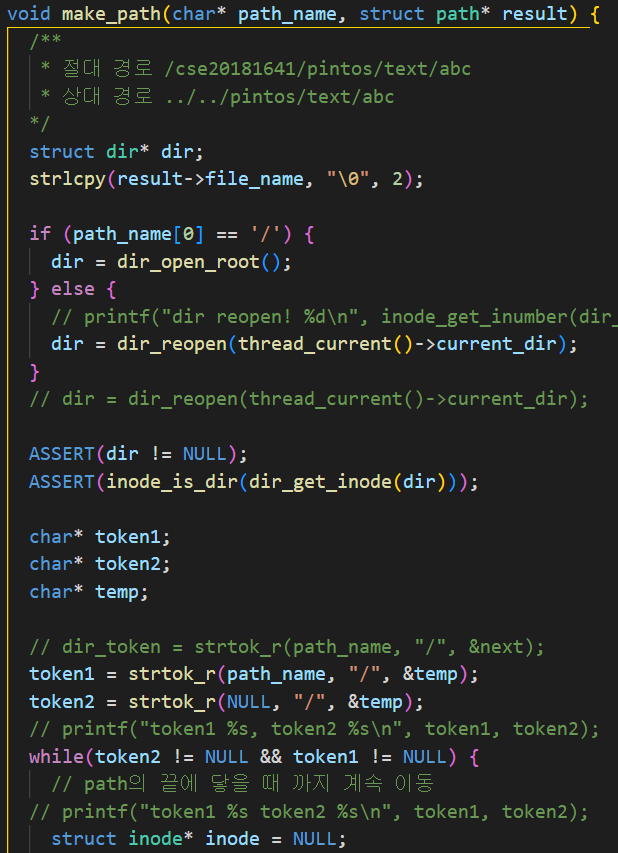
offset + size가 이전 길이와 달라진 경우 블록이 필요한지 확인하고 allocate\_new\_block을 통해 새로 할당해준다.

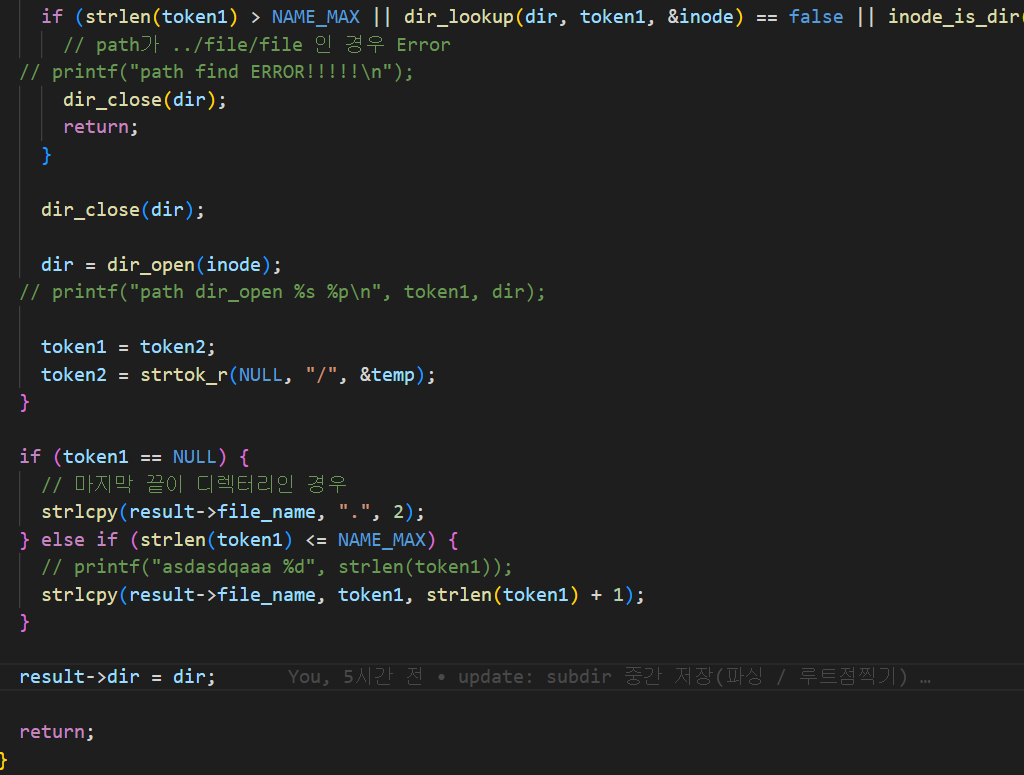


read\_at에서는 기존 byte\_to\_sector를 수정하여 사용하므로 따로 수정할 부분이 없었다.

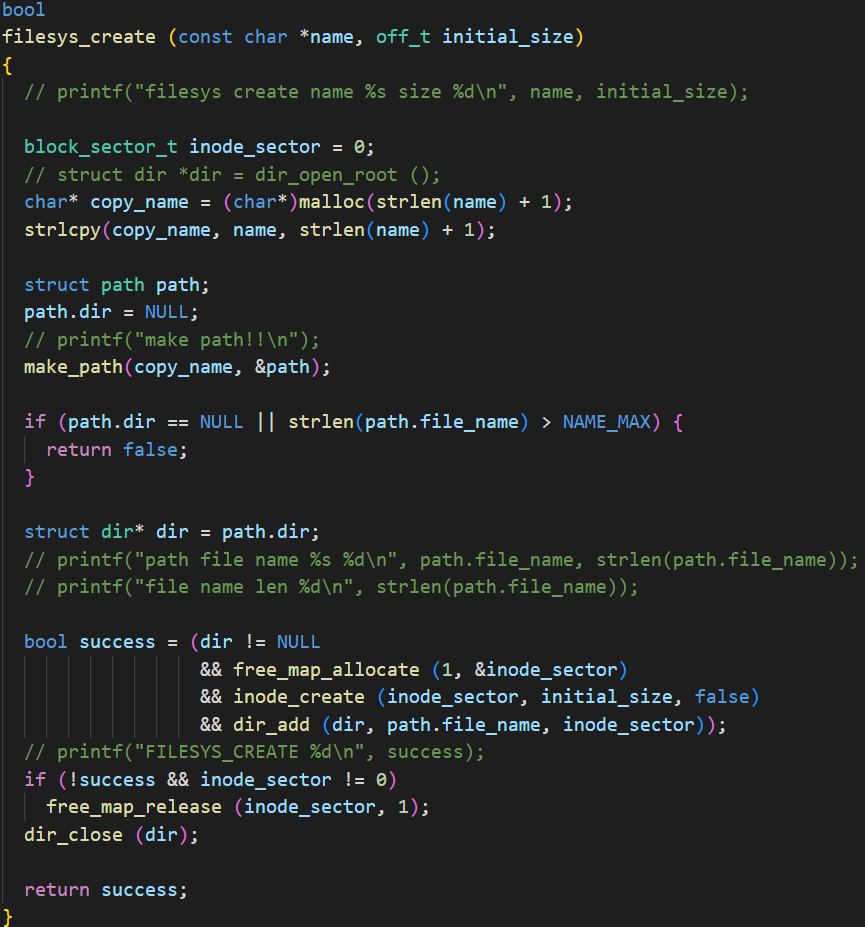
* Subdirectory

우선 경로를 통해 디렉터리와 파일이름을 찾아주는 함수를 구현하였다.





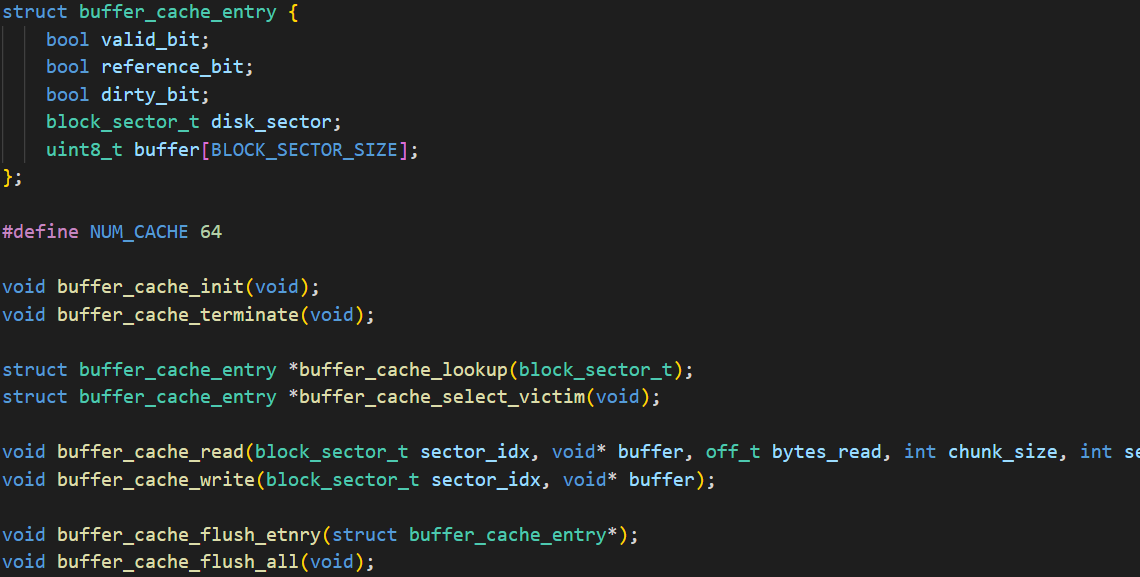
이후 filesys\_create / open / remove 에서 해당 경로 찾아주는 함수를 이용하여 경로 및 파일을 찾고 파일을 관리할 수 있도록 수정해주었다.



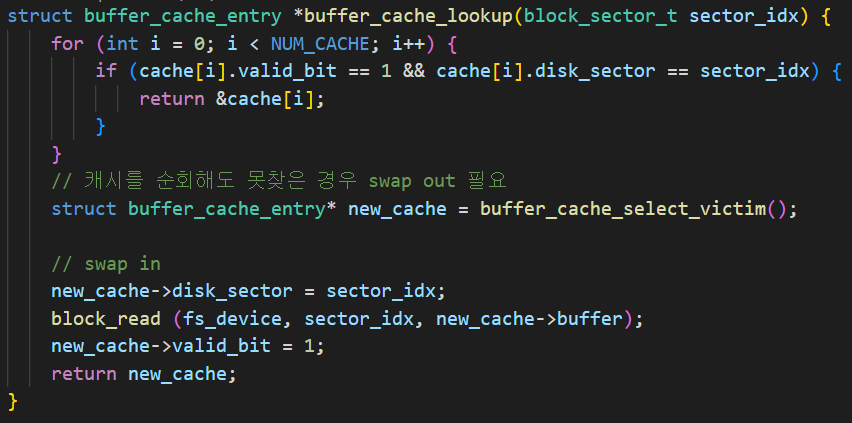
위와 같은 방식으로 make\_path를 이용, dir과 file을 구해 인자를 변경해주었다.

* Buffer cache

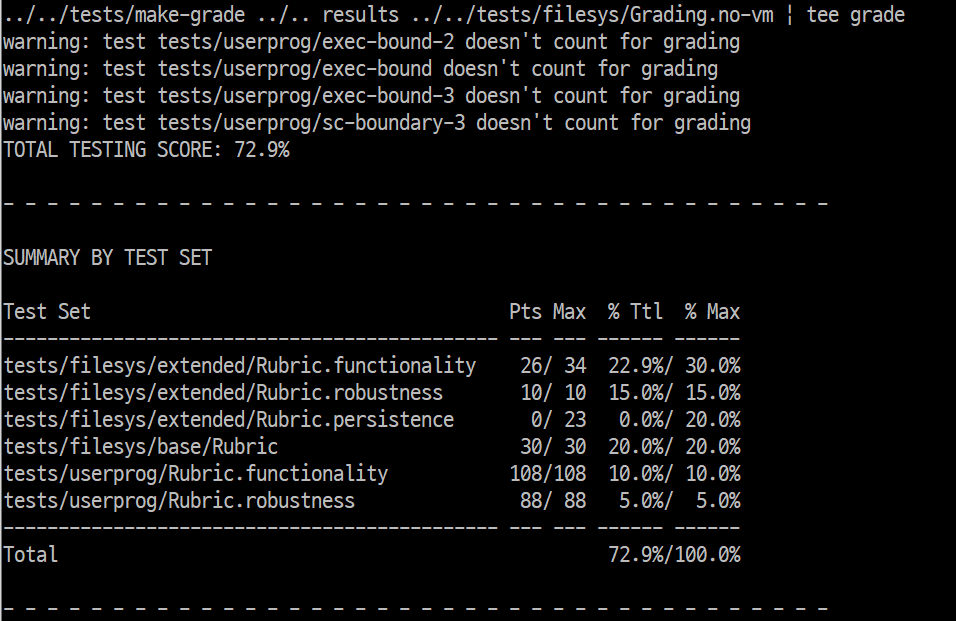
우선 buffer cache 구현을 위해 명세서에 기술된 내용대로 함수를 작성하였다.



victim select 함수 및 flush 함수는 아래와 같이 구현하였다.



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

****