**Pintos Project 5: Filesystem**

담당 교수 : 김영재

조 / 조원 : 한솔

개발 기간 : 2021.11.29 ~ 2021.12.22

1. **개발 목표**

* **해당 프로젝트에서 구현할 내용을 간략히 서술.**

파일시스템에서 하나의 파일이 연속적인 공간을 가지지 않을 수 있도록 구현하고, 또한 파일의 크기를 확장할 수 있도록 한다. 그리고 현재의 핀토스에서 루트 디렉토리만 가능한데, 여기서 서브 디렉토리를 사용할 수 있도록 구현한다. 추가적으로, 디스크에 바로 write, read하지 않고 메모리에 cache를 만들어서 이를 통해 read, write할 수 있도록 buffer cache를 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Extensible file & file growth

기존의 핀토스에서 파일의 크기가 고정되어 있어서 정해진 블락보다 큰 데이터를 입력할 수 없었는데, file growth를 구현하면 파일의 크기를 늘릴 수 있게된다. 또한, 파일이 연속적인 블락을 가지지 않아도 되도록 설계해서 디스크 공간을 더 효율적으로 사용할 수 있게된다.

1. Subdirectory

기존의 핀토스는 루트 디렉토리에만 파일을 추가할 수 있는데, 이를 수정해서 루트 디렉토리 밑에 여러 개의 서브 디렉토리를 만들 수 있게된다.

1. Buffer cache  
   **구현한 경우만 작성**

디스크에서 직접적으로 read, write을 하면 시간적으로 비효율적이기 때문에, 메모리에 cache를 만들고 빠르게 read, write할 수 있게된다.

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

기존의 inode는 파일의 시작 주소와 크기만으로 구성되어있어서 연속적인 블락공간밖에 표현하지 못하는 한계가 있었다. 이를 inode가 가리키는 파일을 구성하는 블락들 각각을 나타내는 index로 수정해서 연속적이지 않은 공간에도 파일이 확장될 수 있도록 하는 것이다.

* Subdirectory
  + Directory entry 관리 방법

새로운 directory entry를 생성하고 첫번째 entry는 부모 directory로 설정한다. 자식노드에서 부모노드로 이동할 수 있어야하기 때문이다. 파일을 생성할 떄 is\_dir라는 변수를 저장해서 그 파일이 directory인지 아닌지를 나타내며, 이 방식으로 directory를 추가할 수 있다.

* Buffer cache
  + Buffer cache eviction 방식

64개의 캐시 block중에서 하나씩 돌아가면서 valid bit와 reference bit를 확인한다. valid하지 않은 경우에는 바로 그 block을 반환하면 되고, 아닌 경우에는 reference bit를 확인한다. reference bit가 true인 경우에 는 false로만 수정해준다. false인 경우에는 그 블락을 반환하면된다. 이렇게 되면 적어도 2바퀴내에 하나의 블락을 반환할 수 있다.

* + Buffer cache flush 방식

flush를 할 때는 flush할 entry의 dirty 비트를 확인한다. true인 경우에는 write이 있었던 것이므로 disk에다가 해당 블락을 쓰고 dirty bit를 false로 수정해주면된다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

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

1. extensible file & file growth 12.02 ~ 12.12
2. subdirectory 12.13 ~ 12.21
3. buffer cache 11.29 ~ 12.01
   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.c파일을 추가해서 disk에 바로 read,write하지 않고 캐시를 거치도록 추가한다. 구체적으로는 buffer\_cache\_init, buffer\_cache\_terminate, buffer\_cache\_select\_victim, buffer\_cache\_lookup, buffer\_cache\_flush\_entry, buffer\_cache\_read, buffer\_cache\_wrtie를 추가한다. select\_victim은 64개의 캐시 메모리중에서 어떤 블락을 사용할 지를 정하는 함수이고, lookup은 블락의 sector를 입력받아서 대응되는 캐시를 반환하는 함수이다. flush는 cache에 데이터를 disk로 옮기는 함수이다.

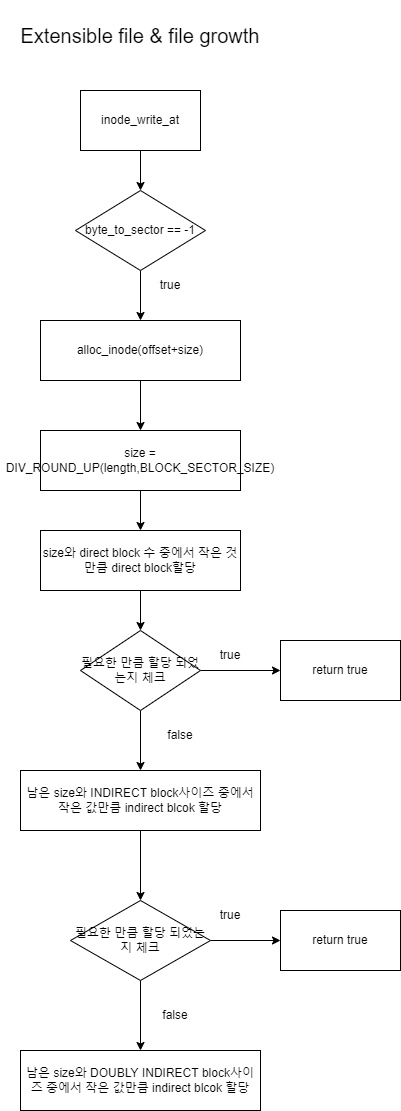
directory.c에서 dir\_create을 수정한다. inode\_create를 한 후에 임시로 자기 자신을 가리키도록 한다. 이는 나중에 dir\_add에서 부모를 가리키도록 변경된다. lookup함수에서 ofs을 0부터 시작하며 directory를 탐색하던 것을 ofs을 sizeof(e)로 수정한다. 왜냐하면 첫번째 entry는 부모 디렉토리를 가리키기 떄문이다. dir\_lookup에서 “.”과 “..”이 입력되었을 때 자기 자신과 부모노드를 가리키도록 수정한다. dir\_add에서는 is\_dir을 입력받아서 directory를 추가하는 것인지 확인한 후에 디렉토리에 이름을 추가하는 것이라면, 그 directory의 첫번째 entry에 부모노드를 입력해준다. dir\_remove에서는 비어있지 않은 디렉토리를 삭제하지 못하도록 삭제하기 전에 directory를 탐색하는 기능을 추가한다. extract\_directory\_filename\_from\_path함수를 추가한다. 이는 path를 입력받아서 directory와 filename을 구분해서 각각 다른 변수에 저장해준다. dir\_open\_path를 추가하는데, 이는 path를 입력받아서 절대경로인지 상대경로인지 판단해서 처리한 후에, path에 해당하는 directory를 반환한다.

filesys.c에서는 우선 filesys\_done함수에서 buffer\_cache\_terminate를 추가해준다. filesys\_create에서는 dir\_open\_root로 루트디렉토리를 오픈하는대신에 extract\_directory\_filename\_from\_path를 통해 입력받은 path에서 directory name와 filename을 분리한 후에 dir\_open\_path를 통해 directory name에 해당하는 directory를 얻도록 수정한다. filesys\_open함수도 기존의 dir\_open\_root를 통해서 루트 디렉토리를 얻는 것에서 extract\_directory\_filename\_from\_path와 dir\_open\_path를 이용해서 수정한다. 또한 얻은 directory가 삭제되었을 경우에 대해서 처리해준다. filesys\_remove도 마찬가지로 위와 같은 처리를 해준다. filesys\_chdir를 추가해주고, 현재 스레드의 current\_dir를 입력받은 path에 해당하는 directory로 수정해주는 기능을 구현한다.

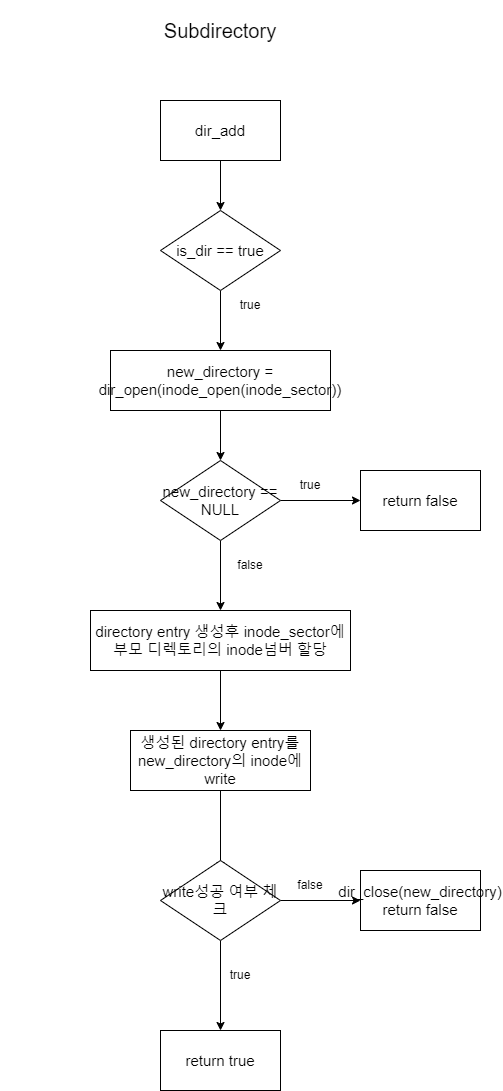
inode.c에서 byte\_to\_sector를 inode가 가리키는 시작 주소에서 일정한 크기를 더하는 것이 아니라 get\_sector\_number라는 새로 작성할 함수를 이용해서 반환한다. get\_sector\_number는 밑에서 설명한다. inode\_create함수에서 free\_map\_allocate가 아니라 새로 작성할 alloc\_inode함수를 이용해서 디스크공간을 할당하는 것으로 수정한다. inode\_write\_at함수에서 더 이상 파일에 공간이 없는 경우 alloc\_inode를 이용해서 공간을 늘려서 write하는 기능을 추가한다. allocate\_block함수를 추가한다. 이는 b입력받은 entry에 블락을 할당하는 함수이다. 또한 alloc\_inode, alloc\_inode\_indirect, alloc\_inode\_doubly\_indirect라는 함수를 추가한다. alloc\_inode는 파일의 크기를 입력된 크기만큼으로 확장시키는 기능을 하고 indirect block과 doubly indirect block을 사용해야하는 경우 alloc\_inode내부에서 alloc\_inode\_indirect, alloc\_inode\_doubly\_indirect함수를 호출해서 사용한다. 또한 inode를 deallocate하기 위해서 dealloc\_inode, dealloc\_inode\_indirect, dealloc\_inode\_doubly\_indirect함수를 추가한다. 마지막으로 블락의 index를 입력받아서 해당 블락의 sector를 반환하는get\_sector\_number함수를 추가한다.

1. **연구 결과**
   1. **Flow Chart**

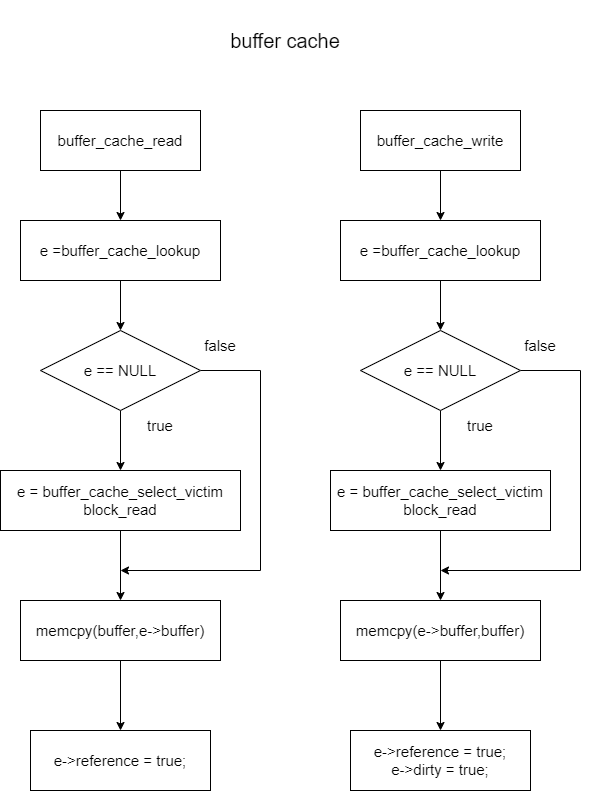
* **II. B. 개발 내용에 대한 Flow Chart를 작성**
* Extensible file & file growth



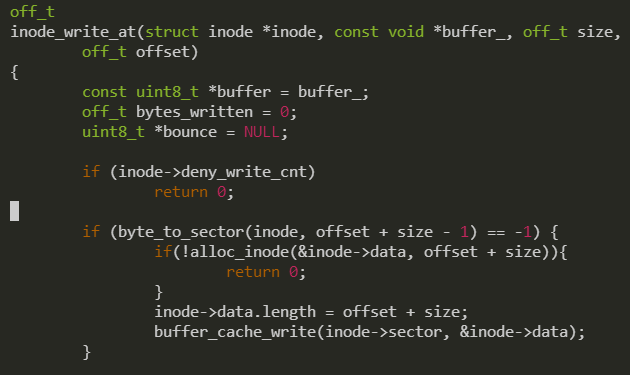
* Subdirectory



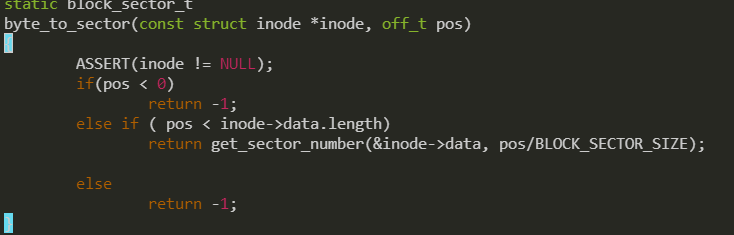
* Buffer cache  
  **구현한 경우만 작성**



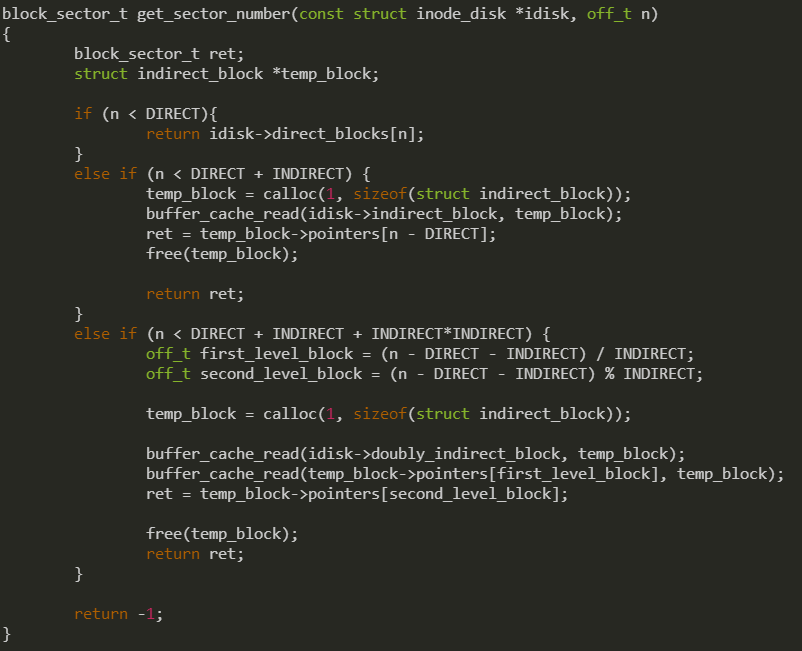
* 1. **제작 내용**
* **II. B. 개발 내용의 실질적인 구현에 대해 코드 관점에서 작성.**
* **구현에 있어 Pintos에 내장된 라이브러리나 자체 제작한 함수를 사용한 경우 이에 대해서도 설명.**
* **개발상 발생한 문제나 이슈가 있으면 이를 간략히 설명하고 해결책에 대해 설명.**
* Extensible file & file growth



inode.c의 inode\_write\_at함수에 위와 같은 부분을 추가한다. offset + size – 1에 해당하는 위치를 입력으로 byte\_to\_sector라는 함수를 실행하는데, 이 함수는 다음과 같다.

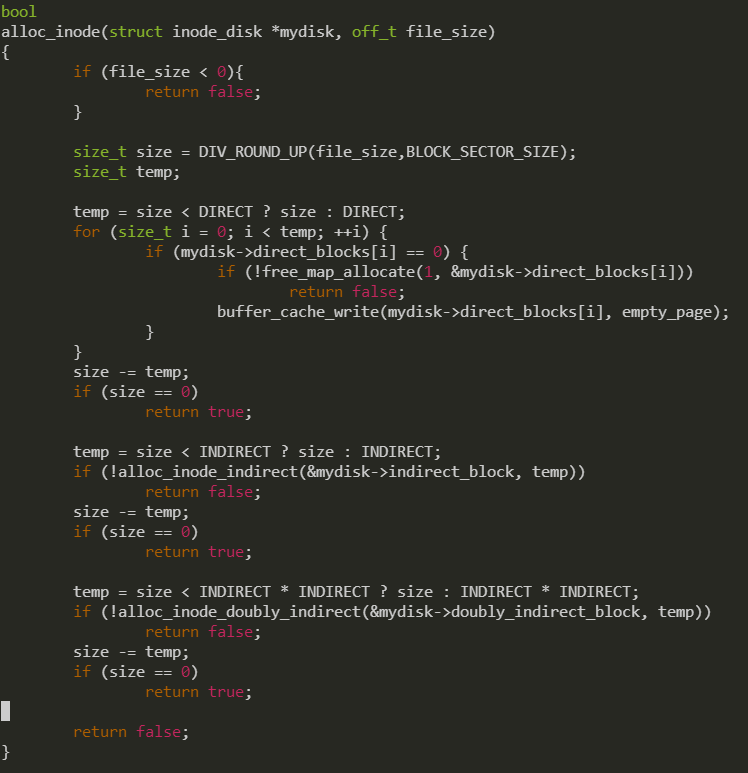


파일의 크기보다 작은 위치에 쓰려고 하는 경우에는 get\_sector\_number를 반환한다. 이는 다음과 같다.

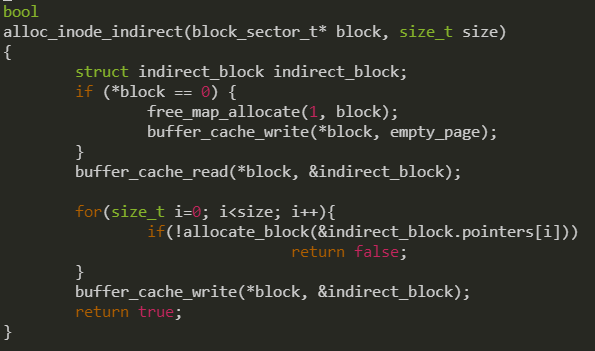


블락의 index를 입력받아서 위와 같이 direct\_block에 해당하는지 indirect\_block에 해당하는지 혹은 doubly\_indirect\_block에 해당하는 지에 따라서 그 블락의 섹터를 반환한다. 즉 inode\_write\_at함수는 여기서 file의 크기를 넘은 곳에 write하려고 할 때 새로 추가한 alloc\_inode함수를 통해서 file을 크기를 확장한다. 그리고 변경된 사항을 다시 disk에 write해준다.

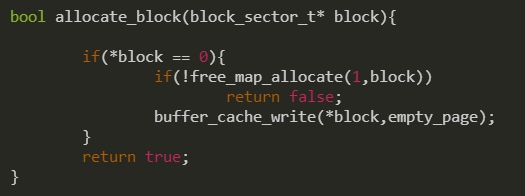
alloc\_inode는 다음과 같다.



확장하려는 파일의 크기를 입력받아서 이를 섹터 단위로 나눠서 몇 개의 섹터인지 파악한다. 그리고 그 섹터 수가 direct block안에 들어간다면 할당되지 않은 block들에 대해서 if(mydisk->direct\_block[i] == 0)안에서 새롭게 할당해준다. 할당해야할 블락이 더 남은 경우에는 alloc\_inode\_indirect를 통해서 indirect\_block을 할당해준다. 이렇게 할당하고도 더 할당해야할 크기가 남아있다면 alloc\_inode\_doubly\_indirect함수를 통해서 나머지를 할당한다. alloc\_inode\_indirect는 다음과 같다.

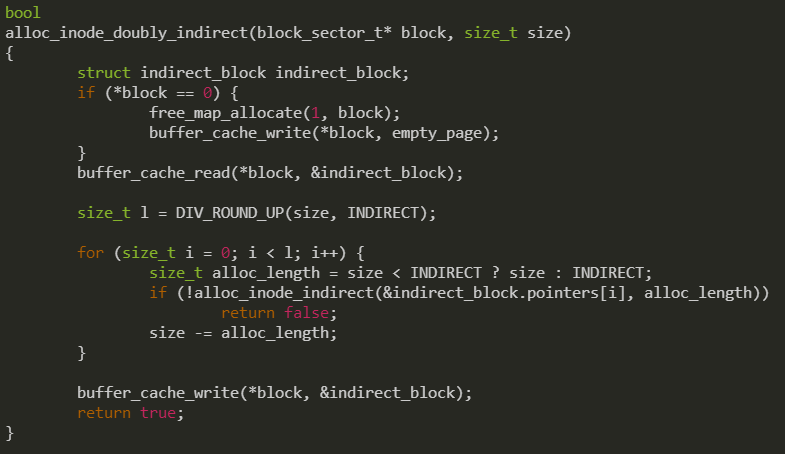


이는 우선 indirect block이 할당되지 않았다면 할당해준다. 그리고 입력해야하는 크기만큼 allocate\_block함수를 호출해서 해당 블락들을 할당한다. allocate\_block함수는 다음과 같다.



allocate\_block함수는 이미 할당된 영역이라면 그냥 true를 반환하고 그렇지 않다면 free\_map\_allocate함수를 이용해서 sector하나를 할당한다.

alloc\_inode\_doubly\_indirect함수는 다음과 같다.

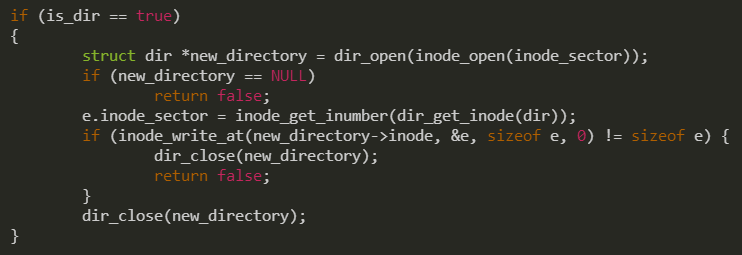


이는 alloc\_inode\_indirect와 유사하지만 차이점은 allocate\_block함수대신에 alloc\_inode\_indirect함수를 통해서 indirect block을 할당한다는 점이다. alloc\_inode\_indirect함수는 다시 allocate\_block을 실행할 것이다.

* Subdirectory



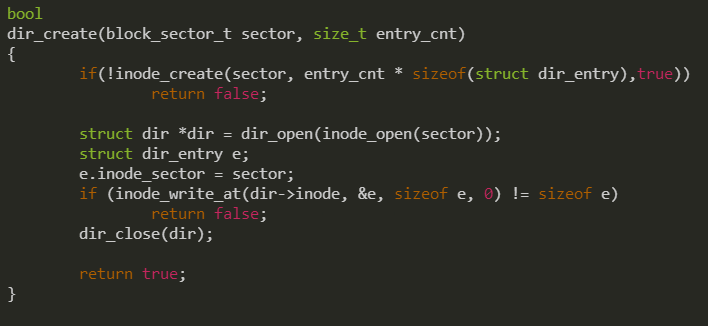
directory.c의 dir\_add의 입력에 bool is\_dir를 추가해서 입력하는 entry가 directory인지 여부를 파악한다. 그리고 dir\_add에 다음과 같은 코드를 추가한다.



이는 name을 입력하려는 entry가 directory인 경우, new\_directory에 dir\_open을 통해서 해당 directory를 얻고, 부모 directory의 inumber를 얻어서 저장한 후에 inode\_write\_at을 통해서 new\_directory의 첫번째 entry에 저장한다. 이는 해당 directory파일의 첫번째 entry에 자신의 부모 directory를 넣는 과정이다. 이것을 통해서 자신의 부모 디렉토리에 접근할 수 있다.



dir\_lookup에 위와 같이 strcmp(name,”.”)과 strcmp(name,”..”)부분을 추가한다. 이는 directory를 open할 때 “.” 또는 “..”가 입력되면 이를 처리하기 위함이다. “.”가 입력되면 가지 자신을 그대로 반환한다. “..”이 입력되면 inode의 첫번째 entry를 읽어서 이에 해당하는 sector를 입력으로 inode를 얻어 이를 반환한다.



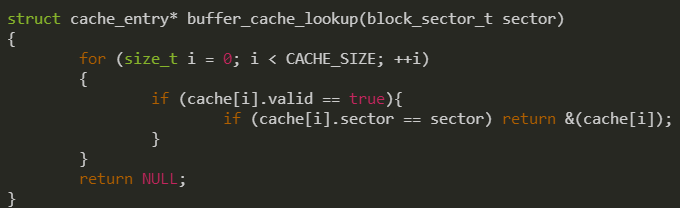
dir\_create함수는 dir\_add에서 부모 디렉토리를 첫번째 entry로 저장하기 전에 실행되는데, 그때 자기 자신의 sector를 첫번째 entry에 스스로 저장하고 있는 과정을 추가했다.

* Buffer cache  
  **구현한 경우만 작성**

텍스트이(가) 표시된 사진

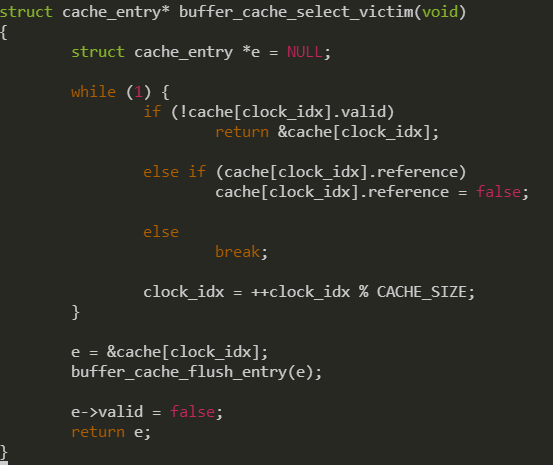
자동 생성된 설명

cache.c의 buffer\_cache\_read와 buffer\_cache\_write은 위와 같다. 우선 disk의 sector에 해당하는 값을 통해서 cache의 entry를 찾는다 이때 buffer\_cache\_lookup함수가 사용된다. 이 함수는 다음과 같다.

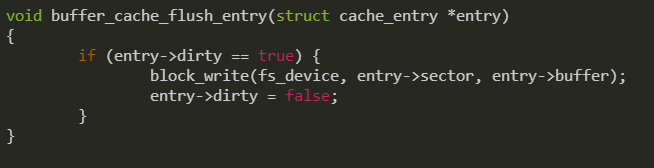


모든 cache를 확인하면서 sector와 같은 값을 가진 sector를 가진 cache를 발견하면 이를 반환한다.

그후 buffer\_cache\_read와 buffer\_cachce\_write은 cache를 찾지 못한 경우에 buffer\_cache\_select\_victim을 통해서 cache를 하나 할당한다. 이 함수는 다음과 같다.



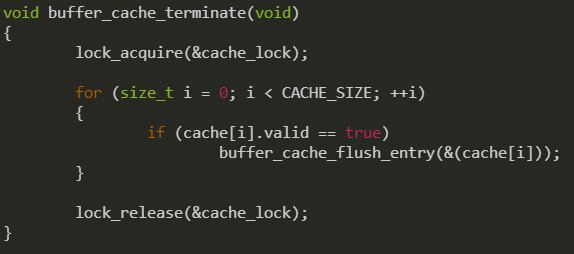
clock algorithm을 통해서 cache의 모든 entry들을 인덱스 순으로 탐색하면서 valid bit가 false라면 그냥 반환하고, 그렇지 않으면 reference bit를 체크한다. reference bit가 true면 false로 변경한다. false라면 이 entry를 선택한다. 선택한 entry를 반환하기 전에 buffer\_cache\_flush\_entry함수를 통해서 flush를 한 후에 해당 캐시의 valid bit를 false로 바꾸고 반환한다. buffer\_cache\_flush\_entry는 다음과 같다.



입력된 cache entry에 write된 적이 있는지를 dirty bit를 통해서 확인하고 있다면 block\_write으로 disk에 해당 cache entry내의 내용을 입력하고 이 entry의 dirty bit를 false로 변경한다.

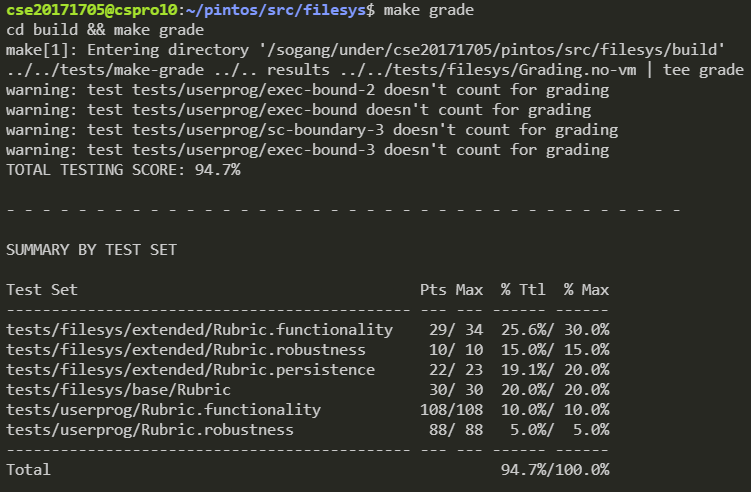
buffer\_cache\_read와 buffer\_cache\_write에서 이와 같이 새로운 cache entry를 evict해서 얻으면, cache와 대응되는 disk의 sector에서 read해 이 cache entry에 넣는다.

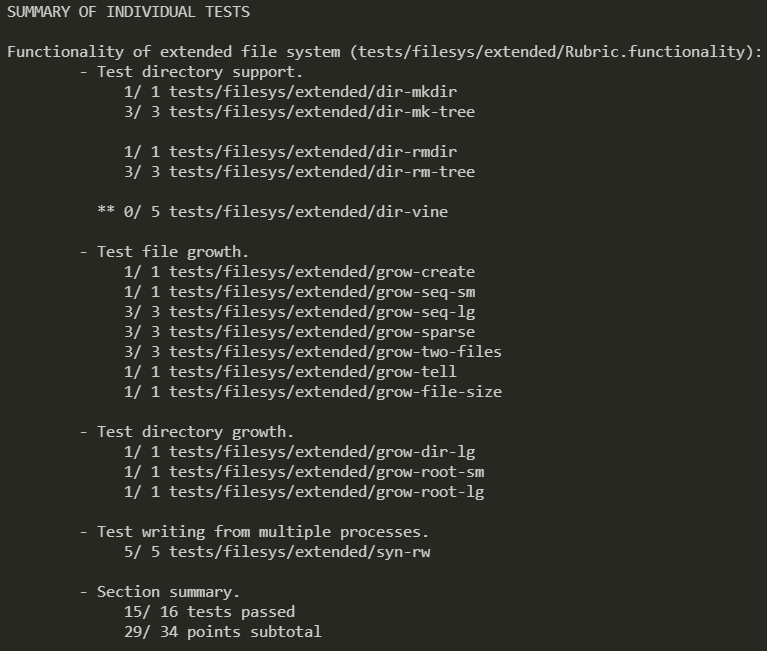
cache entry를 새로 할당받거나, find했으면 buffer\_cache\_read의 경우에는 memcpy를 통해서 buffer에 값을 read하고, buffer\_cache\_write의 경우에는 buffer에 있는 값을 이 캐시에 write해준다. 두 함수 모두 reference bit를 set해준다. 그리고 buffer\_cache\_write에서는 dirty bit를 set한다.

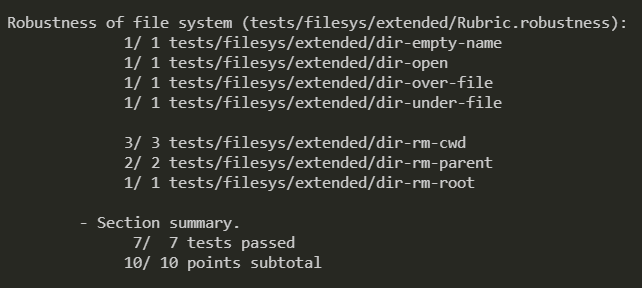


filesys.c의 filesys\_done함수에 buffer\_cache\_terminate를 추가했고, 이를 통해서 종료되기 전에 buffer cache의 내용을 모두 disk에 flush해준다.

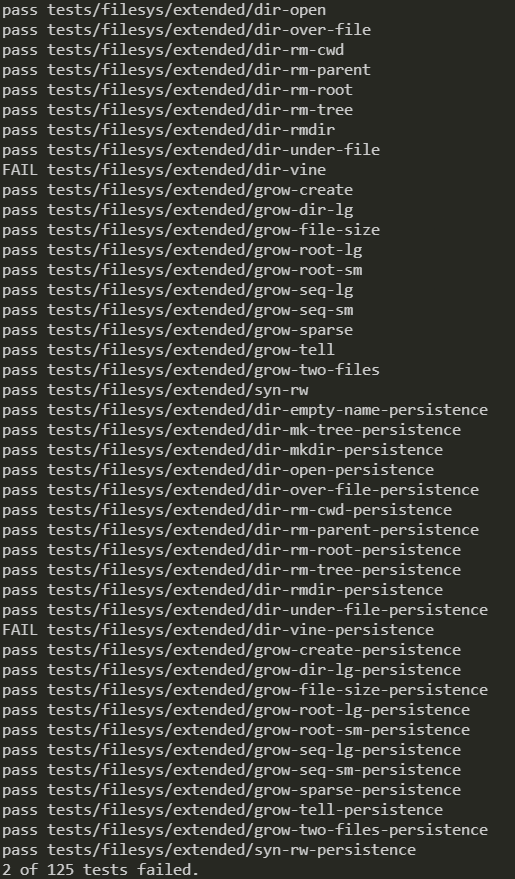
* 1. **시험 및 평가 내용**
* **Src/filesys make grade 수행결과를 캡처 하여 첨부.**

****

****

**텍스트이(가) 표시된 사진

자동 생성된 설명**

****