**Pintos Project 5: Filesystem**

담당 교수 : 김영재 교수님

조 / 조원 : 20171700 최재원

개발 기간 :12/18 ~12/22

1. **개발 목표**

현재 핀소트의 파일 구조는 할당된 크기에서 유동적으로 파일 크기를 키울 수 없다. 기존 할당된 크기보다 더 많은 내용의 파일을 쓰더라도 파일의 크기를 키울 수 있도록 extensible file을 구현한다. 다음으로 subdirectory를 구현하여, multi level directory를 구현함으로써, 현재 핀토스의 root 경로 외의 하위 경로를 추가하도록 한다. 위의 과정들에서 Synchronization을 적절히 유지시켜줘야 하며, Buffer cache를 활용하여 block\_read와 같은 함수를 대신함으로써 자주 사용하는 physical data에 대해서는 caching을 구현한다.

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

현재 핀토스의 디스크 공간은 512 bytes의 sector들로 나누어져 있다. (BLOCK\_SECTOR\_SIZE 로 define 되어있다.) 이 공간들에 대해서 free map을 사용하여 consecutive한 sector들을 관리하다보니, 예를 들어 800 bytes의 data를 write 한다고 했을 때, 전체 공간의 합으로 치면 충분히 수행할 수 있음에도 연속적인 공간이 부족하여 수행할 수 없는 External Fragmentation이 발생한다. 따라서 extensible한 file, 그리고 file growth를 구현하기 위해서 Multi-level indexting을 활용하도록 한다. Inode table에 대해서 일정 부분은 direct로, 그보다 더 큰 파일에 대해서는 single direct 로, 그보다 더 큰 파일에 대해서는 doubly indirect 로 관리함으로써 작은 파일에 대한 latency를 줄이면서 큰 파일까지 관리 가능한 구조를 구현할 수 있게 된다. 그리고 가장 중요한 external fragmentaion이 사라지고, 연속적인 sector 할당을 할 필요가 사라지기 때문에 extensible file 과 file growth를 구현할 수 있게 된다.

1. Subdirectory

서브 디렉토리를 구현하기 위해서는 적절한 tokenizing을 통해서 path를 만들어주고, directory인지 file인지 구별할 필요가 있으며, open\_cnt 가 0이되면 해당 inode를 소거해주고 disk에 write back 해주어야한다. 이런 일괄적인 과정을 userprog의 syscall 및 filesys의 directory.c , filesys.c, cache.c, inode.c 등에서 구현하게 된다. 또한 '.' 과 '..'를 기본적으로 모든 디렉토리에 추가해주며, mkdir, rmdir, chdir 등과 같은 system call도 구현해줌으로써 정상적으로 subdirectory에 대한 기능을 수행할 수 있을 것이다.

1. Buffer cache

buffer cache는 강의 시간에 배운 Track buffer에 해당하는데, write-back 또는 write-through로 구현할 수 있을것이다. 이번 프로젝트에서는 modification이 있을 때 cache에만 기록하고, 해당 data(block)과 연결된 inode가 소멸될 때, 즉 다시 block에 write 해줄 때 일괄적으로 dirty bit이 set된 block들에 대해서 sector에 저장해주는 write back policy를 사용할 것이다. 또한 buffer cache 내에서는 clock algorithm을 통해서 LRU 정책과 비슷한 수행을 해줄 것이며, 이런한 정책 및 알고리즘에 의해서 data persistency와 효율적인 I/O time을 모두 구현할 수 있을 것이다.

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

앞에서 언급한대로, index structure는 multi-level indexing을 활용하도록 한다. Direct, indirect, 그리고 doubly indirect block 까지 관리하도록 함으로써, 작은 파일에 대해서는 direct에서 접근해서 상대적으로 적은 latency에 수행할 수 있도록 하고, 큰 파일에 대해서는 indirect ~ doubly indirect block을 사용하여 관리하여 효율적인 indexing을 하도록 한다.

* Subdirectory
  + Directory entry 관리 방법

Subdirectory에서는 mkdir, chdir, rmdir 등의 system call들과 Root 디렉토리 이외의 하위 디렉토리를 구현할 수 있도록 적절한 tokenizing을 통한 path를 설정 및 direcotory open 및 close를 작성한다. 이 때 file과 directory 모두 하나의 file, 즉 inode(block)를 가지고 있기 때문에 이 둘에 대한 구분과 앞에서 말한 내용들을 활용함으로써 본 프로젝트에서 요구하는 subdirectory를 구현할 수 있을 것이다.

* Buffer cache
  + Buffer cache eviction 방식

Filesys 디렉토리의 cache.c에서 cache eviction이 필요한 경우 victim을 구하여 evict 시켜주는 함수를 구현하게 되는데, 이 때 clock algorithm을 사용하여 victim을 구하게 된다. Reference bit을 활용하여, 한번 접근한 block에 대해서는 다시 그 reference bit을 0으로 만들어 주고, reference bit 이 0이였을 경우, 즉 한번도 접근 안되었던 상황에 대해서는 바로 그 block을 flush하고 현재 필요한 data가 저장되어 있는 block을 buffer cache에 올리게 된다.

* + Buffer cache flush 방식

Buffer cache에 공간이 없어서 flush를 해야하는 경우에 flush\_entry 함수를 구현하고 해당 함수를 호출함으로써 Buffer cache flush를 수행한다. Buffer cache flush 함수에서는 buffer cache에 올라와 있던 해당 block의 dirty bit을 확인하여 1로 set 되어 있었을 경우 다시 block에 write 해줌으로써 data persistency를 유지시키고, 아닌 경우에는 현재 필요한 block data로 해당 index에 overwrite 함으로써 새로운 필요한 data를 buffer cache에 올리게 된다. 또한 buffer cache에 올라와 있는 block에 대해서 해당 inode가 open count가 0이 되어서 remove 되어야 할 경우 마찬가지로 cache flush 함수에서 dirty bit을 확인한 후, 1로 set 되어 있었다면 disk에 write 해주는 수행을 해준다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

|  |  |  |
| --- | --- | --- |
| Date | Main content | Refer |
| 12/18~19 | Buffer cache 구현 | Project5 내용 공부 |
| 12/19~12/21 | File growth 및 subdirectory구현 | Synchronization 추가 |
| 12/22 | Report 작성 |  |

* 1. **개발 방법**

b-1) Extensible file & file growth

* Index structure와 management

Filesys 디렉토리의 Inode.c에서 inode table에 관한 내용을 구현하게 된다. Inode\_disk에 대해서 block\_sector\_t 구조체 자료형의 direct\_blocks를 123개로 설정하고, indirect block, dobule indirect block을 추가한다. 그리고 is\_dir을 추가하여 해당 Inode가 directory인지 file인지 구분할 수 있도록 한다. 그리고 indirect block에 대해서 또다시 block\_sector\_t 구조체 자료형의 block 배열을 128개 설정하여 indirect block이 또 다른 inode를 point할 수 있도록 한다. 해당 block\_sector\_t 구조체 자료형은 device 디렉토리의 block.h에 정의되어 있다.이렇게 설정한 tree 형태의

b-2) Subdirectory

* Directory entry

multi indexing data structure inode table에 대해서 inode\_reserve 그리고 reserve\_indir 함수, inode\_delete 그리고 delete\_indir 함수 등을 활용하여 direct에 대해서는 level = 0, 그리고 indirect에 대해서는 level = 1, double indirect 에 대해서 level = 2로 구분하여 각 level에 맞는 inode table 할당및 해제를 해줄 수 있도록 한다. direct inode(block)들에 대해서는 for 반복문을 사용하여 할당, 해제 해주고, 나머지 level에 대해서는 recursive한 함수 indr\_reserve, indr\_delete 를 통해서 각 block을 할당해 주게 된다.

b-3) Buffer cache

* Buffer cache eviction

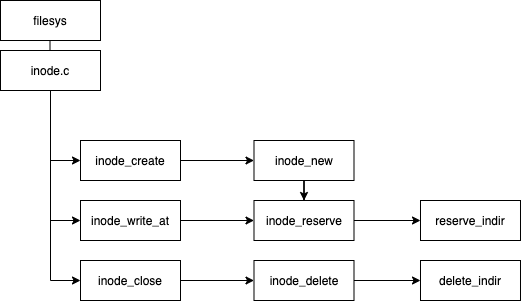
기본적으로 cache에 대한 구조체는 cache.h에 구현되어 있으며, 나머지 buffer cache의 기능과 관련된 함수들을 cache.c에 구현되어 있다. Eviction은 크게 buffer\_cache\_select\_victim 함수와 buffer\_cache\_flush\_entry 함수를 사용하여 이루어지게 된다. Victim 함수에서는 위에서 언급한 대로 clock 알고리즘을 사용하여 eviction 될 block을 설정하게 된다. Reference bit이 0인 block을 victim으로 설정하여 flush 함수를 호출함으로써 buffer cache의 eviction 기능이 구현이 되며, 해당 함수는 buffer\_cache\_write, buffer\_cache\_read 함수에서 buffer cache lookup 함수를 호출하여 cache에 원하는 Data가 없으면 victim을 선택하러 함수를 호출하게 되고, victim을 선정하는 함수에서 clock 알고리즘을 통해서 선정된 block의 dirty bit이 1로 set 되어 있는 경우 flush 함수를 호출하게 되는 것이다.

* Buffer cache flush

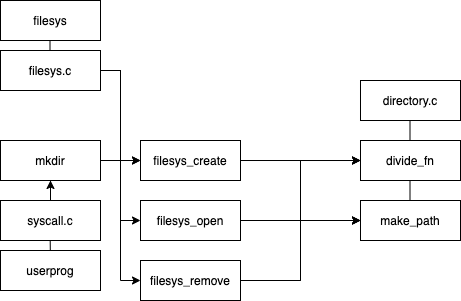
위에서 언급한대로 buffer\_cache\_select\_victim 함수에서 dirty bit이 set 된 block을 선택하게 되면 flush 함수를 호출하게 되는데, 이 때 해당 함수에서는 lock이 잡혀있는지를 확인하고, valid\_bit이 true였는지를 확인하게 된다. 그리고 dirty\_bit이 true인지 마지막으로 다시 한번 확인하고, block\_write 함수를 호출하여 다시 디스크로 data 내용을 옮겨주게 된다. 또한, flush\_entry 함수가 호출되는 다른 상황이 있는데, 이는 buffer\_cache\_terminate 함수가 호출되었을 때, 즉 inode가 remove 되는 상황이다. buffer\_cache\_terminate 함수는 filesys\_done 함수에서 호출된다. terminate 함수에서는 NUM\_CACHE로 정의된 64개의 index만큼을 traverse하면서 flush\_entry 함수를 호출하게된다.

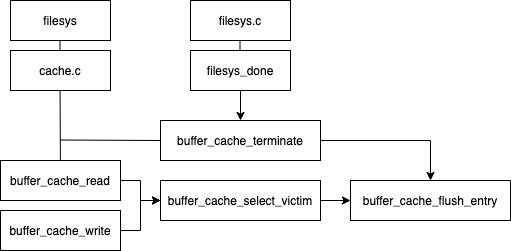
1. **연구 결과**
   1. **Flow Chart**

* Extensible file & file growth



* Subdirectory



* Buffer cache  
  
  1. **제작 내용**
* Extensible file & file growth

텍스트이(가) 표시된 사진

자동 생성된 설명

위와 같이 Inode\_disk 구조체 안의 구조체 변수들을 기존과 다르게 d\_indir\_block과 indir\_block 을 추가하여 indirect에 대한 pointer 변수를 추가해주었고, dir\_blocks[] 배열을 DIRECT로 define 한 123만큼의 크기로 설정하여 direct pointer 역할을 하는 block\_sector\_t 자료형 배열을 선언해 주었다. 다음으로 Indir\_indoe 구조체를 선언하고 그 안에 block 배열을 마찬가지로 선언해 주었고, 해당 구조체는 indirect block과 doubly indirect block이 모두 사용하게 된다. 또한 is\_dir 변수를 추가하여 directory에 대한 Inode 인지를 구분할 수 있도록 하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

위와 같이 inode\_reserve, reserve\_indir 함수를 사용하여 inode table의 block을 할당하게 된다. 오른쪽 함수를 먼저 보면, num\_sec 변수에 bytes\_to\_sectors 함수를 호출하여 sector 수를 할당받고, 해당 num\_sec 변수를 활용하여 indirect로 할당할지, direct로 할당할지 결정하게 된다. Diriect로 결정된 경우 for 반복문을 활용하여 해당반복문 안에서 free\_map\_allocate 함수 및 buffer\_cach\_write 함수를 호출함으로써 inode를 할당 및 buffer cache를 사용하게 된다. 이보다 큰 파일의 경우에는 reserve\_dir 함수를 호출한다. Reserve\_dir 함수는 왼쪽 코드이다. 해당 함수는 level로 1,2인 경우에 따라서 다르게 동작하며, 각각 indirect ,doubly indirect를 의미한다. 해당 함수에서 level이 0이 되는 경우는 recursive 하게 호출하여 indirect block에서 direct pointer를 설정하게 될 때이고, 이 때 free\_map\_allocate 함수 및 buffer\_cache\_write 함수를 호출하게 된다. for문을 활용하여 level 을 1씩 낮추며 recursive 하게 reserve\_dir함수를 호출하게 된다.

위의 텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명reserve에서의 메커니즘과 거의 비슷하게 delete를 수행하게 된다. inode\_delete 함수인 오른쪽 코드에서 이번에는 free\_map\_release 함수를 호출하여 direct로 할당한 block에 대해서 할당 해제를 진행시켜 주고, direct, indirect에 해당하는 상황에 대해서는 delete\_indir 함수를 호출하여 처리하게 된다. 마찬가지로 level에 따라서 그 파일의 크기에 따른 할당 해제 수순이 조금 달라지며, 공통적으로 for 문안에서 recursive 하게 delete\_indir 함수를 호출함으로써 할당해주었던 block들에 대해서 free\_map\_release 하게 되고, 이 때 buffer\_cache\_read 함수를 호출하고 또 그 함수에서 victim 함수를 호출하여 dirty bit인 것들에 대해서는 block에 block\_write 함수를 통해서 restore 해주게 된다.

* 텍스트이(가) 표시된 사진

  자동 생성된 설명Subdirectory

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명앞 서 언급한 내용과 flow chart에서 볼 수 있듯이 filesys.c의 filesys\_create, filesys\_open, filesys\_remove 함수에서 위의 divide\_fn 함수, make\_path 함수를 호출하게 된다. 그리고 userprog의 syscall.c 파일의 system call handler에서 이번 chdir, mkdir, rmdir 등에 대한 시스템 콜을 구현하는데 해당 시스템 콜중 mkdir 시스템 콜에서 filesys\_create를 호출하게 된다. 위의 코드를 보면 알 수 있듯이, 먼저 divide\_fn 함수에서는 주어진 path에서 directory 이름과 filename을 구분하여 tokenizing 해주게 된다. 절대경로와 상대경로에 따른 예외처리를 진행해주며, tokenizing 된 결과, 즉 파일 이름을 f\_token에 저장해주게 된다. 다음으로 make\_path 함수에서는 마찬가지로 절대경로, 상대경로에 대한 구분을 해주고, dir\_lookup 함수를 통해서 각 경로를 지정해주게 된다. 진행 과정에서 dir\_open 함수를 호출해주고, 예외 사항에 대해서는 다시 dir\_close를 진행해준다, 최종적으로 struct dir 구조체 포인터인 Dir 변수를 return 해주게 된다. 왼쪽은 위에서 언급하였던 system call들에 대한 코드이다.

* 텍스트이(가) 표시된 사진

  자동 생성된 설명Buffer cache  
  왼쪽은 cache.h에 작성된 buffer cache entry에 대한 구조체이다. 해당 구조체에는 pdf에서 조언해주신대로 각 meta data에 대한 변수를 추가해주었다. Buffer[] 배열에 block이 할당된다.이 때 block\_sector\_size는 pintos에서 정의된 512byte 이다.

아래는 buffer\_cache\_init 함수와 buffer\_cache\_select\_victim 함수에 대한 코드이다. Cache\_init 함수에서는 전체 cache 배열을 travers 하며, valid bit을 false로 초기화 해주는 것을 확인할 수 있다. 다음으로 buffer\_cache\_select\_victim 함수에서는 앞 서 언급했던 대로 clock algorithm을 사용하는 것을 확인할 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명Cindex는 clock index를 의미하며, 무한 loop 안에서 buffer cache entry 인 block들의 valid bit을 확인하여 false 인 경우에는 바로 return 하며, reference bit이 true인 경우 다시 false로 바꾸어줌으로써 second chance 를 수행하게 된다. Cindex는 최대 NUM\_CACHE, 즉 64까지 traverse하게 된다. 무한 loop을 빠져나온 후 dirty bit을 확인하여 victim으로 선정된 block을 cache에서 flush 한 후 disk 로 block wirte를 할지 결정하게 된다.

왼쪽은 buffer\_cache\_flush\_entry 함수와 terminate 함수이다. 호출되는 상황에 대해서는 앞서 개발 계획과 flow chart에서 설명한 바 있다. 코드를 살펴보면 dirty bit이 set 된 상황 및 예외상황에 대해서 다시 한번 확인해주고, 모두 정상인 상황에 대해서 block write 함수를 호출하여 disk에 다시 data를 작성해 줌으로써 data persistency를 유지함을 알 수 있다. 그리고 terminate 함수에서는 for 문안에서 flush\_entry 함수를 호출해가며 각 entry에 대해서 모두 flush out을 수행해주는 것을 확인할 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

마지막으로 buffer\_cache\_read 함수와 buffer\_cache\_write 함수에 대한 구현이다. 해당 두 함수는 filesys 내의 inode.c의 여러 함수에서 호출된다. 기존의 block\_read 함수 대신 buffer\_cache\_read 함수를 호출하고, 단순하게block\_wirte가 호출되었던 부분에 buffer\_cache\_write 함수를 호출하게 된다. 마찬가지로 단순하게 free\_map\_release를 해주었던 부분에서는 inode\_delete 함수를 호출하게 됨으로써 multi-indexing 방법으로 구현한 inode table에 대해서 적절한 조치를 취하게 된다. 앞 서 구현하였던 extensible file에 대한 구조 inode table이 buffer cache read, writer과도 긴밀한 연관이 있는 것이다. 이제 위의 코드를 살펴보자. 해당 두 함수에서는 기본적으로 buffer\_cache\_lookup 함수를 호출하여 현재 buffer cache에 접근하려는 data에 대한 block이 올라와 있는지 for 문을 사용하여 찾아보고, 있으면 해당 buffer\_entry를 return 하고, 없으면 NULL을 return 한다. NULL인 상황에 대해서는 아직 cache에 올라오지 않은 상황으로 판단하여 buffer\_cache\_select\_victim 함수를 호출하게 된다. 그리고 할당받은 buffer\_cache\_entry에 대해서 각 meta data인 valid bit ,dirty bit 등을 설정하게 되고, block\_read 를 사용하여 해당 영역에 disk로 부터 data를 읽어온다. 그리고 read, write에 대해 memcpy 를 사용하여 data를 옮겨주게 된다. 또한 lock 구조체 변수bc\_lock를 활용하여 synchronization을 구현해 주었다.

* 1. **시험 및 평가 내용**
* **텍스트이(가) 표시된 사진

  자동 생성된 설명Src/filesys make grade 수행결과를 캡처 하여 첨부.**

텍스트이(가) 표시된 사진

자동 생성된 설명

모든 test들에 통과했음을 확인하였다.