**Pintos Project 5: Filesystem**

담당 교수 : 김영재

조 / 조원 : 20171669 이재영

개발 기간 : 21.12.04 ~ 21.12.22

1. **개발 목표**

현재 Pintos의 File system에서의 N개의 block file이 N-block 들이 사용가능해도 할당을 하지 못하는 external fragment의 문제점과 파일크기가 fix되어있고 subdirectory가 없는 문제점을 해결하는 것과 효율적인 read/write를 위한 cache 구현이 프로젝트 5의 목표이다.

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

File의 size가 file create시 정해져서 변하지 못했던 것에서 file을 write 할 때마다 DISK BLOCK을 받아서 사용함으로써 file의 size가 고정되지 않고 growth가 가능하게 된다.

1. Subdirectory

기존의 Pintos에서는 하나의 root directory가 존재하는 단일계층으로 root directory안에 file을 생성할 수 있는데 subdirectory를 구현해서 root directory 내의 다른 directory에도 file을 생성할 수 있는 hierarchy한 구조가 만들어진다.

1. Buffer cache  
   비효율적으로 DISK에서 block을 read, write하는 것이 아니라 buffer cache를 구현해서 block의 정보를 미리 저장해서 DISK로의 비효율적인 탐색을 줄일 수 있다. 이 때, buffer cache안에 있는 block의 정보들은 cache size와 관련해서 저장하지 못할 수 있으므로 block을 evict 하는 algorithm을 구현한다.
   1. **개발 내용**

* Extensible file & file growth
  + Index structure와 management에 대해서 기술

기존의 DISK의 연속된 block을 할당한 것과는 달리 block의 위치에 대해서 direct, single indirect, double indirect의 방식으로 할당되도록 설정한다. direct 방식으로 구현이 가능한 경우 바로 block으로 접근하도록, single indirect 방식의 경우는 indirect block을 거친 후 data block으로, double indirect의 방식으로 접근 가능한 영역은 indirect block을 두 번 거친 후 data block에 접근이 되도록 구현한다.

* Subdirectory
  + Directory entry 관리 방법

기존의 pintos에서 directory에서는 하나만 존재했지만 subdirectory를 구현하기 위해 parent, child 관계로 directory를 관리할 수 있으며 ‘/’로 directory사이를 구분할 수 있으므로 system call함수와 directory함수에서 쉽게 directory의 path를 탐색해가면서 관리가 가능하다.

* Buffer cache
  + Buffer cache eviction 방식

Buffer cache를 eviction하는 방법으로 clock알고리즘을 사용한다. Buffer cache에 빈 공간이 있으면 그 공간을 사용하고 아니라면 second chance를 부여하고 다음 clock으로 넘어간다. 그렇게 clock 방식으로 search하다가 evict할 entry를 찾으면 그 slot을 evict하는 방식을 사용한다.

* + Buffer cache flush 방식

Buffer cache에서 clock 알고리즘 방식으로 evict될 victim entry을 찾았을 때, 만약 그 victim entry가 dirty인 경우 DISK로 block write를 하는 FLUSH 과정이 이루어진다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* 21.12.04 ~ 21.12.10 : Extensible file & file growth을 구현하기 위한 file system 영역 수정
* 21.12.10 ~ 21.12.15 : subdirectory 추가 및 구현, 전반적인 buffer cache위한 struct 구현

21.12.20 ~ 21.12.22 : buffer cache 구현 마무리 및 보고서 작성

* 1. **개발 방법**

<src/userprog/syscall.c>

* 새로 구현한 함수들

Bool sys\_chdir : directory를 change해주는 system call 함수

Bool sys\_readdir : directory를 read해주는 system call 함수로 directory에 inode가 있는지도 확인한다.

Bool sys\_isdir : directory가 존재하는지를 확인하는 system call함수

Int sys\_inumber : inode 정보를 get하는 system call함수

<src/userprog/process.c>

* 수정한 함수

void process\_exit : process를 exit할 시 file descriptor를 탐색하면서 열려 있는 것 fd의 경우 file\_close를 해주는 작업과 현재 thread의 file이 있는 경우 write 가능하도록 file\_allow\_write를 호출한 후 file\_close를 호출한다.

추가로 load함수에서 stack에 instruction을 set하고 file을 현재 thread의 executing file에 넣는 작업을 추가한다.

<src/filesys/filesys.c>

* 새로 구현한 함수

bool filesys\_chdir : directroy의 정보를 가져와서 현재 thread의 current directory를 close하고 새로 가져온 directory로 설정한다.

* 수정한 함수

void filesys\_init / void filesys\_done : 각각 file system을 초기화, close하는 함수로 새로 정의한 buffer\_cache를 초기화하는 buffer\_cache\_init함수와 buffer\_cache를 close하는 buffer\_cache\_close를 추가로 호출한다.

bool filesys\_create : 함수로 받아오는 parameter에서 directory가 존재하는지에 대한 Boolean type인 is\_dir변수를 받아온다. Directory와 file\_name을 나눠서 저장하는 split함수를 호출해서 directory에 대한 path를 open하는 작업을 수행한다.

struct file\* filesys\_open : 받아온 name에서 file name과 directory를 나눠서 저장하고 그 결과로 file name이 없다면 directory만 반환하고 아니면 directory에서 file에 대한 inode를 찾아보고 directory를 닫는 작업을 추가한다.

bool filesys\_remove : 받아온 name에서 file name과 directory를 나눠서 저장하고 directory의 path를 살펴본 후 성공여부를 return한다.

<src/filesys/directory.c>

* 새로 구현한 함수

int split\_path\_filename : path를 directory와 file name으로 나눠서 저장해주는 함수로 ‘/’를 기준으로 문자열을 나눠서 저장한다. 결과값으로 file의 name을 반환해준다.

struct dir\* dir\_open\_path : directory의 path를 open해주는 함수로

* 수정한 함수

bool dir\_create : directory를 만드는 것으로 기존과는 달리 indoe\_create에서 directory가 존재하는지를 확인하므로 is\_dir이 true와 함께 inode\_create함수를 호출한다. 후에 directory를 parent directory를 self-referencing하는 작업을 수행하는 함수이다.

bool dir\_lookup : directory를 살펴보는 함수로 기존에는 directory가 하나만 존재했는데 새로 subdirectory를 만들었기 때문에 parent, current directory에 대한 실행도 추가한다. 현재 directory라면 inode\_reopen, parent directory인 경우는 inode를 read하고 open하는 작업을 추가한다.

bool dir\_add : file이 directory인 경우는 child directroy의 정보를 update하는 작업을 수행한다.

bool dir\_remove : directory를 remove할 시 empty하지 않은 directory를 remove하는 작업을 방지하는 작업을 추가한다. 이는 inode.c에서 directory에 inode가 있는 경우를 return해주는 indoe\_is\_directory를 선언해서 true인 경우에 대해서 작업을 추가한다.

bool dir\_readdir : 다음 directory entry를 read하는 함수로 새로운 subdirectory가 있는 경우도 있기 때문에 inode를 directory의 entry의 size까지 읽는 과정을 추가한다.

<src/filesys/inode.c>

* 수정/추가한 자료구조

struct inode\_disk : on-disk inode에 관한 정보로 direct, single indirect, double indirect 방법으로 block을 관리하기 위해서 각각의 block\_sector\_t에 대한 table을 새로 추가한다.

struct inode\_indirect\_block\_sector : 각각의 block들의 indirect block에 해당하는 구조체로 block table을 추가로 선언한다.

* 새로 구현한 함수

Block\_sector\_t index\_to\_sector : index단위로 sector를 설정해서 direct, single indirect인지 double indirect 방법으로 할 것인지를 설정한다.

static bool inode\_reserve : inode bit를 설정할 수 있는 곳에 inode를 reserve하는 함수이다.

bool inode\_reserve\_indirect : indirect block에서 inode를 reserve하는 함수이다.

bool inode\_deallocate : inode 설정한 것을 deallocate해주는 함수로 재사용이 가능하게 해주는 함수이다.

void inode\_deallocate\_indirect : indirect block에서 inode를 설정한 것을 deallocate해주면서 다시 사용이 가능하도록 해주는 함수이다.

* 수정한 함수

기존에 block을 직접 읽는 block\_read함수를 호출한 것을 block대신 cache를 읽도록buffer\_cache\_read함수를 호출하는 것으로 수정한다.

block\_sector\_t byte\_to\_sector : 기존의 구조체와는 달리 index의 크기에 따라 block의 접근 방식이 다르기 때문에 BLOCK\_SCETOR size로 index를 나누어서 index의 크기에 따른 접근방식으로 block에 접근한 값을 return한다.

bool inode\_create : inode를 crate해주는 함수이다.

Struct inode\* inode\_open : inode의 정보를 알 수 있도록 open해주는 함수이다.

<src/filesys/buffer\_cache.c>

* 새로 구현한 함수

void buffer\_cache\_init : cache를 초기화하는 함수로 valid bit를 false로 초기화한다.

void buffer\_cache\_flush : cache에 있는 entry에 대해서 없애주는 flush과정을 수행해주는 함수이다.

void buffer\_cache\_flush\_all : cache 종료시 NUM\_CACHE만큼 반복문을 돌면서 Valid bit가 true인 경우 flush를 진행한다.

void buffer\_cache\_close : cache를 종료하는 함수로 cache에 있는 entry들 전부를 flush하기 위해 buffer\_cache\_flush\_all을 호출한다.

struct buffer\_cache\_entry\* buffer\_cache\_lookup : cache entry를 탐색하는 함수로 cache hit라면 그 cache entry의 주소를 return해주고 cache miss가 발생한 경우에는 NULL값을 return한다.

struct buffer\_cache\_entry\* buffer\_cache\_select\_victim : clock algorithm을 통해 evict될 entry를 설정하는 함수이다.

void buffer\_cache\_read : cache의 내용을 읽는 함수로 cache의 data를 memory로 copy하는 함수이다.

void buffer\_cache\_write : cache에 내용을 write하는 함수로 memory에서 data를 cache에 copy해주는 함수이다.

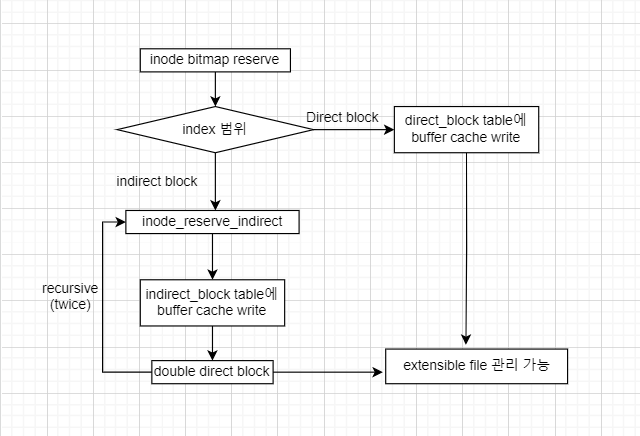
* 추가한 자료구조

struct buffer\_cache\_entry : buffer에 있는 cache entry에 대한 정보를 가지는 구조체를 추가한다. Entry의 내용으로는 dirty, occupied, buffer, access, disk\_sector가 있다.

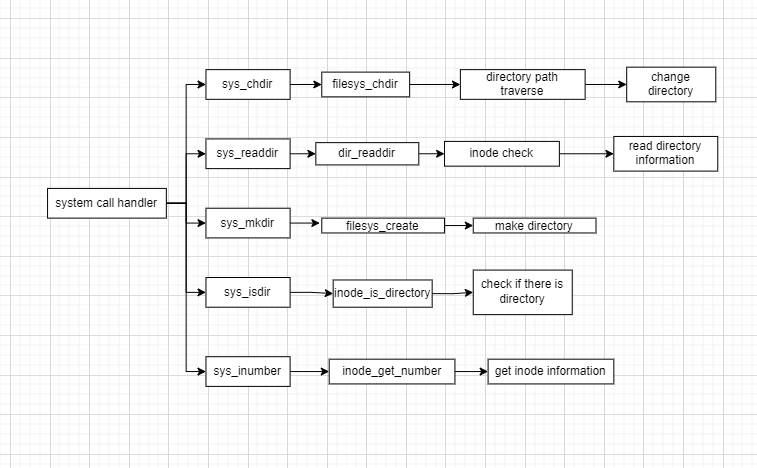
struct buffer\_cache\_entry cache[NUM\_CACHE] : cache entry로 사용되는 전역 배열이다.

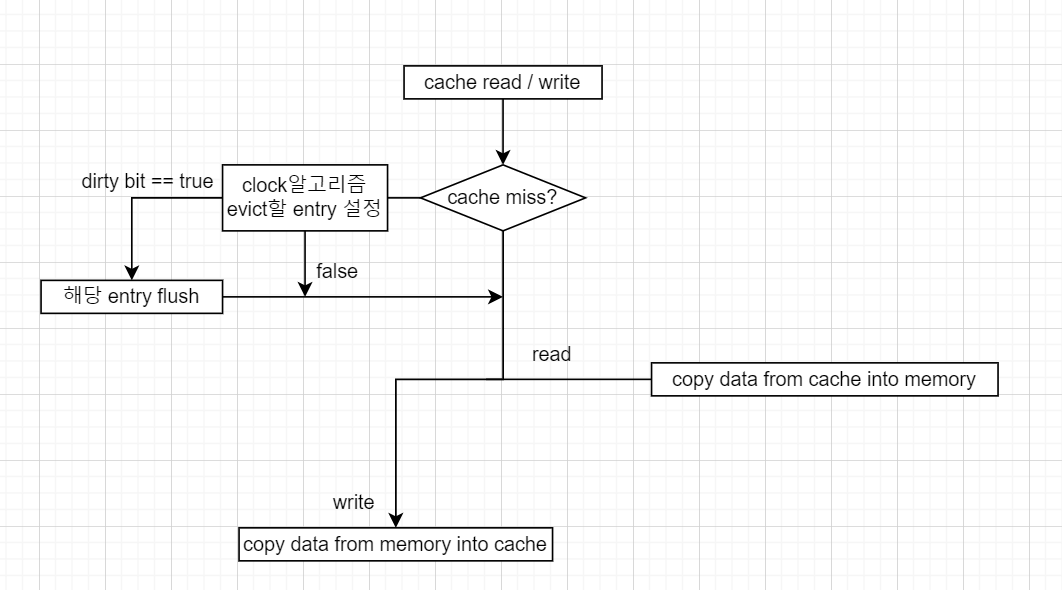
1. **연구 결과**
   1. **Flow Chart**

* Extensible file & file growth



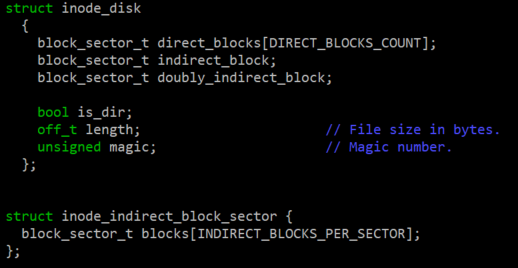
* Subdirectory



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

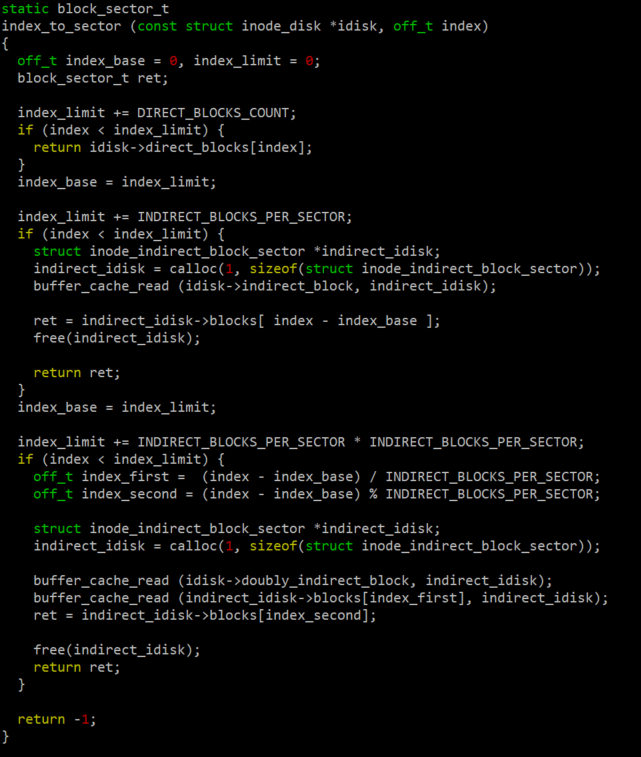
<src/filesys/inode.c>

* + Block을 직접 read, write하는 block\_read, block\_write함수를 호출하는 것을 buffer\_cache에서 구현한 buffer\_cache\_read, buffer\_cache\_write함수 호출로 교체한다.
  + inode\_disk/inode\_indirect\_block\_sector 구조체



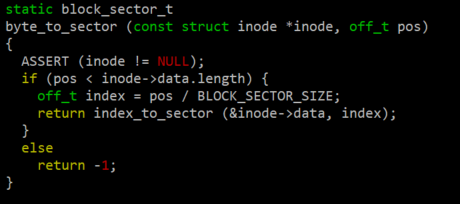
기존의 구조체에서 block manage하는 방법으로 direct\_blocks table, indirect\_block, doubly\_indirect\_block을 선언해서 관리한다. indirect\_block으로는 새로 구조체를 정의해서 추가로 관리한다.

* + index\_to\_sector



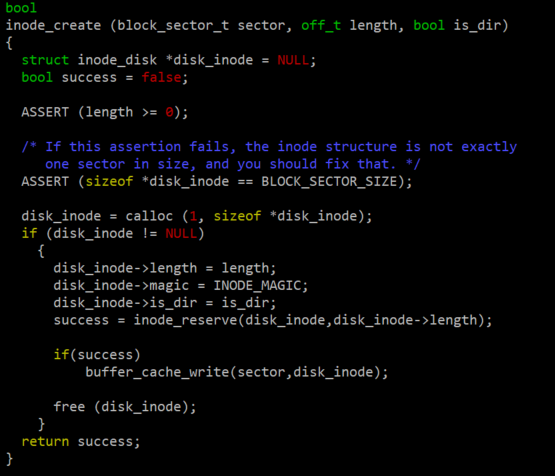
disk를 받아와서 index의 범위를 check해주면서 direct block으로 사용이 가능한 경우는 direct\_block table을 return해주고 single indirect의 범위인 경우는 indirect 구조체를 새로 할당해서 buffer에서 read한 후 해당 index에 해당되는 indirect block table을 return해주고 double indirect를 써야하는 index의 경우에는 2개의 block index에 대해서 block과 doubley\_indirect block table를 사용하면서 각각 buffer에서 읽어들이면서 두번째에 해당되는 index의 indirect block table을 return해준다.

* + Byte\_to\_sector



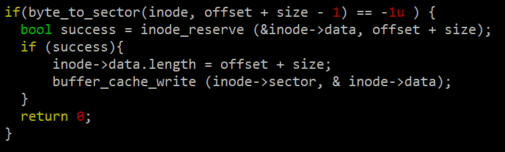
Block에 대해서 sector\_size로 나눈 값을 index로 사용해서 index의 범위에 따라 direct\_block, single indirect, double indirect로 관리할지에 대한 정보를 index\_to\_sector 함수를 호출해 return받은 값을 return해준다.

* + inode\_create



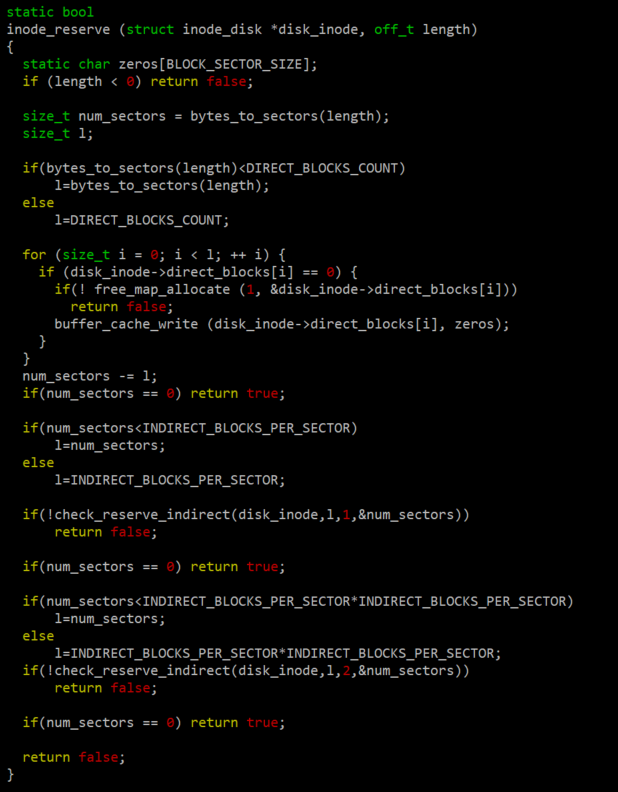
inode를 생성할 때, inode를 설정하는 inode\_reserve 함수를 호출해서 정상적으로 reserve된 경우는 buffer cache에 write를 수행하고 해당 inode를 free해준다.

* + inode\_write\_at(추가된 구현 내용)



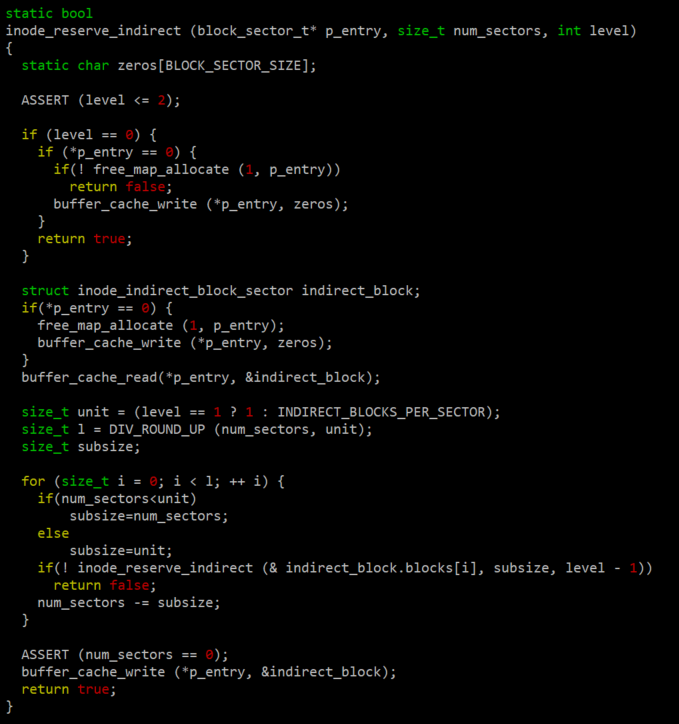
Inode에 buffer 정보를 write하는 함수로 file의 size가 extend가 필요한 경우, 즉 EOF를 벗어난 경우에는 file을 extend하는 작업을 추가로 수행한다. Offset과 file를 더한 값으로 inode를 reserve해서 제대로 reserve가 되었다면 extend된 file의 size를 inode의 데이터의 length로 설정하고 buffer cache에 write하는 작업을 추가로 구현한다.

* + Inode\_reserve



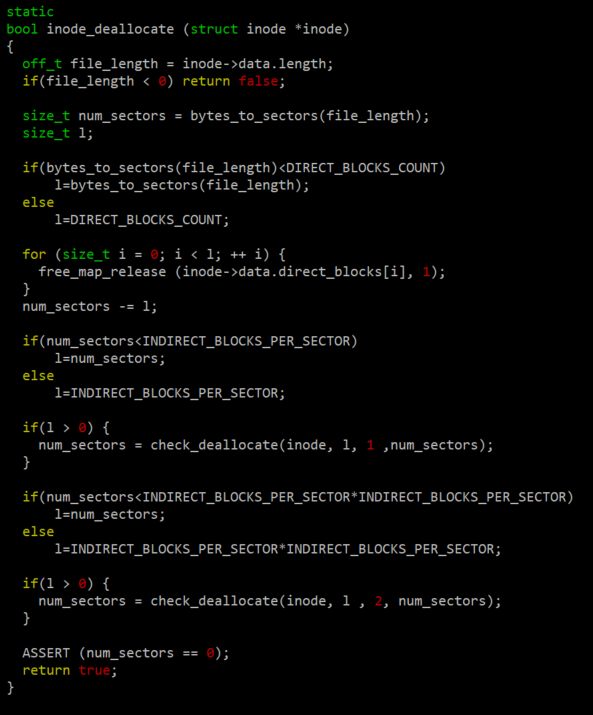
Direct block에 해당되는 경우에는 inode의 direct block index를 살펴보면서 사용할 수 있는 곳에 inode bitmap을 reserve해준다. Single indirect block의 index에 해당되는 경우에는 inode\_reserve\_indirect를 호출해서 indirect block에 대해서 inode를 설정해준다. Double indirect index에 해당되는 경우 또한 inode\_reserve\_indirect를 호출해서 inode를 설정해주는데 double이기 때문에 level을 2로 넘겨준다.

* + Inode\_reserve\_indirect



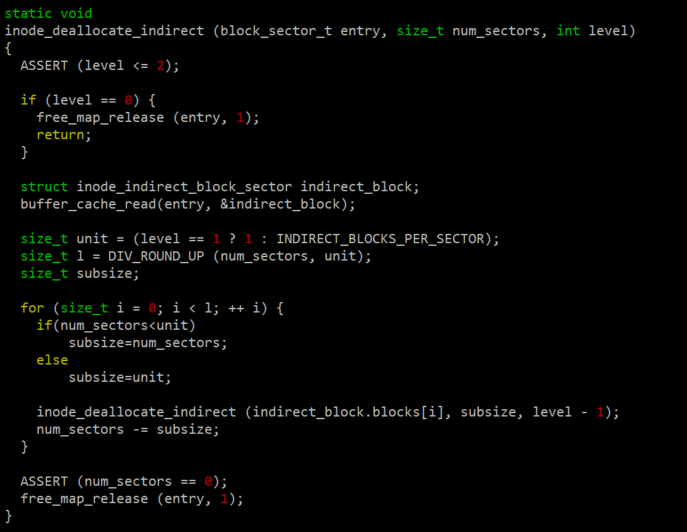
Indirect block에 대해서 inode를 reserve해주는 함수로 double인 경우는 2번, single인 경우는 한번 함수를 recursive방식으로 inode bitmap을 설정해준다.

* + Inode\_deallocate



이미 할당된 direct 또는 indirect block에 reserve되어 있는 inode bitmap에 대해서 free\_map\_release를 해주는 함수로 reserve함수와 마찬가지로 index의 범위에 따라 direct인 경우는 바로 free\_map, single, double indirect block을 사용한 경우에는 각각, 1,2번 inode\_deallocate\_indirect 함수를 호출해서 reserve된 inode bitmap에 대해 free과정을 수행해준다.

* + Inode\_deallocate\_indirect

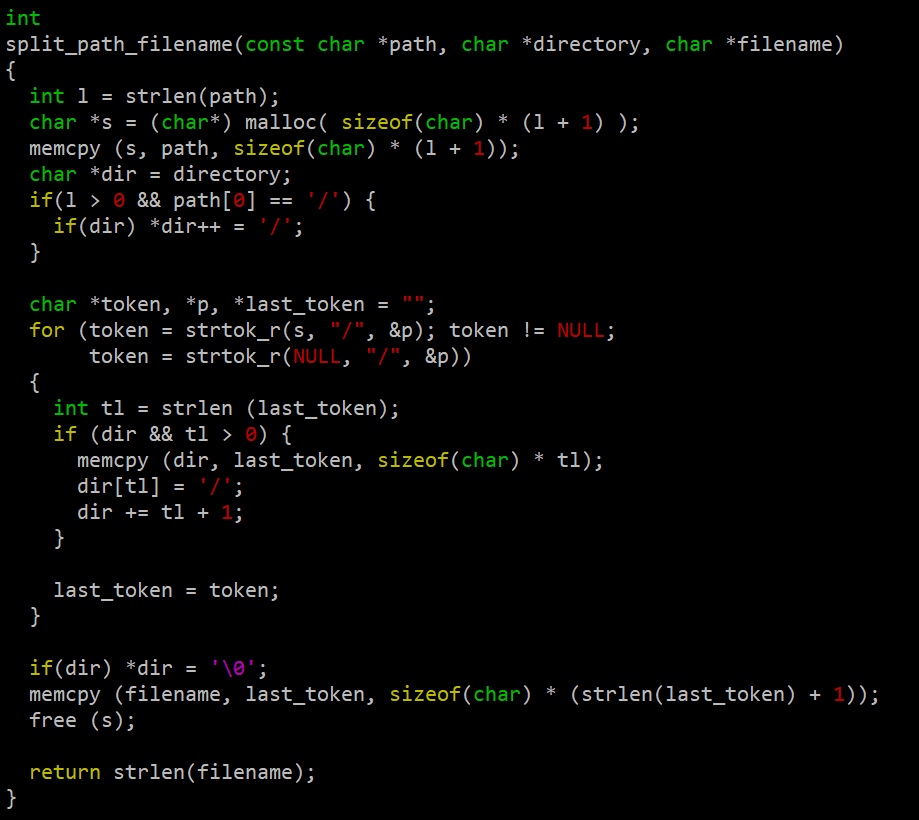


single, double indirect block을 사용한 경우에는 각각, 1,2번 inode\_deallocate\_indirect 함수를 호출해서 reserve된 inode bitmap에 대해 free과정을 수행해준다.

* + Subdirectory

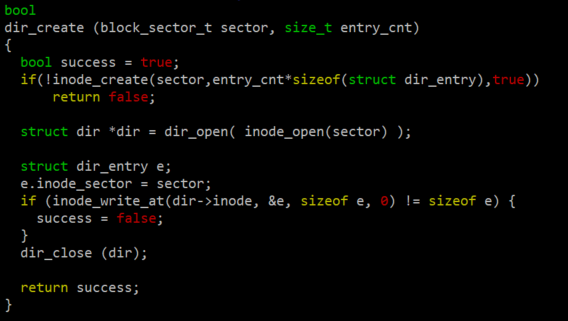
<src/filesys/directory.c>

* + int split\_path\_filename



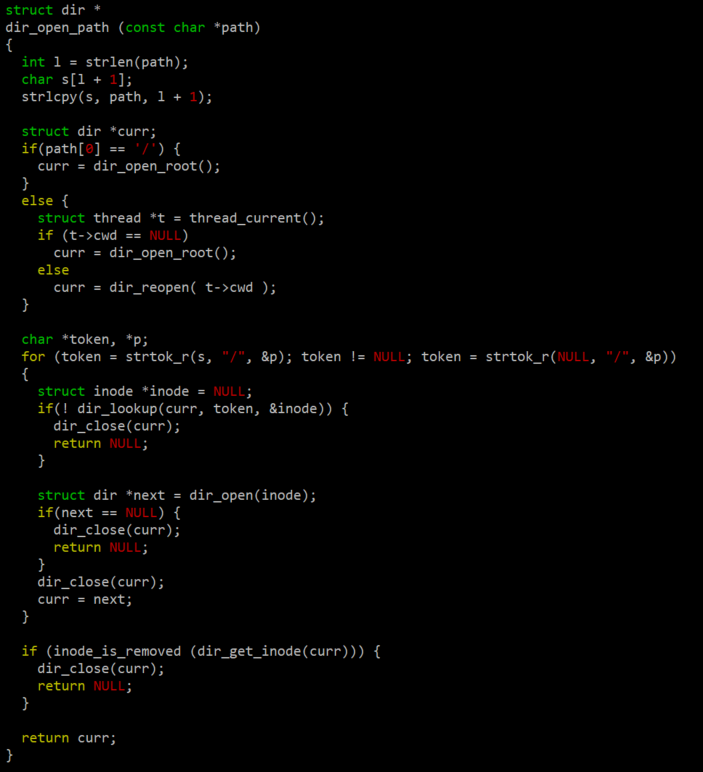
전체 경로의 instruction을 directory와 file로 나누어서 저장하는 함수로 ‘/’을 기준으로 자르고 path의 맨 마지막에 있는 것이 file의 name이기 때문에 그 부분을 file로 저장하고 나머지는 directory에 저장한다. 최종적으로 file string의 길이를 return한다.

* + dir\_create



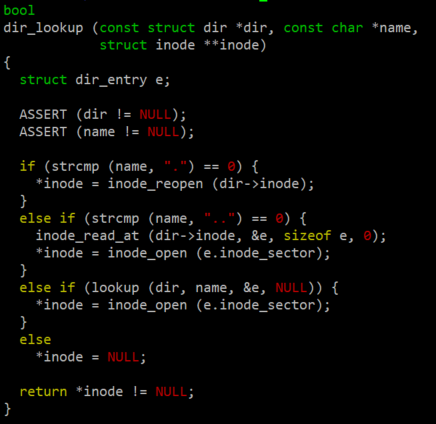
directory를 만드는 것으로 기존과는 달리 indoe\_create에서 directory가 존재하는지를 확인하므로 is\_dir이 true와 함께 inode\_create함수를 호출한다. 후에 directory를 parent directory를 self-referencing하는 작업을 수행한다.

* + dir\_open\_path



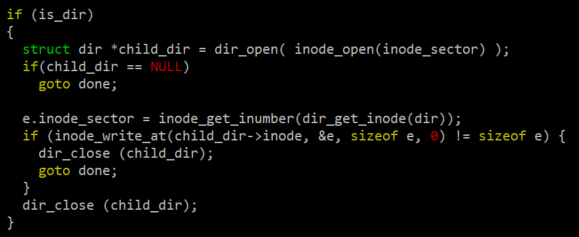
기존 pintos와는 달리 여러 subdirectory가 존재할 수 있기 때문에 path에 존재하는 모든 directory를 open하는 과정을 수행한다. ‘/’를 기준으로 directory에 해당되는 부분들을 root인 경우 dir\_open\_root, 아니라면 현재 thread의 cwd에 따라 dir\_reopen 혹은 dir\_open\_root를 수행한다. 이 경우는 main함수와 같은 경우에 대한 예외처리에 해당한다. 후에 path들을 search하면서 dir\_open과정을 수행한다. 마지막으로 remove된 directory를 open하는 경우를 방지하는 경우에 대한 예외처리를 수행하고 open한 directory를 return해준다.

* + dir\_lookup



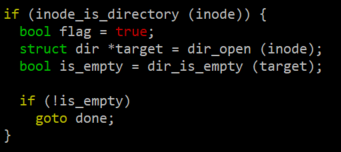
Directory를 search하는 함수로 기존과는 달리 subdirectory가 존재하므로 current directory ‘.’, patent directory ‘..’인 경우에 대해 current directory인 경우에는 inode를 다시 open하는 reopen을 수행하고 parent이 경우는 first entry에 저장이 되어있으므로 그 inode에 대해 open과정을 수행한다.

* + dir\_add(추가된 내용)



Directory를 추가하는 함수로 directory가 존재하는 경우에는 child\_directory에 대해서 정보를 update하는 과정을 추가한다.

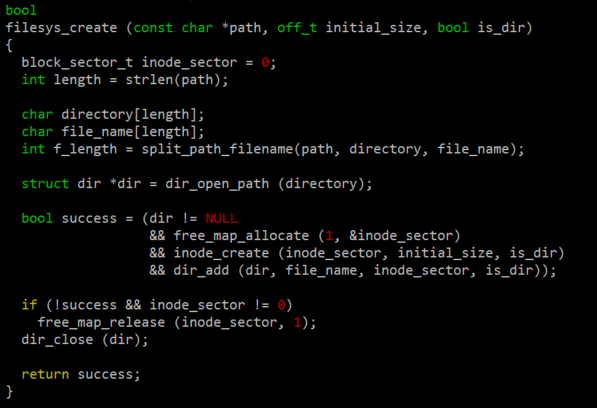
* + dir\_remove(추가된 내용)



Directory가 여러 subdirectory가 존재하게 되면서 directory에 정보가 있는 directory에 대해서 remove되는 것을 방지하기 위한 code를 새로 추가한다.

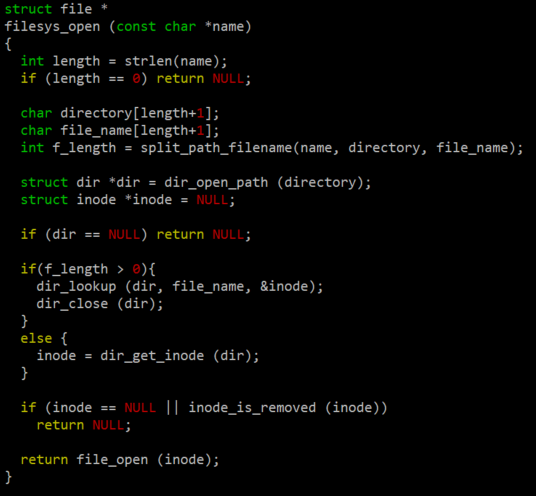
<src/filesys/filesys.c>

* + filesys\_create



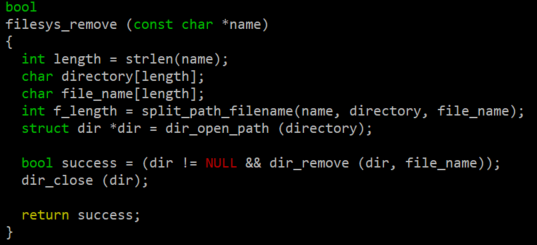
directory가 존재하는지 나타내는 Boolean type인 is\_dir을 새로운 parameter로 추가한다. Path에서 file name을 split해서 해당 결과로 directory에 추가하는 과정을 추가한다.

* + Filesys\_open



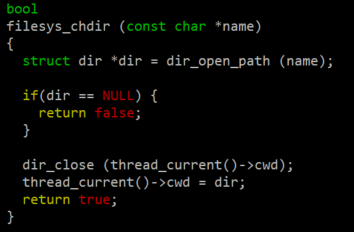
name에서 file name을 split하는 함수를 호출해서 file이 존재하는 경우 directory를 lookup하고 해당 directory를 close하고 file이 존재하지 않으면 directory를 단순히 return한다.

* + Filesys\_remove



File name과 directory를 split한 후 가져온 directory정보로 directory를 open한다.

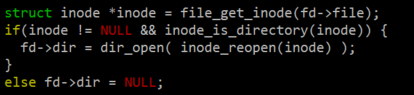
* + Filesys\_chdir



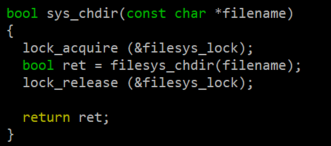
Directory를 open한 후 현재 thread의 directory를 close하고 새로 open한 directory를 현재 thread의 directory로 설정한다.

<src/userprog/syscall.c

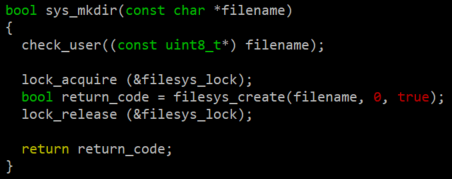
* + sys\_open/ sys\_chdir / sys\_mkdir / sys\_readdir / sys\_isdir / sys\_inumber



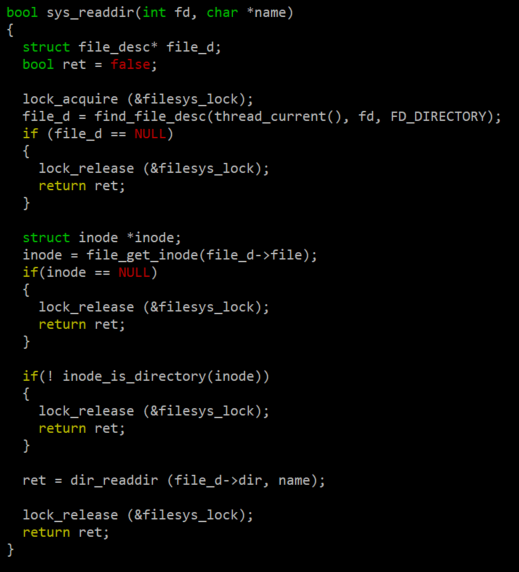
System call 함수 중 open에서 Inode정보를 가져와서 inode가 directory에 있는지를 확인해서 dir\_open을 수행하고 아니라면 directory를 NULL로 설정하는 것을 추가한다.



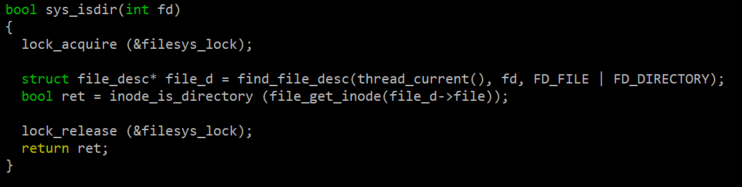
directory의 change를 수행하는 함수로 filesys\_chdir을 수행한다.



Directory를 make해주는 system call 함수로 filesys\_create를 호출하면서 수행한다.



Directory를 read하는 함수로 inode를 get해서 정상적으로 수행이 가능하다면 dir\_readdir를 수행하는 system call 함수이다.



Directory가 있는지를 확인하는 system call 함수로 inode.c에 새로 추가한함수 inode\_is\_directory 를 수행한 값을 return한다.

텍스트이(가) 표시된 사진

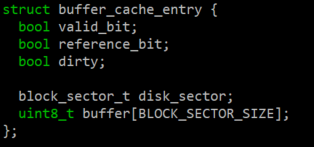
자동 생성된 설명

inode정보를 get하는 함수로 inode\_get\_inumber함수를 호출할 때 file\_get\_inode를 인자로 호출한다.

* Buffer cache

<src/filesys/buffer\_cache.c>

* + Buffer\_cache\_entry 구조체



Valid, dirty, reference bit 정보와 disk\_sector, buffer의 정보를 가지고 있는 cache entry 구조체이다.

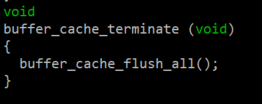
* + buffer\_cache\_init

텍스트이(가) 표시된 사진

자동 생성된 설명

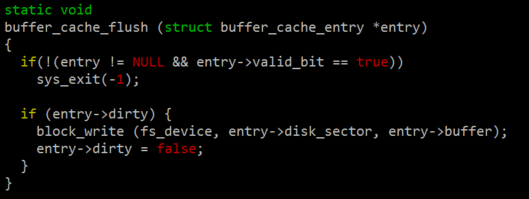
Cache 동작의 synchronization을 위한 lock인 buffer\_cache\_lock을 초기화하고 전역변수로 선언한 cache를 NUM\_CACHE만큼 valid\_bit을 false로 초기화한다.

* + buffer\_cache\_terminate



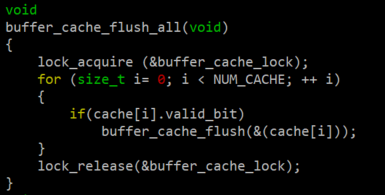
Buffer cache를 종료하는 함수로 cache안의 모든 entry를 flush하는 buffer\_cache\_flush\_all 함수를 호출한다.

* + Buffer\_cache\_flush



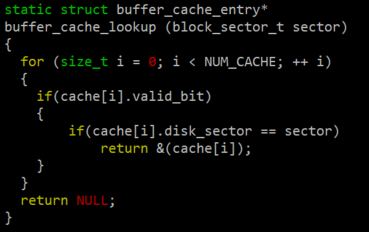
Cache entry의 dirty bit가 true라면 disk device에 block write를 수행하면서 flush를 진행하고 dirty bit를 false로 바꾼다.

* + Buffer\_cache\_flush\_all



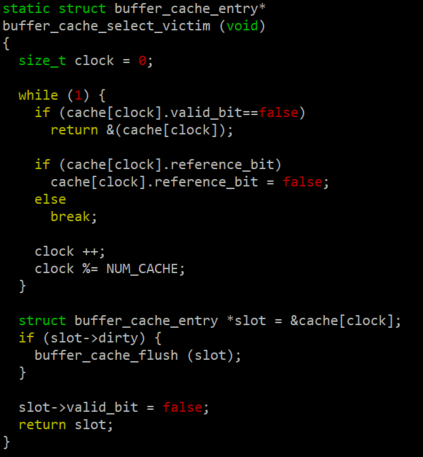
cache에 존재하는 entry를 전부 flush하는 함수로 NUM\_CACHE만큼 반복문을 돌면서 cache에 entry가 존재한다는 valid\_bit가 true이면 buffer\_cache\_flush 함수를 호출하면서 존재하는 모든 entry를 flush한다.

* + buffer\_cache\_lookup



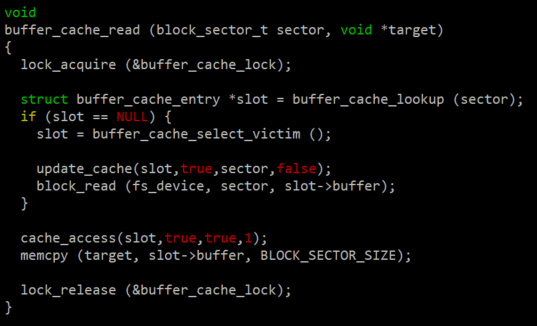
Cache 배열을 NUM\_CACHE 만큼 탐색해서 cache 배열의 valid\_bit가 true인 경우 cache에 저장되어있는 disk sector이 찾고자하는 sector 정보와 같다면 cache hit이기 때문에 해당 index의 cache 배열의 주소를 return해주고 반복문을 반복하는 동안 같은 정보의 sector를 못 찾은 경우는 cache miss이기 때문에 NULL을 return해준다.

* + buffer\_cache\_select\_victim



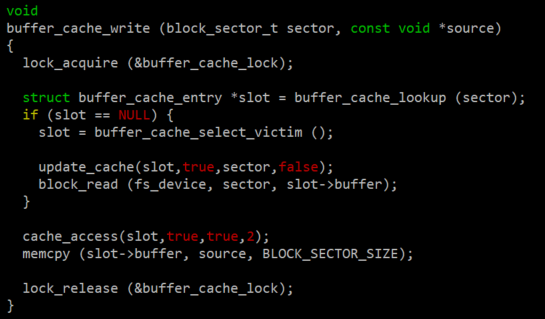
Clock algorithm을 통해 evict할 victim entry를 정하는 함수로 empty slot인 경우에는 해당 index의 cache를 사용하면 되므로 주소를 return해주고 reference bit가 1인 경우, 즉 한번 reference된 경우는 second chance를 위해 false로 bit를 바꾸고 clock 알고리즘을 진행한다. Reference되지 않은 경우는 victim이 되므로 disk에 write back 과정을 수행하고 valid\_bit를 false로 바꿔서 사용할 수 있도록 해준다.

* + buffer\_cache\_read



cache에 있는지 확인하기 위해 buffer\_cache\_lookup을 호출해서 cache miss를 확인한 경우 buffer\_cache\_select\_victim함수를 호출해서 evict시킬 slot을 설정하고 cache slot의 정보를 update하고 DISK에서 정보를 가져온다. 후에 cache의 reference bit를 true로 설정해서 접근했다는 것을 표시한 후 buffer cache의 내용을 memory에 copy하는 작업을 수행한다.

* + buffer\_cache\_write



cache에 있는지 확인하기 위해 buffer\_cache\_lookup을 호출해서 cache miss를 확인한 경우 buffer\_cache\_select\_victim함수를 호출해서 evict시킬 slot을 설정하고 cache slot의 정보를 update하고 DISK에서 정보를 가져온다. 후에 cache의 reference bit를 true로 설정해서 접근했다는 것을 표시하고 write, 수정되었다는 것을 표시하는 dirty bit를 true로 설정한 후 memory의 내용을 buffer cache에 copy하는 작업을 수행한다.

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

**텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명**