**Pintos Project 5: Filesystem**

담당 교수 :박성용 교수님

조 / 조원 :20182202 박종하

개발 기간 :2022 -12 -10 ~ 2022-12-26

1. **개발 목표**

* **해당 프로젝트에서 구현할 내용을 간략히 서술.**

파일 크기를 늘릴 수 없는 핀토스의 파일 시스템을 유동적으로 늘릴 수 있게 인덱스기반의 extensible file 과 file growth 를 구현 한다. 또한 현재 핀토스는 루트 디렉토리에 여러 개의 파일이 존재하도록 되어 있는데 이를 root 경로 외의 하위경로가 존재하게 바꾼다. 또한 메모리 성능 향상을 위해 버퍼 캐쉬를 생성한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Extensible file & file growth

현재 핀토스의 파일 시스템은 contiguous하게 이루어져 있어, file growth 가 안되어 external fragmentation 이 일어난다 따라서 index allocation 을 통해 external fragmentation 을 해결하고 파일 크기를 변경할 수 있게 한다. Inode table 을 direct , indirect , double indirect entry 를 생성시키는 것을 통해 가능하게 한다.

1. Subdirectory

Pintos 의 파일 시스템은 현재 루트 디렉토리에 모든 파일이 저장 되어 있는 구조이다. 따라서 서브 디렉토리를 구현하여 루트 경로 이외의 하위경로를 생성하게 한다. 또한 서브 디렉토리와 관련된 system call 을 구현한다. 이를 통해 subdirectory 와 관련된 기능 을 수행할 수 있다.

1. Buffer cache

Disk 에서 read/ write 하는 양이 많아 지면 i/o 하는 양이 많아져 메모리의 성능이 좋지 않다 . 이를 방지하기 위해 buffer cache 를 구현하여 block 의 정보를 저장한다. 또한 cache 의 사이즈가 한정적이기 때문에 빈 캐쉬가 없을 경우 eviction algorithm 인 second chance algorithm 을 활용하여 evict 하여 결과적으로, 메모리 성능을 높인다.

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

Direct, indirect, double indirect 의 세개의 entry 갖도록 inode 구조체에 변경시켜준다. 이는 direct 는 바로 data block 을 접근하도록, indirect 는 indirect block 가 가르키는 data block 를 접근하고, double indirect는 indirect block을 두 번 거치고 최종적으로 data block 을 접근할 수 있도록 한다. 따라서 각 indirect block 은 disk 의 data block 을 128개 (512/4), double indirect block 은 128 \* 128 개 를 저장할 수 있다. 이를 통해 할당할 수 data sector 의 개수가 늘어난다.

Manage 하는 방법은 파일을 read, write, open 하는 등의 접근할 때 byte 의 길이를 통해 direct, indirect, double direct 의 사용유무를 정해야 하고, write를 할 때에는 기존 inode 에 저장되어 있던 것보다 커질 때, 이를 업데이트 시켜줘야 한다.

* Subdirectory
  + Directory entry 관리 방법

Directory 또한 파일의 종류 중에 하나로 inode 를 부여하고 disk의 block 에 write 한다. 따라서, inode에서 directory 인지 여부의 flag를 추가해줬고, 기존 파일 만을 관리하는 file descriptor 또한 구조체를 일부 수정하였다. File 명에 디렉토리의 경로가 포함 되어 있을 경우 이를 적절하게 tokenize 해야 한다. 따라서 ‘/’ 경로를 타고 들어가며 file,directory명을 분리시켜준다. 이는 디렉토리에 넣을 때 dir\_entry 자료 구조를 활용하였다. 이를 통해 subdirectory 의 구조가 갖춰지면 관련된 시스템 콜을 구현한다 ( chdir, inumber, mkdir, readdir,.. 등)

* Buffer cache
  + Buffer cache eviction 방식

가상 메모리의 page eviction 알고리즘 방식과 동일하다. Clock algorithm 인Second change 알고리즘을 썼는데, 캐쉬 블록 마다 reference bit 이 존재한다. 캐쉬 블록을 돌며 한번 접근할 때 1-> 0 으로 바꾸어주고, 만약 0 이면은 해당 캐쉬 블록은 evict 하는 방식으로 구현을 하였다.

* + Buffer cache flush 방식

Buffer cache 가 flush 될 때는 buffer 가 flush all 될 때와 (filesys\_done 에서 호출) 그리고 eviction 되고 dirty bit 이 1일 때이다. buffer\_cache 가 read,write 를 수행할 때 이때 부족하다면 buffer\_cache evict 가 일어나는데, 이때 Dirty bit 이 1 이라면 즉 write 를 했다는 뜻이고 disk 에 수정사항을 반영해줘야 하므로 , flush entry 하는 방식을 이용했다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* **II. A.의 개발 범위를 포함하여 구현 내용에 대한 일정 작성**
* 12.10 ~ 12.15 버퍼 캐쉬 구현
* 12.21- 12.24 extensible file & file growth 구현
* 12.24 – 12.26 subdirectory & system call 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.c , Buffer\_cache.h
  + 버퍼 캐쉬 사용하기 위해 버퍼 캐쉬 구조체 구현
* struct buffer\_cache\_entry : 버퍼 캐쉬 엔트리를 만들어준다. 이 구조체의 멤버변수로는 valid bit , reference bit , dirty bit , disk sector ,buffer 가 존재한다.
* struct buffer\_cache\_entry cache[NUM\_CACHE] : 버퍼 캐쉬로 사용되는 global 정적 배열을 생성한다.
* Struct lock buffer\_cache\_lock : 버퍼 캐쉬의 concurrency 를 보장한다.
  + Init , read,write, flush, flush\_all , terminate, lookup,select\_victim, evict\_setup 함수 구현.

버퍼 캐쉬를 initialize ,read,write, flush,lookup, select\_victim, evict\_setup 할 수있는 함수를 추가적으로 생성한다.

* void buffer\_cache\_init : cache를 초기화하는 함수로 valid bit, reference\_bit, dirty\_bit 변수를 모든 캐쉬 엔트리에 대해서 false로 초기화한다.
* void buffer\_cache\_flush\_entry : 하나의 캐쉬 엔트리를 flush 하는 함수이다.
* void buffer\_cache\_flush\_all : 모든 캐쉬 엔트리에 대해서 buffer\_cache\_flush\_entry 를 호출해주는 함수이다.
* struct buffer\_cache\_entry\* buffer\_cache\_lookup : cache sector 값을 통해 hit/miss 를 판단하여 hit 하면 그에 해당하는 엔트리를 아닐 시 null 을 반환하는 함수이다.
* struct buffer\_cache\_entry\* buffer\_cache\_select\_victim : 버퍼 캐쉬가 부족할 때, evict algorithm 를 수행하여 buffer 엔트리를 evict 한다. 알고리즘은 second algorithm 을 사용한다.
* void buffer\_cache\_read : 버퍼 캐쉬에서 data 를 읽는다. 캐쉬의 데이터를 메모리로 카피한다.
* void buffer\_cache\_write : 버퍼 캐쉬에 write 하는 함수로, 메모리의 데이터를 캐쉬에 옮겨주는 함수이다.
* void buffer\_cache\_terminate : buffer\_cache\_flush\_all 을 호출해주는 함수이다.
* void buffer\_cache\_evict\_setup : evict 되고 나서 read,write 를 진행할 때 dirty bit 이 있으면 flush 함수를 호출하고 sector 를 set 해주고, valid bit ,dirtybit 을 초기화, block read 를 해주는 함수이다.
* 2. Inode.c , inode.h
  + Inode\_disk index 버전으로 수정

Direct\_block ,indirect block , double indirect block 을 inode\_disk 자료 구조의 멤버변수에 추가한다. 또한 subdirectory 구현을 위해 dir 인지 판단하는 is\_dir 변수도 추가한다.

Inode block 하나에는 512/ 4 = 128개의 entry 가 들어간다. 123 개는 direct , 1개는 indirect, 1개는 double\_indirect , 나머지 3개는 is\_dir, length, magic 에 각각하나씩 할당하여 총 128 개의 엔트리가 만들어진다

또한 indirect block 의 경우에는 또 128 개의 entry 를 가진다. 따라서 indir\_block 구조체를 구현하여 블록 개수인 128 개 정적 배열을 멤버 변수로 할당하였다.

* 파일을 쓰거나 읽을 때 . Direct 부터 시작하여 indirect , double indirect 까지 byte 를 따져 준후 그에 맞는 sector를 알아내고 allocation, free 를 진행한다.
  + Byte\_to\_sector 함수 수정
* Static block\_sector\_t byte\_to\_sector : 파일을 쓰거나 읽을 때 , 어떤 블럭에 저장해야 하는지 알려주는 함수이다. Direct 부터 시작하여 indirect , double indirect 까지 byte 를 따져 준후 그에 맞는 sector를 반환한다.
  + Inode\_allocate, inode\_allocated\_indirect 구현
* Inode\_allocated, inode\_allocated\_incdirect : Disk 위치를 알아내었으면 inode 를 Disk 에 allocation 시켜주는 함수이다. Indirect 를 구현하기 위해 level 이 indirect , double indirect 인 경우에 inode\_allocated\_indirect 함수를 호출한다.
  + Inode\_freed, inode\_freed\_indirect 함수 구현 :
* Disk 위치를 알아내었으면 inode를 Disk 에 free 시켜주는 함수이다. Indirect 를 구현하기 위해 level 이 indirect ,double indirect 인 경우에 inode\_allocated\_indirect 함수를 호출한다.
  + Inode\_create 에서 buffer\_cache 로 바꿔줌(write)
* block write 를 새롭게 구현한 buffer cache write 로 바꾸어 준다
  + Inode\_open에서 buffer\_cache 로 read 함
* block read를 새롭게 구현한 buffer cache read 로 바꾸어 준다
  + Inode\_close 수정

Inode\_close 를 할 때, inode\_freed 함수를 호출한다.

* + Inode\_read\_at buffer\_cache 로 read 함
* block read를 새롭게 구현한 buffer cache read 로 바꾸어 준다
  + Inode\_write\_at 에서 buffer\_cache 로 바꿔줌(write)
* block write 를 새롭게 구현한 buffer cache write 로 바꾸어 준다
* 3. Filesys.c, filesys.h
  + Filesysinit 함수에 Buffer\_cache init 추가하여 수정

File system 초기화 할 때 buffer cache 초기화 하는 함수 넣어준다.

* + Filesysdone 함수에 terminate 함수 추가하여 수정

Filesystem 을 close할 때 buffer cache terminate 하는 함수를 넣어준다.

* + Filesys\_create ,filesys\_open, filesys\_remove,filesys\_chdir 수정
* Filesys\_create : 경로를 dir ,file 로 나눠서 file을 create 하고 success 여부를 반환한다.
* filesys\_open: 경로를 dir,, file 로 나눠서 file 을 오픈하는 함수이다.
* filesys\_remove: 경로를 dir,, file 로 나눠서 file 을 remove 하는 함수이다.
* filesys\_chdir: directory 의 정보를 통해 현재 cwd 를 close 하고 새로운 dir 을 설정해준다.
* 4. Futils.c
  + Filesys\_create 변수 추가
* Filesys\_create 변수에 is\_dir 이 추가되었기 때문에 이를 false 로 설정해준다.
* 5. Directory.c, Directory.h
  + Split\_file\_dir, open\_subdir\_path, dir\_is\_empty 함수 추가
* Split\_file\_dir : path 를 받아와 , file 과 directory 를 나눠주는 함수이다. ‘/’ 를 기준으로 나눈다.
* Open\_subdir\_path : 디렉토리의 path 를 오픈한다. 해당 경로를 타고 가서 directory 정보를 return 해준다.
* Dir\_is\_empty :
  + Dir\_create, dir\_open, look\_up,dir\_lookup,dir\_add,dir\_remove 함수 수정
* Dir\_create :directory 를 생성하는 함수이다.
* Dir\_open : directory 를 open 하는 함수이다. Dir->pos 를 dir\_entry 사이즈로 바꾸어준다.
* Dir\_lookup : 디렉토리를 lookup 하는 함수로,subdirectory 를 구현했기때문에. Parent ,current 디렉토리의 lookup 도 고려를 해야한다. 현재이면 inode\_reopen ,parent 이면 read 후 inode open 작업을 진행한다.
* Dir\_add : 파일이 디렉토리이면 자식디렉토리의 정보를 업데이트 시켜준다.
* Dir\_remove : 파일이 디렉토리이면 지우지 못한다. 이를 추가해준다. Empty 이면 지울 수 있게 한다.
* 6. Syscall.c
  + 파일 디스크립터 구조체 변경에 따라 파일 디스크립터 이용한 Open, close, read,write , seek,file\_size,exit, tell, 함수 그에 따라 수정
  + Chdir, mkdir, readdir,isdir, inumer system call 함수 구현

-chdir : dir 을 변경해주는 system call 이다.

- mkdir : dir 을 만들어주는 system call 이다.

- readdir : 디렉토리를 read 해주는 system call 이다.

- isdir : 디렉토리가 존재하는지 알려주는 system call 이다.

- inumer : inode 의 정보를 받는 system call 이다.

* 7. Process.c
  + Load 함수 수정 (cur\_file)

현재 execute 하는 파일을 load 하는 작업에 추가한다.

* + Process\_exit 함수 수정

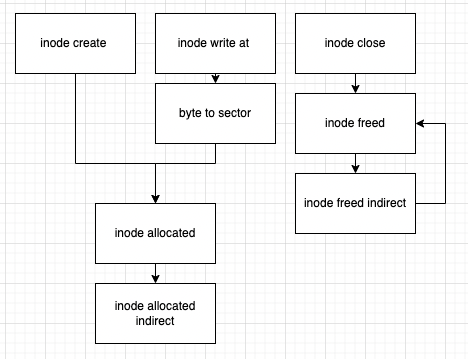
Process exit 할 때 현재 thread 의 파일이 있으면 file\_allow\_write 를 호출해주고 file close 한다.

디렉토리가 존재하면 close 해준다.

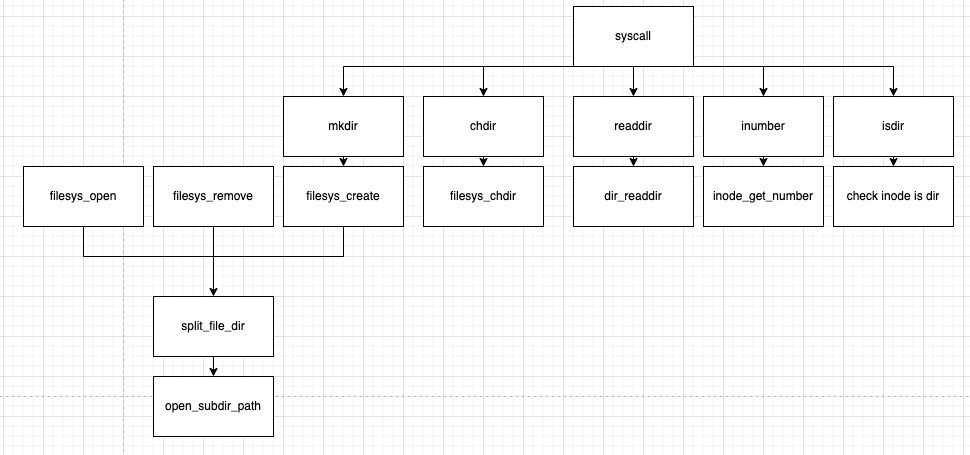
* 8. Thread.c, thread.h
  + File\_descriptor 구조체 수정 (dir ,file 관리할 수 있도록), 초기화시켜줌
* 현재 실행하는 파일인 cur\_file 을 추가 , 초기화 과정
* File descriptor 파일 기존에 file \* 만을 갖고있는 정적 배열이었는데 , dir 과 id 를 갖고 있는 구조체로 수정해줌. 이를 초기화 하는 과정 추가.
* Current working directory 변수 추가 , 초기화 함.

1. **연구 결과**
   1. **Flow Chart**

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

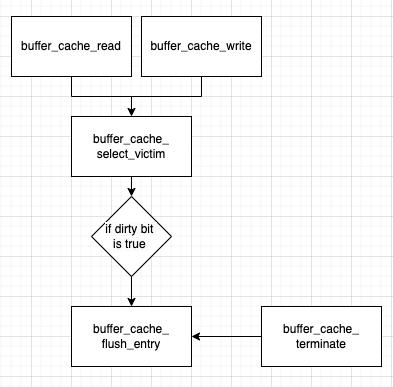


* Subdirectory



* 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

텍스트이(가) 표시된 사진

자동 생성된 설명

다음과 같이 코드에서 direct , indirect ,double direct 블록이 각각 123,1,1 개로 할당된 것을 볼 수있다. 또한 is\_dir 변수는 디렉토리인지 파일인지 구분하는 변수로 추가적으로 설정했다. Indir\_block 은 128 개의 indirect block 이 들어있다. 이는 128 개의 block을 가리킬 수있다는 의미이다.

* + byte to sector

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

Byte 을 받아서 원하는 sector 를 반환하는 함수이다. 블록 단위로 자르기 떄문에 현재 Pos 에 블록 단위로 잘라 index 를 구하고 direct , indirect, double indirect 범위를 계산해나가면서 disk 상의 위치를 알아낸다 read 를 하는 과정 도 버퍼캐쉬를 이용하여 메모리 이용의 효율성을 높인다.

0~direct\_block : direct index

Direct blocks ~ directblocks + indirect\_blocks\_per\_sector : indirect index

directblocks + indirect\_blocks\_per\_sector ~ directblocks + indirect\_blocks\_per\_sector + indirect\_blocks\_per\_sector\* indirect\_blocks\_per\_sector : double indirect index .

* + Inode\_allocated, inode\_allocated\_indirect

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

이는 inode 를 disk 에 할당 하는 함수이다. 일단 byte\_to\_sector 를 호출하여 할당해야하는 사이즈를 알아내서 allocated\_index 에 저장한다. 그후 레벨 즉 Direct , indirect ,double indirect 에 대해서 차례로 할당을 한다. 만약 direct 로 할당이 안되었으면 single indirect 할당을 한다. 이때 inode\_allocate\_indirect 함수를 level 을 indirect 로 호출한다. 이후도 할당이 되지않았으면 inode\_allocate\_indirect 함수의 level 을 double\_indirect 로 호출하여 double indirect allocation 을 수행한다. 할당이 끝난 후 allocated index 는 0 이 되어야 한다. 따라서 0이면 true 아니면 false 를 반환한다. Inode\_allocated\_indirect 함수는 level 에따라서 수행이 달라진다. 처음에 하나를 할당하고 시작하므로 level 이 0 인경우는 그냥 반환하면된다. Level 이 1인 경우는 single 이다. 따라서 limit 만큼의 sector 를 할당한다. Level 2 인경우는 double indirect 이므로 indirect 를 호출하여야 한다. 따라서 limit 만큼의.indirect 를 level1 로 더 호출한다. 그리고 이 과정에서 free\_map\_allocate 함수 및 buffer\_cache\_write 함으로서 index 기반의 inode 를 구현 및 할당 할 수 있다.

* + inode\_freed, inode\_freed\_indirect

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

이는 inode 에서 disk sector 를 프리하는 함수이다. 일단 byte\_to\_sector 함수를 호출해서 free 를 해야하는 sector 의 양을 결정한다. 이는 allocate\_index 에 저장하도록 한다. 그리고 몇개를 프리할 지 direct , indirect, double indirect 순으로 level 별로 나누어서 결정한다. Direct 는 free\_map\_release 로 바로 프리해준다. Direct 만으로 다 프리가 안되면 indirect 로 프리할 수있는 지 본다.이때는 indirect 이므로 inode\_freed\_indirect 함수의 level 을 indirect f로 설정해서 함수를 호출한다. Single\_indirect 로도 다 프리를 하지못하면 double indirect 로 프리를 해주어야한다. 이때에도 level 을 double indirect 로 설정해서 indirect 함수를 호출한다. inode\_freed\_indirect 함수는 레벨 별로 나누어서 동작을 달리한다. 먼저 direct 이면 바로 free\_map\_release 를 하면된다. 그 외의 경우에는 level 별로 range 를 결정하고 range 를 바탕으로 LIMit 를 결정해서 해당 limit 만큼 indirect 함수를 level -1 로 낮추어서 호출하면된다. 마지막으로 프리된 num\_sector 의 수는 0 이어야한다.

* Subdirectory
  + split\_file\_dir\_name, open\_subdir\_path,dir\_is\_empty

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명- split\_file\_dir\_name 함수는 주어진 경로를 인자로 받아서 directory 와 filename 으로 쪼개 는 함수이다. 마지막 ‘/’ 가 나오기 전까지는 directory 에 저장하고, 그 후의 것은 filename 으로 저장한다.

- open\_subdir\_path 함수는 서브 디랙토리를 오픈하는 함수이다. Directory 에 해당 하는 부분이 루트이면 open\_root 를 호출하고 , 아니면 dir\_reopen 또는 dir\_open\_root 를 호출한다. /과 / 사이에 있는 모든 ㅁ문자열에 해당하는 디렉토리를 열고 닫는 과정을 진행한다. 또한 마지막으로는 삭제된 디렉토리를 여는 경우에 대해서 예외처리를 한다.

- dir\_is\_empty 는 파일 remove 과정에서 디렉토리에 파일이 있는지 검사하기위해 구현하였다. 디렉토리 엔트리를 돌면서 사용 여부를 확인한다.

- 위의 split\_file\_dir\_name 과 open\_subdir\_path 함수는 filesys.c의 filesys\_create, filesys\_open, filesys\_remove 함수에서 호출하게 되어, 기존 루트에서 파일 을 관리하던 파일시스템이 subdirectory 를 갖는 구조로 변경된다.

* 파일 디스크립터 구조체

텍스트이(가) 표시된 사진

자동 생성된 설명

* 기존에는 file\* 만을 활용했어도 되었지만 현재는 directory 를 포함하고 있으므로 파일 디스크립터 구조체를 dir 을 포함하도록 변경하였다.
* 시스템 콜 (chdir, mkdir , readdir ,isdir,inumber,search\_file\_descriptor)

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

변경한 파일 디스크립터 구조체와 서브 디렉토리 구조를 통해, chdir, mkdir , readdir ,isdir,inumber 시스템 콜을 확인할 수 있다. 이에 앞서 search\_file\_descriptor 함수를 확인해볼 필요가 있는데 , 이는 파일 디스크립터 넘버와 file type 인지 directory type 인지 플래그를 확인하여 그에 해당하는 파일 디스크립터를 넘겨주는 것이다. chdir, mkdir , readdir ,isdir,inumber 는 각각 filesys\_chdir, filesys\_create, dir\_readdir, isdir 변수,inode\_get\_inumber 함수를 호출하는 방식으로 구현이 되어있다. 또한 기존에 구현했던 함수를 바꿀 필요가 있었다. 구체적으로 exit, open, close 를 주목할 필요가 있는데 , exit 에서는 모든 파일 디스크립터 테이블을 삭제해주는 작업을 진행해준다. Open 에서는 file 을 셋 해주고 추가적으로, dir 일 경우 fd-> dir 을 set 해준다. 또한 만약 파일 디스크립터가 empty 되었으면 0,1,2번을 제외한 3번 부터 할당을 해줘야 하므로 다음과 같이 설정을 했다.  
클로즈 같은 경우는 dir , file 이 있을 때 dir\_close , file\_close 를 각각 진행 해주는 방식으로 했다. 이외의 시스템 콜 같은 경우 파일 디스크립터 구조체의 변경 이외에는 변경점이 없다.

* Thread 구조체에 fd\_list, cur\_work\_dir , cur\_file 선언
* 텍스트이(가) 표시된 사진

  자동 생성된 설명

다음과 같이 파일 디스크립터 구조체 리스트인 fd\_list, 현재 working directory 인 cur\_work\_dir , 현재 실행되는 파일인 cur\_file을 thread구조체에 선언한다.

* Process\_exit
* 텍스트이(가) 표시된 사진

  자동 생성된 설명 텍스트이(가) 표시된 사진

  자동 생성된 설명

Process\_exit 할 때 현재 실행되는 파일이 존재하면 file\_allow\_write 할 수있게 함수를 호출하고 , file close 를 한다.그리고 directory 가 존재할 경우에도 이를 close 해준다.

Load 할때 마지막에 stack을 셋업하고 현재 실행되는 파일을 설정해준다.

* Buffer cache  
  **구현한 경우만 작성**
* 텍스트이(가) 표시된 사진

  자동 생성된 설명
* 버퍼 캐쉬의 구조체는 다음과 같다. Valid bit , reference bit,dirty bit , disk sector buffer 변수를 갖고 있으며, 이는 NUM\_CACHE 만큼의 정적 배열로 선언되어 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

버퍼 캐쉬를 initialize 하는 함수이다. NUM cache 만큼 돌면서 valid, reference, dirty bit 을 false 로 설정해준다. 그리고 lock 또한 initialize 를 해준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Buffer\_cache\_read 함수는 섹터에 해당하는 엔트리의 내용을 user\_buffer 에 복사하는 것이다. Lookup 함수를 이용해 해당 entry 를 찾고 엔트리가 없으면 evict 함수를 호출하여 엔트리를 할당한다. 그 후 evict setup 함수를 통해 set 해주고 reference bit 은 true 가 된다. 그 후 user\_buffer 에 값을 복사해 준다.

Buffer\_cache\_write 함수는 user\_buffer 에 해당하는 내용을 sector 에 해당하는 엔트리로 복사하는 것이다. Lookup 함수를 이용해 해당 entry 를 찾고 엔트리가 없으면 evict 함수를 호출하여 엔트리를 할당한다. 그 후 evict setup 함수를 통해 set 해주고 , write 했으므로 reference bit, dirty bit 을 true 로 설정해 준다.그리고 마지막으로 엔트리에 내용을 복사해준다.

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

Buffer\_cache\_look up 함수는 해당 sector 를 저장하는 캐쉬 엔트리를 반환하는 함수이다. 이는 valid 해야 하고 target 과 같아야지 cache 를 반환하게하였다. Select\_victim 함수는 clock 알고리즘을 활용하여 구현 하였다. Valid bit 이 false 인것은 아직 할당 되지 않았다는 이야기 이므로 바로 반환하고 reference bit 이 1이면 0으로 바꿔주고 다음 clock 으로 간다. 그러다가 reference 가 0 인 엔트리를 다시 만나면 반환하는 알고리즘을 evict 알고리즘을 구현한다.

Flush entry 는. Valid 할때 block\_write 를 통하여 디스크에 변경사항을 옮겨준다. 또한 dirty bit 을 false 로 해줘야한다. Flush all 이에 해당하는 것을 모두해주는 함수이다.

Buffer\_cache\_evict\_set\_up은 read 또는 write 할 때 evict 가 일어날 때 호출해주는 함수로 flush 를 하고 sector 와 bit 를 setup 하기 위해서 새로 생성하였다 dirty bit 이 true 일 때 flush 해주고 disk sector 를 set 해주고 valid ,dirty bit 을 set해준다.그리고 block read 를 호출하여 새로운 entry 정보를 setting 해준다.

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

**텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명**

모든 테스트 케이스에 대해 성공적으로 마침을 알 수가 있다.