Operating Systems
Project #3 Wiki [유민수 교수님]

2019087192 이예진

목차

1. Design
2. Implement
3. Result
4. Trouble shooting

1. Design

이번 과제에서는 (1) Multi Indirect, (2) Symbolic Link (3) Buffered I/O 기능을 구현하여 파일시스템을 개선합니다.

1.1 Multi Indirect

xv6 는 direct 와 Single indirect 를 통해 파일의 정보를 저장합니다. 더 큰 파일을 다르기 위해 multi indirect 를 구현합니다. 따라서 이를 구현하기 위해 기존 코드에 Double Indirect, Triple indirect 를 추가하였습니다.

이렇게 생성된 Double Indirect 에서는 최대 16384 블록(8MB)의 파일을 관리할 수 있고 Triple Indirect 에서는 약 1GB 까지 파일을 관리할 수 있습니다. 이 때 File sysyem 의 최대 데이터 블록 개수 제한을 풀어주기 위해 FSSIZE 를 100000 개로 설정해주었습니다. 검색에 따르면 20 만개 까지 늘릴 수 있다고 합니다.

```
#define FSSIZE       100000  // size of
file system in blocks more than 30000
```

[param.h]에서 FSSIZE 값 1000 -> 100000

관련 상수들은 fs.h 에 저장해주었습니다.

```
#define NDIRECT 10 //(DOU, TRI) minus 2 //init val = 12
#define NINDIRECT (BSIZE / sizeof(uint))
// #define MAXFILE (NDIRECT + NINDIRECT)

#define NINDIRECT_DOU (NINDIRECT * (BSIZE / sizeof(uint)))
#define NINDIRECT_TRI (NINDIRECT_DOU * (BSIZE / sizeof
(uint)))
#define MAXFILE (NDIRECT + NINDIRECT + NINDIRECT_DOU +
NINDIRECT_TRI)

enum FS_ADDR_TYPE {
  FS_ADDR_DIRECT = 0,
  FS_ADDR_SINGLE_INDIRECT = NDIRECT,
  FS_ADDR_DOUBLY_INDIRECT = NDIRECT + 1,
  FS_ADDR_TRIPLE_INDIRECT = NDIRECT + 2
};
```

[fs.h] Double, Triple 값이 추가됨에 따라 내용이 다음과 같이 수정되었습니다. NDIRECT 값도 2 가 됩니다.

1.2 Symbolic Link

xv6 는 Hard Link 를 지원하지만 Symbolic Link 를 지원하지 않습니다. 이 때 ln.c 라는 user program 을 변경하여 shell 에 ln -h [old] [new]과 같이 명령을 입력할 경우에는 hard 링크 파일로 만들고, shell 에 ln -s [old] [new]와 같이 입력하였을 때에는 symbolic 링크 파일로 만드는 함수를 호출해줍니다.

```
int
main(int argc, char *argv[])
{
  if(argc != 3  && (argc != 4 || (argc == 4 && !strcmp(argv[2], "-s")))){
    printf(2, "Usage: ln old new or symlink\n");
    exit();
  }
  if(strcmp(argv[1], "-h") == 0) {
    if(link(argv[2], argv[3]) < 0) {
      printf(2, "link %s %s: failed\n", argv[1], argv[2]);
    }
  }
  if(strcmp(argv[1], "-s") == 0) {
    if (symlink(argv[2], argv[3]) < 0) {
      printf(2, "symlink %s %s: failed\n", argv[2], argv[3]);
    }
  }

  exit();
}
```

[ln.c] 에서 기존 코드 변경

1.3 Buffered I/O (Sync)

기존 xv6 는 write operation 에 대하여 group flush 를 진행해 특정 프로세스가 다수의 write operation 을 발생시킬 때 성능을 저하시킨다. 위의 문제를 해결하기 위해 sync 함수가 호출될 때만 flush 하도록 하는 Buffered I/O 를 구현한다. 메모리에 존재하는 Buffer 에 먼저 담아둔 후, device 에 buffer 의 내용을 옮긴다. filewrite 함수에서는 commit 이 진행중인지 확인하고 없다면 log.outstanding + 1 을 한다. 이후 메모리상의 buffer 에 저장하고 end_op 를 실행한다. 따라서 end_op 에서의 함수 호출을 분리했다.

```
// called at the start of each FS system call.
void
begin_op(void)
{
  acquire(&log.lock);
  while(1){
    if(log.committing){
      sleep(&log, &log.lock);
    } else if(log.lh.n + (log.outstanding+1)*MAXOPBLOCKS > LOGSIZE){
      // this op might exhaust log space; wait for commit.
      sleep(&log, &log.lock);
      // commit_sync(1);
    } else {
      log.outstanding += 1;
      release(&log.lock);
      break;
    }
  }
}
```

```
void
begin_op(void)
{
  acquire(&log.lock);
  while(1){
    if(log.committing){
      sleep(&log, &log.lock);
    } else if(log.lh.n + (log.outstanding+1)*MAXOPBLOCKS >
    LOGSIZE){
      // this op might exhaust log space; wait for commit.
      // sleep(&log, &log.lock);
      commit_sync(1);
    } else {
      log.outstanding += 1;
      release(&log.lock);
      break;
    }
  }
}
```

[log.c] begin_op 위를 아래로 수정하였습니다. 이를 통해 Buffer 에 공간이 부족할 경우 강제적으로 sync 를 발생 시킵니다.

2. Implementation

2.1 Multi Indirect

```
static uint
bmap(struct inode *ip, uint bn)
{
  uint addr, *a;
  struct buf *bp;
  //* check bn
  if(bn < NDIRECT){
    if((addr = ip->addrs[bn]) == 0)
      ip->addrs[bn] = addr = balloc(ip->dev);
    return addr;
  }
  bn -= NDIRECT;

  if(bn < NINDIRECT){
    //* Load indirect block, allocating if necessary.
    if((addr = ip->addrs[FS_SINGLE_INDIRECT]) == 0)
      ip->addrs[FS_SINGLE_INDIRECT] = addr = balloc(ip->dev);
    bp = bread(ip->dev, addr);
    a = (uint*)bp->data;
    if((addr = a[bn]) == 0){
      a[bn] = addr = balloc(ip->dev);
      log_write(bp);
    }
    brelse(bp);
    return addr;
  }
  bn -= NINDIRECT;

  if (bn < NINDIRECT_DOU) {
    //* Load doubly indirect block, allocating if necessary.
    if ((addr = ip->addrs[FS_DOUBLY_INDIRECT]) == 0)
      ip->addrs[FS_DOUBLY_INDIRECT] = addr = balloc(ip->dev);

    // Load block
    bp = bread(ip->dev, addr);
    a = (uint*)bp->data;
    if ((addr = a[bn / NINDIRECT]) == 0) {
      a[bn / NINDIRECT] = addr = balloc(ip->dev);
      log_write(bp);
    }
    brelse(bp);

    // Load address block
    bp = bread(ip->dev, addr);
    a = (uint*)bp->data;
    if ((addr = a[bn % NINDIRECT]) == 0) {
      a[bn % NINDIRECT] = addr = balloc(ip->dev);
      log_write(bp);
    }
    brelse(bp);

    return addr;
  }
  bn -= NINDIRECT_DOU;
```

```
  if (bn < NINDIRECT_TRI) {
    //* Load doubly indirect block, allocating if necessary.
    if ((addr = ip->addrs[FS_TRIPLE_INDIRECT]) == 0)
      ip->addrs[FS_TRIPLE_INDIRECT] = addr = balloc(ip->dev);

    //* first block load
    bp = bread(ip->dev, addr);
    a = (uint*)bp->data;
    if ((addr = a[bn / NINDIRECT_DOU]) == 0) {
      a[bn / NINDIRECT_DOU] = addr = balloc(ip->dev);
      log_write(bp);
    }
    brelse(bp);

    //* second block load
    bp = bread(ip->dev, addr);
    a = (uint*)bp->data;
    if ((addr = a[(bn % NINDIRECT_DOU) / NINDIRECT]) == 0) {
      a[(bn % NINDIRECT_DOU) / NINDIRECT] = addr = balloc(ip->dev);
      log_write(bp);
    }
    brelse(bp);

    //* address block load
    bp = bread(ip->dev, addr);
    a = (uint*)bp->data;
    if ((addr = a[(bn % NINDIRECT_DOU) % NINDIRECT]) == 0) {
      a[(bn % NINDIRECT_DOU) % NINDIRECT] = addr = balloc(ip->dev);
      log_write(bp);
    }
    brelse(bp);

    return addr;
  }

  panic("bmap: out of range");
}
```

[fs.c] bmap 함수의 구현

기존 bmap 함수에 구현된 single indirect pointer 의 구현을 참고하여 bn 이 doubly 나 triple indirection 범위에 포함되면 그 정의에 맞게 adresss 찾을 수 있다.

```c
#define NDIRECT 10 //(DOU, TRI) minus 2 //init val = 12
#define NINDIRECT (BSIZE / sizeof(uint))
// #define MAXFILE (NDIRECT + NINDIRECT)

#define NINDIRECT_DOU (NINDIRECT * (BSIZE / sizeof(uint)))
#define NINDIRECT_TRI (NINDIRECT_DOU * (BSIZE / sizeof(uint)))
#define MAXFILE (NDIRECT + NINDIRECT + NINDIRECT_DOU + NINDIRECT_TRI)

enum FS_TYPE {
  FS_DIRECT = 0,
  FS_SINGLE_INDIRECT = NDIRECT,
  FS_DOUBLY_INDIRECT = NDIRECT + 1,
  FS_TRIPLE_INDIRECT = NDIRECT + 2
};


// On-disk inode structure
struct dinode {
  short type;           // File type
  short major;          // Major device number (T_DEV only)
  short minor;          // Minor device number (T_DEV only)
  short nlink;          // Number of links to inode in file system
  uint size;            // Size of file (bytes)
  uint addrs[NDIRECT+3];   // Data block addresses
};
```

[fs.h] 인자 확장 [DOU, TRI] FS_TYPE = inode 에서 각 pointer 의 시작 index.

3.2 Symbolic Link

sys_symlink 시스템 콜을 만들어 주었고 fcntl.h 에 새로운 flag(O_NOFOLLOW)를 만들었습니다. open system call 을 수정함으로써 어떤 명령어를 입력하던 적절히 적용할 수 있습니다. (ls, cat.. 등등)

```
//* a process specifies O_NOFOLLOW in the flags to open, open should open the symlink
//!DO NOT FOLLOW SYMLINK
if(ip->type == T_SYMLINK && (omode != O_NOFOLLOW)) {
  int count = 0;
  while(ip->type == T_SYMLINK && count < 10) {
    readi(ip, (char*)&length, 0, sizeof(int));
    readi(ip, path, sizeof(int), length + 1);
    iunlockput(ip);
    if((ip = namei(path)) == 0){
      cprintf("Error: Inode cannot found. Original file could be deleted or possible
      inode corruption occured.\n");
      end_op();
      return -1;
    }
    ilock(ip);
    count++;

    if(count >= 100){
        cprintf("Cycle!\n"); //* If the links form a cycle, you must return an error code.
        iunlockput(ip);
        end_op(ROOTDEV);
        return -1;
    }
  }
}
```

[sysfile.c] sys_open 함수

```
struct inode*
get_ip(struct inode *ip, char* path) {
  char length;
  if(ip->type == T_SYMLINK) {
      while(ip->type == T_SYMLINK) {
        // cprintf("get_ip\n");  //* get ip from the symlink
        readi(ip, (char*)&length, 0, sizeof(int));
        readi(ip, path, sizeof(int), length + 1);
        iunlockput(ip);
        if((ip = namei(path)) == 0){ //* if namei return 0, the way is broken.
          cprintf("[ERROR] You can't access the data. It might be deleted.\n");
          end_op();
          return ip;
        }
        ilock(ip);
      } //* if the type is T_SYMLINK
  }
  return ip;
}
```

```c
int sys_get_ip(void) //* get system_ip
{
  char *path;
  struct inode *ip;
  if (argstr(1, &path) < 0) //* check validation
    return -1;
  if((ip = namei(path)) == 0){ //* check validation
      end_op();
      return -1;
    }
  get_ip(ip, path);
 return 0;
}
```

```c
//* if the path is by symlink, have to change ip to point origin.
ip = get_ip(ip,  path);
```

[exec.c] get_ip 함수 (systemcall 로 구현하였습니다.) symlink 로 exec 함수에 접근하면 origin ip 를 호출해줍니다. 사실 이번 과제 구현을 비롯해 다른 상황에서도 수행 가능하도록 하였습니다.

3.3 Symbolic Link

```c
void
end_op(void)
{
  int do_commit = 0;

  acquire(&log.lock);
  log.outstanding -= 1;
  if(log.committing)
    panic("log.committing");
  if(log.outstanding == 0){
    do_commit = 1;
    log.committing = 1;
  } else {
    // begin_op() may be waiting for log space,
    // and decrementing log.outstanding has decreased
    // the amount of reserved space.
    wakeup(&log);
  }
  release(&log.lock);

  if(do_commit){
    // call commit w/o holding locks, since not allowed
    // to sleep with locks.
    commit();
    acquire(&log.lock);
    log.committing = 0;
    wakeup(&log);
    release(&log.lock);
  }
}
```

```c
acquire(&log.lock);
log.outstanding -= 1;

if (log.outstanding == 0)
  wakeup(&log);

release(&log.lock);
```

[log.c] end_op 함수 기존에는 왼쪽과 같이 commit 을 계속 실행하지만 이 때 load 가 너무 크기 때문에 오른쪽과 같이 commit 영역을 삭제했습니다.
그리고 이를 commit_sync 함수로 옮겨 특정 시점에만 commit 을 진행해줍니다.

```
int
commit_sync(int locked)
{
  //* taked the structure of end_op
  //* refered to end_op

  if (!locked) acquire(&log.lock);

  while (log.outstanding > 0)
  {
    sleep(&log, &log.lock);
  }
  if (log.committing)
  {
  while (log.committing)
    sleep(&log, &log.lock);

  if (!locked) release(&log.lock);
    return -1;
  }

  log.committing = 1;
  //* release for the acquire(&log.lock); of begin_op()
  release(&log.lock);
  //log.lh.n
  int buffer_num = log.lh.n; //* save the buffer
  //implement commit w/o
  commit();
  acquire(&log.lock);
  log.committing = 0;
  wakeup(&log);

  if (!locked) release(&log.lock);
  //log.lh.n
  return buffer_num;
}
```

[log.c] commit_sync 함수 생성.

```
int
sys_sync(void)
{
  //* received the num of block from commit_sync function
  int block = commit_sync(0);
  return block;
}
```

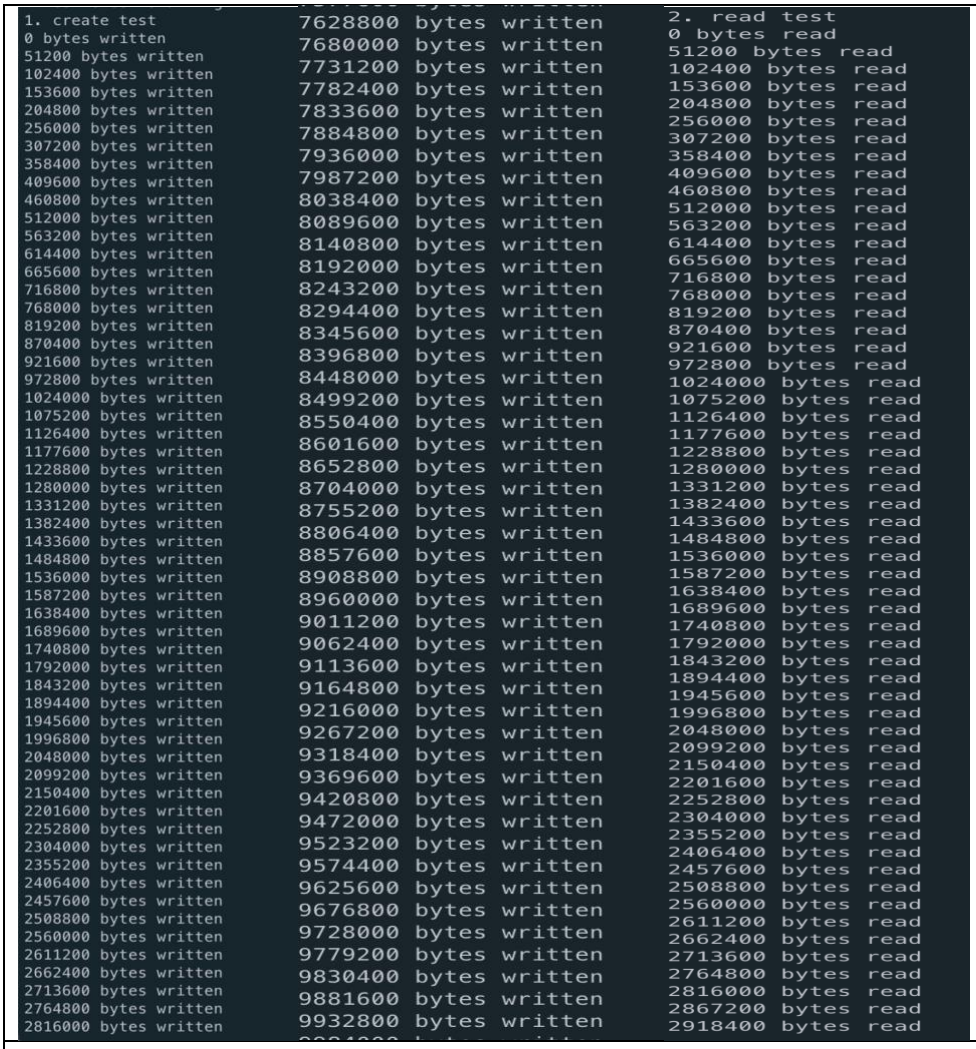
int sync(void)함수

sync 함수를 실행했을 때, Buffered I/O 가 구현되어야 합니다. 또한 return 할 때, 현재 Buffer 에 있는 모든 dirty buffer 가 flush 되며 성공 시 flush 된 block 수가, 실패 시 -1 을 반환해야 합니다. sys_sync 시스템 콜에서 commit_sync 를 통해 받아온 block 을 리턴해주는 방식으로 문제를 해결하였습니다.

3. Result

3.1 Multi Indirect

File Size 를 굉장히 다양하게 테스팅 해보았습니다. 특히 Double Indirect 에서는 최대 16384 블록(8MB)의 파일을 관리할 수 있고 Triple Indirect 에서는 약 1GB 까지 파일을 관리할 수 있어야 하기 때문에 4MB, 500MB 를 각각 테스팅 해보았습니다. 이 테스팅을 위해 filesizetest.c 라는 유저 프로그램을 만들었습니다. 다음은 테스팅 결과 입니다. 4MB, 500MB 에서도 문제 없이 돌아가는 것을 확인할 수 있습니다.



[filesizetest.c] written ,read 모두 잘 구동되는 것을 확인할 수 있습니다.

3.2 Symbolic Link

Hard, Symbolic 링크 파일은 각각 기존의 write, read, open, close 등과 같은 File Operation을 적절하게 진행할 수 있어야 합니다.
symlinktest.c file을 통해 테스팅을 진행하였습니다.

```
mkdir("/testsym");

fd1 = open("/testsym/a", O_CREATE | O_RDWR);
if(fd1 < 0) printf(1,"failed to open a");

r = symlink("/testsym/a", "/testsym/b");
if(r < 0)
  printf(1,"symlink b -> a failed");

if(write(fd1, buf, sizeof(buf)) != 4)
  printf(1,"failed to write to a");

if (stat_slink("/testsym/b", &st) != 0)
  printf(1,"failed to stat b");
if(st.type != T_SYMLINK)
  printf(1,"b isn't a symlink");
```

[symlinktest.c] testsym이라는 폴더를 만든 뒤 a 라는 파일을 생성하게 하였고, a 에 abcd라는 것을 입력해 넣었습니다. (buf[4] = {'a', 'b', 'c', 'd'};) 물론 ln -s 를 이용해 연결 하여도 잘 연결 됩니다.

```
$symlinktest
Start: test symlinks
SYS_SYMLINK
$ ls testsym/b
a                 2 23 4
$ ls testsym/a
a                 2 23 4
```

symlinktest를 호출하고 ls로 b를 호출했을 때 원본파일인 a가 잘 불려져 나온다는 것을 확인할 수 있습니다.

```
$ cat testsym/a/
abcd$ cat testsym/b/
abcd$
```

```
fd2 = open("/testsym/b", O_RDWR);
if(fd2 < 0)
  printf(1,"failed to open b");
if(write(fd1, buf1, sizeof(buf)) != 4)
  printf(1,"failed to write to b");
read(fd2, &c, 1);
if (c != 'a')
  printf(1,"failed to read bytes from b");
```

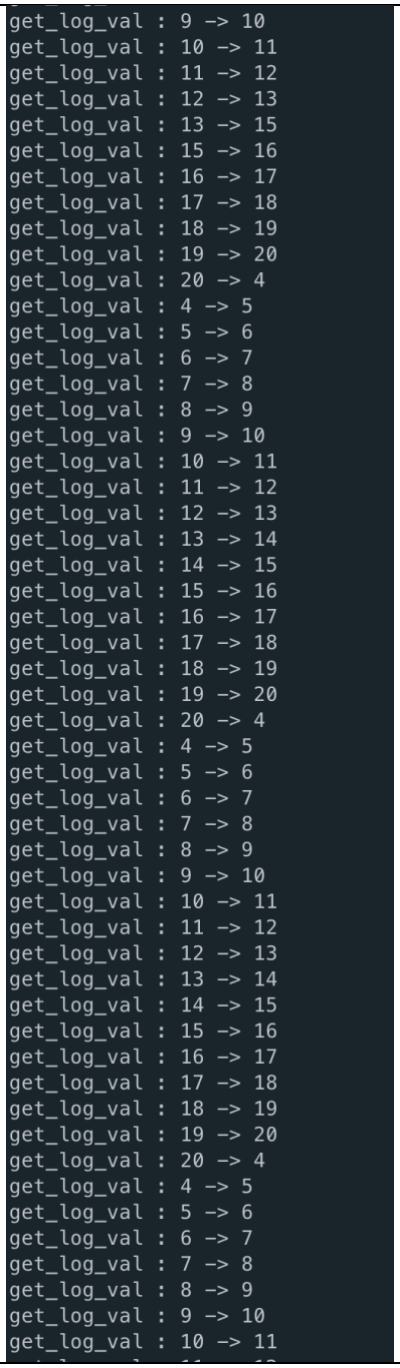
```
$ cat testsym/a/
abcdaaaa$
```

cat을 사용했을 때에도 a안의 내용과 같게 b의 내용이 잘 호출되고 b에 값을 입력하면 a에서도 잘 수정 값이 반영되는 것을 확인할 수 있습니다. (buf1[4] = {'a', 'a', 'a', 'a'};)

```
$ rm testsym/a/
$ ls testsym/b/
[ERROR] You can't access the data. It might be deleted.
ls: cannot open /testsym/a
```

원본 파일을 지웠을 때는 접근 불가능합니다.

3.3 Buffered I/O (Sync)



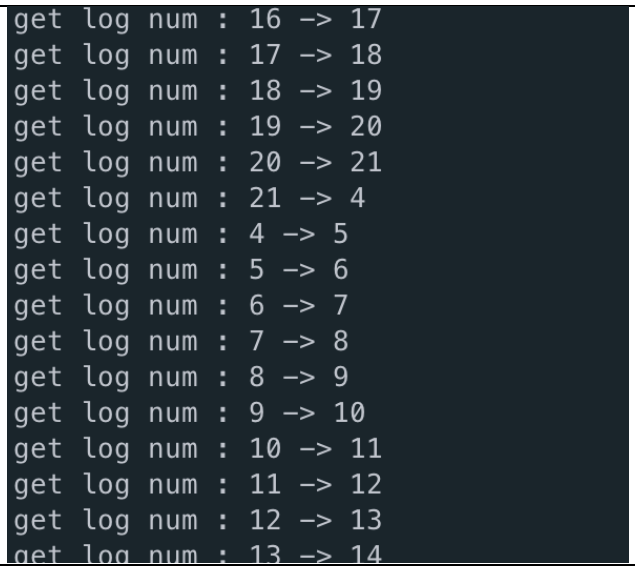
buffered log check 공간이 가득차면, 자동으로 줄어듭니다.

```
get log num : 13 -> 0
get log num : 4 -> 4
get log num : 5 -> 5
get log num : 6 -> 6
get log num : 7 -> 7
get log num : 8 -> 8
get log num : 9 -> 9
get log num : 10 -> 10
get log num : 11 -> 11
get log num : 12 -> 12
sync_buffer_num: 13
```

중간에 sync함수를 호출하면 flush 된 시점의 block 수를 잘 호출하는 것을 확인할 수 있습니다.

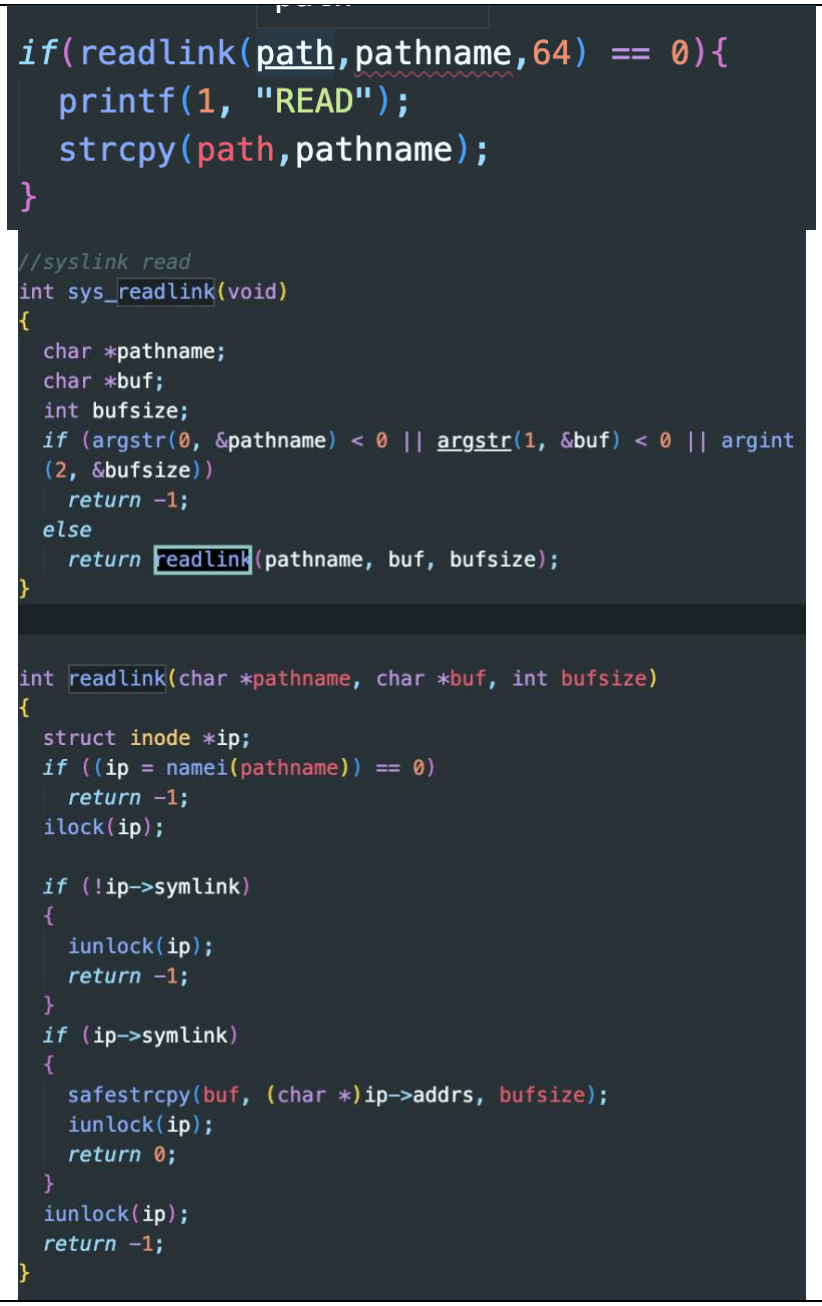
4. Trouble shooting

1. sync( )를 구현하는 과정에 있어서 begin_op 를 어떻게 수정할지 막막해서 그대로 돌려보았더니 파일이 크면 특정 시점에서 log space 가 부족해지고 sleep 상태에 빠져버리며 아무것도 동작하지 못하는 상황이 발생하였습니다. 이를 해결하기 위해 begin_op 에서도 sleep function 을 제거하고 *f*(log.lh.n + (log.outstanding+1)*MAXOPBLOCKS > LOGSIZE -1)일 때 commit_sync 함수를 호출하여 공간을 비워주어 문제를 해결하였습니다. synctest 를 하면서 log_val 을 찍어보면 아래와 같이 출력결과가 나오는 것을 확인할 수 있습니다.



21->4 로 다시 log 의 수가 줄어든 것을 확인할 수 있습니다.

2. symlink 를 구현할 때 cycle 에 빠지는 오류 및 연결이 안 되는 오류. 처음에 구현할 때는 open 함수를 수정해 symlink type 일 때 namei 를 통해 ip 를 return 받아야 겠다는 생각을 못했습니다. 그래서 readlink 라는 함수를 만들어 한정적인 상황에서만 실제 path 를 읽어올 수 있는 함수를 만들었습니다.



```
if(readlink(path,pathname,64) == 0){
  printf(1, "READ");
  strcpy(path,pathname);
}
```

```
//syslink read
int sys_readlink(void)
{
  char *pathname;
  char *buf;
  int bufsize;
  if (argstr(0, &pathname) < 0 || argstr(1, &buf) < 0 || argint
  (2, &bufsize))
    return -1;
  else
    return readlink(pathname, buf, bufsize);
}

int readlink(char *pathname, char *buf, int bufsize)
{
  struct inode *ip;
  if ((ip = namei(pathname)) == 0)
    return -1;
  ilock(ip);

  if (!ip->symlink)
  {
    iunlock(ip);
    return -1;
  }
  if (ip->symlink)
  {
    safestrcpy(buf, (char *)ip->addrs, bufsize);
    iunlock(ip);
    return 0;
  }
  iunlock(ip);
  return -1;
}
```

위 부터 차례로 [ls.c], [sysfile.c]

그러나 이렇게 구현하면 여러 상황에서 버그가 발생했고, 지정한 명령어에서만 잘 수행된다는 것을 확인할 수 있었습니다. (ls 에서만 잘 원하는 결과가 도출되고, cat 과 같은 명령에서는 동작 안함) 따라서 open 함수를 수정하는 방법을 택했습니다.

감사합니다☺