Operating System – Project3 Wiki

2019073181 심기호

Design

명세에서 요구하는 조건에 대해서 어떻게 구현할 계획인지, 어떤 자료구조와 알고리즘이 필요한지, 자신만의 디자인을 서술합니다.

Multi Indirect

-먼저 NDIRECT을 10으로 바꿔주고, inode의 addrs배열에서 NDIRECT+3으로 NIDRECT, NINDIRECT, DINDIRECT, TINIDIRECT순으로 인덱스를 지정해서 구현할 것이고, bmap에서 bn이 NINDIRECT, DINDIRECT, TINIDIRECT 보다 큰지 각각 비교해서 그 함수 안에서 블록의 개수를 할당해 줄 것이다. 그리고 Multi Indirect가 블록을 해제해줄때에도 블록의 개수만큼 해제해야 하기 때문에 itrunc함수에서도 반복문을 통해서 블록을 해제할 것이다.

Symbolic link

-먼저 ln.c에서 link 커맨드를 받게 하고 sysfile.c에서 sys_softlink함수를 만들고 시스템콜에 해당하는 처리를 usys.S, syscall.h 등 여러 파일에서 해줄 것이다. 그 함수에서 create로 type을 SOFT_LINK로 설정해서 inode를 만들고 그 후 inode의 addrs에 old 파일을 넣어줄 것어주는 형식으로 구현할 것이다.

Sync-구현하지 못했습니다.

Implement

본인이 새롭게 구현하거나 수정한 부분에 대해서 무엇이 기존과 어떻게 다른지, 해당코드가 무엇을 목적으로 하는지에 대한 설명을 구체적으로 서술합니다.

Multi Indirect

```
24  #define NDIRECT 10
25  #define DOUBLE_INDIRECT 11
26  #define TRIPLE_INDIRECT 12
27  #define NINDIRECT (BSIZE / sizeof(uint))
28  #define DINDIRECT 128 * 128
29  #define TINDIRECT 128 * 128 * 128
30  #define MAXFILE (NDIRECT + NINDIRECT+DINDIRECT+TINDIRECT)
31
```

먼저 fs.h에서 각 DIRECT의 종류를 선언해준다.

```
bn -= NINDIRECT;
  if (bn < DINDIRECT) // 0~128*128보다 작다면
  {
    if ((addr = ip->addrs[DOUBLE_INDIRECT]) == 0)
      ip->addrs[DOUBLE_INDIRECT] = addr = balloc(ip->dev);
    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);
    }
    index = bn % 128;
    bp1 = bread(ip->dev, addr);
    a1 = (uint *)bp1->data;
    if ((addr = a1[index]) == 0)
    {
      a1[index] = addr = balloc(ip->dev);
      log_write(bp1);
    }
    brelse(bp);
    brelse(bp1);

    return addr;
  }
```

그 후 fs.c의 bmap 함수의 일부분이다. 위 사진은 DINDIRECT을 구현한 캡처이다. NINDIRECT와 비슷하게 구현했는데, 먼저 bn에서 NINDIRECT을 빼준 후, bn이 DINDIRECT보다 작다면 addr을 ip->addrs[11]로 받아오게했고 0이라면 할당을 해주었다. 그 후 block pointer을 할당해주고, 배열 데이터를 a로 받아와서 다시 bn을 NINDIRECT로 나눈 값, 즉 a배열에서 offset이 어디있는지 찾아서 그 주소를 addr로 받게하였고 또한 0이라면 할당을 해준다. 그 찾아간 inner 배열을 a1이라 받고, a1배열에서 찾고자 하는 index(bn%128)을 addr이라 받고 block pointer들은 release를 해주고, addr을 리턴한다.

```
bn -= DINDIRECT;
 if (bn < TINDIRECT) // 0~ 128*128*128
 {
   if ((addr = ip->addrs[TRIPLE_INDIRECT]) == 0)
     ip->addrs[TRIPLE_INDIRECT] = addr = balloc(ip->dev);
   bp = bread(ip->dev, addr);
   a = (uint *)bp->data;
   // cprintf("asdf\n");
   if ((addr = a[bn / (NINDIRECT * NINDIRECT)]) == 0)
   {
     a[bn / (NINDIRECT * NINDIRECT)] = addr = balloc(ip->dev);
     log_write(bp);
   }

   index = bn % (128 * 128);
   bp1 = bread(ip->dev, addr);
   a1 = (uint *)bp1->data;
   if ((addr = a1[index / NINDIRECT]) == 0)
   {
     a1[index / NINDIRECT] = addr = balloc(ip->dev);
     log_write(bp1);
   }
   index = index / NINDIRECT;
   index = index % 128;

   bp2 = bread(ip->dev, addr);
   a2 = (uint *)bp2->data;
   if ((addr = a2[index]) == 0)
   {
     a2[index] = addr = balloc(ip->dev);
     log_write(bp2);
   }

   brelse(bp);
   brelse(bp1);
   brelse(bp2);

   return addr;
 }
```

TINDIRECT도 DINDIRECT와 비슷하게 3번을 배열을 타고 들어가야 한다. 그래서 bp, bp1, bp2 의 block pointer 변수가 3개가 있고, 배열이 a, a1, a2가 3개가 있다. index변수가 배열을 이동할 때마다, 128로 나누기 연산, 나머지 연산을 통해 조정이 된다. 그래서 마지막 배열 a2까지 타고 들어가서 addr을 받아서 리턴해준다.

```
if (ip->addrs[DOUBLE_INDIRECT])
  {
    bp = bread(ip->dev, ip->addrs[DOUBLE_INDIRECT]);
    a = (uint *)bp->data;
    for (i = 0; i < NINDIRECT; i++)
    {
      if (a[i])
      {
        bp1 = bread(ip->dev, a[i]);
        a1 = (uint *)bp1->data;
        for (j = 0; j < NINDIRECT; j++)
        {
          if (a1[j])
            bfree(ip->dev, a1[j]);
        }
        brelse(bp1);
        bfree(ip->dev, a[i]);
      }
    }
    brelse(bp);
    bfree(ip->dev, ip->addrs[DOUBLE_INDIRECT]);
    ip->addrs[DOUBLE_INDIRECT] = 0;
  }
```

다음은 itrunc를 구현했다. 위 캡처는 itrunc함수에서 DINDIRECT을 구현한 부분이다. 만약 ip->addrs[DOUBLE_INDIRECT]값이 0이 아니라면 쓰였다는 걸 의미하기 때문에 bp와 a로 outer 배열을 지정하고 그 배열에서 NINDIRECT만큼 돌고 만약 a[i]가 0이 아니라면 또 bp1과 a1으로 inner 배열을 타고 들어가서 있다면 bfree로 해제를 해주고 a[i]도 역시 해제를 해주고 마지막에서는 ip->addrs[DOUBLE_INDIRECT]또한 해제를 해주고 0으로 바꾸어주었다.

```
if (ip->addrs[TRIPLE_INDIRECT])
  {
    bp = bread(ip->dev, ip->addrs[TRIPLE_INDIRECT]);
    a = (uint *)bp->data;
    for (i = 0; i < NINDIRECT; i++)
    {
      if (a[i])
      {
        bp1 = bread(ip->dev, a[i]);
        a1 = (uint *)bp1->data;
        if (a1)
        {
          for (j = 0; j < NINDIRECT; j++)
          {
            if (a1[j])
            {
              bp2 = bread(ip->dev, a1[j]);
              a2 = (uint *)bp2->data;
              if (a2)
              {
                for (k = 0; k < NINDIRECT; k++)
                {
                  if (a2[k])
                    bfree(ip->dev, a2[k]);
                }
              }
              brelse(bp2);
              bfree(ip->dev, a1[j]);
            }
          }
        }
        brelse(bp1);
        bfree(ip->dev, a[i]);
      }
    }
    brelse(bp);
    bfree(ip->dev, ip->addrs[TRIPLE_INDIRECT]);
    ip->addrs[TRIPLE_INDIRECT] = 0;
  }
  ip->size = 0;
  iupdate(ip);
}
```

위 캡처는 itrunc함수에서 TINDIRECT을 지우는 캡처이다. DINDIRECT와 비슷하게 이번에는 3중for문을 순회하면서 총 배열을 2개를 타고 들어가서 마지막 배열에서 bfree를 해주고 또한 배열을 나오면서도 bfree를 호출해서 0이 아닌 값들에 대해서는 모두 해제를 해준다.

-SYMBOLIC LINK

```c
#include "types.h"
#include "stat.h"
#include "user.h"

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

```

ln.c에서 커맨드를 -h와 -s를 받게끔 고쳐주고 -s라면 softlink함수를 불러준다.

```c
int sys_softlink(void)
{
  char *new, *old;
  struct inode *ip;

  if (argstr(0, &old) < 0 || argstr(1, &new) < 0)
    return -1;

  begin_op();
  ip = create(new, SOFT_LINK, 0, 0);
  if (ip == 0)
  {
    end_op();
    return -1;
  }
  memmove((char*)(ip->addrs),old,strlen(old));
  iupdate(ip);
  iunlockput(ip);
  end_op();

  return 0;
}
```

그 후 sysfile.c에서 sys_softlink라는 system call함수를 만들어준다. Old와 new로 쓰일 파일 인자들을 받아서 type을 SOFT_LINK로 하는 inode를 하나 만들고 그 inode의 addrs에 old값을 memmove함수로 넣어준다. 즉 addrs에는 link시키고 싶은 파일의 이름이 담겨있다.

```c
int readi(struct inode *ip, char *dst, uint off, uint n)
{
  uint tot, m;
  struct buf *bp;

  if (ip->type == SOFT_LINK)
  {
    ip = namei((char *)(ip->addrs));
    if (ip==0) {
      cprintf("fail to open new file : old file removed\n");
      return -1;
    }
    ilock(ip);
    iunlock(ip);
  }
```

그 후 readi에서 new파일을 읽을 때 type이 SOFT_LINK였기 때문에 저 분기에서 걸린다. Inode pointer의 addrs에 있는 파일의 이름을 namei의 인자로 주어 원본 파일의 inodepointer를 받는다. 만약 원본파일이 삭제되었다면 ip는 0이 되어서 cprintf문으로 예외처리를 해주었다.

Project3 > xv6-public > C stat.h > ...

```c
#define T_DIR  1   // Directory
#define T_FILE 2   // File
#define T_DEV  3   // Device
#define SOFT_LINK 4

struct stat {
  short type;  // Type of file
  int dev;     // File system's disk device
  uint ino;    // Inode number
  short nlink; // Number of links to file
  uint size;   // Size of file in bytes
};
```

Ip의 type을 SOFT_LINK로 해주어야 하기 때문에 SOFT_LINK를 stat.h에 선언해주었다.

```
105  extern int sys_uptime(void);
106  extern int sys_softlink(void);
107
108  static int (*syscalls[])(void) = {
109  [SYS_fork]    sys_fork,
110  [SYS_exit]    sys_exit,
111  [SYS_wait]    sys_wait,
112  [SYS_pipe]    sys_pipe,
113  [SYS_read]    sys_read,
114  [SYS_kill]    sys_kill,
115  [SYS_exec]    sys_exec,
116  [SYS_fstat]   sys_fstat,
117  [SYS_chdir]   sys_chdir,
118  [SYS_dup]     sys_dup,
119  [SYS_getpid]  sys_getpid,
120  [SYS_sbrk]    sys_sbrk,
121  [SYS_sleep]   sys_sleep,
122  [SYS_uptime]  sys_uptime,
123  [SYS_open]    sys_open,
124  [SYS_write]   sys_write,
125  [SYS_mknod]   sys_mknod,
126  [SYS_unlink]  sys_unlink,
127  [SYS_link]    sys_link,
128  [SYS_mkdir]   sys_mkdir,
129  [SYS_close]   sys_close,
130  [SYS_softlink] sys_softlink,
131  };
```

```
27  SYSCALL(dup)
28  SYSCALL(getpid)
29  SYSCALL(sbrk)
30  SYSCALL(sleep)
31  SYSCALL(uptime)
32  SYSCALL(softlink)
```

```
20  #define SYS_link   19
21  #define SYS_mkdir  20
22  #define SYS_close  21
23  #define SYS_softlink 22
```

```
24  int sleep(int);
25  int uptime(void);
26  int softlink(const char*, const char*);
27
```

Syscall.c, syscall.h, usys.S, user.h에 systemcall을 구현하기 위한 작업들을 해준다.

Sync-구현 X

Result

Multi Indirect 테스트에 쓰인 testcode이다.

```c
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"

#define N 100
#define SIZE 1024
#define FSIZE 16 * 1024 * 1024
struct test {
    char file[SIZE];
};
char buf[512];

void
save(char* filename)
{
    int fd;
    struct test t;
    for (int i = 0; i < SIZE; i++) t.file[i] = '1';
    fd = open(filename, O_CREATE | O_RDWR);
    if (fd >= 0) {
        printf(1, "Create Success\n");
    }
    else {
        printf(1, "Error: Create failed\n");
        exit();
    }

    int size = sizeof(t);
    printf(1, "[%d]", size);
    for (int i = 0; i < 1024; i++) {
        for (int j = 0; j < 16; j++) {
            if (write(fd, &t, size) != size) {
                printf(1, "Error: Write failed\n");
                exit();
            }
        }
    }
    printf(1, "write ok\n");
    close(fd);
}

void
load(char* filename)
{
    int fd;
    struct test t;
    fd = open(filename, O_RDONLY);
    if (fd >= 0) {
        printf(1, "Open Success\n");
    }
    else {
        printf(1, "Error: open failed\n");
        exit();
    }

    int size = sizeof(t);
    if (read(fd, &t, size) != size) {
        printf(1, "Error: read failed\n");
        exit();
    }
    printf(1, "Read Success\n");
    close(fd);
}
void
printFile(int fd, char* name, int line)
{
    int i, n; //here the size of the read chunk is defined by n, and i is used to keep a track of the chunk index
    int l, c; // here line number is defined by l, and the character count in the string is defined by c

    l = c = 0;

    while ((n = read(fd, buf, sizeof(buf))) > 0)
    {
        for (i = 0;i <= n;i++)
        {
            //print the characters in the line
            if (buf[i] != '\n') {
                printf(1, "%c", buf[i]);
            }
            //if the number of lines is equal to l, then exit
            else if (l == (line - 1)) {
                printf(1, "\n");
                exit();
            }
            //if the number of lines is not equal to l, then jump to next line and increment the value of l
            else {
                printf(1, "\n");
                l++;
            }
        }
    }

    if (n < 0) {
        printf(1, "printFile: read error\n");
        exit();
    }
}

int
main(void)
{
    //exit();
    char filename[5] = "test";
    char n = '0';
    for (int i = 0; i < 4; i++) {
        printf(2, "now %d\n", i);
        n = i + '0';
        filename[4] = n;
        save(filename);
        //printFile(fd, filename, 10);
        load(filename);
        if (unlink(filename) < 0) printf(1, "unlink fail");
    }
    exit();
}
```

위 테스트 코드는 DOUBLE INDIRECT, TRIPLE INDIRECT 수의 크기만큼 FILE을 쓰고 읽는 테스트 코드이다.

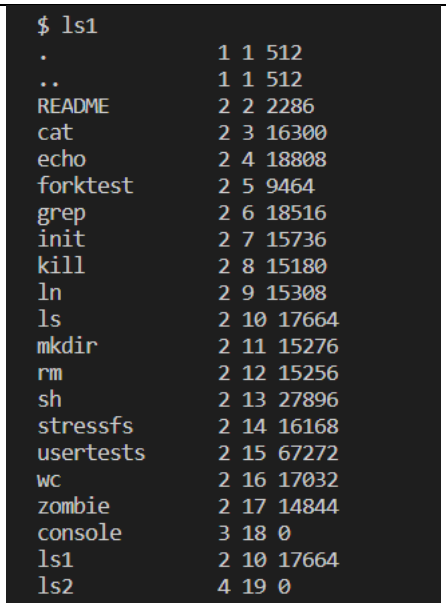
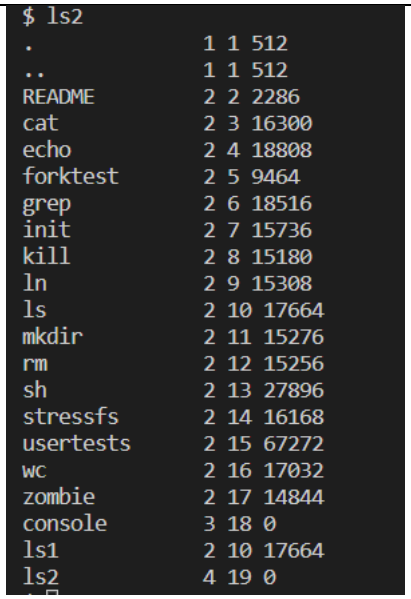
```
now 0
Create Success
[1024]write ok
Open Success
Read Success
now 1
Create Success
[1024]write ok
Open Success
Read Success
now 2
Create Success
[1024]write ok
Open Success
Read Success
now 3
Create Success
[1024]write ok
Open Success
Read Success
$
```

정상적으로 종료된 모습이다.

-SYMBOLIC LINK

```
$ ln -h ls ls1
$ ln -s ls1 ls2
$
```

먼저 ln -h와 ln -s 로 ls를 그대로 하드링크한 ls1과 softlink한 ls2를 만든다.

	
Ls1을 실행한 모습(hardlink로 만들어짐)	Ls2를 실행한 모습(softlink로 만들어짐)

```
$ rm ls1
$ ls2
fail to open new file : old file removed
exec ls2 failed
$
```

Ls1을 지우고 ls1을 바로가기한 ls2를 실행시켰을 때 exec failed하는 모습과 예외처리해준 모습을 볼 수 있다.

Trouble Shooting

Symbolic link를 구현할 때 readi함수에서 SOFT_LINK를 확인하고 ip->addrs의 inode로 ip를 바꿔주기만 하면 ip가 수정이 될 줄 알았는데 계속 안되서 헤매던 중 ilock과 iunlock을 해주었더니 ip가 반영이 되어서 softlink가 제대로 되었다.