Operation System Assignment 3 Wiki

2017033654 류동원

1. 디자인 및 구현

A. Multi Indirect

```
// 10개 direct block + 1개 indirect + 1개 Double indirect + 1개 Triple indirect
#define NDIRECT 10
#define NINDIRECT (BSIZE / sizeof(uint))
#define NDBINDIRECT (NINDIRECT * NINDIRECT)
#define NTRPINDIRECT (NINDIRECT * NINDIRECT * NINDIRECT)
#define MAXFILE (NDIRECT + NINDIRECT + NDBINDIRECT + NTRPINDIRECT)

// 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
};
```

i. direct block 10개 / indirect block 1개 / double indirect block 1개 / triple indirect block 1개를 구현하기 위해 fs.h 파일을 수정한다.

```
struct inode {
  uint dev;           // Device number
  uint inum;          // Inode number
  int ref;            // Reference count
  struct sleeplock lock; // protects everything below here
  int valid;          // inode has been read from disk?

  short type;         // copy of disk inode
  short major;
  short minor;
  short nlink;
  uint size;
  uint addrs[NDIRECT+3];
```

ii. inode 구조체의 addrs 멤버도 수정해준다.

```
#define FSSIZE       2200000  // size of file system in blocks
```

iii. param.h 파일의 파일 시스템의 블럭 사이즈 수도 수정해준다.

iv. bmap

1. double indirect와 triple indirect를 구현하기 위해 fs.c 파일의 bmap 함수를 수정한다.

2. bmap 함수는 struct inode 포인터와 파일 블럭 번호(bn)를 파라미터로 받고, 해당 파일 블럭과 mapping된 파일 시스템 블럭의 번호를 찾아 반환하는 역할을 한다.

3. bn < NDIRECT 일 경우, bn < NINDIRECT 일 경우, bn < NDBINDIRECT 일 경우, bn < NTRPINDIRECT 일 경우를 순차적으로 탐색해서 알맞은 블럭 매핑 정보를 찾는다.

4. single indirect block은 addrs[NDIRECT]], double indirect block은 addr[NDIRECT+1], triple indirect block은 addr[NDIRECT+2]의 블럭을 사용한다.

5. 몫(/)가 나머지(%) 연산을 이용하여 double indirect와 triple indirect의 적절한 블럭을 찾는다.

```
// Double Indirect
if(bn < NDBINDIRECT){
  // Load indirect block, allocating if necessary.
  if((addr = ip->addrs[NDIRECT+1]) == 0)
    ip->addrs[NDIRECT+1] = 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);
  }
  brelse(bp);

  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 -= NDBINDIRECT;
```

```
// Triple Indirect
if(bn < NTRPINDIRECT){
  // Load indirect block, allocating if necessary.
  if((addr = ip->addrs[NDIRECT+2]) == 0)
    ip->addrs[NDIRECT+2] = addr = balloc(ip->dev);

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

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

  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;
}
```

B. Symbolic Link

i. symbolic link는 new 파일의 경로를 old 파일의 부모 디렉토리의 inode에 저장함으로써 구현을 하였다.

```
// in-memory copy of an inode
struct inode {
  uint dev;           // Device number
  uint inum;          // Inode number
  int ref;            // Reference count
  struct sleeplock lock; // protects everything below here
  int valid;          // inode has been read from disk?

  short type;         // copy of disk inode
  short major;
  short minor;
  short nlink;
  uint size;
  uint addrs[NDIRECT+3];

  char symlink_data[60];  // symbolic link의 path를 위한 field 추가
};
```

ii. 우선, inode 구조체에 symbolic link의 파일 경로를 저장할 수 있는 symmlink_data 멤버를 추가한다.

iii. 그 후, fs.c 파일에 symlink 시스템 콜 함수를 구현하였다.

```
// symbolic link 생성
int
symlink(char *new, char *old)
{
  char name[DIRSIZ];
  struct inode *dp;
  cprintf("error1\n");

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

  begin_op();

  // dp는 부모 디렉토리의 inode
  if((dp = nameiparent(new, name)) == 0)
    goto bad;
  ilock(dp);

  // symbolic link path 저장
  strncpy(dp->symlink_data, new, strlen(new));

  iunlockput(dp);
  end_op();
  cprintf("error2\n");
  return 0;

bad:
  cprintf("error3\n");
  end_op();
  return -1;
}
```

iv. symlink 시스템 콜 함수는 old 파일의 부모 디렉토리 inode(변수 dp)의 symlink_data 멤버에 new 파일을 저장함으로써 symbolic link를 구현한다.

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

v. 그리고, ln.c 파일에서 4개의 argument를 받고 ln -h old new 이면 기존의 hard link를 생성하고 symbolic -s old new이면 방금 구현한 symbolic link를 생성하는 symlink 시스템 콜이 호출되도록 구현한다.

```
int
sys_symlink(void)
{
  char *arg1, *arg2;

  if(argstr(0, &arg1) < 0){
      return -1;
  }

  if(argstr(1, &arg2) < 0){
      return -1;
  }

  return symlink(arg1, arg2);
}
```

vi. 이후, sysproc.c 파일에 symlink 시스템 콜의 wrapper function을 구현한다.

```
int symlink(char *, char *);
```

vii. user.h 파일에 symlink 시스템 콜 함수를 선언해주어 ln.c 파일에서 호출될 수 있도록 한다.

```
#define SYS_symlink 22
```

viii. syscall.h 파일에 symlink 시스템 콜 함수 번호를 정의한다.

```
extern int sys_symlink(void);
[SYS_symlink] sys_symlink,
```

ix. syscall.c 파일에 symlink 시스템 콜 함수를 선언한다.

```
SYSCALL(symlink)
```

x. usys.S 파일을 다음과 같이 수정해준다.

2. 결과

A. Multi indirect 테스트 수행하지 않음

B. symbolic link 테스트 수행하지 않음

3. Trouble shooting

A. Sync는 시간이 부족하여 구현하지 못하였다.