운영체제 Project3 Wiki

2019019016 서시언

<디자인 및 구현>

1. bmap (fs.c)

```
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[NDIRECT]) == 0)
    ip->addrs[NDIRECT] = 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 < NDOUBLE){
  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);
  }
  bn = bn % NINDIRECT;

  bp1 = bread(ip->dev,addr);
  a = (uint*)bp1->data;
  if((addr = a[bn]) == 0){
    a[bn] = addr = balloc(ip->dev);
    log_write(bp1);
  }
  brelse(bp);
  brelse(bp1);
  return addr;
}

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

- double indirect에서, 바깥쪽 블락의 0번째 블락 포인터가 가리키는 블락은, bn이 0~127인 데이터 블락을 가리키고, 1번째 블락 포인터가 가리키는 블락은, bn이 128~255인 데이터 블락을 가리키도록 한다.
    - 즉, 바깥쪽 블락에서의 index는 bn/128, 안쪽 블락에서의 index는 bn%128이다. (이때 bn은 NDIRECT와 NINDIRECT를 뺀 뒤의 값)

- bn >= NINDIRECT이면, bn -= NINDIRECT

- bn < NDOUBLE 이면
    - addr = ip -> addrs[NDIRECT+1]
        - ip -> addrs[NDIRECT+1]가 0이면 balloc을 통해 블락을 할당해준다.
    - bread를 통해 디스크에서 addr에 해당하는 블락(바깥쪽 블락)을 읽어와 bp에 저장한다.
    - a에 bp->data를 할당한다.
    - addr = a[bn/NINDIRECT]
        - a[bn/NINDRECT]가 0이면 balloc으로 블락을 할당해주고, log_write를 해준다.
    - bn에 bn%NINDIRECT을 저장한다.
    - bread를 통해 디스크에서 addr에 해당하는 블락(안쪽 블락)을 읽어와 struct buf type의 bp1에 저장한다.
    - a에 bp1->data를 할당한다.
    - addr = a[bn]
        - a[bn]이 0이면 balloc으로 블락을 할당해주고, log_write를 해준다.
    - brelse를 통해 bp, bp1을 release해주고, addr을 리턴한다.

2. itrunc (fs.c)

```
static void
itrunc(struct inode *ip)
{
  int i, j, k,l ;
  struct buf *bp, *bp1;
  uint *a, *b, addr;

  for(i = 0; i < NDIRECT; i++){
    if(ip->addrs[i]){
      bfree(ip->dev, ip->addrs[i]);
      ip->addrs[i] = 0;
    }
  }

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

  if(ip->addrs[NDIRECT+1]){
    bp = bread(ip->dev, ip->addrs[NDIRECT+1]);
    a = (uint*)bp->data;
    for(k=0; k <NINDIRECT; k++){
      if(a[k]){
        addr = a[k];
        bp1 = bread(ip->dev, addr);
        b = (uint*)bp1->data;
        for(l=0; l<NINDIRECT; l++){
          if(b[l])
            bfree(ip->dev, b[l]);
        }
        brelse(bp1);
        bfree(ip->dev, addr);
      }
    }
    brelse(bp);
    bfree(ip->dev, ip->addrs[NDIRECT+1]);
    ip->addrs[NDIRECT+1] = 0;
  }

  ip->size = 0;
  iupdate(ip);
}
```

- ip->addrs[NDIRECT+1]이면

  - for문을 돌면서 바깥쪽 블락이 포인팅하고 있는 모든 안쪽 블락에 대해, for문을 돌면서 안쪽 블락이 포인팅하고 있는 모든 데이터 블락들을 bfree를 통해 free해준다.

  - 데이터 블락들을 free하고 나면 안쪽 블락도 bfree를 통해 free해준다.

  - 안쪽 블락들도 모두 free하고 나면 double indirect (ip->addrs[NDIRECT+1])도 bfree를 통해 free해준다.

- ip->addrs[NDIRECT+1]이면

  - bread를 통해 bp에 ip->addrs[NDIRECT+1]에 해당하는 블락(바깥쪽 블락)을 가져온다.

  - a에 bp->data를 할당해준다.

  - for문을 돌면서 a[k]가 할당되어 있다면,

    - addr에 a[k]를 저장하고, bread를 통해 bp1에 addr에 해당하는 블락(안쪽 블락)을 가져온다.

    - b에 bp1->data를 할당해준다.

    - for문을 돌면서 b[l]이 할당되어 있다면, bfree를 통해 b[l]을 free해준다.

    - brelse를 통해 bp1을 release해주고, bfree를 통해 addr (a[k])에 해당하는 블락을 free해준다

  - brelse를 통해 bp를 release해주고, bfree를 통해 ip->addrs[NDIRECT+1]을 free해준 뒤, ip->addrs[NDIRECT+1]을 0으로 초기화 한다.

3. 매크로 상수값 변경

- fs.h

```
#define NDIRECT 11
#define NINDIRECT (BSIZE / sizeof(uint))
#define NDOUBLE (NINDIRECT * NINDIRECT)
#define MAXFILE (NDIRECT + NINDIRECT + NDOUBLE)
```

- NDIRECT를 11로 수정
- (NINDIRECT* NINDIRECT)값을 갖는 NDOUBLE을 추가
- MAXFILE을 (NDIRECT + NINDIRECT + NDOUBLE)로 수정

- param.h

```
#define NBUF          (MAXOPBLOCKS*3)  // size of disk block cache
#define FSSIZE        30000  // size of file system in blocks
```

- FSSIZE를30000으로수정

- inode, dinode구조체에 변경사항 반영

```
// 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+2];   // Data block addresses
};
```

```
// 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+2];
};
```

- struct dinode의 addrs[NDIRECT+1]를 addrs[NDIRECT+2]로 수정

- struct inode의 addrs[NDIRECT+1]을 addrrs[NDIRECT+2]로 수정

<실행결과>

- TEST1에서 파일에 8MB를 쓰고, TEST2에서는 8MB를 읽고, TEST3에서는 TEST1,2를 10회 반복하며 문제가 발생하지 않는지 확인한다.

```
$ file_test
Test 1: Write 8388608 bytes
Test 1 passed

Test 2: Read 8388608 bytes
Test 2 passed

Test 3: repeating test 1 & 2
Loop 1: 1.. 2.. ok
Loop 2: 1.. 2.. ok
Loop 3: 1.. 2.. ok
Loop 4: 1.. 2.. ok
Loop 5: 1.. 2.. ok
Loop 6: 1.. 2.. ok
Loop 7: 1.. 2.. ok
Loop 8: 1.. 2.. ok
Loop 9: 1.. 2.. ok
Loop 10: 1.. 2.. ok
Test 3 passed
All tests passed!!
$
```

<트러블슈팅>

- 아래 사진처럼 'main-loop: WARNING: i/o thread spun for 1000 iterations' 이 뜰 때도 있고 아닐 때도 있어서 문제가 되는 줄 알고, 조교님께 문의한 결과, 문제가 아니라는 것을 알게 됨.