Operating Systems
- Project03 Wiki–

2017029916
양동해

Ⅰ. Double indirect

1. 디자인

실습 수업시간에서도 보았듯, double indirect를 구현하기 위해선 우선 inode와 dinode 구조체를 먼저 수정해야한다. 수정 방법은 기존의 direct addr를 한 개 줄여서, addr12는 INDIRECT로, addr13은 새로 추가할 double indirect로 만들어 준다. 또한 문제에서도 요구되어 있듯이 FSSIZE 도 20000이상으로 변경 해야하기 때문에 param.h 파일도 수정해주어야 한다. 즉, 수정해야할 파일들은 다음과 같다.

=> param.h 파일(FSSIZE), file.h 파일(inode 구조체), fs.h(dnode 구조체)

그리고 구체적인 함수는 inode 상의 bn 번째 data block을 가져오는 bmap() 함수와, bmap()에서 할당해준 data block을 놓아주는 itrunc() 함수를 수정해야 한다. 이 함수들은 fs.c 파일에 존재하며, 기존에 이미 구현되어 있는 부분을 응용하여 만들었기 때문에 자세한 수정 방법은 "2. 구현" 파트에 기술하였다.

2. 구현

우선 param.h 파일내의 FSSIZE 값을 20000으로 수정했다.

```
#define NDEV          10  // maximum major device number
#define ROOTDEV        1  // device number of file system root disk
#define MAXARG        32  // max exec arguments
#define MAXOPBLOCKS   10  // max # of blocks any FS op writes
#define LOGSIZE      (MAXOPBLOCKS*3)  // max data blocks in on-disk log
#define NBUF         (MAXOPBLOCKS*3)  // size of disk block cache
#define FSSIZE       20000  // size of file system in blocks
```

[param.h 파일 안에, FSSIZE 값을 포함한 일부]

그 다음에 fs.h 파일내의 NDIRECT 값을 12에서 11로 변경해 주었다. NDIRECT는 "1. 디자인" 파트에서도 언급했듯이, direct addr의 갯수 이므로, 이를 하나 줄여 마지막 addr13을 double indirect로 바꾸기 위함이다. 그리고 double indirect의 갯수를 갖는 값인 NDBLINDIRECT를 추가해 주었다. 이 값은 indirect 갯수의 제곱 만큼의 값을 가진다. 마지막으로 double indirect 구현으로 증가된 파일 크기의 최대값인 MAXFILE을 수정해 주었다.

```
#define NDIRECT 11 // 12: SINGLE INDIRECT, 13: DOUBLE INDIRECT
#define NINDIRECT (BSIZE / sizeof(uint))
#define NDBLINDIRECT (NINDIRECT * NINDIRECT) // NINDIRECT의 제곱 개 만큼 들어갈 수 있음
#define MAXFILE (NDIRECT + NINDIRECT + NDBLINDIRECT) // MAXFILE 크기 증가
```

[fs.h 파일 안에, define된 값]

위에서 NDIRECT를 1 줄였기 때문에, dinode와 inode의 addr 배열의 크기도 수정해 주었다. 원래 13개의 값을 가져야 하므로, 단순히 1 증가만을 통해 이를 구현할 수 있다. inode도 dinode와 동일한 과정이므로 사진은 dinode 구조체로만 대체한다. inode 구조체는 file.h 파일에서 확인할 수 있다.

```
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 // 11로 줄였기 때문에 1 증가
};
```

[fs.h 파일 안에, dinode 구조체]

본격적으로 fs.c 파일에서 bmap() 함수를 수정했다. 수정은 이미 구현되어 있던 bmap() 함수의 하단부분에 로직을 추가하는 것으로 구현했으며, 수정된 모습은 아래 사진과 같다.

```
bn -= NINDIRECT;

if(bn < NDBLINDIRECT){
  // Load indirect block, allocating if necessary.
  if((addr = ip->addrs[NDIRECT+1]) == 0) // 13번째 addr 체크
    ip->addrs[NDIRECT+1] = addr = balloc(ip->dev);
  bp = bread(ip->dev, addr);
  a = (uint*)bp->data;

  if((addr = a[bn/NINDIRECT]) == 0){ // NINDIRECT로 나누어 첫 번째 level 구분 (block number와 같은 개념)
    a[bn/NINDIRECT] = addr = balloc(ip->dev);
    log_write(bp);
  }
  brelse(bp);

  // 두 번째 level 구분하기 위해 bp 다시 bread 진행
  bp = bread(ip->dev, addr);
  a = (uint*)bp->data;

  if((addr = a[bn%NINDIRECT]) == 0){ // NINDIRECT로 나눈 나머지를 통해 두 번째 level 구분 (block offset과 같은 개념)
    a[bn%NINDIRECT] = addr = balloc(ip->dev);
    log_write(bp);
  }
  brelse(bp);

  return addr;
```

[fs.c 파일 안에, bmap() 함수의 일부]

우선 맨 윗 부분을 보면 bn을 indirect 갯수만큼 빼주는 것을 볼 수 있다. 이 부분은 기존에 bn이 direct와 indirect 수의 합보다 큰 값을 가져야만 수행되기 때문에, double indirect의 시작점을 세팅하기 위해서 NINDIRECT를 빼주었다.

다음 부분은 bn이 double indirect 갯수보다 작은 경우에만 실행할 수 있게 if 문으로 묶은 것이며, 안의 내용은 기존에 indirect를 처리하는 방식과 유사하다.

addr에 13번째 addr가 올 수 있게 addr[NDIRECT+1]을 할당하고 bp와 a를 세팅해 주었다. 여기서 a는 data block의 배열을 의미하는데, 이 배열은 첫 번째 level의 block이므로 후에 진짜 data block이 존재하는 두 번째 level의 block을 찾아야 한다. 우선 첫 번째 level의 block을 찾는 방법은 a[bn/NINDIRECT]로 찾을 수 있는데, indirect의 갯수(128)로 나누어 주면 첫 번째 level의 block이 무엇인지 알 수 있다.

그렇게 찾은 block에서 다시 한번 진짜 data가 담겨있는 block을 찾는 과정을 거치는데, 이는 a[bn%NINDIRECT]로 찾을 수 있다. bn을 NINDIRECT로 나눈 나머지가 offset의 개념이 되어서 그 위치를 알아낼 수 있게 된다.

다음으로 fs.c 파일에서 itrunc() 함수를 수정했다. 수정은 이미 구현되어 있던 itrunc() 함수의 하단부분에서 로직을 추가하는 것으로 구현했으며, 수정된 모습은 아래 사진과 같다.

```
if(ip->addrs[NDIRECT+1]){ // 13번째 addr 체크
  bp = bread(ip->dev, ip->addrs[NDIRECT+1]);
  a = (uint*)bp->data;
  for(k = 0; k < NINDIRECT; k++){ // 첫 번째 level
    if(a[k]){
      bpp = bread(ip->dev, a[k]);
      aa = (uint*)bpp->data;
```

```
      for(l = 0; l < NINDIRECT; l++){ // 두 번째 level
        if(aa[l])
          bfree(ip->dev, aa[l]);
      }
      brelse(bpp);
      bfree(ip->dev, a[k]);
    }
  }
  brelse(bp);
  bfree(ip->dev, ip->addrs[NDIRECT+1]);
  ip->addrs[NDIRECT+1] = 0;
}
```

[fs.c 파일 안에, itrunc() 함수의 일부]

이 부분은 기존에 12번째 addr를 모두 free 해주고 나면 진행되는 코드인데, 앞서 구현한 double indirect가 들어가 있는 13번째 addr를 free하는 부분이다. 기존에 짜여진 코드와 방식은 매우 유사하며, 2-level로 되어 있는 구조를 free하기 위해 이중 for문으로 구현했다. bp는 inode의 13번째 addr이고, bpp는 첫 번째 level의 block의 addr 하나를 의미한다. 여기서 for문으로 indirect 갯수(128) 만큼 매번 안에 있는 data를 free해주고, 자기 자신의 두 번째 level에 있는 모든 data block이 free가 되면 자기 자신도 free 하는 구조이다. 이렇게 13번째 addr가 모두 free가 되면 비로소 addr13이 초기화 될 수 있게 구현했다.

나머지 테스트 프로그램을 통해 결과를 확인할 수 있게, file_test.c 파일을 추가해 주었고 이와 관련된 부분을 Makefile에서 수정해 주었다. 이 부분에 대한 자세한 코드는 file_test.c 파일과 Makefile 파일을 통해 확인할 수 있다.

3. 실행 결과

이를 실행하기 위해, make clean, make, make fs.img를 차례로 수행하고 xv6를 실행시키면 다음과 같은 결과가 나타난다. (권한 설정만 해준다면, xv6의 실행은 실습시간에 만든 bootxv6.sh의 사용으로도 가능하다.)

```
$ 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!!
```

[xv6에서 file_test 실행 결과]

Test1은 8MB 정도의 파일을 쓰는 과정이고, Test2는 그 파일을 읽는 과정이다. 그리고 Test3은 Test1과 Test2의 과정을 10번 반복 실행한다. 시간은 다소 걸렸지만 모두 성공적으로 pass한 것을 볼 수 있다.

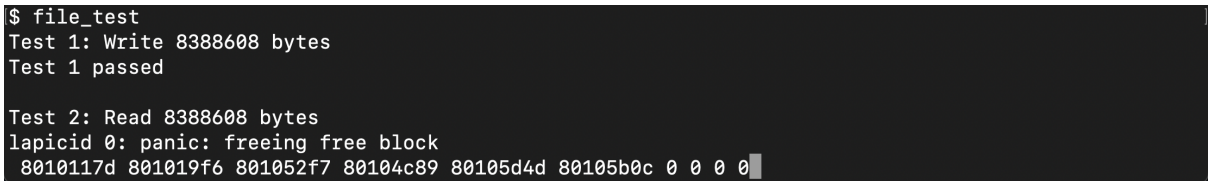
4. 트러블슈팅

문제 명세와 실습 강의를 통해, 사전지식이 충분히 숙지 된 상황에서 프로젝트를 진행하였기 때문에 특별한 트러블 이슈는 존재하지 않았다. 다만 코드를 작성하는 과정 중 오류가 조금 있었는데, 다음 사진은 처음 테스트 프로그램을 실행했을 때 받은 결과이다.

[xv6에서 file_test 실행 결과(수정 전 결과)]

처음에는 어떤 부분에서 코드를 잘못 작성했는지 알지 못했지만, 여러가지 파일을 탐색해보던 중 fs.h 파일내의 MAXFILE 값이 변경되어야 한다는 것을 깨달았다. 기존에는 NDIRECT + NINDIRECT 값만 가졌지만, 추가적으로 double indirect를 구현했기 때문에 NDBLINDIRECT를 더해주어야 한다. 이를 더함으로써 위 에러는 해결했다.

두 번째로는 사소한 코딩 실수로 나온 에러이다. 이도 처음에는 에러만 보고 당황했지만, 코드를 리뷰하던 중 변수명을 잘못 쓴 경우로, 바로 해결할 수 있었다. 다음 사진은 그 때 나온 에러 사진이다.



[xv6에서 file_test 실행 결과(수정 전 결과)]