운영체제 project3

2018009125 조성우

3-1 Multi indirect

1 Multi indirect의 이해

inode에서는 파일의 정보를 저장하기 위해 direct와 single, double, triple indirect를 사용할 수 있습니다. Direct를 바로 data블록을 가리키는 것이고, single은 다른 A index block를 한번 가리키고 A index block은 data블록을 가리키게 됩니다. Double은 A index block를 가리키고 A index block는 B index block를 가리키고 B index block가 data블록을 가리키게 됩니다. Triple은 여기서 index block를 가리키는 한번의 과정을 더 추가하면 됩니다. Xv6에서는 direct와 single indirect만을 사용합니다. 따라서 direct로 가리키는 data블록 12개와 하나의 single indirect inode로 가리키는 128개의 data블록 총 140개의 data블록에 저장할 수 있습니다. 하나의 data블록은 512B를 저장할 수 있으므로 총 저장공간은 70kb가 됩니다. 여기에 double indirect를 추가하면 128*128개의 블록을 추가로 저장할 수 있게 되고 triple indirect를 추가하면 128 * 128 *128개의 블록을 추가로 저장할 수 있게 되며 대략 1GB가 됩니다.

2. 코드 구현

우선 param.h에 있는 FSSIZE부터 조정하였습니다.

```
9   #define MAXARG       32  // max exec arguments
10  #define MAXOPBLOCKS  10  // max # of blocks any FS op writes
11  #define LOGSIZE      (MAXOPBLOCKS*3)  // max data blocks in on-disk log
12  #define NBUF         (MAXOPBLOCKS*3)  // size of disk block cache
13  #define FSSIZE       2097152 // size of file system in blocks(cover up to triple indirect) 2097152
14
15
```

Triple indirect는 block의 개수가 128 * 128 * 128 = 2097152개 필요하므로 triple indirect까지 커버할 수 있도록 FSSIZE를 설정하였습니다.

```
24  #define NDIRECT 10
25  #define NINDIRECT (BSIZE / sizeof(uint))
26  #define DNINDIRECT (128 * 128)
27  #define TNINDIRECT (128 * 128 * 128)
28  #define MAXFILE (NDIRECT + NINDIRECT + DNINDIRECT + TNINDIRECT)
29
30  // On-disk inode structure
31  struct dinode {
32    short type;           // File type
33    short major;          // Major device number (T_DEV only)
34    short minor;          // Minor device number (T_DEV only)
35    short nlink;          // Number of links to inode in file system
36    uint size;            // Size of file (bytes)
37    uint addrs[NDIRECT+3];   // Data block addresses , NDIRECT : single, NDIRECT+1 : double, NDIRECT+2 : triple indirect
38  };
```

그 다음은 fs.h입니다. NDIRECT를 본래 12에서 10으로 줄였습니다. 따라서 direct의 개수가 10개로 줄었습니다. 그 대신 addrs[NDIRECT]가 single indirect inode를 addrs[NDIRECT+1]이 double indirect inode를 addrs[NDIRECT+2]가 triple indirect inode를 가리키게 하였습니다. 그리고 26줄,

27줄에서 새로이 DNINDIRECT와 TRINDIRECT를 정의하였습니다. 각각은 double과 triple에서 사용되는 data block의 개수입니다. 또한 MAXFILE을 triple indirect까지 커버할 수 있도록 변경하였습니다.

다음은 fs.c에 있는 bmap 함수입니다.

```
372  static uint
373  bmap(struct inode *ip, uint bn)
374  {
375    uint addr, i_bn, ii_bn, iii_bn, *a, *a1, *a2;
376    struct buf *bp, *bbp, *bbbp;
377
378    if(bn < NDIRECT){
379      if((addr = ip->addrs[bn]) == 0)
380        ip->addrs[bn] = addr = balloc(ip->dev);
381      return addr;
382    }
383    bn -= NDIRECT;
384
385    if(bn < NINDIRECT){
386      // Load indirect block, allocating if necessary.
387      if((addr = ip->addrs[NDIRECT]) == 0)
388        ip->addrs[NDIRECT] = addr = balloc(ip->dev);
389      bp = bread(ip->dev, addr);
390      a = (uint*)bp->data;
391      if((addr = a[bn]) == 0){
392        a[bn] = addr = balloc(ip->dev);
393        log_write(bp);
394      }
395      brelse(bp);
396      return addr;
397    }
398    bn -= NINDIRECT;
```

375줄에 여러 변수들을 새로 선언하였고 나머지 부분은 원래 xv6에 있던 single indirect까지 구현된 코드입니다. Single indirect를 보면 single indirect block(addr[NDIRECT])이 존재하지 않으면 balloc을 통해 블록을 새로 할당합니다. 그리고 그 블록을 bp에 bread를 통해 읽어 들이고 a를 통해 data를 가리킵니다. a는 index block입니다. a[bn]을 통해 data block을 지정하고 해당 data block이 할당이 되어있지 않다면 새로 할당하고 해당 data block을 return 합니다. 여기 까지가 bn 이 NDIRECT + NINDIRECT보다 작을 때의 상황입니다. Block의 index bn이 NDIRECT + NINDIRECT보다 작다면 direct 와 single indirect를 통해 커버가 가능합니다. 다음은 double 과 triple의 코드입니다. Double과 triple 코드를 작성할 때 single indirect를 참고하여 동일한 매커니즘으로 작성하였습니다. 반복에 반복이라고 보시면 됩니다.

```
398     bn -= NINDIRECT;
399
400     if(bn < DNINDIRECT){
401       i_bn = bn/128;
402       ii_bn = bn % 128;
403       // Load double indirect block, allocating if necessary.
404       if((addr = ip->addrs[NDIRECT+1]) == 0)
405         ip->addrs[NDIRECT+1] = addr = balloc(ip->dev);
406       bp = bread(ip->dev, addr);
407       a = (uint*)bp->data;
408       if((addr = a[i_bn]) == 0){
409         a[i_bn] = addr = balloc(ip->dev);
410         log_write(bp);
411       }
412       bbp = bread(ip->dev, addr);
413       a1 = (uint*)bbp->data;
414       if((addr = a1[ii_bn]) == 0){
415           a1[ii_bn] = addr = balloc(ip->dev);
416           log_write(bbp);
417       }
418       brelse(bbp);
419       brelse(bp);
420       return addr;
421     }
422     bn -= DNINDIRECT;
```

먼저 double indirect를 다루는 코드입니다. 먼저 i_bn을 보면 bn/128을 했습니다. 이는 두번째 index block을 정하기 위한 index입니다. 그리고 128로 나눈 나머지인 ii_bn은 data block을 정하기 위한 index입니다. 처음에 addrs[NDIRECT+1]을 확인하고 할당되어 있지 않다면 새로 할당합니다. 그리고 single indirect와 마찬가지로 진행하다가 첫번째 index block a에서 i_bn번째 index를 확인합니다. 마찬가지로 할당되어 있지 않다면 새로 할당합니다. 이제 a[i_bn]은 두번째 index block을 가리키게 되었습니다. 첫번째 index block a를 찾는 것과 동일한 방법으로 두번째 index block a1을 찾은다음 a1[ii_bn]을 확인합니다. 이 위치가 return 할 data block입니다. 마지막으로 triple indirect입니다.

```
422     bn -= DNINDIRECT;
423
424     if(bn < TNINDIRECT){
425       i_bn = bn / DNINDIRECT;
426       ii_bn = (bn % DNINDIRECT) / 128;
427       iii_bn = (bn % DNINDIRECT) % 128;
428       ///Load triple indirect block, allocating if necessary.
429       if((addr = ip->addrs[NDIRECT+2]) == 0)
430         ip->addrs[NDIRECT+2] = addr = balloc(ip->dev);
431       bp = bread(ip->dev, addr);
432       a = (uint*)bp->data;
433       if((addr = a[i_bn]) == 0){
434         a[i_bn] = addr = balloc(ip->dev);
435         log_write(bp);
436       }
437       bbp = bread(ip->dev, addr);
438       a1 = (uint*)bbp->data;
439       if((addr = a1[ii_bn]) == 0){
440           a1[ii_bn] = addr = balloc(ip->dev);
441           log_write(bbp);
442       }
443       bbbp = bread(ip->dev, addr);
444       a2 = (uint*)bbbp->data;
445       if((addr = a2[iii_bn]) == 0){
446           a2[iii_bn] = addr = balloc(ip->dev);
447           log_write(bbbp);
448       }
449       brelse(bbbp);
450       brelse(bbp);
451       brelse(bp);
452       return addr;
453     }
454
455     panic("bmap: out of range");
456   }
```

Double과 과정이 동일합니다. 다만 한 번 더 index block(여기서는 a2)을 구한다는 점이 다릅니다.

다음은 itrunc함수입니다. Inode가 link가 사라졌을 때 작동하는 함수입니다.

```
463   static void
464   itrunc(struct inode *ip)
465   {
466     int i, j, k;
467     struct buf *bp, *bbp, *bbbp;
468     uint *a, *a1, *a2;
469
470     for(i = 0; i < NDIRECT; i++){
471       if(ip->addrs[i]){
472         bfree(ip->dev, ip->addrs[i]);
473         ip->addrs[i] = 0;
474       }
475     }
476
477     if(ip->addrs[NDIRECT]){                              //single INDIRECT
478       bp = bread(ip->dev, ip->addrs[NDIRECT]);
479       a = (uint*)bp->data;
480       for(j = 0; j < NINDIRECT; j++){
481         if(a[j])
482           bfree(ip->dev, a[j]);
483       }
484       brelse(bp);
485       bfree(ip->dev, ip->addrs[NDIRECT]);
486       ip->addrs[NDIRECT] = 0;
487     }
```

470줄부터 475줄 까지는 direct block을 free하는 구간이고 477부터 487은 single indirect block을 free하는 구간입니다.

```
488     if(ip->addrs[NDIRECT+1]){                                   //double INDIRECT
489       bp = bread(ip->dev, ip->addrs[NDIRECT+1]);
490       a = (uint*)bp->data;
491       for(j = 0; j < NINDIRECT; j++){
492         if(a[j]){
493           bbp = bread(ip->dev, a[j]);
494           a1 = (uint*)bbp->data;
495           for(i = 0; i < NINDIRECT; i++){
496             if(a1[i])
497               bfree(ip->dev, a1[i]);
498           }
499           brelse(bbp);
500           bfree(ip->dev, a[j]);
501         }
502       }
503       brelse(bp);
504       bfree(ip->dev, ip->addrs[NDIRECT+1]);
505       ip->addrs[NDIRECT+1] = 0;
506     }
```

Double indirect block을 free하는 코드입니다. 이중 for문을 통해 index block을 타고 들어가 첫번째 index block인 a에서 a[j]가 가리키고 있는 index block인 a1이 가리키고 있는 모든 block들을 먼저 free합니다. 그리고 그전 index block인 a[j]를 free합니다. 이를 j가 0부터 127까지 반복합니다. 그러면 double indirect에서 할당된 모든 block들을 free하게 됩니다. 다음은 triple indirect block을 free하는 코드입니다. 삼중 for문을 통해 진행하며 매커니즘은 위와 동일합니다.

```
508   if(ip->addrs[NDIRECT+2]){                                   //triple INDIRECT
509     bp = bread(ip->dev, ip->addrs[NDIRECT+2]);
510     a = (uint*)bp->data;
511     for(j = 0; j < NINDIRECT; j++){
512       if(a[j]){
513         bbp = bread(ip->dev, a[j]);
514         a1 = (uint*)bbp->data;
515         for(i = 0; i < NINDIRECT; i++){
516           if(a1[i]){
517             bbbp = bread(ip->dev, a1[i]);
518             a2 = (uint*)bbbp->data;
519             for(k = 0; k < NINDIRECT; k++){
520               if(a2[k])
521                 bfree(ip->dev, a2[k]);
522             }
523             brelse(bbbp);
524             bfree(ip->dev, a1[i]);
525           }
526         }
527         brelse(bbp);
528         bfree(ip->dev, a[j]);
529       }
530     }
531     brelse(bp);
532     bfree(ip->dev, ip->addrs[NDIRECT+2]);
533     ip->addrs[NDIRECT+2] = 0;
534   }
535   ip->size = 0;
536   iupdate(ip);
537 }
```

3. 실행결과

동일 xv6폴더 내에 myapp.c라는 구글링을 통해 얻은 테스트코드를 삽입하였습니다. 이 테스트 코

```
103  void
104  create_test(void)
105  {
106    createTestFile(1 KB);
107    createTestFile(4 KB);
108    createTestFile(15 KB);
109    createTestFile(1 MB);
110    createTestFile(8 MB);
111
112    closeAllFDs();
113  }
```

드는 create_test라는 함수를 호출하는데 이 함수는 각각 1kb, 4kb, 15kb, 1mb, 8mb용량의 파일을 생성합니다. 이 testcode를 실행한 결과

```
Create Test 0 Started!
Create Test 0 Finished!
Create Test 1 Started!
Create Test 1 Finished!
Create Test 2 Started!
Create Test 2 Finished!
Create Test 3 Started!
Create Test 3 Finished!
Create Test 4 Started!
Create Test 4 Finished!
```

시간이 좀 오래 걸리더라도 정상적으로 모든 크기의 파일이 생성된 것을 확인할 수 있었습니다.

4. trouble shooting

가장 어려웠던 점은 xv6 파일의 어느 부분을 고쳐야 하는가에 대한 문제였습니다. 고쳐야 되는 부분을 알고 나니 나머지 코드는 사실 반복적으로 수행만 하면 되는 것이라 수월했던 것 같습니다.

3-2 symbolic link

1.symbolic link 이해

Symbolic link는 바로가기 파일입니다. 원래 파일이 삭제되면 symbolic link는 의미가 없어집니다.

2. 코드 구현

먼저 stat.h에서

```
#define T_DIR  1   // Directory
#define T_FILE 2   // File
#define T_DEV  3   // Device
#define T_SYM  4   // Symbolic link
```

새로 T_SYM이라는 파일 type을 정의하였습니다. 그 다음은 sys_symlink 시스템콜입니다.

```
169  int
170  sys_symlink(void)
171  {
172    char *new, *old;
173    struct inode *ip;
174    if(argstr(0, &old) < 0 || argstr(1, &new) < 0)
175      return -1;
176    begin_op();
177    if((ip = create(new,T_SYM,0,0)) == 0){
178      end_op();
179      return -1;
180    }
181    memmove(ip->symname, old, sizeof(char) * strlen(old));
182
183    iupdate(ip);
184    iunlock(ip);
185
186    end_op();
187    return 0;
188
189  }
```

Sysfile.c에 구현을 하였습니다. 유의할 점은 create()함수를 사용하였고 T_SYM이라는 type을 사용하였다는 점입니다.

```
13  struct inode {
14    uint dev;           // Device number
15    uint inum;          // Inode number
16    int ref;            // Reference count
17    struct sleeplock lock; // protects everything below here
18    int valid;          // inode has been read from disk?
19
20    short type;         // copy of disk inode
21    short major;
22    short minor;
23    short nlink;
24    uint size;
25    uint addrs[NDIRECT+3];
26    char symname[30];         //name of symlinked file
27  };
```

그리고 file.h에 있는 inode에 symname을 설정하였습니다. Symlink에 있는 memmove를 통해ip->symname에 oldfile의 이름이 들어가게 됩니다. 그리고 이는 readi에서 재사용됩니다. 다음은 readi입니다.

```
554  readi(struct inode *ip, char *dst, uint off, uint n)
555  {
556    uint tot, m;
557    struct buf *bp;
558    if(ip->type == T_SYM){
559      if((ip = namei(ip->symname)) == 0){
560        return -1;
561      }
562    }
```

여기서 trick을 사용하였습니다. Ip의 type이 T_SYM이라면 ip->symname에 해당하는 inode를 가져

오게 됩니다. 그리고 나머지는 마치 그 inode에 해당하는 파일을 읽는 것처럼 코드가 진행됩니다.

사실 이 부분에서 코드가 잘 작동하지 않는 것 같습니다. 무슨 문제인지 시간이 없어서 파악이 덜된 상태입니다. 그리고 나머지는 ln.c입니다.

```
5   int
6   main(int argc, char *argv[])
7   {
8     char s[3];
9     s[0] = '-';
10    s[1] = 's';
11    if(argc != 4){
12      printf(2, "Usage: ln old new\n");
13      exit();
14    }
15    if(!strcmp(s, argv[1])){
16      if(symlink(argv[2],argv[3]) < 0)
17        printf(2, "link %s %s: failed\n", argv[2], argv[3]);
18    }
19    else{
20      if(link(argv[2], argv[3]) < 0)
21        printf(2, "link %s %s: failed\n", argv[2], argv[3]);
22    }
23
24    exit();
25  }
```

-s를 받으면 symlink와 연결하였고 그 외의 문자를 받으면 본래있던 link와 연결하였습니다.

3. 실행 결과

사실 제대로 작동하지 않아서 실행 결과는 따로 적지 않겠습니다.

4. trouble shooting

제가 확인한 바로는 readi에서 문제가 발생하였습니다. 아마 namei를 호출하는 과정에서 무슨 문제가 발생한 듯한데 파악하지 못했습니다. 그리고 좀 급하게 작성하여 설명이 부족한 점 양해 부탁드립니다.

Git에 마지막으로 수정한 버전인 latestproject03을 확인해주시면 됩니다. 감사합니다.