xv6 Double indirect

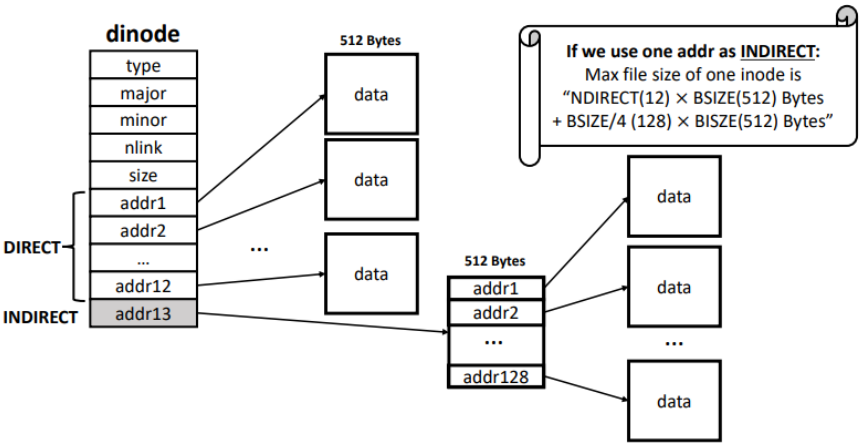
관련 과목	운영체제
사용언어	C
상태	완료
시작 날짜	@2021년 6월 10일 오후 7:54

1. 명세서

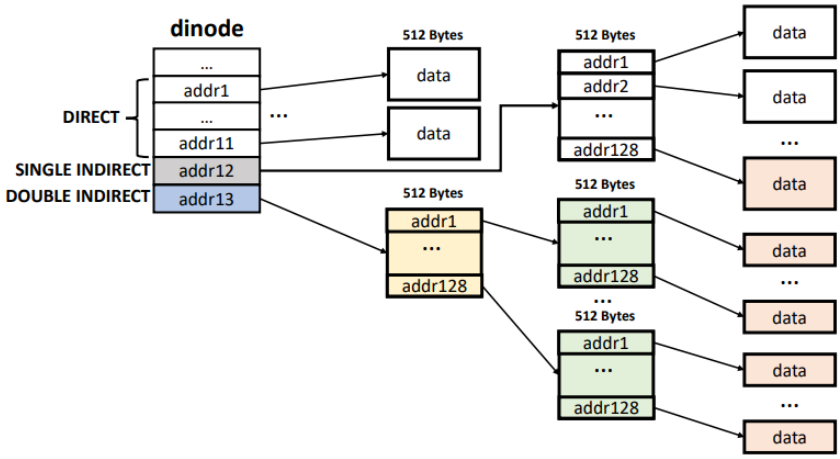
- xv6의 inode는 큰 파일을 다루기 위해 한 번의 indirect를 지원한다.

- 1개의 indirect를 사용하면 최대 140개의 블록에 해당하는 크기의 파일을 작성가능하다. 이를 2개의 indirect를 사용하는 방식으로 구현하여라

- 구현 전 original inode



- 구현 후 double indirect



- 수정해야 할 함수는 두가지 bmap과 itrunc이다.

- fs.h, param.h에 정의된 매크로 상수들의 값들을 일부 변경해야 한다.
> FSSIZE는 20000이상의 값 사용

- 변경 사항이 inode와 dinode 구조체에 바르게 반영되어야 한다.

2. Xv6 분석

A) bmap함수

`bmap` 함수는 인자로 받은 `inode` 상의 `bn` 번째 데이터 블록을 가져오는 함수이다. 만약 NDIRECT(1~12)에 해당하는 addr이라면 해당 addr에 저장된 bn번째 데이터블록을 참조하여 return하고, NDIRECT를 넘어가는 숫자(13~140)이라면 NDIRECT값을 빼주고 INDIRECT block으로 넘어가 해당 addr에 존재하는 데이터블록을 가져오게 된다. 만약 참고하고자 하는 블록 넘버가 NINDIRECT로 표현 가능한 범위를 넘어선다면 panic을 호출한다.

```
// fs.c
static uint
bmap(struct inode * ip, uint bn)
{
    uint addr, *a;
    struct buf *bp;
    if(bn < NDIRECT){
        if((addr = ip->addrs[bn])==0)
            ip->addrs[bn] = addr = balloc(ip->dev);
        return addr;
    }
    bn -= NDIRECT;

    if(bn < NINDIRECT){
        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;
    }
    panic("bmap : out of range");
}
```

B) itrunc함수

`itrunc` 는 inode가 가진 모든 data block을 deallocate하고 size를 0으로 바꿔준다. 해당하는 inode가 어떤 datablock도 가리키지 않도록 indirect block을 포함한 모든 addrs를 0으로 바꿔주며 각 블록을 `bfree` 를 통해 해제한다.
주의할 점은 deallocate의 순서가 다음과 같이 진행되어야 한다는 점이다.
1. direct data block을 해제
2. Indirect data block을 해제
3. 마지막으로 indirect block을 해제
>>만약 double indirect로 한다면 제일 안쪽 그러니까 double indirect datablock을 해제하고 double indirect block, double indirect block을 가리키는 block이 순으로 해제해야 할것으로 예상된다.

```
//fs.c
...
static void
itrunc(struct inode *ip)
{
    int i,j;
    struct buf *bp;
    uint *a;
    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;
    }
    ip->size =0;
    iupdate(ip);
...
}
```

3. 구현

A) fs.h와 param.h에 상수 변경

```
//fs.h
...
//구현 후 double indirect의 형태는 dinode 12번이 indirect block
// 13번이 double indirect block으로 되어있기 때문에
// NDIRECT값을 12에서 11로 수정하고 새로운 상수 NDINDIRECT 를 선언했다.
#define NDIRECT 11
#define NINDIRECT (BSIZE / sizeof(uint))
#define NDINDIRECT (BSIZE / sizeof(uint))*(BSIZE / sizeof(uint))
// double indirect 구성으로 인하여 최대 파일 크기가 수정되었기 때문에
// NDINDIRECT를 추가해준다.
#define MAXFILE (NDIRECT + NINDIRECT + NDINDIRECT)
//itrucnc에서 사용할 변수 BLOCKROW를 추가한다.
#define BLOCKROW (BSIZE / sizeof(uint))
...

//param.h
//명세서의 요구에 FSSIZE를 20000이상 사용하라고 나와있었기 때문에 변경
...
#define FSSIZE 20000
```

B) fs.c의 bmap수정

```
static uint
bmap(struct inode *ip, uint bn)
{
  uint addr, *a;
  struct buf *bp;
```

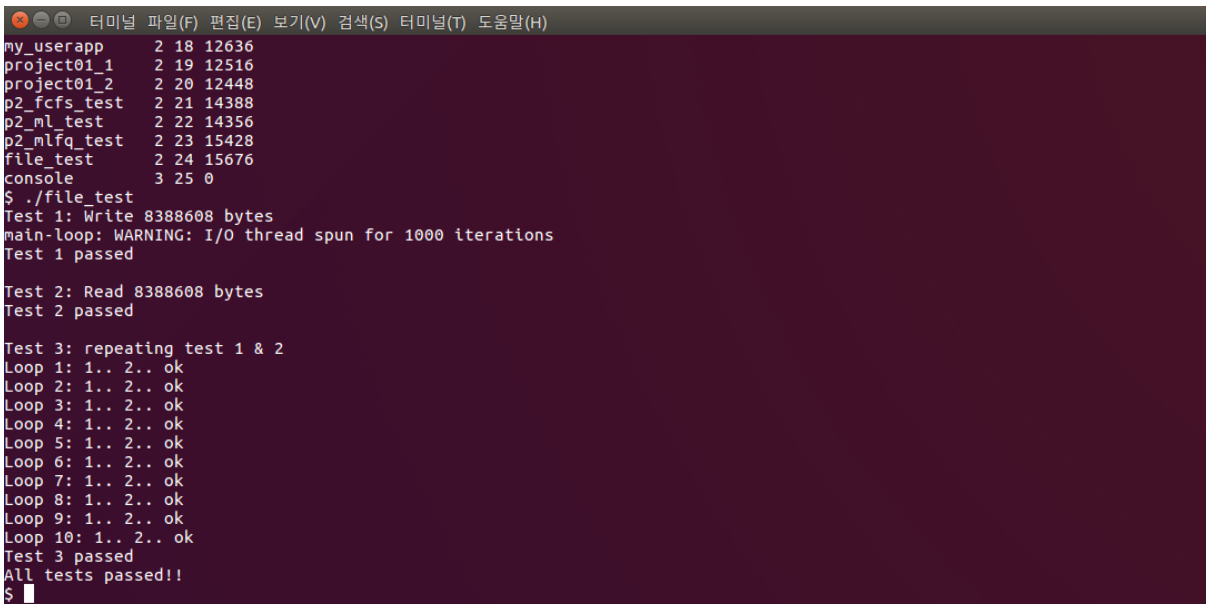
```
  if(bn < NDIRECT){
    if((addr = ip->addrs[bn]) == 0)
      ip->addrs[bn] = addr = balloc(ip->dev);
    return addr;
  }
  bn -= NDIRECT;

  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;
  }
//여기까진 동일하며 이 아래부터 새롭게 코드를 추가한다.
//bn 1~11은 direct로 탐색하고
//bn 12~139는 indirect를 탐색하는 코드가 위에 구현되어있다.
//위 코드를 응용하여 bn에서 NINDIRECT, 128을 빼주고
//그 값이 1~ 128*128 안에 있다면 탐색을 진행하도록 하자.
  bn -= NINDIRECT;

  //check double indirect block
//여기가 탐색부분이다.
//12~139를 탐색했던 코드를 응용하여 작성하였다.
  if(bn < NDINDIRECT){
//addrs[NDIRECT+1] , addrs[12]번째를 확인해 할당받고
    if((addr=ip->addrs[NDIRECT+1])==0)
      ip->addrs[NDIRECT+1] = addr = balloc(ip->dev);
    bp = bread(ip->dev, addr);
    a = (uint*)bp->data;
    //dinode -> double indirect block 1
//그 데이터 블록을 사용하여 첫 번째 indirect를 탐색한다.
//이 때 a에 접근할 때 128개 중에 몇번째 데이터 블록에 접근할것인가를
// 알아내기 위해 bn을 BLOCKROW로 나눈 몫의 값으로 접근한다.
    if((addr = a[bn/BLOCKROW])==0){
      a[bn/BLOCKROW]=addr=balloc(ip->dev);
      log_write(bp);
    }
    brelse(bp);
    //double indirect block 1 -> double indirect data block
//위에서 얻어낸 값으로 접근한다. 즉 두번째 indirect 블록에 접근하는 것이다.
//이때는 몫이 아닌 나머지 값을 사용하여 몇 번째 data block에 접근할 것인지 구한다.
    bp = bread(ip->dev,addr);
    a = (uint*)bp->data;
    if((addr = a[bn%BLOCKROW])==0){
      a[bn%BLOCKROW]=addr=balloc(ip->dev);
      log_write(bp);
    }
    brelse(bp);
//블록을 다 썼음을 알리고 addr을 반환한다.
    return addr;
  }

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

c) fs.c의 itrunc구현

```
//fs.c
//이 코드도 기존 코드를 활용했다.
static void
```

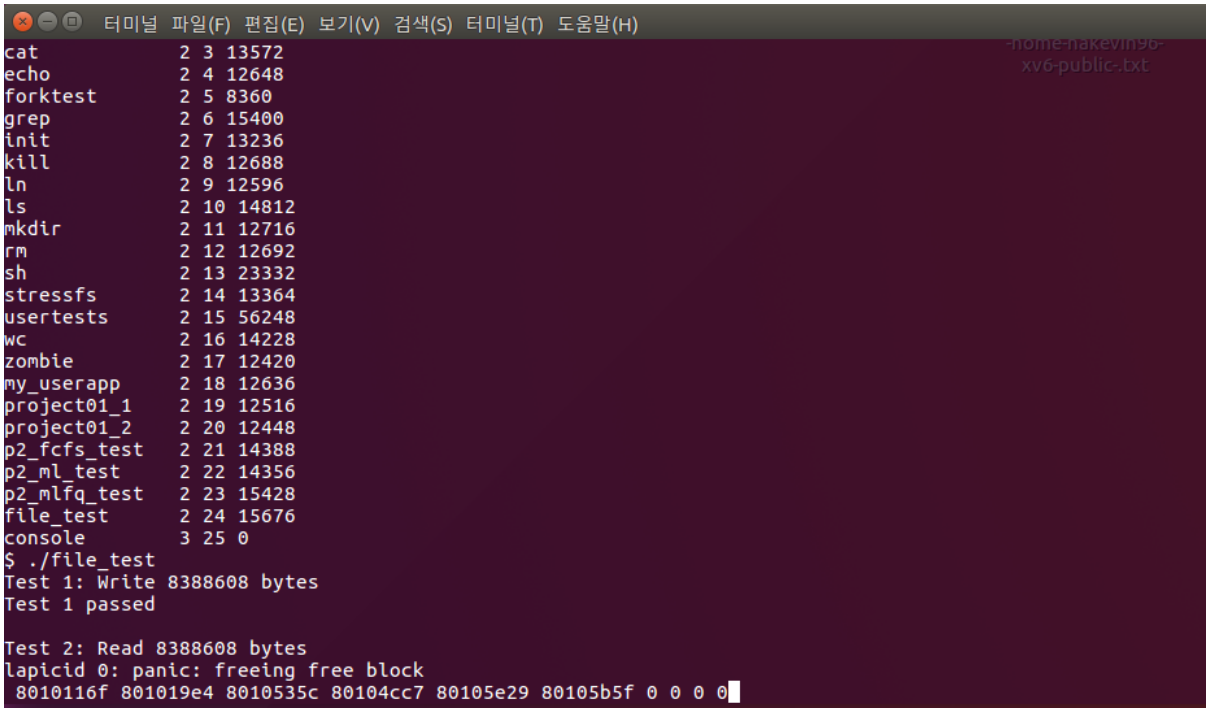
```
itrunc(struct inode *ip)
{
//루프 반복문을 위한 변수 k와 l을 추가하고
//double indirect block에 접근할 때 쓰기 위해 dbp와 da를 추가 선언하였다.
  int i, j, k, l;
  struct buf *bp;
  uint *a;
  //for double indirect block
  struct buf * dbp;
  uint * da;

  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;
  }

  //project 3 -> double indirect block itrunc
  //check last block NDIRECT + 1
//이 부분이 project 3를 진행할 때 itrunc에 추가한 부분이다.
//itrunc의 경우 반드시 삭제 과정을 지워야 한다고 했기 때문에
// 1. double indirect block 두 개 중 data block을 가리키는 안쪽 indirect block 삭제
// 2. 바깥쪽 double indirect block 삭제
// 3. dinode에 double indirect block을 가리키던 부분 0으로 변경
// 이 순으로 삭제를 진행하였다.

//만약 ip->addrs[NDIRECT+1], 즉 addrs[12]라 doble indirect block을 가리킨다면
//아래코드를 실행한다.
  if(ip->addrs[NDIRECT+1]){
//1차 접근
    bp = bread(ip->dev, ip->addrs[NDIRECT+1]);
    a = (uint*)bp->data;
//2차 접근 (1번째 double indirect block)
    for(k=0; k<NINDIRECT; k++){
      if(a[k]){
        dbp = bread(ip->dev, a[k]);
        da = (uint*) dbp->data;
//3차 접근 (2번째 double indirect block)
        for(l=0; l<NINDIRECT ; l++){
//(위 주석에서 1번 부분실행
          if(da[l])
            bfree(ip->dev,da[l]);
        }
        brelse(dbp);
//위 주석에서 2번 부분 진행
        bfree(ip->dev, a[k]);
      }
    }
    brelse(bp);
//위 주석에서 3번부분 진행
    bfree(ip->dev,ip->addrs[NDIRECT+1]);
    ip->addrs[NDIRECT+1]=0;
  }

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

4. 결과



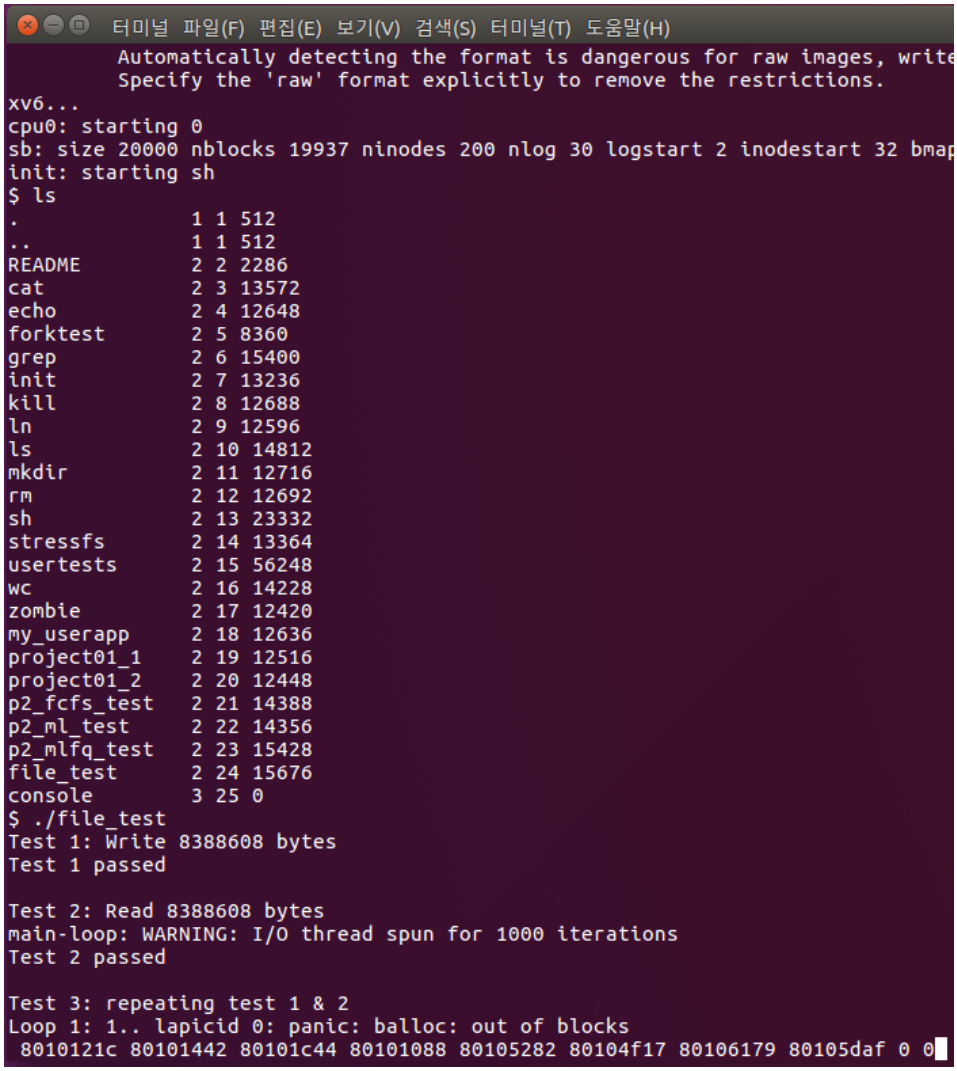
5. feedback

에러 1 : 한번 free해준 block을 또 free시켜서 발생한 에러



`bfree(ip->dev,ip->addrs[NDIRECT+1]);` 이 코드부분을 `bfree(ip->dev, a[k]);` 이렇게 작성하여 발생한 문제였다. 비슷한 구성이라 복사 붙여넣기로 수정하였더니 해결되었다. 발견은 bfree부분을 확인해가며 금방 할 수 있었다. 다음부터는 이런 실수를 하지 않아야겠다.

에러 2 : 원인 itrunc내 반복문 범위 NDIRECT로 설정해서 에러남



itrunc내에 I을 사용한 반복문에서 NINDIRECT까지 반복을 해서 지웠어야 했는데 실수로 NDIRECT까지라고 적어 컴파일시에는 문제가 없었으나 프로그램 동작시 에러가 발생했다. 한번 쓰고 읽는것 까지는 문제가 없었으나 해당 데이터블록을 제대로 free시켜주지 않아 balloc시에 문제가 발생하였다.
balloc은 block을 할당하는 함수이며 block들 중에 사용가능한 block을 block이 free인지 아닌지로 확인하기 때문에 itrunc내의 bfree부분을 다시 살펴보았고 잘못 코드를 작성했음을 발견할 수 있었다.