컴퓨터학부 20142385 엄재식

1. 과제 개요

Indirect block 구현

1. lbn\_to\_pbn 구현
2. inode\_write & inode\_read 구현
3. 상세설계
4. lbn\_to\_pbn
5. lbn이 1이하면 directblock을 바로 참조함

sn\_nlink를 검사해 directblock에 pbn이 할당되어 있는지 확인 후 할당한다.

1. lbn이 2이상이면 indirectblock을 이용

블록크기와 포인터크기를 이용해 indirectblock 당 할당 가능한 블록 수를 계산

할당 가능한 블록 수를 이용해 indirectblock 배열 인덱스를 구함

배열에 블록 pbn이 할당되어 있는지 확인 후 할당

인다이렉트 블록을 블록 버퍼로 읽은 뒤 lbn에 맞는 자리의 값을 읽는다

pbn이 할당되어 있는지 확인 후 할당

* + - a) 혹은 b) 를 수행한 뒤 pbn을 리턴

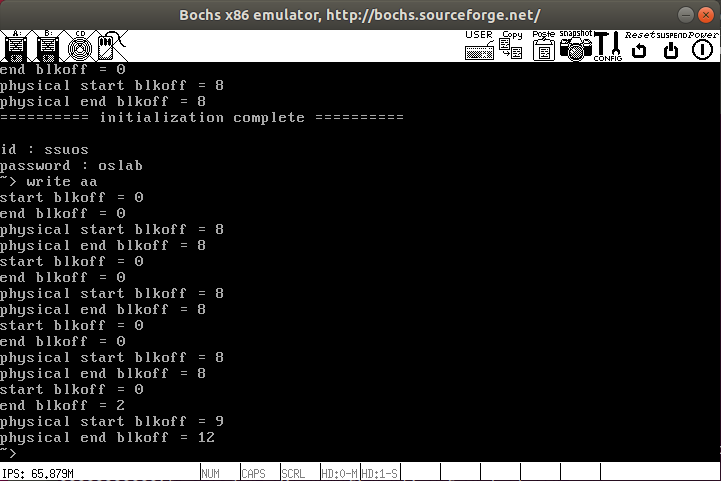
1. inode\_write
   * + lbn\_to\_pbn 함수를 이용해 pbn값을 구함
     + pbn 블록을 블록버퍼로 읽은 뒤 버퍼를 파라미터 버퍼에 맞게 수정
     + 수정한 블록버퍼를 pbn 블록에 복사
     + 수정한 길이만큼 len값을 감소

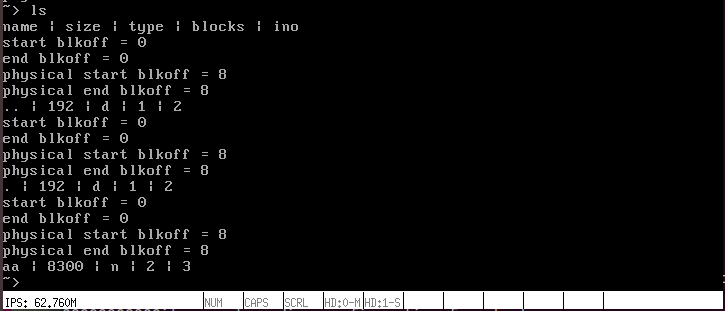
len이 0 보다 큰 동안 위의 과정을 반복

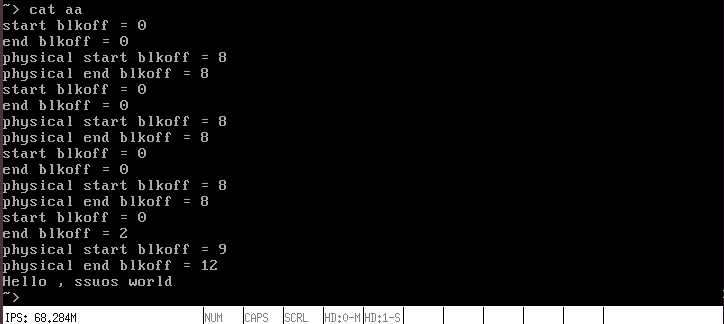
1. inode\_read
   * + lbn\_to\_pbn 함수를 이용해 pbn값을 구함
     + pbn 블록을 블록버퍼로 읽은 뒤 맞는 위치의 값을 파라미터 버퍼에 저장
     + 버퍼에 저장한 길이만큼 len값을 감소

len이 0보다 큰 동안 위의 과정을 반복

1. 실행결과







1. 소스코드
2. lbn\_to\_pbn

int lbn\_to\_pbn(struct inode \* in, uint32\_t lbn )  
{  
    struct ssu\_fs \* fs = in->sn\_fs;  
    struct bitmap \*bmap = fs->fs\_blkmap;  
    int pbn=0;  
    int indirect\_idx=0;  
    int idx=0;  
    int point\_idx=0;  
    if (lbn < 0)  
        return -1;  
    //complete this function for indirect block.  
    //1이하면 다이렉트 블럭  
    if (lbn < NUM\_DIRECT)  
    {  
        //할당유무 체크 후 할당  
        if (lbn >= in->sn\_nlink)  
        {      
            balloc(bmap, &(in->sn\_directblock[lbn]));  
            in->sn\_nlink++;  
            sync\_bitmapblock(fs);  
        }  
        pbn = in->sn\_directblock[lbn];  
    }  
    else  
    {      
        //인다이렉트 배열 인덱스와 인다이렉트 블럭의 인덱스   
        indirect\_idx = (lbn - NUM\_DIRECT) / (SSU\_BLOCK\_SIZE / sizeof(uint32\_t));  
        idx = (lbn - NUM\_DIRECT) % (SSU\_BLOCK\_SIZE / sizeof(uint32\_t));  
  
        if (indirect\_idx >= NUM\_INDIRECT)  
            return -1;  
        else if (indirect\_idx >= in->cnt\_data\_block) {  
            //인다이렉트 배열 블럭 할당  
            balloc(bmap, &(in->sn\_indirectblock[indirect\_idx]));  
            in->cnt\_data\_block++;  
            sync\_bitmapblock(fs);  
        }  
          
        //인다이렉트 블럭 읽기  
        memset(tmpblock\_indirect, 0, SSU\_BLOCK\_SIZE);  
        fs\_readblock(fs, in->sn\_indirectblock[indirect\_idx], tmpblock\_indirect);  
        memcpy(&pbn, &tmpblock\_indirect[idx\*sizeof(uint32\_t)], sizeof(uint32\_t));  
          
        if(pbn == 0)//블럭이 할당 안돼있을 경우  
        {  
            //블럭 할당  
            balloc(bmap, &tmpblock\_indirect[idx\*sizeof(uint32\_t)]);  
            fs\_writeblock(fs, in->sn\_indirectblock[indirect\_idx], tmpblock\_indirect);              
            sync\_bitmapblock(fs);  
        }  
          
        memcpy(&pbn, &tmpblock\_indirect[idx\*sizeof(uint32\_t)], sizeof(uint32\_t));  
          
    }  
  
    return pbn;  
}

1. inode\_write

int inode\_write(struct inode \*in, uint32\_t offset, char \* buf, int len)  
{  
  
    //modify the inode\_write to activate indirect blocks.  
    int result=0;  
    struct ssu\_fs \* fs = in->sn\_fs;  
    struct bitmap \*bmap = fs->fs\_blkmap;  
    uint32\_t blkoff = offset / SSU\_BLOCK\_SIZE;  
    uint32\_t start\_blkoff = blkoff;  
    uint32\_t res\_off = offset % SSU\_BLOCK\_SIZE;  
    int pbn;  
    int lenlen = len;  
      
    if(offset > in->sn\_size)  
        return -1;  
      
    while (len > 0) {  
        //피지컬 블럭 넘버 받아옴  
        pbn = lbn\_to\_pbn(in, blkoff);  
        memset(tmpblock, 0, SSU\_BLOCK\_SIZE);  
        //블럭 읽기          
        fs\_readblock(fs, pbn, tmpblock);  
        //블럭버퍼 수정  
        if (len >= SSU\_BLOCK\_SIZE - res\_off) {  
            memcpy(tmpblock + res\_off, buf, SSU\_BLOCK\_SIZE - res\_off);  
            len -= SSU\_BLOCK\_SIZE - res\_off;  
            buf += SSU\_BLOCK\_SIZE - res\_off;  
        }  
        else {  
            memcpy(tmpblock + res\_off, buf, len);  
            len = 0;  
        }  
        //블럭버퍼 블럭에 복사  
        fs\_writeblock(fs, pbn, tmpblock);  
        res\_off = 0;  
        blkoff++;  
    }  
    blkoff--;  
    if(in->sn\_size < offset+lenlen)  
        in->sn\_size = offset+lenlen;  
    sync\_inode(fs, in);  
      
    printk("start blkoff = %d\n", start\_blkoff);  
    printk("end blkoff = %d\n", blkoff);  
    printk("physical start blkoff = %d\n", lbn\_to\_pbn(in, start\_blkoff));  
    printk("physical end blkoff = %d\n", lbn\_to\_pbn(in, blkoff));  
    return result;  
    //  
}

1. inode\_read

int inode\_read(struct inode \* in, uint32\_t offset, char \* buf, int len)  
{  
  
    //modify the inode\_write to activate indirect blocks.  
    int result=0;  
    struct ssu\_fs \* fs = in->sn\_fs;  
    uint32\_t blkoff = offset / SSU\_BLOCK\_SIZE;  
    uint32\_t start\_blkoff = blkoff;  
    uint32\_t res\_off = offset % SSU\_BLOCK\_SIZE;  
    int pbn;  
    if(offset > in->sn\_size)  
        return -1;  
      
    while (len > 0) {  
        //피지컬 블럭 넘버 받아옴  
        pbn = lbn\_to\_pbn(in, blkoff);  
        memset(tmpblock, 0, SSU\_BLOCK\_SIZE);  
        //블럭 읽기          
        fs\_readblock(fs, pbn, tmpblock);  
        //블럭내용 버퍼에 저장  
        if (len >= SSU\_BLOCK\_SIZE - res\_off) {  
            memcpy(buf, tmpblock + res\_off, SSU\_BLOCK\_SIZE - res\_off);  
            len -= SSU\_BLOCK\_SIZE - res\_off;  
            buf += SSU\_BLOCK\_SIZE - res\_off;  
        }  
        else {  
            memcpy(buf, tmpblock + res\_off, len);  
            len = 0;  
        }  
        blkoff++;  
        res\_off = 0;  
    }  
    blkoff--;  
    printk("start blkoff = %d\n", start\_blkoff);  
    printk("end blkoff = %d\n", blkoff);  
    printk("physical start blkoff = %d\n", lbn\_to\_pbn(in, start\_blkoff));  
    printk("physical end blkoff = %d\n", lbn\_to\_pbn(in, blkoff));  
    return result;  
  
    //  
}