컴퓨터학부 20192403 박상철

1. 개요

할당시 가상 메모리 테이블에만 할당하고 사용시에 물리 메모리에 할당이 이루어지는 ssualloc을 구현함으로서 가상 메모리와 물리 메모리에 대한 개념을 깨우치고 xv6의 메모리 관리 구조에 대해 분석하는 과제입니다. 또한 가상메모리와 물리 메모리의 수를 파악하는 함수를 작성함으로써 메모리 관련 상수와 자료구조에 대해서도 분석할 수 있습니다.

두번째 과제는 직접 접근 블록과 간접 접근 블록 이중간접 접근 블록으로 이루어진 기존의 xv6에 삼중 간접 접근 블록을 추가하고 파일 시스템을 관리 할 수 있도록 구현하는 과제입니다. 해당 과제를 통해 inode의 개념과 파일시스템의 구조를 이해 할 수 있습니다.

2. 상세설계

2-1) ssualloc

Ssualloc 시스템 콜은 인자를 받아 가상 메모리의 파일 크기를 증가시키고 가상 메모리의 주소를 반환하도록 구현되어 있습니다. 따라서 가상 페이지만 증가하고 물리 메모리는 할당 되지 않습니다. 가상 메모리만 할당된 영역을 접근한다면 페이지 폴트가 발생하여 트랩으로 넘어갑니다. Trap에서는 T\_PGFLT일때 페이지폴트가 발생한 지점을 물리 주소로 받아와서 공간을 할당합니다.

2-2) ssufs

Inode와 dinode의 값과 전처리문 상수값을 과제 형식에 맞게 수정하고 bmap 함수를 수정합니다. 기존의 직접 간접만 있었는데 2중과 3중 간접 블록에 대한 값도 처리하도록 수정합니다. (bmap 함수는 주어진 inode에서 물리주소를 반환하는 함수입니다.)

3. 결과

ssualloc\_test

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

ssufs\_test

텍스트, 스크린샷, 메뉴, 폰트이(가) 표시된 사진

자동 생성된 설명

4. 소스코드

**ssualloc\_test**

trap.c (가장 핵심)

case T\_PGFLT:

fault\_a = rcr2(); // Page fault가 발생한 가상 주소를 가져옴

fault\_a = PGROUNDDOWN(fault\_a); // 가상 주소를 페이지의 시작 주소로 변경

mem = kalloc(); // 물리 메모리 할당

if(!mem) {

cprintf("out of memory\n");

break;

}

memset(mem, 0, PGSIZE);

// 페이지 테이블에 새로 할당한 물리 페이지를 매핑

mappages(myproc()->*pgdir*, (char\*)*fault\_a*, PGSIZE, V2P(mem), PTE\_W|PTE\_U);

break;

sysprog.c

int

sys\_getvp(void)

{

return myproc()->sz / PGSIZE;

}

int

sys\_getpp(void) {

struct *proc* \*p = myproc();

int res = 0;

for(*uint* i=0; i<myproc()->sz; i+=PGSIZE) {

*pte\_t* \*pte = walkpgdir(p->pgdir, (void \*)i, 0);

if(pte && (\*pte & PTE\_P))

res++;

}

return res;

}

int

sys\_ssualloc(void)

{

int addr; // 증가할 메모리 주소를 저장할 변수

int n; // 증가시킬 크기를 저장할 변수

// 시스템 콜에서 전달된 인자로부터 크기를 얻어옴

if (argint(0, &n) < 0 || n <= 0 || n%PGSIZE != 0) {

return -1; // 유효하지 않은 값을 입력받았을 때 -1을 반환

}

// 현재 프로세스의 주소 반환

addr = myproc()->sz;

// growproc 함수를 호출하여 프로세스의 크기를 조절

myproc()->sz+=n;

// 증가된 메모리의 시작 주소를 반환

return addr;

}

**ssufs\_test**

file.h

struct *inode* {

uint dev; // Device number

...

uint addrs[NDIRECT+3]; // addrs 배열 크기 변경

};

fs.c 핵심

static *uint*

bmap(struct *inode* \**ip*, *uint* *bn*)

{

*uint* addr, \*a;

struct *buf* \*bp;

// direct

if(*bn* < NDIRECT){

// 블록이 할당되어 있지 않다면 할당하고 반환

if((addr = *ip*->addrs[*bn*]) == 0)

*ip*->addrs[*bn*] = addr = balloc(*ip*->dev);

return addr;

}

// indirect

*bn* -= NDIRECT;

if(*bn* < NINDIRECT){

// 간접 블록을 로드하고 필요한 경우 할당

if((addr = *ip*->addrs[NDIRECT]) == 0)

*ip*->addrs[NDIRECT] = addr = balloc(*ip*->dev);

// 간접 블록을 읽어오기

bp = bread(*ip*->dev, addr);

a = (*uint*\*)bp->data;

// bn에 해당하는 블록이 할당되어 있지 않다면 할당하고 로그에 기록

if((addr = a[*bn*]) == 0){

a[*bn*] = addr = balloc(*ip*->dev);

log\_write(bp);

}

// 버퍼 캐시에서 간접 블록 해제

brelse(bp);

// 물리 블록 번호 반환

return addr;

}

*bn* -= NINDIRECT;

//double indirect

if( *bn* < NDOUBLY\_INDIRECT){

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);

}

brelse(bp);

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);

}

brelse(bp);

return addr;

}

*bn* -= NDOUBLY\_INDIRECT;

//triple indirect

if(*bn* < NTRIPLY\_INDIRECT){

if((addr = *ip*->addrs[NDIRECT+2]) == 0)

*ip*->addrs[NDIRECT+2] = addr = balloc(*ip*->dev);

bp = bread(*ip*->dev, addr);

a = (*uint*\*)bp->data;

if((addr = a[*bn*/(NDOUBLY\_INDIRECT)]) ==0){

a[*bn*/(NDOUBLY\_INDIRECT)] = addr = balloc(*ip*->dev);

log\_write(bp);

}

brelse(bp);

*bn* = *bn*%(NDOUBLY\_INDIRECT);

//2nd indirect layer

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);

}

brelse(bp);

//3rd indirect layer

bp = bread(*ip*->dev, addr);

a = (*uint*\*)bp->data;

//load doubly indirect block, allocating if necessary

if((addr = a[*bn*%(NINDIRECT)]) == 0){

a[(*bn*%NINDIRECT)] = addr = balloc(*ip*->dev);

log\_write(bp);

}

brelse(bp);

return addr;

}

// 범위를 벗어난 경우 panic

panic("bmap

fs.h

#define NDIRECT 6

#define NINDIRECT (BSIZE / sizeof(*uint*))

#define NDOUBLY\_INDIRECT NINDIRECT\*NINDIRECT

#define NTRIPLY\_INDIRECT NINDIRECT\*NINDIRECT\*NINDIRECT

#define MAXFILE (NDIRECT + NINDIRECT\*4 + NDOUBLY\_INDIRECT\*2 + NTRIPLY\_INDIRECT)

// 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+7]; // Data block addresses

};