

Operating System

HW 4

(라) 분반

IT대학 소프트웨어학부

20220221 소희연

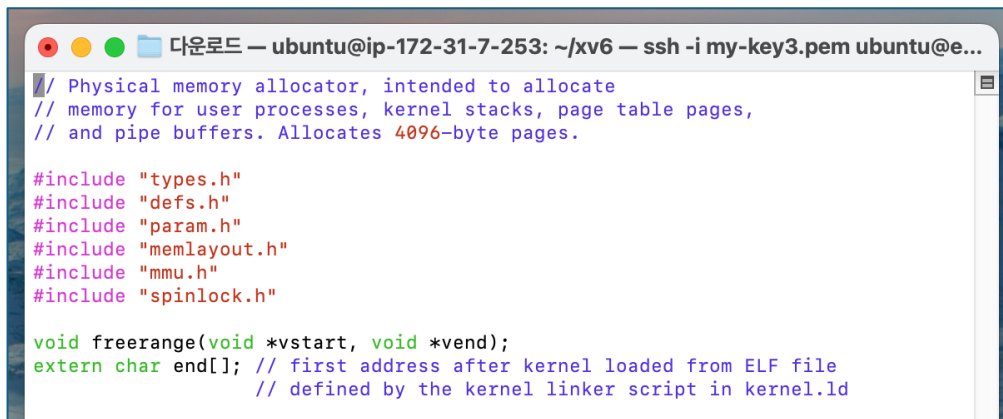
1. 개발 환경

- 운영체제
 - MacOS(M2)
- AWS E2C (Elastic Compute Cloud), Ubuntu Server 22.04 LTS
 - AWS에서 제공하는 가상 서버를 이용해 인스턴스를 생성하였고, 인스턴스 상에 Ubuntu 22.04를 올려 과제를 수행함
- SSH를 통해 로컬 터미널에서 xv6 구동

2. 단계별 구현 과정

2-1. getNumFreePages 시스템 콜 추가

kalloc.c를 수정하기 전에, 이 파일이 어떤 기능을 하는 지 알아보자. 이와 관련된 내용은 파일 최상단에 주석 처리 되어있다.



// 물리 메모리 할당자(Physical memory allocator)와 관련.

// 이 할당자는 사용자 프로세스, 커널 스택, 페이지 테이블 페이지, 파이프 버퍼에 필요한 메모리 할당.

// 메모리 할당 단위는 4096 byte(4KB)

전역변수 num_free_pages를 kinit1 함수 내에서 0으로 초기화였고, 위와 마찬가지로 주석 처리 된이 함수의 역할을 보면 다음과 같다.

```
// Initialization happens in two phases.
// 1. main() calls kinit1() while still using entrypghdir to place just
// the pages mapped by entrypghdir on free list.
// 2. main() calls kinit2() with the rest of the physical pages
// after installing a full page table that maps them on all cores.
void
kinit1(void *vstart, void *vend)
{
    initlock(&kmem.lock, "kmem");
    num_free_pages = 0;
    kmem.use_lock = 0;
    freerange(vstart, vend);
}
```

// 초기화는 다음의 두 단계로 나타남.

// 1. main 함수가 kinit1 함수를 호출. entrypghdir(초기 페이지 디렉터리)을 사용하여 물리 메모리 중 일부를 할당하고, 이 과정에서 entrypghdir에 매핑된 페이지들이 자유(free) 리스트에 추가됨.

// 2. main 함수가 kinit2를 호출. 나머지 물리 페이지들에 대해 동작. 모든 코어에서 사용할 수 있도록 전체 페이지 테이블을 설정한 후 호출.

명세에서 제시한대로 kalloc, kfree에 num_free_pages를 각각 --, ++ 조작을 하도록 추가하였다.

```
void
kfree(char *v)
{
    struct run *r;

    if((uint)v % PGSIZE || v < end || V2P(v) >= PHYSTOP)
        panic("kfree");

    // Fill with junk to catch dangling refs.
    memset(v, 1, PGSIZE);

    if(kmem.use_lock)
        acquire(&kmem.lock);
    r = (struct run*)v;
    r->next = kmem.freelist;
    kmem.freelist = r;

    ++num_free_pages;
    if(kmem.use_lock)
        release(&kmem.lock);
}
```

```
char*
kalloc(void)
{
    struct run *r;

    if(kmem.use_lock)
        acquire(&kmem.lock);
    r = kmem.freelist;
    if(r)
        kmem.freelist = r->next;
    num_free_pages--;
    if(kmem.use_lock)
        release(&kmem.lock);
    return (char*)r;
}
```

getNumFreePages 시스템 콜을 이전 과제에서 했던 것처럼 똑같은 절차에 따라 추가하였고, 시스템 콜의 원함수와 wrapping 함수는 kalloc.c 최하단에 추가하였다. (시스템 콜 추가 과정은 너무 길고, 이전 과제와 동일하기 때문에 보고서에서는 생략한다.)

```
int
getNumFreePages(void) {
    return num_free_pages;
}

int
sys_getNumFreePages(void) {
    return getNumFreePages();
}
```

2-2. 각 메모리 페이지들을 대상으로 reference counter 적용

‘uint pgrefcount[PHYSTOP >> PTXSHIFT];’를 추가하기 전에, 이게 어떤 실행을 하는지 명시적으로 알기가 어려워 찾아보았는데, 결과적으로 다음과 같은 결론에 도달했다.

- PGSHIFT
 - PGSHIFT는 12로 정의되어 있는데, 이는 4096바이트(4KB) 페이지 크기에 해당하는 비트 수이다.
 - PHYSTOP >> PGSHIFT는 물리 메모리 크기(PHYSTOP)를 4096바이트 단위로 나누어 몇 개의 페이지가 있는지 계산하는 연산이다.
 - 이 연산은 가상 주소에서 하위 12비트(offset)를 무시하고 상위 비트를 다룬다.
- PTXSHIFT와 PGSHIFT의 관계
 - 이미 코드 상에서 PTXSHIFT라는 이름으로 동일한 값(12)이 정의되어 있으며, 이 값은 페이지 테이블 관련 계산에 쓰이고, PGSHIFT와 역할이 겹친다.

결론적으로, PTXSHIFT와 PGSHIFT가 xv6 내에서 동일한 역할을 하기 때문에 PGSHIFT 을 별도로 사용하기 보다는 이미 있는 PTXSHIFT을 재사용 하는 게 더 효율적이라는 생각이 들었고, 후술할 과정에서도 이와 같이 수행하였다.

kalloc.c 최상단에 아래와 같이 전역변수 pgrefcount을 선언하였다

```
#include "types.h"
#include "defs.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "spinlock.h"

uint num_free_pages;
uint pgrefcount[PHYSHOP >> PTXSHIFT];
```

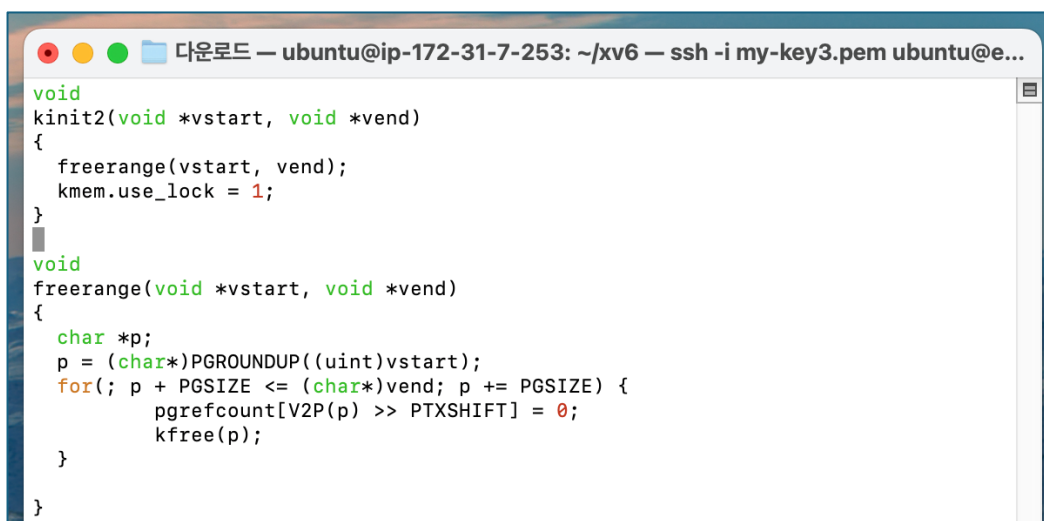
get_refcount는 물리 주소가 주어지면 pgrefcount[해당 주소]를 리턴하며, inc_refcount은 같은 상황에서 pgrefcount[해당 주소]를 하나 올린다. dec_refcount는 이와 반대의 역할을 한다. 이에 맞춰 kalloc.c 최하단에 아래와 같이 추가하였다.

```
uint
get_refcount(uint pa) {
    return pgrefcount[pa >> PTXSHIFT];
}

void
inc_refcount(uint pa) {
    ++pgrefcount[pa >> PTXSHIFT];
}

void
dec_refcount(uint pa) {
    --pgrefcount[pa >> PTXSHIFT];
}
```

다음으로는 kalloc.c 내에 있는 freerange 함수에서 pgrefcount 배열을 0으로 초기화 하였다.



```
void
kinit2(void *vstart, void *vend)
{
    freerange(vstart, vend);
    kmem.use_lock = 1;
}

void
freerange(void *vstart, void *vend)
{
    char *p;
    p = (char*)PGROUNDUP((uint)vstart);
    for(; p + PGSIZE <= (char*)vend; p += PGSIZE) {
        pgrefcount[V2P(p) >> PTXSHIFT] = 0;
        kfree(p);
    }
}
```

kfree 함수는 메모리를 해제할 때 사용하는 함수로, 기존 kfree 함수는 refcount를 고려하지 않고 실행됐으나, refcount가 0일 때만(아직 사용하는 곳이 남아 있지는 않은지 확인하는 절차) 안전하게 메모리를 해제하도록 아래와 같이 변경하였다.

```

void
kfree(char *v)
{
    struct run *r;

    if((uint)v % PGSIZE || v < end || V2P(v) >= PHYSTOP) {
        panic("kfree");
    }

    if(kmem.use_lock)
        acquire(&kmem.lock);

    if(get_refcount(V2P(v)) > 0) {
        dec_refcount(V2P(v));
    }

    if(get_refcount(V2P(v)) == 0) {
        memset(v, 1, PGSIZE);
        r = (struct run*)v;
        r->next = kmem.freelist;
        kmem.freelist = r;
        ++num_free_pages;
    }

    if(kmem.use_lock)
        release(&kmem.lock);
}

// Allocate one 4096-byte page of physical memory.

```

kalloc 함수에서 refcount를 1로 설정하였다.

```

// Returns a pointer that the kernel can use.
// Returns 0 if the memory cannot be allocated.
char*
kalloc(void)
{
    struct run *r;

    if(kmem.use_lock)
        acquire(&kmem.lock);
    r = kmem.freelist;

    if(r) {
        kmem.freelist = r->next;
        pgrefcount[V2P((char*)r) >> PTXSHIFT] = 1;
        num_free_pages--;
    }

    if(kmem.use_lock)
        release(&kmem.lock);
    return (char*)r;
}

```

2-3. copyvm 수정

2-2에서 했어야 할 inc_refcount 함수를 호출하는 것과 함께 주어진 명세에 맞게 copyvm 함수를 수정해야 한다.

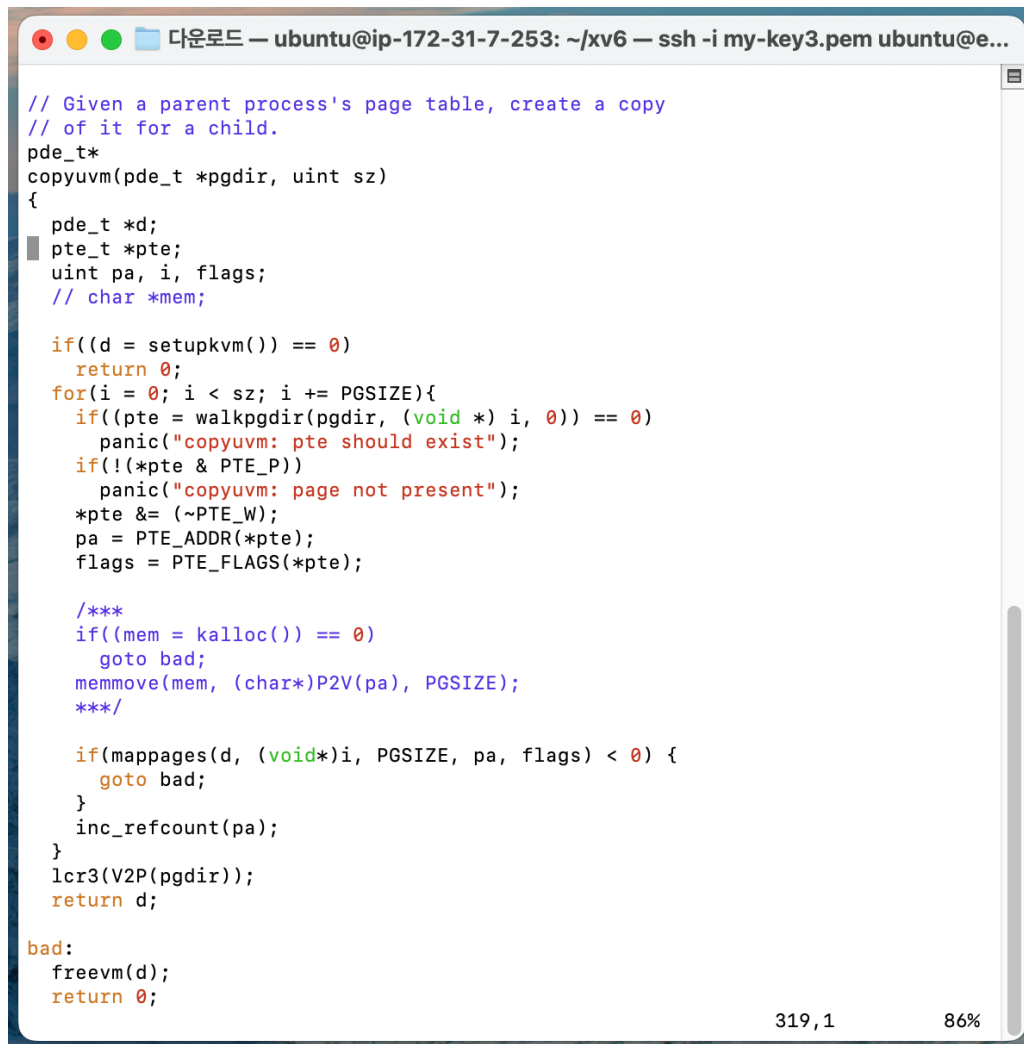
명세에 나온 것처럼 proc.c 내에 있는 fork 함수에서 copyvm을 호출하는 부분은 다음과 같다. 이 부분은 fork 함수에서 copyvm을 호출하여 부모 프로세스의 가상 메모리 구조를 자식 프로세스로 복사한다.

```
// Copy process state from proc.
if((np->pgdir = copyvm(curproc->pgdir, curproc->sz)) == 0){
    kfree(np->kstack);
    np->kstack = 0;
    np->state = UNUSED;
    return -1;
}
```

과제에서 제시한 copyvm 수정 조건을 하나씩 뜯어서 수정한 copyvm과 연결지어 설명하면 다음과 같다.

- 메모리 페이지를 생성하지 않고, 부모 프로세스의 메모리 페이지를 사용
 - 새로운 메모리 페이지를 할당하지 않고 $pa = PTE_ADDR(*pte)$ 를 통해 부모의 물리적 주소를 참조
 - Write 권한을 제거($pte \&= \sim PTE_W$)
- inc_refcount를 호출하여 reference counter 증가시키기
 - inc_refcount(pa)를 통해 해당 페이지의 참조 횟수를 증가시킴
- write 권한 제거 후 올바른 매핑
 - write 권한 제거 후, mappages 함수로 새로운 페이지 테이블 엔트리를 설정하
- 실제 데이터 복사 방지
 - memmove()를 제거하고 부모 페이지를 직접 사용

상기 내용에 따라 아래와 같이 수정하였고, 기존 코드에서 필요가 없는 부분은 주석처리했다.



```

// Given a parent process's page table, create a copy
// of it for a child.
pde_t*
copyuvm(pde_t *pgdir, uint sz)
{
    pde_t *d;
    pte_t *pte;
    uint pa, i, flags;
    // char *mem;

    if((d = setupkvm()) == 0)
        return 0;
    for(i = 0; i < sz; i += PGSIZE){
        if((pte = walkpgdir(pgdir, (void *) i, 0)) == 0)
            panic("copyuvm: pte should exist");
        if(!(*pte & PTE_P))
            panic("copyuvm: page not present");
        *pte &= (~PTE_W);
        pa = PTE_ADDR(*pte);
        flags = PTE_FLAGS(*pte);

        /**
        if((mem = kalloc()) == 0)
            goto bad;
        memmove(mem, (char*)P2V(pa), PGSIZE);
        ***/

        if(mappages(d, (void*)i, PGSIZE, pa, flags) < 0) {
            goto bad;
        }
        inc_refcount(pa);
    }
    lcr3(V2P(pgdir));
    return d;
bad:
    freevm(d);
    return 0;

```

319,1 86%

2-4. page fault handler 구현

traps.h에는 trapno가 14일 때 pagefault라고 정해 있으나, 기존 trap.c의 trap 함수에는 이와 관련한 switch가 나와있지 않아 아래와 같이 수정하였다.

```
//PAGEBREAK: 41
void
trap(struct trapframe *tf)
{
    if(tf->trapno == T_SYSCALL){
        if(myproc()->killed)
            exit();
        myproc()->tf = tf;
        syscall();
        if(myproc()->killed)
            exit();
        return;
    }

    switch(tf->trapno){
        case T_PGFLT:
            pagefault();
            break;
        case T_IRQ0 + IRQ_TIMER:
            if(cpuid() == 0){
                acquire(&tickslock);
                ticks++;
                wakeup(&ticks);
                release(&tickslock);
            }
            lapiceoi();
            break;
    }
```

이제 마지막으로 명세에서 제시한대로 pagefault 함수를 구현하면 다음과 같으며, 자세한 설명은 주석 처리 하였다.

```
void pagefault(void) {

    // rcr2() 함수를 호출하여 page fault 가 발생한 VA 읽어옴
    uint pgflt_va = rcr2();

    // va가 속한 pte를 가져옴
    struct proc* curproc = myproc();
    pte_t *pte = walkpgdir(curproc->pgdir, (void*)pgflt_va, 0);

    // pa와 rc를 얻음
    uint pa = PTE_ADDR(*pte);
    uint rc = get_refcount(pa);

    // reference counter 가 1 보다 큰 경우
    if(rc > 1){

        // child에 새로운 page 할당
        char* mem = kalloc();

        // 복사
        memmove(mem, (char*)P2V(pa), PGSIZE);

        *pte = V2P(mem) | PTE_P | PTE_U | PTE_W;

        // 새로운 페이지를 할당받은 후 refcount 감소 시킴
        dec_refcount(pa);
    }

    // reference counter 가 1 인 경우
    else if(rc == 1){
        //현재 pte에 write 권한만 추가
        *pte |= PTE_W;
    }
}
```

392,1

95%

2-5. test

t3.c 파일을 아래와 같이 만들었다.

```
다운로드 — ubuntu@ip-172-31-7-253: ~/xv6 — ssh -i my-key3.pem ubuntu...
#include "types.h"
#include "stat.h"
#include "user.h"

int main(void) {
    int p;

    printf(1, "Parent Process PID : %d\n", getpid());
    printf(1, "Step1. (Parent) Before fork\n");
    printf(1, "Available free pages : %d\n\n", getNumFreePages());

    p = fork();

    if(p < 0){
        printf(1, "fork error\n");
    }
    else if(p > 0){
        wait();

        printf(1, "Step3. (Parent) After child process terminates\n");
        printf(1, "Available free page : %d\n\n", getNumFreePages());
    }
    else{
        printf(1, "Child Process PID : %d\n", getpid());
        printf(1, "Step 2: (Child) After fork\n");
        printf(1, "Available free pages: %d\n\n", getNumFreePages());
    }

    exit();
}
```

실행 결과 정상적으로 작동하며, free page의 변화를 볼 수 있다.

```
다운로드 — ubuntu@ip-172-31-7-253: ~/xv6 — ssh -i my-key3.pem ubuntu...
dd if=kernel of=xv6.img seek=1 conv=notrunc
413+1 records in
413+1 records out
211656 bytes (212 kB, 207 KiB) copied, 0.000692078 s, 306 MB/s
/home/ubuntu/qemu/i386-softmmu/qemu-system-i386 -nographic -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ t3
Parent Process PID : 3
Step1. (Parent) Before fork
Available free pages : 56733

Child Process PID : 4
Step 2: (Child) After fork
Available free pages: 56665

Step3. (Parent) After child process terminates
Available free page : 56733

$
```

3. 문제점과 해결 방법

사실 이번 과제는 HW 3에 비해서는 복잡하지 않아 구현이 어렵지는 않았지만, 2-2 과정에서 PGSHIFT을 별도로 사용하지 않고 이미 있는 PTXSHIFT를 재활용 하는 걸로 결론을 짓는 과정에서 이 둘이 같은 수행을 하는 게 맞는지 정확히 파악하기가 힘들어 이와 관련된 내용을 xv6 공식 문서와 구글링 하여 찾는 과정이 제일 힘들었다.

결론적으로는 <https://oasess.tistory.com/44> 해당 링크에서 실마리를 찾을 수 있었다.