Project 3

Report

2017314331

김도엽

신동군 교수님

운영체제

0. mmap(), munmap(), freemem() 3가지 syscall 과 pagefault_handler의 구현

1) 모든 syscall에 사용된 함수는 pdf에서 설명하는 모든 case를 handle 하도록 구현되었습니다.

2) 코드의 설명은 코드 각 줄에 존재하는 주석을 통해 진행되었습니다.

1. 3가지 syscall을 추가하기 위한 기본 setting. 알파벳 순으로 각 파일에 추가 및 수정된 코드를 설명하겠습니다.

1) def.h에 관련 함수 prototype 추가.

```
// file.c
Int read_from_file_area(struct file *f, int offset, char *mem, int size);

// proc.c
int             add_mmap_area(struct proc *p, uint addr, int length, int prot,
int flags);
//sys_mmap.c
uint            mmap(uint, int, int, int, int, int);
int             munmap(uint);
int             freemem(void);

// vm.c
pte_t *         walkpgdir(pde_t *pgdir, const void *va, int alloc);
int             mappages(pde_t *pgdir, void *va, uint size, uint pa, int perm);
```

2) file.h에 file을 읽어오기 위한 함수 추가.

```
int read_from_file_area(struct file *f, int offset, char *mem, int size) {
  int original_offset = f->off;
  f->off = offset;
  int result = fileread(f, mem, size);
  f->off = original_offset;
  return result;
}
```

3) 기존의 file.h의 헤더파일을 전방선언하지 못해 컴파일 문제가 발생. 이 문제를 해결하기 위한 file1.h 을 만들어 file.h에 존재하는 struct file 구조체를 선언.

```
// file1.h
struct file {
    enum { FD_NONE, FD_PIPE, FD_INODE } type;
    int ref;
    char readable;
    char writable;
    struct pipe *pipe;
```

```
    struct inode *ip;
    uint off;
```

4) kalloc.c에 freemem 함수에서 쓰일 freed page를 세는 전역변수 선언.
memory allocation과 free가 일어날 때 해당 변수를 센다.

```
void
kfree(char *v)
{
//..기존 field
  free_page_count++; // 추가
}
void
kalloc(void)
{
//..기존 field
  free_page_count--; // 추가
}
```

5) param.h에 prot 변수, mmaping memory flag 변수 및 mmapbase 선언.

```
#define PROT_READ 0x1
#define PROT_WRITE 0x2
#define MAP_ANONYMOUS 0x1
#define MAP_POPULATE 0x2
#define MMAPBASE 0x40000000
#define MAX_MMAP_AREA 64
```

6) proc.c에 mmap에 필요한 함수 선언.
6-1) find_mmap_area : process와 page address를 이용하여 mmap_area를 search하는 함수.

```
struct mmap_area *find_mmap_area(struct proc *p, uint addr) {
  uint pg_addr = PGROUNDDOWN(addr);
  for (int i = 0; i < MAX_MMAP_AREA; i++) {
    struct mmap_area *area = &p->mmap_areas[i];
    if (area->addr <= pg_addr && pg_addr < area->addr + area->length) {
      return area;
    }
  }
  return 0;
}
```

6-2) allocproc() 에서 mmap_area initializing 진행.

```
static struct proc*
allocproc(void)
{
//.. written codes
for (int i = 0; i < MAX_MMAP_AREA; i++) {
    p->mmap_areas[i].f = 0;
    p->mmap_areas[i].addr = 0;
    p->mmap_areas[i].length = 0;
    p->mmap_areas[i].offset = 0;
    p->mmap_areas[i].prot = 0;
    p->mmap_areas[i].flags = 0;
    p->mmap_areas[i].p = 0;
  }
//..written codes
}
```

6-3) fork() 에서 자식 프로세스가 부모 프로세스의 mmap 을 복제

```
int
fork(void)
{

memmove(np->mmap_areas, curproc->mmap_areas, sizeof(curproc->mmap_areas));
  for (int i = 0; i < MAX_MMAP_AREA; i++) {
      if (np->mmap_areas[i].addr != 0) {
          np->mmap_areas[i].p = np;
          if (np->mmap_areas[i].f)
              filedup(np->mmap_areas[i].f);
      }
  }
}
```

7) proc.h에서 mmap area array 선언.

```
struct mmap_area {
  struct file *f;
  uint addr;
  int length;
  int offset;
  int prot;
  int flags;
  struct proc *p; // the process with this mmap_area
};
struct proc {
  //..written codes
       struct mmap_area mmap_areas[MAX_MMAP_AREA];
}
```

8) syscall.c 에서의 syscall list 삽입 및 syscall 함수 prototype 선언.

```
extern int sys_mmap(void);
extern int sys_munmap(void);
extern int sys_freemem(void);

static int (*syscalls[])(void) = {
[SYS_mmap]        = sys_mmap,
[SYS_munmap]      = sys_munmap,
[SYS_freemem]     = sys_freemem,
}
```

9) syscall.h syscall number 추가.

```
#define SYS_mmap 26
#define SYS_munmap 27
#define SYS_freemem 28
```

10) sysporc.c 에서 wrapper syscall 함수 등록을 위한 sys_mmap, sys_munmap, sys_freemem 함수 선언.

```
int
sys_mmap(void)
{
  //uint addr;
  int addr,length, prot, flags, fd, offset;

  if (argint(0, &addr) < 0 ||
      argint(1, &length) < 0 ||
      argint(2, &prot) < 0 ||
      argint(3, &flags) < 0 ||
      argint(4, &fd) < 0 ||
      argint(5, &offset) < 0)
    return -1;

  return (int)mmap(addr, length, prot, flags, fd, offset);
}

int
sys_munmap(void)
{
  uint addr;
  if (argint(0, (int *)&addr) < 0)
    return -1;
```

```
  return munmap(addr);
}

int
sys_freemem(void)
{
  return freemem();
}
```

11) user.h 에 sys_call에 쓰일 실제 구현된 mmap(), munmap() , freemem() 함수 prototype 전역 선언.

```
uint mmap(uint addr, int length, int prot, int flags, int fd, int offset);
int munmap(uint addr);
int freemem(void);
```

12) usys.S에 SYSCALL 매크로 함수 선언

```

SYSCALL(mmap)
SYSCALL(munmap)
SYSCALL(freemem)
```

2. sys_mmap.c라는 이름의 별도의 파일에 mmap, munmap, freemem 함수 선언 및 작성.

1) mmap() 함수 선언

```
uint
mmap(uint addr, int length, int prot, int flags, int fd, int offset)
{
   struct proc *p = myproc();
    struct mmap_area *area = 0;
    struct file *f = 0;

    // Check that addr is page aligned and length is positive
    if (addr % PGSIZE != 0 || length <= 0)
        return -1;
```

```
    if ((flags & MAP_ANONYMOUS) == 0) {
        if (fd == -1)
            return -1; // Invalid file descriptor

        f = p->ofile[fd];
        if (!f)
            return -1; // Invalid file descriptor

        if ((prot & PROT_WRITE) && (f->writable == 0))
            return -1; // Protection mismatch with file's open flags

        filedup(f);  // increase the reference count of the file
    }

    // Check for overlapping mappings
    for (int i = 0; i < MAX_MMAP_AREA; i++) {
        if (p->mmap_areas[i].addr != 0 &&
            ((p->mmap_areas[i].addr <= addr && p->mmap_areas[i].addr + p-
>mmap_areas[i].length > addr) ||
            (p->mmap_areas[i].addr >= addr && addr + length > p-
>mmap_areas[i].addr))) {
            return -1; // Overlapping mapping detected
        }
    }

    // Find an available mmap_area
    for (int i = 0; i < MAX_MMAP_AREA; i++) {
        if (p->mmap_areas[i].addr == 0) {
            area = &p->mmap_areas[i];
            break;
        }
    }

    if (!area)
        return -1; // No free mmap_area available
    area->f = f;
    area->addr = MMAPBASE + addr;
    area->length = length;
    area->offset = offset;
    area->prot = prot;
    area->flags = flags;
    area->p = p;

    if (flags & MAP_POPULATE) {
        // Allocate physical pages and create the page table entries for the
entire mapping area
```

```
        for (uint a = area->addr; a < area->addr + area->length; a += PGSIZE)
{
            char *mem = kalloc();
            cprintf("mmap called with address: %x, length: %d\n", addr,
length);
            if (!mem) {
                // Handle error: failed to allocate memory
                return -1;
            }

            if(walkpgdir(p->pgdir, (void*)a, 0) != 0) {
                cprintf(">> Address already mapped: %x\n", a);
                continue;  // Skip this address
            }
            if(mappages(p->pgdir, (void*)a, PGSIZE, V2P(mem), PTE_W|PTE_U) <
0) {
                // Handle error: failed to map pages
                kfree(mem);  // Don't forget to free the allocated page in
case of error
                return -1;
            }
        }
    }
    else if (flags & MAP_ANONYMOUS) {
        // For anonymous mapping, we still need to allocate physical pages and
create the page table entries
        for (uint a = area->addr; a < area->addr + area->length; a += PGSIZE)
{
            char *mem = kalloc();
            if (!mem) {
                // Handle error: failed to allocate memory
                return -1;
            }
            memset(mem, 0, PGSIZE);
            if (mappages(p->pgdir, (void*)a, PGSIZE, V2P(mem), PTE_W|PTE_U) <
0) {
                // Handle error: failed to map pages
                cprintf(">> Handle error: failed to map pages\n");
                kfree(mem);
       // Don't forget to free the allocated page in case of error
                return -1;
            }
        }
    }
    return area->addr;
}
```

2) munmap() 함수 구현을 위한 내부 함수 및 munmap() 함수 :

```
void
freepgtable(pde_t *pgdir, void *va, uint size)
{
    char *a, *last;
    pte_t *pte;

    a = (char*)PGROUNDDOWN((uint)va);
    last = (char*)PGROUNDDOWN(((uint)va) + size - 1);
    for(; a <= last; a += PGSIZE){
        if((pte = walkpgdir(pgdir, a, 0)) == 0)
        continue;
        if(*pte & PTE_P){
        char *v = P2V(PTE_ADDR(*pte));
        kfree(v);
        }
        *pte = 0;
    }
}
int
remove_mmap_area(struct proc *p, uint addr)
{
    int found = -1;

    // Traverse the array of mmap_areas
    for (int i = 0; i < MAX_MMAP_AREA; i++) {
        if (p->mmap_areas[i].addr == addr) { // If the current area is the
target area
            found = i;
            break;
        }
    }

    // If the target area wasn't found, return -1
    if (found == -1)
        return -1;

    // Mark the found mmap_area as unused. Here I'm assuming that an addr of 0
means the area is unused.
    p->mmap_areas[found].addr = 0;
    p->mmap_areas[found].length = 0;
    p->mmap_areas[found].flags = 0;
    p->mmap_areas[found].prot = 0;
    p->mmap_areas[found].offset = 0;
    p->mmap_areas[found].f = 0;
```

```
    p->mmap_areas[found].p = 0;

    return 1;
}

int
munmap(uint addr)
{
    struct proc *p = myproc();
    struct mmap_area *area = find_mmap_area(p, addr);
    if (!area)
        return -1;

    pte_t *pte;
    char *a;
    uint pa;

    for (a = (char*)PGROUNDDOWN((uint)area->addr); a < (char*)(area->addr +
area->length); a += PGSIZE) {
        pte = walkpgdir(p->pgdir, a, 0);
        if (!pte)
            continue;
        if (*pte & PTE_P) {
            pa = PTE_ADDR(*pte);
            if(pa == 0)
                panic("kfree");
            char *v = P2V(pa);
            memset(v, 1, PGSIZE);
            kfree(v);
            *pte = 0;
        }
    }

    // Removing corresponding mmap_area structure
    if(remove_mmap_area(p, addr) < 0)
        return -1;

    return 1;
}
```

3) **freemem() 함수 : kalloc에 선언되어 있는 free_page_count 변수를 return.**

```
int
freemem(void)
{
    extern int free_page_count; // Import the global variable from kalloc.c
    return free_page_count;
}
```