Null pointer dereference

Makefile

```
148  _%: %.o $(ULIB)
149      $(LD) $(LDFLAGS) -N -e main -Ttext 0x1000 -o $@ $^
150      $(OBJDUMP) -S $@ > $*.asm
151      $(OBJDUMP) -t $@ | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$$/d' > $*.sym
```

mmu.h 확인 결과 xv6의 PGSIZE는 0x1000으로 과제에서 주어진 대로 149번째 줄 주소를 0에서 0x1000으로 변경하여 사용자 프로그램의 시작 주소를 첫 번째 페이지에서 두 번째 페이지로 변경하였다.

exec.c

```
42    sz = PGSIZE;
43    for(i=0, off=elf.phoff; i<elf.phnum; i++, off+=sizeof(ph)){
44      if(readi(ip, (char*)&ph, off, sizeof(ph)) != sizeof(ph))
45        goto bad;
46      if(ph.type != ELF_PROG_LOAD)
47        continue;
48      if(ph.memsz < ph.filesz)
49        goto bad;
50      if(ph.vaddr + ph.memsz < ph.vaddr)
51        goto bad;
52      if((sz = allocuvm(pgdir, sz, ph.vaddr + ph.memsz)) == 0)
53        goto bad;
54      if(ph.vaddr % PGSIZE != 0)
55        goto bad;
56      if(loaduvm(pgdir, (char*)ph.vaddr, ip, ph.off, ph.filesz) < 0)
```

exec는 새로운 프로그램을 로드하기 때문에 sz=PGSIZE로 변경하여 exec에서 사용자 프로그램이 0x1000에서 시작하도록 변경하였다.

vm.c

```
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 = PGSIZE; 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");
    pa = PTE_ADDR(*pte);
    flags = PTE_FLAGS(*pte);
```

copyuvm은 가상 메모리를 복사하여 새로운 페이지 디렉토리에 매핑하는 함수이므로 i = PGSIZE로 변경하여 마찬가지로 첫 번째 페이지를 비우도록 한다.
위 과정을 통해 널 포인터 역참조에 대한 예외처리를 할 수 있다.

memory_protection

vm.c

```c
int mprotect(void *addr, int len){
  struct proc *curproc = myproc();
  pte_t *pte;

  // 만약 addr이 페이지에 정렬되지 않았다면 -1을 반환
  if((uint)addr % PGSIZE != 0) {
    cprintf("addr is not page aligned");
    return -1;
  }

  // 만약 현재 주소 공간이 아니라면 -1 반환
  if (curproc->sz <= (uint)addr || (uint)addr + len > curproc->sz) {
    cprintf("addr is not the current address space");
    return -1;
  }

    //만약 len이 0보다 작거나 같다면 -1 반환
  if(len <= 0){
    cprintf("len is less than or equal to zero");
    return -1;
  }

  for(int i = 0; i < len; i+= PGSIZE){
    pte = walkpgdir(curproc->pgdir, addr + i, 0);
    if(!pte)
    return -1;
  *pte &= ~PTE_W;
  }

  lcr3(V2P(curproc->pgdir));  //Update CR3 register
  cprintf("\nPTE: 0x%x \n", pte);
  return 0;
}
```

```c
int munprotect(void *addr, int len){

  struct proc *curproc = myproc();
  pte_t *pte;

    // 만약 addr이 페이지에 정렬되지 않았다면 -1을 반환
  if((uint)addr % PGSIZE != 0) {
    cprintf("addr is not page aligned");
    return -1;
  }

  // 만약 현재 주소 공간이 아니라면 -1 반환
  if (curproc->sz <= (uint)addr || (uint)addr + len > curproc->sz) {
    cprintf("addr is not the current address space");
    return -1;
  }

    //만약 len이 0보다 작거나 같다면 -1 반환
  if(len <= 0){
    cprintf("len is less than or equal to zero");
    return -1;
  }

  for(int i = 0; i < len; i+=PGSIZE){
    pte = walkpgdir(curproc->pgdir, addr + i, 0);
    if(!pte)
      return -1;

    *pte |= PTE_W;
  }
  lcr3(V2P(curproc->pgdir));  //Update CR3 register
  cprintf("\nPTE: 0x%x \n", pte);
  return 0;
}
```

과제에서 주어진 대로 vm.c에서 mprotect와 munprotec 두 함수를 구현하였다.
오류에 대한 처리를 위해 주어진 오류 조건 addr이 페이지에 정렬되지 않은 경우, 현재 주소 공간이 아닌 경우 그리고 len이 0보다 작거나 같은 경우에 대해 오류 문구를 출력하고 return -1을 하도록 하였다.
오류 없이 위 내용을 통과하면 반복문을 통해 길이만큼 주소 범위를 페이지 단위로 순회하며 다음 내용을 수행한다.

mprotect

- 페이지 테이블 엔트리를 얻어오고 만약 존재하지 않는다면 "-1"을 반환한다.
- 페이지 테이블 엔트리가 존재한다면, mmu.h에서 정의된 쓰기 비트인 PTE_W(0x002)를 뒤집어 pte와 '&' 시켜 쓰기 비트를 제거하여 해당 페이지를 읽기 전용으로 변경한다.

munprotect

- 페이지 테이블 엔트리를 얻어오고 만약 존재하지 않는다면 "-1"을 반환한다.
- 페이지 테이블 엔트리가 존재한다면, mmu.h에서 정의된 쓰기 비트인 PTE_W(0x002)를 뒤집어 pte와 '&' 시켜 쓰기 비트를 설정하여 해당 페이지를 쓰기 가능한 상태로 변경한다.

위 과정이 종료되면 lcr3함수를 사용하여 하드웨어가 페이지 테이블 항목의 변경 사항을 인식할 수 있도록 CR3 레지스터를 업데이트한다.

이후 두 함수를 systemcall로 추가하기 위한 동작을 수행한다.

defs.h

```
173  // vm.c
174  void            seginit(void);
175  void            kvmalloc(void);
176  pde_t*          setupkvm(void);
177  char*           uva2ka(pde_t*, char*);
178  int             allocuvm(pde_t*, uint, uint);
179  int             deallocuvm(pde_t*, uint, uint);
180  void            freevm(pde_t*);
181  void            inituvm(pde_t*, char*, uint);
182  int             loaduvm(pde_t*, char*, struct inode*, uint, uint);
183  pde_t*          copyuvm(pde_t*, uint);
184  void            switchuvm(struct proc*);
185  void            switchkvm(void);
186  int             copyout(pde_t*, uint, void*, uint);
187  void            clearpteu(pde_t *pgdir, char *uva);
188  int             mprotect(void *addr, int len);      //구현부
189  int             munprotect(void *addr, int len);
```

vm.c에 추가한 두 함수를 정의한다.

user.h / usys.S

```
4   // system calls
5   int fork(void);
6   int exit(void) __attribute__((noreturn));
7   int wait(void);
8   int pipe(int*);
9   int write(int, const void*, int);
10  int read(int, void*, int);
11  int close(int);
12  int kill(int);
13  int exec(char*, char**);
14  int open(const char*, int);
15  int mknod(const char*, short, short);
16  int unlink(const char*);
17  int fstat(int fd, struct stat*);
18  int link(const char*, const char*);
19  int mkdir(const char*);
20  int chdir(const char*);
21  int dup(int);
22  int getpid(void);
23  char* sbrk(int);
24  int sleep(int);
25  int uptime(void);
26  //구현부
27  int mprotect(void *addr, int len);
28  int munprotect(void *addr, int len);
```

```
4   // system calls
5   int fork(void);
6   int exit(void) __attribute__((noreturn));
7   int wait(void);
8   int pipe(int*);
9   int write(int, const void*, int);
10  int read(int, void*, int);
11  int close(int);
12  int kill(int);
13  int exec(char*, char**);
14  int open(const char*, int);
15  int mknod(const char*, short, short);
16  int unlink(const char*);
17  int fstat(int fd, struct stat*);
18  int link(const char*, const char*);
19  int mkdir(const char*);
20  int chdir(const char*);
21  int dup(int);
22  int getpid(void);
23  char* sbrk(int);
24  int sleep(int);
25  int uptime(void);
26  //구현부
27  int mprotect(void *addr, int len);
28  int munprotect(void *addr, int len);
```

사용자 레벨에서 함수에 대한 인터페이스를 제공하고 커널 간의 동작을 위해 user.h와 usys.S 파일에 추가한 두 함수를 정의한다.

syscall.c

```
85   extern int sys_chdir(void);
86   extern int sys_close(void);
87   extern int sys_dup(void);
88   extern int sys_exec(void);
89   extern int sys_exit(void);
90   extern int sys_fork(void);
91   extern int sys_fstat(void);
92   extern int sys_getpid(void);
93   extern int sys_kill(void);
94   extern int sys_link(void);
95   extern int sys_mkdir(void);
96   extern int sys_mknod(void);
97   extern int sys_open(void);
98   extern int sys_pipe(void);
99   extern int sys_read(void);
100  extern int sys_sbrk(void);
101  extern int sys_sleep(void);
102  extern int sys_unlink(void);
103  extern int sys_wait(void);
104  extern int sys_write(void);
105  extern int sys_uptime(void);
106  extern int sys_mprotect(void);
107  extern int sys_munprotect(void);
```

```
109  static int (*syscalls[])(void) = {
110  [SYS_fork]    sys_fork,
111  [SYS_exit]    sys_exit,
112  [SYS_wait]    sys_wait,
113  [SYS_pipe]    sys_pipe,
114  [SYS_read]    sys_read,
115  [SYS_kill]    sys_kill,
116  [SYS_exec]    sys_exec,
117  [SYS_fstat]   sys_fstat,
118  [SYS_chdir]   sys_chdir,
119  [SYS_dup]     sys_dup,
120  [SYS_getpid]  sys_getpid,
121  [SYS_sbrk]    sys_sbrk,
122  [SYS_sleep]   sys_sleep,
123  [SYS_uptime]  sys_uptime,
124  [SYS_open]    sys_open,
125  [SYS_write]   sys_write,
126  [SYS_mknod]   sys_mknod,
127  [SYS_unlink]  sys_unlink,
128  [SYS_link]    sys_link,
129  [SYS_mkdir]   sys_mkdir,
130  [SYS_close]   sys_close,
131  [SYS_mprotect]  sys_mprotect,
132  [SYS_munprotect]  sys_munprotect,
133  };
```

운영체제에서 동작하는 systemcall 함수의 동작을 위해 syscall.c 파일에 두 함수를 추가한다.

syscall.h

```
1   // System call numbers
2   #define SYS_fork    1
3   #define SYS_exit    2
4   #define SYS_wait    3
5   #define SYS_pipe    4
6   #define SYS_read    5
7   #define SYS_kill    6
8   #define SYS_exec    7
9   #define SYS_fstat   8
10  #define SYS_chdir   9
11  #define SYS_dup    10
12  #define SYS_getpid 11
13  #define SYS_sbrk   12
14  #define SYS_sleep  13
15  #define SYS_uptime 14
16  #define SYS_open   15
17  #define SYS_write  16
18  #define SYS_mknod  17
19  #define SYS_unlink 18
20  #define SYS_link   19
21  #define SYS_mkdir  20
22  #define SYS_close  21
23  #define SYS_mprotect 22
24  #define SYS_munprotect 23
```

추가한 두 함수에 대해 systemcall 번호에 해당하는 매크로를 정의한다.

sysproc.c

```
10  //구현부
11  int
12  sys_mprotect(void)
13  {
14    void *addr;
15    int len;
16
17    if(argptr(0, (void *)&addr, sizeof(*addr)) < 0 || argint(1, &len) < 0)
18      return -1;
19
20    return mprotect(addr, len);
21
22  }
23
24  int sys_munprotect(void)
25  {
26    void *addr;
27    int len;
28
29    if(argptr(0, (void *)&addr, sizeof(*addr)) < 0 || argint(1, &len) < 0)
30      return -1;
31
32    return munprotect(addr, len);
33  }
```

사용자 프로세스와 커널간 통신을 위해 sys_mprotect와 sys_munprotect를 추가한다. 두 함수 모두 argptr 함수와 argint 함수를 사용하여 인자들을 가져온다. 해당 과정에 실패하면 -1을 반환하고 인자들을 해당 함수에 반환한다.

결과

```
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ test_protection

PTE: 0x8dee1010
protected value = 100

PTE: 0x8df26010
After unprotecting the value became = 5

Watch this,I'll trap now
pid 3 test_protection: trap 14 err 7 on cpu 1 eip 0x1056 addr 0x4000--kill proc
$ test_null
pid 5 test_null: trap 14 err 4 on cpu 1 eip 0x1011 addr 0x0--kill proc
```

에뮬레이션 진행 과정.

1. make qemu-nox
2. test_protection
3. test_null