OS PA_3 report 소프트웨어학과 2018314827 차승일

Abstract

```
modified:   defs.h
modified:   file.h
modified:   kalloc.c
modified:   my_test.c
modified:   param.h
modified:   proc.c
modified:   proc.h
modified:   sleeplock.h
modified:   syscall.c
modified:   syscall.h
modified:   sysproc.c
modified:   trap.c
modified:   user.h
modified:   usys.S
modified:   vm.c
```

바뀐 파일들의 리스트

defs.h

```
66  // kalloc.c
67  char*           kalloc(void);
68  void            kfree(char*);
69+ int             get_free_count(void);
70  void            kinit1(void*, void*);
71  void            kinit2(void*, void*);
72
73  // kbd.c
74  void            kbdintr(void);
```

free count를 반환하는 unsigned int형 함수 정의

```
126  void            ps(int);
127+ unsigned int    mmap(unsigned int,int,int,int,int,int);
128+ int             munmap(unsigned int);
129+ int             freemem(void);
130+ int             pagefault_handle(unsigned int, unsigned int, struct proc*);
131
132  // swtch.S
```

쓸 시스템콜+페이지 폴트 핸들러 선언

```
191 pde_t*          copyuvm(pde_t*, uint);
192 void            switchuvm(struct proc*);
193 void            switchkvm(void);
194 int             copyout(pde_t*, uint, void*, uint);
195 void            clearpteu(pde_t *pgdir, char *uva);
196+ unsigned int*   walkpgdir(pde_t *pgdir, const void *va, int alloc);
197+ int             mappages(pde_t *pgdir, void *va, uint size, uint pa, int perm, char is_usermode);
198
199  // number of elements in fixed-size array
```

이미 vm.c에 있는 함수들이지만 defs.h에 선언해줘야 다른 곳에서 쓸 수 있는것 같아서 선언해줌. walkpgdir은 pte_t라는 vm.c내에서 정의된 특별한 타입을 타입으로 갖는데 어짜피 그게 unsigned int를 `typedef uint pte_t;` 로 한거라 걍 `unsigned int*`로 선언해줌

file.h

```
1+ #include "sleeplock.h"
2+
3  struct file {
4    enum { FD_NONE, FD_PIPE, FD_INODE } type;
5    int ref; // reference count
6    char readable;
7    char writable;
```

이 헤더파일이 필요하길래 include 해줌

kalloc.c

```
31+
32+ int free_count=-1;
33+
34  void
35  kinit1(void *vstart, void *vend)
36  {
37    initlock(&kmem.lock, "kmem");
38    kmem.use_lock = 0;
39    freerange(vstart, vend);
40+
41+   free_count = 0;
42  }
```

free count를 나타내는 전역변수 하나를 선언해줌. kinit1일때 이걸 0으로 초기화해준다. 초기화를 -1로 준 이유는 kinit1 순서가 꼬여서 free_count가 음수인 시점이 발견되었을 때 핸들할 가능성을 열어두기 위함이다.

```
64  void
65  kfree(char *v)
66  {
67    struct run *r;
68
69    if((uint)v % PGSIZE || v < end || V2P(v) >= PHYSTOP)
70      panic("kfree");
71
72    // Fill with junk to catch dangling refs.
73    memset(v, 1, PGSIZE);
74
75    if(kmem.use_lock)
76      acquire(&kmem.lock);
77    r = (struct run*)v;
78    r->next = kmem.freelist;
79    kmem.freelist = r;
80    if(kmem.use_lock)
81      release(&kmem.lock);
82+   free_count += 1;
83+ }
```

kfree를 하면 공간이 하나 생긴다 free count를 높여준다

```
84+
85+
86+ int get_free_count(void){
87+   return free_count;
88  }
89
90  // Allocate one 4096-byte page of physical memory.
91  // Returns a pointer that the kernel can use.
92  // Returns 0 if the memory cannot be allocated.
93  char*
94  kalloc(void)
95  {
96    struct run *r;
97
98    if(kmem.use_lock)
99      acquire(&kmem.lock);
100   r = kmem.freelist;
101   if(r)
102     kmem.freelist = r->next;
103   if(kmem.use_lock)
104     release(&kmem.lock);
105+   free_count -= 1;
106+   if(free_count<0){
107+     cprintf("Something went wrong for free count!!\n");
108+   }
109   return (char*)r;
110 }
```

kalloc도 똑같이 구현, get_free_count를 부르면 free count에 접근이 가능

param.h

```
#define NCPU          8  // maximum number of CPUs
#define NOFILE       16  // open files per process
#define NFILE       100  // open files per system
#define NINODE       50  // maximum number of active i-nodes
#define NDEV         10  // maximum major device number
#define ROOTDEV       1  // device number of file system root disk
#define MAXARG       32  // max exec arguments
#define MAXOPBLOCKS  10  // max # of blocks any FS op writes
#define LOGSIZE      (MAXOPBLOCKS*3)  // max data blocks in on-disk
#define NBUF         (MAXOPBLOCKS*3)  // size of disk block cache
#define FSSIZE       1000  // size of file system in blocks
#define PROT_READ    0x1
#define PROT_WRITE   0x2
#define MAP_ANONYMOUS 0x1
#define MAP_POPULATE 0x2
#define MMAPBASE 0x40000000
```

구현하라고 요구받은 것과 내가 쓸 mmap base도 따로 선언했다

proc.h

```
#include <stddef.h>

// Per-CPU state
struct cpu {
  uchar apicid;                // Local APIC ID
  struct context *scheduler;   // swtch() here to en
  struct taskstate ts;         // Used by x86 to fin
  struct segdesc gdt[NSEGS];   // x86 global descrip
  volatile uint started;       // Has the CPU starte
```

헤더파일 include

```
67
68  struct mmap_area{
69      struct file *f;
70      unsigned int addr;
71      int length;
72      int offset;
73      int prot;
74      int flags;
75      struct proc *p; // the process with this mmap_area
76      char mark; // 'e' for empty, 'n' for non empty but not available, 'a' for available
77    };// Manage all mmap areas created by each mmap() call in one mmap_area array.
78    // Maximum number of mmap_area array is 64.
79
```

mmap_area 구조체이다. 슬라이드에 나와있는 그대로에 'mark'라는 구조체 멤버만 추가해줬다. mark 는 char형 변수로 내가 mmap_area를 핸들할때 편의를 위해 둔 변수이다. 이 mark가 e면 해당 구조체는 어떤 process도 점거하지 않았다는 뜻 n이면 점거했지만 메모리가 매핑 안 되었다는 뜻(페이지 폴트 유발 조건) a면 available, 즉 프로세스가 점거도 했고 매핑도 되었다는 뜻이다

proc.c

```
1+ #include <stddef.h>
2  #include "types.h"
3+ #include "mmu.h"
4  #include "defs.h"
5  #include "param.h"
6  #include "memlayout.h"

7  #include "x86.h"
8  #include "proc.h"
9  #include "spinlock.h"
0+ #include "fs.h"
1+ #include "file.h"
2+
3+ struct mmap_area mmap_area_list[64] = {
4+   [0 ... 63] = { .f = NULL, .addr = 0, .length = 0, .offset = 0, .prot = 0, .flags = 0, .p = NULL, .mark = 'e' }
5+ };
6
```

필요한 헤더파일들(mmu.h, fs.h, file.h는 파일과 메모리 관련 헤더파일들)을 include 해주고, mmap_area 64짜리 배열을 선언해주고 해당 조건들로 초기화해주자

아래는 mmap_area에 하나의 index의 mmap_area_list에 대해 정보를 갱신하는 함수이다.

```
211+ void allocate_mmap_area_list_member(
212+   int idx, struct file* file_to_use,
213+   unsigned int start_addr, int length,
214+   int offset, int prot, int flags, char mark,
215+   struct proc* p
216+ ) {
217+   if (idx >= 64){
218+     cprintf("Wrong idx!\n");
219+   }
220+   struct mmap_area *entry = &mmap_area_list[idx];
221+ 
222+   entry->f = file_to_use;
223+ 
224+   if ((start_addr % 4096)){
225+     cprintf("Page not aligned for start addr\n");
226+   }
227+   entry->addr = start_addr;
228+ 
229+   if ((length % 4096)){
230+     cprintf("Page not aligned length\n");
231+   }
232+   entry->length = length;
233+ 
234+   if (offset<0){
235+     cprintf("Wrong option for offset\n");
236+   }
```

조건 체크를 해준다

```
233+
234+   if (offset<0){
235+     cprintf("Wrong option for offset\n");
236+   }
237+   entry->offset = offset;
238+   entry->prot = prot;
239+   entry->flags = flags;
240+
241+   if (!(mark == 'e' || mark == 'n' || mark == 'a')){
242+     cprintf("Wrong mark condition!\n");
243+   }
244+   entry->mark = mark;
245+   entry->p = p;
246+ }
```

정보를 삽입한다.

proc.c-fork함수

```
249  int
250  fork(void)
251  {
252    int i, pid;
253    struct proc *np;
254    struct proc *curproc = myproc();
255+   int return_value = 0;
256
257    // Allocate process.
258    if((np = allocproc()) == 0){
259      return -1;
260    }
261
262    // Copy process state from proc.
263    if((np->pgdir = copyuvm(curproc->pgdir, curproc->sz)) == 0){
264      kfree(np->kstack);
265      np->kstack = 0;
266      np->state = UNUSED;
267      return -1;
268    }
269+
```

에러가 일어났을때 리턴할 변수 return_value를 만들어준다

```
287+  int parent_idx=0;
288+  int parent_idx_list[64];
289+
290+  // Find every parent idx
291+  for (int i=0; i<64; i++){
292+    parent_idx_list[i] = -1;
293+  }
294+
295+  i = 0;
296+
297+  for(parent_idx=0; parent_idx<64; parent_idx++){
298+    if(
299+      mmap_area_list[parent_idx].mark == 'n' ||
300+      mmap_area_list[parent_idx].mark == 'a'
301+    ){
302+      if(mmap_area_list[parent_idx].p == curproc){
303+        // Found our parent process!
304+        // Append this to parent idx list group
305+        parent_idx_list[i] = parent_idx;
306+        i++;
307+      }
308+    }
309+  }
```

복사해야하는 parent index들을 다 찾아준다. 즉 프로세스가 curproc인 mmap_area를 다 찾는거다(리스트에 넣어줌)

```
if (i){//If there is more than 1 membver for parent_idx_list
  for (i=0; i<64; i++){//Let's go
    parent_idx = parent_idx_list[i];
    if (parent_idx == -1){
      break;
    }

    int is_done = 0;
    for(int child_idx=0; child_idx<64; child_idx++){
      if(child_idx != parent_idx){
        if((mmap_area_list[child_idx].mark == 'e')&&!is_done){//Find empty slot
          struct mmap_area *parent_area = &mmap_area_list[parent_idx];
          allocate_mmap_area_list_member(//Input the slot
            child_idx,
            parent_area->f,
            parent_area->addr,
            parent_area->length,
            parent_area->offset,
            parent_area->prot,
            parent_area->flags,
            parent_area->mark,
            np
          ); // Copy parent's mmap area
          char is_file_mapping = parent_area->flags%2==0;
          char is_already_allocated = (parent_area->mark=='a');

          if (is_already_allocated){
            int page_size = 4096;
```

모든 parent_idx마다 mmap_area를 empty slot을 찾아서 복사해주는 과정이다.이미 page가 할당이 되어있는(is_already_allocated, mark가 ‘a’인 경우)) 비어있는 slot에 새로 매핑도 해서 그것또한 넣는다.

```
339+              char *memory_pointer = NULL;
340+
341+              for(
342+                int count=0;
343+                count<(int)(parent_area->length/page_size);
344+                count++
345+              ){
346+                if(!(memory_pointer = kalloc())){
347+                  return return_value;
348+                }
349+                else{
350+                  memset(memory_pointer, 0, page_size);
351+
352+                  if (is_file_mapping){
353+                    struct file* file_to_use = parent_area->f;
354+                    file_to_use->off = parent_area->offset;
355+
356+                    file_to_use->ref += 1;
357+                    fileread(
358+                      file_to_use, memory_pointer, page_size
359+                    );
360+                    file_to_use->ref -= 1;
361+                  }
```

parent idx의 mmap_area가 already allocated인 경우 페이지를 kalloc하는 과정이다. 파일매핑이면 파일을 읽어줄때 ref를 높였다 다읽으면 낮춘다.

```
362+
363+                  char is_usermode = 1;
364+                  int page_mapping_success = mappages(
365+                    np->pgdir,
366+                    (void*)(
367+                      parent_area->addr + count*page_size
368+                    ),
369+                    page_size, V2P(memory_pointer), parent_area->prot,
370+                    is_usermode
371+                  );
372+
373+                  if (!(page_mapping_success+1)) return return_value;
374+                }
375+              }
376+            }
377+            is_done=1;
378+          }
379+        }
380+      }
381+    }
382+  }
```

페이지를 매핑하는 과정이다 끝나면 is_done으로 끝났음을 표시해준다. mappages 안에 is_usermode라는 것이 보이는데 이건 뒤에서 mappages를 설명할때 설명하겠다.

proc.c mmap 함수

```
unsigned int mmap(
  unsigned int addr, int length,
  int prot, int flags, int fd, int offset
){
  struct file *file_to_use = NULL;
  char is_file_mapping = 1;
  if (fd == -1 && offset == 0) {
    is_file_mapping = 0;
  }
  unsigned int return_value = 0; // fail

  unsigned int start_addr = MMAPBASE+addr;
  if (is_file_mapping) {
    file_to_use = myproc()->ofile[fd];
  }

  if (prot == PROT_READ){
    // fail handler
    if (flags%2){ // anonmous mapping
      if (is_file_mapping){
        //cprintf("wrong1\n");
        return return_value;
      }
    }
    else { // flags == MAP_POPULATE, file mapping
      if (
        file_to_use->type != 2 ||
        !is_file_mapping ||
        file_to_use->readable == '0'
      ){
        //cprintf("wrong2\n");
        return return_value;
      } // It's not anonymous, but when the fd is -1
    }
  }
```

is_file_mapping이란 변수로 파일 맵핑인지 아닌지 나타내준다. MMAPBASE를 addr에 더해 start_addr로 앞으로 쓸 변수를 정의해주자

파일매핑이면 현프로세스에서 ofile[fd]로 접근한다.

prot이 read인데 flags가 어나니머스 맵핑인데 filemapping으로 된경우에는 잘못된거라 에러핸들링해줬다.
그다음 맵 populate인데 file mapping인 경우에는(flags로 확인) 우리가 사용하려는 파일이 FD_INODE 타입이 아니라 다른 타입이거나 앞에서 얻은 is_file_mapping 조건과 충돌하거나 파일이 readabe이 아니면 똑같이 return_value(에러핸들링)을 리턴한다.

```
else { // prot == PROT_WRITE
  if (flags%2){ // anonmous mapping
    if (is_file_mapping){
      //cprintf("wrong3\n");
      return return_value;
    }
  }
  else { // flags == MAP_POPULATE, file mapping
    if (
      file_to_use->type != 2 ||
      !is_file_mapping ||
      file_to_use->writable == '0'
    ){
      //cprintf("wrong4\n");
      return return_value;
    } // It's not anonymous, but when the fd is -1
  }
}
//cprintf("Wrong check passed\n");

int idx = 0;
char is_done = 0;
```

그담에 prot이 prot_write을 갖고있나 확인한다. 주석에는 `// prot == PROT_WRITE` 이라 되어 있지만 사실 `// prot == PROT_READ|PROT_WRITE` 이다.
즉 prot은 1 or 3이다. 이는 sysproc.c에서 핸들링 해놨다.

여기서도 위와 모든게 동일하지만, 파일 맵핑일때 사용하려는 파일이 writable한지 확인하다는 점만 다르다.

```
1155+  int idx = 0;
1156+  char is_done = 0;
1157+
1158+  int found_idx = 64;
1159+  for (idx=0; idx< 64; idx++){
1160+    if (mmap_area_list[idx].mark == 'e' && !is_done) {
1161+      if (is_file_mapping){
1162+        //cprintf("filed up\n");
1163+        file_to_use = filedup(file_to_use);
1164+      }
1165+      allocate_mmap_area_list_member(
1166+        idx, file_to_use, start_addr,
1167+        length, offset, prot, flags, 'n', myproc()
1168+      );
1169+      //cprintf("Mmap allocated to idx: %d\n", idx);
1170+      is_done=1;
1171+      found_idx=idx;
1172+    }
1173+  }
1174+  if(found_idx>=64) cprintf("No empty slot!");
1175+  idx=found_idx;
1176+
```

그다음 일단 mark를 n으로 넣고 나머지것들도(addr도 start_addr로 갈아끼우고) 전부 채워준다. populate 하지 않은 애들까지 전부 공통으로 적용되는 부분이다. 파일 맵핑이면 중간에 파일을 읽고, mmap_area_list에서 빈공간 찾아서 넣는다.

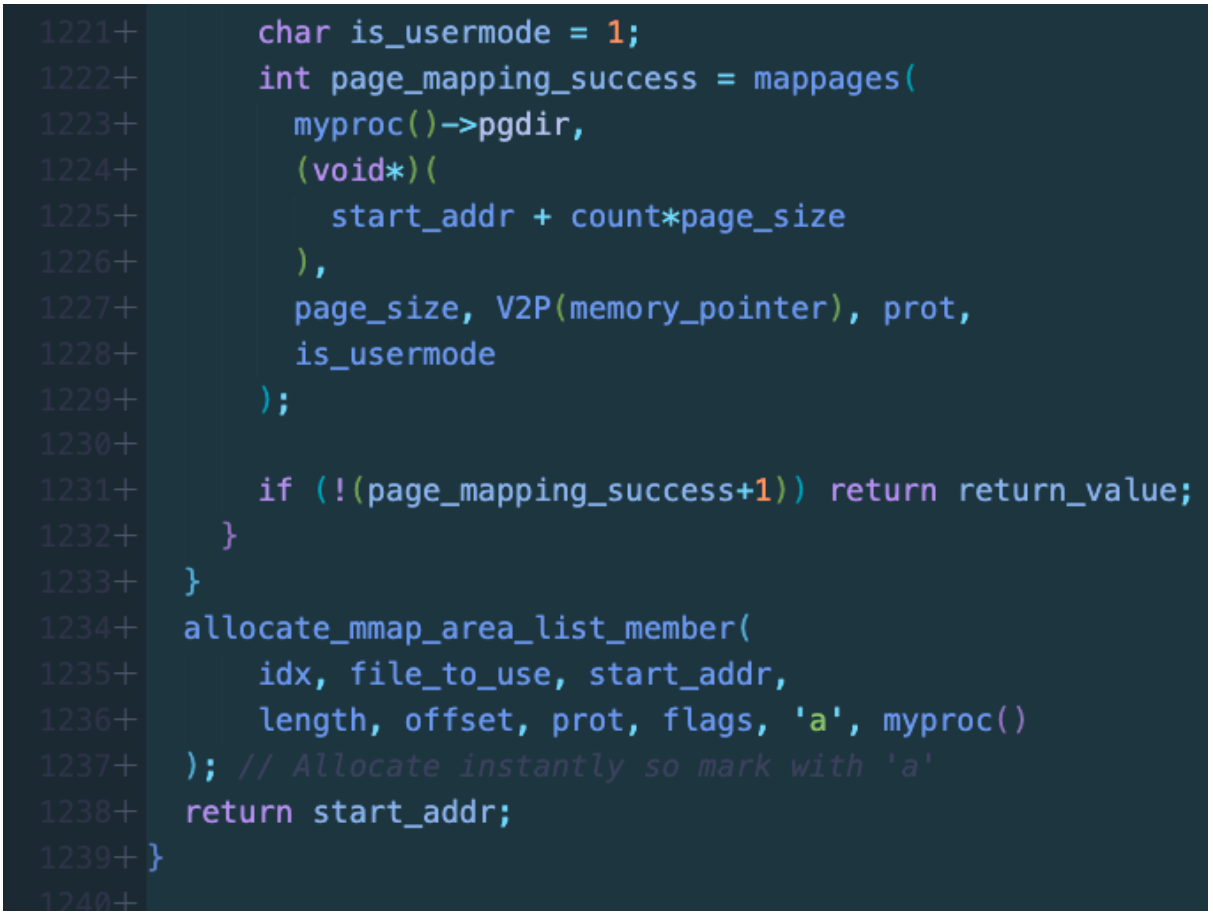
```
1177+  // mmap 마지막에 넣었으니 이제 시작
1178+  // !Populate인데 file=mapped | anonymous 인 경우
1179+  // 그냥 주소 return -> why? 나중에 page fault 일어나면 handle
1180+
1181+  if (!(flags/2)){
1182+    //cprintf("Not populate case\n");
1183+    return start_addr; // not populate
1184+  }
1185+  // 2 cases : 1. Populate & Anonymous 2. Populate & FileMapped
1186+
1187+  //cprintf("Populate case\n");
1188+
1189+  // 1. Anonymous case
1190+  int page_size = 4096;
1191+
1192+  char *memory_pointer = NULL;
1193+  file_to_use->off = offset;
1194+
1195+  //cprintf("Starting allocate populate case\n");
```

flags가 0이나 1이면 populate이 아니라 일단 리턴한다.(나중에 핸들)

page_size는 지주 쓸 변수라 정의해준다.
그다음 mem point를 정의해준다
file을 이용할거라 오프셋도 바꿔준다.

```
for(
  int count=0;
  count<(int)(length/page_size);
  count++
){
  //cprintf("Allocate: %d번째 case\n", count);

  //cprintf("KALLOC\n");
  if(!(memory_pointer = kalloc())){
    return return_value;
  }
  else{
    memset(memory_pointer, 0, page_size);

    //cprintf("MEMSET\n");

    if (is_file_mapping){
      file_to_use->ref += 1;
      fileread(
        file_to_use, memory_pointer, page_size
      );
      file_to_use->ref -= 1;
    }
```

이제 length가 몇개의 페이지 길이만큼인지 연산한후 allocate해준다.
kalloc이 실패하면 return_value(실패했을때 값)을 리턴한다
memset을 통해 메모리포인터에 memset을 페이지 사이즈만큼 해준다(why? length가 몇페이지만큼인지만큼 for문 돌리므로)
파일을 읽는다. 읽기전에 ref를 높여주고 다읽고 나면 낮춰준다.



```
1221+       char is_usermode = 1;
1222+       int page_mapping_success = mappages(
1223+         myproc()->pgdir,
1224+         (void*)(
1225+           start_addr + count*page_size
1226+         ),
1227+         page_size, V2P(memory_pointer), prot,
1228+         is_usermode
1229+       );
1230+
1231+       if (!(page_mapping_success+1)) return return_value;
1232+     }
1233+   }
1234+   allocate_mmap_area_list_member(
1235+       idx, file_to_use, start_addr,
1236+       length, offset, prot, flags, 'a', myproc()
1237+   ); // Allocate instantly so mark with 'a'
1238+   return start_addr;
1239+ }
1240+
```

mappages를 따로 좀 바꿨기 때문에 is_usermode라는 변수를 넣어줬다. 바뀐 mappages는 뒤에 설명하겠다.
mappages가 실패하면 return_value를 리턴한다.

그다음에 위의 for문이 모두 끝난 후에 mmap_area_list의 idx에 해당되는 멤버의 mark를 'a'로 표시해줘 allocate 되었다는걸 적어준다.
그담에 start_addr 리턴해준다

proc.c-munmap 함수

```
1241+ int munmap(unsigned int addr){
1242+   int return_value = -1;
1243+   int temp = 0;
1244+   int idx = 0;
1245+   int page_size = 4096;
1246+
1247+   if (addr%page_size) return return_value;
1248+
1249+   int found_idx = 0;
1250+   char is_done = 0;
1251+
1252+   for(
1253+     temp=0;
1254+     temp<64;
1255+     temp++
1256+   )
1257+     if(
1258+       mmap_area_list[temp].addr == addr
1259+       && mmap_area_list[temp].p == myproc()
1260+       && !is_done
1261+     ){
1262+       is_done = 1;
1263+       found_idx = temp;
1264+       break;
1265+     }
1266+   if (temp < 64){
1267+     idx = found_idx;
1268+     if(mmap_area_list[idx].mark=='n'){
1269+         mmap_area_list[idx].mark = 'e';
1270+       return 1;
1271+     }
1272+   }
```

addr가 page_size로 align안되어있으면 -1리턴함. 이제 munmap요청받은 주소에 해당되는 mamp_area_list 인덱스를 찾아준다.
인덱스를 찾았다면 진행하는데, mark가 'n'라면(allocate 안된 상태라면) 그냥 'e'로 바꿔주고 성공(1) return 한다

```
1273+   else{
1274+     return return_value;
1275+   }
```

mmap_area_list에서 해당 주소로 못 찾은 경우에는 뭔가 잘못된거므로 return_value로 -1리턴

```
1277+  for(
1278+    int count=0;
1279+    count<(int)(mmap_area_list[idx].length/page_size);
1280+    count++
1281+  ){
1282+    unsigned int *page_table_entry = walkpgdir(
1283+      myproc()->pgdir,
1284+      (char *)(addr+count*page_size), 0
1285+    );
1286+    if((*page_table_entry%2)){
1287+      if (page_table_entry){
1288+        kfree(P2V(PTE_ADDR(*page_table_entry)));
1289+        *page_table_entry = !(*page_table_entry%2);
1290+      }
1291+      else return return_value;
1292+    }
1293+    else{
1294+      continue;
1295+    }
1296+  }
1297+
1298+  mmap_area_list[idx].mark = 'e';
1299+  return 1;
1300+ }
```

allocate 했던 과정을 거꾸로 deallocate 하면 됨.
for 문을 lenght/page_size만큼 돌리면서 walkpgdir했을때 *page_table_entry%2인 애들한테만 kfree한다.(why? -> present bit이 1인 애들이므로 그걸 kfree하면 됨)
해당 과정에서 page_table_entry가 0인 것이 발견되면 잘못된 것이므로 return_value를 리턴해 에러 핸들링

```
*page_table_entry = !(*page_table_entry%2)
```

는 *page_table_entry%2값을 재활용하기 위함이지 그냥 *page_table_entry = 0과 같다

그다음엔 free 끝났으므로 'e'로 마킹해주고 1을 리턴함(성공이므로)

proc.c freemem 함수

```
1302+ int freemem(void){
1303+   return get_free_count();
1304+ }
```

get_free_count를 부른다

proc.c pagefault_handle 함수

trap.c를 설명 후 설명하겠다.

sleeplock.h

```
1+ #ifndef SLEEPLOCK_H
2+ #define SLEEPLOCK_H
3+
4  // Long-term locks for processes
5  struct sleeplock {
6    uint locked;       // Is the lock held?
7    struct spinlock lk; // spinlock protecting thi
8
9    // For debugging:
10   char *name;        // Name of lock.
11   int pid;           // Process holding lock
12 };
13
14+ #endif
```

이 조건이 필요하길래 추가해줬다

syscall.c

```
109+ extern int sys_mmap(void);
110+ extern int sys_munmap(void);
111+ extern int sys_freemem(void);
112
113  static int (*syscalls[])(void) = {
114  [SYS_fork]    sys_fork,
115  [SYS_exit]    sys_exit,
116  [SYS_wait]    sys_wait,
117  [SYS_pipe]    sys_pipe,
118  [SYS_read]    sys_read,
119  [SYS_kill]    sys_kill,
120  [SYS_exec]    sys_exec,
121  [SYS_fstat]   sys_fstat,
122  [SYS_chdir]   sys_chdir,
123  [SYS_dup]     sys_dup,
124  [SYS_getpid]  sys_getpid,
125  [SYS_sbrk]    sys_sbrk,
126  [SYS_sleep]   sys_sleep,
127  [SYS_uptime]  sys_uptime,
128  [SYS_open]    sys_open,
129  [SYS_write]   sys_write,
130  [SYS_mknod]   sys_mknod,
131  [SYS_unlink]  sys_unlink,
132  [SYS_link]    sys_link,
133  [SYS_mkdir]   sys_mkdir,
134  [SYS_close]   sys_close,
135  [SYS_getnice] sys_getnice,
136  [SYS_setnice] sys_setnice,
137  [SYS_ps]      sys_ps,
138+ [SYS_mmap]    sys_mmap,
139+ [SYS_munmap]  sys_munmap,
140+ [SYS_freemem] sys_freemem,
141  };
```

추가해줬다

syscall.h

```
20  #define SYS_link   19
21  #define SYS_mkdir  20
22  #define SYS_close  21
23  #define SYS_getnice 22
24  #define SYS_setnice 23
25  #define SYS_ps 24
26+ #define SYS_mmap 25
27+ #define SYS_munmap 26
28+ #define SYS_freemem 27
29
```

추가해줬다

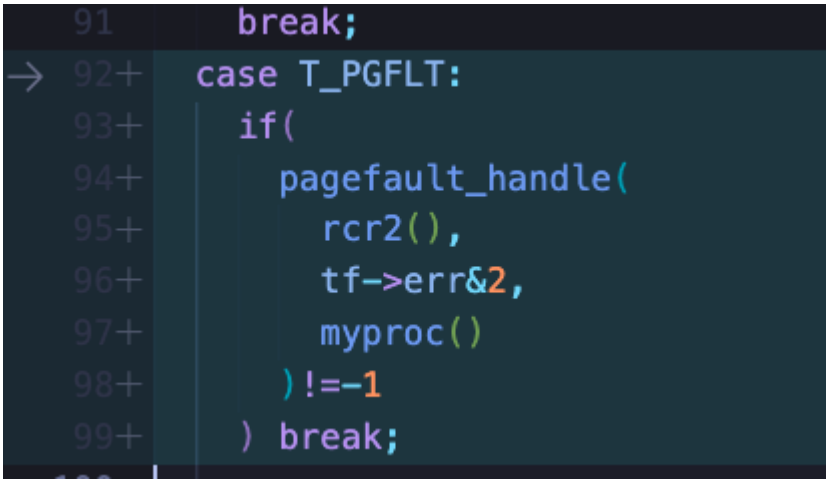
sysproc.c

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

  unsigned int return_value = 0;

  if(
    argint(0, &addr)>=0 &&
    !(argint(0, &addr)%4096) &&
    argint(1, &length)>=0 &&
    !(argint(1, &length)%4096) &&
    (argint(2, &prot)==0 ||
    argint(2, &prot)==1 || argint(2, &prot)==3) &&
    argint(3, &flags)>=0 && argint(3, &flags)<=3 &&
    (argint(4, &fd)>=-1) &&
    argint(5, &offset)>=0
  ) return_value = mmap(addr, length, prot, flags, fd, offset);

  return return_value;
}

int
sys_munmap(void){
  int addr;

  int return_value = -1;
  if(
    argint(0, &addr) >= 0 &&
    !(argint(0, &addr) % 4096)
  ) return_value=munmap(addr);

  return return_value;
}
```

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

```

잘못된 조건은 거르고 에러 핸들링 하고 아니면 불러야 할 함수를 부르도록 짰다

trap.c



```
91      break;
92+   case T_PGFLT:
93+     if(
94+       pagefault_handle(
95+         rcr2(),
96+         tf->err&2,
97+         myproc()
98+       )!=-1
99+     ) break;
```

슬라이드의 힌트를 그대로 사용해서 구현해 크게 설명할 부분은 없다.
rcr2(), tf->err&2와 myproc()는 페이지폴트 핸들러에서 쓸 예정이다. 슬라이드의 힌트대로 핸들러의 리턴값이 -1이 아니면 break한다

페이지폴트 핸들러(proc.c내에 존재)

```
1306+int pagefault_handle(
1307+  unsigned int address,
1308+  unsigned int err_2place_bit,
1309+  struct proc* curproc
1310+){
1311+  int return_value = -1;
1312+  int temp = 0;
1313+  int idx = 0;
1314+  int page_size = 4096;
1315+
1316+  char is_done = 0;
1317+  int found_idx = 64;
1318+
1319+  for(
1320+    temp=0;
1321+    temp<64;
1322+    temp++
1323+  ){
1324+    if(
1325+      mmap_area_list[temp].addr <= address &&
1326+      (mmap_area_list[temp].p==curproc) &&
1327+      (
1328+        address - mmap_area_list[temp].addr<
1329+        mmap_area_list[temp].length
1330+      ) &&
1331+      !is_done
1332+    ){
1333+      is_done = 1;
1334+      found_idx = temp;
1335+    }
1336+  }
1337+  temp = found_idx;
```

정의할걸 정의하고 현재 트랩이 걸린 address를 범위로 갖고 현재 트랩이 걸린 프로세스를 멤버로 갖고있는 mmap_area_lsit의 index를 찾는다

```
1341+  if(!(temp < 64)){
1342+    return return_value;
1343+  }
1344+  else{
1345+    idx = temp;
1346+  }
1347+
1348+  if(
1349+    (
1350+      mmap_area_list[idx].prot != 3
1351+      && err_2place_bit
1352+    )
1353+  ) return return_value;
1354+
1355+  if(
1356+    mmap_area_list[idx].mark == 'n'
1357+  ){
1358+    char *memory_pointer = NULL;
1359+    char is_file_mapping = mmap_area_list[idx].flags%2==0;
1360+
1361+    struct file* file_to_use = mmap_area_list[idx].f;
1362+    file_to_use->off = mmap_area_list[idx].offset;
1363+
1364+    for(
1365+      int count=0;
1366+      count<(int)(mmap_area_list[idx].length/page_size);
1367+      count++
1368+    ){
1369+
1370+      if(!(memory_pointer = kalloc())){
1371+        return return_value;
1372+      }
```

앞으로는 그냥 '에러가 났을떄 리턴하는 값'을 return_value 라고 칭하겠다. 물론 값 자체는 함수마다 다르다
여기서는 위의 함수들과 마찬가지로 index를 못 찾으면 return_value를 리턴한다
tf->err&2가 1이거나, 해당 idx쪽에서 read write 둘다 안되면(3이 아니면) return_value 리턴
밑에도 나와있지만 해당 mmap_area_list 엔트리의 mark는 'n'이어야 한다 . 아니면 뭔가 잘못된거다.(why?->애초에 'n'인 애를 접근했을때 일어나는게 페이지폴트이므로)
사실 그다음에 allocate 하는건 위의 나온것들과 똑같으므로 설명은 생략하겠따.

```
    else{
      memset(memory_pointer, 0, page_size);

      if (is_file_mapping){
        file_to_use->ref += 1;
        fileread(
          file_to_use, memory_pointer, page_size
        );
        file_to_use->ref -= 1;
      }

      char is_usermode = 1;
      int page_mapping_success = mappages(
        curproc->pgdir,
        (void*)(
          mmap_area_list[idx].addr + count*page_size
        ),
        page_size, V2P(memory_pointer),
        mmap_area_list[idx].prot,
        is_usermode
      );

      if (!(page_mapping_success+1)) return return_value;
    }
  }
  allocate_mmap_area_list_member(
      idx, file_to_use, mmap_area_list[idx].addr,
      mmap_area_list[idx].length, mmap_area_list[idx].offset,
      mmap_area_list[idx].prot, mmap_area_list[idx].flags, 'a',
      mmap_area_list[idx].p
  ); // Allocate instantly so mark with 'a'
  return 0;
}
//cprintf("Invalid operation!");
```

설명 생략(성공하면 0 리턴)

```
  //cprintf("Invalid operation!");
  return -1;
}
```

마크가 n이 아니면 -1 리턴

user.h

```
27  int setnice(int, int);
28  void ps(int);
29+ unsigned int mmap(unsigned int,int,int,int,int,int);
30+ int munmap(unsigned int);
31+ int freemem(void);
```

쓸거 추가하였다

usys.S

```
33  SYSCALL(setnice)
34  SYSCALL(ps)
35+ SYSCALL(mmap)
36+ SYSCALL(munmap)
37+ SYSCALL(freemem)
38
```

추가할거 추가하였다.

vm.c

```
35- static pte_t *
36  walkpgdir(pde_t *pgdir, const void *va, int al
37  {
38    pde_t *pde;
39    pte_t *pgtab;
40
41    pde = &pgdir[PDX(va)];
42    if(*pde & PTE_P){
43      pgtab = (pte_t*)P2V(PTE_ADDR(*pde));
44    } else {
45      if(!alloc || (pgtab = (pte_t*)kalloc()) ==
46        return 0;
47      // Make sure all those PTE_P bits are zero
48      memset(pgtab, 0, PGSIZE);
49      // The permissions here are overly generou
50      // be further restricted by the permission
51      // entries, if necessary.
52      *pde = V2P(pgtab) | PTE_P | PTE_W | PTE_U;
53    }
54    return &pgtab[PTX(va)];
55  }
```

```
35+ pte_t*
36  walkpgdir(pde_t *pgdir, const void *va, int alloc)
37  {
38    pde_t *pde;
39    pte_t *pgtab;
40
41    pde = &pgdir[PDX(va)];
42    if(*pde & PTE_P){
43      pgtab = (pte_t*)P2V(PTE_ADDR(*pde));
44    } else {
45      if(!alloc || (pgtab = (pte_t*)kalloc()) == 0)
46        return 0;
47      // Make sure all those PTE_P bits are zero.
48      memset(pgtab, 0, PGSIZE);
49      // The permissions here are overly generous, but
50      // be further restricted by the permissions in t
51      // entries, if necessary.
52      *pde = V2P(pgtab) | PTE_P | PTE_W | PTE_U;
53    }
54    return &pgtab[PTX(va)];
55  }
```

static이 있으면 작동을 안 한다. 따라서 static을 없애주었다.

```
60+ int
61+ mappages(pde_t *pgdir, void *va, uint size, uint pa, int perm, char is_usermode)
62  {
63    char *a, *last;
64    pte_t *pte;
65
66+   int perm_origin = perm;
67+   if(is_usermode) perm = perm|0b100;
68    a = (char*)PGROUNDDOWN((uint)va);
69    last = (char*)PGROUNDDOWN(((uint)va) + size - 1);
70    for(;;){
71      if((pte = walkpgdir(pgdir, a, 1)) == 0)
72        return -1;
73      if(*pte & PTE_P)
74        panic("remap");
75      *pte = pa | perm | PTE_P;
76      if(a == last)
77        break;
78      a += PGSIZE;
79      pa += PGSIZE;
80    }
81+
82+   perm = perm_origin;
83    return 0;
84  }
```

위에서 언급했던 mappages의 수정 버전이다. usermode의 변수로 user가 접글할수 있도록 한다. perm의 오른쪽에서 3번째 bit(100)이 1이어야 유저모드로 쓸 수 있다. 따라서 이를 핸들하는 옵션을 추가해줬다.

그다음은 vm.c에서 원래 mappages를 쓰던 부분들에 대해 usermode 옵션이 추가된 만큼 이를 커버하는 부분을 추가해준 부분들이다. 전부 vm.c에 있는 부분들이다.

```
// Set up kernel part of a page table.
pde_t*
setupkvm(void)
{
  pde_t *pgdir;
  struct kmap *k;

  if((pgdir = (pde_t*)kalloc()) == 0)
    return 0;
  memset(pgdir, 0, PGSIZE);
  if (P2V(PHYSTOP) > (void*)DEVSPACE)
    panic("PHYSTOP too high");
  for(k = kmap; k < &kmap[NELEM(kmap)]; k++)
    if(mappages(pgdir, k->virt, k->phys_end - k->phys_start,
                (uint)k->phys_start, k->perm, 0) < 0) {
      freevm(pgdir);
      return 0;
    }
  return pgdir;
}
```

```
// sz must be less than a page.
void
inituvm(pde_t *pgdir, char *init, uint sz)
{
  char *mem;

  if(sz >= PGSIZE)
    panic("inituvm: more than a page");
  mem = kalloc();
  memset(mem, 0, PGSIZE);
+ mappages(pgdir, 0, PGSIZE, V2P(mem), PTE_W|PTE_U, 0);
  memmove(mem, init, sz);
}
```

```
// newsz, which need not be page aligned.  Returns new size or 0 on error
int
allocuvm(pde_t *pgdir, uint oldsz, uint newsz)
{
  char *mem;
  uint a;

  if(newsz >= KERNBASE)
    return 0;
  if(newsz < oldsz)
    return oldsz;

  a = PGROUNDUP(oldsz);
  for(; a < newsz; a += PGSIZE){
    mem = kalloc();
    if(mem == 0){
      cprintf("allocuvm out of memory\n");
      deallocuvm(pgdir, newsz, oldsz);
      return 0;
    }
    memset(mem, 0, PGSIZE);
+   if(mappages(pgdir, (char*)a, PGSIZE, V2P(mem), PTE_W|PTE_U, 0) < 0){
      cprintf("allocuvm out of memory (2)\n");
      deallocuvm(pgdir, newsz, oldsz);
      kfree(mem);
      return 0;
    }
  }
  return newsz;
}
```

```
// 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");
    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, V2P(mem), flags, 0) < 0) {
      kfree(mem);
      goto bad;
    }
  }
```