


OS PA_4 report 소프트웨어학과

2018314827 차승일

바뀐파일

```
Changes not staged for commit:
  (use "git add <file>..." to
   (use "git restore <file>..." to
    modified:   defs.h
    modified:   fs.c
    modified:   kalloc.c
    modified:   proc.c
    modified:   trap.c
    modified:   vm.c
```

defs.h

```
→ 194+ 
195+ void mov_buff(int oldoff, int maximum_swap, int newoff);
196+ void pagelist_insertion(char* virtual_addr, int success, unsigned int* pgdir);
197+ int getpid(void);
198+ void page_fault_handle(unsigned int trap_no, unsigned int fault_addr, unsigned int* page_dir);
199+ unsigned int* walkpgdir(unsigned int *pgdir, const void* va, int alloc);
200+ char page_list_remove(char* virtual_addr, int success, unsigned int* pgdir);
201+ void swap_send(unsigned int* page_dir, int next_offset, unsigned int * next_pgdir, int idx);
```

쓸 함수들 정의 뒤에서 자세히
설명하겠다

```

710
→ 711+ int get_swap_idx(int offset, int block_idx){
712+     return offset * 8 + SWAPBASE + block_idx;
713+ }
714+
715+ void mov_buff(int from, int maximum_swap, int to)
716+ { // from to 둘다 offset
717+     int block_idx; //
718+     if(to && to < maximum_swap/8){
719+         for(
720+             block_idx = 0;
721+             block_idx < 8;
722+             block_idx++
723+         ) // Block enumerate
724+         {
725+             // Lock이 된 버퍼 할당
726+             struct buf* from_block = bread(
727+                 0, get_swap_idx(from, block_idx)
728+             );
729+             struct buf* to_block = bread(
730+                 0, get_swap_idx(to, block_idx)
731+             );
732+
733+             from_block->refcnt += 1;
734+             to_block->refcnt += 1;
735+
736+             memmove(to_block->data, from_block->data, 512);
737+             bwrite(to_block);
738+
739+
740+             from_block->refcnt -= 1;
741+             to_block->refcnt -= 1;
742+
743+             // 락 해제
744+             brelse(from_block);
745+             brelse(to_block);
746+         }
747+     }
748+     else{
749+         cprintf("Something wrong with condition!!");
750+         exit();
751+     }
752+ }
753

```

get swap idx는 오프셋과 블록 인덱스로 블록번호를 구하는 함수이다. 오프셋엔 8

곱하고 스왑베이스에 더하고 인덱스를 더한다.

mov buff는 한 버퍼에서 다른 새로운 버퍼로 안의 것들을 이동하는 함수이다

kalloc.c

```
14 // defined by the kernel
15+ char parity_check(void);
16+ void swap_out(struct page*);
17+
18+ struct spinlock paging_lock;
19+ struct spinlock clock_algorithm_lock;
20+ int pages_valid_bits[PHYSTOP/4096];
21
```

정의할 것들을 정의해줬다. 락(페이지락, lru락)과 valid pages들을 나타내는 배열을 만들었다.

```
Defined by the kernel link
#define PHYSTOP 0xE000000
Top physical memory
다음으로 확장:
0xE000000
[PHYSTOP/4096];
```

PHYSTOP는 탑 피지컬 메모리라 여기서 멈춰야한다.

<pre> 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"); </pre>	<pre> 70 void 71 kfree(char *v) 72 { 73 struct run *r; 74 75 if((uint)v % PGSIZE v < end V2P(v) >= PHYSTOP) 76 panic("kfree"); 77 else{ 78 num_free_pages++; 79 } </pre>
---	--

kfree에서 free를 성공하면 free pages수를 증가한다.

```

95 // Returns 0 if the memory cannot be allocated.
96 char*
97 kalloc(void)
98 {
99     struct run *r;
100     while(1){
101         if(kmem.use_lock)
102             acquire(&kmem.lock);
103         if(!(r = kmem.freelist))
104         {
105             if(!kmem.use_lock){
106                 if(!parity_check())
107                 {
108                     panic("OOM\n"); return 0;
109                 }
110             }
111             else{
112                 release(&kmem.lock);
113             }
114         }
115         else break;
116     }
117     num_free_pages--;
118     kmem.freelist = r->next;

```

alloc을 성공할때까지 무한루프를 돌리고
free page 수 줄인다. `parity_check`은 뒤에서 설명하겠다.

```
124+ void pagelist_insertion(  
125+     char* virtual_addr, int success, unsigned int *page_dir  
126+ )  
127+ {  
128+     acquire(&clock_algorithm_lock);  
129+  
130+     struct page* pge = 0;  
131+  
132+     int idx = 0;  
133+  
134+     for(  
135+         idx=0;  
136+         idx<0xE000000/0x1000;  
137+         idx++  
138+     )  
139+         if(!pages[idx].pgdir){  
140+             success = 1;  
141+             break;  
142+         }  
143+  
144+     if (success) pge = &pages[idx];  
145+  
146+     if(pge){  
147+         pge->pgdir=page_dir;  
148+         pge->vaddr=virtual_addr;  
149+  
150+         if(num_lru_pages < 0) cprintf("Wrong num lru condition!");  
151+         else if(num_lru_pages/2){  
152+             pge->next = page_lru_head;  
153+             pge->prev = page_lru_head->prev;  
154+  
155+             page_lru_head->prev->next=pge;  
156+             page_lru_head->prev = pge;  
157+         }  
158+         else if(num_lru_pages%2){  
159+             pge->next=page_lru_head;  
160+             pge->prev=page_lru_head;  
161+  
162+             page_lru_head->prev=pge;  
163+             page_lru_head->next=pge;  
164+         }  
165+         else{  
166+             page_lru_head=pge;  
167+  
168+             page_lru_head->next=pge;  
169+             page_lru_head->prev=pge;  
170+         }  
171+         release(&clock_algorithm_lock);  
172+     }  
173+     else{  
174+         cprintf("Page array does not have free space");  
175+     }
```

○ 페이지들의 리스트에 삽입하는 함수이다. 클락 알고리즘² 락을 잡고, **page** 리스트에서 삽입할 위치를 찾는다. 그다음 페이지들의 링크드 리스트를 연결하는데, 만약에 지금 현재 경우가 첫번째거나, 2번째거나 아니면 그 이후인지에 따라 행동이 달라진다. 간단한 링크드 리스트 삽입 알고리즘이므로 설명은 생략하겠다.

```

178+ char page_list_remove(
179+     char* virtual_addr, int success, unsigned int *page_dir
180+ )
181+ {
182+     char default_return_value = 'n';
183+
184+     acquire(&clock_algorithm_lock);
185+     unsigned int* page_table_entry= walkpgdir(
186+         page_dir, virtual_addr, 0
187+     );
188+
189+     if(num_lru_pages){
190+         if((*page_table_entry&0x100) >= 0x100) {
191+             struct page* pge = page_lru_head;
192+             while(1)
193+             {
194+                 if(
195+                     pge->vaddr == virtual_addr &&
196+                     pge->pgdir == page_dir
197+                 ){
198+                     if(pge!=page_lru_head){
199+                         pge->next->prev=pge->prev;
200+                         pge->prev->next=pge->next;
201+
202+                         num_lru_pages -= 1;
203+                         break;
204+                     }
205+                     else{
206+                         page_lru_head=pge->next;
207+                     }
208+                 }
209+                 struct page* before = pge;
210+                 pge=pge->next;
211+                 if(before == pge){
212+                     break;
213+                 }
214+             }
215+         }
216+         else
217+         {
218+             pages_valid_bits[
219+                 *page_table_entry/4096
220+             ]=num_lru_pages;
221+             success = 1;
222+         }
223+     }
224+
225+     release(&clock_algorithm_lock);
226+     if(!success) return default_return_value;
227+     else return 'e';
228+ }

```

페이지를 지울때 사용. 원리가 삽입과 비슷하므로 설명은 생략하겠다. 'e'는 삭제가 성공한 경우에 리턴

```
230+ void page_fault_handle(  
231+     unsigned int trap_no, unsigned int faddress, unsigned int *page_dir  
232+ )  
233+ {  
234+     if(trap_no!=14) panic("Wrong trap condition!");  
235+     unsigned int* fault_entry = walkpgmdir(  
236+         page_dir, (void*)faddress, 0  
237+     );  
238+     if(*fault_entry&0x100){  
239+         char parity_check_value = parity_check();  
240+         if(  
241+             !parity_check_value && !num_free_pages  
242+         ) panic("parity_check failed\n");  
243+         char* allocated_memory = kalloc();  
244+         if(allocated_memory){ // success  
245+             pages_valid_bits[*fault_entry / PGSIZE] = 0;  
246+             memset(allocated_memory, 0, 4096);  
247+             *fault_entry = V2P(allocated_memory) |  
248+                 (*fault_entry % PGSIZE - 0b1111111111);  
249+             pagelist_insertion((char*)faddress, 0, page_dir);  
250+             swpread(allocated_memory,*fault_entry/PGSIZE);  
251+         }  
252+     }  
253+ }
```

페이지 폴트 핸들링하는 함수이다.
페이지 공간이 충분하면 parity check을
하고 kalloc을 호출, 페이지폴트난 곳을
스왑read.


```

262+ char parity_check()
263+ {
264+     char return_value = 0;
265+     char success = 1;
266+
267+     struct page* position = page_lru_head;
268+     if(!num_lru_pages) return return_value;
269+
270+ go_on:
271+ {
272+     unsigned int* pte = walkpgdir(
273+         position->pgdir,
274+         (void*)position->vaddr,
275+         return_value
276+     );
277+     if(!((*pte&0x020)) && ((*pte&0x004)))
278+     {
279+         swap_out(position);
280+         return success;
281+     }
282+     if(page_lru_head != position->next){
283+         position=position->next;
284+         goto go_on;
285+     }
286+     else{
287+         goto end_option;
288+     }
289+ }
290+ end_option:
291+ {
292+     position=page_lru_head;
293+     char found=0;
294+
295+     int page_num;
296+     char is_done = 0;
297+
298+     for(
299+         page_num=0;
300+         page_num<num_lru_pages;
301+         page_num++
302+     )
303+     {
304+         if(!is_done){
305+             if(
306+                 (*walkpgdir(
307+                     position->pgdir, (void*)position->vaddr, (int) return_value
308+                 )
309+                 &0x004)
310+             ){
311+                 is_done = 1;
312+                 found=1;
313+             }
314+             position=position->next;
315+         }
316+     }
317+     char value_to_return;
318+     if(!found){
319+         value_to_return = return_value;
320+     }
321+     else{
322+         swap_out(position);
323+         value_to_return = success;
324+     }
325+     return value_to_return;
326+ }
327+ }

```

함수 작명을 잘못된 듯 하긴 하다. 최대한 lru head쪽에 있는 페이지 중 0x20와 bitwise and했을때 0인 페이지를 찾는다. 찾았을때 있으면?-> 스왑아웃 없으면?-> 제일 앞에있는 lru head를 스왑아웃 victim으로 정함

```
330+ void swap_send(  
331+     unsigned int * page_dir, int next_offset, unsigned int * next_pgdir, int idx  
332+ )  
333+ {  
334+     char done = 0;  
335+     int j_idx;  
336+  
337+     unsigned int* page_table_entry = walkpgdir(  
338+         page_dir,  
339+         (void*)idx,  
340+         0  
341+     );  
342+     int before_offset = *page_table_entry/PGSIZE;  
343+  
344+     for(j_idx=0; j_idx<PHYSTOP/4096; j_idx++)  
345+     {  
346+         if (!j_idx) continue;  
347+         if(!pages_valid_bits[j_idx] && !done)  
348+         {  
349+             next_offset=j_idx;  
350+             done = 1;  
351+         }  
352+     }  
353+     mov_buff(before_offset,SWAPMAX,next_offset);  
354+  
355+     unsigned int* next_entry = walkpgdir(next_pgdir, (void*) idx, 0);  
356+     *next_entry= (*page_table_entry%PGSIZE)+(next_offset*PGSIZE);  
357+ }
```

스왑send.copyuvm에서 현 스왑과 똑같은 애를 만들어 전달할때 쓴다.

```

359+ void swap_out(struct page* pages_to_out)
360+ {
361+     int out_offset=0;
362+     char done = 0;
363+     int idx = 0;
364+
365+     acquire(&paging_lock);
366+
367+     unsigned int* pte = walkpgdir(
368+         pages_to_out->pgdir,
369+         (void*)pages_to_out->vaddr, 0
370+     );
371+
372+     for(
373+         idx=1;
374+         idx<PHYSTOP/PGSIZE;
375+         idx++
376+     ){
377+         if (!idx) continue;
378+         if(pages_valid_bits[idx]==0 && !done)
379+         {
380+             out_offset=idx;
381+             done=1;
382+         }
383+     }
384+
385+     pages_valid_bits[out_offset]=1;
386+
387+     page_lru_head=pages_to_out->next;
388+     num_lru_pages--;
389+
390+
391+     pages_to_out->prev->next=pages_to_out->next;
392+     pages_to_out->next->prev=pages_to_out->prev;
393+
394+
395+     swapwrite((char*)P2V(
396+         PTE_ADDR((out_offset*PGSIZE) | 0x100 | (*pte%PGSIZE-1))
397+     ),out_offset);
398+     kfree((char*)P2V(PTE_ADDR(
399+         (out_offset*PGSIZE) | 0x100 | (*pte%PGSIZE-1)
400+     )));
401+
402+     release(&paging_lock);
403+
404+ }

```

스왑아웃 함수이다. walkpgdir해서
스왑타겟d의 offset을 구한다.
스왑되었는지 아닌지 구분할 방법도
정의하고, 쫓아내면 pages_valid_bits 에
마킹해서 추후 핸들링할때 편리하게
했다.

proc.c

```
14  
15 static struct proc *initproc;  
→ 16+ extern struct spinlock paging_lock;  
17+ extern struct spinlock clock_algorithm_lock;  
18  
19 int nextpid = 1;
```

락이다.

```

122 void
123 userinit(void)
124 {
125     struct proc *p;
126     extern char _binary_initcode_start[], _binary_initcode_size[];
127
128     p = allocproc();
129
130     initproc = p;
131     if((p->pgdir = setupkvm()) == 0)
132         panic("userinit: out of memory?");
133     inituvm(p->pgdir, _binary_initcode_start, (int)_binary_initcode_size);
134     p->sz = PGSIZE;
135     memset(p->tf, 0, sizeof(*p->tf));
136     p->tf->cs = (SEG_UCODE << 3) | DPL_USER;
137     p->tf->ds = (SEG_UDATA << 3) | DPL_USER;
138     p->tf->es = p->tf->ds;
139     p->tf->ss = p->tf->ds;
140     p->tf->eflags = FL_IF;
141     p->tf->esp = PGSIZE;
142     p->tf->eip = 0; // beginning of initcode.S
143
144     safestrcpy(p->name, "initcode", sizeof(p->name));
145     p->cwd = namei("/");
146
147     // this assignment to p->state lets other cores
148     // run this process. the acquire forces the above
149     // writes to be visible, and the lock is also needed
150     // because the assignment might not be atomic.
151     acquire(&ptable.lock);
152
153     initlock(
154         &clock_algorithm_lock,
155         "clock_algorithm"
156     );
157     p->state = RUNNABLE;
158     initlock(
159         &paging_lock,
160         "paging"
161     );
162

```

첨에 초기화할때 락 만들어준다.(init)

```
461     }  
462 }  
→ 463+ int getpid()  
464+ {  
465+     struct proc* curproc = myproc();  
466+     int got_pid = curproc->pid;  
467  
→ 468+     return got_pid;  
469+ }  
470 //PAGEBREAK!
```

뭐가 문제인지 모르겠으나 **getpid**를 정의 안하면 에러가 나서 정의해줬다.

trap.c

```

47 /
48
49 switch(tf->trapno){
50 case T_IRQ0 + IRQ_TIMER:
51     if(cpuid() == 0){
52         acquire(&tickslock);
53         ticks++;
54         wakeup(&ticks);
55         release(&tickslock);
56     }
57     lapiceoi();
58     break;
59 case T_IRQ0 + IRQ_IDE:
60     ideintr();
61     lapiceoi();
62     break;
63 case T_IRQ0 + IRQ_IDE+1:
64     // Bochs generates spurious IDE1 interrupts.
65     break;
→ 66+ case T_PGFLT:
67+     if (tf->trapno!=14){
68+         cprintf("Something wrong");
69+     }
70+     //cprintf("T_PGFLT");
71+     struct proc* curproc = myproc();
72+     page_fault_handle(
73+         tf->trapno, PGROUNDOWN(rcr2()), curproc->pgdir
74+     );
75+     break;
76 case T_IRQ0 + IRQ_KBD:
77     kbdintr();
78     lapiceoi();
79     break;
80 case T_IRQ0 + IRQ_COM1:
81     uartintr();
82     lapiceoi();
83     break;
84 case T_IRQ0 + 7:
85 case T_IRQ0 + IRQ_SPURIOUS:
86     cprintf("cpu%d: spurious interrupt at %x:%x\n",
87         cpuid(), tf->cs, tf->eip);
88     lapiceoi();
89     break;

```

트랩에서 페이지 폴트를 핸들링 하는
부분. 페이지 폴트 핸들러 함수를
부른다. 아까 정의한.

vm.c

```
33 // that corresponds to virtual address va. If alloc!=0,
34 // create any required page table pages.
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 they can
50         // be further restricted by the permissions in the page table
51         // entries, if necessary.
52         *pde = V2P(pgtab) | PTE_P | PTE_W | PTE_U;
53     }
54     return &pgtab[PTX(va)];
55 }
```

저번 과제도 그렇고 이 앞에 **static**을 안
떼면 왜 에러가 나오는지 모르겠으나,
일단 떼주었다.


```

179
180 // Load the initcode into address 0 of pgdir.
181 // sz must be less than a page.
182 void
183 inituvm(pde_t *pgdir, char *init, uint sz)
184 {
185     char *mem;
186
187     if(sz >= PGSIZE)
188         panic("inituvm: more than a page");
189     mem = kalloc();
190     memset(mem, 0, PGSIZE);
191     mappages(pgdir, 0, PGSIZE, V2P(mem), PTE_W|PTE_U);
192     memmove(mem, init, sz);
193+    if(0x004 & *(walkpgdir(pgdir, 0, 0))){
194+        pagelist_insertion(0, 0, pgdir);
195+    }
196 }
197

```

inituvm 에서 페이지 리스트에
삽입하였다.

```

223 // newsz, which need not be page aligned. Returns new size or 0 on error.
224 int
225 allocuvm(pde_t *pgdir, uint oldsz, uint newsz)
226 {
227     char *mem;
228     uint a;
229
230     if(newsz >= KERNBASE)
231         return 0;
232     if(newsz < oldsz)
233         return oldsz;
234
235     a = PGROUNDUP(oldsz);
236     for(; a < newsz; a += PGSIZE){
237         mem = kalloc();
238         if(mem == 0){
239             cprintf("allocuvm out of memory\n");
240             deallocuvm(pgdir, newsz, oldsz);
241             return 0;
242         }
243         memset(mem, 0, PGSIZE);
244         if(mappages(pgdir, (char*)a, PGSIZE, V2P(mem), PTE_W|PTE_U) < 0){
245             cprintf("allocuvm out of memory (2)\n");
246             deallocuvm(pgdir, newsz, oldsz);
247             kfree(mem);
248             return 0;
249         }
250+
251+         if((*walkpgdir(pgdir, (char*)a, 0) & 0b101) == 0b101){
252+             pagelist_insertion((char*)a, 0, pgdir);
253+         }
254     }
255     return newsz;
256 }

```

allocuvm도 이렇게 핸들링

```

261 // process size. Returns the new process size.
262 int
263 deallocvm(pde_t *pgdir, uint oldsz, uint newsz)
264 {
265     pte_t *pte;
266     uint a, pa;
267
268     if(newsz >= oldsz)
269         return oldsz;
270
271     a = PGROUNDUP(newsz);
272     for(; a < oldsz; a += PGSIZE){
273         pte = walkpgdir(pgdir, (char*)a, 0);
274         if(!pte)
275             a = PGADDR(PDX(a) + 1, 0, 0) - PGSIZE;
276         else if((*pte & PTE_P) != 0){
277             pa = PTE_ADDR(*pte);
278             if(pa == 0)
279                 panic("kfree");
280+         if(page_list_remove((char*)a, 0, pgdir)=='n'){
281             char *v = P2V(pa);
282             kfree(v);
283             *pte = 0;
284         }
285     }
286+ }
287     return newsz;
288 }

```

deallocvm은 삭제하는 부분 만들었다.
 아까 page_listremove에서 성공했는지
 나타내는 'n', 성공했을때 나타내는 'e'를
 이요한다.

```

323 // of it for a child.
324 pde_t*
325 copyuvm(pde_t *pgdir, uint sz)
326 {
327+ char insertion_condition = 0;
328 pde_t *d;
329 pte_t *pte;
330 uint pa, i, flags;
331 char *mem;
332
333 if((d = setupkvm()) == 0)
334     return 0;
335 for(i = 0; i < sz; i += PGSIZE){
336     if((pte = walkpgdir(pgdir, (void *) i, 0)) == 0)
337         panic("copyuvm: pte should exist");
338     if(!(*pte & PTE_P))
339+ {
340+         if(!(*pte & 0x100)){
341             panic("copyuvm: page not present");
342+         }
343+         else{
344+             swap_send(pgdir, 0, d, i);
345+         }
346+     }
347+     else
348+     {
349         pa = PTE_ADDR(*pte);
350         flags = PTE_FLAGS(*pte);
351         if((mem = kalloc()) == 0)
352             goto bad;
353         memmove(mem, (char*)P2V(pa), PGSIZE);
354         if(mappages(d, (void*)i, PGSIZE, V2P(mem), flags) < 0) {
355             kfree(mem);
356             goto bad;
357+         }
358+     }
359+
360+     if((*pte & 0b101) == 0b101){
361+         insertion_condition = 1;
362+     }
363+
364+     if(insertion_condition){
365+         pagelist_insertion((char*) i, 0, d);
366+     }
367 }
368 }
369 return d;
370

```

copyuvm. 아까 스왑아웃에서 정의한 방법으로 스왑되었는지 확인하고 처리한다. **swapsend**로 똑같은 것을 만들어 주는 부분도 처리. **pagelist**에 삽입하는 부분도 만들어주었다.