OS Project-4 Report

PAGE DEMAND - Clock Algorithm

Software of Department

2016312568

Jung Hee Yoon

이번 Project 3는 페이지를 관리하는 기능을 만드는 것이다. 페이지를 관리하는 알고리즘은 clock algorithm 이다. 이를 위해서는 페이지를 링크드리스트로 연결하기위하여 struct page pages[]배열을 만든다. 또한 스왑하는 위치 (이하 스왑스택)에 빈공간을 찾기 위하여 valid[]배열을 만들었다. 페이지 스왑이 가능한 종류는 user page이므로 이 프로그램내에서 스왑가능 한 기준은 PTE_U를 만족하는 것으로 하였다. 처음에 들어온 pte의 값에서 처음에는 PTE_U를 만족하지만 곧 PTE_U를 Clear한다. 고로 나는 reclaim함수 내에서 PTE_U를 만족하는 것을 구하였으며 만약 PTE_U가 만족하는 것이 없거나 lru 링크드 리스트가 비었을 경우에 return 0을 하여준다. 자세한 설명은 코드를 첨부하여 하겠다.

1. kalloc function

아래의 코드는 kalloc.c에 있던 기존 함수를 약간 변형한 것이다. kem.freelist에 free한 공간이 있으면 이를 리턴하고 아니면 reclaim을 시도한다. reclaim은 clock algorithm으로 lru 링크드 리스트에서 빼줄 페이지를 선택한후 Swap write하여주는 함수이다. reclaim을 실패하는 경우는 lru가 없을경우 즉, 유저페이지 (Swapable page)가 없을경우에는 return 0을 하여준다.

```
char*
kalloc(void)
{
  struct run *r;
try_again:
  if(kmem.use_lock)
    acquire(&kmem.lock);
  r = kmem.freelist;
  if(!r)
  {
      if(kmem.use_lock) release(&kmem.lock);
      if(!reclaim())
      {
          cprintf("fault reclaim() : Out Of Memory\n");
          return 0;
      }
      goto try_again;
  }
  if(r)
  {
      kmem.freelist = r->next;
      num_free_pages--;
  }
  if(kmem.use_lock)
    release(&kmem.lock);
  return (char*)r;
}
```

2. reclaim function

리클레임 함수는 위에서 말했던 것처럼 head와 가까우며 PTE_A가 0인 페이지를 찾아 SwapOut하여준다. 만약 PTE_A가 0인 페이지가 없다면 헤드를 Swap Out하여준다.

```
120 int reclaim()
121 {
122     print_lru_list();
123     acquire(&lrulock);
124     if(num_lru_pages<=0) {
125         release(&lrulock);
126         return 0;
127     }
128 
129     struct page* n = page_lru_head;
130     while(1)
131     {
132         pte_t* pte = walkpgdir(n->pgdir,(void*)n->vaddr,0);
133         if(((*pte&PTE_U)!=0)&&((*pte&PTE_A)==0))
134         {
135             swap_out(n->vaddr,n->pgdir);
136             return 1;
137         }
138         if(n->next==page_lru_head) break;
139         n=n->next;
140     }
141     int ck=0;
142     n=page_lru_head;
143     for(int i=0;i<num_lru_pages;i++)
144     {
145         pte_t* pte = walkpgdir(n->pgdir,(void*)n->vaddr,0);
146         if((*pte&PTE_U)!=0) {
147             ck=1;
148             break;
149         }
150         n=n->next;
151     }
152     if(ck)
153     {
154         swap_out(n->vaddr,n->pgdir);
155         return 1;
156     }
157     release(&lrulock);
158     return 0;
159 }
```

3. swap_out function

Swap_out 함수는 스왑할 장소의 offset에 해당하는 pte의 값으로 수정하여주고 플래그에서 PTE_P를 clear한다. 이때 나는 mmu.h에 PTE_SWAP = 0x100이라는 플래그를 새로 주었으며 PTE_SWAP이 set되면 스왑되어있는 것이고 PTE_SWAP이 clear되어있으면 스왑되지 않은 상태이다. 나는 스왑스택에서 빈공간을 찾기위하여 valid[] 배열을 선언하였으며 각 공간에 스왑이 되어있으면 1, 아니면 0으로 세팅하여준다.

```
5 void swap_out(struct page* n)
6 {
7     acquire(&pagelock);
8     pte_t* pte = walkpgdir(n->pgdir,(void*)n->vaddr,0);
9     int offset=0;
0     for(int i=1;i<PHYSTOP/PGSIZE;i++)
1     {
2         if(valid[i]==0)
3         {
4             offset=i;
5             break;
6         }
7     }
8
9     uint origin_addr = PTE_ADDR(*pte);
0     int flags = *pte%4096-1;
1     valid[offset]=1;
2     page_lru_head=n->next;
3     n->prev->next=n->next;
4     n->next->prev=n->prev;
5     n->prev=0;
6     n->next=0;
7     n->pgdir=0;
8     n->vaddr=0;
9     num_lru_pages--;
0     *pte = (offset*4096)|PTE_SWAP|flags;
1     release(&pagelock);
2     swapwrite((char*)P2V(origin_addr),offset);
3     kfree((char*)P2V(origin_addr));
4     return;
5 }
```

4. add_pglist function

```
void add_pglist(char* va, pde_t *pgdir)
{
    acquire(&lrulock);
    struct page* n = find_page();
    if(n==0) panic("Full pages array");
    num_lru_pages++;

    n->vaddr=va;
    n->pgdir=pgdir;

    if(num_lru_pages==1)
    {
        page_lru_head=n;
        page_lru_head->prev=page_lru_head;
        page_lru_head->next=page_lru_head;
    }
    else if(num_lru_pages==2)
    {
        n->prev=page_lru_head;
        n->next=page_lru_head;
        page_lru_head->next=n;
        page_lru_head->prev=n;
    }
    else
    {
        n->prev = page_lru_head->prev;
        n->next = page_lru_head;
        page_lru_head->prev->next=n;
        page_lru_head->prev = n;
    }
    release(&lrulock);
}
```

이 함수는 vm.c에서 inituvm, copyuvm, allocuvm 함수내에서 적혀있는 함수로, page 가 추가될때 virtual address와 page directory를 가져와서 그에 해당하는 페이지 배열에 삽입하고 링크드 리스트형태로 만들어 준다. 이때 값이 영향이 받지 않도록 락을 걸어주었다.

5. print_lru_list function

아래의 함수는 lru의 리스트를 출력하여 주는 함수이다. 디버깅용으로 사용하였다.

```
229 }
230 void print_lru_list(void)
231 {
232     cprintf("------------lru page list----num_lru_pages=%d, num_free_p
233     int i=0;
234     struct page* n = page_lru_head;
235     while(1)
236     {
237         cprintf("%dth *pte = %x\n",i,*walkpgdir(n->pgdir,(void*)n->vad
238         if(n->next==page_lru_head) break;
239         i++;
240         n=n->next;
241     }
242
```

6. delete_pages function

이는 deallocuvm에 사용되는 함수이며 각 페이지를 remove할때 사용된다.

```
int delete_pages(char* va, pde_t* pgdir)
{
    acquire(&lrulock);
    pte_t* pte= walkpgdir(pgdir,va,0);
    if(num_lru_pages==0) {
        release(&lrulock);
        return 0;
    }
    int index=*pte/4096;
    if((*pte)&PTE_SWAP)
    {
        valid[index]=0;
        release(&lrulock);
        return 2;
    }
    else
    {
        struct page* n = page_lru_head;
        for(;;)
        {
            if(n->pgdir == pgdir && n->vaddr == va)
            {
                if(n==page_lru_head) page_lru_head=n->next;
                n->prev->next=n->next;
                n->next->prev=n->prev;
                n->prev=0;
                n->next=0;
                n->pgdir=0;
                n->vaddr=0;
                num_lru_pages--;
                release(&lrulock);
                return 1;
            }
            n=n->next;
            if(n==page_lru_head) break;
        }
    }
    release(&lrulock);
    return 0;
}
```

7. PGFLT_proc function

여기는 trap.c에서 T_PGFLT의 플래그가 반환되었을때 PAGEFAULT HANDLER을 해주는 함수이다. 남은 페이지가있다면 reclaim을 하지않고 남은 페이지가 없다면 reclaim을 한후에 kalloc()을 시켜 PAGEFAULT된 주소를 SWAPREAD하여 주어 PAGEFAULT를 처리한다.

```
79 void PGFLT_proc(uint fault_addr, pde_t *pgdir)
80 {
81
82     pte_t* pte_fault = walkpgdir(pgdir,(void*)fault_addr,0);
83     int swap_index = *pte_fault/4096;
84     if((*pte_fault&PTE_SWAP)==0) return;
85     if(num_free_pages==0&&reclaim()==0)
86     {
87         cprintf("reclaim fault\n");
88         exit();
89     }
90     char* mem = kalloc();
91     if(mem==0) return;
92     valid[swap_index]=0;
93     memset(mem,0,PGSIZE);
94     *pte_fault=V2P(mem)|(*pte_fault%4096+1-0x100);
95     add_pglist((char*)fault_addr,pgdir);
96     swapread(mem,swap_index);
97 }
```

8. Find_pages

add page할때에 있어 pages배열에서 빈공간을 찾아서 리턴해주는 함수이다.

```
6 struct page* find_page()
7 {
8     for(int i=0;i<PHYSTOP/PGSIZE;i++)
9         if(pages[i].pgdir==0) return &pages[i];
0     return 0;
1 }
```

9. Copy swap

이 함수는 Copyuvm에서 swaped인 페이지를 복사하여줄때 쓰이는 함수이다.

```
306 void copy_swap(pde_t* pgdir, pde_t* new_pgdir, int i)
307 {
308     pte_t* pte = walkpgdir(pgdir,(void*)i,0);
309     int oldoff=*pte/4096;
310     int newoff=0;
311     for(int i=1;i<PHYSTOP/PGSIZE;i++)
312     {
313         if(valid[i]==0)
314         {
315             newoff=i;
316             break;
317         }
318     }
319     copy_swap_data(oldoff,newoff);
320     pte_t* new_pte=walkpgdir(new_pgdir,(void*)i,0);
321     *new_pte=(newoff*4096)+(*pte%4096);
322 }
```

10. Copy_swap_data

이 함수는 Copy_swap에서 부모의 버퍼에 있는 데이터를 옮기는 함수이다.

```
700 void copy_swap_data(int oldoff, int newoff)
701 {
702     struct buf* new;
703     struct buf* old;
704     if( newoff < 0 || newoff >= SWAPMAX/8)
705         panic("copy_swap : blkno exceed range");
706
707     for(int i=0;i<8;i++)
708     {
709         new = bread(0, newoff*8+SWAPBASE+i);
710         old = bread(0, oldoff*8+SWAPBASE+i);
711         memmove(new->data, old->data, BSIZE);
712         bwrite(new);
713         brelse(old);
714         brelse(new);
715     }
716 }
```

Example 1. fork test

아래의 함수는 forktest.c파일로써 fork함에 있어 문제가 없는지 확인하는 파일이다. 여기서는 fork를 1000번하여서 문제가 없는지 확인하는 코드이다. 이를 검사하기 위해서 kinit2의 범위를 다음과 같이 낮추고 하였다. 이럴 경우에 대게 free pages는 150개 가량생기며 게다가 fork에서 생기는 setkvm으로 인하여 User permission이 없는 경우로, 가장 아래와 같은 Out of memory 출력이 뜨며 fork가 종료된다. 또한 이후에 kfree에러가 뜨기도 하는데 이는 reclaim이 0을 리턴하였는데 panic이나 프로세스를 죽이지 않음으로 나오는 현상이다.

```
File  Edit  View  Search  Terminal  Help
11 printf(int fd, const char *s, ...)
12 {
13   write(fd, s, strlen(s));
14 }
15
16 void
17 forktest(void)
18 {
19   int n, pid;
20
21   printf(1, "fork test\n");
22
23   for(n=0; n<N; n++){
24     pid = fork();
25     if(pid < 0)
26       break;
27     if(pid == 0)
28       exit();
29   }
30
31   if(n == N){
32     printf(1, "fork claimed to work N times!\n", N);
33     exit();
34   }
35
36   for(; n > 0; n--){
37     if(wait() < 0){
38       printf(1, "wait stopped early\n");
39       exit();
40     }
41   }
42
43   if(wait() != -1){
44     printf(1, "wait got too many\n");
45     exit();
46   }
47
48   printf(1, "fork test OK\n");
49 }
50
51 int
52 main(void)
53 {
54   forktest();
```

```
17 int
18 main(void)
19 {
20   kinit1(end, P2V(4*1024*1024)); // phys page all
21   kvmalloc();      // kernel page table
22   mpinit();        // detect other processors
23   lapicinit();     // interrupt controller
24   seginit();       // segment descriptors
25   picinit();       // disable pic
26   ioapicinit();    // another interrupt controlle
27   consoleinit();   // console hardware
28   uartinit();      // serial port
29   pinit();         // process table
30   tvinit();        // trap vectors
31   binit();         // buffer cache
32   fileinit();      // file table
33   ideinit();       // disk
34   startothers();   // start other processors
35   kinit2(P2V(4*1024*1024), P2V(4*1024*1024+100));
36   userinit();      // first user process
37   mpmain();        // finish this processor's set
38 }
39
```

```
jhy@ubuntu:~/Desktop/xv6_init2$ XV6
qemu-system-i386 -nographic -drive file=fs.img,index=1,media=disk,format=r
aw -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
xv6...
cpu1: starting 1
8010b460 2c
cpu0: starting 0
sb: size 100000 nblocks 99917 ninodes 200 nlog 30 logstart 2 inodestart 32
 bmap start 58
init: starting sh
$ forktest
fork test
fault reclaim() : Out Of Memory
fork test OK
```

Example 2. test file

아래의 코드는 memory를 말록으로 받아와서 메모리를 직접적으로 건드린뒤 맞게 결과가 나오는지 확인하는 코드이다. 결과는 우측과같다. 또한 이를 포함하는 중간과정에서 PGFLT, Swap out, Add, Delete, Reclaim등 전부를 테스트 할 수있다. 이를 중간과정으로 보면서 확인은 하였으나 페이지를 적당히 줄여 테스트를 해보아도 페이지가 200개가 넘어 첨부는 못하였다.

```
10
11 char buf[8192];
12 char name[3];
13 char *echoargv[] = { "echo", "ALL", "TESTS", "PASSED", 0 };
14 int stdout = 1;
15 //#define TOTAL_MEMORY (2 << 20) + (1 << 18) + (1 << 17)
16 #define TOTAL_MEMORY (2<<19) //+ (1<<18) + (1<<17)
17 void
18 mem(void)
19 {
20     void *m1 = 0, *m2, *start;
21     uint cur = 0;
22     uint count = 0;
23     uint total_count;
24
25     printf(1, "mem test\n");
26
27     m1 = malloc(4096);
28     if (m1 == 0)
29         goto failed;
30     start = m1;
31
32     while (cur < TOTAL_MEMORY) {
33         m2 = malloc(4096);
34         if (m2 == 0)
35             goto failed;
36         *(char**)m1 = m2;
37         ((int*)m1)[2] = count++;
38         m1 = m2;
39         cur += 4096;
40     }
41     ((int*)m1)[2] = count;
42     total_count = count;
43
44     count = 0;
45     m1 = start;
46
47     while (count != total_count) {
48         if (((int*)m1)[2] != count)
49             goto failed;
50         m1 = *(char**)m1;
51         count++;
52     }
53
```

```
qemu-system-i386 -nographic -drive file=fs.img,ind
ive file=xv6.img,index=0,media=disk,format=raw -sm
xv6...
cpu1: starting 1
8010b460 2c
cpu0: starting 0
sb: size 100000 nblocks 99917 ninodes 200 nlog 30
start 58
init: starting sh
$ test1
memtest starting
mem test
mem ok
$
```