OS project 4

2016310936 우승민

이번 프로젝트의 목표는 **swapwrite, swapread** 두개의 주어진 함수를 사용하여 xv6 가 **page swaping** 을 하도록 구현하여 free memory 가 부족할 때 **swap out** 을 하여 빈 공간을 활용하고, **page faul** 가 발생할 때 **swap in** 을 가능하게 만드는 것입니다.

```
// Page table/directory entry flags.
#define PTE_P           0x001   // Present
#define PTE_W           0x002   // Writeable
#define PTE_U           0x004   // User
#define PTE_PS          0x080   // Page Size
#define PTE_A           0x020
```

우선 **mmu.h** 헤더파일에 lru list 의 **clock algorithm** 을 위해 PTE 에 **PTE_A flag** 를 추가해주었습니다.

```
// kalloc.c
char*           kalloc(void);
void            kfree(char*);
void            kinit1(void*, void*);
void            kinit2(void*, void*);
void            mklist(pde_t *pgdir, char *vaddr, uint pa);
void            dllist(pde_t *pgdir, char *vaddr);
int             page_handler(uint rcr2);
"defs.h" 197L, 5723C written
```

```
struct spinlock bitmap;
```

```
extern char bitm[4096];

struct page pages[PHYSTOP/PGSIZE];
struct page *page_lru_head;
int SS;


struct spinlock lru;
```

Mklist 와 **dllist** 는 **lru list** 를 insert, remove 해주는 함수입니다. **kalloc** 함수에 선언하였지만, **vm.c** 에서도 사용하므로 **defs.h** 에 선언해주었습니다. **Page_handler** 는 **trap.c** 에서 page_fault 가 발생하였을 때 다루도록 만들어주었습니다. **bitm** 은 **swap space** 를 관리하기 위해 만들어준 **bitmap** 입니다. **Vm.c** 에서도 사용할 수 있도록 extern 해주었습니다. **Lru** 와 bitmap 은 thread 간에 공유되는 data 이므로 **lock** 을 만들어주었습니다. **SS** 변수는 **lock** 을 위한 **conditional variable** 인데 나중에 다시 설명하겠습니다.

Lru list 를 연결하는 과정은 **mklist** 함수를 통해서 해주었습니다.

```
void mklist(pte_t *pgdir, char *vaddr, uint pa)
{
  acquire(&lru);
  struct page *p;
  p = &pages[pa/PGSIZE];
  p->pgdir = pgdir;
  p->vaddr = vaddr;
  if(!page_lru_head){
        page_lru_head = p;
        page_lru_head -> next = p;
        page_lru_head -> prev = p;
  }
  else{
        p -> prev = page_lru_head -> prev;
        p -> next = page_lru_head;
  }
  page_lru_head -> prev -> next = p;
  page_lru_head -> prev = p;
  release(&lru);
}
```

page directory 와 **virtual address** 를 받아 list 각각에 넣어주었고, **physical address** 로 hash table 의 key 값을 지정해주었습니다.

그리고 기존에 있던 page 의 마지막으로 insert 해주면서 head 의 prev 로 연결해주었습니다.

```
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_A|PTE_W|PTE_U);
  mklist(pgdir,0, V2P(mem));
  memmove(mem, init, sz);
}
```

```
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_A|PTE_W|PTE_U) < 0){
      cprintf("allocuvm out of memory (2)₩n");
      deallocuvm(pgdir, newsz, oldsz);
      kfree(mem);
      return 0;
    }
    mklist(pgdir,(char *)a, V2P(mem));
  }
  return newsz;
}
```

kernel 영역이 아닌 **user pages** 들만 **swappable** 하기 때문에 **inituvm** **allocuvm** **copyuvm** 함수에서만 **mklist** 함수를 호출하도록 하였습니다. 또한 **mappages** 함수를 호출할 때 flag 에 PTE_A 를 추가해 주었습니다.

copyuvm 함수의 경우 고려해야 할 점이 더 있기 때문에 뒤에서 설명하겠습니다.

```
char*
kalloc(void)
{
  struct run *r;
  try_again :
  while(SS);
  if(kmem.use_lock)
    acquire(&kmem.lock);
  r = kmem.freelist;
  if(r){
    kmem.freelist = r->next;
  } else{
    SS = 1;
    acquire(&bitmap);
    int i=0;
    while(bitm[i])
        i++;
    release(&bitmap);
    acquire(&lru);
    struct page *p;
    p = page_lru_head;
    if(!p)
        panic("OOM ERROR₩n");
    pte_t *pte;
    pte = walkpgdir(p->pgdir, (void *)(p->vaddr), 0);
    while((*pte &PTE_A)==PTE_A){
        *pte = *pte - PTE_A;
        p = p -> next;
        pte = walkpgdir(p->pgdir, (void *)p->vaddr, 0);
    }
    release(&lru);
```

```
    char *mem = P2V(PTE_ADDR(*pte));
    if(kmem.use_lock)
     release(&kmem.lock);
    swapwrite(mem, i);

    if(kmem.use_lock)
     acquire(&kmem.lock);
    SS = 0;
    acquire(&lru);
    acquire(&bitmap);
    bitm[i] = 1;
    release(&bitmap);
    lcr3(V2P(p->pgdir));
    p-> prev ->next = p-> next;
    p-> next -> prev = p-> prev;
    page_lru_head = p -> next;
    *pte = *pte & 0x00000FFF;
    *pte = *pte & ~PTE_P;
    *pte = *pte | i<<12;
    release(&lru);
    if(kmem.use_lock)
      release(&kmem.lock);
    kfree(mem);
    goto try_again;
  }
  if(kmem.use_lock)
    release(&kmem.lock);
  return (char*)r;
}
```

kalloc 함수에서 free page 가 없을 경우에는 else 문으로 들어가게 되어 6 개의 과정을 진행합니다.

1. while 문으로 **bitmap** 의 빈공간을 찾아줍니다.
2. while 문에서 ***pte** 와 PTE_A 값을 비교하여 오래된 파일부터 list 를 돌며 **swap out** 할 page 를 골라줍니다.
3. **Swap out** 을 진행하고 해당되는 위치의 **bitmap** 을 set 합니다.
4. 해당 page 를 **lru list** 에서 제거합니다.
5. ***pte** 값을 flag 부문만 남기고 PTE_P 를 지운 후 **PFN field** 에 swap space 의 **offset** 을 넣어줍니다.
6. Swap 한 page 를 free 해준 후 다시 **kalloc** 함수 시작부문으로 돌아갑니다.

위 코드에서 **SS** 라는 **conditional variable** 을 사용해주었는데, 이유는 **swapwrite** 를 진행할 때 어느 하나라도 lock 이 걸려있으면 오류가 발생하여 **kmem.lock** 을 release 해야하는데 만약 **swapwrite** 를 진행하는 동안 다른 thread 에서 **kalloc** 을 들어와서 **kmem.lock** 의 acquire 뺏을 수 있기 때문에 **SS** 라는 변수를 else 문 시작부문에 1 로 만들어 주고, **swapwrite** 진행 후 **kmem.lock** 의 acquire 을 얻은 후 **SS** 를 0 으로 바꾸어 **kalloc** 함수의 while(SS)문을 지나갈 수 있기 해주었습니다.

또한 bitmap 과 lru list 각각을 수정할 때 또한 **lock** 을 걸어주었습니다. 추가적으로 고려할 사항으로 free page 가 없는 상황에 lru list 에 page 가 없으면(page_lru_head 에 값이 없음) panic("OOM ERROR₩n")이 발생하게 해주었습니다.

참고로 **pte** 값을 가져오기 위해서는 **walkpgdir** 함수를 사용해야 하는데 이 함수는 **vm.c** 에 static 으로 선언되어 있기 때문에 **defs.h** 로 연결을 하기 보다는 **kalloc.c** 에 똑같이 하나를 만들어 주었습니다.

```
static pte_t * walkpgdir(pde_t *pgdir, const void *va, int alloc)
{
  pde_t *pde;
  pte_t *pgtab;

  pde = &pgdir[PDX(va)];
  if(*pde & PTE_P){
        pgtab = (pte_t*)P2V(PTE_ADDR(*pde));
  } else{
        if(!alloc || (pgtab = (pte_t*)kalloc()) ==0)
          return 0;
        memset(pgtab, 0 ,PGSIZE);
        *pde = V2P(pgtab) | PTE_P | PTE_W | PTE_U;
    }
   return &pgtab[PTX(va)];
}

"kalloc.c" 249L, 5291C written
```

다음은 **swap in** 을 하는 상황인 page_fault 가 발생할 때를 설명하겠습니다. 우선 page_fault 가 발생하면 가장 먼저 **trap.c** 로 가게 됩니다.

page_fault 가 발생하면 **page_handler** 함수로 이동하고 만약 page_handler 의 return 값이 0 이 아니면 실행 process 를 죽이게 만들었습니다.

```
void
trap(struct trapframe *tf)
{
  if(tf->trapno == T_SYSCALL){
    if(myproc()->killed)
      exit();
    myproc()->tf = tf;
    syscall();
    if(myproc()->killed)
      exit();
    return;
  }

  switch(tf->trapno){
  case T_PGFLT:
     if(page_handler(rcr2())!=0){
        myproc()->killed = 1;
        }
     break;
```

```
int page_handler(uint rcr2)
{
   pte_t *pte;
   pte = walkpgdir(myproc()->pgdir, (char*)rcr2, 0);
   int a = *pte & 0xFFFFF000;
   a = a>>12;

   if(pte && !(*pte & PTE_P)){
       char *mem;
       mem = kalloc();
       swapread(mem, a);
       *pte = *pte - (a<<12);
       *pte = *pte | ((uint)V2P((uint)mem) & 0xFFFFF000 );
       *pte = *pte | PTE_P | PTE_A;
       mklist(myproc()->pgdir, (char *)PGROUNDDOWN((uint)rcr2), V2P(mem));
       acquire(&bitmap);
       bitm[a] = 0;
       release(&bitmap);
       return 0;
   }
   return -1;
}
```

Page_handler 에서는 발생한 주소의 **pte** 값을 받은 후 integer a 에 **swap space** 의 **offset** 을 넣어주었습니다.

if 문을 통해 **pte** 값이 존재하고, ***pte** 의 PTE_P 가 set 되어 있지 않으면 새로운 physical page 를 할당해 준 후 **swapread** 함수를 통해 **swap in** 해주었습니다. ***pte** 값은 새로 할당된 physical memory 의 address 로 다시 **PFN field** 를 바꾸어 주고, PTE_P 와 PTE_A flag 를 set 해주었습니다. 이후 lru list 에 추가해주고 bitmap 을 clear 해 준 후 0 을 return 하게 해주었습니다.

추가적으로 고려할 사항인 **user virtual memory** 가 **copy** 되거나 **deallocate** 될 경우 설명 드리겠습니다.

```
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)){
        if((*pte & PTE_U)==PTE_U){
          char *pp;
          if((pp = kalloc())==0)
            goto bad;
          int q = *pte >>12;
          swapread(pp, q);
          int j=0;
          acquire(&bitmap);
          while(!bitm[j]) j++;
          bitm[j] = 1;
          release(&bitmap);
          swapwrite(pp, j);
          kfree(pp);
        }
       else
         panic("copyuvm: page not present");
    }
    else{
      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), PTE_A|flags) < 0) {
        kfree(mem);
        goto bad;
      }
      mklist(d,(char *)i, V2P(mem));
    }
  }
```

우선 **copyuvm** 이 성공하는 경우에는 **inituvm** **allocuvm** 과 마찬가지로 **mappages** 에 PTE_A flag 를 추가해주었고, **lru list** 에 추가하도록 **mklist** 함수를 호출하였습니다.

그런데 만약 parent 의 PTE_P 값이 clear 되어 있다면, parent 가 swap out 되어 있다는 것이고 이 상황에는 child 또한 swap out 해주어야 하므로 새로운 physical memory 를 할당한 후 기존에 parent 에서 swap out 한 block 에서 내용을 불러온 후 그대로 새로 swap out 을 실행하게 해주었습니다.

다음으로 **virtual memory** 가 **deallocate** 될 때에는 두 가지 상황으로 나누었습니다.

```
int
deallocuvm(pde_t *pgdir, uint oldsz, uint newsz)
{
  pte_t *pte;
  uint a, pa;

  if(newsz >= oldsz)
    return oldsz;

  a = PGROUNDUP(newsz);
  for(; a  < oldsz; a += PGSIZE){
    pte = walkpgdir(pgdir, (char*)a, 0);
    if(!pte)
      a = PGADDR(PDX(a) + 1, 0, 0) - PGSIZE;
    else if((*pte & PTE_P) != 0){
      pa = PTE_ADDR(*pte);
      if(pa == 0)
        panic("kfree");
      char *v = P2V(pa);
      if(*pte & PTE_U)
        dllist(pgdir, (char*)a);
      kfree(v);
      *pte = 0;
    }
   else if((*pte & PTE_U) == PTE_U){
        int k = *pte>>12;
        acquire(&bitmap);
        bitm[k] = 0;
        release(&bitmap);
        *pte = 0;
   }
  }
  return newsz;
}
```

1. PTE_P 값이 set 되어있는 경우
2. PTE_P 값이 clear 되어있고 PTE_U 값이 set 되어있는 경우

먼저 PTE_P 값이 set 되어있다는 것은 swap out 되지 않았다는 것임으로 **lru_list** 에서 제거하는 것만 추가해주었습니다.

PTE_P 값이 claer 되어있는데 PTE_U 값이 set 되어있는 경우는 swap out 해준 경우이므로 이때에는 이미 **lru_list** 에서 제거되어 있으므로 bitmap 에서 해당 page 만 clear 해주었습니다.

제가 작성한 test 코드는 다음과 같습니다.

```
#define EXTMEM  0x100000
#define PHYSTOP 0x400000
#define DEVSPACE 0xFE000000
```

```
int *mem;
int *k;
int main(int argc, char**argv){
        printf(1,"ktest₩n");
        int x,y,j;
        mem = malloc(40960);
        for(int i=0; i<60; i++){
                k = malloc(40960);
                for(j=0; j<10240; j++) k[j]=j+1;
                swapstat(&x, &y);
                printf(1, "before : %d %d₩n", x,y);
        }
        if(fork()==0){
                swapstat(&x,&y);
                printf(1, "fork : %d, %d₩n", x,y);
        }
        for(int i=0; i<10240; i++)
                mem[i]=i;
        swapstat(&x, &y);
        printf(1, "after : %d, %d₩n", x,y);

        printf(1, "=== TEST END ===₩n");
        exit();

}
"ktest.c" 31L, 544C written
```

우선 free page 의 양을 줄여 주기위해 **PHYSTOP** 을 최대한 낮추었고, 처음에 **mem** 을 malloc 으로 할당해준 후 for 문으로 malloc 을 반복시켜준 후 fork 로 해당 data 들을 한번 copy 해준 후 마지막에 다시 처음에 malloc 해주었던 **mem** 에 접근하여 **swap in** 이 잘 되는지 확인해 주었습니다.

```
init: starting sh
$ ktest
ktest
before : 0 0
before : 0 0
before : 0 0
before : 0 0
before : 0 0
before : 0 0
before : 0 0
before : 0 0
before : 0 0
before : 0 0
before : 0 0
before : 0 0
before : 0 0
```

```
before : 0 0
before : 0 0
before : 0 0
fork : 0, 4433
after : 2, 4440
=== TEST END ===
after : 8, 4440
=== TEST END ===
$
```

출력결과는 위와 같이 나옵니다.