Mmap

Design

> You should implement enough `mmap` and `munmap` functionality to make the `mmaptest` test program work. If `mmaptest` doesn't use a `mmap` feature, you don't need to implement that feature.

What is mmap?

The `mmap` and `munmap` system calls allow UNIX programs to exert detailed control over their address spaces.

In this lab, my job is to mapping files in fs to memory space and sync these 2 parts.

How to mmap ?

1. maintain all vma:

```
struct vma{
  struct spinlock lock;
  int pid;
  uint64 rootva;
  uint64 start;
  uint64 end;
  //only when changing length the lock will be held
  int length;
  int prot;
  int flags;
  struct file *f;
  struct vma *next;
};
struct vma vmaPool[MAXMMAP];

```

2. allocate a new vma

```
 //alloc a vma
  struct vma *v = vmaAlloc();
  //printf("alloc a vma ok\n");
  if (v==0){
    printf("no vma block remain\n");
    return -1;
  }
```

```
    //printf("lock dddd      \n");
    v->pid = p->pid;
    v->rootva = start;
    v->start = start;
    v->end = PGROUNDUP(start + size);
    acquire(&v->lock);
    v->length = v->end - v->start;
    release(&v->lock);
    v->prot = pteFlag;
    v->flags = flags;
    v->f = f;
    v->next = 0;

    //add new vma to pcb
    struct vma * vp = p->vp;
    if (vp==0){
      p->vp = v;
      //printf("l1 vma %d, l2 vma %d\n",p->vp,p->vp->next);
    }else{
      while (vp->next!=0){
        //printf("go down\n");
        vp = vp->next;
      }
      vp->next = v;
    }
```

3. allocate free space in process's pagetable

```
   p->sz = v->end;
    printf("map from %p to %p\n",v->start,v->end);
```

Handle page fault

If process wanna access to this vma page, it will throw a page fault. my job is to read the data from file, put it to newly-allocated phyiscal page for that va, and mapping them.

```
int mmapHandler(uint64 va, uint64 cause){
  struct proc *p = myproc();
  struct vma* v = p->vp;
  if (va >= MAXVA || va ==0){
    return -1;
  }
  while(v != 0){
    if(va >= v->start && va < v->end){
      goto found ;
    }
    v = v->next;
```

```
  }
  printf("addr not in vma\n");
  return -1;
found:

  if(cause == 13 && (v->prot & PTE_R)==0){
    //read unreadable vma
    return -1;
  }
  if(cause == 15 && (v->prot & PTE_W)==0){
    //write unwriteable vma
    return -1;
  }

  char * mem = kalloc();
  if (mem==0){
    printf("no physical memory for vma mapping\n");
    return -1;
  }
  memset(mem,0,PGSIZE);
  int pteFlags = v->prot ;
  if (cause == 15){
    pteFlags |= PTE_D;
  }
  if( mappages(p->pagetable,PGROUNDDOWN(va),PGSIZE,(uint64)mem,pteFlags) !=0 ) {
    kfree(mem);
    printf("mapping error in mmap trap\n");
    return -1;
  }
  struct file *fl = v->f;
  ilock(fl->ip);
  readi(fl->ip, 0, (uint64)mem, PGROUNDDOWN(va) - v->start, PGSIZE);
  iunlock(fl->ip);
  printf("load va %p 's data in memory from fs\n",va);

  return 0;
}
```

How to unmmap?

> There are different free method:
>
> 1. free the front part of vma address space
> 2. free the back part of vma address space
> 3. free a whole vma address space

1. modify vma:

```
  if(va == v->start){
```

```
    if (vaEnd == v->end){
      //free all mapping
      //free this vma to pool
      //fileclose(v->f);
      v->start = 0;
      v->end = 0;
      if(pv==0){
        //only 1 vma
        p->vp = 0;
      }else{
        pv->next = v->next;
      }
    }else {
      //free a part of the front
      v->start = vaEnd;
    }
  }else{
    //free a part of the end
    v->end = va;
  }
```

2. sync unmmap memory back to file in fs

```
//write back [va,vaEnd] to fs
  pte_t * pte;
  if( (v->prot & PTE_W) && (v->flags & MAP_SHARED)  ){
    for (uint64 addr = va; addr <vaEnd; addr+= PGSIZE){
       pte = walk(p->pagetable,addr,0);
      if (pte !=0 && (*pte & PTE_D)){
        writebackPage(v->f,addr,PGSIZE,addr - v->rootva);
      }
    }
  }
```

3. free the cache memory in process's pagetable

```
  //free unmap pages
   uvmunmap(p->pagetable,va,(vaEnd -va)/PGSIZE,1);
   if(v->start == v->end){
     fileclose(v->f);
   }
   acquire(&v->lock);
   v->length = v->end -v->start;
   release(&v->lock);
   printf("after unmap %p,%p\n",v->start,v->end);
   return 0;
```

Result

pass all tests

```
nel.sym
make[1]: Leaving directory '/home/vielo/cod
== Test running mmaptest ==
$ make qemu-gdb
(5.5s)
== Test   mmaptest: mmap f ==
  mmaptest: mmap f: OK
== Test   mmaptest: mmap private ==
  mmaptest: mmap private: OK
== Test   mmaptest: mmap read-only ==
  mmaptest: mmap read-only: OK
== Test   mmaptest: mmap read/write ==
  mmaptest: mmap read/write: OK
== Test   mmaptest: mmap dirty ==
  mmaptest: mmap dirty: OK
== Test   mmaptest: not-mapped unmap ==
  mmaptest: not-mapped unmap: OK
== Test   mmaptest: two files ==
  mmaptest: two files: OK
== Test   mmaptest: fork_test ==
  mmaptest: fork_test: OK
== Test usertests ==
$ make qemu-gdb
usertests: OK (103.2s)
== Test time ==
time: OK
Score: 140/140
(base)
```