**虚拟内存管理**

# 实验目的

了解虚拟内存的Page Fault异常处理实现

了解页替换算法在操作系统中的实现

# 实验内容

完成 Page Fault异常处理和FIFO页替换算法的实现;

结合磁盘提供的缓存空间，从而能够支持虚 存管理，提供一个比实际物理内存空间“更大”的虚拟内存空间给系统使用;

练习1：给未被映射的地址映射上物理页

ptep=get\_pet(mm->dir,addr,1);

if(ptep == NULL){ //页表项不存在

cprintf("get\_pte in do\_pgfault failed\n");

   goto failed;

}

if (\*ptep == 0) {

 //物理页不在内存之中  //判断是否可以分配新页

        if (pgdir\_alloc\_page(mm->pgdir, addr, perm) == NULL) {             cprintf("pgdir\_alloc\_page in do\_pgfault failed\n");

        goto failed;        }

} else{  if(swap\_init\_ok) {

            struct Page \*page=NULL;

   ret = swap\_in(mm, addr, &page);

   if(ret != 0){ //判断页面可否换入

 cprintf("swap\_in in do\_pgfault failed\n");

 goto failed;             }

   //建立映射

 page\_insert(mm->pgdir, page, addr, perm);

 swap\_map\_swappable(mm, addr, page, 1);

  }

     else {

            cprintf("no swap\_init\_ok but ptep is %x, failed\n",\*ptep);

       goto failed;

   }    }

   ret = 0;

failed:

    return ret; }

# 练习2：补充完成基于FIFO算法

 \_fifo\_map\_swappable(struct mm\_struct \*mm, uintptr\_t addr, struct Page \*page, int swap\_in){

  list\_entry\_t \*head=(list\_entry\_t\*)

mm->sm\_priv;

 list\_entry\_t \*entry=&(page->pra\_page\_link);

    assert(entry != NULL && head!=NULL);

 list\_add(head,entry);

return 0;  }

pra\_page\_link

用来构造按页的第一次访问时间进行排序的一个链表，这个链表的开始表示第一次访问时间最近的页，链表的尾部表示第一次访问时间最远的页。

 sm\_priv指向用来连接记录页访问情况的链表头。

\_fifo\_swap\_out\_victim(struct mm\_struct \*mm, struct Page \*\* ptr\_page, int in\_tick)

{   list\_entry\_t \*head=(list\_entry\_t\*) mm->sm\_priv;

  assert(head != NULL);   assert(in\_tick==0);

//获取最远端的页   list\_entry\_t \*le = head->prev;

 assert(head != le);

  //取下该页

struct Page \*p = le2page(le,pra\_page\_link);

  //释放该页

 list\_del(le);

assert(p!=NULL);   //将该页存入\*ptr\_page中

\*ptr\_page = p;

 return 0; }