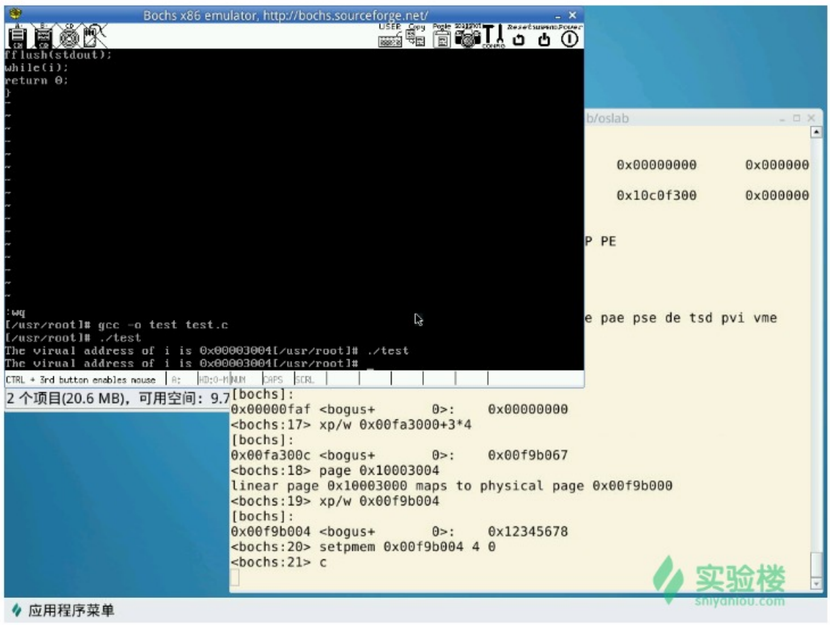
HUST_AT L32 2016-08-30 11:12 Linux 计算机专业课

实验代码

“操作系统原理与实践”实验报告

地址映射与共享



应用程序菜单

那个改内存的实验跟着指导做就可以了，最后结果如图所示，确实退出了。。

```
//shmget()那几个系统调用，看APUE,也没完全看明白，都是进程间通信比较靠后的章节了，等把APUE从前往后看
那里了再补这个实验吧。但是我在《内核注释》中看到0.11版的memory.c
//文件后面其实还写了两个进程间内存共享的函数。我重新注释了一下，虽然打不过还是书上原话。
//try_to_share()在任务p中检查位于地址address处的页面，看页面是否存在，是否干净。如果是存在且干净
略，就与当前任务共享
//这个应该是父子进程间的共享，注释中没说清楚，但我觉得应该是这样的，不然下面不会注释没有修好才能共享。
//参数address是进程中的逻辑地址，也相当于偏移吧。如果进程p address处的页面存在，且没有被修改过的话，
就当前进程与p进程共享之。同时还需要验证当前进程address对应地址处
//是否已经申请了页面，如果是，则出错，因为既然要想从P共享某一页面，那自己肯定是先前没有那一页的，如果有
那肯定是一个错误请求
static int try_to_share unsigned long struct
{
    unsigned long from;
    unsigned long to;
    unsigned long from_page;
    unsigned long to_page;
    unsigned long phys_addr;
    //首先求得p进程中逻辑地址address处对应的页目录。
    from_page=to_page=((address>> 20 )&0xffc ); //此处还不是真正的页目录项地址，因为address是相对
避p的start_code为起始地址的偏移
    from_page+=((p ->start_code>> 20 )&0xffc ); //此时加上start_code的页目录项的起始地址，from_pa
g才是address的页目录项地址
    to_page+=((current ->start_code>> 20 )&0xffc ); //当前进程address处对应的页目录项地址
    //下面首先对p进程的表项进行操作，目的是取得p进程中address对应的物理内存的页面地址，我们希望这一页
避在且干净的因为只有这样才能共享，如果不是，则函数返回。
    //如果存在且干净，那么把address算出的物理页面的对应的页表项中的内容存在phys_addr中。
    from=*( unsigned long *)from_page; //现在fron中存的就是p进程address对应页目录项中的内容了
    if (!(from& 1))
        return 0;//如果p address对应的页目录项中P位是0，则说明这个物理页不存在，也就不用共
享，函数返回
    from&= 0xfffff000 ;//取页表项的前 20位，这指定了p进程address页表的起始物理地址
    from_page=from+((address>> 10 )&0xffc ) //address>>10)&0xffc实际是取address的中间几位数，
算页表的偏移，所以from_page算出的就是页表项的地址
    phys_addr=*( unsigned long *)from_page;
    if ((phys_addr& 0x41 )!= 0x01 )
        return 0;//0x41正好对应页表项的D和P标志，如果页面不干净或无效则返回
    //如果通过以上检查，则取出正在物理页的物理地址送入phys_addr中，最后还有检查下物理页面的有效性，看
避否在范围里面
    phys_addr&= 0xfffff000 ;
    if (phys_addr>=HIGH_MEMORY||phys_addr<LOW_MEM)
        return 0;
    //下面开始真正的共享操作，目标是取得当前进程address对应的页表项地址，并且改页表项还没有映射物理页
即P=0
    to=*( unsigned long *)to_page; //to存储目录项内容
    if (!(to& 1)) //如果页表不存在，则创建页表，且设置好相应的页目录项
        if (to=get_free_page())
            *( unsigned long *)to_page=to| 7;
        else
            oom();
    //否则取页目录项中的页表地址到to
    to&= 0xfffff000 ;
    to_page=to+((address>> 10 )&0xffc ); //页表项地址，为什么是这样，上面有类似代码，有解释
    if ( 1&*( unsigned long *)to_page)
        panic( "try_to_share:to_page already exists" );
    //接下来开始最后的共享操作，就是设置进程P的address对应页的写保护标志（R/W=0，只读），然后让当前
避复制进程P的的这个页表项，即完成工作
    *( unsigned long *)from_page&=~ 2;
    *( unsigned long *)to_page=*( unsigned long *)from_page;
    invalidate();
    phys_addr -=LOW_MEM;
    phys_addr>>= 12 ;
    mem_map[phys_addr]++;
    return 1;
}
```