Operating System Principle, OS

《操作系统原理》

华中科技大学网安学院

2024年10月-2024年12月

实验三：第7章 内存管理、第8章设备管理

- 一、实验目的
  - （1）理解页面淘汰算法原理，编写程序演示页面淘汰算法。
  - （2）验证Linux虚拟地址转化为物理地址的机制
  - （3）掌握Linux驱动程序的编写流程和基本编程技巧
- 二、实验内容
  - （1）Windows/Linux模拟实现OPT,FIFO,LRU等淘汰算法。
  - （2）Linux下利用/proc/pid/pagemap计算变量或函数虚拟地址对应的物理地址等信息。
  - （3）在Linux平台编写一个字符设备的驱动程序和测试用的应用程序。
  - （4）在Linux平台编写一个字符设备的驱动程序和测试用的应用程序。驱动程序的功能：内部维护一个32字节的缓冲区
- 三、实验要求
  - 1,2任选一，3,4任选1。现场检查任意1个。

实验三：第7章 内存管理、第8章设备管理

- 四、实验指南
  - （1）Windows/Linux模拟实现OPT,FIFO,LRU等淘汰算法。
  - [以下模拟过程仅供参考，不是唯一方案！百度参考其他方案！]
    - ◆提示1：程序指令执行过程采用遍历数组的操作来模拟；
    - ◆提示2：用1个较大数组A（例如2400个元素）模拟进程，数组里面放的是随机数，每个元素被访问时就使用printf将其打印出来，模仿指令的执行。数组A的大小必须是设定的页大小（例如10个元素或16个元素等等）的整数倍。
    - ◆提示3：用3-8个小数组(例如数组B，数组C，数组D等)模拟分得的页框。小数组的大小等于页大小（例如10条指令的大小，即10的元素）。小数组里面复制的是大数组里面的相应页面的内容（自己另外构建页表，描述大数组的页与小数组序号的关系）。
    - ◆提示4：利用不同的次序访问数组A，次序可以是：顺序，跳转，分支，循环，或随机，自己构建访问次序。不同的次序也一定程度反映程序局部性。
    - ◆提示5：大数组的访问次序可以用 rand( )函数定义，模拟产生指令访问序列，对应到大数组A的访问次序。然后将指令序列变换成相应的页地址流，并针对不同的页面淘汰算法统计“缺页”情况。缺页即对应的“页面”没有装到小数组中（例如数组B，数组C，数组D等）。
    - ◆提示6：实验中页面大小，页框数，访问次序，淘汰算法都应可调。
    - ◆提示7：至少实现2个淘汰算法。

实验三：第7章 内存管理、第8章设备管理

- 四、实验指南
  - （2）Linux下利用/proc/pid/pagemap技术计算某个变量或函数虚拟地址对应的物理地址等信息。
    - ◆提示1：Linux的/proc/pid/pagemap文件允许用户查看当前进程虚拟页的物理地址等相关信息。
      - □每个虚拟页包含一个64位的值
      - □注意分析64位的信息
    - ◆提示2：获取当前进程的pagemap文件的全名

```
49    pid  = getpid();
50    sprintf(buf, "%d", pid);
51    strcat(pagemap_file, "/proc/");
52    strcat(pagemap_file, buf);
53    strcat(pagemap_file, "/pagemap");
```

    - ◆提示3：可以输出进程中某个或多个全局变量或自定义函数的虚拟地址，所在页号，所在物理页框号，物理地址等信息。
    - ◆思考：（1）如何通过扩充实验展示不同进程的同一虚拟地址对应不同的物理地址。（2）如何通过扩充实验验证不同进程的共享库具有同一的物理地址。

实验三：第7章 内存管理、第8章设备管理

- 四、实验指南
  - （3）在Linux平台编写一个字符设备的驱动程序和测试用的应用程序。功能：驱动程序接收应用程序write输入的整数，并用read输出最近连续两次输入的整数的和或其最大值。
    - ◆提示1：至少实现xx_open,xx_write,xx_read等函数
    - ◆提示2：功能：
      - □xx_write( )写1个整数
      - □xx_read( )读回结果（和或最大值）（注意处理异常，例如，只有一个数时！）
    - ◆提示3: [可选的设备注册方式，其余方式参考baidu]

      ```
      struct miscdevice    mydemodrv_misc_device ;
      ret = misc_register( &mydemodrv_misc_device );
      ```

    - ◆提示4：内核设计3个全局变量：int a=0,b=0,flag=0.每次write时取反flag，当flag为0时，写入的数据赋给a，否则赋给b。每次read时，返回a,b的和或最大值。

实验三：第7章 内存管理、第8章设备管理

```
static ssize_t
demodrv_read(struct file *file, char __user *buf,
             size_t lbuf, loff_t *ppos)
{
    printk("%s enter\n", __func__);
    return 0;
}

static ssize_t
demodrv_write(struct file *file, const char __user *buf,
              size_t count, loff_t *f_pos)
{
    printk("%s enter\n", __func__);
    return 0;

}
```

实验三：第7章 内存管理、第8章设备管理

- 四、实验指南
  - （4）在Linux平台编写一个字符设备的驱动程序和测试用的应用程序。功能：内部维护一个32字节的缓冲区，应用程序可以多次调用read或write函数读写该缓冲区。 read或write操作相互之间无关联，但要求每次read或write要接着上操作的位置继续。如果读写失败返回错误。
    - ◆提示1：参考任务3
    - ◆提示2：至少实现xx_open,xx_write,xx_read等函数
    - ◆提示3：功能：
      - 内核分配一定长度的缓冲区，比如64字节。

```
static char *device_buffer;
#define MAX_DEVICE_BUFFER_SIZE 64
device_buffer = kmalloc(MAX_DEVICE_BUFFER_SIZE, GFP_KERNEL);
```

      - xx_write()写进去若干字符，注意维护写入位置。下次继续写的话，接着该位置往后写，直到缓冲区末尾。要返回实际写入字数。
      - xx_read()读出若干字符串，注意维护读出位置。下次继续读的话，接着该位置往后读，直到缓冲区末尾。要返回实际读回字数。
      - （注意处理异常，例，read或write时剩下的空闲区不够！）