《操作系统原理》实验报告

姓名	汪闻韵	学号	U202012056	专业班级	网安 2002 班	时间	2022.12.28

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) 理解和验证缺页处理的流程。
4) 理解设备是文件的概念。
5) 掌握 Linux 模块、驱动的概念和编程流程

二、实验内容

1) Windows/Linux 模拟实现 FIFO 或 LRU 页面淘汰算法。
2) 编写一个 Linux 内核模块，并完成模块的安装/卸载等操作。
3) 编写 Linux 驱动程序（字符类型或杂项类型）并编程应用程序测试。 功能：write 几个整数进去，read 出其和或差或最大值。

三、实验过程

3.1 模拟 FIFO 或 LRU 页面淘汰

FIFO 实现过程中的核心在于记录每个页表项上一次访问的时间；此处利用数组 waittimes [bsize]记录物理块中的进程等待次数，同时用 maxProc 记录存放最久未被使用的进程，maxWait 用来记录未被使用进程的时间最大值。实现能够记录每一个页表项目的访问时间后，每次需要淘汰时只需要依次遍历页表项并比较页表项之间等待次数的大小，更新此时的 waitTimes。对每一次需要访问的目标，在当前队伍列表中查询是否有空闲物理块，是否命中，并找出此时等待次数最大的进程序列。

```
for (int i = 0; i < psize; i++) {
    avbBlock = samProc = -1;
    maxWait = -1, maxProc = 0;
    for (int j = 0; j < bsize; j++) {
        //查找空闲物理块
        if (phb[j] == 0 && avbBlock == -1)
            avbBlock = j;
        //查找相同进程
        if (phb[j] == pro[i] && samProc == -1)
            samProc = j;
        //找 waitTimes 值最大的
        if (waitTimes[j] > maxProc && maxWait == -1) {
            maxProc = waitTimes[j];
            maxWait = j;
        }
    }
```

更新此轮页表情况后，分为三种情况进行替换。若存在空闲的物理块且该页表未命中，则将当前进程号填入该空闲物理块，并将缺页标志增加；若不存在空闲物理块且未命中，则按照记录的 waitTimes 进行页表项的替换，淘汰最早进来的进程并记录缺页；若物理块中存在相同的进程，则可以不进行淘汰。需要注意的是，不管是哪一种情况，在此轮结束后 waitTimes 数组均需要自加一。

```
    //不存在相同进程、存在空闲物理块
    if (samProc == -1 && avbBlock != -1) {
        phb[avbBlock] = pro[i];   //进程号填入该空闲物理块
        missPage++;//缺页
        waitTimes[avbBlock] = 0;
        for (int j = 0; j <= avbBlock; j++)
            waitTimes[j]++;
    }
    //不存在相同进程、不存在空闲物理块
    else if (samProc == -1 && avbBlock == -1) {
        phb[maxWait] = pro[i];
        waitTimes[maxWait] = 0;
        for (int j = 0; j < bsize; j++)
            waitTimes[j]++;
        missPage++;//缺页
    }
```

```
        //存在相同的进程
        else if (samProc != -1) {
            phb[samProc] = pro[i];
            for (int j = 0; j < bsize; j++)
                waitTimes[j]++;
        }
        //记录结果
        for (int j = 0; j < bsize; j++)
            result[j][i] = phb[j];
```

与 FIFO 相似，模拟实现 LRU 函数中也分为三种情况，存在相同进程时直接记录结果即可，在此不做过多赘述。当页表项中不存在相同进程时，函数利用页表对象中的时间记录成员，取得在内存中停留最久的页面,默认状态下为最早调入的页面，访问后该页面时间记录成员清零并记录缺页。

```
        void _LRU(int fold, Page *b)
        {
            // LRU 核心部分
            int samProc = -1;
            // 判断页面是否已在内存中
            for (int i = 0; i < bsize; i++){
                if (fold == b[i].num)
                    samProc = i;
            }
            // 存在相同进程
            if (samProc >= 0){
                b[samProc].time = 0;
                for (int i = 0; i < bsize; i++)
                    if (i != samProc)
                        b[i].time++;
            }
            // 不存在相同进程
            else{
                // 取得在内存中停留最久的页面,默认状态下为最早调入的页面
                int Longest = -1;
                for (int i = 0; i < bsize; i++){
                    if (b[i].time > Longest){
                        Longest = b[i].time;
                        samProc = i;
                    }
```

```
            }
            ++missPage; // 缺页
            b[samProc].num = fold;
            b[samProc].time = 0;
            for (int i = 0; i < bsize; i++)
                if (i != samProc)
                    b[i].time++;
        }
    }
```

除此之外，对于辅助函数，main 函数实现菜单功能；buildNum()函数用来生成随机数并该序列存入进程数组；Init()用来在每次执行前初始化内存单元、缓冲区；output()函数将 result 数组中存储的结果以更形象的方式输出。

3.2 完成 Linux 简单模块的安装/卸载

编写模块函数如下所示。

```
#include <linux/kernel.h>
#include <linux/module.h>

static int hello_init(void){
    printk("hello_init");
    return 0;
}

static void hello_exit(void){
    printk("hello_exit");
}

MODULE_LICENSE("GPL");
module_init(hello_init);
module_exit(hello_exit);
```

Makefile 文件代码如下图所示。其中 KERNEL_PATH 为内核的绝对路径，PWD 需要安装进去的内核文件所在的路径。

```
KERNEL_PATH := /lib/modules/`uname -r`/build
PWD := $(shell pwd)
MODULE_NAME := test1
obj-m := $(MODULE_NAME).o
all:
    $(MAKE) -C $(KERNEL_PATH) M=$(PWD)
clean:
    rm -rf .*.cmd *.o *.mod.c *.order *.symvers  *.ko *.mod
```

执行编译并安装代码如下。

```
make
sudo insmod test1.ko
sudo dmesg -c
```

卸载该模块代码如下。

```
sudo rmmod test1.ko
sudo dmesg -c
```

3.3 编写 Linux 驱动程序

对于驱动程序的编写核心在于代码中定义的一个 file_operations 类型的结构体 pstruct，该结构体内的成员通过键值对进行赋值，表明了编写时对该驱动自定义的函数与 Linux 给出的标准读写等 API 之间的映射关系。具体指明了 owner，以及标准的读写函数、装卸时调用函数与编写时自定义函数之间的对应关系。

```
static struct file_operations pxa270_fops = {
    .owner = THIS_MODULE,
    .write = dv_write,
    .read = dv_read,
    .open = dv_open,
    .release = dv_release,
};
```

与模块相同，驱动程序也有相应装卸时的调用函数，使用 module_init()和 module_exit()进行注册。其中，在安装驱动调用的函数 dv_init()中，使用了 register_chrdev()函数来注册一个字符型的设备，需要传入参数主设备号，设备名以及定义的驱动函数映射的结构体 pstruct;而在删除驱动调用的函数 dv_exit()中，使用了 unregister_chrdev()对注册的设备进行删除。

```
static int __init dv_init(void){
    int ret;
    ret = register_chrdev(0, DEVICE_NAME, &pxa270_fops);
    if (ret < 0){
        printk(DEVICE_NAME "can't get major number\n");
        return ret;
    }
    rwdMajor = ret;
    printk("dv module major number is %d\n", ret);
    return 0;
}

void __exit dv_exit(void){
    unregister_chrdev(rwdMajor, DEVICE_NAME);
}
```

分别实现驱动程序中的自定义读写函数，其中要注意函数 copy_from_user()和 copy_to_user()的合理调用，同时需要注意的是在 my_write()函数中，需要检查用户的输入字符数量，限制在一定长度内，防止过长。

```
static ssize_t dv_write(struct file *filp, const char __user
*buffer, size_t count, loff_t *ppos)
{
    if (count > MAX_BUF_LEN)
        count = MAX_BUF_LEN;
    copy_from_user(drv_buf, buffer, count);
    WRI_LENGTH = count;
    printk("user write data to driver\n");

    int len = 0;
    int temp = myatoi(drv_buf, &len) + myatoi(drv_buf + len, &len);
    sprintf(drv_buf, "%d", temp);
    return count;
}
static ssize_t dv_read(struct file *filp, char __user *buffer,
size_t count, loff_t *ppos)
{
    if (count > MAX_BUF_LEN)
        count = MAX_BUF_LEN;
    copy_to_user(buffer, drv_buf, count);
    printk("user read data from driver\n");
    return count;
}
```

编译安装该模块。编写 Makefile 文件，与上一节中的 Makefile 文件相同，详见 3.2 节中 Makefile 文件的编写代码，生成一系列的编译文件后，对模块进行安装。

```
make
sudo insmod my_dv.ko
```

查看节点的驱动号，如图 3-1 可知此设备 my_dv 驱动号为 240，进一步创建设备节点和设备挂钩。其中/dev/my_dv 为驱动模块的名称，c 表明为字符型设备，240 为主设备号，0 为次设备号,成功创建设备并挂钩后如图 3-2 所示。

```
#查看驱动号
cat /proc/devices
#创建设备节点和设备挂钩
sudo mknod /dev/my_dv c 240 0
#设备节点的详细信息
ls -l /dev/hello
```

图 3-1 查看驱动号

```
willow@willow-VMware-Virtual-Platform:/dev$ ls -l /dev/my_dv
crw-r--r-- 1 root root 240, 0 12月  30 00:00 /dev/my_dv
willow@willow-VMware-Virtual-Platform:/dev$
```

图 3-2 查看设备

编写测试文件 test2.c 后进行编译运行，对模块进行测试，执行如下命令。

```
gcc test2.c
#最后的参数为设备绝对路径
sudo ./a.out /dev/my_dv
```

```
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <stdio.h>
#define MAX_LEN 32
int main(char argc, char *argv[])
{
    int fd;
    int a, b;
    char write_buf[1024];
    char read_buf[1024];
    /* 将要打开的文件的路径通过 main 函数的参数传入 */
    if (argc != 2){
        printf("Usage: %s <filename>\n", argv[0]);
        return -1;
    }
    fd = open(argv[1], O_RDWR);
    if (fd < 0)
    {
        perror("fail to open file");
        return -1;
    }

    printf("Input First number:  ");
    scanf("%d", &a);
    printf("Input Second Number: ");
    scanf("%d", &b);
    sprintf(write_buf, "%d %d ", a, b);
    write(fd, write_buf, strlen(write_buf) + 1);
    printf("Read From Device: ");
    read(fd, read_buf, MAX_LEN);
    printf("%d\n", atoi(read_buf));
    close(fd);
    return 0;
}
```

四、实验结果

4.1 模拟 FIFO 或 LRU 页面淘汰

完成代码编写后运行，选择“0”则退出，如图 4-1 所示。

图 4-1 退出界面

选择“1”后进入 FIFO 模拟界面，系统随机产生一系列的随机数，来模拟序列号的生成，随后运用函数 output()来打印出易于理解的过程，最后显示出缺页次数以及缺页率。如图 4-2 所示。

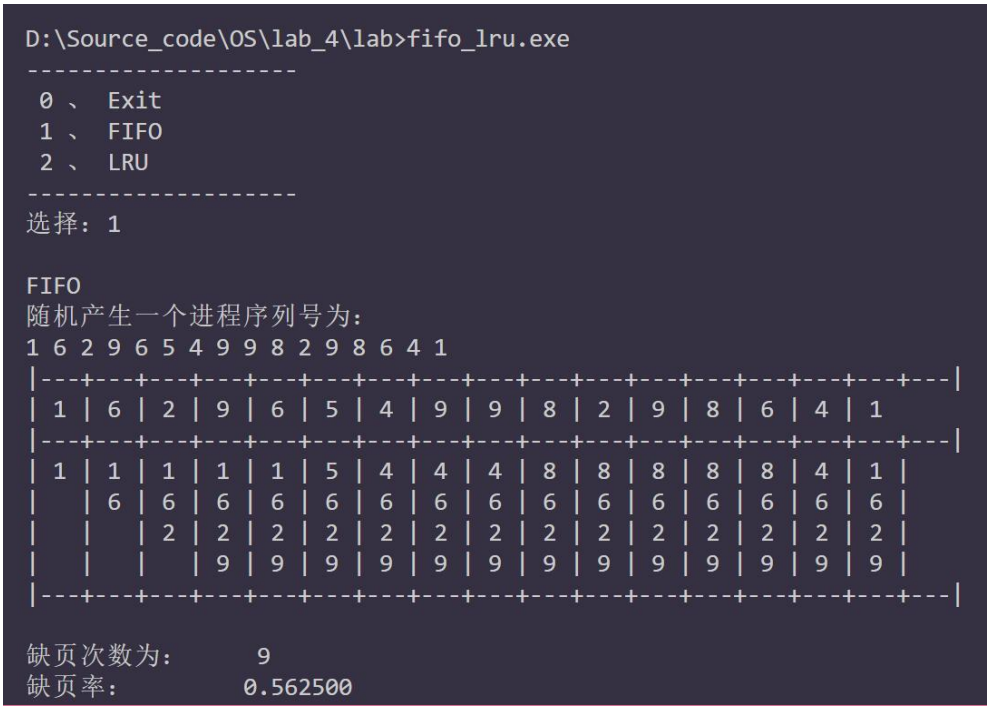

图 4-2 FIFO 运行结果

选择“2”后进入 LRU 模拟界面，与 FIFO 运行结果显示结构类似，如图 4-3 所示。

```
--------------------
 0 、 Exit
 1 、 FIFO
 2 、 LRU
--------------------
选择：2

LRU
随机产生一个进程序列号为：
1 4 2 2 10 5 2 1 1 4 6 6 7 7 2 7
|---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---|
| 1 | 4 | 2 | 2 |10 | 5 | 2 | 1 | 1 | 4 | 6 | 6 | 7 | 7 | 2 | 7
|---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---|
| 1 | 1 | 1 | 1 | 1 | 5 | 5 | 5 | 5 | 5 | 6 | 6 | 6 | 6 | 6 | 6 |
|   | 4 | 4 | 4 | 4 | 4 | 4 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 |
|   |   | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 7 | 7 | 7 | 7 |
|   |   |   |   |10 |10 |10 |10 |10 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
|---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---|

缺页次数为：     10
缺页率：         0.625000
```

图 4-3 LRU 运行结果

4.2 完成 Linux 简单模块的安装/卸载

完成自行编写模块的编写、编译和装卸过程之后，使用 dmesg 命令查看装卸时调用的 printk 函数输出的内容，可以看到与我们编写的提示信息一致，实验达到预期。

```
[27791.334081] hello_exit
[27885.356724] hello_init
willow@willow-VMware-Virtual-Platform:~/桌面/new$
```

图 4-4 LRU 安装/卸载模块

4.3 编写 Linux 驱动程序

测试该模块的安装情况，如图可以显示出模块 my_dv.ko 的详细信息，证明该模块安装成功。

```
willow@willow-VMware-Virtual-Platform:~/桌面/dv$ modinfo my_dv.ko
filename:       /home/willow/桌面/dv/my_dv.ko
author:         WWY
license:        GPL
srcversion:     7773ACC74CA060A93057605
depends:
retpoline:      Y
name:           my_dv
vermagic:       5.10.0-5-generic SMP mod_unload modversions
```

图 4-5 模块详细信息

对测试程序 test2.c 进行编译并运行后，根据提示输入两个需要参与运算的数 3 和 4，可以得到正确的运行结果 7，表明驱动程序成功装载并正常工作，达到实验预期。

```
willow@willow-VMware-Virtual-Platform:~/桌面/dv$ sudo ./a.out /dev/my_dv
[sudo] willow 的密码：
Input First number:  3
Input Second Number: 4
Read From Device: 7
willow@willow-VMware-Virtual-Platform:~/桌面/dv$
```

图 4-6 运行结果

完成自行编写模块的编写、编译过程之后，使用 dmesg 命令查看装卸时调用的 printk 函数输出的内容，可以看到与我们编写的提示信息一致，实验达到预期。

```
willow@willow-VMware-Virtual-Platform:~/桌面/dv$ dmesg
[25608.417656] Device open success£¡
[25608.417657] user write data to driver
[25608.417658] user read data from driver
[25608.417729] device release£¡
[25962.030162] Device open success£¡
[25967.511655] user write data to driver
[25967.511664] user read data from driver
[25967.511679] device release£¡
willow@willow-VMware-Virtual-Platform:~/桌面/dv$
```

图 4-6 dmesg 查看信息

五、实验错误排查和解决方法

5.1 模拟 FIFO 或 LRU 页面淘汰

Q：理论变成写代码实现较困难。

A：缺页中断情况可以分成两种：内存中存在空闲位置，直接将当前调度的页面调入内存中（一般出现在调度的开始）内存中不存在空闲位置，就需要选择内存中的一个页面调出，让当前调度页面进入内存，此过程就是页面置换，淘汰的依据就是上述 visit 值大者。我查阅了网上的一些资料和一些代码之后，写出了相应的代码算法。

5.2 完成 Linux 简单模块的安装/卸载

A：在编写 Linux 内核模块的过程中，由于不清楚一些具体命令应该要怎么写和对相关指令的不熟悉，我一开始并不知道要怎么下手。后来我在网上查找了大量的相关资料后，弄明白了内核模块程序所需要的一些内容，与一些具体指令 module_init 和 module_exit 等的用法后，我成功完成了内核模块程序的编写。

Q：makefile 文件的编写

A：在网上查询到了一些可以使用的 Makefile 文件并仿照进行修改后，成功完成了 make 编译和内核模块的安装与卸载。

5.3 编写 Linux 驱动程序

Q：相关 API 不清楚使用方法。

A:在任务 1 的基础上，需要另外完成驱动程序设备的申请创建，write 相关功能和 read 相关的功能操作。但是对于一些具体功能操作我其实并不清楚，还有设备申请创建和连接的相关命令也完全不知晓。后来花了很多时间在网上查阅了大量的资料后我弄清楚了相应的过程，并搜索到了一些具体参数如 file_operation 字符设备操作集合的一系列操作指令。关于设备注册可以通过静态定义并注册设备号或者动态申请并注册设备号的方法，我在综合考虑之后选择了动态申请的方法。此方法无需自己去确定哪个设备号可用，内核会查询哪个设备号没有被使用，然后分配给当前驱动进行注册。

Q：只能输出第一个输入的数据，后面数据的正确性会出问题

A：增加缓冲区 buf，将输入的两个数据读入后，进行相加并存入缓冲区中，之后 read 即可读出相加后的数据。

Q：无法编译成功一直显示找不到设备

A：没有创建设备并挂钩，只是安装了模块。需要创建设备后再运行测试程序。

六、实验参考资料和网址

（1）教学课件

（2）https://blog.csdn.net/weixin_44518102/article/details/124951165

（3）https://blog.csdn.net/hanp_linux/article/details/90445164

（4）https://blog.csdn.net/hanp_linux/article/details/90441869