**4.2虚拟存储器管理**

1、实验目的

学习Linux虚拟存储实现机制；编写代码，测试虚拟存储系统的缺页错误（缺页中断）发生频率。

2、实验内容

修改存储管理软件以便可以判断特定进程或者整个系统产生的缺页情况，达到一下目标

（1）预置缺页频率参数

（2）报告当前缺页频率

3、实验原理

由于每发生一次缺页都要进入缺页中断服务函数do\_page\_fault一次，所以可以认为执行函数的次数就是系统发生缺页的次数。因此可以定义一个全局的变量pfcount作为计数变量，在执行do\_page\_fault时，该变量加1。系统经历的时间可以利用原有系统的变量jiffies，这是一个系统计时器。在内核加载以后开始计时，以10ms为计时单位

实现施可采用2种方案

1. 通过提供一个系统调用来访问内核变量pfcount和jiffies。但是增加系统变量存在居多的不便，如重新编译内核等，而且容易出错以致系统崩溃。
2. 通过/proc文件系统以模块的方式提供内核变量的访问接口。在/proc文件系统下建立目录pf以及在该目录下的文件Pfcount和Jiffies。

4、实验步骤

**（1）下载Linux内核源代码**

在<https://www.kernel.org/>上下载了linux-4.8.15.tar.xz放在了home文件夹中，并解压

tar -xvf linux-4.8.15.tar.xz

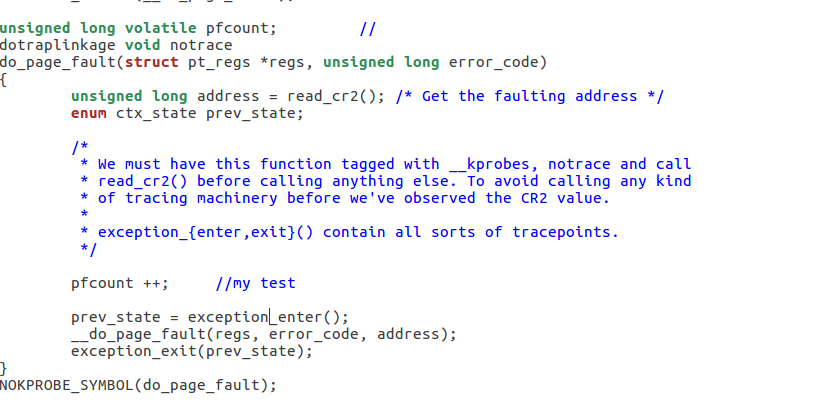
**（2）修改内核源代码**

a、添加统计变量

在/home/linux-4.8.15/arch/x86/mm/fault.c中的do\_page\_fault函数的上一行定义统 计缺页次数的全局变量pfcount，加上unsigned long volatile pfcount;

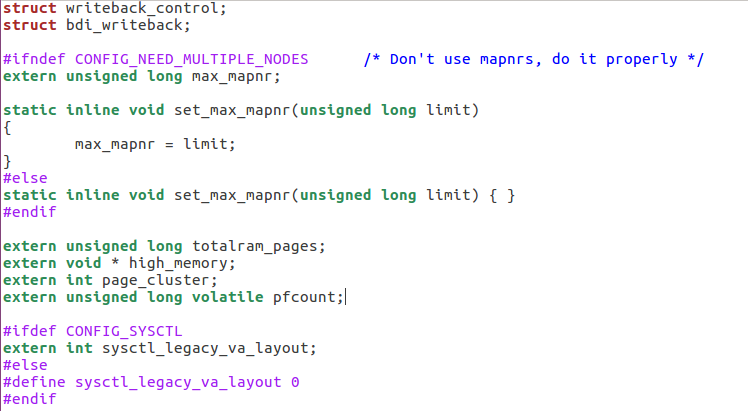
将pfcount加入到do\_page\_fault全局范围，用以统计缺页次数

pfcount++;



b、修改内存管理代码

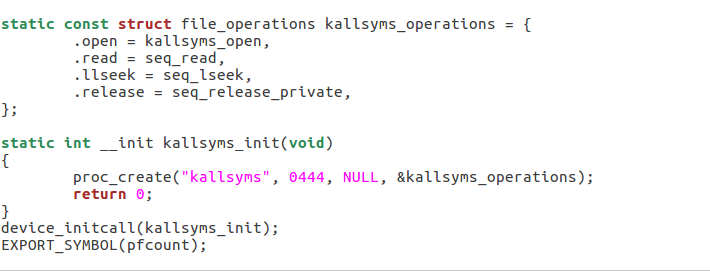
在/home/linux-4.8.15/include/linux/mm.h中加入全局变量pfcount的声明，代码加 在extern int page\_cluster;语句之后，写入extern unsigned long volatile pfcount;



c、导出pfcount全局变量

为让整个内核（包括模块）都可以访问pfcount，在 /home/linux-4.8.15/kernel/kallsyms.c中加入

EXPORT\_SYMBOL(pfcount);



**（3）重新编译内核**

**（4）编写内核模块代码pf.c**

a、pf.c源代码

#include <linux/init.h>

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/mm.h>

#include <linux/proc\_fs.h>

#include <linux/fs.h>

#include <linux/string.h>

#include <asm/uaccess.h>

//#define MODULE

#define USER\_ROOT\_DIR "pf"

#define MODULE1 "Pfcount"

#define MODULE2 "Jiffies"

extern unsigned long volatile pfcount;

extern unsigned long volatile jiffies;

struct proc\_dir\_entry \*proc\_pfcount;

struct proc\_dir\_entry \*proc\_jiffies;

struct proc\_dir\_entry \*my\_dir;

static ssize\_t read\_pfcount(struct file \*file, char \_\_user \*pszPage, size\_t size, loff\_t \*off)

{

int len = 0;

len = sprintf(pszPage, "%d \n", pfcount);

if (len <= \*off)

{

return 0;

}

len -= \*off;

\*off += len;

return len;

}

static ssize\_t read\_jiffies(struct file \*file, char \_\_user \*pszPage, size\_t size, loff\_t \*off)

{

int len = 0;

len = sprintf(pszPage, "%d \n", jiffies);

if (len <= \*off)

{

return 0;

}

len -= \*off;

\*off += len;

return len;

}

static const struct file\_operations proc\_fops1=

{

.owner = THIS\_MODULE,

.read = read\_pfcount,

};

static const struct file\_operations proc\_fops2=

{

.owner = THIS\_MODULE,

.read = read\_jiffies,

};

int pf\_init(void)

{

printk("init\_module()\n");

my\_dir = proc\_mkdir(USER\_ROOT\_DIR, NULL);

proc\_pfcount = proc\_create(MODULE1,0,my\_dir,&proc\_fops1);

proc\_jiffies = proc\_create(MODULE2,0,my\_dir,&proc\_fops2);

return 0;

}

void pf\_exit(void)

{

remove\_proc\_entry(MODULE1,my\_dir);

remove\_proc\_entry(MODULE2,my\_dir);

remove\_proc\_entry(USER\_ROOT\_DIR,NULL);

}

module\_init(pf\_init);

module\_exit(pf\_exit);

MODULE\_LICENSE("GPL");

b、编译内核模块Makefile文件

ifneq ($(KERNELRELEASE),)

obj-m := pf.o

else

PWD := $(shell pwd)

KVER := $(shell uname -r)

KDIR := /lib/modules/$(KVER)/build

all:

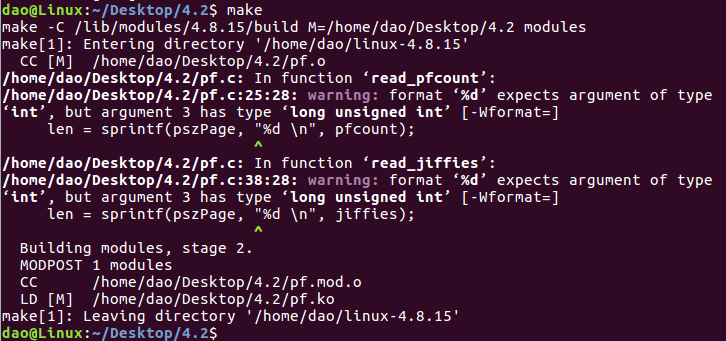
$(MAKE) -C $(KDIR) M=$(PWD) modules

clean:

rm -rf \*.ko \*.o \*.mod.o \*.mod.c \*.symvers

endif

编译完成之后生成pf.ko模块文件。



c、加载内核模块

输入sudo insmod pf.ko模块命令装载内核模块

E:\学习\作业\操作系统\课程实验\4.2\screencut\insmod.png

d、测试

输入命令cat /proc/pf/Pfcount查看pfcount的数量

E:\学习\作业\操作系统\课程实验\4.2\screencut\pfcount.png

输入命令cat /proc/pf/Jiffies查看jiffies的数量

E:\学习\作业\操作系统\课程实验\4.2\screencut\jiffies.png