浙江大学实验报告

课程名称：操作系统 实验类型：综合

实验项目名称：添加系统调用

学生姓名：专业：自动化 学号：

电子邮件地址：实验地点：手机：5

玉泉曹光彪2期503 实验日期：2019 年 11 月 30 日

一、实验目的

1. 学习重建 Linux 内核。

2. 学习 Linux 内核的系统调用，理解、掌握 Linux 系统调用的实现框架、用户界面、参数传递、进入 返回过程。阅读 Li nux 内核源代码，通过添加一个简单的系统调用实验，进一步理解 Linux 操作系统处理系统调用的统一流程。了解 Linux 操作系统缺页处理，进一步掌握 task_struct 结构的作用。

二、实验内容

在现有的系统中添加一个不用传递参数的系统调用。这个系统调用的功能是实现统计操作系统缺页总次数、当前进程的缺页次数及每个进程的“脏”页面数，严格来说这里讲的“缺页次数”实际上是页错误次数，即调用 do_page_fault 函数的次数。实验主要内容：

- 在 linux 操作系统环境下重建内核
- 添加系统调用的名字
- 利用标准 C 库进行包装
- 添加系统调用号
- 在系统调用表中添加相应表项
- 修改统计缺页次数相关的内核结构和函数
- sys_mysyscall 的实现
- 编写用户态测试程序

三、主要仪器设备

1. VMware workstation pro
2. Ubuntu16.04 LTE
3. Intel core i5-7300HQ
4. 虚拟机RAM：2G

四、操作方法与实验步骤

1．查看当前内核版本并下载一份内核源代码

```
liu@ubuntu:/usr/src/linux$ uname -r
5.0.0-23-generic
```

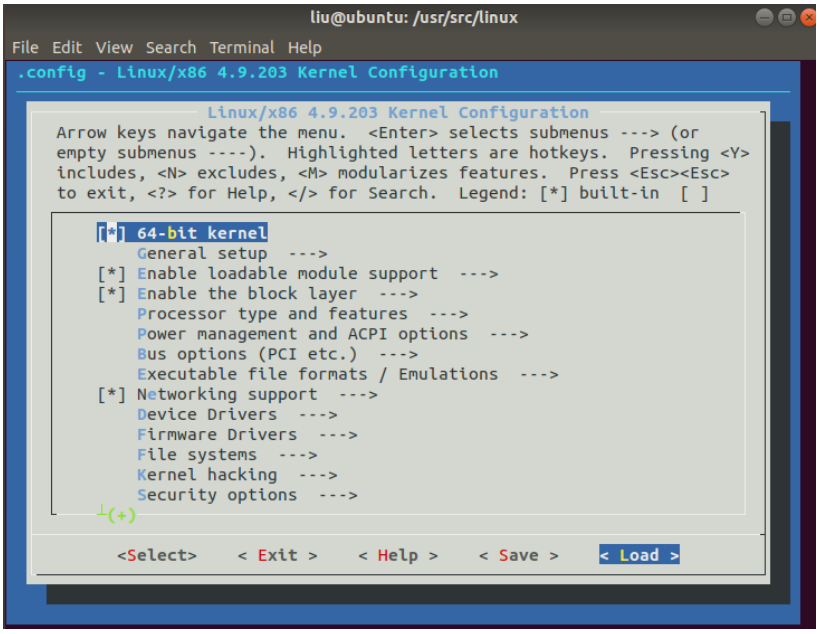
2．部署内核源代码

```
liu@ubuntu:/home$ sudo xz -d linux-5.4.tar.xz
liu@ubuntu:/home$ sudo tar xvf linux-5.4.tar
liu@ubuntu:~$ sudo cp linux-5.4 /usr/src -rf
liu@ubuntu:/usr/src$ ln -s /usr/src/linux-5.4/ linux
liu@ubuntu:/usr/src$ sudo apt-get install libncurses5-dev
```

3．配置内核

```
liu@ubuntu:/usr/src/linux$ sudo make mrproper
[sudo] password for liu:
  CLEAN   .
  CLEAN   arch/x86/entry/vdso
  CLEAN   arch/x86/kernel/cpu
  CLEAN   arch/x86/kernel
  CLEAN   arch/x86/purgatory
  CLEAN   arch/x86/realmode/rm
  CLEAN   arch/x86/lib
  CLEAN   certs
  CLEAN   crypto/asymmetric_keys
  CLEAN   crypto
  CLEAN   drivers/firmware/efi/libstub
```

```
liu@ubuntu:/usr/src/linux$ sudo cp /boot/config-5.0.0-23-ge
neric .config
```



依次执行 Load->OK->save->ok->exit->exit;

4 添加系统调用号

```
liu@ubuntu:/usr/src/linux$ sudo vim /usr/include/asm-generic/unistd.h
```

```
#define __NR_mysyscall 335
__SYSCALL(__NR_mysyscall, sys_mysyscall)
```

```
liu@ubuntu:/usr/src/linux$ sudo vim /usr/src/linux/include/uapi/asm-generic/uni
std.h
```

```
#define __NR_mysyscall 335
__SYSCALL(__NR_mysyscall, sys_mysyscall)
```

5. 在系统调用表中添加或修改相应表项

```
liu@ubuntu:/usr/src/linux$ sudo vim arch/x86/entry/syscalls/syscall_64.tbl
```

```
335     64      mysyscall               sys_mysyscall
```

4和5这里操作与实验指导书不同，因为所用64位系统，需要在syscall_64.tbl中修改，相应的系统调用号我使用的是335。

6. 修改统计系统缺页次数和进程缺页次数的内核代码

```
iu@ubuntu:/usr/src/linux/include/linux$ sudo vim mm.h
```

```
extern unsigned long pfcount;
extern atomic_long_t _totalram_pages;
static inline unsigned long totalram_pages(v
{
```

```
liu@ubuntu:/usr/src/linux/include/linux$ sudo vim sched.h
```

```
struct task_struct {
#ifdef CONFIG_THREAD_INFO_IN_TASK
        /*
         * For reasons of header soup (see current_thread_info()), this
         * must be the first element of task_struct.
         */
        struct thread_info              thread_info;
#endif

        unsigned long pf;
        /* -1 unrunnable, 0 runnable, >0 stopped: */
        volatile long                   state;

        /*
         * This begins the randomizable portion of task_struct. Only
         * scheduling-critical items should be added above here.
         */
        randomized_struct_fields_start

        void                            *stack;
        refcount_t                      usage;
        /* Per task flags (PF_*), defined further below: */
        unsigned int                    flags;
        unsigned int                    ptrace;
-- INSERT --                                                633,19-26      31%
```

```
liu@ubuntu:/usr/src/linux/kernel$ sudo vim fork.c
```

```
 err = arch_dup_task_struct(tsk, orig);

 tsk->pf = 0;/*Initial page fault number to be 0*/
 /*
```

```
liu@ubuntu:/usr/src/linux/arch/x86/mm$ sudo vim fault.c
```

```
unsigned long pfcount;
```

```
static noinline void
__do_page_fault(struct pt_regs *regs, unsigned long error_code,
                unsigned long address)
{
        struct vm_area_struct *vma;
        struct task_struct *tsk;
        struct mm_struct *mm;
        int fault, major = 0;
        unsigned int flags = FAULT_FLAG_ALLOW_RETRY | FAULT_FLAG_KILLABLE;
        u32 pkey;

        tsk = current;
        mm = tsk->mm;

        pfcount++;
        current->pf++;
        /*
         * Detect and handle instructions that would cause a page fault for
         * both a tracked kernel page and a userspace page.
         */
        if (kmemcheck_active(regs))
                kmemcheck_hide(regs);
        prefetchw(&mm->mmap_sem);

        if (unlikely(kmmio_fault(regs, address)))
                                                                1213,1        81%
```

7. sys_mysyscall的实现

```
liu@ubuntu:/usr/src/linux/kernel$ sudo vim sys.c
```

```
extern unsigned long pfcount;
asmlinkage int sys_mysyscall(void)
{
        struct task_struct* p =NULL;
        printk("---------START 335 ----------\n");
        printk("System page fault: %lu \n",pfcount);
        printk("Current process page fault:%lu \n",current->pf);
        printk("-----------------------------------\n");
        printk("Dirty pages of every process:\n");

        for(p = &init_task; (p=next_task(p))!=&init_task;)
        {
                printk("PID:---------%d---------\n",p->pid);
                printk("NAME: %s \n",p->comm);
                printk("DIRTY PAGE: %d \n",p->nr_dirtied);
        }
        printk("----------DONE----------\n");
        return 0;
}
```

nr_dirtied 即为该进程的“脏”页面数

8. 编译内核和重启内核

```
liu@ubuntu:/usr/src/linux$ sudo make bzImage -j2
Setup is 17020 bytes (padded to 17408 bytes).
System is 7277 kB
CRC 6c8479dd
Kernel: arch/x86/boot/bzImage is ready  (#1)
liu@ubuntu:/usr/src/linux$ sudo make modules -j2
liu@ubuntu:/usr/src/linux$ sudo make modules_install
liu@ubuntu:/usr/src/linux$ sudo make install
liu@ubuntu:/usr/src/linux$ sudo update-grub2
Sourcing file `/etc/default/grub'
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-5.4.0
Found initrd image: /boot/initrd.img-5.4.0
Found linux image: /boot/vmlinuz-5.0.0-23-generic
Found initrd image: /boot/initrd.img-5.0.0-23-generic
Found memtest86+ image: /boot/memtest86+.elf
Found memtest86+ image: /boot/memtest86+.bin
done
```

reboot 重启

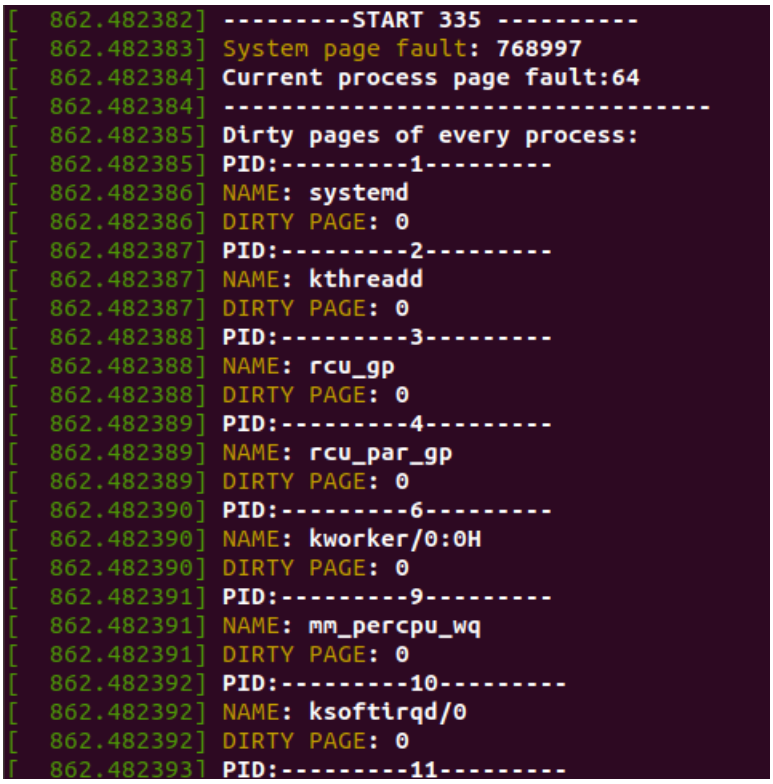
```
liu@ubuntu:~$ uname -r
5.4.0
```

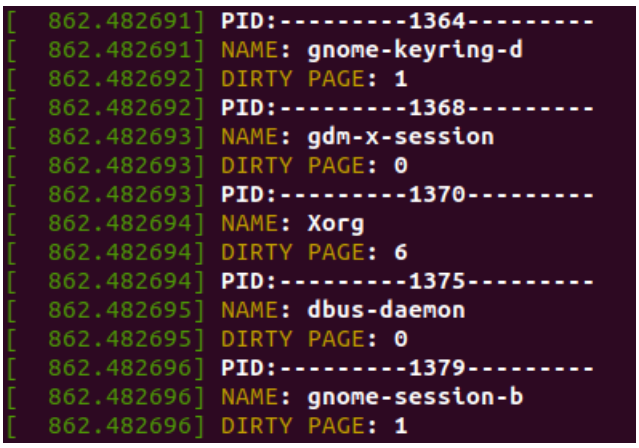
重启后看到内核版本已经更新成功。

9. 编写用户态测试程序

```
liu@ubuntu:/home$ sudo vim test.c
liu@ubuntu:/home$ sudo gcc -o test test.c
liu@ubuntu:/home$ ./test
```

编译成功，下面检测运行效果





可以看到这里已经打印出了操作系统缺页总次数，当前进程的缺页次数，以及每个进程的“脏”页面数。

五、讨论和心得

心得体会:

本次实验主要遇到了三个问题：第一个是 unistd.h 文件有很多份，还有 unistd_32.h 等文件，不确定应该修改哪一个。后来阅读里面内容并在网上查找相关调用关系之后，确定到了两个需要修改的文件，一个是内核自己的，一个是所用 Linux 内核版本的。

第二个问题是困扰时间比较长的问题，在按照试验指导书做完后，运行程序编译正确但并没有结果输出。往回一步一步检查，在网上查了很久之后终于发现，实验指导书上是

所用32位系统并在在 syscall_32.tbl 中修改相应表项。而在64位系统中需要修改在 syscall_64.tbl。修改之后运行成功。

第三个问题是：需要让虚拟机有足够大的存储空间（至少40G），否则无法完成编译，会出现 I/O error 的情况。

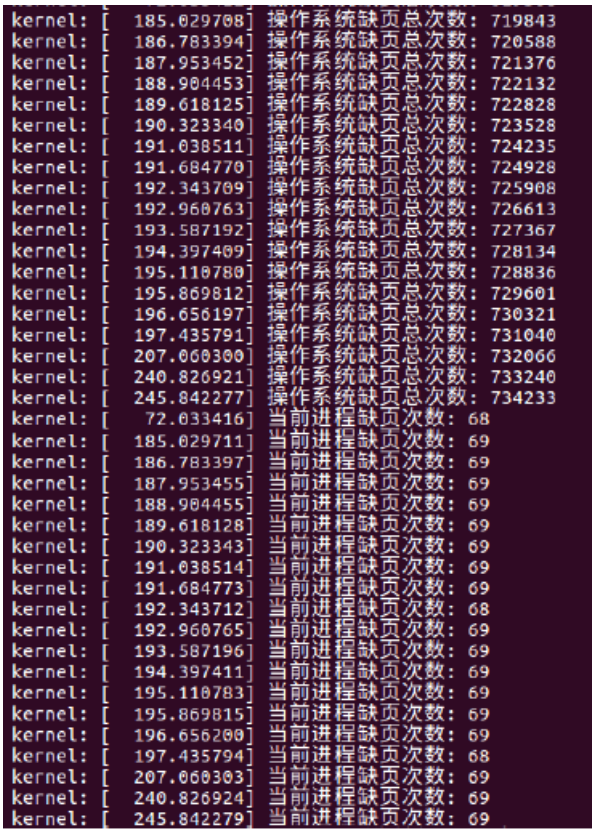
此外还有一些遇到的小问题，比如添加系统函数之后是否需要添加 asmlinkage 相应的头文件，是否能够使用更低的内核版本，都到网上找到了相关的信息。

总的来说，这次实验感觉是对上一次实验以及研讨中相关内容的综合，再一次熟悉了内核重建的同时也新学会了系统调用的相关知识。

思考题:

1. 多次运行 test 程序，每次运行 test 后记录下系统缺页次数和当前进程缺页次数，给出这些数据。test 程序打印的缺页次数是否就是操作系统原理上的缺页次数？有什么区别？

   答：运行20次程序结果如下图：可以看到操作系统缺页总次数在增加，而当前进程缺页次数几乎不变。



   我们得到的是页错误次数，与操作系统原理上的缺页次数不完全相同。Page fault 其实包括三种：major page fault、minor page faullt、invalid(segment fault)

major page fault：指需要访问的内存不在虚拟地址空间，也不在物理内存中，需要从慢速设备载入。

minor page fault：指需要访问的内存不在虚拟地址空间，但是在物理内存中，只需要 MMU 建立物理内存和虚拟地址空间的映射关系即可

invalid(segment fault)：也称为 segment fault, 指进程需要访问的内存地址不在它的虚拟地址空间范围内，属于越界访问，内核会报 segment fault 错误。

2. 除了通过修改内核来添加一个系统调用外，还有其他的添加或修改一个系统调用的方法吗？如果有，请论述。

   答：还可以通过模块进行内核添加。在 init 函数中先用一个函数指针保存系统调用表中的调用位置的系统调用，然后使内核空间可写，用自己的系统调用替换该位置。在 exit 函数中再将系统调用恢复即可。写好 Makefile 问价你之后 insmod 插入，再写一个用户态下的文件调用 syscall（）即可使用。

3. 对于一个操作系统而言，你认为修改系统调用的方法安全吗？请发表你的观点。

   答：个人认为不够安全，一方面直接在内核上修改容易造成混乱，不同版本各个文件路径也会有差别，容易产生误改动或者多改了别的地方的情况。另一方面，作为一个自己添加的系统调用，我们这样修改是相当于修改了源代码而造成永久性的改动，这种静态的修改相比于动态的方式也不够安全。

六、附录

1. mysyscall（）

```
1.  extern unsigned long pfcount;
2.  asmlinkage int sys_mysyscall(void)
3.  {
4.          struct task_struct* p =NULL;
5.          printk("---------START 335 ----------\n");
6.          printk("System page fault: %lu \n",pfcount);
7.          printk("Current process page fault:%lu \n",current->pf);
8.          printk("-----------------------------------\n");
9.          printk("Dirty pages of every process:\n");
10.
11.         for(p = &init_task; (p=next_task(p))!=&init_task;)
12.         {
```

```
                printk("PID:---------%d---------\n",p->pid);
                printk("NAME: %s \n",p->comm);
                printk("DIRTY PAGE: %d \n",p->nr_dirtied);
        }
        printk("----------DONE----------\n");
        return 0;
}
```

2. test.c

```
#include<unistd.h>
#include<stdio.h>
#include<sys/syscall.h>
#include<stdlib.h>
#define __NR_mysyscall 335

int main()
{
        syscall(__NR_mysyscall);
        system("dmesg");
        return 0;
}
```