浙江大学

本科实验报告

课程名称：操作系统

姓　　名：臧可

学　　院：计算机科学与技术学院

系：计算机科学与技术系

专　　业：计算机科学与技术

学　　号：3180102095

指导教师：夏莹杰

2020 年 12 月 5 日

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：臧可 专业：计算机科学与技术 学号：3180102095

电子邮件地址：KeZang@zju.edu.cn 手机：17396876307

实验地点：玉泉曹光彪西503 实验日期：2020年12月5日

一、实验目的和要求

学习重建Linux内核。

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Linux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，以及每个进程的“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、“脏”页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

host machine:

windows 10

处理器：Intel(R) Core(TM) i5-8250U CPU @
1.60GHz

RAM：8.00GB

系统类型：64位操作系统，基于x64的处理器

guest machine:

虚拟机：VMware

Ubuntu 16.04.7(64位)

RAM：4GB

四、操作方法和实验步骤

1.下载一份内核源代码

Linux受GNU通用公共许可证（GPL）保护，其内核源代码是完全开放的。现在很多Linux的网站都提供内核代码的下载。推荐你使用Linux的官方网站：http://www.kernel.org。在这里你可以找到所有的内核版本。

2.部署内核源代码

此过程比较机械、枯燥，因而容易出错。请严格按下述步骤来操作。

首先，把linux-4.6.tar.xz包放在主目录下，解开linux-4.6.tar.xz包：

tar -xvf linux-4.6.tar.xz

解压出来的内核代码存放在/usr/src/linux-4.6目录下。为了方便操作及一致性，可以通过路径/usr/src/linux去访问它，这只要建一个符号链接：

```
ln  -s  /usr/src/linux-4.6/  /usr/src/linux
```

3.配置内核

第1次编译内核的准备：

在ubuntu环境下，用命令make menuconfig对内核进行配置时，需要用终端模式下的字符菜单支持软件包libncurses5-dev，因此你是第一次重建内核，需要下载并安装该软件包，下载并安装命令如下：

```
sudo apt-get install libncurses5-dev
```

若上面这一步提示错误信息，则输入下面的命令sudo apt-get －f install ，建立库依赖关系

1~3按照实验2的内核重建步骤操作.

4.添加系统调用号

系统调用号在文件unistd.h里面定义。这个文件可能在你的Linux系统上会有两个版本：一个是C库文件版本，出现的地方是在 ubuntu 16.04 自带的/usr/include/asm-generic/unistd.h；另外还有一个版本是内核自己的unistd.h，出现的地方是在你解压出来的内核代码的对应位置（比如include/uapi/asm-generic/unistd.h）。当然，也有可能这个C库文件只是一个对应到内核文件的链接。现在，你要做的就是在文件unistd.h中添加我们的系统调用号：__NR_mysyscall，x86体系架构的64位系统调用号335没有使用，我们新的系统调用号定义为335号，如下所示：

ubuntu 16.04为：/usr/include/asm-generic/unistd.h

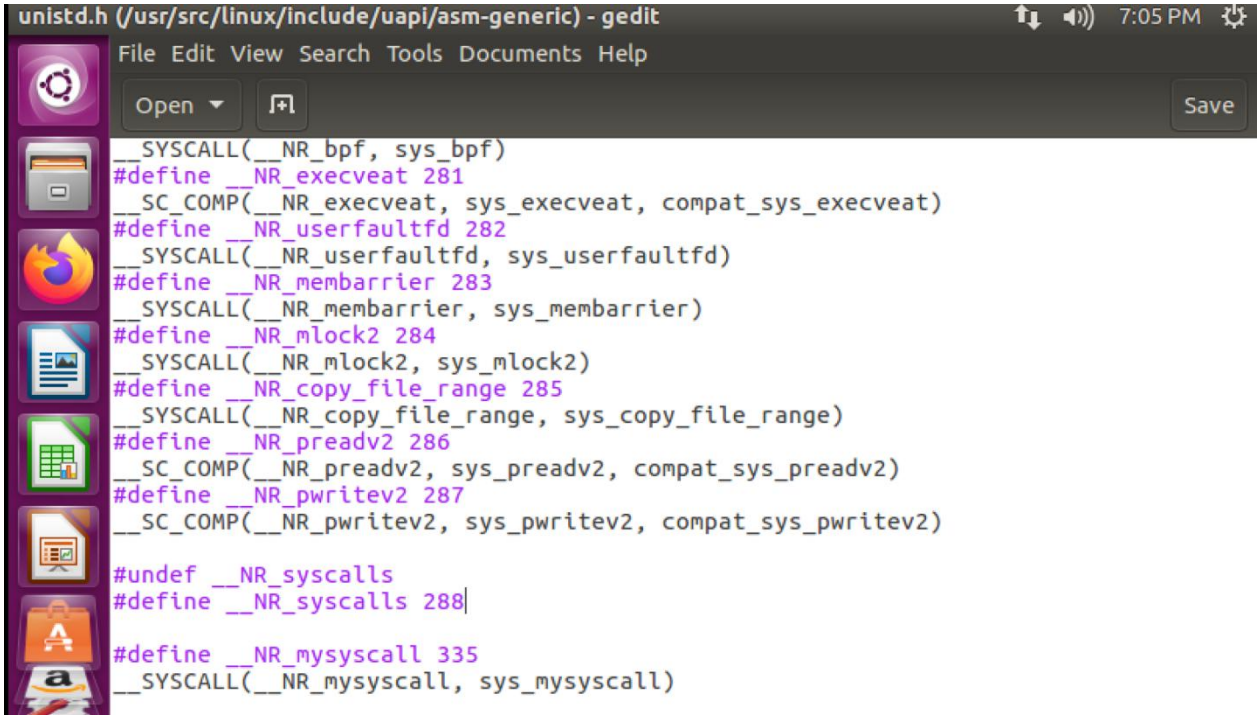
kernel 4.6为：include/uapi/asm-generic/unistd.h

在/usr/include/asm-generic/unistd.h文件中的插入定义335号的行，作如下修改：

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

```
#define __NR_mysyscall 335
__SYSCALL(__NR_mysyscall, sys_mysyscall)
/*
 * All syscalls below here should go away really,
```

在文件include/uapi/asm-generic/unistd.h中做同样的修改



添加系统调用号之后，系统才能根据这个号，作为索引，去找syscall_table中的相应表项。所以说，我们接下来的一步就是：

5.在系统调用表中添加或修改相应表项

我们前面讲过，系统调用处理程序（system_call）会根据eax中的索引到系统调用表（sys_call_table）中去寻找相应的表项。所以，我们必须在那里添加我们自己的一个值。

arch/x86/entry/syscalls/syscall_64.tbl

```
  ++335   common      mysyscall           sys_mysyscall
328     64      pwritev2                sys_pwritev2

335     common  mysyscall               sys_mysyscall
#
# x32-specific system call numbers start at 512 to avoid cache impact
# for native 64-bit operation.
#
512     x32     rt_sigaction            compat_sys_rt_sigaction
```

到现在为止，系统已经能够正确地找到并且调用sys_mysyscall。剩下的就只有一件事情，那就是sys_mysyscall的实现。

6.修改统计系统缺页次数和进程缺页次数的内核代码

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，所以可以认为执行该函数的次数就是系统发生缺页的次数。可以定义一个全局变量pfcount作为计数变量，在执行do_page_fault时，该变量值加1。在当前进程控制块中定义一个变量pf记录当前进程缺页次数，在执行do_page_fault时，这个变量值加1。

先在include/linux/mm.h文件中声明变量pfcount：

++ extern unsigned long pfcount;

```
struct mempolicy;
struct anon_vma;
struct anon_vma_chain;
struct file_ra_state;
struct user_struct;
struct writeback_control;
struct bdi_writeback;

extern unsigned long pfcount;

#ifndef CONFIG_NEED_MULTIPLE_NODES      /* Don't use mapnrs, do it properly */
extern unsigned long max_mapnr;
```

要记录进程产生的缺页次数，首先在进程task_struct中增加成员pf，在include/linux/sched.h文件中（第1394行jjj）的task_struct结构中添加pf字段：

++ unsigned long pf;

```
struct task_struct {
        unsigned long pf;

        volatile long state;    /* -1 unrunnable, 0 runnable, >0 stopped */
        void *stack;
        atomic_t usage;
        unsigned int flags;     /* per process flags, defined below */
        unsigned int ptrace;
```

统计当前进程缺页次数需要在创建进程是需要将进程控制块中的pf设置为0，在进程创建过程中，子进程会把父进程的进程控制块复制一份，实现该复制过程的函数是kernel/fork.c文件中的dup_task_struct()函数，修改该函数将子进程的pf设置成0：

```
static struct task_struct *dup_task_struct(struct task_struct *orig)
{
        ...
        tsk = alloc_task_struct_node(node);
        if (!tsk)
                return NULL;
        ......
```

```
++ tsk->pf=0;
```

```
……
```

```
}
```

```
#endif
        tsk->splice_pipe = NULL;
        tsk->task_frag.page = NULL;
        tsk->wake_q.next = NULL;

        account_kernel_stack(ti, 1);

        kcov_task_init(tsk);

        tsk->pf=0;

        return tsk;

free_ti:
        free_thread_info(ti);
free_tsk:
        free_task_struct(tsk);
        return NULL;
}
```

在arch/x86/mm/fault.c文件中定义变量pfcount；并修改arch/x86/mm/fault.c中do_page_fault()函数。每次产生缺页中断，do_page_fault()函数会被调用，pfcount变量值递增1,记录系统产生缺页次数，current->pf值递增1，记录当前进程产生缺页次数：

```
…
```

```
++ unsigned long pfcount;
```

```
#define CREATE_TRACE_POINTS
#include <asm/trace/exceptions.h>

unsigned long pfcount;

/*
 * Page fault error code bits:
```

```
__do_page_fault(struct pt_regs *regs, unsigned long error_code)
```

```
{
    ...
 ++ pfcount++;
 ++ current->pf++;
    ...
}
```

```
__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;

        tsk = current;
        mm = tsk->mm;

        pfcount++;
        current->pf++;
```

7.sys_mysyscall的实现

我们把这一小段程序添加在kernel/sys.c里面。在这里，我们没有在kernel目录下另外添加自己的一个文件，这样做的目的是为了简单，而且不用修改Makefile，省去不必要的麻烦。

mysyscall系统调用实现输出系统缺页次数、当前进程缺页次数，及每个进程的“脏”页面数。

```
asmlinkage int sys_mysyscall(void)
{
  printk(KERN_INFO"System Page Fault Number:%lu\n", pfcount);
  printk(KERN_INFO"Current Process Page Fault Number:%lu\n", current->pf);//printk(“当前进程缺页次数：%lu,current->pf”)
  printk(KERN_INFO"Each Process:\n");
  struct task_struct *p;
  for_each_process(p){

  printk(KERN_INFO"%-20s %-6d %lu\n", p->comm, p->pid, p->nr_dirtied);//每个进程的“脏”页面数
  }

  return 0;
}
```

```
asmlinkage int sys_mysyscall(void)
{
        printk(KERN_INFO"System Page Fault Number:%lu\n", pfcount);
        printk(KERN_INFO"Current Process Page Fault Number:%lu\n", current->pf);//printk(“当前进程缺页次数：%lu,current->pf”)
        printk(KERN_INFO"Each Process:\n");
        struct task_struct *p;
        for_each_process(p){
                printk(KERN_INFO"%-20s %-6d %lu\n", p->comm, p->pid, p->nr_dirtied);//每个进程的“脏”页面数
        }

        return 0;
}
```

8.编译内核和重启内核

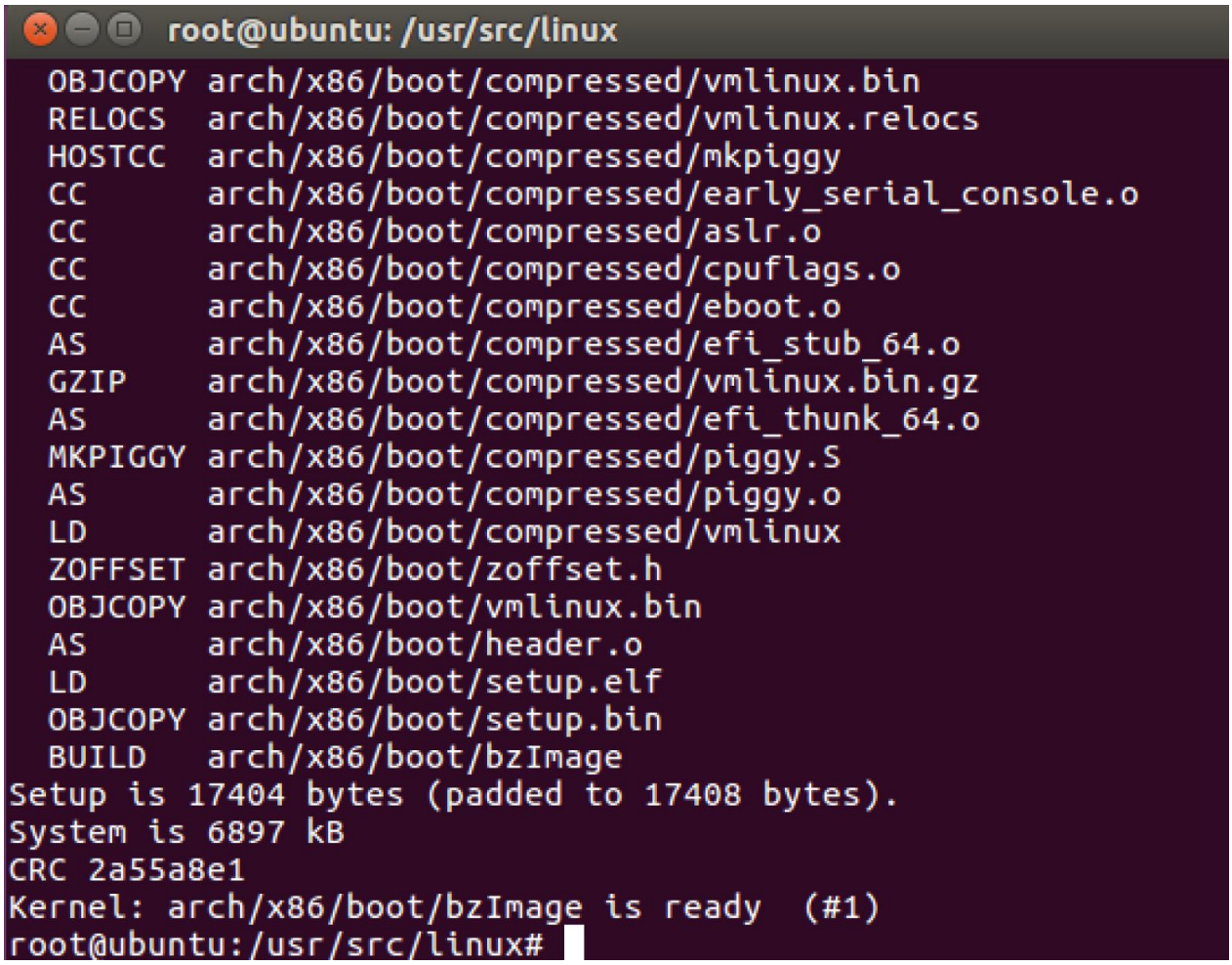
用make工具编译内核：

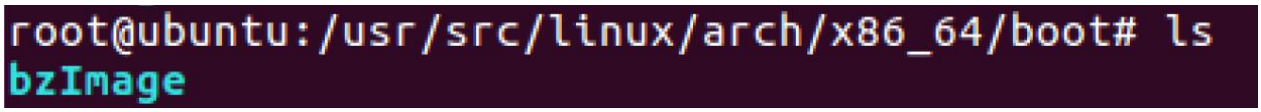
sudo make bzImage -j8

编译内核需要较长的时间，具体与机器的硬件条件及内核的配置等因素有关。完成后产生的内核文件bzImage的位置在/usr/src/linux/arch/i386/boot目录下，当然这里假设用户的CPU是Intel x86型的，并且你将内核源代码放在/usr/src/linux目录下。

如果编译过程中产生错误，你需要检查第4、5、6、7步修改的代码是否正确，修改

后要再次使用make命令编译，直至编译成功。





如果选择了可加载模块，编译完内核后，要对选择的模块进行编译，然后安装。用下面的命令编译模块并安装到标准的模块目录中：

```
sudo make modules -j8 #我的主机是4核的
sudo make modules_install
```

通常，Linux在系统引导后从/boot目录下读取内核映像到内存中。因此我们如果想要使用自己编译的内核，就必须先将启动文件安装到/boot目录下。安装内核命令：

```
sudo make install
```

```
root@ubuntu:/usr/src/linux# sudo make install
sh ./arch/x86/boot/install.sh 4.6.0 arch/x86/boot/bzImage \
        System.map "/boot"
run-parts: executing /etc/kernel/postinst.d/apt-auto-removal 4.6.0 /boot/vmlinuz
-4.6.0
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 4.6.0 /boot/vmlinuz-
4.6.0
update-initramfs: Generating /boot/initrd.img-4.6.0
run-parts: executing /etc/kernel/postinst.d/pm-utils 4.6.0 /boot/vmlinuz-4.6.0
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 4.6.0 /boot/vmli
nuz-4.6.0
run-parts: executing /etc/kernel/postinst.d/update-notifier 4.6.0 /boot/vmlinuz-
4.6.0
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 4.6.0 /boot/vmlinuz-4
.6.0
Generating grub configuration file ...
Warning: Setting GRUB_TIMEOUT to a non-zero value when GRUB_HIDDEN_TIMEOUT is se
t is no longer supported.
Found linux image: /boot/vmlinuz-4.15.0-112-generic
Found initrd image: /boot/initrd.img-4.15.0-112-generic
Found linux image: /boot/vmlinuz-4.6.0
Found initrd image: /boot/initrd.img-4.6.0
Found linux image: /boot/vmlinuz-4.6.0.old
Found initrd image: /boot/initrd.img-4.6.0
Found memtest86+ image: /boot/memtest86+.elf
Found memtest86+ image: /boot/memtest86+.bin
done
root@ubuntu:/usr/src/linux#
```

grub是管理ubuntu系统启动的一个程序。想运行刚刚编译好的内核，就要修改对应的grub。

sudo mkinitramfs 4.6.0 -o /boot/initrd.img-4.6.0

sudo update-grub2

```
root@ubuntu:/usr/src/linux# sudo mkinitramfs  4.6.0  -o  /boot/initrd.img-4.6.0
root@ubuntu:/usr/src/linux# sudo update-grub2
Generating grub configuration file ...
Warning: Setting GRUB_TIMEOUT to a non-zero value when GRUB_HIDDEN_TIMEOUT is se
t is no longer supported.
Found linux image: /boot/vmlinuz-4.15.0-112-generic
Found initrd image: /boot/initrd.img-4.15.0-112-generic
Found linux image: /boot/vmlinuz-4.6.0
Found initrd image: /boot/initrd.img-4.6.0
Found linux image: /boot/vmlinuz-4.6.0.old
Found initrd image: /boot/initrd.img-4.6.0
Found memtest86+ image: /boot/memtest86+.elf
Found memtest86+ image: /boot/memtest86+.bin
done
root@ubuntu:/usr/src/linux#
```

我们已经编译了内核bzImage，放到了指定位置/boot。现在，请你重启主机系统，期待编译过的Linux操作系统内核正常运行！

sudo reboot

9.编写用户态程序

要测试新添加的系统调用，需要编写一个用户态测试程序（test.c）调用mysyscall系统调用。mysyscall系统调用中printk函数输出的信息在/var/log/messages文件中

(ubuntu 为/var/log/kern.log文件)。/var/log/messages (ubuntu 为/var/log/kern.log文件)文件中的内容也可以在shell下用dmesg命令查看到。

用户态程序

```
#include <linux/unistd.h>
#include <sys/syscall.h>
#include <stdio.h>
#define __NR_mysyscall 335
int main()
{
     syscall(__NR_mysyscall);
     system("dmesg 1>log");
}
```

```
#include <linux/unistd.h>
#include <sys/syscall.h>
#include <stdio.h>
#define __NR_mysyscall 335
int main()
{
        syscall(__NR_mysyscall);
        system("dmesg 1>log");
}
```

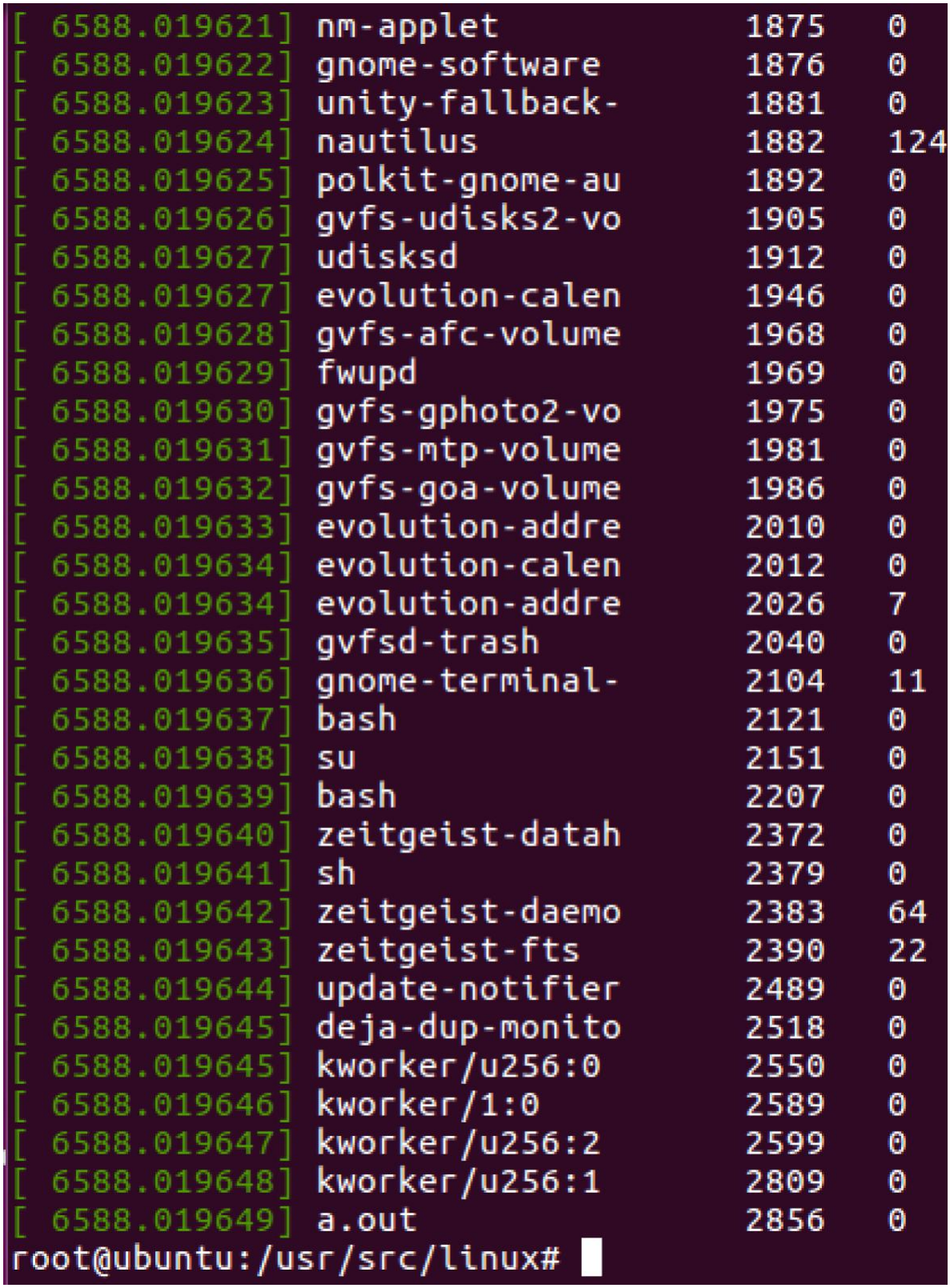
用gcc编译源程序

```
gcc test.c
```

运行程序

```
./a.out
```

```
root@ubuntu:/usr/src/linux# gcc test.c
test.c: In function ‘main’:
test.c:7:2: warning: implicit declaration of function ‘syscall’ [-Wimplicit-func
tion-declaration]
  syscall(__NR_mysyscall);
  ^
test.c:8:2: warning: implicit declaration of function ‘system’ [-Wimplicit-funct
ion-declaration]
  system("dmesg 1>log");
  ^
root@ubuntu:/usr/src/linux# ./a.out
[ 6588.012529] System Page Fault Number:755030
[ 6588.012532] Current Process Page Fault Number:67
[ 6588.012532] Each Process:
[ 6588.012534] systemd                 1       0
[ 6588.012535] kthreadd                2       0
[ 6588.012536] ksoftirqd/0             3       0
[ 6588.012537] kworker/0:0H            5       0
[ 6588.012537] rcu_sched               7       0
[ 6588.012538] rcu_bh                  8       0
[ 6588.012539] migration/0             9       0
[ 6588.012540] watchdog/0              10      0
[ 6588.012541] cpuhp/0                 11      0
[ 6588.012542] cpuhp/1                 12      0
[ 6588.012543] watchdog/1              13      0
[ 6588.012543] migration/1             14      0
[ 6588.012544] ksoftirqd/1             15      0
[ 6588.012545] kworker/1:0H            17      0
[ 6588.012546] cpuhp/2                 18      0
[ 6588.012547] watchdog/2              19      0
[ 6588.012547] migration/2             20      0
[ 6588.012548] ksoftirqd/2             21      0
[ 6588.012549] kworker/2:0H            23      0
[ 6588.012550] cpuhp/3                 24      0
[ 6588.012551] watchdog/3              25      0
[ 6588.012551] migration/3             26      0
[ 6588.012552] ksoftirqd/3             27      0
[ 6588.012553] kworker/3:0             28      0
[ 6588.012554] kworker/3:0H            29      0
[ 6588.012555] kdevtmpfs               30      0
[ 6588.012556] netns                   31      0
[ 6588.012556] khungtaskd              32      0
[ 6588.012557] oom_reaper              33      0
[ 6588.012558] writeback               34      0
```



【回答问题】

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？

```
[ 6588.012529] System Page Fault Number:755030
[ 6588.012532] Current Process Page Fault Number:67
```

```
[ 7506.122074] System Page Fault Number:765567
[ 7506.122078] Current Process Page Fault Number:68
```

```
[ 7659.355365] System Page Fault Number:766458
[ 7659.355367] Current Process Page Fault Number:68
```

```
[ 7974.319439] System Page Fault Number:767094
[ 7974.319441] Current Process Page Fault Number:69
```

```
[ 8041.227227] System Page Fault Number:767646
[ 8041.227229] Current Process Page Fault Number:68
```

```
[ 8127.106654] System Page Fault Number:768176
[ 8127.106657] Current Process Page Fault Number:69
```

可以看到，操作系统原理上的缺页次数随着test程序调用次数的增加而增加,而test程序打印的缺页次数稳定在一定范围内.因为test程序打印的缺页次数并不是操作系统原理上的缺页次数.操作系统原理上的缺页次数是页面置换次数乘以物理块数，而test程序打印的是__do_page_fault 函数执行的次数，即页访问出错的次数.

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。

存在. 可以采用采用系统调用拦截，改变某一个系统调用号对应的服务程序为我们自己的编写的程序，从而相当于添加了我们自己的系统调用.

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。

不安全.因为系统调用是应用程序主动进入操作系统内核的入口，如果可以修改系统调用，就可以通过应用程序非法访问内核，可能导致程序崩溃.

五、讨论和心得

本次实验我收获很多，除了关于操作添加系统调用这一块非常熟练了之外，因为在做实验的过程中总是出错甚至自己胡乱操作一通把系统搞崩掉，所以重装了两次虚拟机和重建了很多次内核，现在对于虚拟机的安装和内核重建也非常熟练了.

1. 第一次出错是因为没有按照试验手册按步骤进行操作，以为不需要重建内核直接修改系统调用文件编译就可以了，并且使用的是较新的内核linux5.9.2，和实验手册上的函数不符之处有点多，在发现arch/x86/mm/fault.c里面没有__do_page_fault函数的时候无法继续实验，并且后来操作不当导致系统频繁卡顿，于是重装了虚拟机.

2. 第二次还是用的ubuntu20.04，自带的内核版本是linux5.x，我按照lab3的步骤来重建4.6的内核了，发现lab3的步骤比lab2简洁，以为不要紧，结果在编译的时候频繁出错. 通过搜索解决了几个error，比如自己写补丁文件，猜测这一步出错和在自己重建内核的时候没有打补丁有关. 后来遇到了解决不了的error，查询网络说是因为ubuntu20.04自带的系统文件版本过高，需要降级，降级的过程十分复杂，同时gcc也需要降级，过程过于繁琐且降级过程中又出现了很多bug，遂放弃.

3. 第三次为了谨慎起见，下载了ubuntu16.04.7(64位)的版本，虚拟机建好后自带内核4.15.x,开始重建内核4.6.一切顺利，直到sudo make modules -j8之后，重启了uanme -r还是4.15.x版本. 询问助教后知道了是降级的情况下默认开机会调用最新版的内核，需要通过修改grub配置文件，在开机中显示选择内核版本的菜单栏，使用gedit打开配置文件，目录为/etc/default/grub.把修改GRUB_HIDDEN_TIMEOUT=10,GRUB_TIMEOUT=10(网上说修改两种的都有为了防止没有效果我都给修改了)，sudo reboot重启.这次在开机的时候出现的10秒倒计时，按esc键进入内核选择界面.

选择Advanced options for Ubuntu

选择Unbuntu, with Linux 4.6.0 (这里出现了选择Unbuntu, with Linux 4.6.0.old是因为我后来出错又又又建了一遍内核)

enter进去, 终于调用了4.6.0的内核.接下来编译运行test.c文件, 没有报错, 但是也没有结果, 我落泪了.询问助教后才知道因为自己使用的是64位虚拟机, 需要修改的是syscall_64.tbl, 并且64位ubuntu中的223位是已经被占用了的, 在助教的建议下用335位实现systemcall.

4. 修改后进行了第四次编译, 这次也没有结果. 后来反复检查试验过程发现还是是自己把32位修改64的时候test.c文件没有同步把systemcall的系统调用号改成335, 导致没有结果. 改完test.c后快乐地出现了结果.

总结: ubuntu和linux最好不要用太新的版本, 不然需要降级; 内核重建参考lab2的完整做法最为保险, 一定要记得打补丁; 内核重建如果是降级需要修改grub文件, 在开机的时候选择旧版本的内核; 如果使用的是64位操作系统, 系统调用的修改和23位略有不同, 要注意区分.

六、附录

[1] ubuntu内核，在http://kernel.ubuntu.com/~kernel-ppa/mainline/

[2] Linux官方内核，在https://www.kernel.org/pub/linux/kernel/

[3] Linux公社http://www.linuxidc.com/Linux/2016-06/132707.htm

[4] ubuntu上安装 Linux Kernel 4.4，http://www.linuxidc.com/Linux/2016-

01/127383.htm