

浙江大学

本科实验报告

课程名称：操 作 系 统

实验名称：添加一个系统调用

姓 名：张 溢 弛

学 院：计算机科学与技术学院

系：软件工程

专 业：软件工程

学 号：3180103772

指导教师：季江民

2020 年 12 月 15 日

目 录

一、实验环境.....	3
二、实验内容和结果及分析.....	3
2.1 实验设计思路.....	3
2.2 实验具体步骤和截图.....	4
2.3 结果分析.....	19
2.4. 源程序.....	19
2.5 回答问题.....	20
三、讨论、心得（20分）.....	22

浙江大学实验报告

课程名称：____ 操作系统 _____ 实验类型：____ 综合型 _____

实验项目名称：____ 添加系统调用 _____

学生姓名：____ 张溢弛 _____ 学号：____ 3180103772 _____

电子邮件地址：____ 3180103772@zju.edu.cn _____

实验日期：____ 2020 年 11 月 30 日 _____

一、实验环境

操作系统：Windows 10

虚拟机：VMware Workstation Pro 16

Linux 版本：Ubuntu 16.04

内核版本：原内核 4.15.0-generic 要安装的内核版本是 4.8.0

代码编辑器：Visual Studio Code/gedit

二、实验内容和结果及分析

2.1 实验设计思路

本次实验的设计思路在实验指导书中详细给出，非常清晰，只需要按照实验指导书上面的步骤来做就可以，具体的步骤有：

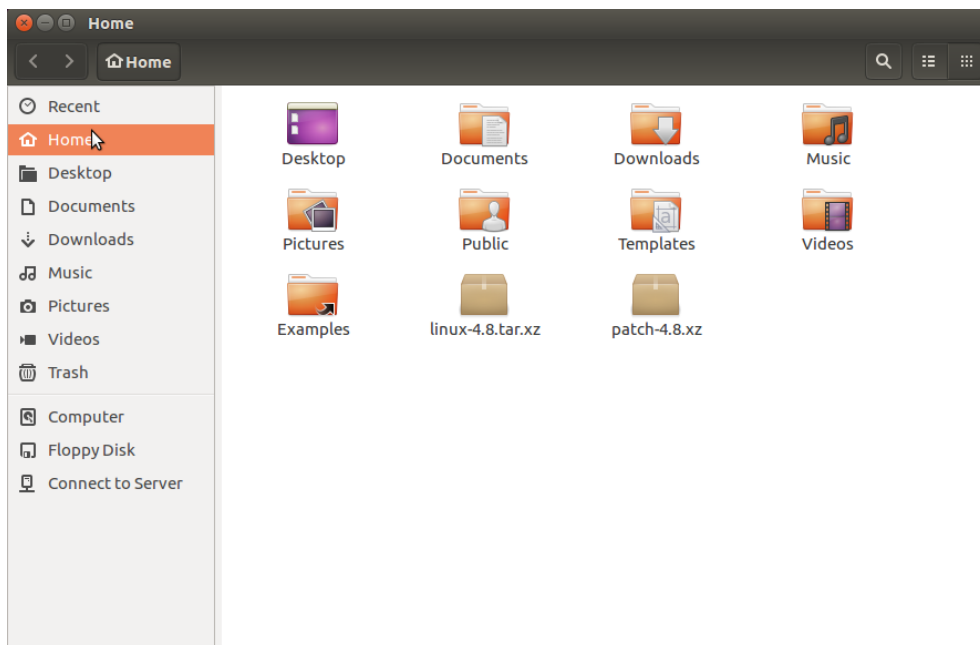
- 下载解压内核和补丁，并给内核打补丁
- 配置内核
- 添加系统调用号(我用的是 223)
- 在系统调用表中添加对应的项目
- 修改对应文件中的内核代码

- 实现简单的 `sys_mysyscall`
- 编译并重启内核
- 编写用户态程序并输出结果

2.2 实验具体步骤和截图

1 下载一份内核源代码

从 <http://mirrors.aliyun.com/linux-kernel/> 下载 4.8 版本的内核和补丁：
`linux-4.8.tar.xz` 和 `patch-4.8.xz`（补丁）文件，我将其存放在主目录下面



2. 部署内核源代码

把内核代码文件 `linux-4.8.tar.xz` 存放在主目录（`~`）中。解压内核包，生成的内核源代码放在 `linux.4.8` 目录中：

```
tar -xvf linux-4.8.tar.xz
```

在 `linux.4.8` 目录中打内核补丁：

```
xz -d patch-4.8.xz | patch -p1
```

```

linux-4.8/tools/vm/.gitignore
linux-4.8/tools/vm/Makefile
linux-4.8/tools/vm/page-types.c
linux-4.8/tools/vm/page_owner_sort.c
linux-4.8/tools/vm/slabinfo-gnuplot.sh
linux-4.8/tools/vm/slabinfo.c
linux-4.8/usr/
linux-4.8/usr/.gitignore
linux-4.8/usr/Kconfig
linux-4.8/usr/Makefile
linux-4.8/usr/gen_init_cpio.c
linux-4.8/usr/initramfs_data.5
linux-4.8/virt/
linux-4.8/virt/Makefile
linux-4.8/virt/kvm/
linux-4.8/virt/kvm/Kconfig
linux-4.8/virt/kvm/arm/
linux-4.8/virt/kvm/arm/arch_timer.c
linux-4.8/virt/kvm/arm/hyp/
linux-4.8/virt/kvm/arm/hyp/timer-sr.c
linux-4.8/virt/kvm/arm/hyp/vgic-v2-sr.c
linux-4.8/virt/kvm/arm/pmu.c
linux-4.8/virt/kvm/arm/trace.h
linux-4.8/virt/kvm/arm/vgic/
linux-4.8/virt/kvm/arm/vgic/vgic-init.c
linux-4.8/virt/kvm/arm/vgic/vgic-irqfd.c
linux-4.8/virt/kvm/arm/vgic/vgic-tts.c
linux-4.8/virt/kvm/arm/vgic/vgic-kvm-device.c
linux-4.8/virt/kvm/arm/vgic/vgic-mmio-v2.c
linux-4.8/virt/kvm/arm/vgic/vgic-mmio-v3.c
linux-4.8/virt/kvm/arm/vgic/vgic-mmio.c
linux-4.8/virt/kvm/arm/vgic/vgic-mmio.h
linux-4.8/virt/kvm/arm/vgic/vgic-v2.c
linux-4.8/virt/kvm/arm/vgic/vgic-v3.c
linux-4.8/virt/kvm/arm/vgic/vgic.c
linux-4.8/virt/kvm/arm/vgic/vgic.h
linux-4.8/virt/kvm/async_pf.c
linux-4.8/virt/kvm/async_pf.h
linux-4.8/virt/kvm/coalesced_mmio.c
linux-4.8/virt/kvm/coalesced_mmio.h
linux-4.8/virt/kvm/eventfd.c
linux-4.8/virt/kvm/irqchip.c
linux-4.8/virt/kvm/kvm_main.c
linux-4.8/virt/kvm/vfio.c
linux-4.8/virt/kvm/vfio.h
linux-4.8/virt/tlb/
linux-4.8/virt/tlb/Kconfig
linux-4.8/virt/tlb/Makefile
linux-4.8/virt/tlb/irgbypass.c
randomstar@ubuntu:~$

```

```

randomstar@ubuntu:~$ xz -d patch-4.8.xz | patch -p1
randomstar@ubuntu:~$

```

1. 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，需要下载并安装 `libncurses5-dev`，下载并安装命令如下：

```
apt-get install libncurses5-dev libssl-dev
```

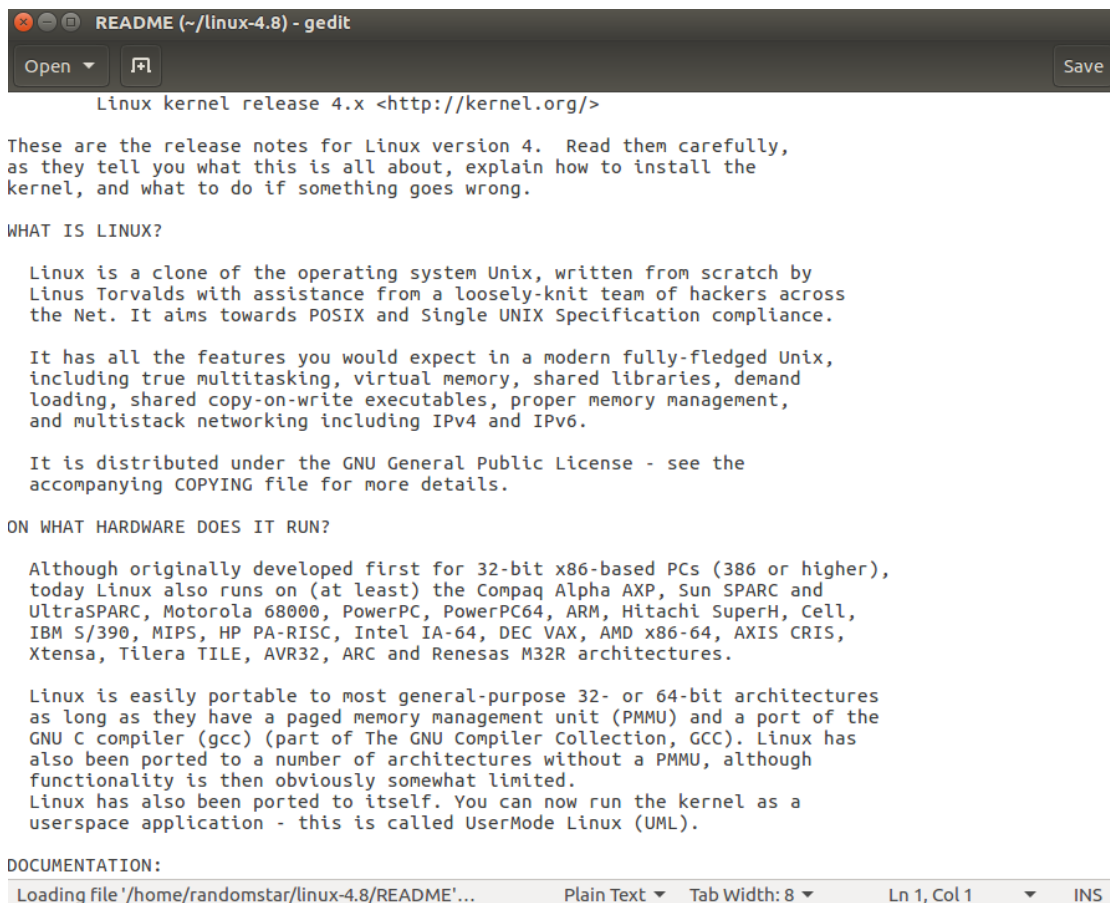
若上面这一步提示错误信息，则输入下面的命令 `sudo apt-get -f install`，建立库依赖关系。

```

randomstar@ubuntu:~$ sudo apt-get install libncurses5-dev libssl-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libssl-doc libssl1.0.0 libtinfo-dev zlib1g zlib1g-dev
Suggested packages:
  ncurses-doc
The following NEW packages will be installed:
  libncurses5-dev libssl-dev libssl-doc libtinfo-dev zlib1g-dev
The following packages will be upgraded:
  libssl1.0.0 zlib1g
2 upgraded, 5 newly installed, 0 to remove and 406 not upgraded.
Need to get 5,594 kB of archives.
After this operation, 16.1 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://us.archive.ubuntu.com/ubuntu xenial-updates/main i386 zlib1g i386 1:1.2.8.dfsg-2ubuntu4.3 [52.3 kB]
Get:2 http://us.archive.ubuntu.com/ubuntu xenial-updates/main i386 libssl1.0.0 i386 1.0.2g-1ubuntu4.17 [909 kB]
10% [2 libssl1.0.0 257 kB/909 kB 28%]

```

查看 README 文件：



```
Linux kernel release 4.x <http://kernel.org/>

These are the release notes for Linux version 4. Read them carefully,
as they tell you what this is all about, explain how to install the
kernel, and what to do if something goes wrong.

WHAT IS LINUX?

Linux is a clone of the operating system Unix, written from scratch by
Linus Torvalds with assistance from a loosely-knit team of hackers across
the Net. It aims towards POSIX and Single UNIX Specification compliance.

It has all the features you would expect in a modern fully-fledged Unix,
including true multitasking, virtual memory, shared libraries, demand
loading, shared copy-on-write executables, proper memory management,
and multistack networking including IPv4 and IPv6.

It is distributed under the GNU General Public License - see the
accompanying COPYING file for more details.

ON WHAT HARDWARE DOES IT RUN?

Although originally developed first for 32-bit x86-based PCs (386 or higher),
today Linux also runs on (at least) the Compaq Alpha AXP, Sun SPARC and
UltraSPARC, Motorola 68000, PowerPC, PowerPC64, ARM, Hitachi SuperH, Cell,
IBM S/390, MIPS, HP PA-RISC, Intel IA-64, DEC VAX, AMD x86-64, AXIS CRIS,
Xtensa, Tilera TILE, AVR32, ARC and Renesas M32R architectures.

Linux is easily portable to most general-purpose 32- or 64-bit architectures
as long as they have a paged memory management unit (PMMU) and a port of the
GNU C compiler (gcc) (part of The GNU Compiler Collection, GCC). Linux has
also been ported to a number of architectures without a PMMU, although
functionality is then obviously somewhat limited.
Linux has also been ported to itself. You can now run the kernel as a
userspace application - this is called UserMode Linux (UML).

DOCUMENTATION:
Loading file '/home/randomstar/linux-4.8/README'... Plain Text Tab Width: 8 Ln 1, Col 1 INS
```

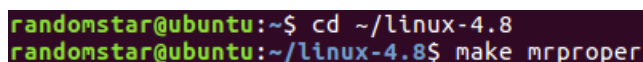
配置内核：

在编译内核前，一般来说都需要对内核进行相应的配置。配置是精确控制新内核功能的机会。配置过程也控制哪些需编译到内核的二进制映像中(在启动时被载入)，哪些是需要时才装入的内核模块（module）。

```
cd ~/.linux-4.8
```

第一次编译的话，有必要将内核源代码树置于一种完整和一致的状态。因此，我们推荐执行命令 `make mrproper`。它将清除目录下所有配置文件和先前生成核心时产生的.o 文件：

```
make mrproper
```



```
randomstar@ubuntu:~$ cd ~/.linux-4.8
randomstar@ubuntu:~/linux-4.8$ make mrproper
```

为了与正在运行的操作系统内核的运行环境匹配，可以先把当前已配置好的文件复制到当前目录下，新的文件名为.config 文件：

```
cp /boot/config-`uname -r` .config
```

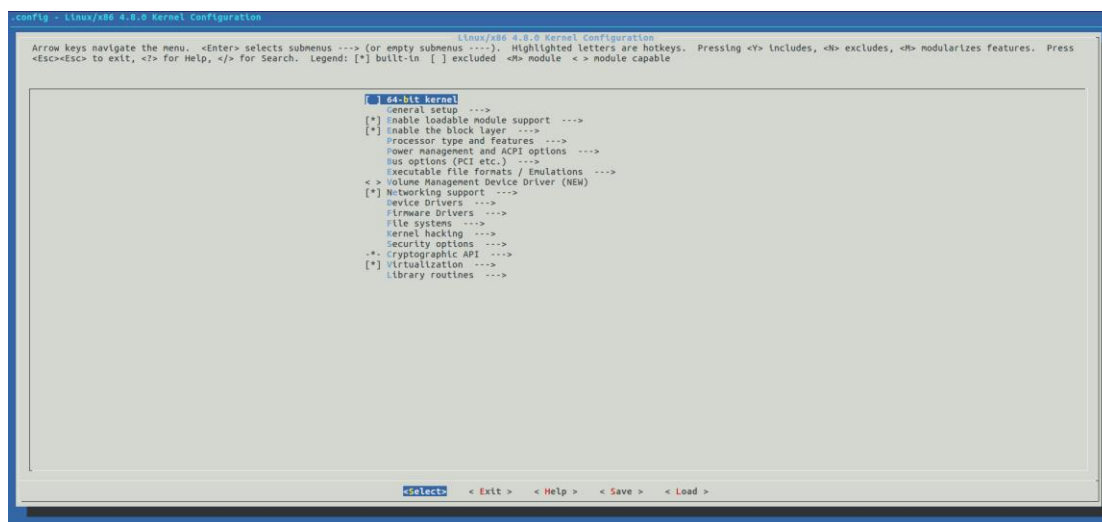
```
make menuconfig
```

```
randomstar@ubuntu:~/linux-4.8$ ls /boot/config-4.15.0-45-generic
/boot/config-4.15.0-45-generic
randomstar@ubuntu:~/linux-4.8$ sudo cp /boot/config-`uname -r` .config
randomstar@ubuntu:~/linux-4.8$
```

```
randomstar@ubuntu:~/linux-4.8$ make menuconfig
HOSTCC scripts/basic/fixdep
HOSTCC scripts/kconfig/mconf.o
SHIPPED scripts/kconfig/zconf.tab.c
SHIPPED scripts/kconfig/zconf.lex.c
SHIPPED scripts/kconfig/zconf.hash.c
HOSTCC scripts/kconfig/zconf.tab.o
HOSTCC scripts/kconfig/lxdialog/checklist.o
HOSTCC scripts/kconfig/lxdialog/util.o
HOSTCC scripts/kconfig/lxdialog/inputbox.o
HOSTCC scripts/kconfig/lxdialog/textbox.o
HOSTCC scripts/kconfig/lxdialog/yesno.o
HOSTCC scripts/kconfig/lxdialog/menubox.o
HOSTLD scripts/kconfig/mconf
scripts/kconfig/mconf Kconfig
.config:4304:warning: symbol value 'm' invalid for GPIO_MB86S7X
configuration written to .config

*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

randomstar@ubuntu:~/linux-4.8$
```



2. 添加系统调用号

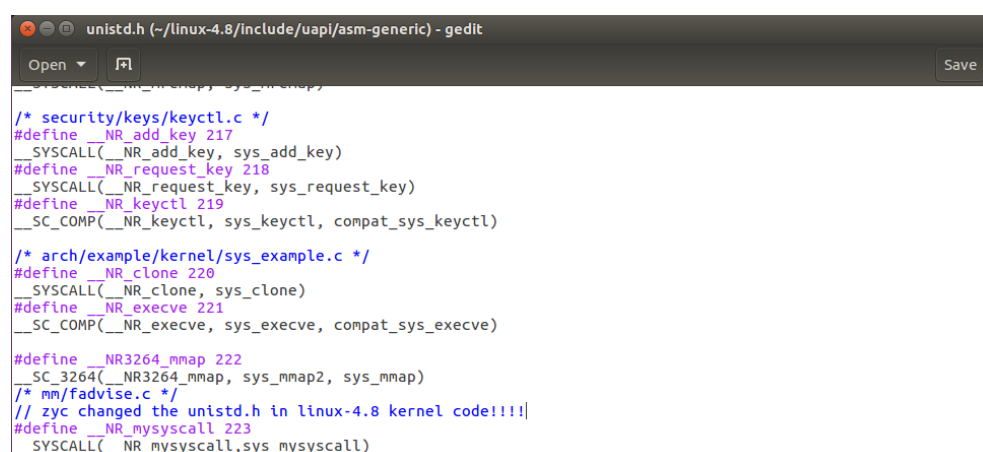
系统调用号在文件 `unistd.h` 里面定义。这个文件可能在你的 Linux 系统上会有两个版本：一个是 C 库文件版本，出现的地方是在 ubuntu 16.04 只修改 `/usr/include/asm-generic/unistd.h`；另外还有一个版本是内核自己的 `unistd.h`，出现的地方是在你解压出来的内核代码的对应位置（比如

include/uapi/asm-generic/unistd.h)。当然，也有可能这个 C 库文件只是一个到对应内核文件的连接。现在，你要做的就是文件 unistd.h 中添加我们的系统调用号：__NR_mysyscall，x86 体系架构的系统调用号 223 没有使用，我们新的系统调用号定义为 223 号，如下所示：

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

kernel 4.8 为：include/uapi/asm-generic/unistd.h

修改内核中的代码



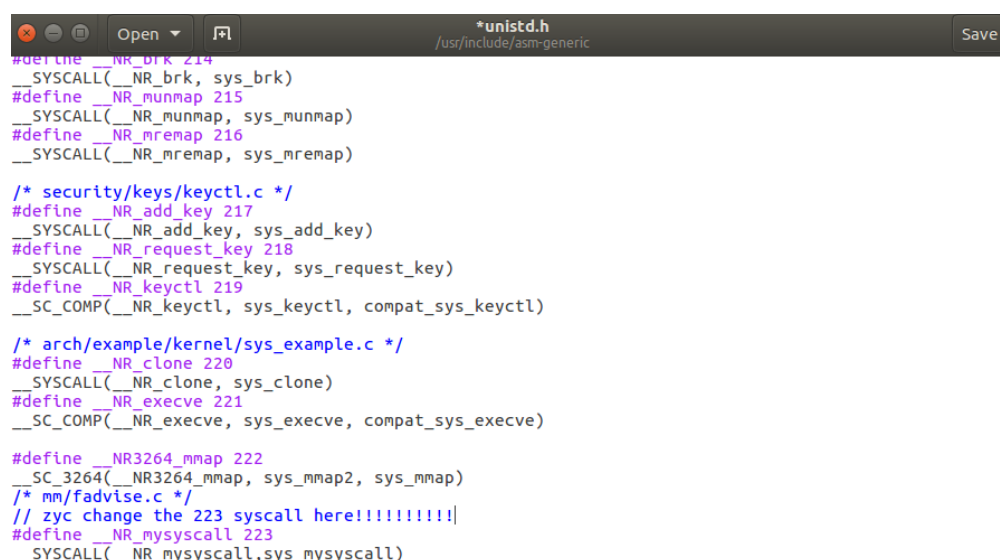
```
unistd.h (/linux-4.8/include/uapi/asm-generic) - gedit
Open Save

/* security/keys/keyctl.c */
#define __NR_add_key 217
__SYSCALL(__NR_add_key, sys_add_key)
#define __NR_request_key 218
__SYSCALL(__NR_request_key, sys_request_key)
#define __NR_keyctl 219
__SC_COMP(__NR_keyctl, sys_keyctl, compat_sys_keyctl)

/* arch/example/kernel/sys_example.c */
#define __NR_clone 220
__SYSCALL(__NR_clone, sys_clone)
#define __NR_execve 221
__SC_COMP(__NR_execve, sys_execve, compat_sys_execve)

#define __NR3264_mmap 222
__SC_3264(__NR3264_mmap, sys_mmap2, sys_mmap)
/* mm/fadvise.c */
// zyc changed the unistd.h in linux-4.8 kernel code!!!!
#define __NR_mysyscall 223
__SYSCALL(__NR_mysyscall, sys_mysyscall)
```

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



```
*unistd.h
/usr/include/asm-generic
Open Save

#define __NR_brk 214
__SYSCALL(__NR_brk, sys_brk)
#define __NR_munmap 215
__SYSCALL(__NR_munmap, sys_munmap)
#define __NR_mremap 216
__SYSCALL(__NR_mremap, sys_mremap)

/* security/keys/keyctl.c */
#define __NR_add_key 217
__SYSCALL(__NR_add_key, sys_add_key)
#define __NR_request_key 218
__SYSCALL(__NR_request_key, sys_request_key)
#define __NR_keyctl 219
__SC_COMP(__NR_keyctl, sys_keyctl, compat_sys_keyctl)

/* arch/example/kernel/sys_example.c */
#define __NR_clone 220
__SYSCALL(__NR_clone, sys_clone)
#define __NR_execve 221
__SC_COMP(__NR_execve, sys_execve, compat_sys_execve)

#define __NR3264_mmap 222
__SC_3264(__NR3264_mmap, sys_mmap2, sys_mmap)
/* mm/fadvise.c */
// zyc change the 223 syscall here!!!!!!!!!!
#define __NR_mysyscall 223
__SYSCALL(__NR_mysyscall, sys_mysyscall)
```

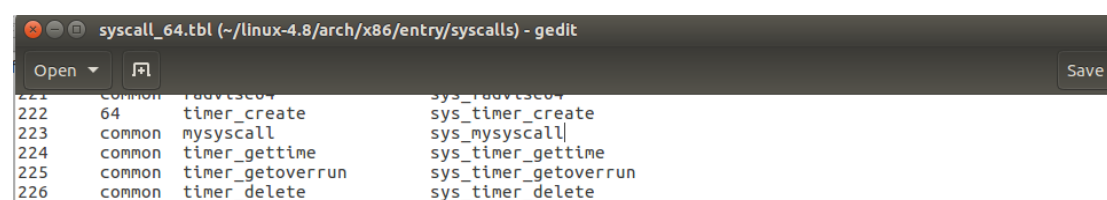
3. 在系统调用表中添加或修改相应表项

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)
```

223	common	mysyscall	sys_mysyscall
-----	--------	-----------	---------------

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。

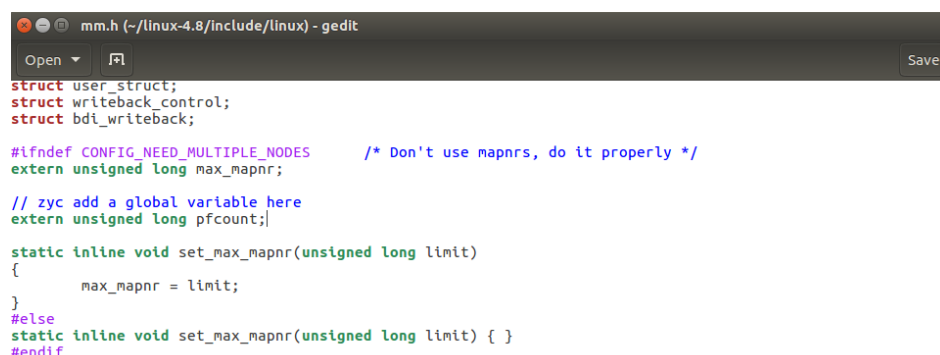


```
syscall_64.tbl (~/.linux-4.8/arch/x86/entry/syscalls) - gedit
Open Save
221 common readvsc04 sys_readvsc04
222 64 timer_create sys_timer_create
223 common mysyscall sys_mysyscall
224 common timer_gettime sys_timer_gettime
225 common timer_getoverrun sys_timer_getoverrun
226 common timer_delete sys_timer_delete
```

4. 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：



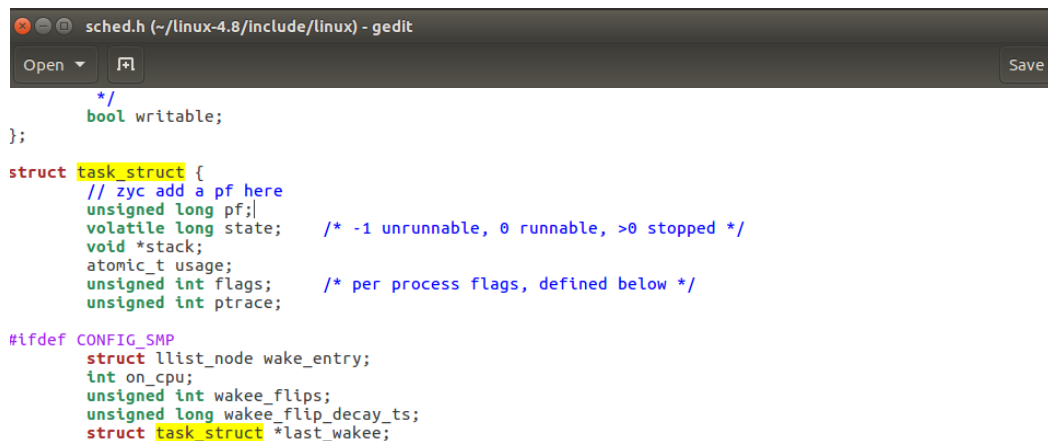
```
mm.h (~/.linux-4.8/include/linux) - gedit
Open Save
struct user_struct;
struct writeback_control;
struct bdi_writeback;

#ifdef CONFIG_NEED_MULTIPLE_NODES /* Don't use mapnr, do it properly */
extern unsigned long max_mapnr;

// zyc add a global variable here
extern unsigned long pfcount;

static inline void set_max_mapnr(unsigned long limit)
{
    max_mapnr = limit;
}
#else
static inline void set_max_mapnr(unsigned long limit) { }
#endif
```

要记录进程产生的缺页次数，首先在进程 `task_struct` 中增加成员 `pf01`，在 `include/linux/sched.h` 文件中的 `task_struct` 结构中添加 `pf` 字段：

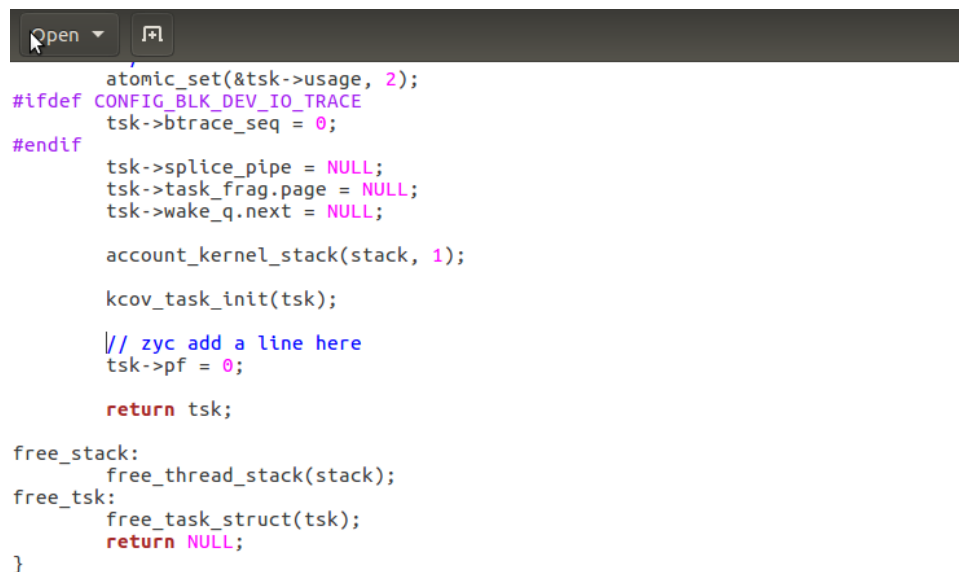


```
*/
bool writable;
};

struct task_struct {
    // zyc add a pf here
    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;

#ifdef CONFIG_SMP
    struct llist_node wake_entry;
    int on_cpu;
    unsigned int wakee_flips;
    unsigned long wakee_flip_decay_ts;
    struct task_struct *last_wakee;
#endif
};
```

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：



```
atomic_set(&tsk->usage, 2);
#ifdef CONFIG_BLK_DEV_IO_TRACE
    tsk->btrace_seq = 0;
#endif
tsk->splice_pipe = NULL;
tsk->task_frag.page = NULL;
tsk->wake_q.next = NULL;

account_kernel_stack(stack, 1);

kcov_task_init(tsk);

// zyc add a line here
tsk->pf = 0;

return tsk;

free_stack:
    free_thread_stack(stack);
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：

```

~/linux-4.8/arch/x86/mm) - gedit
Open  [icon]

#include <asm/trace/exceptions.h>

/*
 * Page fault error code bits:
 */
*   bit 0 ==    0: no page found          1: protection fault
*   bit 1 ==    0: read access            1: write access
*   bit 2 ==    0: kernel-mode access    1: user-mode access
*   bit 3 ==    1: use of reserved bit detected
*   bit 4 ==    1: fault was an instruction fetch
*   bit 5 ==    1: protection keys block access
*/
enum x86_pf_error_code {
    PF_PROT          = 1 << 0,
    PF_WRITE         = 1 << 1,
    PF_USER          = 1 << 2,
    PF_RSVD          = 1 << 3,
    PF_INSTR         = 1 << 4,
    PF_PK            = 1 << 5,
};

// zyc add a line here
unsigned long pfcount;

static ninline 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;

    // zyc add 2 line here!
    pfcount++;
    current->pf++;

    tsk = current;
    mm = tsk->mm;

```

5. 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。

```

linux-4.8/kernel) - gedit
Open ▾
#include <linux/uidgid.h>
#include <linux/cred.h>

#include <linux/kmsg_dump.h>
/* Move somewhere else to avoid recompiling? */
#include <generated/utsrelease.h>

#include <asm/uaccess.h>
#include <asm/io.h>
#include <asm/unistd.h>

// zyc add mysyscall here
extern unsigned long pfcount;
asmlinkage int sys_mysyscall(void) {
    printk("zyc_syscall\n");
    printk("zyc all apge fault: %lu\n", pfcount);
    printk("zyc current page fault: %lu\n", current->pf);
    struct task_struct *p = &init_task;
    for(;;(p=next_task(p)) != &init_task;) {
        printk("zyc the dirty page of process %d: %d\n", p->pid, p->nr_dirtied);
    }
    printk("zyc syscall end!");
    return 0;
}

```

6. 编译内核和重启内核

用 make 工具编译内核：`make` 此步花费了一个多小时的时间

```

IHEX      firmware/cis/SW_8xx_SER.cis
IHEX      firmware/ositech/Xilinx70D.bin
IHEX      firmware/advansys/mcode.bin
IHEX      firmware/advansys/38C1600.bin
IHEX      firmware/advansys/3550.bin
IHEX      firmware/advansys/38C0800.bin
IHEX      firmware/qlogic/1040.bin
IHEX      firmware/qlogic/1280.bin
IHEX      firmware/qlogic/12160.bin
IHEX      firmware/korg/k1212.dsp
IHEX      firmware/ess/maestro3_assp_kernel.fw
IHEX      firmware/ess/maestro3_assp_minisrc.fw
IHEX      firmware/yamaha/ds1_ctrl.fw
IHEX      firmware/yamaha/ds1_dsp.fw
IHEX      firmware/yamaha/ds1e_ctrl.fw
IHEX      firmware/yamaha/yss225_registers.bin
IHEX      firmware/tehuti/bdx.bin
IHEX      firmware/tigon/tg3.bin
IHEX      firmware/tigon/tg3_tso.bin
IHEX      firmware/tigon/tg3_tso5.bin
IHEX      firmware/3com/typhoon.bin
IHEX2FW   firmware/emi26/loader.fw
IHEX2FW   firmware/emi26/firmware.fw
IHEX2FW   firmware/emi26/bitstream.fw
IHEX2FW   firmware/emi62/loader.fw
IHEX2FW   firmware/emi62/bitstream.fw
IHEX2FW   firmware/emi62/spdif.fw
IHEX2FW   firmware/emi62/midi.fw
IHEX      firmware/kaweth/new_code.bin
IHEX      firmware/kaweth/trigger_code.bin
IHEX      firmware/kaweth/new_code_fix.bin
IHEX      firmware/kaweth/trigger_code_fix.bin
IHEX      firmware/ti_3410.fw
IHEX      firmware/ti_5052.fw
IHEX      firmware/mts_cdma.fw
IHEX      firmware/mts_gsm.fw
IHEX      firmware/mts_edge.fw
H16TOFW   firmware/edgeport/boot.fw
H16TOFW   firmware/edgeport/boot2.fw
H16TOFW   firmware/edgeport/down.fw
H16TOFW   firmware/edgeport/down2.fw
IHEX      firmware/edgeport/down3.bin
IHEX2FW   firmware/whiteheat_loader.fw
IHEX2FW   firmware/whiteheat.fw
IHEX2FW   firmware/keyspan_pda/keyspan_pda.fw
IHEX2FW   firmware/keyspan_pda/xircom_pgs.fw
IHEX      firmware/cpia2/stv0672_vp4.bin
IHEX      firmware/yam/1200.bin
IHEX      firmware/yam/9600.bin
randomstar@ubuntu:~/linux-4.8$

```

sudo make modules

```

randomstar@ubuntu:~/linux-4.8$ sudo make modules
[sudo] password for randomstar:
CHK      include/config/kernel.release
CHK      include/generated/uapi/linux/version.h
CHK      include/generated/utsrelease.h
CHK      include/generated/bounds.h
CHK      include/generated/timeconst.h
CHK      include/generated/asm-offsets.h
CALL     scripts/checksyscalls.sh
Building modules, stage 2.
MODPOST 4588 modules

```

sudo make modules_install

```
INSTALL arch/x86/kernel/cpuid.ko
INSTALL arch/x86/kernel/msr.ko
INSTALL arch/x86/kvm/kvm-amd.ko
INSTALL arch/x86/kvm/kvm-intel.ko
INSTALL arch/x86/kvm/kvm.ko
INSTALL arch/x86/oprofile/oprofile.ko
INSTALL arch/x86/platform/atom/punit_atom_debug.ko
INSTALL arch/x86/platform/iris/iris.ko
INSTALL arch/x86/platform/scx200/scx200.ko
INSTALL crypto/842.ko
INSTALL crypto/ablk_helper.ko
INSTALL crypto/af_alg.ko
INSTALL crypto/algif_aead.ko
INSTALL crypto/algif_hash.ko
INSTALL crypto/algif_rng.ko
INSTALL crypto/algif_skcipher.ko
INSTALL crypto/ansi_cprng.ko
INSTALL crypto/anubis.ko
INSTALL crypto/arc4.ko
INSTALL crypto/asymmetric_keys/pkcs7_test_key.ko
INSTALL crypto/async_tx/async_memcpy.ko
INSTALL crypto/async_tx/async_pq.ko
INSTALL crypto/async_tx/async_raid6_recov.ko
INSTALL crypto/async_tx/async_tx.ko
INSTALL crypto/async_tx/async_xor.ko
INSTALL crypto/authenc.ko
INSTALL crypto/authencscn.ko
INSTALL crypto/blowfish_common.ko
INSTALL crypto/blowfish_generic.ko
INSTALL crypto/camellia_generic.ko
INSTALL crypto/cast5_generic.ko
INSTALL crypto/cast6_generic.ko
INSTALL crypto/cast_common.ko
INSTALL crypto/ccm.ko
INSTALL crypto/chacha20_generic.ko
INSTALL crypto/chacha20poly1305.ko
INSTALL crypto/cmac.ko
INSTALL crypto/crc32_generic.ko
INSTALL crypto/cryptd.ko
INSTALL crypto/crypto_user.ko
INSTALL crypto/deflate.ko
INSTALL crypto/des_generic.ko
INSTALL crypto/ecdh_generic.ko
INSTALL crypto/echainiv.ko
INSTALL crypto/fcrypt.ko
INSTALL crypto/keywrap.ko
INSTALL crypto/khazad.ko
INSTALL crypto/lrw.ko
INSTALL crypto/lz4.ko
```

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：

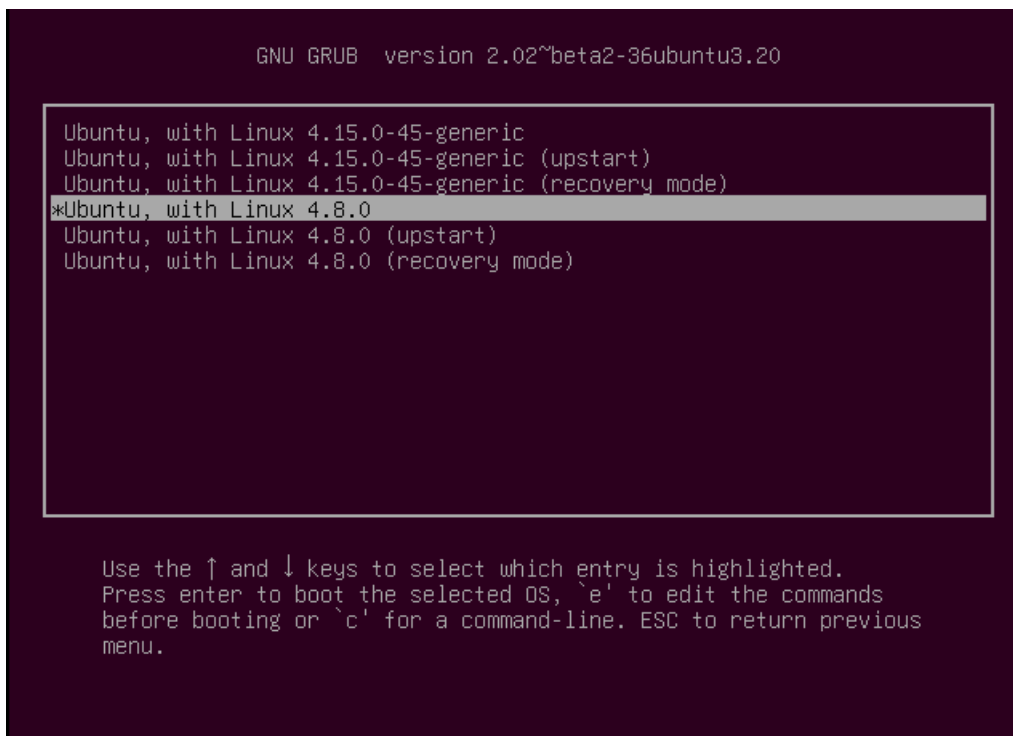
```
sudo make install
```

```
randomstar@ubuntu:~/linux-4.8$ sudo make install
sh ./arch/x86/boot/install.sh 4.8.0 arch/x86/boot/bzImage \
    System.map "/boot"
run-parts: executing /etc/kernel/postinst.d/apt-auto-removal 4.8.0 /boot/vmlinuz-4.8.0
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 4.8.0 /boot/vmlinuz-4.8.0
update-initramfs: Generating /boot/initrd.img-4.8.0
run-parts: executing /etc/kernel/postinst.d/pm-utils 4.8.0 /boot/vmlinuz-4.8.0
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 4.8.0 /boot/vmlinuz-4.8.0
run-parts: executing /etc/kernel/postinst.d/update-notifier 4.8.0 /boot/vmlinuz-4.8.0
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 4.8.0 /boot/vmlinuz-4.8.0
Generating grub configuration file ...
Warning: Setting GRUB_TIMEOUT to a non-zero value when GRUB_HIDDEN_TIMEOUT is set is no longer supported.
Found linux image: /boot/vmlinuz-4.15.0-45-generic
Found initrd image: /boot/initrd.img-4.15.0-45-generic
Found linux image: /boot/vmlinuz-4.8.0
Found initrd image: /boot/initrd.img-4.8.0
Found memtest86+ image: /boot/memtest86+.elf
Found memtest86+ image: /boot/memtest86+.bin
done
```

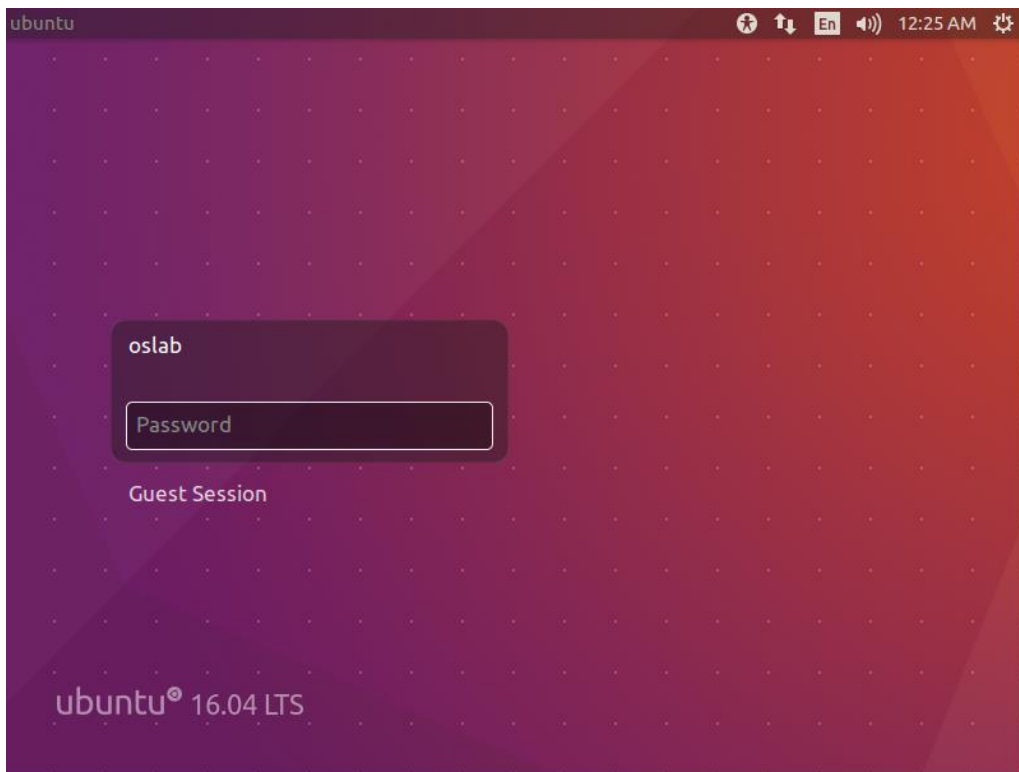
我们已经编译了内核 bzImage，放到了指定位置/boot。现在，请你重启主机系统，期待编译过的 Linux 操作系统内核正常运行！

`sudo reboot`

此处需要在关机之后立即按 **ESC** 键进入内核选择页面，并选择新内核



然后我们成功进入新的内核对应的操作系统界面



可以输入指令查看当前内核的版本

```
randomstar@ubuntu: ~  
randomstar@ubuntu:~$ uname -r  
4.8.0  
randomstar@ubuntu:~$
```

7. 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。

此处我们把用户态程序分为了调用系统调用和读取日志文件中的信息两个部分，为了方便读取信息的 debug 等操作，防止过多的系统调用，两个程序的详细源代码可以查看源代码这一部分，这里不再赘述

调用了系统调用之后可以用 `dmesg` 命令查看到输出：


```

mysyscall:
total page fault:112372  current page fault:127
1      systemd 0
2      kthreadd 0
4      kworker/0:0H 0
6      mm_percpu_wq 0
7      ksoftirqd/0 0
8      rcu_sched 0
9      rcu_bh 0
10     migration/0 0
11     watchdog/0 0
12     cpuhp/0 0
13     cpuhp/1 0
14     watchdog/1 0
15     migration/1 0
16     ksoftirqd/1 0
18     kworker/1:0H 0
19     kdevtmpfs 0
20     netns 0
21     rcu_tasks_kthre 0
22     kauditd 0
24     khungtaskd 0
25     oom_reaper 0
26     writeback 0
27     kcompactd0 0
28     ksmd 0
29     khugepaged 0
30     crypto 0
31     kintegrityd 0
32     kblockd 0
33     ata_sff 0
34     md 0
35     edac-poller 0
36     devfreq_wq 0
37     watchdogd 0
40     kswapd0 0
41     ecryptfs-kthrea 0
83     kthrotld 0
85     acpi_thermal_pm 0
87     scsi_eh_0 0
88     scsi_tmf_0 0
89     scsi_eh_1 0
90     scsi_tmf_1 0

```

```

1700    indicator-messa 0
1701    indicator-bluet 0
1704    indicator-power 0
1707    indicator-datet 0
1709    indicator-keybo 0
1710    indicator-sound 0
1713    indicator-print 0
1714    indicator-sessi 0
1717    indicator-appli 0
1726    pulseaudio 2
1749    evolution-sourc 0
1782    dconf-service 109
1798    compiz 3
1801    evolution-calen 0
1823    nm-applet 0
1824    unity-fallback- 0
1830    polkit-gnome-au 0
1837    evolution-calen 0
1842    nautilus 126
1850    vmttoolsd 0
1871    evolution-addre 0
1873    evolution-calen 0
1878    gvfs-udisks2-vo 0
1889    udisksd 0
1897    evolution-addre 7
1921    gvfs-gphoto2-vo 0
1927    gvfs-mtp-volume 0
1939    gvfs-goa-volume 0
1944    gvfs-afc-volume 0

```

也可以在日志 kern.log 中查看到相同的内容:

```

Nov 13 05:16:16 ubuntu kernel: [ 3838.081076] msyscall:
Nov 13 05:16:16 ubuntu kernel: [ 3838.081076] total page fault:112372 current page fault:127
Nov 13 05:16:16 ubuntu kernel: [ 3838.081076] 1 systemd 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081077] 2 kthreadd 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081078] 4 kworker/0:0H 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081078] 6 mm_percpu_wq 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081079] 7 ksoftirqd/0 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081079] 8 rcu_sched 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081080] 9 rcu_bh 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081081] 10 migration/0 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081082] 11 watchdog/0 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081082] 12 cpuhp/0 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081083] 13 cpuhp/1 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081083] 14 watchdog/1 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081084] 15 migration/1 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081085] 16 ksoftirqd/1 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081085] 18 kworker/1:0H 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081086] 19 kdevtmpfs 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081086] 20 netns 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081087] 21 rcu_tasks_kthre 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081088] 22 kauditd 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081089] 24 khungtaskd 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081089] 25 oom_reaper 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081090] 26 writeback 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081090] 27 kcompactd0 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081091] 28 ksmd 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081092] 29 khugepaged 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081092] 30 crypto 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081093] 31 kintegrityd 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081093] 32 kblockd 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081094] 33 ata_sff 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081095] 34 md 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081095] 35 edac-poller 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081096] 36 devfreq_wq 0
Nov 13 05:16:16 ubuntu kernel: [ 3838.081097] 37 watchdogd 0

```

然后我们运行编写的读取日志文件程序，输出的结果如下图所示：

```

randomstar@ubuntu:~/Desktop$ g++ user.cpp
randomstar@ubuntu:~/Desktop$ ./a.out
mysyscall:
total page fault:112372 current page fault:127
1    systemd 0
2    kthreadd 0
4    kworker/0:0H 0
6    mm_percpu_wq 0
7    ksoftirqd/0 0
8    rcu_sched 0
9    rcu_bh 0
10   migration/0 0
11   watchdog/0 0
12   cpuhp/0 0
13   cpuhp/1 0
14   watchdog/1 0
15   migration/1 0
16   ksoftirqd/1 0
18   kworker/1:0H 0
19   kdevtmpfs 0
20   netns 0
21   rcu_tasks_kthre 0
22   kauditd 0
24   khungtaskd 0
25   oom_reaper 0
26   writeback 0
27   kcompactd0 0
28   ksmd 0
29   khugepaged 0
30   crypto 0
31   kintegrityd 0
32   kblockd 0
33   ata_sff 0
34   md 0
35   edac-poller 0
36   devfreq_wq 0
37   watchdogd 0
40   kswapd0 0
41   ecryptfs-kthrea 0
83   kthrotld 0
85   acpi_thermal_pm 0
87   scsi_eh_0 0
88   scsi_tmf_0 0
89   scsi_eh_1 0
90   scsi_tmf_1 0
92   ipv6_addrconf 0
102  kstrp 0
119  charger_manager 0
169  mpt_poll_0 0
170  mpt/0 0

```

2.3 结果分析

上述结果表明，我的本次实验完成了所有的实验要求，并且我们发现大部分进程的脏页数都是 0，也有部分进程有较多的脏页，可以看出数据是比较合理的，实验取得了成功。

2.4. 源程序

系统调用部分代码（实验中的部分截图是前几次尝试失败时候的截图，代码可能和最终版本略有区别，但是无伤大雅，所有代码均以最终版本为准）

```
extern unsigned long pfcount;
asm linkage int sys_mysyscall(void)
{
    printk("mysyscall:\ntotal page fault:%d\ncurrent page
fault:%d\n", pfcount, current->pf);
    struct task_struct *p=NULL;
    for(p=&init_task;(p=next_task(p))!=&init_task;) {
        printk("%d\t%s\t%d\n", p->pid, p->comm, p->nr_dirtied);
    }
    printk("end of mysyscall.\n");
    return 0;
}
```

用户态程序 user.cpp，另一个程序 test.c 只有一行即调用系统调用，就不放出来了

```
#include <iostream>
#include <fstream>
#include <string>
```

```

using namespace std;

int main() {
    ifstream fin("/var/log/kern.log", ios::in);
    string s;
    bool can_print = false;
    while (getline(fin, s)) {
        if (s.find("mysyscall") != -1) {
            can_print = true;
        }
        if (s.find("end of") != -1) {
            can_print = false;
        }
        if (can_print) {
            cout << s.substr(46) << endl;
        }
    }
    return 0;
}

```

2.5 回答问题

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?

- 有一定的区别, 这里的缺页次数实际上指的是调用 `do_page_fault()` 的次数, 真实情况中的缺页次数应该不会超过程序中打印出来的缺页次数, 因为一次缺页可能调用多次 `do_page_fault()`

问题 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。

- 有, 还可以通过添加内核模块的方式添加一个系统调用, 因为内核模块也是内核态的程序, 可以调用一些内核中编写好的 api 和库函数, 获取和修改系统中的一些信息, 因此通过内核模块来编写一个系统调用
- 而且通过修改内核直接添加系统调用不仅风险高, 而且消耗的时间比较长 (需要花较长时间编译内核), 不如随时可以装载和卸载的内核模块来的方便。安全, 并且易于调试
- 内核模块添加系统调用的基本步骤有
 - 找系统调用表在内存中的位置
 - 找到一个空闲的系统调用号并修改寄存器的写保护位
 - 实现系统调用函数
 - 编译之后装载内核模块

问题 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。

- 我认为不太安全, 因为我这个实验做了好多次, 碰到过因为修改错一行代码而导致内核编译崩溃、内核换上之后 reboot 时崩溃等情况, 而且内核编译消耗的时间非常长, 需要等待比较久的时间。
- 就算是添加内核模块也存在很多的安全隐患, 比如实验 1 中如果日志输出的格式不规范可能会破坏系统的日志结构, 而实验 2 中修改内核代码和重新编译并更换内核也是充满了各种不确定性因素
- 究其原因还是我们对 Linux 内核缺乏认知, 不了解其总体的架构细节和具体的代码实现, 操作其内核相关的代码就犹如黑盒实验, 不知道下一步会出现什么新的问题, 这种情况下如果贸然开始实验是非常容易失败的, 也很容易踩到 Linux 系统的各种坑, 使得内核崩溃。

三、讨论、心得（20 分）

本次实验虽然看上去操作难度较低,实验步骤相比于实验 1 也更加清晰明确,但其实在真正完成实验的时候我却发现这个实验异常困难,主要碰到过以下几个麻烦:

- Ubuntu 发行版本选错导致换内核的时候失败
- 忘记打补丁导致换内核的时候失败
- 添加和修改内核代码时没有按照要求添加也导致内核编译失败
- 我的电脑本身有点问题导致虚拟机中的 Ubuntu 编译内核时编译到一半宿主机崩溃了,结果导致了编译失败,需要重新编译

此外我还遇到了不少意想不到的错误和问题,总之这次实验对我的心态是一个非常大的挑战,好几次尝试的过程中不仅虚拟机崩溃了,我人也崩溃了,在多次询问老师和同学,尝试了若干次之后,终于在正确的 Ubuntu 和内核版本下添加了正确的代码,并编译成功换上了内核。

本次实验我最大的心得体会就是做实验心态一定要保证好,也希望此类实验能够有更完善的实验指导书,我认为这类和虚拟机版本以及环境高相关度的实验并不适合用作教学,希望后面这些实验会改进吧。