

浙江大学实验报告

课程名称：____操作系统分析及实验____实验类型：____设计性____
实验项目名称：____实验 2 添加系统调用____
学生姓名：____蒋仕彪____专业：____求是科学班 1701____学号：____3170102587____
电子邮件地址：____969519450@qq.com____手机：____18888921530____
实验日期：____2019____年____11____月____28____日

一、实验目的

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、实验器材

安装了 Vmware 的 Windows10 计算机。

虚拟机中的 Linux 为 Ubuntu-64，自带的内核版本为 4.15.0，本次实验将安装 4.6.0

四、操作方法和实验步骤

4.1 下载和部署内核源代码

- 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
```



- 建立符号链接：

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

- 查看当前内核版本号：

```
linux-4.6/virt/lib/
linux-4.6/virt/lib/Kconfig
linux-4.6/virt/lib/Makefile
linux-4.6/virt/lib/irqbypass.c
jsb@ubuntu:~$ uname -r
4.15.0-70-generic
jsb@ubuntu:~$
```

- 安装必要的库：

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

- 配置内核（使用缺省值）：

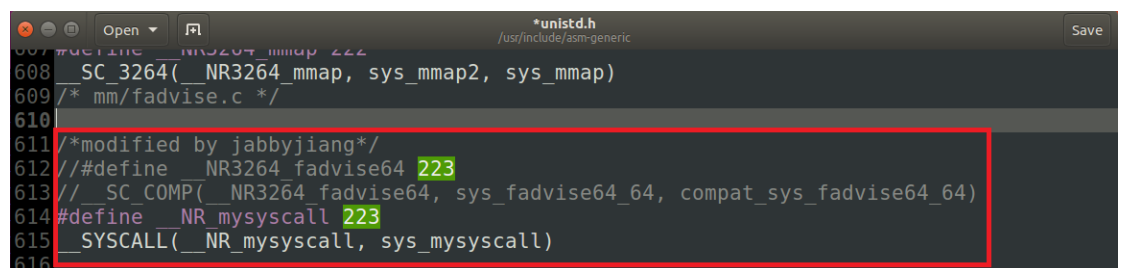
```
sudo make menuconfig
```

4.2 添加系统调用号

- 打开头文件目录（在/usr 里必须用 root 权限才能编辑）：

```
jsb@ubuntu:/usr/include$ sudo gedit /usr/include/asm-generic/unistd.h
```

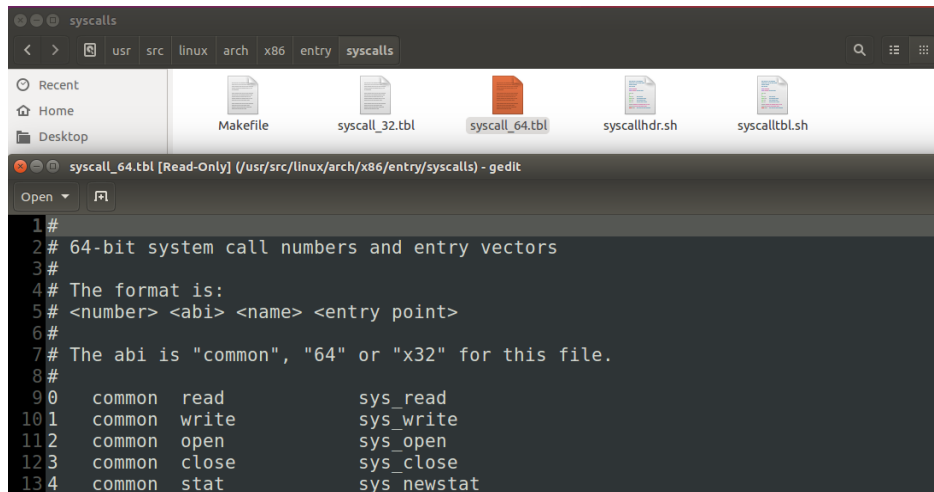
- 查找定义 223 号的行，作如下修改：



- 在内核源码 include/uapi/asm-generic/unistd.h 中做同样的修改。

4.3 在系统调用表中添加或修改相应表项

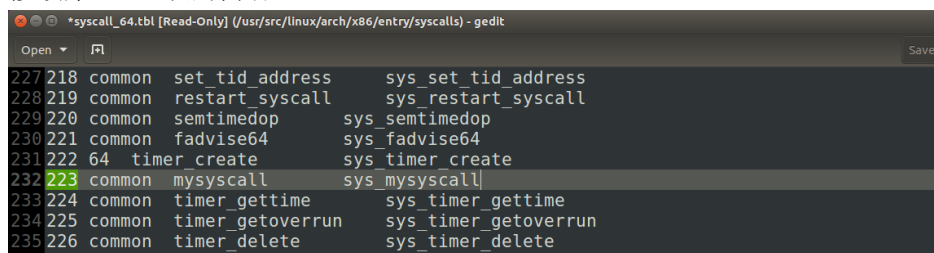
- 先找到系统调用表，进行一番研究：



```
syscalls
< > usr src linux arch x86 entry syscalls
Recent
Home
Desktop
Makefile
syscall_32.tbl
syscall_64.tbl
syscallhdr.sh
syscalltbl.sh

syscall_64.tbl [Read-Only] (/usr/src/linux/arch/x86/entry/syscalls) - gedit
Open
1#
2# 64-bit system call numbers and entry vectors
3#
4# The format is:
5# <number> <abi> <name> <entry point>
6#
7# The abi is "common", "64" or "x32" for this file.
8#
9 0 common read sys_read
10 1 common write sys_write
11 2 common open sys_open
12 3 common close sys_close
13 4 common stat sys_newstat
```

- 修改第 223 项的内容。

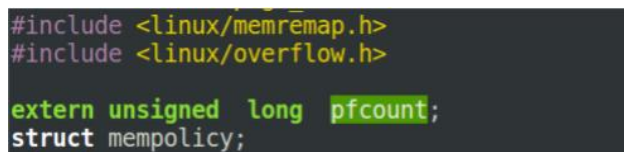


```
*syscall_64.tbl [Read-Only] (/usr/src/linux/arch/x86/entry/syscalls) - gedit
Open Save
227 218 common set_tid_address sys_set_tid_address
228 219 common restart_syscall sys_restart_syscall
229 220 common semtimedop sys_semtimedop
230 221 common fadvise64 sys_fadvise64
231 222 64 timer_create sys_timer_create
232 223 common msyscall sys_msyscall
233 224 common timer_gettime sys_timer_gettime
234 225 common timer_getoverrun sys_timer_getoverrun
235 226 common timer_delete sys_timer_delete
```

可能是因为我用的是 64 位 Linux，现象和实验指导上说的不一样：233 项并不是空的，而是一个和时间有关的函数。因为这不是关键的系统调用，我就强行按照指导上的步骤，把 233 号系统调用替换掉了。

4.4 修改统计系统缺页次数和进程缺页次数的内核代码

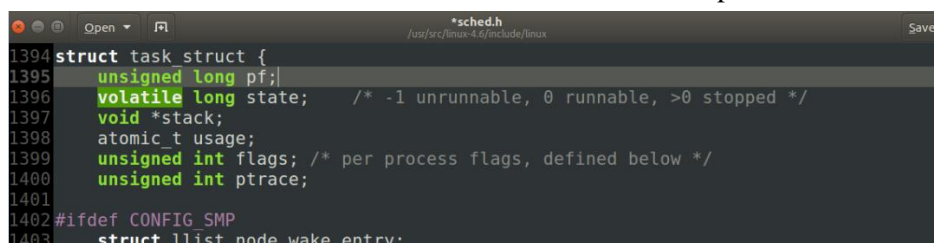
- 先在 include/linux/mm.h 文件中声明变量 pfcoun



```
#include <linux/memremap.h>
#include <linux/overflow.h>

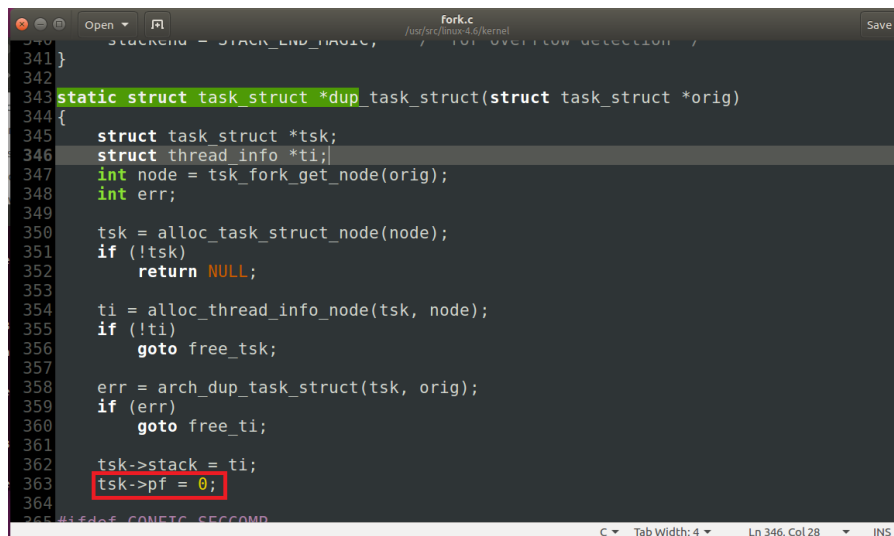
extern unsigned long pfcoun;
struct mempolicy;
```

- 在 include/linux/sched.h 文件中的 task_struct 结构中添加 pf 字段：



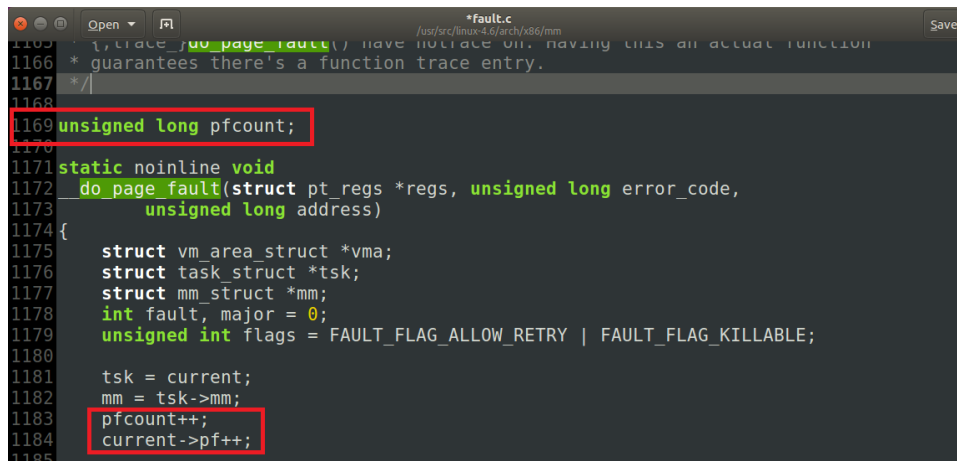
```
*sched.h
/usr/src/linux-4.6/include/linux
Open Save
1394 struct task_struct {
1395     unsigned long pf;
1396     volatile long state; /* -1 unrunnable, 0 runnable, >0 stopped */
1397     void *stack;
1398     atomic_t usage;
1399     unsigned int flags; /* per process flags, defined below */
1400     unsigned int ptrace;
1401
1402 #ifdef CONFIG_SMP
1403     struct llist node wake entry;
```

- 修改 kernel/fork.c 文件中的 dup_task_struct() 函数:



```
341}
342
343static struct task_struct *dup_task_struct(struct task_struct *orig)
344{
345    struct task_struct *tsk;
346    struct thread_info *ti;
347    int node = tsk_fork_get_node(orig);
348    int err;
349
350    tsk = alloc_task_struct_node(node);
351    if (!tsk)
352        return NULL;
353
354    ti = alloc_thread_info_node(tsk, node);
355    if (!ti)
356        goto free_tsk;
357
358    err = arch_dup_task_struct(tsk, orig);
359    if (err)
360        goto free_ti;
361
362    tsk->stack = ti;
363    tsk->pf = 0;
364
365    #ifdef CONFIG_SECCOMP
```

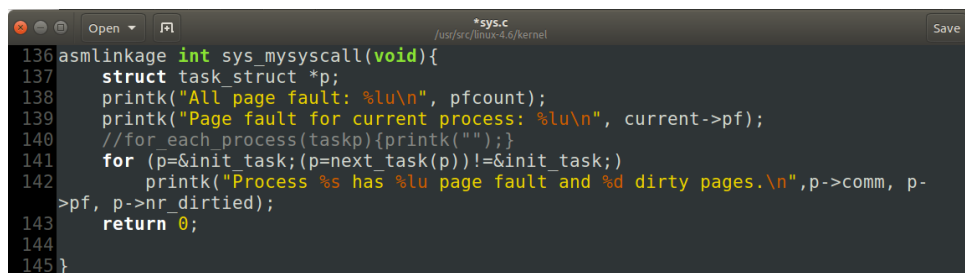
- 在 arch/x86/mm/fault.c 文件中定义变量 pfcoun; 并修改 arch/x86/mm/fault.c 中 do_page_fault() 函数。每次产生缺页中断, do_page_fault() 函数会被调用, pfcoun 变量值递增 1 和 current->pf 值都递增 1, 分别记录缺页情况。



```
1165 * trace, do_page_fault() have notice on, having this an actual function
1166 * guarantees there's a function trace entry.
1167 */
1168
1169 unsigned long pfcoun;
1170
1171 static noline void
1172 do_page_fault(struct pt_regs *regs, unsigned long error_code,
1173               unsigned long address)
1174 {
1175     struct vm_area_struct *vma;
1176     struct task_struct *tsk;
1177     struct mm_struct *mm;
1178     int fault, major = 0;
1179     unsigned int flags = FAULT_FLAG_ALLOW_RETRY | FAULT_FLAG_KILLABLE;
1180
1181     tsk = current;
1182     mm = tsk->mm;
1183     pfcoun++;
1184     current->pf++;
1185 }
```

4.5 sys_mysyscall 的实现

- 把这段程序添加在 kernel/sys.c 里面, 将各个进程的缺页次数打印到系统日志里:



```
136 asmlinkage int sys_mysyscall(void){
137     struct task_struct *p;
138     printk("All page fault: %lu\n", pfcoun);
139     printk("Page fault for current process: %lu\n", current->pf);
140     //for_each_process(taskp){printk("");}
141     for (p=&init_task;(p=next_task(p))!=&init_task;)
142         printk("Process %s has %lu page fault and %d dirty pages.\n",p->comm, p->pf, p->nr_dirtied);
143     return 0;
144 }
145 }
```

其中, task_struct 是 Linux 用来描述进程信息的结构。中间那个循环用来枚举当前所有的进程。P->comm 是进程名字, p->nr_dirtied 是脏页面数量, p->pf 即为我们定义的缺页的次数。用 printk 来输出。

4.6 编译和安装内核

- 执行以下编译内核的命令：

```
sudo make bzImage -j2
```

- 发现报错：

```
jsb@ubuntu: /usr/src/linux
HOSTCC scripts/mod/file2alias.o
HOSTCC scripts/recordmcount
HOSTCC scripts/sortextable
HOSTCC scripts/mod/sumversion.o
HOSTCC scripts/asn1_compiler
HOSTCC arch/x86/tools/relocs_32.o
HOSTLD scripts/mod/modpost
HOSTCC scripts/sign-file
HOSTCC scripts/extract-cert
scripts/sign-file.c:25:30: fatal error: openssl/opensslv.h: No such file or directory
compilation terminated.
scripts/Makefile.host:91: recipe for target 'scripts/sign-file' failed
make[1]: *** [scripts/sign-file] Error 1
make[1]: *** Waiting for unfinished jobs....
HOSTCC arch/x86/tools/relocs_64.o
scripts/extract-cert.c:21:25: fatal error: openssl/bio.h: No such file or directory
compilation terminated.
scripts/Makefile.host:91: recipe for target 'scripts/extract-cert' failed
make[1]: *** [scripts/extract-cert] Error 1
HOSTCC arch/x86/tools/relocs_common.o
Makefile:552: recipe for target 'scripts' failed
make: *** [scripts] Error 2
```

- 排查了一下原因，发现有一个库没有装上。安装一下即可。

```
jsb@ubuntu: /usr/src/linux$ sudo apt install libelf-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  libelf-dev
0 upgraded, 1 newly installed, 0 to remove and 95 not upgraded.
Need to get 54.5 kB of archives.
After this operation, 359 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu xenial-updates/main amd64 libelf-dev a
md64 0.165-3ubuntu1.2 [54.5 kB]
Fetched 54.5 kB in 1s (28.6 kB/s)
Selecting previously unselected package libelf-dev:amd64.
(Reading database ... 252060 files and directories currently installed.)
Preparing to unpack .../libelf-dev_0.165-3ubuntu1.2_amd64.deb ...
Unpacking libelf-dev:amd64 (0.165-3ubuntu1.2) ...
Setting up libelf-dev:amd64 (0.165-3ubuntu1.2) ...
```

- 执行以下模块编译和安装的命令：

```
sudo make modules -j2
sudo make modules_install
```

- 到 `modules_install` 这一步时，又发现报错：

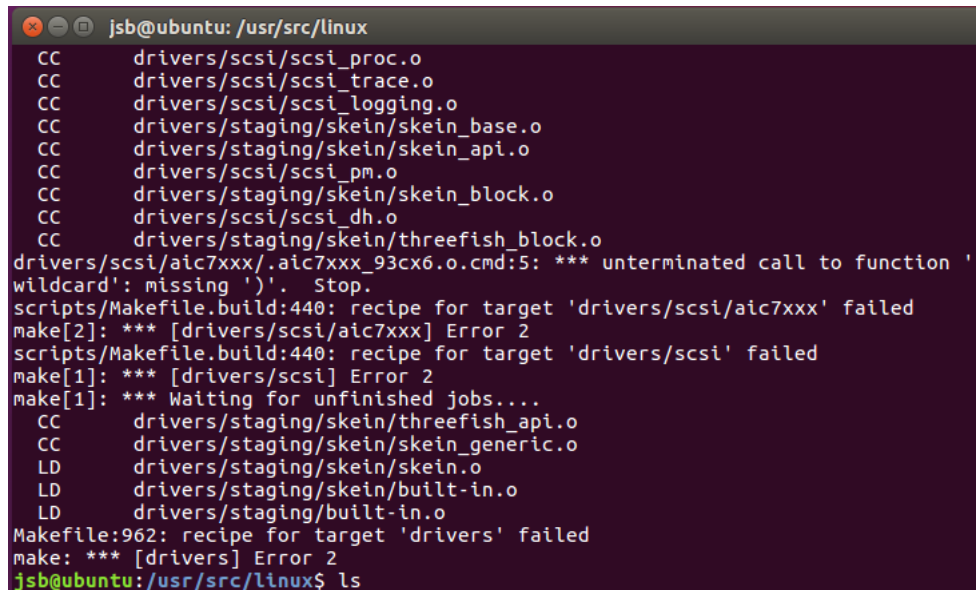
```
jsb@ubuntu: /usr/src/linux$ sudo make modules_install
cp: cannot stat './modules.order': No such file or directory
Makefile:1164: recipe for target '_modinst_' failed
make: *** [_modinst_] Error 1
```

- 互联网上没有发现类似的问题，多次尝试依然是这个问题。自己分析了一下，可能是之前的模块安装的那句话没有执行好，于是重新执行 `make modules -j2`。

- 重新执行以下命令：

```
sudo make modules -j2
```

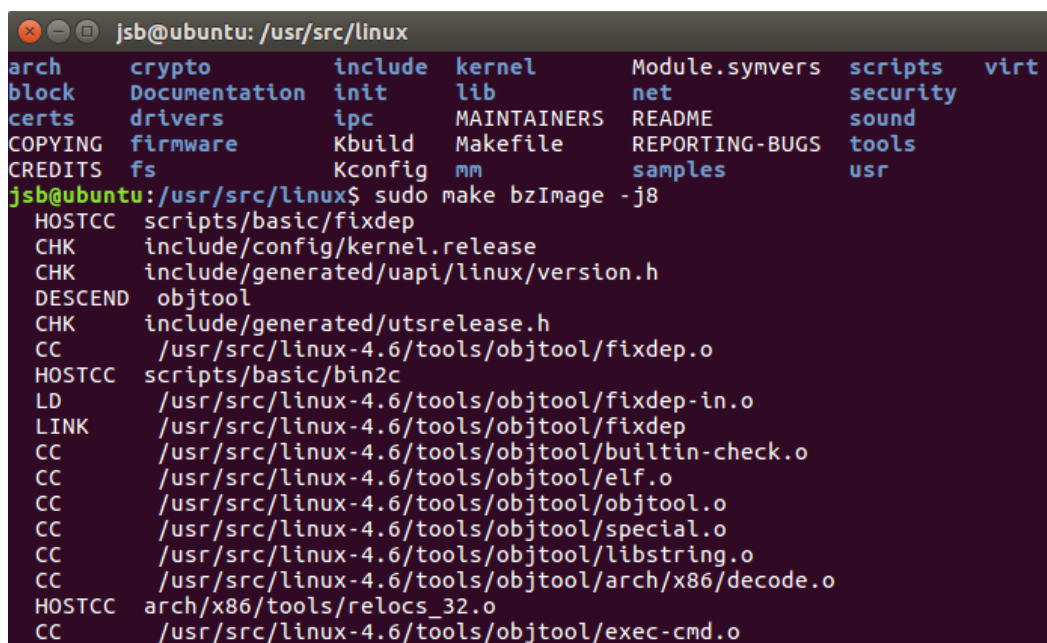
- 然后报了一个更奇怪的错误：



```
jsb@ubuntu: /usr/src/linux
CC      drivers/scsi/scsi_proc.o
CC      drivers/scsi/scsi_trace.o
CC      drivers/scsi/scsi_logging.o
CC      drivers/staging/skein/skein_base.o
CC      drivers/staging/skein/skein_api.o
CC      drivers/scsi/scsi_pm.o
CC      drivers/staging/skein/skein_block.o
CC      drivers/scsi/scsi_dh.o
CC      drivers/staging/skein/threefish_block.o
drivers/scsi/aic7xxx/.aic7xxx_93cx6.o.cmd:5: *** unterminated call to function '
wildcard': missing ')'. Stop.
scripts/Makefile.build:440: recipe for target 'drivers/scsi/aic7xxx' failed
make[2]: *** [drivers/scsi/aic7xxx] Error 2
scripts/Makefile.build:440: recipe for target 'drivers/scsi' failed
make[1]: *** [drivers/scsi] Error 2
make[1]: *** Waiting for unfinished jobs....
CC      drivers/staging/skein/threefish_api.o
CC      drivers/staging/skein/skein_generic.o
LD      drivers/staging/skein/skein.o
LD      drivers/staging/skein/built-in.o
LD      drivers/staging/built-in.o
Makefile:962: recipe for target 'drivers' failed
make: *** [drivers] Error 2
jsb@ubuntu: /usr/src/linux$ ls
```

怀疑是：二次编译模块时，之前残留的 .o 文件会造成影响。于是重头编译内核。

- 第二次编译内核时，采用了更多线程的编译（-j8），速度果然快了很多：



```
jsb@ubuntu: /usr/src/linux
arch      crypto      include  kernel      Module.symvers  scripts      virt
block     Documentation  init     lib          net             security
certs     drivers        ipc      MAINTAINERS  README          sound
COPYING   firmware      Kbuild   Makefile     REPORTING-BUGS  tools
CREDITS   fs             Kconfig  mm           samples         usr
jsb@ubuntu: /usr/src/linux$ sudo make bzImage -j8
HOSTCC    scripts/basic/fixdep
CHK        include/config/kernel.release
CHK        include/generated/uapi/linux/version.h
DESCEND    objtool
CHK        include/generated/utsrelease.h
CC         /usr/src/linux-4.6/tools/objtool/fixdep.o
HOSTCC    scripts/basic/bin2c
LD         /usr/src/linux-4.6/tools/objtool/fixdep-in.o
LINK       /usr/src/linux-4.6/tools/objtool/fixdep
CC         /usr/src/linux-4.6/tools/objtool/builtin-check.o
CC         /usr/src/linux-4.6/tools/objtool/elf.o
CC         /usr/src/linux-4.6/tools/objtool/objtool.o
CC         /usr/src/linux-4.6/tools/objtool/special.o
CC         /usr/src/linux-4.6/tools/objtool/libstring.o
CC         /usr/src/linux-4.6/tools/objtool/arch/x86/decode.o
HOSTCC    arch/x86/tools/relocs_32.o
CC         /usr/src/linux-4.6/tools/objtool/exec-cmd.o
```

- 重复了一遍流程，上述问题不再发生。于是顺次执行安装内核和配置 grub 的命令：

```
sudo make install
sudo mkinitramfs 4.6.0 -o /boot/initrd.img-4.6.0
sudo update-grub2
```


- 成功安装和配置完成

```
jsb@ubuntu: /usr/src/linux
INSTALL /lib/firmware/edgeport/down.fw
INSTALL /lib/firmware/edgeport/down2.fw
INSTALL /lib/firmware/edgeport/down3.bin
INSTALL /lib/firmware/whiteheat_loader.fw
INSTALL /lib/firmware/whiteheat.fw
INSTALL /lib/firmware/keyspan_pda/keyspan_pda.fw
INSTALL /lib/firmware/keyspan_pda/xircom_pgs.fw
INSTALL /lib/firmware/cpia2/stv0672_vp4.bin
INSTALL /lib/firmware/yan/1200.bin
INSTALL /lib/firmware/yan/9600.bin
DEPMOD 4.6.0
jsb@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/vmlinuz-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 set is no longer sup
ported.
Found linux image: /boot/vmlinuz-4.15.0-70-generic
Found initrd image: /boot/initrd.img-4.15.0-70-generic
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-4.6.0
Found initrd image: /boot/initrd.img-4.6.0
Found memtest86+ image: /boot/memtest86+.elf
Found memtest86+ image: /boot/memtest86+.bin
done
jsb@ubuntu: /usr/src/linux$ sudo mkinitramfs 4.6.0 -o /boot/initrd.img-4.6.0
jsb@ubuntu: /usr/src/linux$ sudo update-grub2
grub-mkconfig: You must run this as root
jsb@ubuntu: /usr/src/linux$ sudo update-grub2
```

重启即可。

4.7 重启内核并完成用户态程序

- 内核安装完后我发现：无论怎么配置 **grub**，重启后都是原来的内核。该问题搁置了很久都没有解决，网上的攻略也没有用。自己的猜测是：**4.6.0** 是低版本内核，重启的时候可能默认是开更高版本的。群里有个同学说，reboot 后狂按 **Esc** 键，点击 **Advanced Options** 后，就可以人工选择需要启动的内核版本了。
- 用户态程序的编写很简单，我们只需直接调用这个系统调用号：

```
#include <unistd.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#define __NR_mysyscall 223
int main(){
    syscall(__NR_mysyscall);
}
```

- 编译这个 **go.c** 后，用 **dmesg** 打开系统日志，即可看到最终结果。

```
jsb@ubuntu: ~/Desktop/syscall
jsb@ubuntu:~$ uname -r
4.6.0
jsb@ubuntu:~$ cd Desktop/syscall/
jsb@ubuntu:~/Desktop/syscall$ gcc go.c -o go
jsb@ubuntu:~/Desktop/syscall$ ./go
jsb@ubuntu:~/Desktop/syscall$ dmesg
```

五、实验结果和分析

5.1 用户态程序的测试结果

```
69.441120] All page fault: 455684
69.441122] Page fault for current process: 70
69.441125] Process systemd has 8535 page fault and 0 dirty pages.
69.441126] Process kthreadd has 0 page fault and 0 dirty pages.
69.441127] Process ksoftirqd/0 has 0 page fault and 0 dirty pages.
69.441128] Process kworker/0:0 has 0 page fault and 0 dirty pages.
69.441129] Process kworker/0:0H has 0 page fault and 0 dirty pages.

69.441396] Process unity-settings- has 2436 page fault and 0 dirty pages.
69.441397] Process gnome-session-b has 1734 page fault and 2 dirty pages.
69.441398] Process unity-panel-ser has 2028 page fault and 0 dirty pages.
69.441400] Process dconf-service has 396 page fault and 171 dirty pages.
69.441401] Process indicator-messa has 385 page fault and 0 dirty pages.
69.441402] Process indicator-bluet has 295 page fault and 0 dirty pages.
69.441402] Process indicator-power has 560 page fault and 0 dirty pages.
69.441404] Process indicator-datet has 1035 page fault and 0 dirty pages.
69.441404] Process indicator-keybo has 1497 page fault and 0 dirty pages.
69.441405] Process indicator-sound has 686 page fault and 0 dirty pages.
69.441406] Process indicator-print has 1585 page fault and 0 dirty pages.
69.441407] Process indicator-sessi has 388 page fault and 0 dirty pages.
69.441408] Process indicator-appli has 738 page fault and 0 dirty pages.
69.441409] Process pulseaudio has 849 page fault and 2 dirty pages.
```

该程序如预期输出每个进程的缺页和脏页结果。

5.2 内核编译的研究

我在编译内核的时候发现，`sudo make bzImage -jk` 中的 `k` 的确起到了多线程的效果。

我第一次编译的时候，用到了 `-j2`，它基本上是按照字典序去编译文件：显示在控制台上的 `.o` 文件是按字典序排列的。

我第二次编译的时候，用到了 `-j8`，发现导出的 `.o` 不再严格按照字典序排列（尽管总体是按字典序排列的）。以下是截到的一张图。

```
jsb@ubuntu: /usr/src/linux
CC [M] sound/pci/ice1712/vt1720_mobo.o
LD [M] net/bridge/br_netfilter.o
CC [M] net/caif/caif_dev.o
CC [M] fs/xfs/libxfs/xfs_bmap.o
CC [M] drivers/gpio/gpio-twl4030.o
CC [M] sound/pci/ice1712/pontis.o
CC [M] drivers/gpu/drm/drm_memory.o
CC [M] fs/ocfs2/cluster/netdebug.o
CC [M] drivers/gpio/gpio-twl6040.o
CC [M] net/caif/cfcnf.o
CC [M] drivers/gpu/drm/drm_drv.o
CC [M] drivers/gpu/drm/drm_vm.o
CC [M] sound/pci/ice1712/prodigy192.o
^A CC [M] drivers/gpio/gpio-ucb1400.o
LD [M] fs/ocfs2/cluster/ocfs2_nodemanager.o
CC [M] fs/ocfs2/dlm/dlmdomain.o
CC [M] net/caif/cfmuxl.o
CC [M] drivers/gpio/gpio-viperboard.o
CC [M] drivers/gpu/drm/drm_scatter.o
CC [M] drivers/gpu/drm/drm_pci.o
CC [M] sound/pci/ice1712/prodigy_hifi.o
CC [M] sound/pci/hda/patch_realtek.o
CC [M] drivers/gpio/gpio-vx855.o
```


六、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？

➤ 我间隔随机地运行了六次，依次记录了系统缺页次数和当前进程缺页次数。

次数	当前进程缺页次数	系统缺页总次数
1	70	455684
2	72	502709
3	70	503133
4	70	658383
5	71	667925
6	70	683838

随着次数的递增，当前进程缺页次数稳定，系统缺页总次数不断增加。

- 不是。`do_page_fault` 函数并不是只有发现缺页的时候会调用，因此我们的 `pfcount` 实际要偏大。
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。我们不一定要内核里写某个系统调用函数。可以通过改变某一个系统调用号对应的服务程序为我们自己的函数模块，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。

➤ 我觉得显然是不安全的。即使是我们这种实验性质的修改，如果一旦操作不慎（比如覆盖了某些系统调用），整个 Linux 就会运行不正常。

七、讨论心得

1. 本次实验相比于实验一，思维难度降低了很多，重点是动手操作。经过这个实验，我对 Linux 的命令行操作又熟悉了不上。
2. 内核的编译实在是太头疼了（主要是我对 Linux 的一些报错毫无头绪）。我总共编译了四遍才过。第一遍是学期初，照着内核编译指导玩了一遍，成功安装完了但是打开的时候依然是原来的内核，搞了很久也没弄好，甚至把整个 Linux 玩得开不了机了。后三遍就是在做实验二的时候编译安装的，中间陆续遇到各种错误。有些错误可以在 StackOverflow 上找到，如果找不到也看不懂的话就只能重新编译了。
3. 编译内核时遇到过很多问题：①权限问题，前面加 `sudo` 来启动 root 权限；②有些相关的库没有安装，用 `apt-install` 安装一下；③没有按流程走（跳过了某一步），接下来的步骤自然会报错（比如没有编译过模块，之后安装模块时就会报错）；④还有一个一直没弄懂的问题（写在之前的实验流程里了），重做了一次后解决了。
4. `make bzImage -j2` 其实是可以加速的。把后面的常数有选择地开大，即可用更多线程去编译。编译得多了我还发现，编译内核是按字典序的顺序编译的。所以在 `-j2` 的时候，控制台上输出的 `.o` 文件是按照字典序出现的。然而，当开了 `-j8` 的时候，因为并行度增加了，`.o` 文件呈现相对乱序的样子出现在控制台上。
5. “纸上得来终觉浅”，光看着实验要求里那几行看似简单的命令一定是没有收获的；必须自己实践过才会遇到很多问题，才能出真知。