操作系统课程设计实验报告

实验一：Linux 内核编译及添加系统调用

作者： 丁 乃 文
学号：15198704

2018-4-14

操作系统课程设计实验报告

实验一：Linux 内核编译及添加系统调用

项目设计方案：

总体设计思路：

系统调用的本质是调用内核函数，以内核态运行程序。为了在内核态下运行，本实验针对 Linux 的内核进行修改，增加自定义系统调用函数实现用户态程序对任意进程的 nice 值进行修改或者读取来进行测试。

主要函数的接口设计：

核心态程序：

```
SYSCALL_DEFINE3(mysetnice, pid_t, pid, int, flag, int, nicevalue)
```

其中 pid 为选择进程的进程标识符

flag 是操作符，设计为 0 时读取 nice 值，设计为 1 时进行修改操作。

nice 值为一返回变量，返回内核程序读取到的进程 nice 值

共计三个变量

使用 SYSCALL_DEFINE 声明添加到~/kernel/sys.c 文件中

项目实现过程：

1、准备

本次实验使用的环境为装载在 Vmware Workstation Pro 14 中的 Ubuntu 17.10 镜像

镜像下载地址：http://mirrors.zju.edu.cn/ubuntu-releases/17.10/

为了编译执行顺利，我给虚拟机分配了 8GB RAM 和 100GB 虚拟硬盘空间，同时 vCPU 分配了 8 个核心，以便 make -jn 命令的使用

安装 ubuntu 的时候，一定要注意查看/boot 分区是否足够，如果不是工作环境最好直接将/boot 分区挂载在根目录下，否则编译的时候产生的文件会塞满/boot 分区导致编译安装失败。

2、初次内核编译

为了在添加系统调用后可能发生报错的情况下理清是内核编译问题还是程序编写问题，建议先完成一次纯净内核编译工作。

我们从 https://www.kernel.org 网站下载需要的 Linux 操作系统内核，我选择下载的 Linux 内核版本号是 4.15.10，对于一般情况来讲，内核版本越低，完成编译所需的工作量越小，因此可以选择低一些的版本号下载以方便配置较低的机器顺利完成编译任务。

编译工作需要在高级权限下运行，因此先使用 su 命令或者没有设置 root 密码的话使用 sudo -i 切换到 root 账户进行接下来的操作。

使用 cd 命令切换到/usr/src 目录

使用 wget 命令下载内核

wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.15.10.tar.xz

等待下载完成，下载完成后对内核进行解压缩并将工作目录切换到解压出来文件夹下

xz -d linux-4.15.10.tar.xz && tar -xvf linux-4.15.10.tar && cd linux-4.15.10

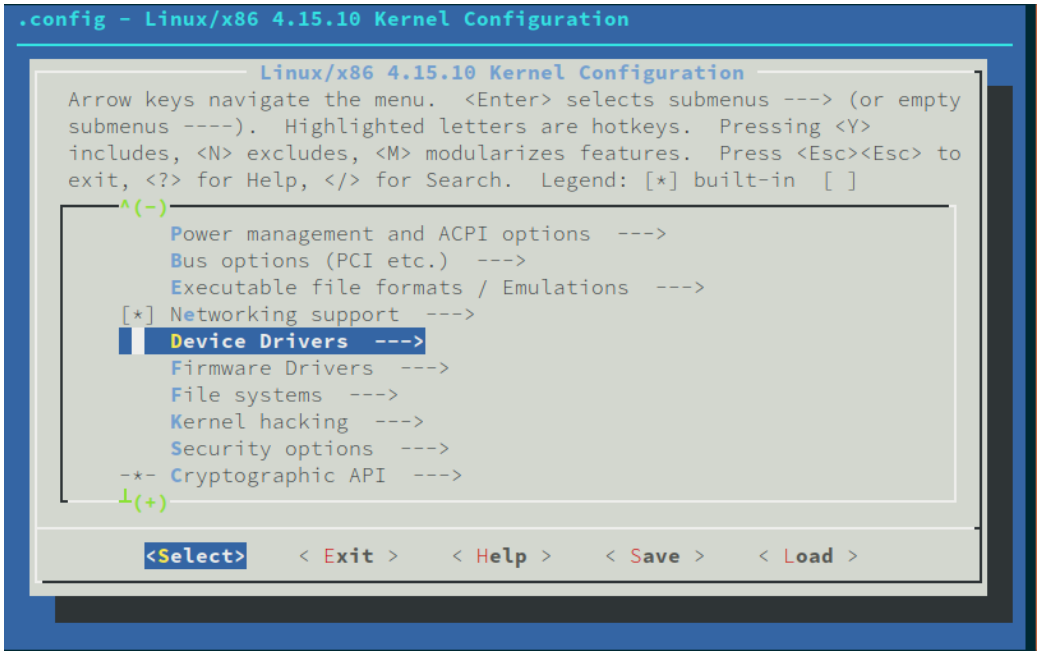
进入目录之后，因为这是刚下载的没有经过编译的内核包，因此可以不用执行 make clean 命令。

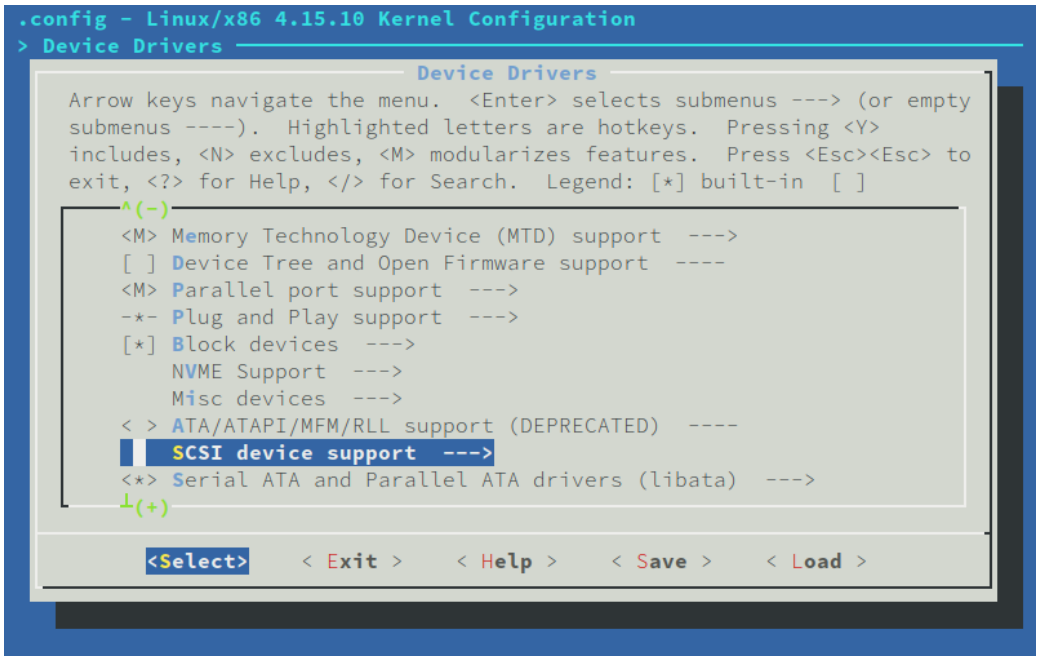
为了后续的工作方便，我们使用 aptitude 包管理器安装一下需要的软件和工具

执行命令 apt install make libncurses5-dev libssl-dev vim gcc -y

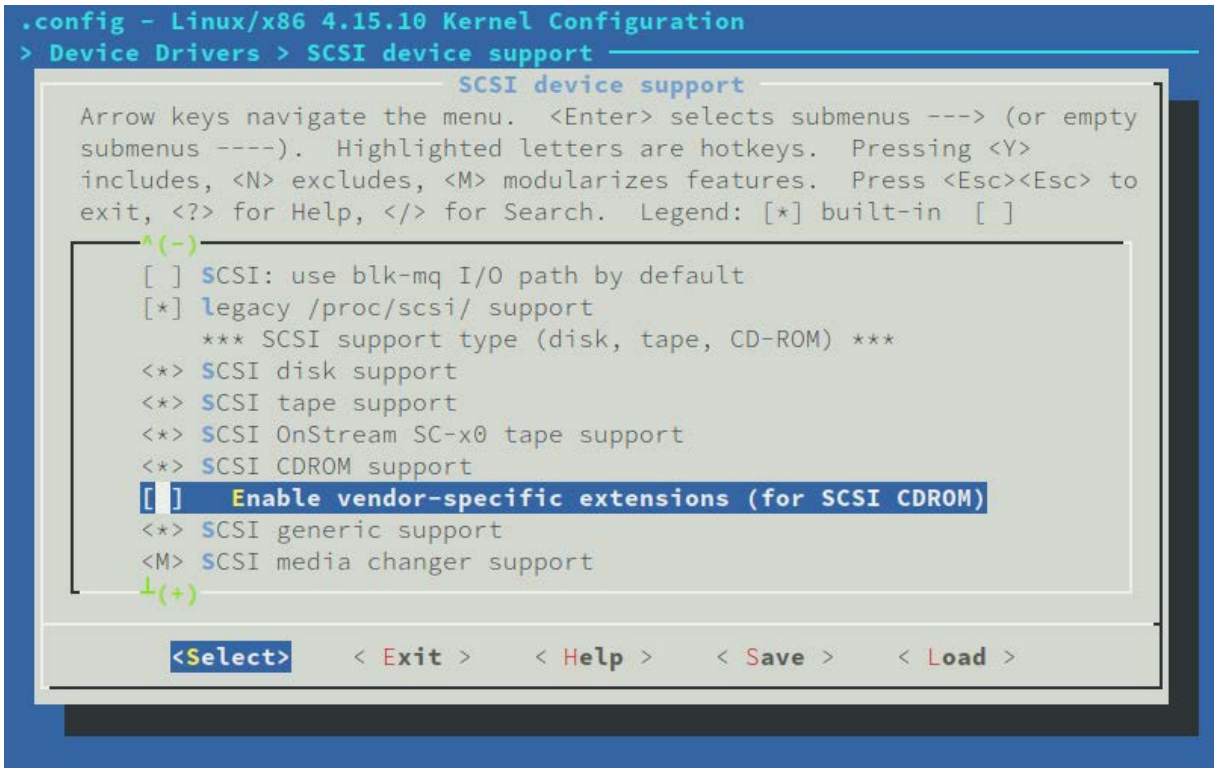
然后配置内核选项

输入命令 make menuconfig 进入 GUI 配置界面，在这里我们需要将关于 SCSI 的内容做一些更改，配置见下图所示





将以下内容打上*



保存配置文件，我们就可以开始启动映像的编译了

运行指令 `make bzImage -j8`

等待编译完成，终端提示 Done，使用了-jn 指令多核编译后，速度很快，只花了 5 分钟就编译完成了

然后继续进行模块的编译

运行指令 `make modules -j8`,这里大概花了 15 分钟左右

这里需要我们注意的一点是，如果采用多核编译，有些核心的输出是不会输出在控制台的，虽然我也不知道它们输出在了哪里 TAT

同样要等待终端提示 Done 后安装模块

运行指令 `make modules_install`

完成模块安装后即可安装核心了

运行指令 `make install`

```
1: top, zsh
File descriptor 135 (/usr/share/fonts/truetype/liberation/LiberationMono-Bold.ttf) leaked on l
vs invocation. Parent PID 58398: /bin/sh
File descriptor 136 (/usr/share/fonts/truetype/liberation/LiberationSerif-Bold.ttf) leaked on
lvs invocation. Parent PID 58398: /bin/sh
File descriptor 137 (/usr/share/fonts/truetype/liberation/LiberationSerif-Bold.ttf) leaked on
lvs invocation. Parent PID 58398: /bin/sh
File descriptor 138 (/dev/shm/.org.chromium.Chromium.YwWikr (deleted)) leaked on lvs invocatio
n. Parent PID 58398: /bin/sh
File descriptor 139 (/dev/shm/.org.chromium.Chromium.nmTuTX (deleted)) leaked on lvs invocatio
n. Parent PID 58398: /bin/sh
File descriptor 142 (/usr/share/fonts/truetype/liberation/LiberationSans-Bold.ttf) leaked on l
vs invocation. Parent PID 58398: /bin/sh
File descriptor 143 (/usr/share/fonts/truetype/liberation/LiberationSans-Bold.ttf) leaked on l
vs invocation. Parent PID 58398: /bin/sh
File descriptor 156 (/usr/share/fonts/opentype/noto/NotoSansCJK-Bold.ttc) leaked on lvs invoca
tion. Parent PID 58398: /bin/sh
File descriptor 157 (/usr/share/fonts/opentype/noto/NotoSansCJK-Bold.ttc) leaked on lvs invoca
tion. Parent PID 58398: /bin/sh
File descriptor 158 (/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf) leaked on lvs invocation
. Parent PID 58398: /bin/sh
done
christianswift-virtual-machine#
```

这里应该会出现一条 update-initramfs: Generating /boot/initrd.img-4.15.10

如果没有的话可以手动运行指令

mkinitramfs 4.15.10 -o /boot/initrd-4.15.10.img

完成映像安装之后让系统自动配置 grub

运行指令 update-grub2

配置完成之后执行 reboot 重启

不出意外的话，重新启动的系统在终端运行 uname -a 应当提示以下内容

```
christianswift-virtual-machine# uname -a
Linux christianswift-virtual-machine 4.15.10 #6 SMP Sat Apr 14 17:57:45 CST 2018 x86_64 x86_64
 x86_64 GNU/Linux
```

这样我们可以保证内核的正确编译不出问题了。

3、添加一个系统调用

首先查询一下当前版本 Linux 内核的系统调用号，使用命令 cat /usr/src/linux-4.15.10/arch/x86/entry/syscalls/syscall_64.tbl 输出本内核的系统调用号

我们这里使用的操作系统是 amd64 的处理器，所以查找调用号时应当关注的是 x64 的系统调用号，不必关注 32 位系统的系统调用号

以下是该版本的系统调用情况

```
329 common  pkey_mprotect       sys_pkey_mprotect
330 common  pkey_alloc      sys_pkey_alloc
331 common  pkey_free       sys_pkey_free
332 common  statx           sys_statx


#
# 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
513 x32 rt_sigreturn        sys32_x32_rt_sigreturn
```

我们来添加一个系统调用号

将这个系统调用表添加一条

333　64 mysetnice　sys_mysetnice

在 x32 系统调用号之上

使用 vim 编辑器打开/usr/src/linux-4.15.10/include/linux/syscalls.h 来设置系统调用号

输入:$跳转到文件末尾，进入插入模式

在文件末尾添加

asmlinkage long sys_mysetnice(pid_t pid, int flag, int nicevalue);

然后编写服务例程，以下为程序示例：

```
2199  SYSCALL_DEFINE3(mysetnice, pid_t, pid, int, flag, int, nicevalue){
2200      int error = 0;
2201      struct task_struct *p;
2202      for(p = &init_task;(p = next_task(p)) != &init_task;){
2203          if(p->pid == pid){
2204              if(flag == 0){
2205                  printk("this process's nice value is %d\n",task_nice(p));
2206                  printk("this process's piro value is %d\n",task_prio(p));
2207              } else if(flag == 1){
2208                  set_user_nice(p,nicevalue);
2209                  printk("this process's nice now changed to %d\n",task_nice(p));
2210                  printk("this process's piro value is %d\n",task_prio(p));
2211              } else {
2212                  error = -EFAULT;
2213              }
2214          }
2215      }
2216      return error;
2217  }
```

（这里要注意一点，printk 内容一定要加\n，否则 dmesg 不会输出记录，要等下一条才会输出）

然后进行重新编译，编译时使用 make clean 指令清楚编译历史文件。

4、编写用户态程序和测试

重启系统之后，打开文本编辑器，编写用户态程序，以下为我的程序示例

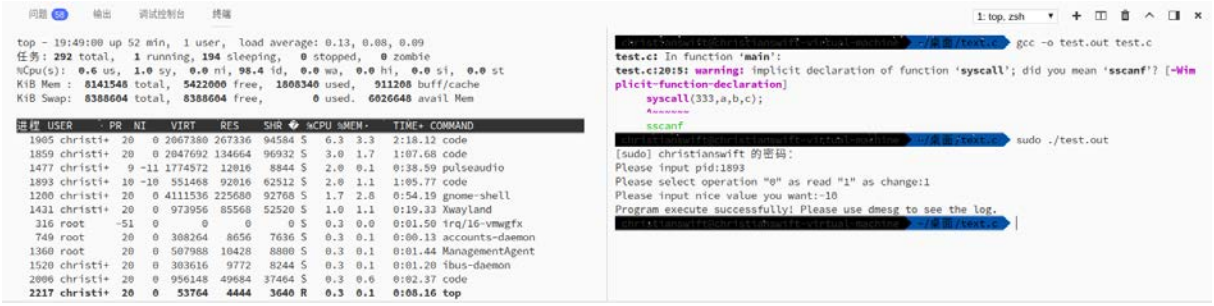
test.c　sys.c

```
1   #include<linux/unistd.h>
2   #include<sys/syscall.h>
3   #include <stdio.h>
4   #include <stdlib.h>
5   //#define __NR_mysetnice 333
6   int main()
7   {
8       int a,b,c;
9       printf("Please input pid:");
10      scanf("%d",&a);
11      printf("Please select operation \"0\" as read \"1\" as change:");
12      scanf("%d",&b);
13      if(b==0){
14          c=0;
15      }
16      else{
17          printf("Please input nice value you want:");
18          scanf("%d",&c);
19      }
20      syscall(333,a,b,c);
21      printf("Program execute successfully! Please use dmesg to see the log.\n");
22      return 0;
23  }
```

使用 gcc -o test.out test.c 编译程序，然后./test.out 执行用户态程序

使用 top 指令随意找一个进程，这里用 vscode 的进程做一下示范

输入进程号将其 nice 改为-10，可以看到进程的 nice 值已经修改成功了



然后运行一下 dmesg -T 查看内核输出

```
[Sat Apr 14 18:56:45 2018] IPv6: ADDRCONF(NETDEV_UP): ens33: link is not ready
[Sat Apr 14 18:56:45 2018] e1000: ens33 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: No
ne
[Sat Apr 14 18:56:45 2018] IPv6: ADDRCONF(NETDEV_CHANGE): ens33: link becomes ready
[Sat Apr 14 18:56:50 2018] NET: Registered protocol family 40
[Sat Apr 14 18:57:04 2018] usb 1-1: reset high-speed USB device number 2 using ehci-pci
[Sat Apr 14 18:57:05 2018] usb 1-1: current rate 0 is different from the runtime rate 48000
[Sat Apr 14 18:57:05 2018] usb 1-1: current rate 0 is different from the runtime rate 48000
[Sat Apr 14 18:57:06 2018] Bluetooth: RFCOMM TTY layer initialized
[Sat Apr 14 18:57:06 2018] Bluetooth: RFCOMM socket layer initialized
[Sat Apr 14 18:57:06 2018] Bluetooth: RFCOMM ver 1.11
[Sat Apr 14 18:57:12 2018] usb 1-1: timeout: still 11 active urbs on EP #86
[Sat Apr 14 18:57:13 2018] usb 1-1: timeout: still 10 active urbs on EP #86
[Sat Apr 14 18:57:13 2018] rfkill: input handler disabled
[Sat Apr 14 19:29:01 2018] Microsoft.VSCod[3529]: segfault at 8 ip 0000000000728534 sp 00007f5
09f7fd240 error 4 in Microsoft.VSCode.CPP.IntelliSense.Msvc.linux[400000+6d4000]
[Sat Apr 14 19:48:50 2018] this process's nice now changed to -10
[Sat Apr 14 19:48:50 2018] this process's piro value is 10
christianswift@christianswift-virtual-machine ~/桌面/text.c
```

程序实现的创新性和改进思路：

相对于教材上的说明教程，本次实验的创新点在于：

1、使用了多核编译，编译的时间大大缩短，相比较传统编译动辄 2~3 小时，采用多核编译方式只需要 15 分钟左右即可完成编译，减少了等待编译的时间，可以提供更多的时间来探索不同的内核程序会带来什么样的结果，比如本次实验就通过多次编译发现了在 printk 函数的文本内容不加\n 不会输出在 dmesg 控制台中。

2、发现了/boot 分区在 16.04 上是另划一个分区，导致了我首次编译无法完成，经查发现 ubuntu 在 17.04 改版后将/boot 分区挂载在了/目录下，因此推荐使用 ubuntu1704 之后的版本进行本次实验，因为采用虚拟机屏幕分辨率限制导致无法更改/boot 分区位置。

参考文献和资料

[0]赵伟华等.计算机操作系统[M].杭州电子科技大学计算机学院,2018,7(7.1):277-283
[1]后台君. linux 内核实现添加系统调用，实现修改和读取 nice 值[EB/OL].
http://blog.tiaozaoj.com/index.php/archives/188/.
[2]mmshixing.linux 进程调度,优先级、进程 nice 值[EB/OL].
https://blog.csdn.net/mmshixing/article/details/51305138.

程序完整代码

地址：https://github.com/ChristianSwift/Computer_Operation_System_Experiment