|  |
| --- |
|  |
| 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：

## 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。

## 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

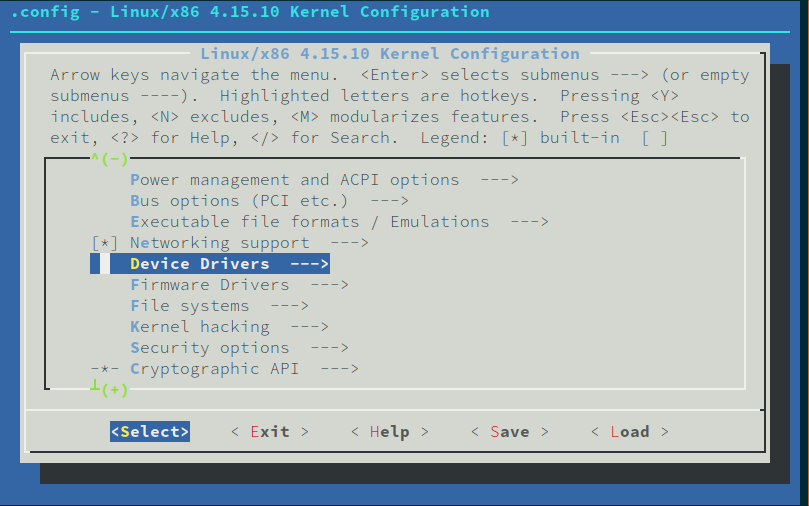
进入目录之后，因为这是刚下载的没有经过编译的内核包，因此可以不用执行make clean命令。

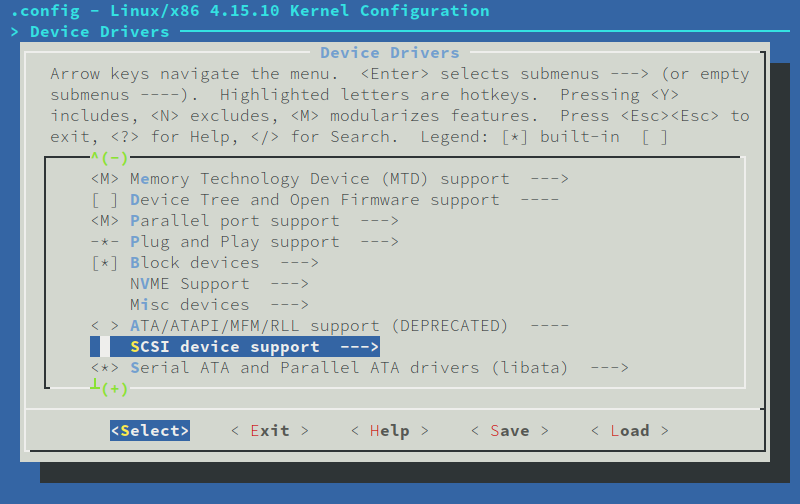
为了后续的工作方便，我们使用aptitude包管理器安装一下需要的软件和工具

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

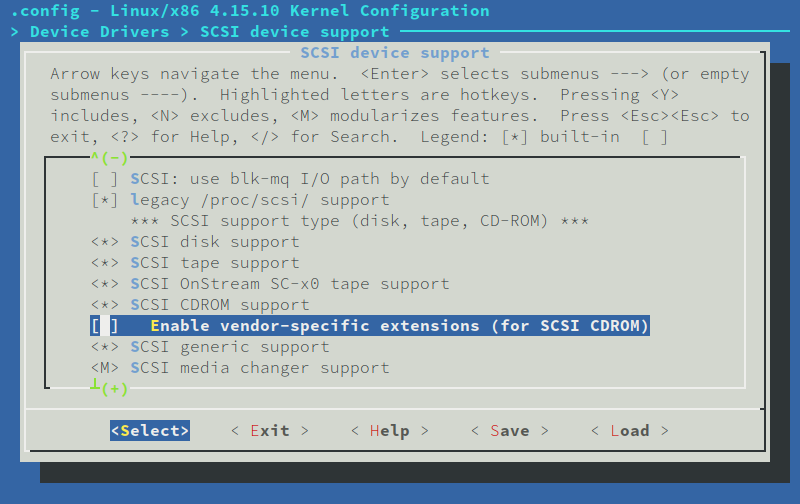
然后配置内核选项

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



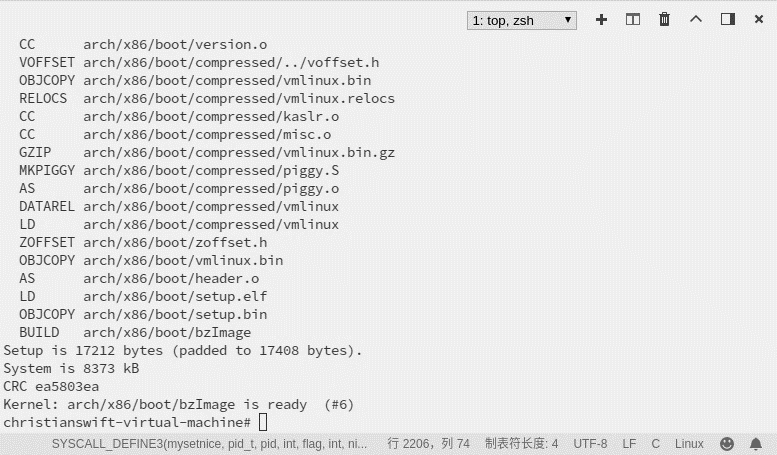


将以下内容打上\*



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

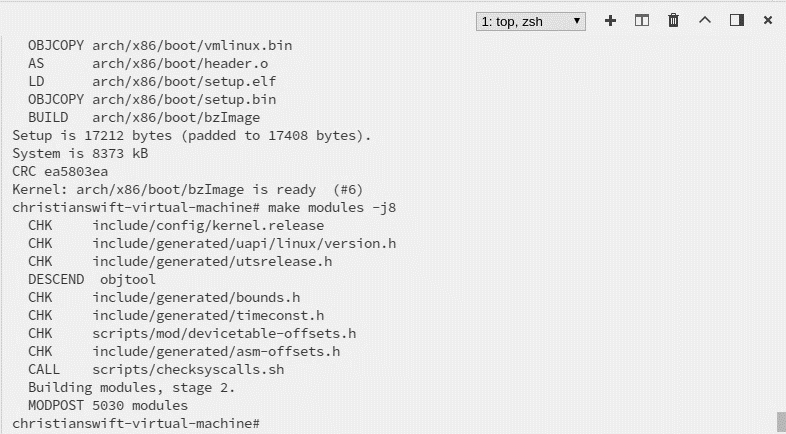


等待编译完成，终端提示Done，使用了-jn指令多核编译后，速度很快，只花了5分钟就编译完成了

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



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



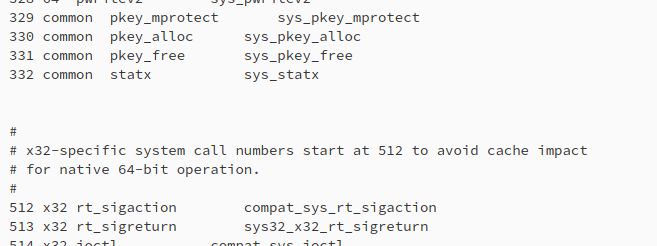
这样我们可以保证内核的正确编译不出问题了。

## 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



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

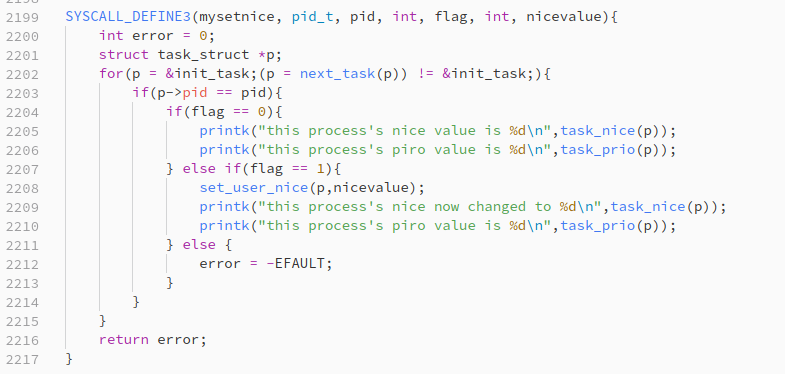
使用vim编辑器打开/usr/src/linux-4.15.10/include/linux/syscalls.h来设置系统调用号

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：

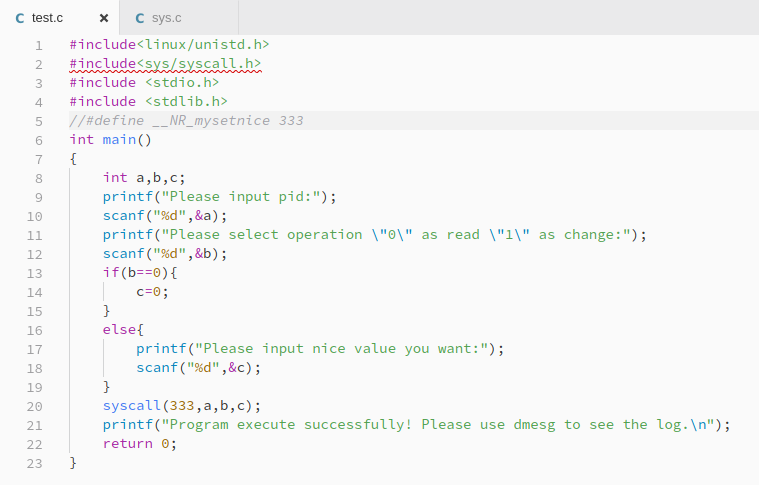


（这里要注意一点，printk内容一定要加\n，否则dmesg不会输出记录，要等下一条才会输出）

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。

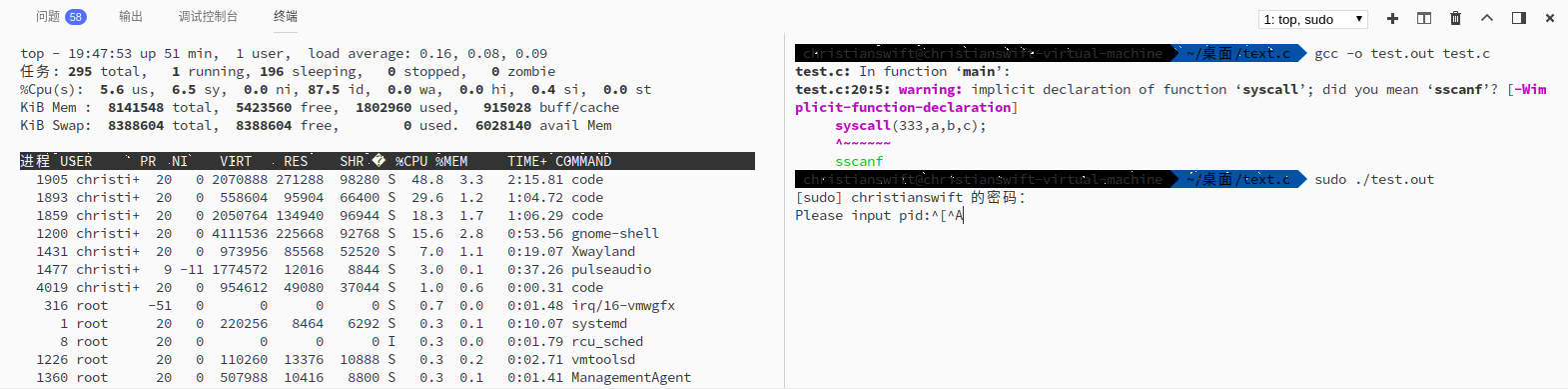
## 编写用户态程序和测试

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

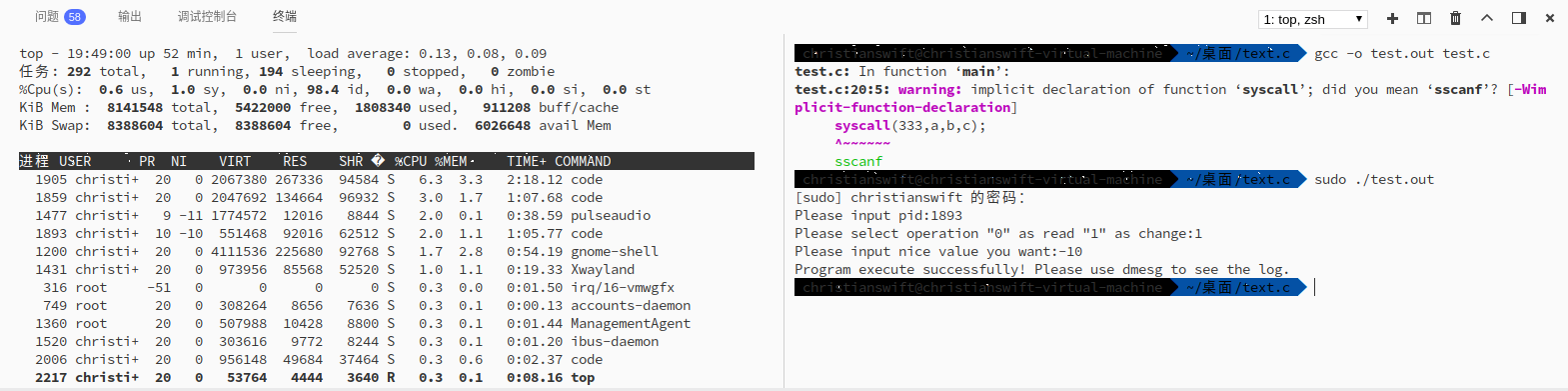


使用gcc -o test.out test.c编译程序，然后./test.out执行用户态程序

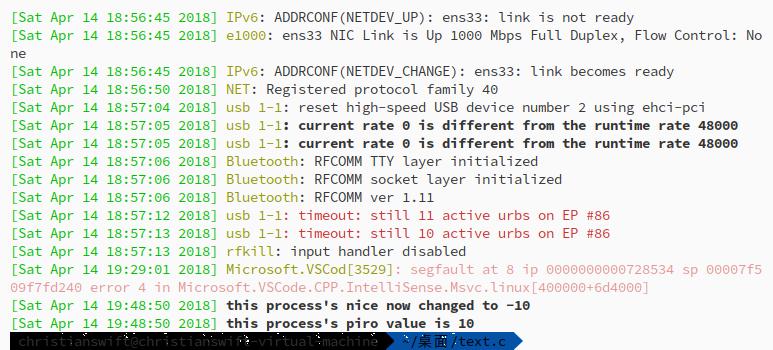
使用top指令随意找一个进程，这里用vscode的进程做一下示范



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



# 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>

