**操作系统课程设计实验报告**

实验题目： 实现Linux内核编译及系统调用

姓 名： 廖越强

学 号： 18041618

组 号： NULL

专 业： 网络工程

班 级： 18272412

老师姓名： 张祯

日 期： 2020 年 3 月 1 日

目 录

[一 题目介绍 1](#_Toc515881940)

[二 实验思路 1](#_Toc515881941)

[三 遇到问题及解决方法 1](#_Toc515881942)

[四 核心代码及实验结果展示 1](#_Toc515881943)

[五 个人实验改进与总结 1](#_Toc515881944)

[5.1 个人实验改进 1](#_Toc515881945)

[5.2 个人实验总结 1](#_Toc515881946)

[六 参考文献 1](#_Toc515881947)

# 一 题目介绍

**实验内容：**

1、  安装操作系统Linux

2、  查看操作系统内核版本 uname –a

3、  下载操作系统内核源码  /usr/src/kernel/，版本略微高一点，解压

4、  编译新内核，挂载，验证新内核版本

5、  添加一个简单系统API，输出“hello，自己的学号”，编译验证

6、  添加要求的系统API，实现对指定进程的 nice 值的修改或读取功能

7、  自己设计一个新的系统API，实现并验证

**考察内容：**

linux内核编译的基本操作，初步了解内核结构，并能添加简单的系统调用

**问题关键点：**

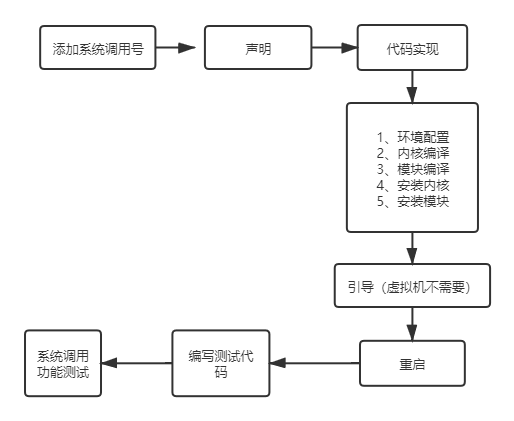
理解内核编译过程中每个步骤的作用，了解系统调用的工作原理

（大家注意，正文部分格式我已设定好，大家不要改，按这个格式书写就可以了）

**（正文部分一律用 宋体，五号字，1.25倍间距，首行缩进两格）**

**（根据实验设计指导书，描述实验题目内容，考察的知识点，问题的关键点）**

# 二 实验思路



1、进入文件分配系统调用号：#vim arch/x86/entry/syscalls/syscall\_64.tbl

2、进入文件添加系统调用原型声明：#vim include/linux/syscalls.h

3、进入文件实现服务例程：#vim kernel/sys.c

4、清除残留的.config和.o文件：#make mrproper

5、配置内核： #make menuconfig

6、编译内核：#make –j4（这里用的是4核处理器）

7、编译模块：#make modules

8、安装模块和安装内核： #make modules\_install 和 make install

9、引导：#update-grub2（虚拟机不需要这一步）

10、立即重启：#reboot –n

**（再次强调，此处需要画人肉眼能清晰的流程图，不清晰的图不要放）**

**（图居中，并且下面的图注格式为： 图1. 名字 ）**

**（对于每个流程图需要配以文字详细阐述其内容，及其关键点，如实验中用到的每个信号量，为什么要用这个信号量，为什么用两个就够了，而不用三个）**

# 三 遇到问题及解决方法

**（此处请同学们务必认真规范书写，对于每个实验，写几条遇到的问题，以条目化形式呈现，如 1. 2. 3. ）**

1. 配置环境和编译的时候遇到很多报错，根据提示安装缺少的包

#sudo apt install build-essential //安装make和gcc等

#apt-get install libncurses5-dev //安装ncurses-devel

#sudo apt-get install flex //安装flex

#sudo apt-get install bison //安装bison

#apt-get install libssl-dev //安装openssl

2、64位的系统调用约300多行（三百多个64位系统调用），注意如果涉及的是64位的系统调用，不能到下半部分的x32那一块里面，否则会产生错误

3、系统调用代码写完后，一定要确保没有语法错误、拼写错误等。因为现在好像并没有可以测试代码准确性的办法。代码一旦出错，结局可能就是make编译的时候，编译了一两个小时最后提示报错，这样就只能回去检查系统调用的代码，编译的时间就浪费掉了。

# 四 核心代码及实验结果展示

**（不需要粘贴所有的代码，把核心的代码粘贴即可，实验结果也需要截图放进来，每个图需要按照图片格式编写）**

**核心代码：**

1. 添加系统调用号
2. 333     64      mysyscall           sys\_mysyscall
3. 334     64      mysetnice           sys\_mysetnice
4. 335     64      mygetcpu            sys\_mygetcpu

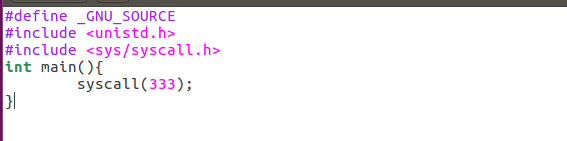
（2）声明

1. asmlinkage **long** sys\_mysyscall(**void**);
2. asmlinkage **long** sys\_mysetnice(pid\_t pid,**int** flag,**int** nicevalue,**void** \_\_user \* prio,**void** \_\_user \* nice);
3. asmlinkage **long** sys\_mygetcpu(**void**);

（3）代码实现

1. SYSCALL\_DEFINE0(mysyscall)
2. {
3. printk("Hello, this is lyq's syscall test!\n");
5. **return** 0;
6. }
7. SYSCALL\_DEFINE5(mysetnice, pid\_t, input\_pid, **int**, flag, **int**, nicevalue, **void** \_\_user \*, prio, **void** \_\_user \*, nice)
8. {
9. **struct** pid \* kpid;
10. **struct** task\_struct \* task;
11. **int** nicebef;
12. **int** priobef;
13. kpid = find\_get\_pid(input\_pid);  //return pid
14. if(kpid == NULL){
15. printk("invalid pid input !!!");
16. **return** ESRCH;
17. }
18. else{
19. task = pid\_task(kpid, PIDTYPE\_PID);  //return task\_struct
20. nicebef = task\_nice(task);  //get value of nice before change
21. priobef = task\_prio(task);  //get value of prio before change
23. **if**(flag == 1){
24. set\_user\_nice(task, nicevalue);
25. printk("the value of nice before change: %d\tafter change: %d\n", nicebef, nicevalue);
26. **return** 0;
27. }
28. **else** **if**(flag == 0){
29. copy\_to\_user(nice, (**const** **void**\*)&nicebef, **sizeof**(nicebef));
30. copy\_to\_user(prio, (**const** **void**\*)&priobef, **sizeof**(priobef));
31. printk("The nice value of the process: %d\n", nicebef);
32. printk("THe prio value of the peocess: %d\n", priobef);
33. **return** 0;
34. }
35. printk("Invalid flag, which should be 0 or 1");
36. **return** EINVAL;
37. }
38. }
39. SYSCALL\_DEFINE0(mygetcpu)
40. {
41. **int** cpu\_num;
42. cpu\_num =  num\_online\_cpus();
43. printk("Available cpu: %d\n",cpu\_num);
45. **return** cpu\_num;
46. }

**系统调用测试以及实验结果图片**



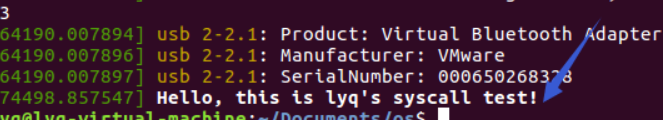
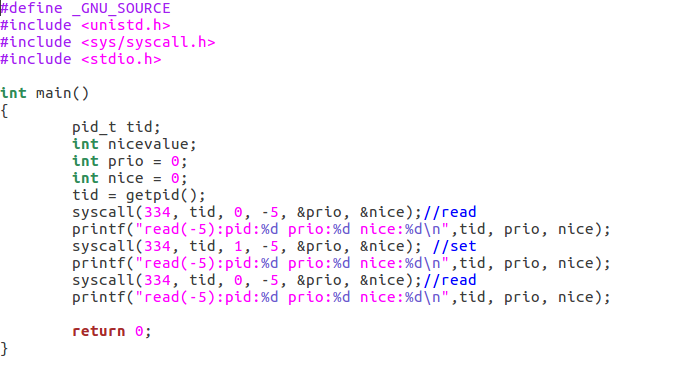
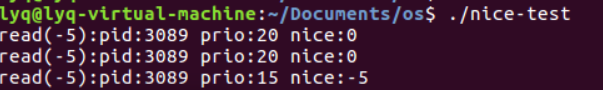


图1、图2 简单系统调用测试





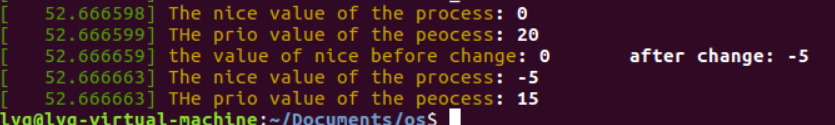
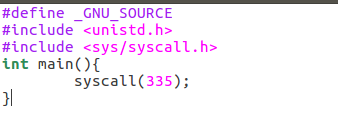


图2、图3、图4 nice值得读取与修改



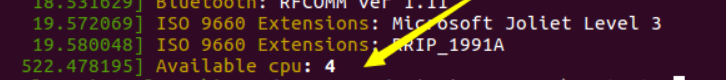


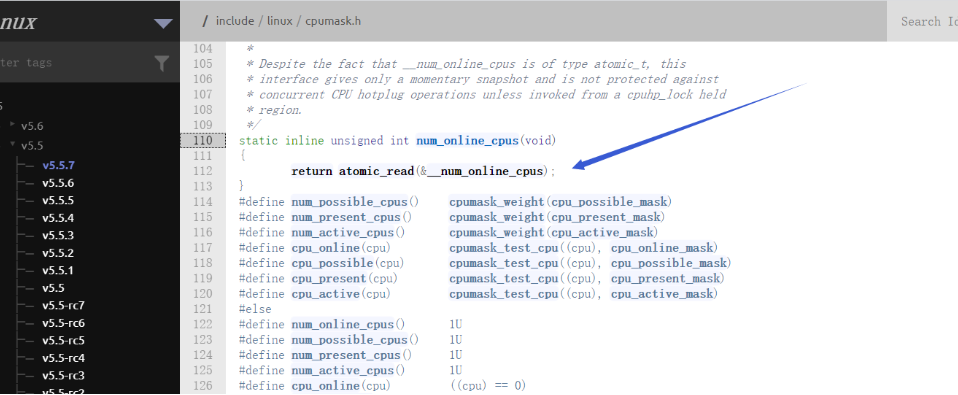
图5、图6 自己设计的查看可用cpu数的系统调用

# 五 个人实验改进与总结

## 5.1 个人实验改进

**（大部分同学的代码都是参考网上资料，或者借鉴已完成同学的，此处重点突出自己如何改进，为什么这样改进，说出自己改进部分的优缺点，有效文字不少于200字）**

第二个实验（修改nice值）中，应该对于用户传入的pid值是否有效进行判断。如果pid值非法的话，最终find\_get\_pid函数返回的是NULL，会导致意料之外的情况出现，所以这里我添加了一个对返回值kpid判断，只有当返回值kpid合法时，才进行后续的读取、修改nice值操作。第三个自己设计系统调用想了很久。一开始一直没有什么思路，自己想到的一些功能，好像那300多个系统调用都已经实现了，想了好一段时间，最后去看看内核的源码，无意间发现了一些对关于cpu的函数，于是就挑了一个试试看。



## 5.2 个人实验总结

**（此处类似于写观后感，同学们辛辛苦苦完成实验，单独编译内核可能就折腾很久，相信会有很多的感触和收获，让你的笔在第一时间记录这美好的瞬间，有效文字不得少于200字）**

1、make编译前要多留意代码正确性，依次检查系统调用号、声明和代码实现三个文件。确定没有错误后再编译，避免时间浪费。

2、每次编译前最好都清理一下上次编译残留的.config和.o文件，避免内存不够导致编译出错。

3、如果使用linux虚拟机的话，配置尽量还是给高一点吧，我这里给了4G内存，4个处理器，因此编译的时候可以make -4j进行四核编译（会快一点），处理器给多点也可以降低编译过程虚拟机卡死的情况。（编译了半天突然卡死是真的难受）

4、为了避免冲突，变量名不能随便起。这里的话，我在修改nice值那个实验中把pid参数起名为input\_pid，避免与struct pid冲突。

# 六 参考文献

**（此处罗列同学们在做实验过程中 所参考的文献、书籍、网站， 或者向某个高手同学请教都可以写， 大多数同学应该是参考 网站、书籍和问人比较多，以条目化的形式呈现， 如1. 2. 3. ）**

1、内核源码

<https://elixir.bootlin.com/linux/v2.6.28.6/source/include>

1. 系统调用示例

https://blog.csdn.net/fengfeng0328/article/details/70174984