本科生实验报告

实验课程:_操作系统原理_____

实验名称:_linux 内核 5.10 添加系统调用_____

专业名称:_信息与计算科学_____

学生姓名:_张文沁_____

学生学号:_20337268_____

实验地点:_实验楼 D402_____

实验成绩:_____

报告时间:_2022.03.18_____

1. 实验要求

1. 在 Linux 操作系统内核如 5.10 或者 Linux 0.11 中添加新的系统调用，并且编译、启动新的内核，测试新加入的系统调用的有效性
2. 编写实验报告、结合实验过程来谈谈你完成实验的思路和结果，最后需要提供实验的新加入系统调用的运行结果截屏来证明你完成了实验。
3. 实验不限语言， C/C++/Rust 都可以。
4. 实验不限平台， Windows、Linux 和 MacOS 等都可以。
5. 实验不限 CPU， ARM/Intel/Risc-V 都可以。

注意：编译调试内核比较繁琐，可以利用操作系统实验 1 中学到的知识，利用 qemu 启动新的内核，同时编写测试程序，制作 initramfs 启动测试程序，观察结果，并利用 gdb 远程调试。当然也可以不用 qemu，利用 Virtualbox 启动新的内核操作系统，测试系统调用。

本指导的添加系统调用方式只是诸多方式的一种，同学们可以自由选择其他方式添加系统调用。

2. 实验过程

3. 关键代码

4. 实验结果

一共完成了两个系统调用：一为助教实验指导中所给，一为自己完成的“Hello World”系统调用，如下为一实验：

1. 安装相关软件

```
sudo apt-get update && sudo apt-get upgrade
sudo apt install binutils
sudo apt install gcc
sudo apt install nasm
sudo apt install qemu
sudo apt install cmake
sudo apt-get install libc6-dev
sudo apt-get install libelf-dev
sudo apt-get install libncurses5-dev libssl-dev
sudo apt-get install build-essential openssl
sudo apt-get install libidn11-dev libidn11
sudo apt-get install zlibc minizip
sudo apt-get install bison
sudo apt-get install flex
sudo apt-get install pkg-config
```

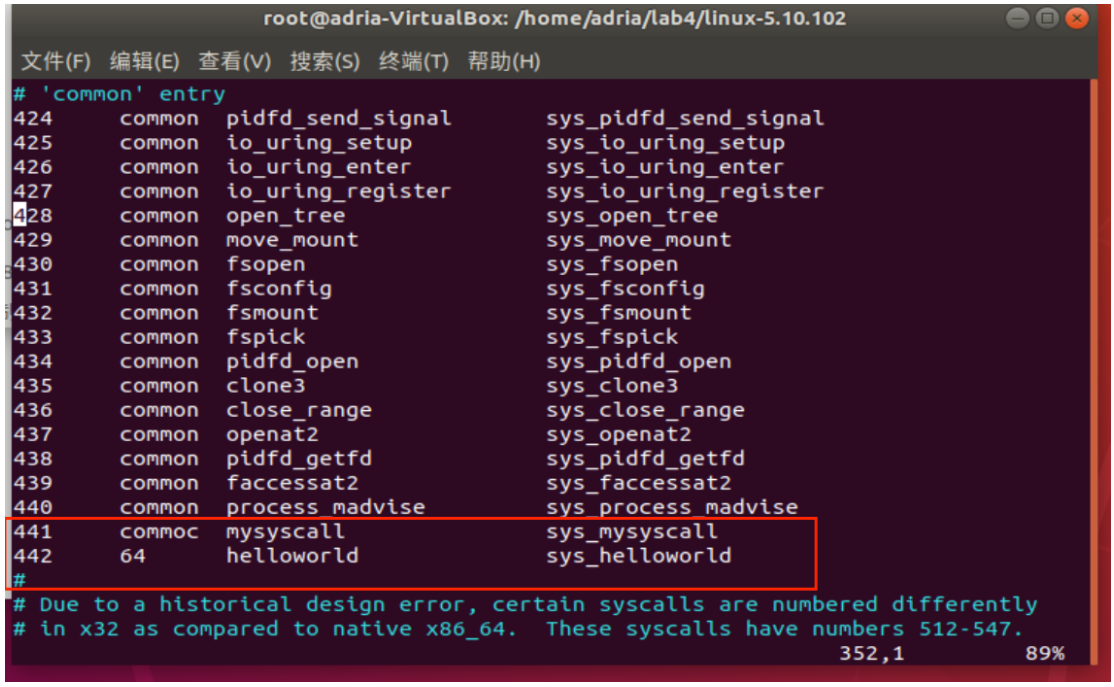
2. 添加系统调用：

1) 在 syscall_64.tbl 下添加系统调用的标志符号：

```
sudo vim arch/x86/entry/syscalls/syscall_64.tbl

441     common  mysyscall               sys_mysyscall
442     64      helloworld              sys_helloworld
```

结果如下：441 为一实验，442 为自我实现



2) 在 syscall.h 中添加系统调用函数的解释

```
sudo vim include/linux/syscalls.h

/* My Own syscall */
asmlinkage long sys_mysyscall(int number);
asmlinkage long __x64_sys_helloworld(void);
```

结果如下:

```
int __sys_setsockopt(int fd, int level, int optname, char __user *optval,
                int optlen);
asmlinkage long __x64_sys_helloworld(void);
#endif
                                                              1375,1        底端
```

3) 在 sys.c 中添加系统调用函数

```
sudo vim kernel/sys.c

/* My Own syscall */
SYSCALL_DEFINE1(mysyscall,int,number)
{
        printk("mysyscall\n");
        printk("The Number You Enter Is %d\n",number);
        return number;
}
asmlinkage long __x64_sys_helloworld(void){
    printk("Hello world Hello Adria! ");
    return 0;
}
```

结果如下，都是在 linux-5.10.102 下直接打开

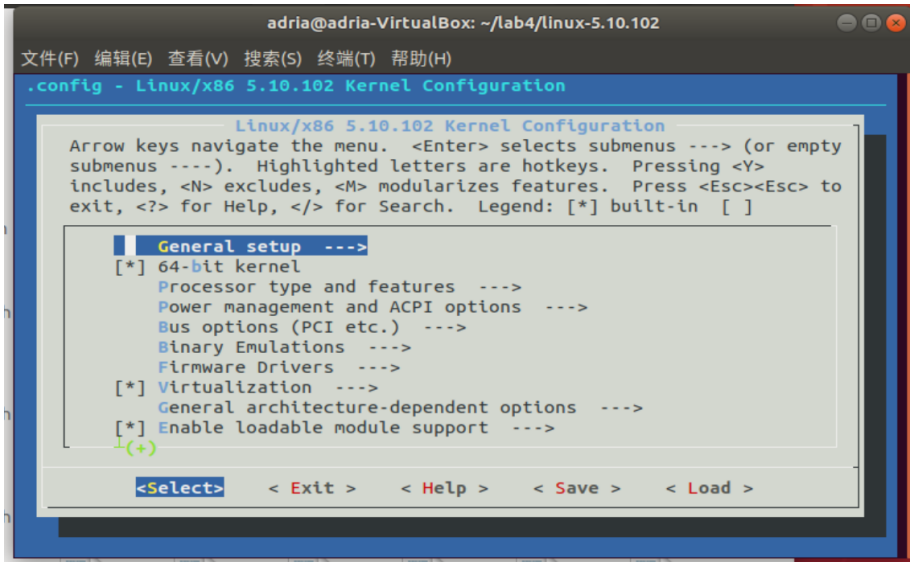
```
}
SYSCALL_DEFINE1(mysyscall,int number)
{
        printk("mysyscall\n");
        printk("The number you enter is %d\n",number);
        return number;
}
#endif /* CONFIG_COMPAT */
                                C ▾  制表符宽度：8 ▾   第 2693 行，第 33 列 ▾   插入
asmlinkage long __x64_sys_helloworld(void){
        printk("Hello world Hello Adria!");
        return 0;
}
#endif /* CONFIG_COMPAT */
                                C ▾  制表符宽度：8 ▾   第 2688 行，第 18 列 ▾   插入
```

3. 编译内核:

代码如下:

```
cd /usr/src/linux-5.11.7
make menuconfig
```

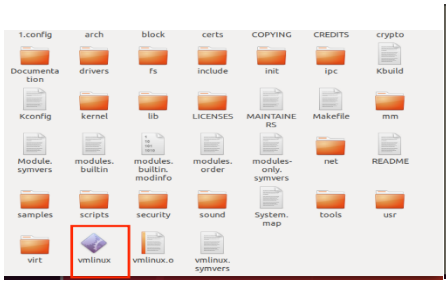
进入，没有什么需要修改，直接保存退出：

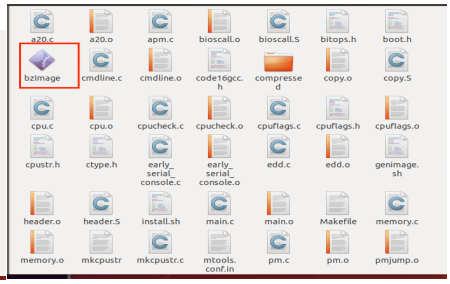


直接：

```
make -j8
```

Linux 压缩镜像 linux-5.10.102/arch/x86/boot/bzImage 和符号表 linux-5.10.102/vmlinux 已经生成





4. 制作 initrmfs

按照步骤完成：

将代码编译成可执行文件

```
gcc -o testsyscall -static testsyscall.c
```

用 cpio 打包 initramfs

```
echo testsyscall |  cpio -o --format=newc >  testsyscall-initramfs
```

5. 启动内核

使用所给指令完成

```
qemu-system-x86_64 -kernel linux-5.10.105/arch/x86_64/boot/bzImage -initrd
testsyscall-initramfs -s -S -append "console=ttyS0 rdinit=testsyscall" -nographic
```

6. gdb 调试

在另外一个Terminal下启动gdb，注意，不要关闭qemu所在的Terminal。

```
cd linux-5.10.105/
gdb
```

在gdb下，加载符号表

```
file linux-5.10.105/vmlinux
```

在gdb下，连接已经启动的qemu进行调试

```
target remote:1234
```

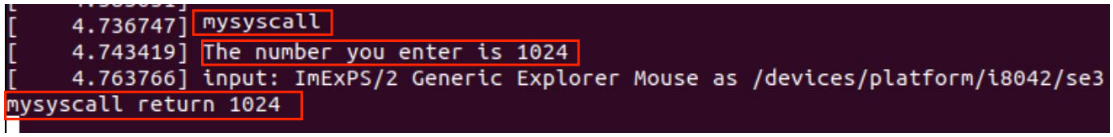
在gdb下，为start_kernel函数设置断点。

```
break start_kernel
```

在gdb下，输入c运行。

```
c
```

7. 最终结果：



8. Hello world 系统调用：

在编译内核之后稍有不同：具体如下所示：

1) 安装内核：

```
make modules_install
make install
```

2) 增加启动项：

```
grub2-mkconfig -o /boot/grub2/grub.cfg
```

3) 重启

4) 增加测试函数

```
#include<sys/time.h>
#include<stdio.h>
#include<unistd.h>
int main(void){
    syscall(442);
    return 0;
}
```

5) 编译：

```
gcc test.c
```

6) 查看结果：

```
./a.out
dmesg
```

dmesg 可查看开机信息。内核中使用的 printk()并不是将消息输出在终端上，而是内核的 ring buffer 中。

5. 总结

1. Sudo:vim 找不到命令：安装或更新 vim

2. 多次编译内核失败：检查压缩镜像和符号表是否已经生成，多次重新尝试，猜测可能是过程中修改了相关文件，导致无法正常编译。