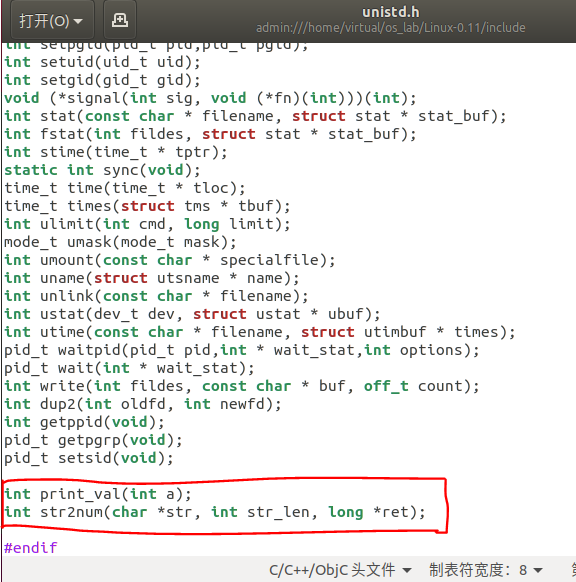
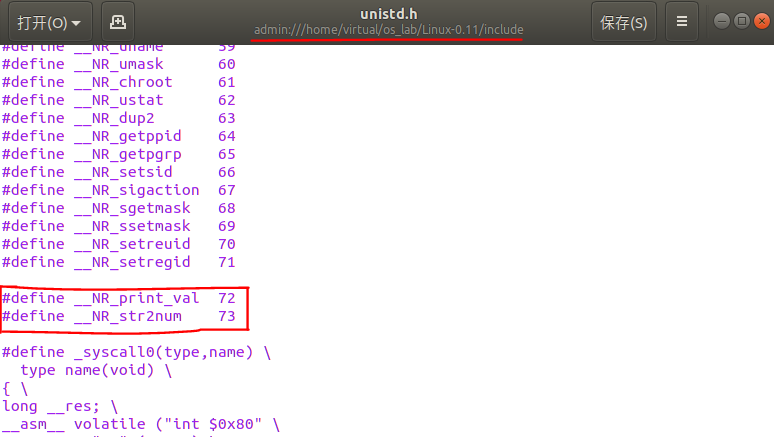
**操作系统实验二**

**PB18151866 龚小航**

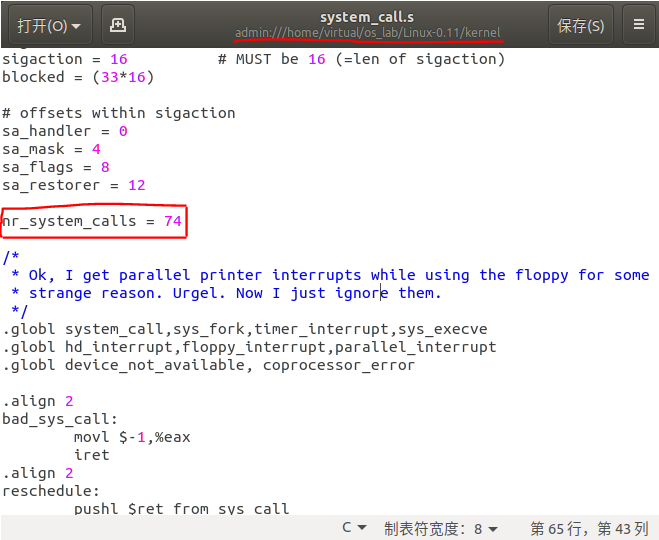
**实验内容一：添加Linux系统调用**

实验目标：了解系统调用如何实现，并在Linux 0.11中添加两个系统调用。

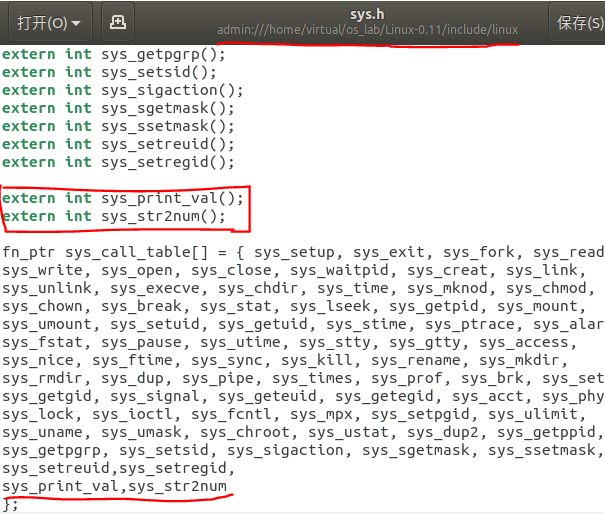
实验步骤：

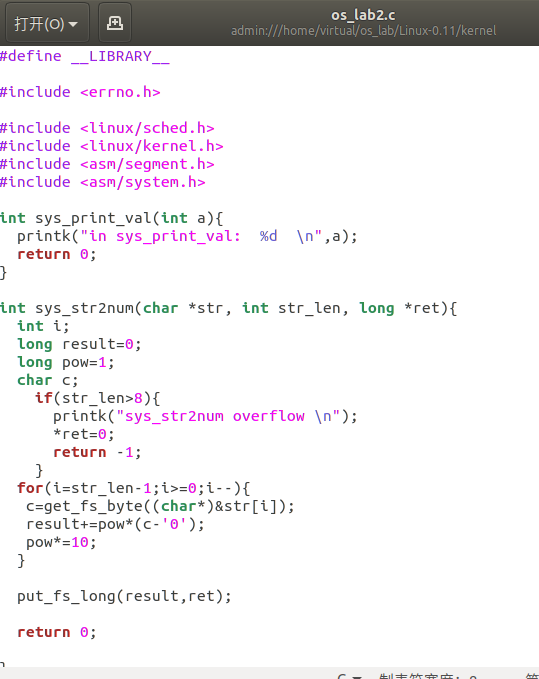
①在os\_lab/Linux-0.11/include/unistd.h下定义系统调用号，并声明系统调用函数的形式。

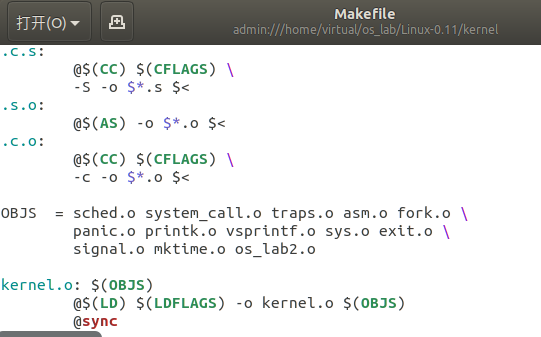
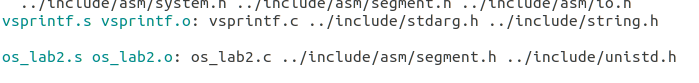
②在os\_lab/Linux-0.11/kernel/system\_call.s中修改系统调用的个数，以使此系统调用被调用时，可以识别到。



　　③在os\_lab/Linux-0.11/include/linux/sys.h中添加extern头，再在sys\_call\_table[]中加入系统调用的‘地址’。

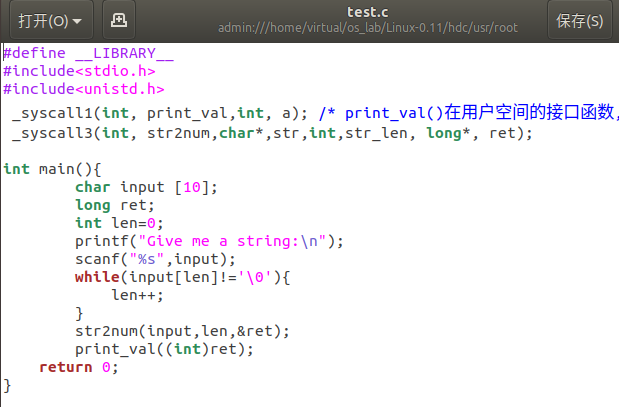


④在linux-0.11/kernel中实现该系统调用，源代码如下。 并修改Makefile文件。修改如下：



⑤将os\_lab/Linux-0.11/hdc/usr/include/unistd.h文件中添加系统调用号和系统调用的声明。

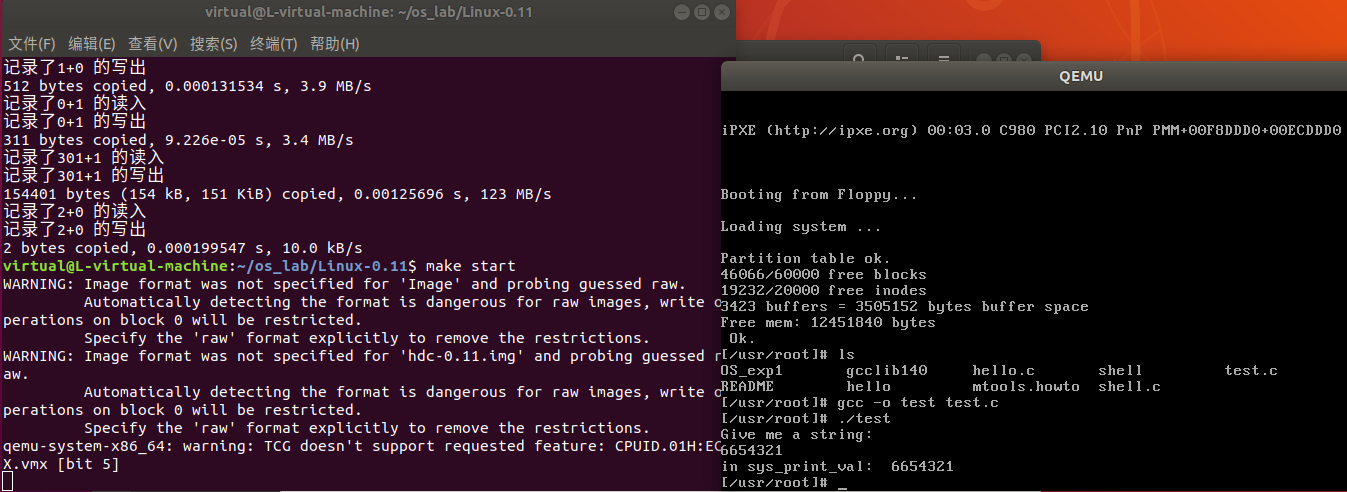
　　 这一步操作需要挂载img镜像文件。这一步和第一步更改的内容相同。这里略去。

⑥写测试程序进行测试。测试代码如下，将其放入镜像文件的root目录下.

实验结果：

在主机终端中依次输入命令make clean; make; make start，编译内核并进入 Linux-0.11 系统。

使用gcc编译刚才的测试文件，运行，实现实验要求的输出输入。至此完成第一部分实验。



**实验第一部分问答：**

**问题1：简要描述如何在Linux 0.11添加一个系统调用**

① 定义系统调用号，并声明系统调用函数的形式。

　　② 修改系统调用的个数，以使此系统调用被调用时，可以识别到。

　　③ 添加extern头，并在sys\_call\_table[]中加入系统调用的‘地址’。

　　④ 写调用的具体代码，实现该系统调用，并修改Makefile文件。

　　⑤ 将挂载目录下的unistd.h文件中添加系统调用号和系统调用的声明。（和第一步修改内容一样）

　　⑥写测试程序进行测试。

**问题2：系统是如何通过系统调用号索引到具体的调用函数的?**

1、API把系统调用的编号存入EAX中，把函数参数存入其他通用寄存器中并触发0x80号中断；

2、调用system\_call函数，函数中语句call sys\_call\_table(,%eax,4)会在中断向量表中找到相应的系统调用的函数地址；

3、通过这个地址执行真正的系统调用函数。

**问题3：在Linux-0.11中, 系统调用最多支持几个参数? 有什么方法可以超过这个限制吗?**

在Linux-0.11中，系统调用最多支持3个参数。

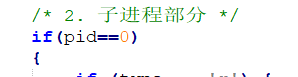
可以通过写一个新的宏来支持4个参数的系统调用，也可以将多余的参数传入其他的通用寄存器中。

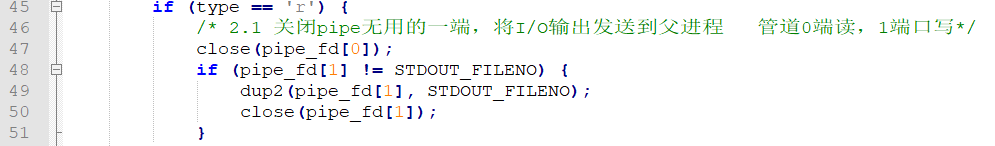
**实验内容二：熟悉Linux下常见的系统调用函数**

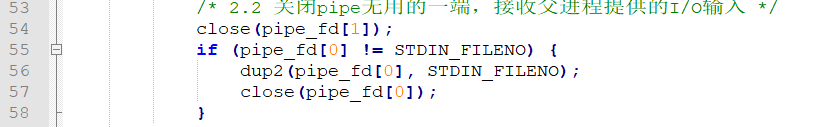
实验目标：利用Linux提供的系统调用，实现一个简单shell程序

实验步骤：在给出的代码框架上，补全shell代码。

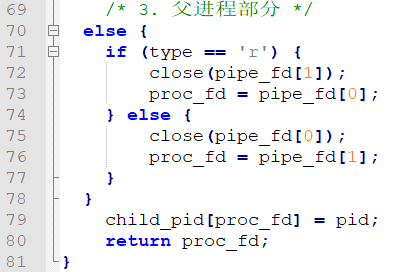
 第一个添加部分，使用fork（）创建一个新的进程。

 子进程pid=0，这也是区分子进程与父进程的标志，它们执行不同的代码。

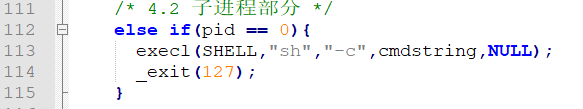
 接下来的2.1修改处，pipe无用端是读端口，因此关闭0端，将1端指向标准输出。完成后关闭1端。

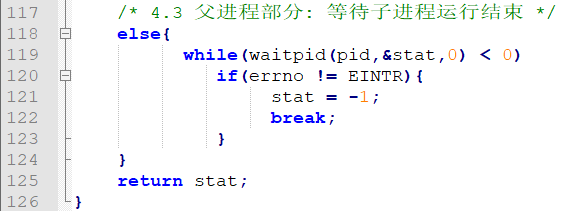
 2.2修改处与上一处相似，pipe无用端是写端口，因此关闭1将0端指向标准输入。完成后关闭。

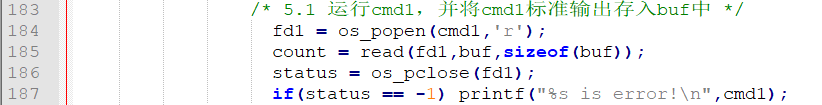
 下面的修改处补全execl函数，通过该系统调用来执行shell命令。

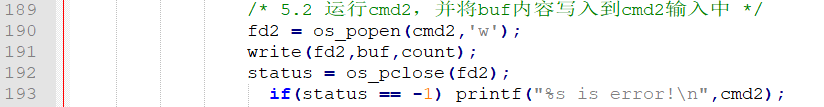
 接下来是父进程部分，和子进程部分类似。Type==r时关写开读，反之亦然。

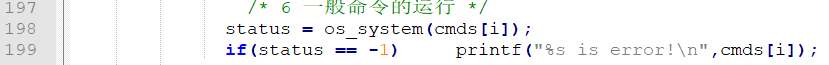
 4.1处，通过fork() 创建新进程，if 判断创建是否成功，如果失败标志变量 stat 值即为-1。

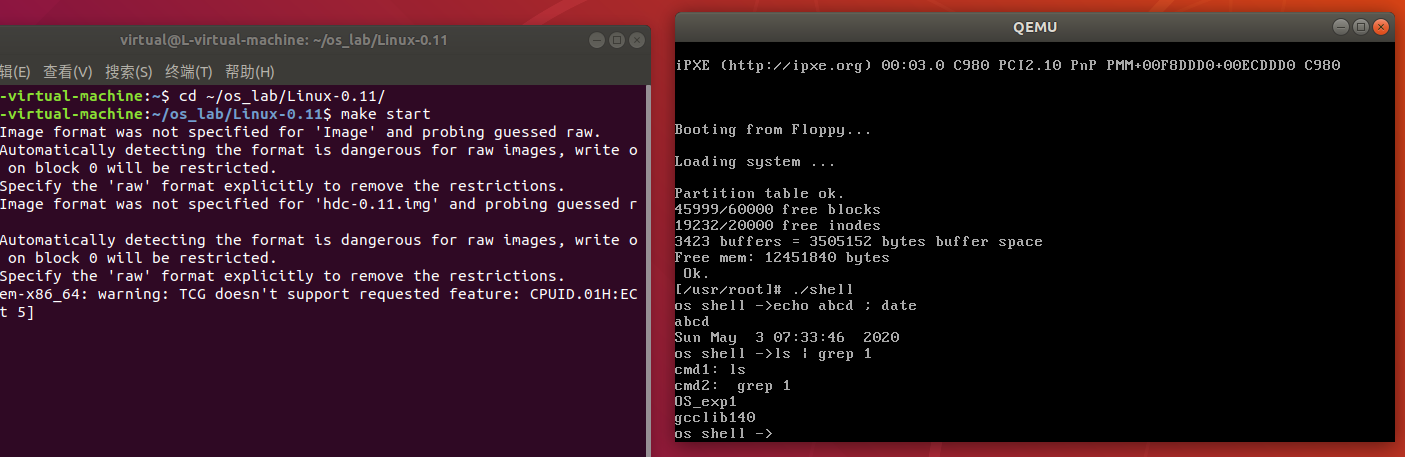
 4.2处，判断子进程，并使ececl系统调用来执行命令：

 4.3处，父进程部分，通过waitpid()等待子进程运行结束后继续运行，如果出错则将标志变量 stat 赋为-1.

 5.1处，这部分命令需要管道。先通过 os \_popen() 打开管道，然后用 read()将 cmd1 标准输出存入buf中，存入 字的数量返回赋值给 count，做完以后，利用os\_pclose() 关闭管道，最后再判断关闭是否成功。

 5.2，与上一处完全相同，只不过是将 buf 的内容写入到 cmd2的输入中：

 最后一处需要补充普通命令的运行，即不需要管道。用 os\_system system()函数运行命令即可。

**第二部分实验结果：**

输入输出符合预期，第二部分实验结束。