1. 针对你所下载的（非运行的）Linux内核，系统有多少个系统调用？（对内核源码中相关文件内容截屏）

答：Linux-0.11中有72个系统调用，源代码中相关内容（include/linux/sys.h）的截屏如下：

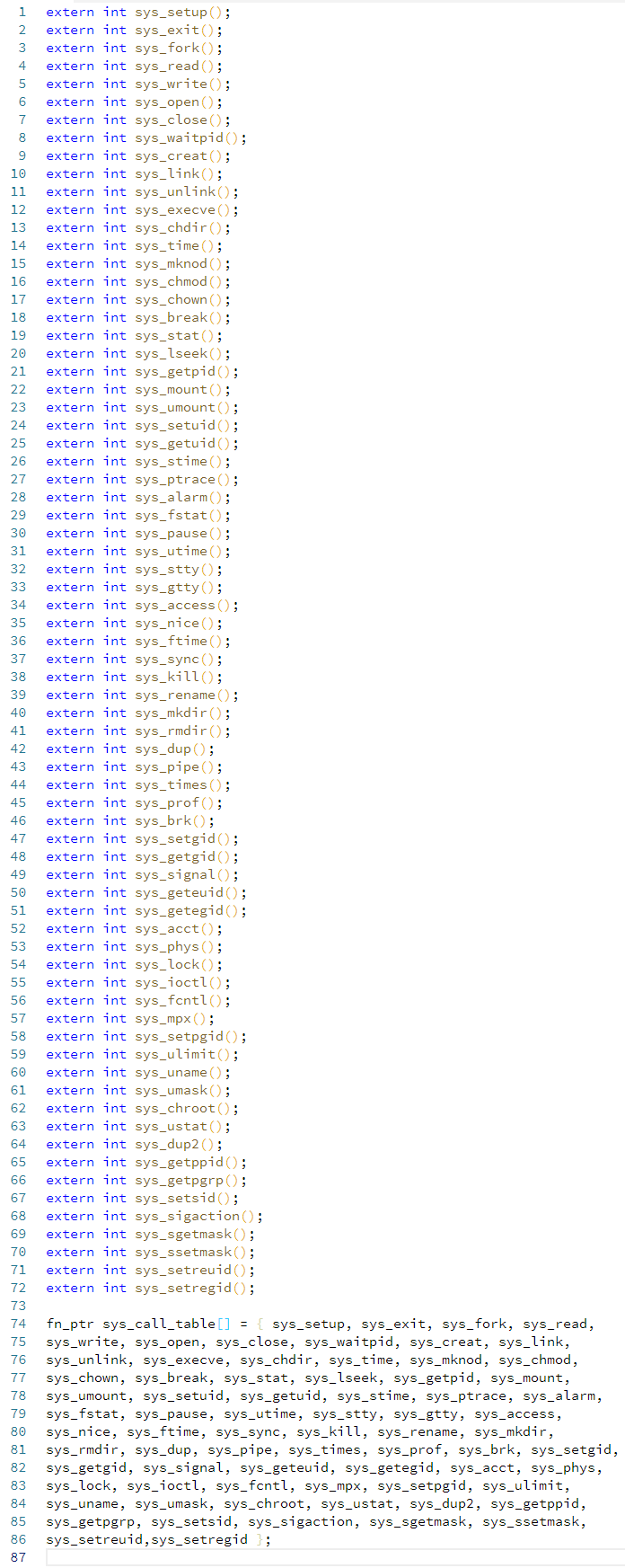


图 1

1. 以helloword程序为例（主要是printf()调用）解释在此Linux中系统调用（从int0x80开始）的处理过程（包括中断的响应、系统调用参数的传递、具体调用函数（sys-）入口地址的确定、系统调用返回、中断返回等细节）。

答：当用户态进程发起一个系统调用，CPU将切换到内核态并开始执行一个内核函数。内核函数负责响应应用程序的要求，例如操作文件、进行网络通讯或者申请内存资源等。

**（1）调用流程：**

我们以一个假设的系统调用xyz为例，介绍一次系统调用的所有环节。

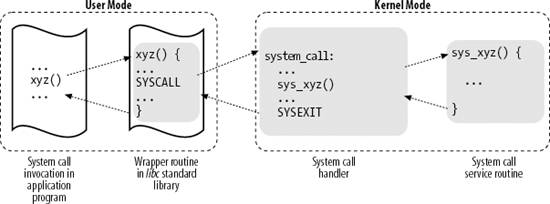


图 2

如上图，系统调用执行的流程如下：

1. 应用程序代码调用系统调用(xyz)，该函数是一个包装系统调用的库函数；
2. 库函数(xyz)负责准备向内核传递的参数，并触发软中断以切换到内核；
3. CPU被软中断打断后，执行中断处理函数，即系统调用处理函数(system\_call)；
4. 系统调用处理函数调用系统调用服务例程(sys\_xyz)，真正开始处理该系统调用；

**（2）执行态切换**

应用程序(application program)与库函数(libc)之间，系统调用处理函数(system call handler)与系统调用服务例程(system call service routine)之间，均是普通函数调用，而库函数与系统调用处理函数之间，由于涉及用户态与内核态的切换，要复杂一些。

Linux通过软中断实现从用户态到内核态的切换。用户态与内核态是独立的执行流，因此在切换时，需要准备执行栈并保存寄存器。

内核实现了很多不同的系统调用(提供不同功能)，而系统调用处理函数只有一个。因此，用户进程必须传递一个参数用于区分，这便是系统调用号(system call number)。在Linux中，系统调用号一般通过eax寄存器来传递。

总结起来，执行态切换过程如下：

1. 应用程序在用户态准备好调用参数，执行int指令触发软中断，中断号为0x80；
2. CPU被软中断打断后，执行对应的中断处理函数，这时便已进入内核态；
3. 系统调用处理函数准备内核执行栈，并保存所有寄存器(一般用汇编语言实现)；
4. 系统调用处理函数根据系统调用号调用对应的C函数——系统调用服务例程；
5. 系统调用处理函数准备返回值并从内核栈中恢复寄存器；
6. 系统调用处理函数执行ret指令切换回用户态；

printf()函数是建立在系统调用之上，更高层次的库函数，应用程序要输出文字，需调用write这个系统调用。int指令触发软中断0x80，程序将陷入内核态并由内核执行系统调用。

这个过程中将会首先进入系统调用处理程序（kernel/system\_call.s，如图3）：

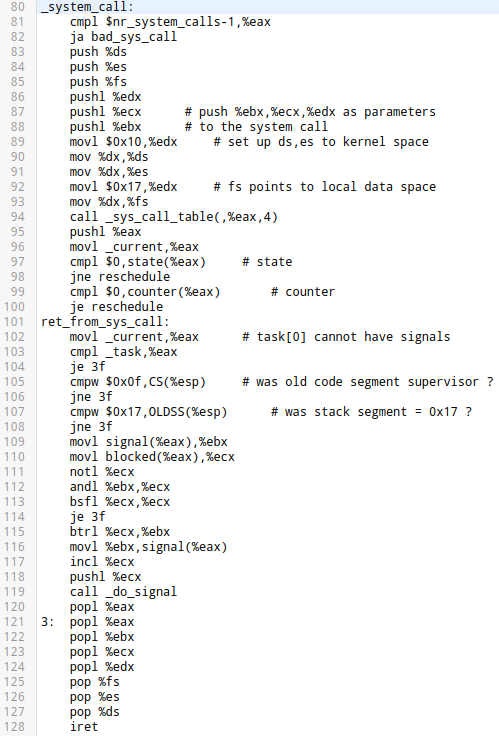


图 3

系统调用处理程序首先检查寄存器eax中存放的系统调用号（sys\_write的系统调用号为4），并在include/unistd.h（图4）中检查相应的系统调用号是否存在：

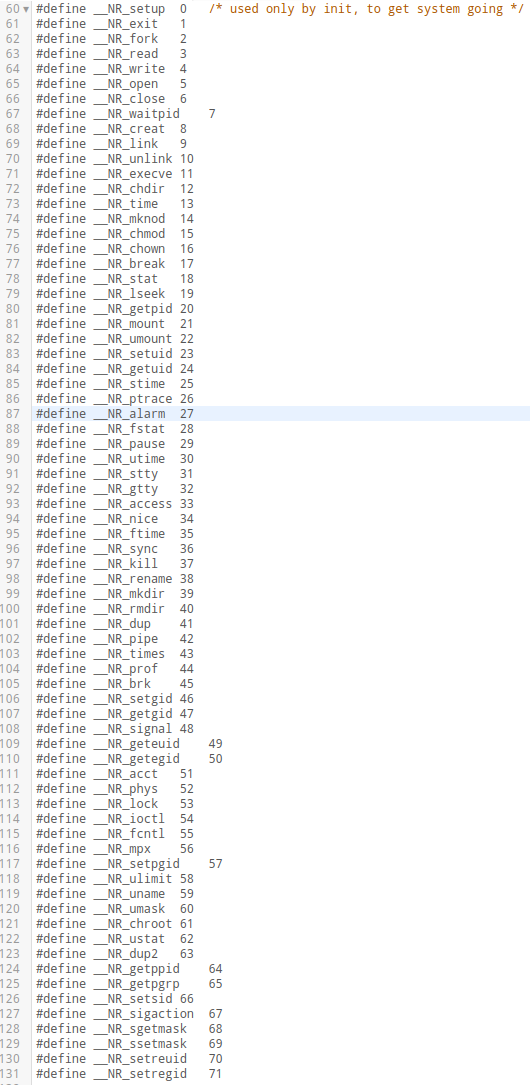


图 4

然后根据include/linux/sys.h（图1）中的sys\_call\_table函数指针数组找到相应的write系统调用服务例程（sys\_write函数位于linux/fs/read\_write.c，如图5）：

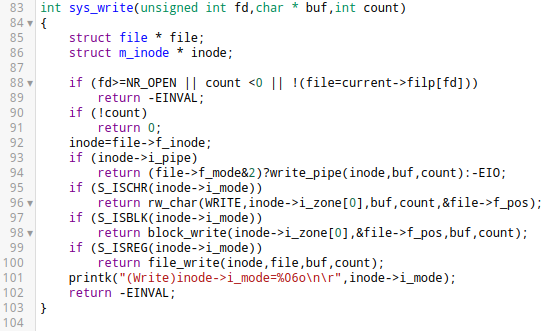


图 5

使用寄存器ebx、ecx、edx传递参数，执行完毕后在eax中保存写入的字节数并向调用程序返回0；若执行出错则将错误类型码取反存入全局变量errno，并向调用程序返回-1，然后执行ret\_from\_sys\_call结束系统调用，通过iret结束中断。系统调用执行完毕后，内核将负责切换回用户态，恢复之前保存的所有寄存器，应用程序继续执行之后的指令。

参考文献：

[] 赵炯．Linux内核完全注释修正版3.0[DB/OL]．2007-06-07．

[2] 小菜学编程．系统调用原理[DB/OL]．<https://learn-linux.readthedocs.io/zh_CN/latest/system-programming/syscall/principle.html>．