1.6 open系统调用

前面，copy代码假设文件描述符已经设置好了。但是一般情况下，我们需要能创建文件描述符，最直接的创建文件描述符的方法是open系统调用。下面是一个叫做open的源代码，它使用了open系统调用。

```
[crash] cat -n open.c
     1
     2  // open.c: create a file, write to it.
     3
     4  #include "kernel/types.h"
     5  #include "user/user.h"
     6  #include "kernel/fcntl.h"
     7
     8  int
     9  main()
    10  {
    11    int fd = open("output.txt", O_WRONLY | O_CREATE);
    12    write(fd, "ooo\n", 4);
    13
    14    exit(0);
    15  }
[crash]
```

> 学生提问：字节流是什么意思？
>
> Robert教授：我。。。我的意思是，如果一个文件包含了一些字节，假设包含了数百万个字节，你触发了多个read，每个read读取100个字节，第一次read会读取前100个字节，第二次读取101-200个字节，第三次读取201-300个字节，这就是我的意思。（字节流就是一段连续的数据按照字节的长度读取）

这个叫做open的程序，会创建一个叫做output.txt的新文件，并向它写入一些数据，最后退出。我们看不到任何输出，因为它只是向打开的文件中写入数据。

但是我们可以查看output.txt的内容。

并看到open程序写入的“ooo”。所以，代码中的第11行，执行了open系统调用，将文件名output.txt作为参数传入，第二个参数是一些标志位，用来告诉open系统调用在内核中的实现：我们将要创建并写入一个文件。open系统调用会返回一个新分配的文件描述符，这里的文件描述符是一个小的数字，可能是2，3，4或者其他的数字。

之后，这个文件描述符作为第一个参数被传到了write，write的第二个参数是数据的指针，第三个参数是要写入的字节数。数据被写入到了文件描述符对应的文件中。

文件描述符本质上对应了内核中的一个表单数据。内核维护了每个运行进程的状态，内核会为每一个运行进程保存一个表单，表单的key是文件描述符。这个表单让内核知道，每个文件描述符对应的实际内容是什么。这里比较关键的点是，每个进程都有自己独立的文件描述符空间，所以如果运行了两个不同的程序，对应两个不同的进程，如果它们都打开一个文件，它们或许可以得到相同数字的文件描述符，但是因为内核为每个进程都维护了一个独立的文件描述符空间，这里相同数字的文件描述符可能会对应到不同的文件。

对于open系统调用和这里的小程序，大家有什么问题吗?

> 学生提问：我不太熟悉C语言，这里的文件描述符与非C语言中有什么区别?如果使用Python的话，语法上会不会简单点?
>
> Robert教授：Python肯定提供了对于open的较好的封装。通常来说，Python提供的是更高级的函数，比如说Python不会使用指向内存的指针，并且Python会为你做更多的错误检查。当我们在Python中打开文件或者写入文件时，你在Python中的调用最终会走到跟我们例子中一样的系统调用来，这是个好的回答吗?