《Linux应用程序开发实验报告》

题目: 文件IO编程

班级: 17计科3

学号: 10417307

姓名: 薛冬冬

东南大学成贤学院计算机系

2020年3月25日

**实验题目 文件IO编程**

1. **实验目的**

1.了解Linux系统下文件读写操作。

1. **实验内容**

1.掌握copy\_file、读写锁执行过程。

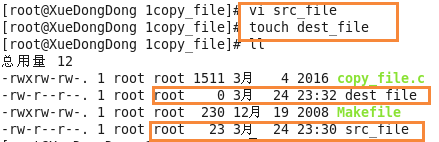
2. 掌握文件IO多路复用以及用文件读写模拟生产者消费者经典问题程序的执行过程。

1. **实验具体要求**

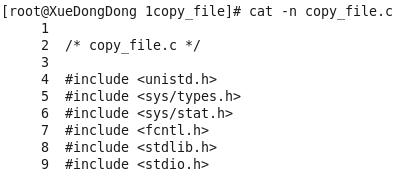
**★1、将copy\_file.c源程序使用GDB单步调试进行追踪，搞清楚这个源程序实际的工作流程。使用截图显示追踪过程，并且在关键位置显示中文说明。**

**答：**

◆创建源文件src\_file和目标文件dest\_file



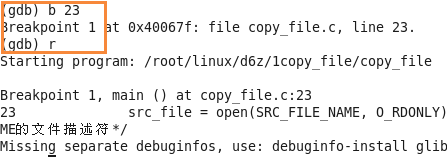
◆另外打开一个终端显示copy\_file.c的源代码



◆进入调试状态，并显示行号



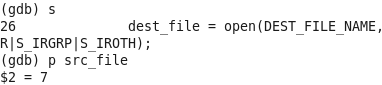
◆设置断点b 23并运行代码至断点



◆查看变量src\_file的值

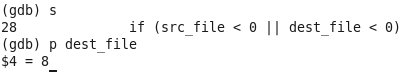


◆运行第23行代码，并查看此时变量src\_file和dest\_file的值

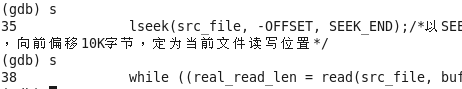




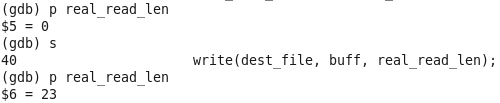
◆运行第26行代码，并查看此时dest\_file的值



◆预判代码运行结果，运行后验证代码正确性



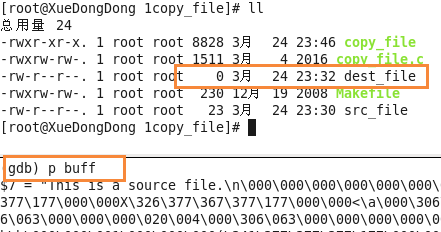
◆先查看real\_read\_len变量值，运行后再查看一下real\_read\_len变量值（变量值应为23，为src\_file文件字节长度）



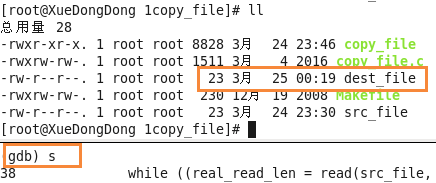
说明此时read()函数已成功将scr\_file中23字节放入buff字符数组内

◆先查看buff变量值和dest\_file文件大小，运行后再查看dest\_file文件大小（应为23，说明成功将源文件中数据复制到目标文件）。

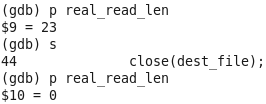
运行前：



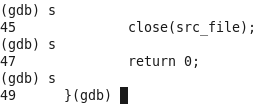
运行后：



◆先查看real\_read\_len变量值，运行后再查看一下real\_read\_len变量值（应为0）



◆关闭文件，释放资源



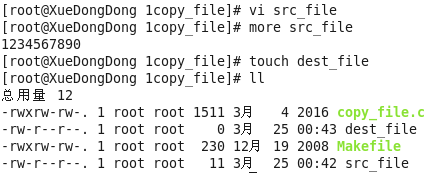
◆查看此时dest\_file文件内容



**★2、根据教材改写copy\_file.c，以只写方式打开目标文件,权限741;将源文件读写指针移动到距离文件开头5字节处，从此处将剩余文件内容读到目标文件中；src\_file源文件内容为：1234567890；需要把题目要求的内容全部展示出来。**

**答：**

◆新建源文件src\_file和目标文件dest\_file



◆分析题目含义

●只写方式打开目标文件,权限741

（1）以只写方式打开文件

O\_WRONLY

（2）权限741

user:读、写、执行 RWX

S\_IRUSR| S\_IWUSR| S\_IXUSR

group:只读 R

S\_IRGRP

other:只能执行 X

S\_IXOTH

●将源文件读写指针移动到距离文件开头5字节处

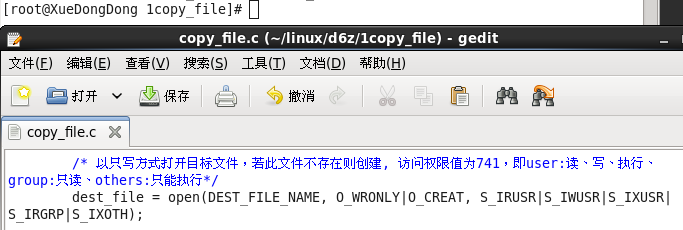
#define OFFSET 5

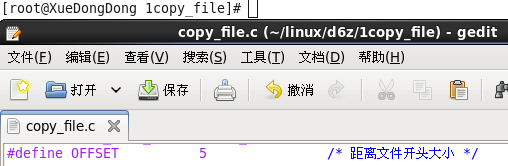
lseek(src\_file, OFFSET,SEEK\_SET);

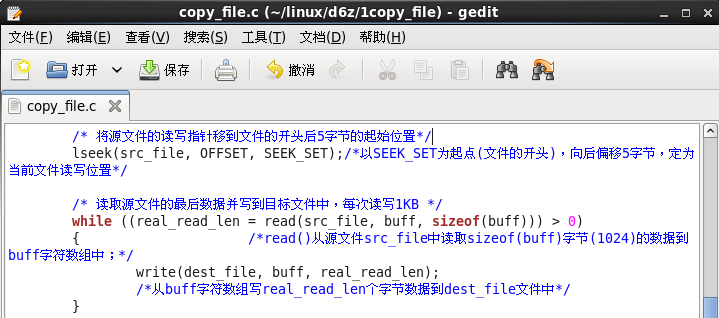
从此处将剩余文件内容读到目标文件中

源代码不变

◆根据题意，改写copy\_file.c



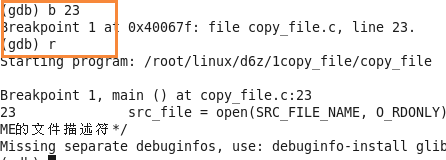




◆进入调试状态，并显示行号



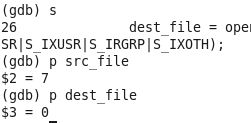
◆设置断点b 23并运行代码至断点



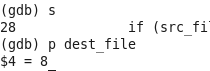
◆查看变量src\_file的值



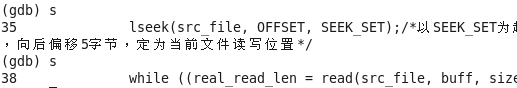
◆运行第23行代码，并查看此时变量src\_file和dest\_file的值



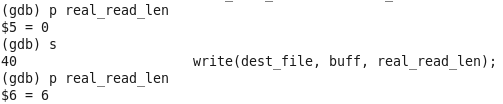
◆运行第26行代码，并查看此时dest\_file的值



◆预判代码运行结果，运行后验证代码正确性



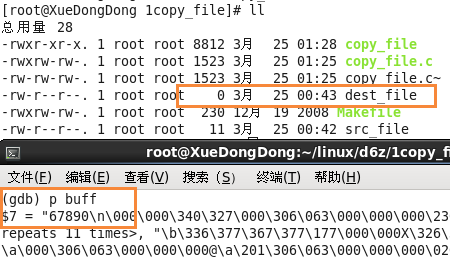
◆先查看real\_read\_len变量值，运行后再查看一下real\_read\_len变量值（变量值应为6，为src\_file文件开头5字节之后数据“67890”的长度）



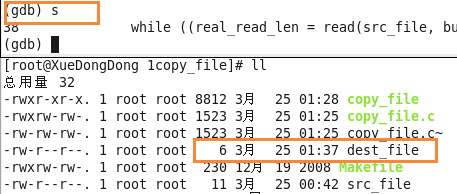
说明此时read()函数已成功将scr\_file中6字节放入buff字符数组内

◆先查看buff变量值和dest\_file文件大小，运行后再查看dest\_file文件大小（应为6，说明成功将源文件中数据复制到目标文件）。

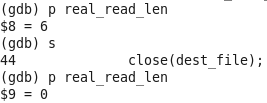
运行前：



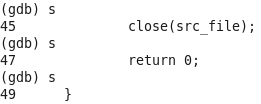
运行后：



◆先查看real\_read\_len变量值，运行后再查看一下real\_read\_len变量值（应为0）



◆关闭文件，释放资源



◆查看此时dest\_file文件内容



**★3、fcntl()给文件上锁需要执行几次，为什么？回答教材fcntl()实例之后的思考题，然后将hello文件上写锁后同时再上一个读锁，效果看我上传的视频，应该怎么做？**

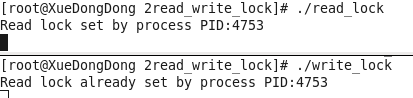
**答：**

（1）fcntl()给文件上锁需要执行两次。

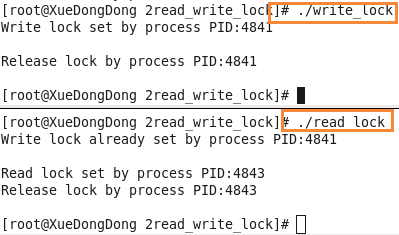
第一次执行：F\_GETLK参数将文件状态设为解锁状态，fcntl()函数返回-1，表达的含义：经过测试该文件可以上锁。

第二次执行fcntl()函数执行可上锁，fcntl()函数返回0，表示上锁成功。

（2）答：在一个终端设置读取锁后，在另一个终端可以进行正常写入锁，且写入锁进程号与读取锁进程号相同。



（3）答：将hello文件上写锁后同时再上一个读锁，效果是两个操作的进程号不相同。可见hello文件被分为了两部分，一部分在进行写锁，另一部分在进行读锁，使得两个操作不冲突，进而达到将hello文件同时上写锁和读锁。



**★4、关于multiplex\_select.c源程序问题：**

**（1）什么是I/O多路复用，用自己的话描述？**

**答：**

I/O多路复用是指使仅供一个进程使用的键盘实现一个键盘同时供多个进程使用，提高键盘的使用效率。

**（2）如何判断文件描述符在fd\_set中？**

**答：**

使用FD\_ISSET(int fd,fd\_set\*set)函数判断，若文件描述符在fd\_set中，则返回非0值。在本次实验中为：FD\_ISSET(fds[0],&inset)|| FD\_ISSET(fds[1],&inset) || FD\_ISSET(fds[2], &inset)。

**（3）如何让程序执行case 0: 分支语句？**

**答：**

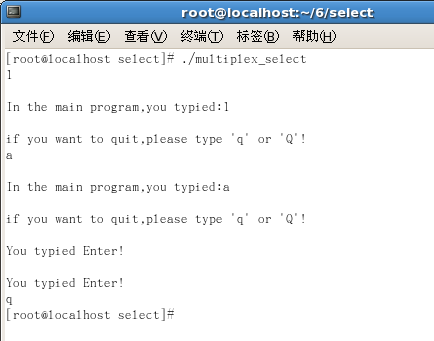
超过等待时间timeout未检测到任何文件描述符准备好，超时，select()函数返回值为0，执行case 0。

**（4）程序代码第81行tmp\_inset = inset; 去掉这行代码行不行，为什么？**

**答：**

不行，这段代码将文件描述符集赋值给另一个文件描述符集，进行文件描述符集合的备份，避免inset每次进行初始化。

**（5）改写multiplex\_select.c源程序，使之运行呈现以下效果：**



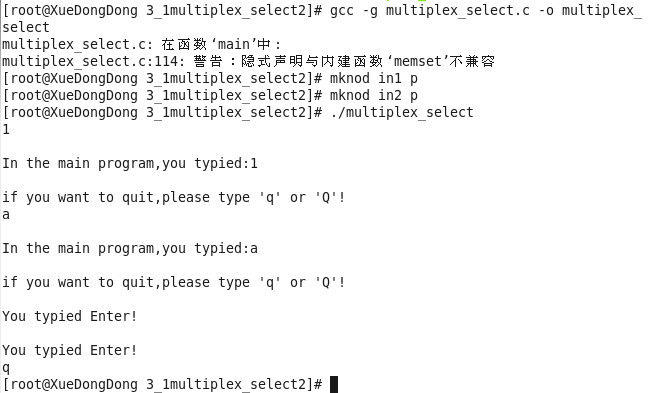
前两个测试用例是分别敲入1和a之后点击回车的效果；You typied Enter是连续敲入两次回车之后的效果，最后一个测试用例是敲入q之后点击回车的效果，程序直接退出；按照图示改写程序，要求展示改动的代码以及编译运行的效果。

**答：**

代码更改位置：



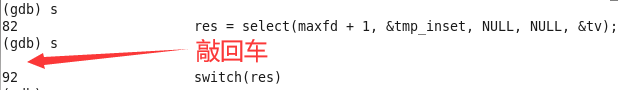
编译运行效果图：

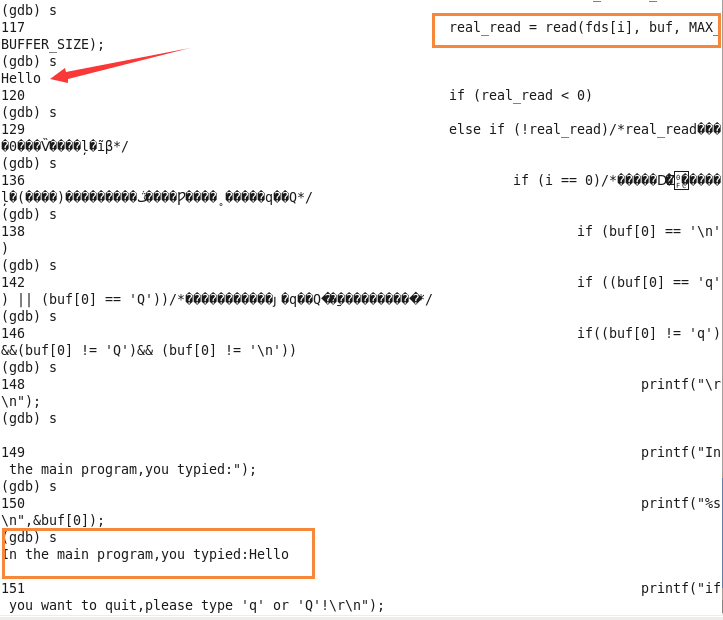


**（6）在调试状态下**，select函数很容易监测到管道文件in1和in2的字符输入；但是主程序./multiplex\_select处于运行状态时的字符输入，gdb下的select函数却监测不到？这很奇怪，因为主程序的输入字符也是由标准输入文件键盘输入的，而初始状态键盘fds[0]也存在于文件描述符集tmp\_inset当中，select函数理应能监测到，而事实上却监测不到，**这是为什么？**

**答：**

使用gdb调试，也就相当于正在运行一个multiplex\_select程序，再打开一个终端进行./ multiplex\_select对它没有任何影响。在运行的过程中，在res = select(maxfd + 1, &tmp\_inset, NULL, NULL, &tv);位置，直接敲回车跳过即可，等程序运行到real\_read = read(fds[i], buf, MAX\_BUFFER\_SIZE);这段代码时再敲入自己要打印的字符，这个地方输入字符的操作就相当于之前在终端执行./ multiplex\_select后输入字符，功能是从标准输入文件键盘输入字符并将其读取至buff字符数组中。具体操作如下图：

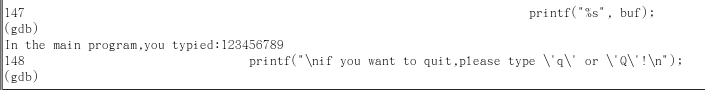




上面的问题比较难回答，下面给出一个稍容易点的问题：

**在调试环境下，select函数如何能监测到标准输入文件（键盘）的字符输入？**形如下图所示：

**gdb监测到标准输入文件的字符输入**



**答：**

res = select(maxfd + 1, &tmp\_inset, NULL, NULL, &tv);

select()函数执行成功后会返回准备好的文件描述符的数目，即res非0和非-1，也就是表示某管道或标准输入文件有数据输入。再使用read()函数读取fd[i]中的有效数据（real\_read = read(fds[i], buf, MAX\_BUFFER\_SIZE);），在存在有效数据的情况下（real\_read>0），若i==0则监测到是标准输入文件的字符输入。

**（7）select()函数到底是做什么用的？**

**答：**

select函数的功能就是用来检测多个文件描述符，一旦有文件描述准备好，那么该函数就返回。成功返回大于0。失败返回-1。超时返回0。

**★5、标准I/O函数和系统调用函数有什么区别？**

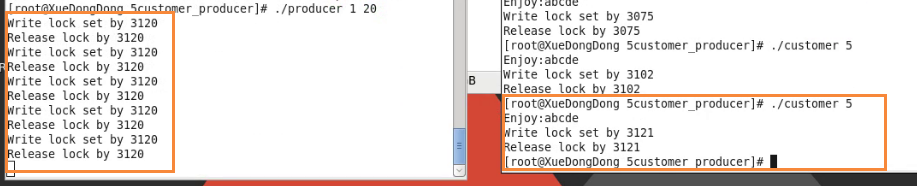
**答：**

标准I/O函数提供缓冲存储，而系统调用函数不提供缓冲存储。

标准I/O函数基于流缓冲，而系统调用函数基于文件描述符。

**★6、有关课后生产者程序**：**./producer 1 20，代表每隔一秒生产一个字符，总共生产20个字符；消费者程序./customer 5代表消费5个字符；生产者进程、消费者进程同时进行；在生产者进程生产完5个字符后，在另一个终端的消费者进程就可以消费5个字符了，请问消费者为什么能消费abcde5个字符？因为生产者生产字符时是给myfifo文件上写锁的，而写锁是排他锁，它锁住时，其他进程是不能加锁的。所以在生产者还未生产完字符时，消费者进程应该消费不了字符，为什么实际执行时能消费字符？？？具体的视频效果请浏览我上传视频的49分30秒位置查看。还有，myfifo文件里面的abcde5个字符是如何被消费者进程消费的，关键代码是什么？**

**答：**



（1）根据我抓取截图可见，消费者是在生产者生产完第5个字符后立刻消费的，因为product()函数的功能是生产一个字符并写入到仿真FIFO文件中，所以在每生产完一个字符后都会进行解锁（即上方截图中Release lock），因此在【第5个字符生产完成后myfifo文件被解锁~准备第6个字符生产即再次给myfifo文件上写锁】这期间消费者进行了消费。

此时虽然生产者还未生产完字符，但生产者是每生产一个字符就存入myfifo文件然后解锁，这就使得消费者可以在满足消费资源即5个字符且myfifo文件处于解锁状态下进行消费。消费者消费的进程号和生产者生产进程号不同，可见myfifo文件当时被分为了两个部分，一部分在进行写锁一部分在进行读锁，使得两个操作不冲突。

（2）myfifo文件里面的abcde5个字符被消费者进程消费的关键代码是：customing(fifo\_file, need);即customing()函数内fputc(buff, stdout);这句代码。

**借助GDB调试工具，仿照视频的演示，搞懂生产者消费者程序执行的流程，并用自己的话描述两个源程序执行的关键步骤。**

**答：**

producer.c

流程：

开始->生产一个资源->上锁->将生产的字符写入仿真FIFO文件->解锁->判断是否生产完->若生产完直接结束，反之等待1秒，继续执行生产一个资源直至生产完

关键步骤：

生产一个资源并将文件上锁，上锁后将生产的字符写入FIFO结构文件，然后将文件解锁，判断资源是否生产完。

customer.c

流程：

开始->消费资源->判断消费的资源是否够->不够继续消费，反之进行上锁->将剩余数据拷贝至临时文件tmp中->用临时文件tmp覆盖原数据文件，模拟FIFO结构->解锁->删除临时文件->结束

关键步骤：

判断现有资源是否满足消费，若不满足继续消费，反之将文件进行上锁，然后将剩余的数据拷贝至临时文件tmp，再使用临时文件覆盖原数据文件，这样就可以实现文件的消费，然后解锁，接着删除临时文件。

1. **实验感悟**

第六周实验相较于前几次实验，多了更多的思考，需要思考如何更改源文件和hello文件同时上写锁和读锁却不冲突的原因。本次实验的前两个实验，使我更好地理解了gdb调试的好处，复习了gdb调试相关的知识，当然，最重要的是学会了文件的I/O编程：文件的打开、关闭、读取数据、写数据和文件指针的定位。第三个实验使我了解并掌握了文件锁相关知识：读取锁、写入锁、解锁。

第七周实验整体看来需要较为仔细的分析，大部分题目没有固定的答案，需要自己根据资料去总结对于问题自己的看法，还需要经常思考为什么，需要我们深入透彻理解才能回答得出来。本章实验的重点就是生产者消费者程序，通过老师的讲解和资料，使我知道了程序的执行流程和两个.c源程序中每条语句的含义，使我较好掌握了select()函数的用法。

赵老师在视频中对源代码的讲解清晰易懂，再加上我记笔记两层功效，使我对源代码的理解更为透彻，在做实验第二小题时思路清晰明确，很快就完成了。和以往一样，赵老师在视频中将程序一步一步调试给我们看，使得我在自己动手操作时极为顺利，并没有遇到什么需要费脑筋解决的问题。第七周实验花费时间最多的还是第4条的第6小条，读完题目后有些迷，题目问为什么，不知道刚开始该从哪里入手研究，只好将代码读了一遍又一遍，然后再使用gdb进行调试，尽自己最大的努力回答了题目中的为什么。

在结合了理论和实践后，使我对知识的理解和掌握更为牢固。在完成实验后，还增加了自己的成就感，赵老师的这种教学方式挺赞。