1. 实验目的：理解进程间通信，掌握常用进程间通信的方式；通过采用共享内存的方式实现文件复制来加深对进程间通信的理解。

2. 实验目标：

（1）学习使用管道命令 | 。使用ls –l | more 和 cat 一个大一点的文件 | more 等命令加深对管道的理解。

（2）通过共享内存机制实现文件复制（建议使用mmap）。

自己的扩展目标：

（1）学习过两个进程之间的文件复制之后，同学们可以试着用多个进程实现文件的复制过程。

3.实验原理：

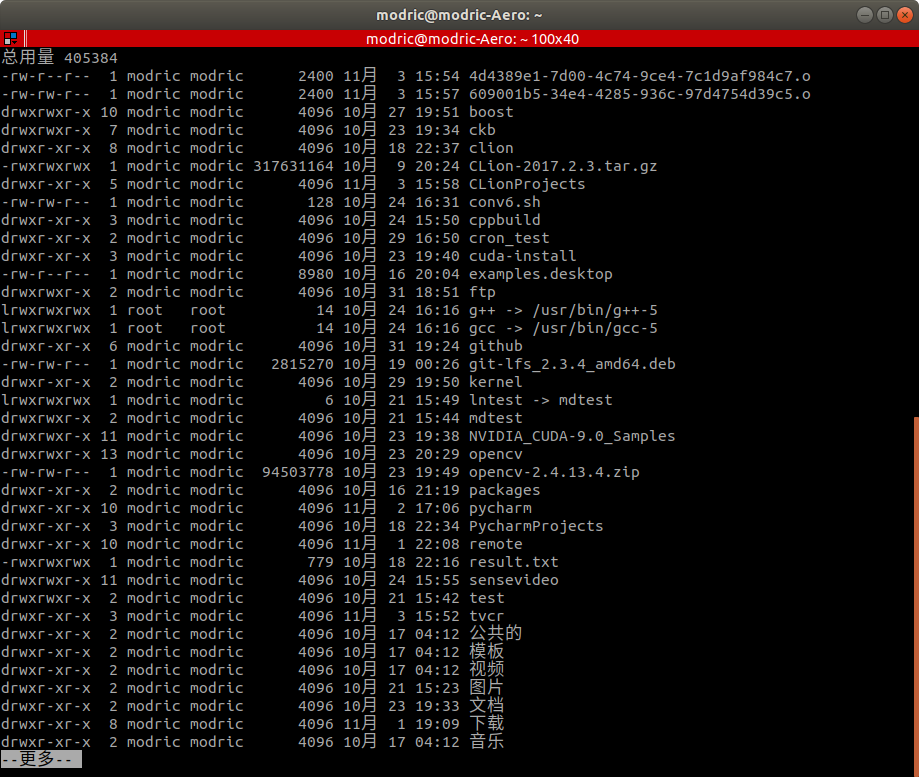
（1）管道命令非常实用，可以对前一个命令的输出进行进一步操作。例如，使用apt list命令可以看到apt仓库的软件清单，但是一行一行的找非常麻烦。此时使用管道命令结合grep，就可以快速找到想要安装的软件的名字。

| more 命令可以对前一个命令输出的内容进行分页浏览。在前一个命令输出内容非常多，可能会超出缓冲区的时候，| more 命令非常实用。

（2）mmap可以将文件或其他内容映射进内存。可以设定映射区域为多进程共享，实现多进程间的资源共享。

4.实验方案：

(1) ls -l | more 直接执行即可

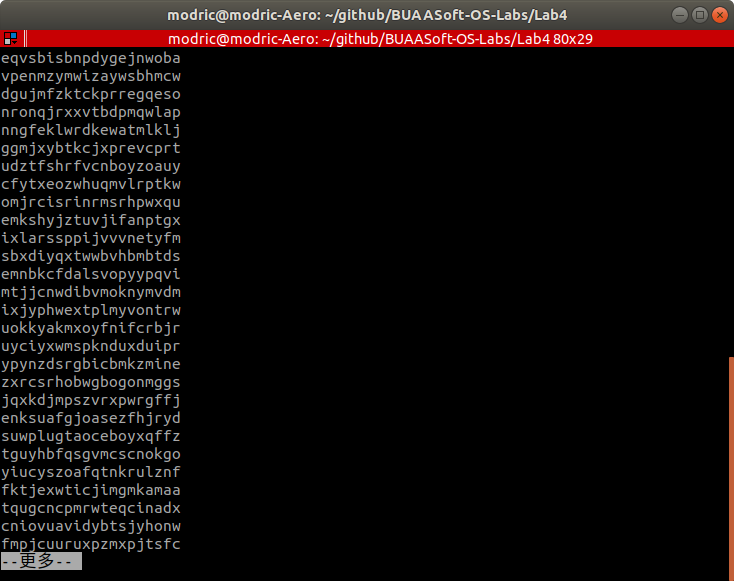


可以使用空格键下翻

先使用如下代码生成一个大文件：

#include **<stdio.h>**#include **<stdlib.h>**#include **<time.h>  
  
int** main() {  
 freopen(**"data.txt"**, **"w"**, stdout);  
 srand(time(0));  
 **const int** lines = 5000;  
 **const int** lenth\_per\_line = 20;  
 **for** (**int** i = 0; i < lines; i++) {  
 **for** (**int** j = 0; j < lenth\_per\_line; j++) {  
 printf(**"%c"**, (**char**) (rand()%26 + **'a'**));  
 }  
 printf(**"\n"**);  
 }  
}

然后对生成的文件执行cat data.txt | more



可以看到，确实起到了分页浏览的效果

（2）利用mmap实现文件拷贝的代码如下(data\_gen用于生成一个大文件)：

#include **<stdio.h>**#include **<stdlib.h>**#include **<string.h>**#include **<unistd.h>**#include **<sys/mman.h>**#include **<sys/stat.h>**#include **<fcntl.h>**#include **<time.h>  
  
void** data\_gen() {  
 FILE \*f = fopen(**"in.txt"**, **"w"**);  
 srand(time(0));  
 **const int** lines = 5000;  
 **const int** lenth\_per\_line = 20;  
 **for** (**int** i = 0; i < lines; i++) {  
 **for** (**int** j = 0; j < lenth\_per\_line; j++) {  
 fprintf(f, **"%c"**, (**char**) (rand()%26 + **'a'**));  
 }  
 fprintf(f, **"\n"**);  
 }  
 fclose(f);  
}  
  
**int** main() {  
 data\_gen();  
 **int** file\_in = open(**"in.txt"**, O\_RDONLY);  
 **int** file\_out = open(**"out.txt"**, O\_RDWR | O\_CREAT, ~0);  
  
 **struct** stat stat\_file\_in;  
 fstat(file\_in, &stat\_file\_in);  
  
 lseek(file\_out, stat\_file\_in.st\_size - 1, 0);  
 write(file\_out, **""**, 1);  
  
 **void** \*src = mmap(NULL, stat\_file\_in.st\_size, PROT\_READ, MAP\_SHARED, file\_in, 0);  
 **void** \*dst = mmap(NULL, stat\_file\_in.st\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, file\_out, 0);  
  
 memcpy(dst, src, stat\_file\_in.st\_size);  
  
 munmap(src, stat\_file\_in.st\_size);  
 munmap(dst, stat\_file\_in.st\_size);  
  
 close(file\_in);  
 close(file\_out);  
  
 **return** 0;  
}

原理是将源文件和目标文件都映射到内存，随后对映射的内存区域做内存拷贝，实现文件的复制。需要注意的是，由于映射空间的内存大小是根据文件的大小决定的，因此需要先将目标文件的大小扩大到源文件的大小，否则会出错。

扩展内容：

我认为这里是让我们了解学习mmap能被多个进程共享这一点。我的操作是，完成mmap后进行fork，父子进程各执行一半的memcpy，来验证mmap的多进程共享。

代码如下(data\_gen用于生成一个大文件)：

#include **<stdio.h>**#include **<stdlib.h>**#include **<string.h>**#include **<unistd.h>**#include **<sys/mman.h>**#include **<sys/stat.h>**#include **<fcntl.h>**#include **<time.h>**#include **<wait.h>  
  
void** data\_gen() {  
 FILE \*f = fopen(**"in.txt"**, **"w"**);  
 srand(time(0));  
 **const int** lines = 500000;  
 **const int** lenth\_per\_line = 20;  
 **for** (**int** i = 0; i < lines; i++) {  
 **for** (**int** j = 0; j < lenth\_per\_line; j++) {  
 fprintf(f, **"%c"**, (**char**) (rand()%26 + **'a'**));  
 }  
 fprintf(f, **"\n"**);  
 }  
 fclose(f);  
}  
  
**int** main() {  
 data\_gen();  
 **int** file\_in = open(**"in.txt"**, O\_RDONLY);  
 **int** file\_out = open(**"out.txt"**, O\_RDWR | O\_CREAT, ~0);  
  
 **struct** stat stat\_file\_in;  
 fstat(file\_in, &stat\_file\_in);  
  
 lseek(file\_out, stat\_file\_in.st\_size - 1, 0);  
 write(file\_out, **""**, 1);  
  
 **void** \*src;  
 **void** \*dst;  
 src = mmap(NULL, stat\_file\_in.st\_size, PROT\_READ, MAP\_SHARED, file\_in, 0);  
 dst = mmap(NULL, stat\_file\_in.st\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, file\_out, 0);  
  
 pid\_t pid;  
 pid = fork();  
 \_\_off\_t mid = stat\_file\_in.st\_size/2;  
  
 **if** (pid > 0) {  
 memcpy(dst, src, mid);  
 } **else** {  
 memcpy(dst + mid, src + mid, stat\_file\_in.st\_size - mid);  
 exit(0);  
 }  
 waitpid(pid, NULL, 0);  
  
 munmap(src, stat\_file\_in.st\_size);  
 close(file\_in);  
 munmap(dst, stat\_file\_in.st\_size);  
 close(file\_out);  
 **return** 0;  
}

5.实验结论：

在linux系统中，管道命令可以让用户方便的处理各类输出，结合more/grep等命令使用，功能非常强大。

mmap可以把文件或其他内容映射到内存区域，随后可以像操作内存一样操作这些内容，非常方便。mmap还支持进程间共享。