

暨 南 大 学

本 科 生 课 程 论 文

课程名称 Linux高级编程

学 院 智能科学与工程学院

学 系

专 业 人工智能

姓 名 罗杨

学 号 2021101353

指导教师 李军

二○二四年 六月一日

目录

[1. Linux安装 3](#_Toc169031374)

[2. 常用命令 5](#_Toc169031375)

[3. 常用开发工具 8](#_Toc169031376)

[4. 用户与用户组管理 9](#_Toc169031377)

[5. 分区与文件系统 10](#_Toc169031378)

[6. Shell编程-程序结构、变量、运算、if语句 12](#_Toc169031379)

[7. Shell编程-分支、循环 14](#_Toc169031380)

[8. 文件编程 16](#_Toc169031381)

[9. Linux进程管理 19](#_Toc169031382)

[10. Linux信号 21](#_Toc169031383)

[11. 匿名通信 24](#_Toc169031384)

[12. 进程、线程对比 25](#_Toc169031385)

[13. 信号量实现生产者-消费者模型 27](#_Toc169031386)

[14. 信号量实现生产者-消费者模型 28](#_Toc169031387)

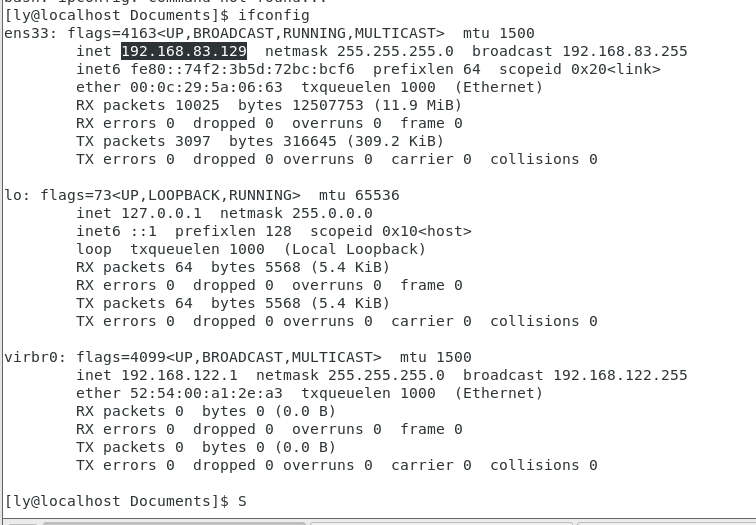
1. Linux安装（Rocky9.5）

在VMware中新建虚拟机，选中提前准备好的Rocky9.5光盘映像文件，跟着安装向导安装即可。安装完成进入Linux桌面：

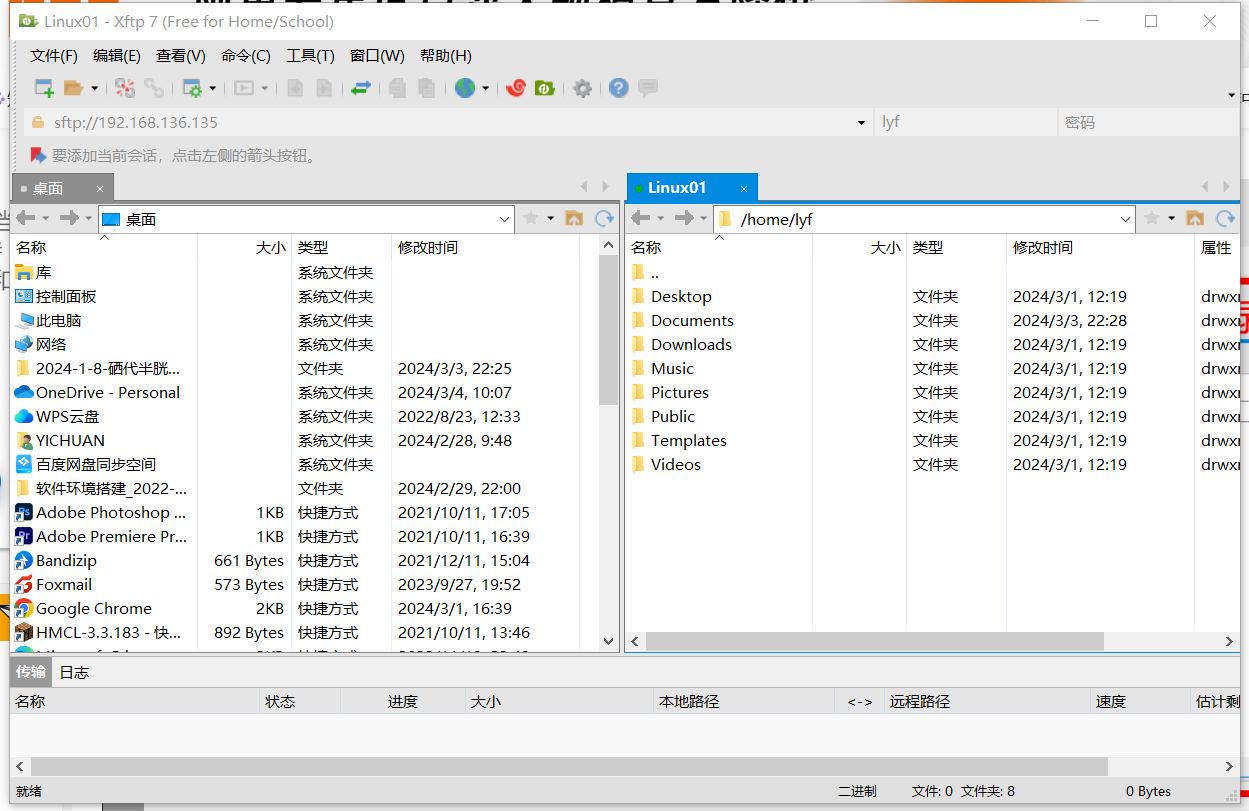
图形用户界面, 应用程序

AI 生成的内容可能不正确。

在Windows上安装Xftp并传输文件, 使用ifconfig查找虚拟机IP地址：



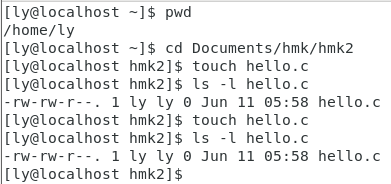
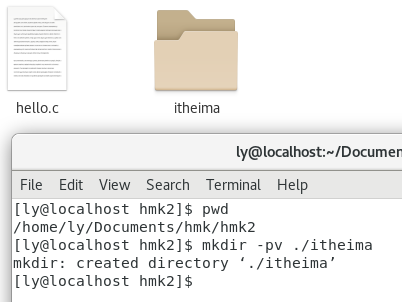
连接Windows主机和Linux虚拟机。可以直接拖动文件进行传输，如将text02拖入Linux01的documents文件夹中



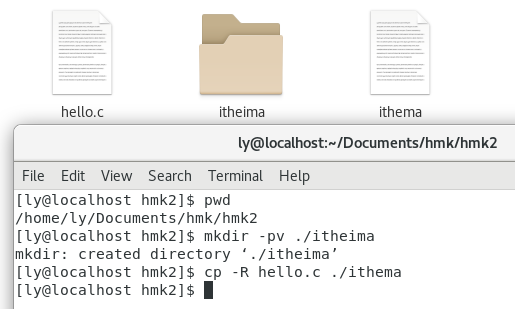
1. 常用命令

文件操作命令

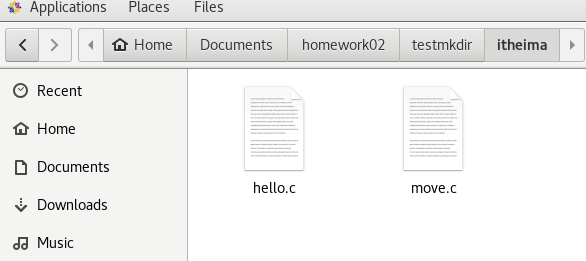
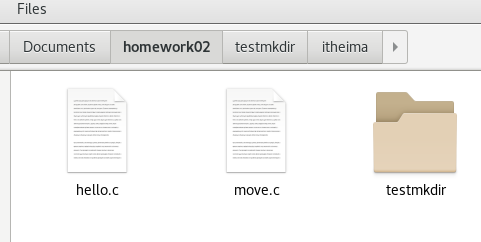
在这里我先新建了一个文本文件hello.c，然后再用touch命令更新，可以看到其时间标签发生了变化。使用mkdir命令创建了一个testmkdir以及它的子目录itheima

使用Cp命令将一个或多个源文件复制到指定目录。可以看到这里将hello.c文件复制到了itheima文件夹中。

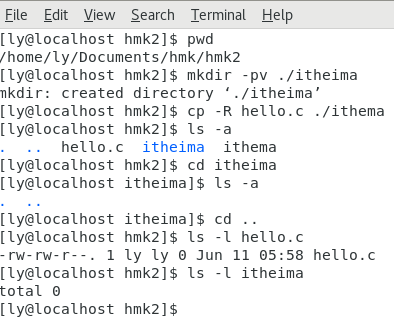
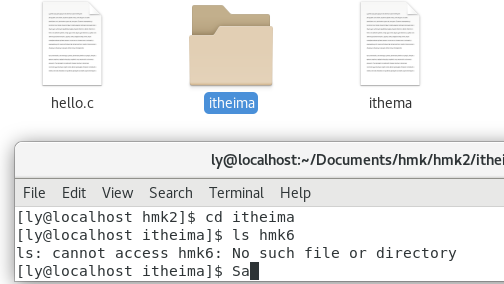


使用Mv命令移动文件或目录。将homework02中的move.c移动到了itheima中。



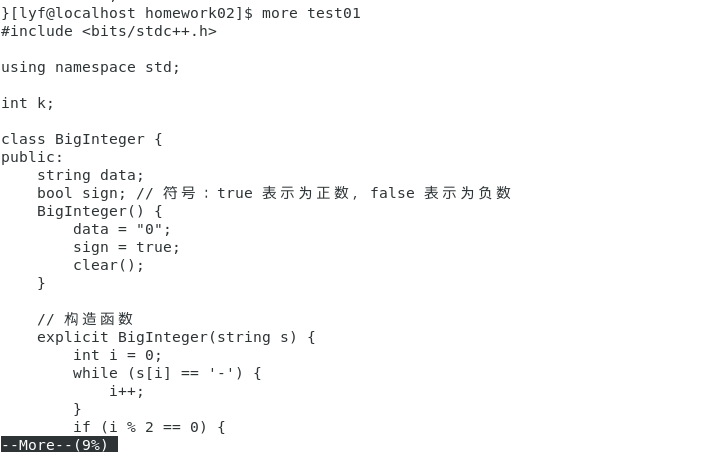
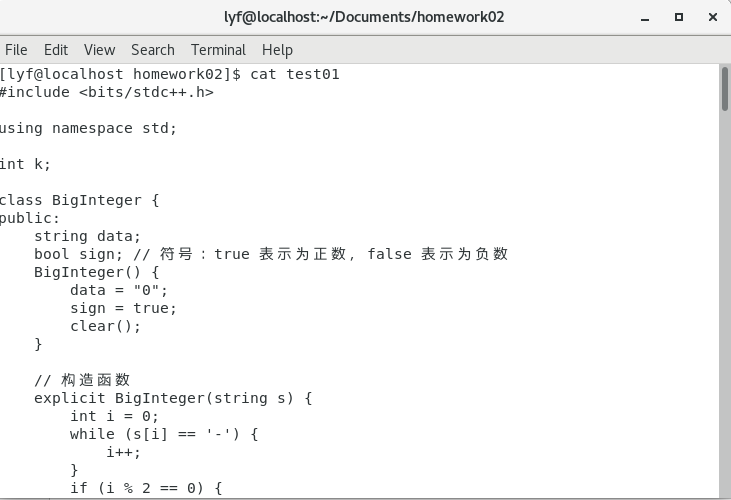
用ls查看文件夹和文件状态。使用RM命令：删除目录中的文件或目录。

这里删除了testmkdir文件夹以及其中的所有文件夹和文件。

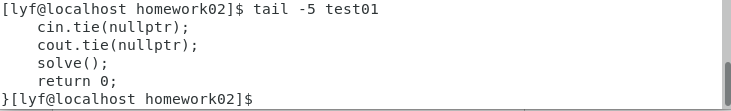
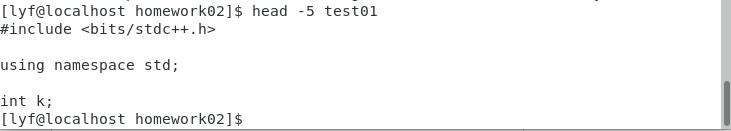
 

文件查看命令

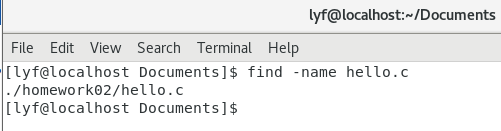
Cat命令：打印文件内容到输出设备。More命令：分页显示文件内容。f/Space显示下一页Enter显示下一行，q/Q退出

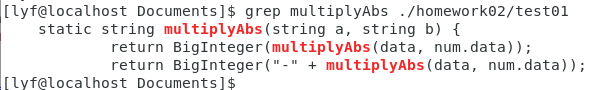


Head命令：查看文件的前n行。Tail命令：查看文件的后n行。



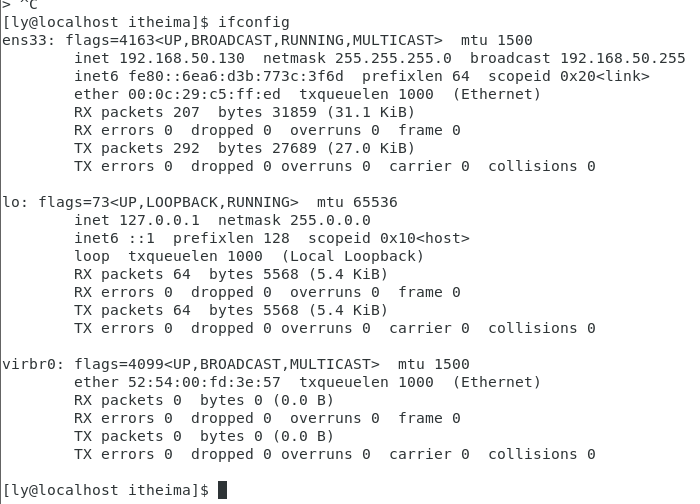
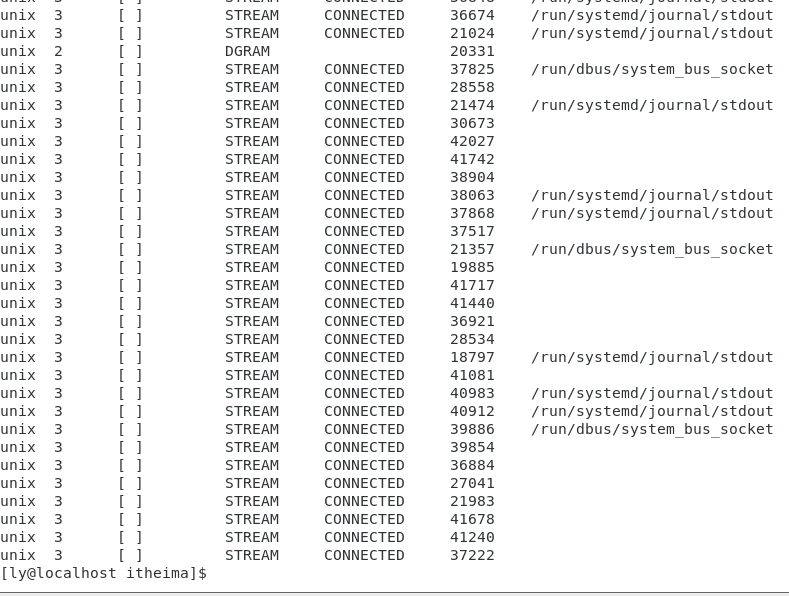
Find命令：借助搜索关键字（文件名、文件大小、文件所有者等）查找文件或目录。Grep命令：在文件中搜索与字符串匹配的行并输出。



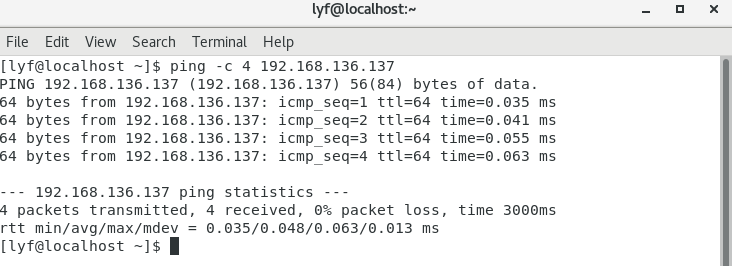


网管命令

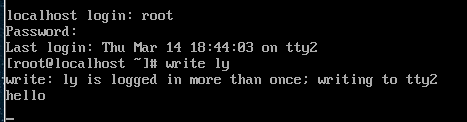
Ifconfig命令：配置和显示Linux内核中网络接口参数。Netstat命令：打印Linux系统中网络系统的状态信息。

Ping命令：测试主机之间网络的连通性。

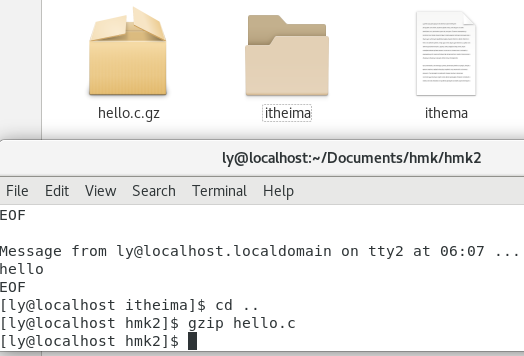
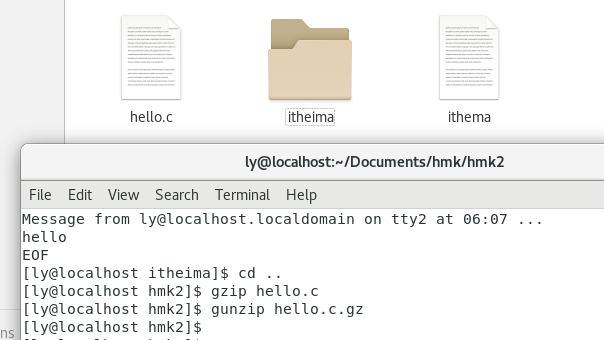


Write命令：使当前用户向另一个用户发送信息

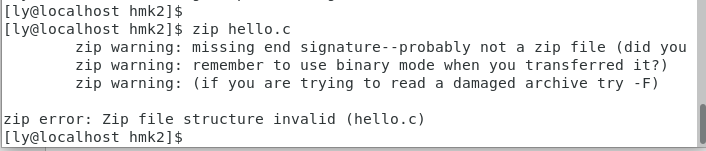
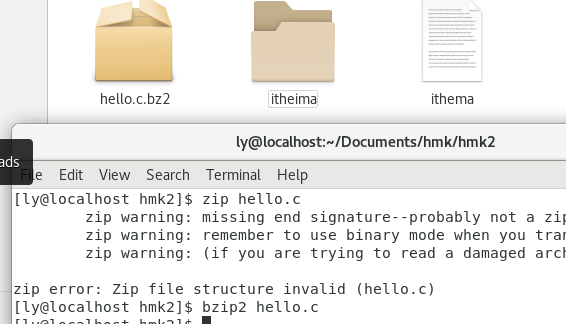
 

压缩解压命令

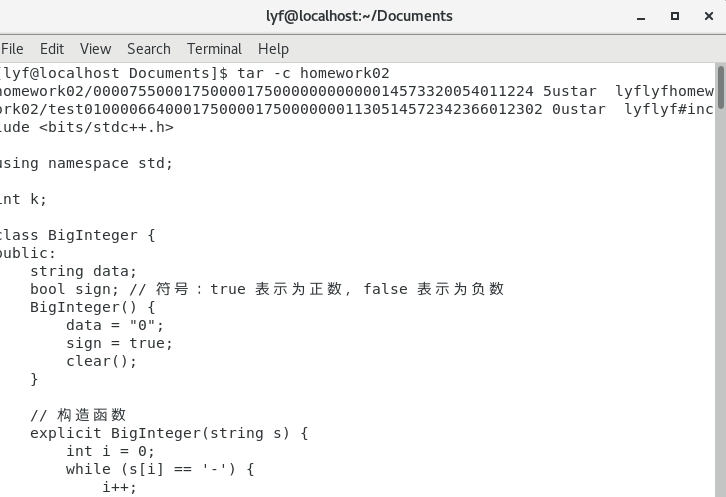
gzip/gunzip命令：压缩文件，获得.gz格式的压缩包，压缩后不保存源文件，若同时列出多个文件，则每个文件会被单独压缩。

Zip/unzip命令：压缩文件，获得.zip格式的压缩包，压缩后保存源文件bzip2/bunzip2命令：压缩文件，获得.bz2格式的压缩包，使用选项-k时保留源文件

Tar命令：打包多个目录或文件，通常与压缩命令一起使用。

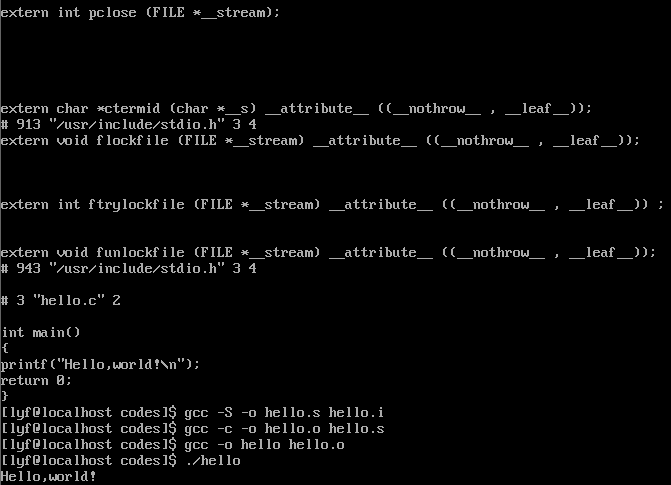


1. 常用开发工具

单文件编译首先进入vi编译器，打开我们要编写的文件hello.c，进入插入模式使用：wq保存退出。

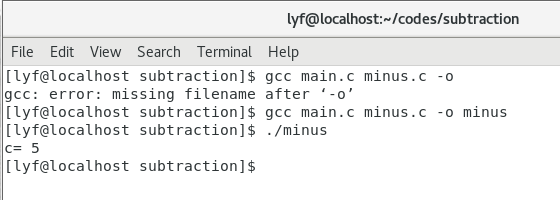
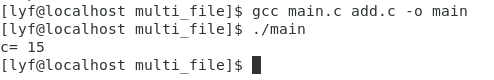
cdc8b21906e73ed050010cdb5ec1bab

gcc编译：



多文件编译

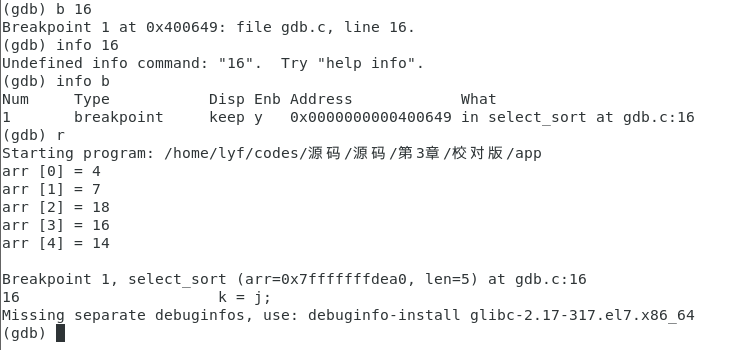
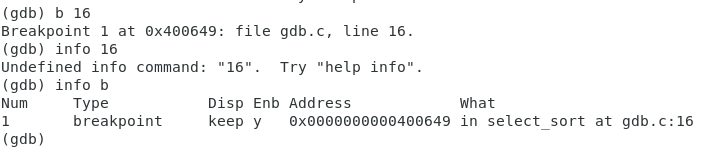
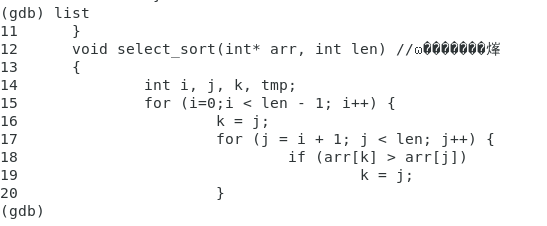
Add文件编译运行，Minus文件编译运行



启用gdb调试



显示行号、设置断点

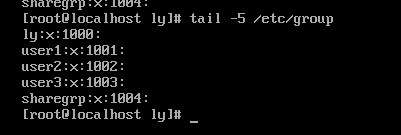
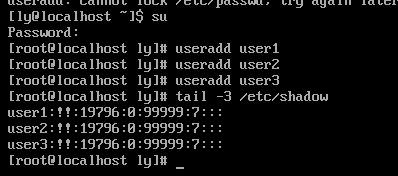


分析程序：发现在select\_sort函数中需要跟踪的变量只有3个,即i、j、k。使用 display命令跟踪这三个变量。在程序执行的过程中,变量k的值一直为0,而正常情况下k应保存外层循环i的值,因此可以判断k的赋值应该有问题。观察代码,发现代码第19行应为k=i。

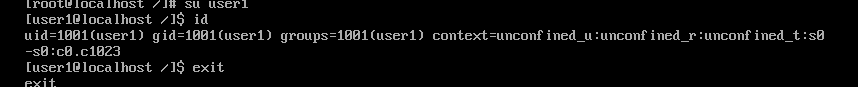
1. 用户与用户组管理

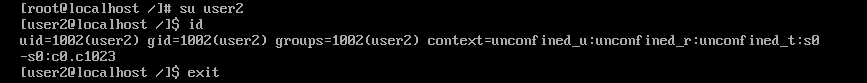
1. 创建用户、用户组：

进入root，使用’adduser’添加三个用户，并使用tail命令查看，使用’groupadd’创建用户组：

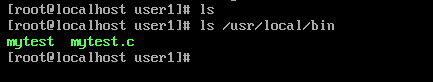
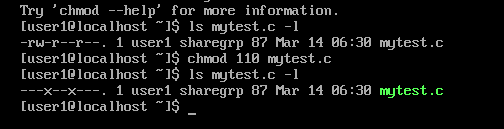


usermord user1 -g sharegrp ,-G修改附加组把它作为user1,user2的附加组，使用id查看用户组信息：

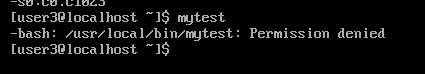
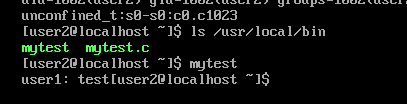




查看并修改mytest.c文件属性，将mytest, mytest.c文件移动到指定目录中，注意需要在root模式下使用mv命令

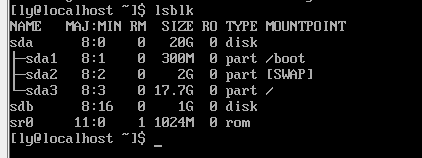
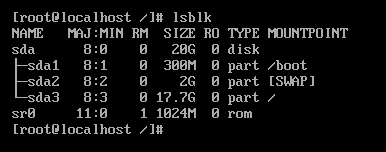


因为当前用户是user1，对于mytest文件有执行权限，所以在根目录下直接运行并使用user2，user3运行程序：



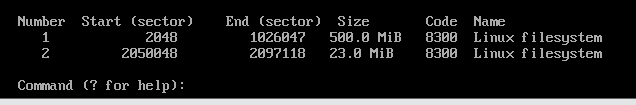
1. 分区与文件系统

启动linux后，用lsblk命令查看机器上的磁盘（块设备),截图。然后关闭linux（命令为shutdown -h now）1. 在安装了linux的虚拟机上，再添加一块虚拟硬盘。重新启动linux之后，再用lsblk命令查看磁盘，观察机器上新增的硬盘，观察它的名称

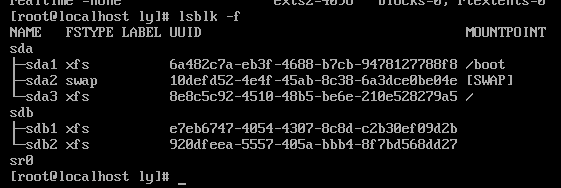


添加硬盘后发现新增sdb硬盘，大小为1G。

用gdisk命令把新添加的硬盘分成2个区，观察它们的名称是什么？

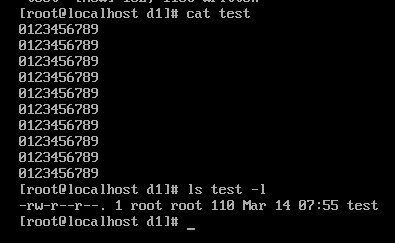
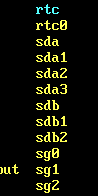


用mkfs.xfs命令，把上面的2个分区格式化为XFS文件系统lsblk 命令可以列出所有可用的块设备的信息，包括它们的文件系统类型，-f 选项告诉 lsblk 显示文件系统信息。输出如下：



查看目录/dev下面是否有上述两个分区，然后把其中一个分区挂载到/mnt目录下。在/mnt目录下，建立一个目录d1,用ls命令查看其大小；然后在d1里建立一个100字符左右的文件，用ls命令再次查看d1的大小；

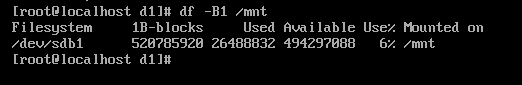
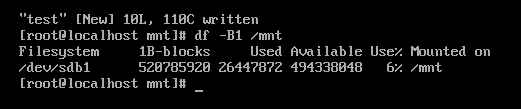
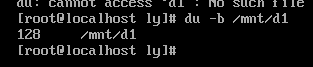




对比发现d1目录大小无显著变化，这是因为当使用 ls -l 命令时，显示的通常是文件的“大小”字段，它表示的是文件的内容所占用的字节数。但是，对于目录而言，这个“大小”字段实际上显示的是目录本身元数据的大小，而不是目录内所有文件内容的总大小。

在 d1 目录中创建一个约100字符的文件。由于字符文件的实际大小通常比字符数要大，所以文件实际占用的空间可能远大于100字节。但是，即使文件的大小超过100字节，d1 目录本身的大小可能仍然不会有显著变化，因为目录的大小主要与其内部文件和子目录的引用有关，而不是这些文件内容的大小。

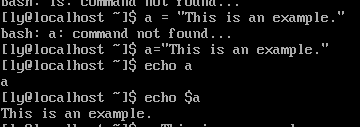
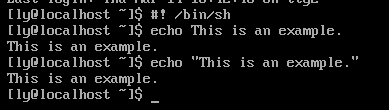
用du(加-b表示以字节为单位显示)显示d1目录的大小，这个命令将会递归地计算 /mnt/d1 目录下所有文件和子目录的大小，并以字节为单位显示。输出将是一个数字，表示 d1 目录及其所有内容的总大小。用df命令(加-B1选项表示以1byte为单位)显示d1所在分区已使用空间；然后删除d1,再令查看该分区的已使用空间，前后对比说明新建的文件确实在你新添加的硬盘上。



前后对比发现使用空间由26388832B变为了26447872B，因此文件添加到了新硬盘上。

1. Shell编程-程序结构、变量、运算、if语句

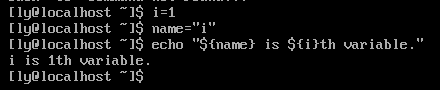
echo同时输出多变量输出echo This is an example和echo "This is an example." 将句子赋值给变量a



结论：在没有变量替换的情况下，单引号和双引号的输出效果是一样的。

当涉及到变量替换时，双引号允许变量扩展，而单引号不会。这意味着在双引号中，变量会被替换为其值，而在单引号中，变量名会被原样输出。

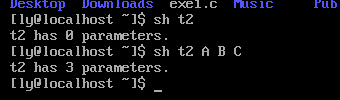
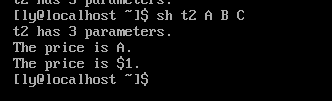
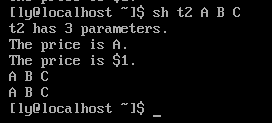
变量引用echo "${name} is ${i}th variable." 会输出"i is 1th variable."这样的效果。



${i}th 表示变量’i’后面加上字符串’th’，若没有花括号则表示ith，会返回空值。

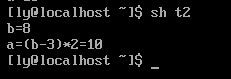
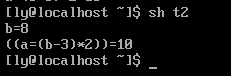
调用参数

编写一个脚本，假设它的文件名为t2,执行时给它几个参数，假设为3，(1) 在t2中添加echo "The price is $1.", echo 'The price is $1.'添加echo $\*, echo $@。

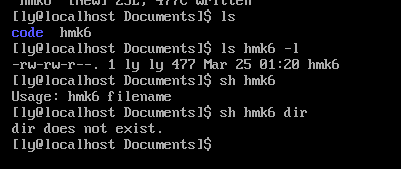
  

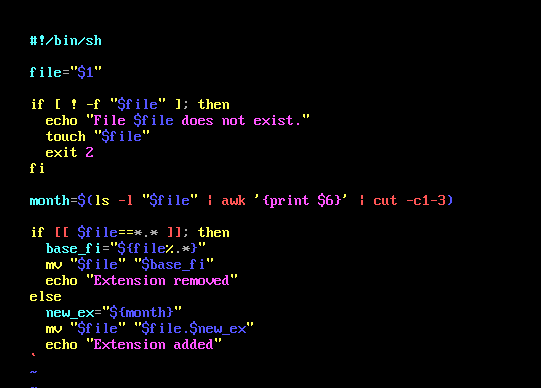
发现输出没有区别

算术运算：

用$(comand）执行命令并返回该命令的输出。没有生成目录时的结果和创建fi文件后的结果：





1. Shell编程-分支、循环

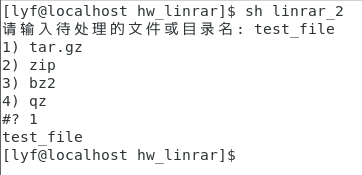
编写脚本代码如下：



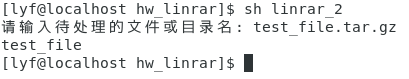
创建测试文件



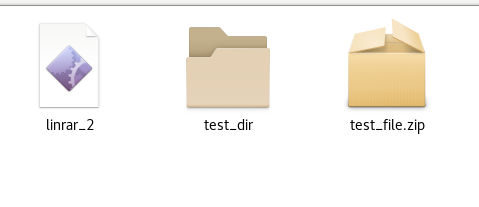
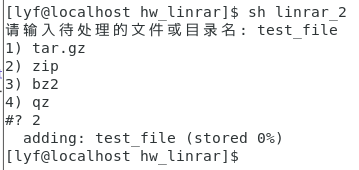
压缩/解压文件

（1）tar.gz压缩结果和解压tar.gz格式压缩文件结果：

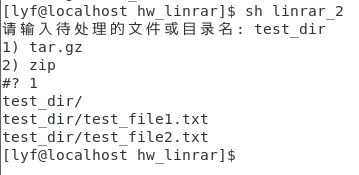


1. zip压缩结果

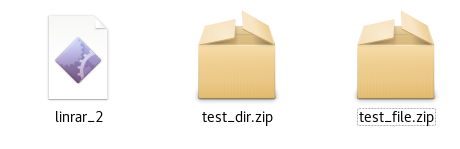
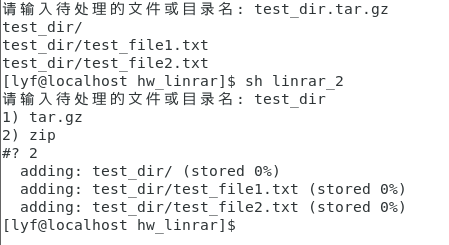


1. 压缩/解压目录

（1）tar.gz格式压缩和压缩结果：

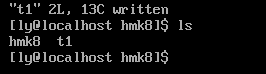


（2）zip格式压缩和解压结果：



1. 文件编程

在命令行运行echo a1=10 >t1;echo a2=-10 >> t1.请你说明此命令的含义

这个命令的含义是创建一个文件t1，然后将字符串"a1=10"写入文件t1中，并覆盖文件原有内容；再将字符串"a2=-10"追加到文件t1的末尾。

用vi打开t1,然后在命令模式下，输入命令!xxd,让vi以16进制显示该文件内容。(!xxd -r可返回正常显示)

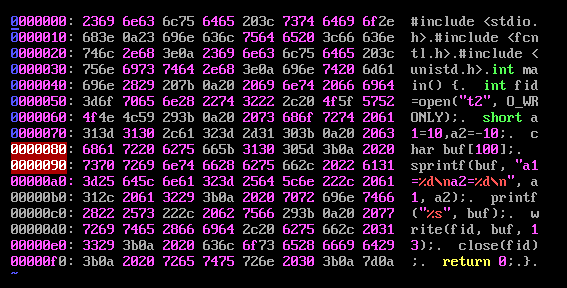


请你说明6131..3130 0a这些16进制数分别表示什么含义。可以借助附件中的ASCII码表。该文件占13字节的空间，为什么？

61表示a，31表示1，以此类推。所以这些十六进制数就是a1=10 a2=-10。

它们字符串本身占11个字节，后面还有两个\n，所以一共13个字节。

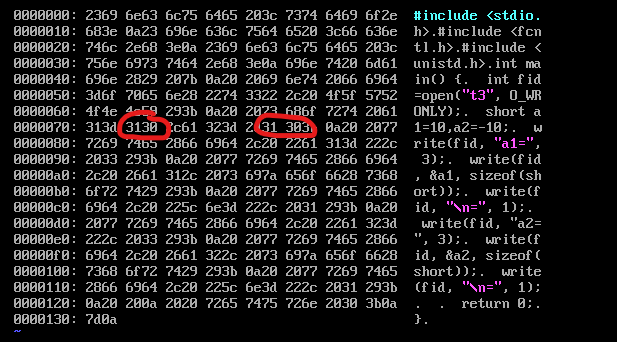
生成文本文件。用touch命令建立新的空文件t2,然后编写下面的程序，它编译运行后，在16进制下查看，t2与t1的内容一样吗？

****

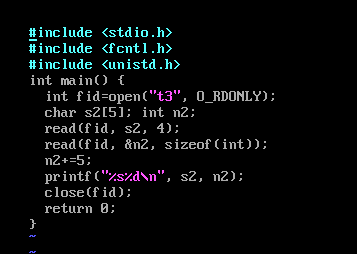
t2的十六进制如上图，可见和t1不一样。

生成二进制文件。用touch命令建立新的空文件t3,然后编写下面的程序，它编译运行后，在16进制下查看，t3与t2的内容一样吗?请你将t3的16进制形式截图，并在图上标出10，-10这2个数字的补码范围。

t3和t2不一样



按数据类型读取文件。将t3中的“a2="的内容读取到变量s2,-10读取到变量n2中。将n2加5后，用下列语句打印输出。printf("%s%din",s2,n2);

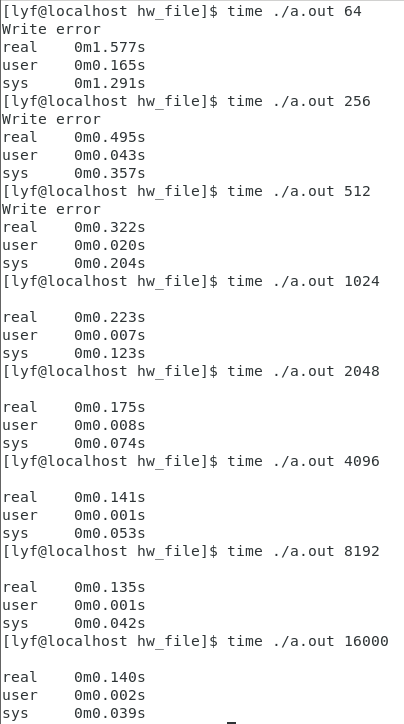


大文件读写read 和 wite函数的参数count指明了读写缓冲区(buf)的大小,若文件较大，读写缓冲区可能远小于文件的大小，这样就需要多次读写文件。那么分配多大的缓冲区，既占内存不大，同时读写效率较高?

先编写如下类似的程序，写入数据到文件t4。该程序运行时必须在命令行给定一个整型参数来指定缓冲区的大小。运行时，使用命令time来测量该程序的执行时间。假设可执行文件为a.out,命令行输入:time ./a.out 512

运行结果的第1行显示的是程序的实际执行时间。

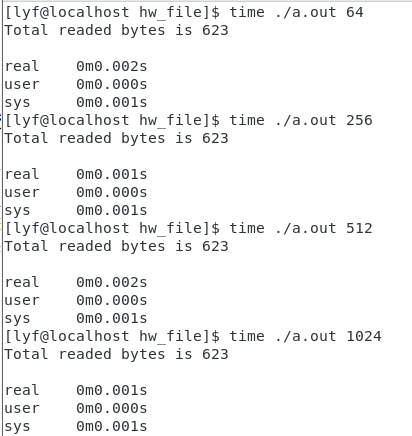
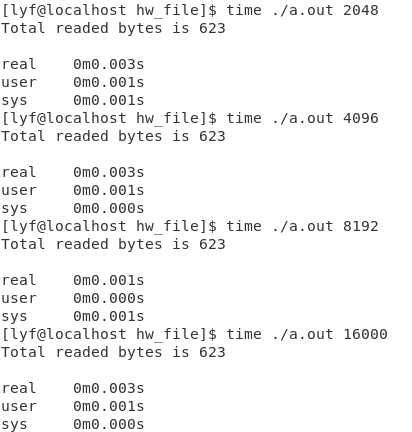
指定不同的缓冲区大小，64，256，512，1024，2048，4096，8192，16000等，多测测量时间。分析一下当前后2个缓冲区各为多大时，节省的时间占执行时间的比例低于50%?



观察结果,随着缓冲区大小的增加,执行时间逐步减少。当缓冲区大小大于等于2048字节时，执行时间节省超过50%。

原因分析:小缓冲区每次写入数据需要更多次系统调用,开销大，而系统调用开销是主要消耗,占整体时间的大头。大缓冲区每次写入的数据量更多,减少系统调用次数,节省系统调用开销。当缓冲区大于等于2KB时,系统调用次数大幅减少,节省的时间超过总时间的50%。所以,当前后2个缓冲区大小2048字节和4096字节时,节省的时间就已经超过总执行时间的50%了。

先编写如下类似的程序，从文件t4分批读取数据。用上面类似的方法，测试缓冲区大概多大时，空间和时间效率比较平衡。

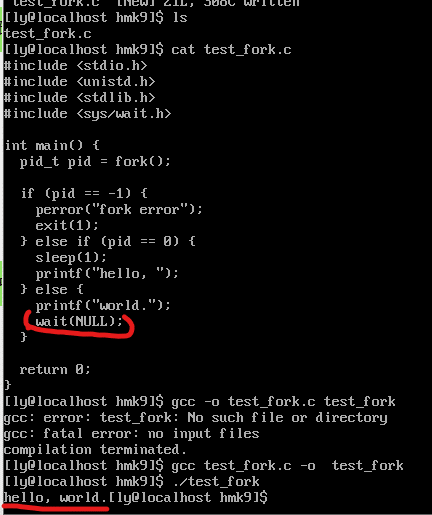
通过对比时间可以看出,2048字节左右时效率最高。

1. Linux进程管理

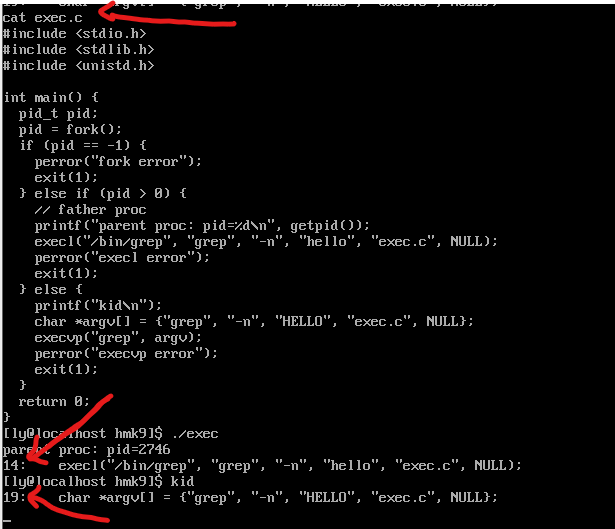
编写一个程序，main函数里开启一个子进程。在main函数里输出“world.”,子进程睡眠一秒后输出“hello,”



可以看见输出的顺序是world hello，原因是在大多数操作系统中，进程的执行是并发的，这意味着两个进程的执行顺序是由操作系统的调度器决定的，而不是由程序的编写顺序决定的。在父进程中使用wait函数后，等待子进程完成后，父进程再输出。观察屏幕输出。可以看到输出了正确的hello world



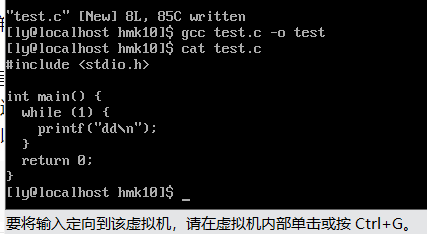
exec函数族。设你编写的源程序名为exec.c,父子线程分别用execl, execvp来分别启动命令grep，在exec.c中分别查找“execl”, "execvp"所在的行。输出结果中间夹杂着命令提示符没关系。



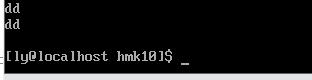
1. Linux信号

SIGINT信号发送

在main函数中，放置一个死循环。编译运行后，按ctrl+c键，测试能否终止程序运行。

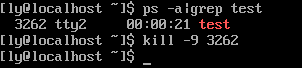
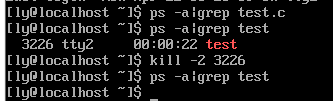


运行并终止：



再次运行以上程序，再打开另一个终端，在该终端中，先用ps -a命令查看该程序的pid,然后通过kill 命令向运行中的程序发送SIGINT信号，效果与1中直接按ctrl+c键一样否？

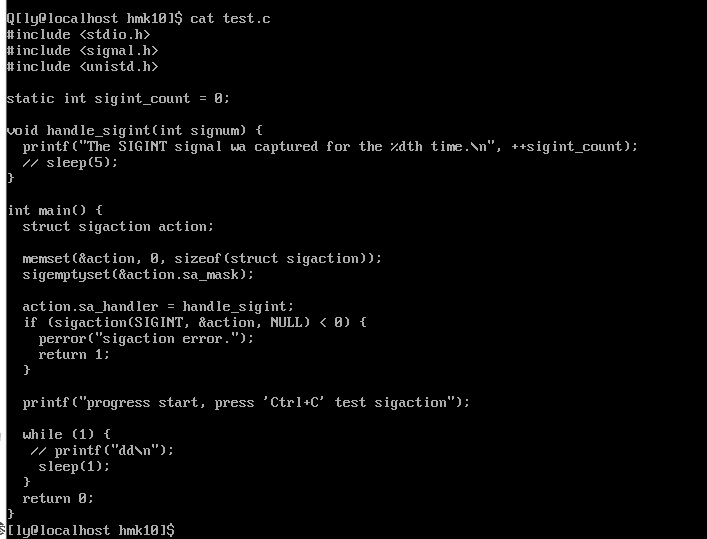
使用ps -a命令可以列出当前终端的所有进程，包括后台运行的进程。找到你的程序的进程ID（PID），然后使用kill命令加上SIGINT信号（或者信号编号2）发送给该PID，可以模拟Ctrl+C的行为。例如，如果PID是1234，那么命令将是kill -2 1234。用kill命令发送9号信号给运行中的程序。



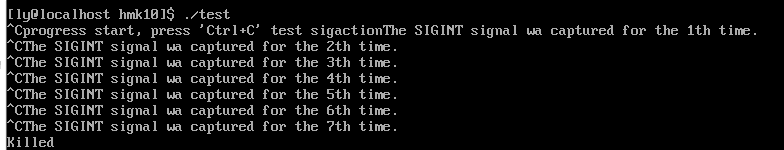
二、捕获SIGINT

1.在程序中用sigaction()函数为SIGINT安装信号处理函数。参考教材P166，但代码的17行是没有必要的。信号处理函数中：

（1）输出你已经第几次捕获了SIGINT信号。应该在信号处理函数里设一个static变量，记录次数。



编写程序测试信号处理次数，运行结果如下：

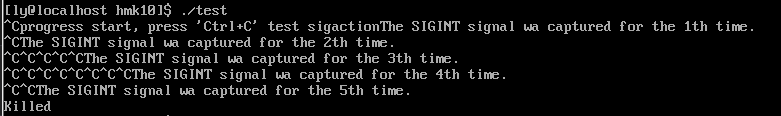


可以看到键入Ctrl+C后立刻打印捕捉次数。

（2）调用sleep(5)，让信号处理函数模拟经历了较长时间的任务处理。

运行程序后，再用ctrl+c键测试，特别包括连续的几次击键测试“未决”信号，体会信号处理期间，SIGINT已经被屏蔽；当信号处理函数运行完，之前的连续ctrl+c键，只被再递送1次。

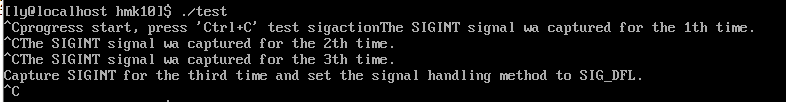
取消sleep(5)的注释，结果如下图：



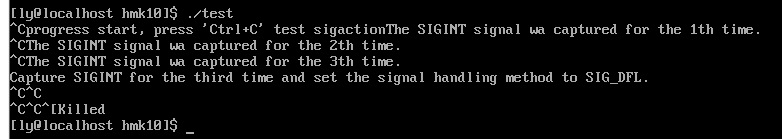
在键入多个Ctrl+C后只被递送一次。

2.在信号处理函数的末尾，判断SIGINT被捕获的次数是否为3，若已到达3次，则用sigaction()函数设置新的信号处理方式分别为SIG\_DFL，SIG\_IGN,参考教材P164倒数第2段关于SIG\_DFL，SIG\_IGN含义的描述。在sigaction结构体的sa\_handler字段应该设置成SIG\_DFL或SIG\_IGN。分别测试它们的效果

设置成SIG\_DFL，三次后会自动退出程序。



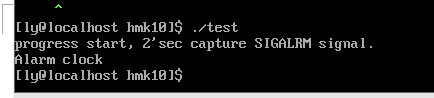
设置成SIG\_IGN：



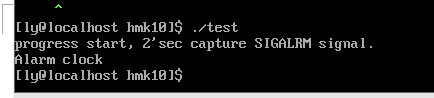
不能自动退出程序，需要在tty3杀死进程。

定时器

1.在main函数中用alarm函数设置一个2秒的定时器，然后运行一个死循环。观察到两秒后程序会终止。。



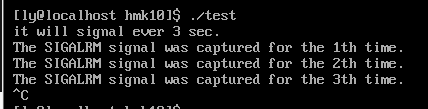
2.sigaction()函数设置捕获SIGALRM信号，在信号处理函数中输出一个字符串。



3.用setitimer函数代替alarm函数，设置一个周期性的1秒定时器。在其信号处理函数中，记录触发的次数，当触发了3次后，再次调用setitimer函数，设置定时时间为0，以便取消该定时器。

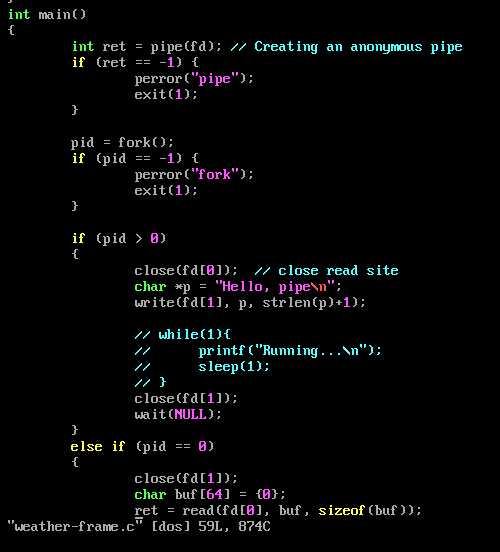
提示：设定结构体itimerval的值时，可以简洁点。

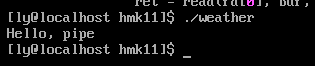
struct itimerval it={{1,0},{1,0}};



1. 匿名通信

父子进程通过匿名管道通信，父子程序实现匿名管道通信：





子进程先延时2秒，产生天气信息“Sunny”,再延迟2秒，产生天气信息“Cloudy”.当子进程把天气信息写入管道后，则给父进程发信号SIGUSR1

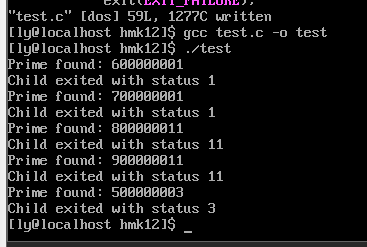
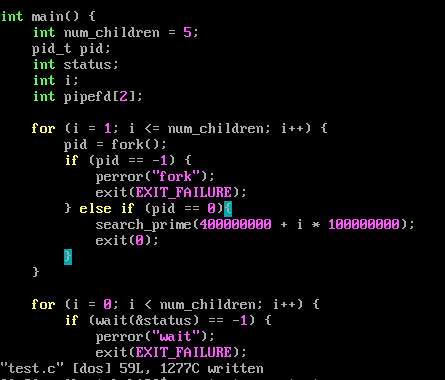
父进程每隔1秒，显示一条“Running...”信息，在这个过程中，若收到天气通知信号，则读取天气信息并显示。父进程收到来自用户的SIGINT信号，则终止父进程。



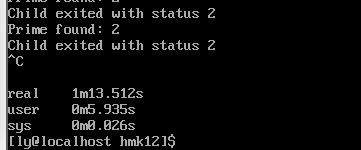
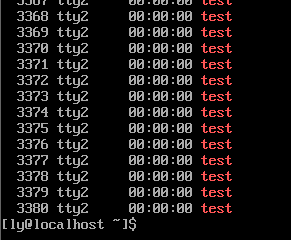
捕获信号，输出，结束进程

1. 进程、线程对比

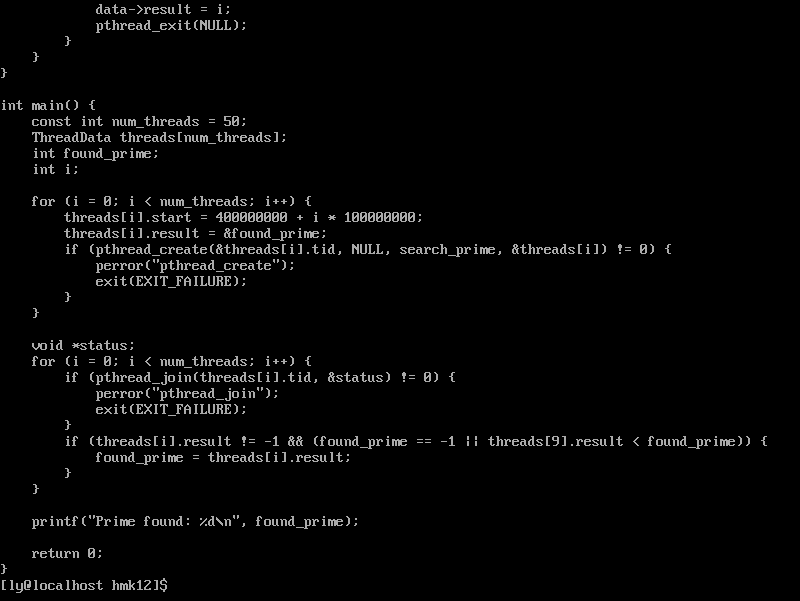
编写程序来搜索大的素数，使用多进程：



查看进程空间大小使用命令：ps -a|grep test, 测试运行时。结果如下：



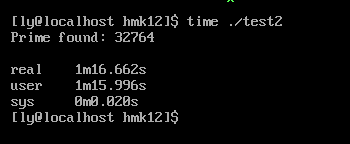
使用线程完成任务：



查看任务空间



测试任务时间子线程为50个：



进程、线程对比：

|  |  |  |
| --- | --- | --- |
| **特性** | **进程** | **线程** |
| 地址空间 | 每个进程拥有独立的地址空间 | 同一进程内线程共享地址空间 |
| 内存分配 | 需要独立的内存空间 | 共享进程的内存空间 |
| 创建开销 | 较大，涉及内存空间分配 | 较小，共享内存空间 |
| 销毁开销 | 较大，需要操作系统进行大量工作 | 较小，配置线程特定资源 |
| 运行时间 | 创建和销毁进程开销较大 | 创建和销毁线程开销较小 |

1. 信号量实现生产者-消费者模型

问题：某工厂有两个生产车间和一个装配车间，两个生产车间分别生产A、B两种零件，装配车间负责组装零件A、B。两个生产车间每生产一个零件后都要分别将这两个零件送到装配车间的货架F1(容量4)、F 2(容量6)上，装配车间每次在组装零件时都要从货架F1、F 2上分别取下零件A和零件B。分析题目，写出实现题目描述问题需要定义的信号量并说明每个信号量的功能和初值。

答：在这个场景中，可以定义以下信号量来实现生产者-消费者模型：

1. sem\\_F1：初值为4，用于表示货架F1上的零件数量，生产车间每生产一个零件A都会将其放入F1上，装配车间每次组装（消费）一个零件A时会从F1上取走一个零件A。

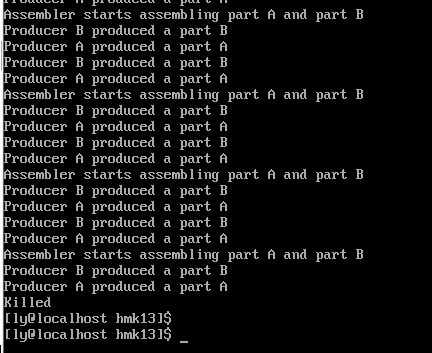
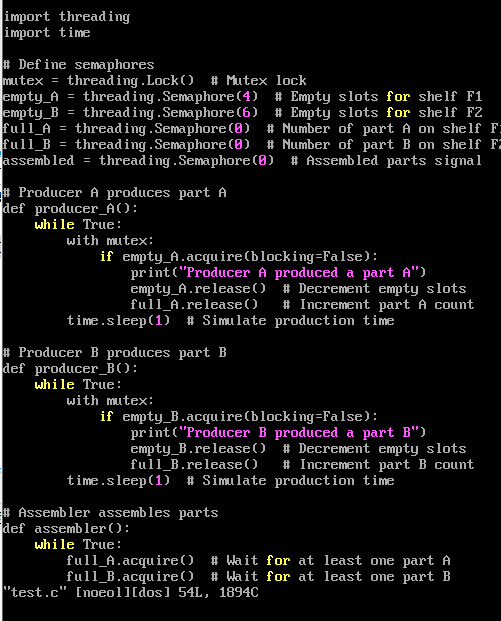
2. sem\\_F2：初值为6，用于表示货架F2上的零件数量，生产车间每生产一个零件B都会将其放入F2上，装配车间每次组装（消费）一个零件B时会从F2上取走一个零件B。

3. mutex：初值为1，用于实现对货架F1和F2的互斥访问，确保一次只有一个车间可以访问货架。

4. empty ：初值为10，用于表示可以生产的零件数量。每个生产车间每生产一个零件时都会将empty减一，装配车间每次组装一个零件时会将empty加一，确保生产和消费的平衡。

通过正确地操作这些信号量，就可以实现生产者-消费者模型，保证每个车间的正常运作。

代码和运行结果如下：



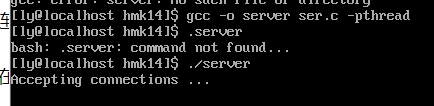
1. 信号量实现生产者-消费者模型

并行下载

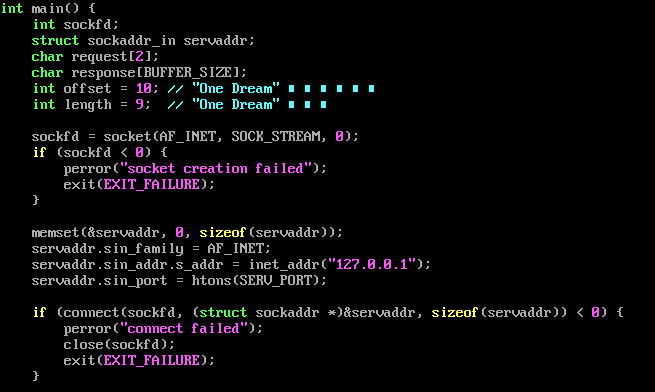
编写一个2个客户端程序，从服务器下载一个文件。一个是单进程(或线程)，另一个是4个子进程(或线程)，分别下载一个文件的不同分段。提交代码，记录它们下载速度的差异。

服务器模拟网络延迟，每隔1秒才能发送一个字节。它的代码见附件。

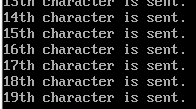
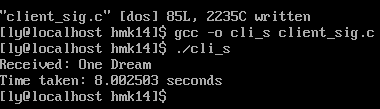
运行服务器并编译ser.c文件



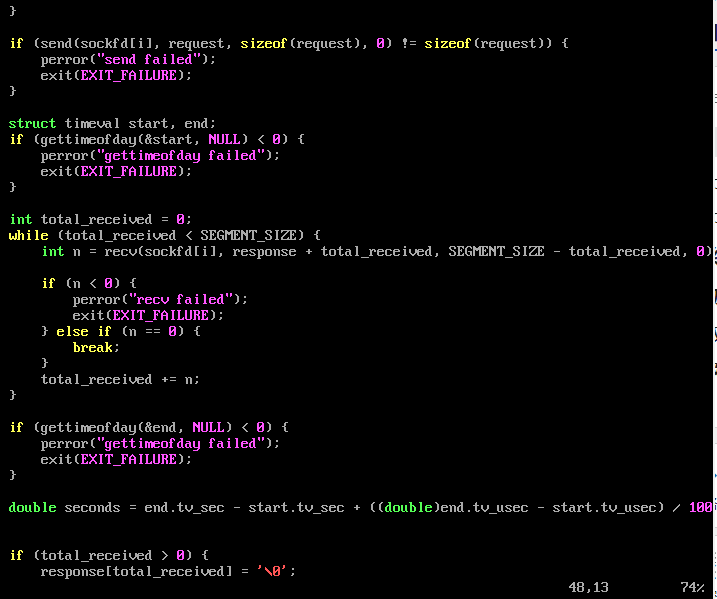
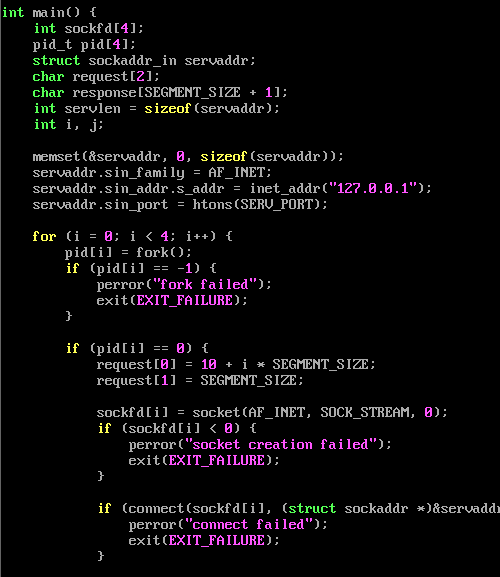
在tty3上运行客户端代码



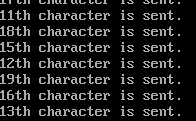
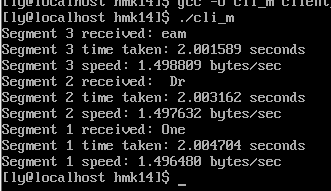
客户端响应 和 服务器端响应：



多线程下载’One Dream’



客户端响应，服务器响应：



数据对比

|  |  |  |
| --- | --- | --- |
|  | Client\_sig | Client\_mul |
| 下载时间 | 8.002503 | 6.009455 |