**目录**

[实验一题目：Linux操作系统基本命令 2](#_Toc180589117)

[实验二题目：用户界面与shell命令 9](#_Toc180589118)

[实验三题目：进程管理及进程通信 18](#_Toc180589119)

[实验四题目：LINUX进程调度与系统监视 41](#_Toc180589120)

[实验五题目：用户与群组管理 52](#_Toc180589121)

[实验六题目：shell编程 74](#_Toc180589122)

# 实验一题目：Linux操作系统基本命令

**姓名：汪江豪 学号：22121630 实验日期：2024/8/29**

**实验环境：**

Linux操作系统，ubuntu版本

**实验目的：**

1. 了解Linux运行环境，熟悉交互式分时系统、多用户环境的的运行机制。 2. 练习Linux系统命令接口的使用，学会Linux基本命令、后台命令、管道命令等命令 的操作要点。

**实验内容：**

通过终端或虚拟终端，在基于字符的交互界面中进行Shell的基本命令的操作。

**操作过程1：**

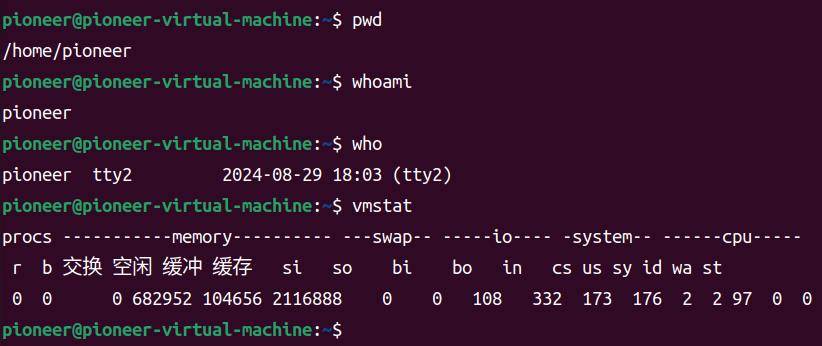
① 执行 pwd 查看当前目录

② 用 who am i 看看当前用户信息

③ 通过 who 看看有谁在系统中

④ 用 vmstat 显示系统状态

**结果1：**



**思考：**你的用户名、用户标识、组名、组标识是什么？当前你处在系统的哪个位置中？现在 有哪些用户和你一块儿共享系统？

**答：**键入whoami显示用户名pioneer，键入id -u显示用户标识1000，键入id pioneer可查看组名组标识，可知道组标识1000，组名pioneer等，键入pwd可知当前处于/home/pioneer,键入who可知当前只有自己一人，无其他人共享系统。

**操作过程2：**

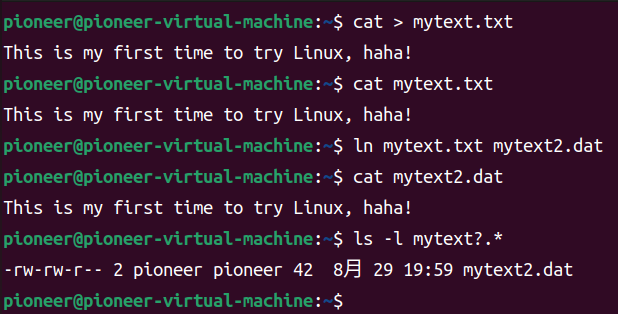
① 执行 cat > mytext.txt 通过键盘输入“hello world!”

② 执行 cat mytext.txt

③ 执行 ln mytext.txt mytext2.dat cat mytext2.dat

④ 执行 ls –l mytext?.\*

**结果2：**



**思考：**文件链接是什么意思？有什么作用？

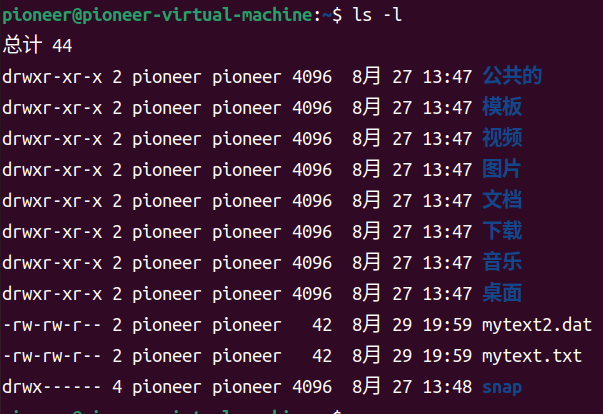
**答：**文件链接是一种机制，它允许一个文件名指向另一个文件或目录的路径。作用是可使得多个文件共享相同的存储空间。

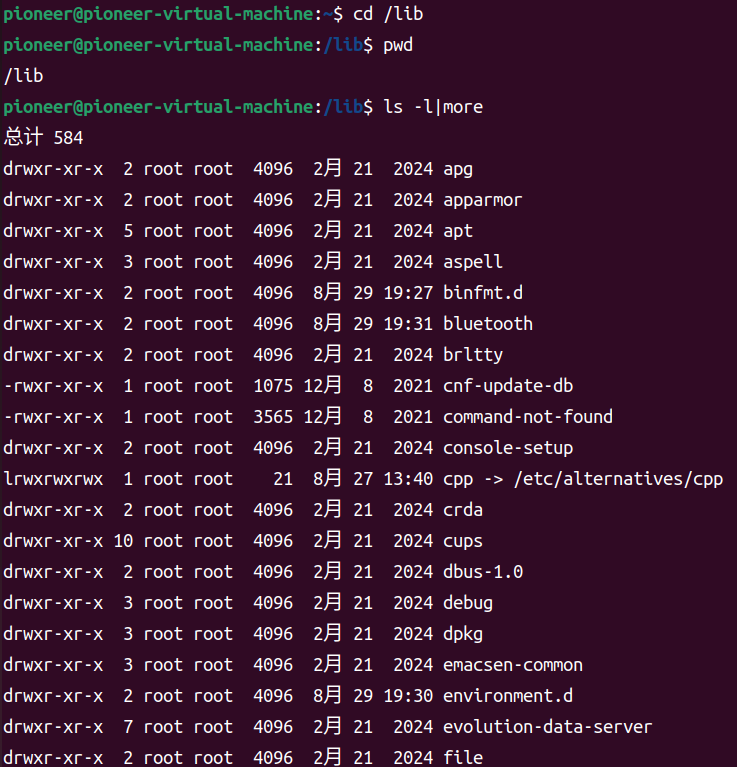
**操作过程3：**

① 执行 ls –l

② 执行 cd /lib ls –l|more

**结果3：**





**思考：**Linux 文件类型有哪几种？文件的存取控制模式如何描述？

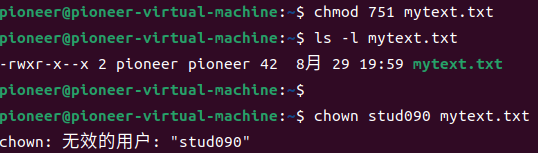
**答：**Linux文件类型包括普通文件，目录文件，链接文件，块设备文件，字符设备文件，套接字文件，管道文件等。文件存取的控制模式通常由三个部分组成，包括：用户权限，组权限，其他用户权限。

**操作过程4：**

1. 执行 chmod 751 mytext.txt ls –l mytext.txt

2. 执行 sudo chown root mytext.txt

**结果4：**



**思考：**执行了上述操作后，若想再修该文件，看能不能执行。为什么？

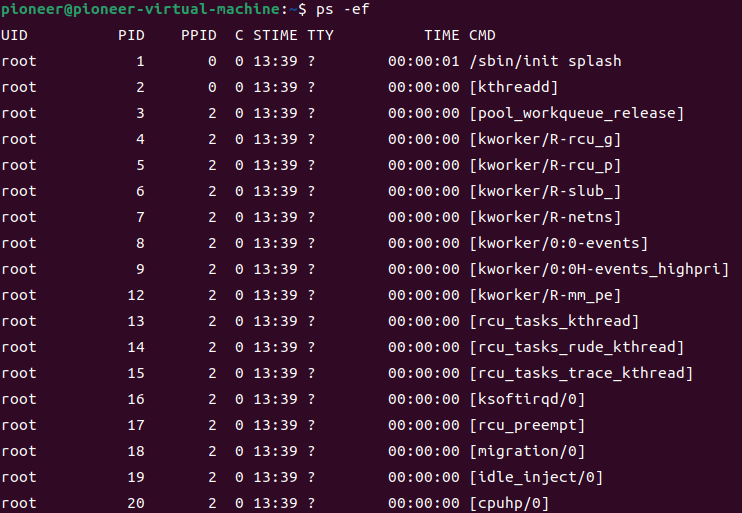
**答：**如果主机有其他用户stud090，该操作会将文件所有者改为stud090，并且文件权限改为751后，所有者有读、写、执行权限，所属组有读、执行权限，其他用户只有执行权限。现在pioneer只能读或执行该文件，不能修改文件。

**操作过程5：**

1. 执行 ps -ef

2. 执行 sleep 5 & wait $!

**结果5：**



**思考：**系统如何管理系统中的多个进程？进程的家族关系是怎样体现的？有什么用？

**答：**ps查看当前运行的进程，top实时监控进程和系统资源使用情况，kill与进程通信.

进程家族关系包括进程树，父子关系，会话和进程组。每个进程有唯一的PID和一个可能的父进程PPID，所有进程都是以init进程（PID为1）为根的树状结构。当一个进程创建另一个进程时，创建者为父进程，新进程为子进程，子进程继承了父进程某些特性，如环境变量，权限等。进程可以属于一个进程组，进程组中的进程可以同时被信号影响；进程也可以属于一个会话，会话通常由控制终端控制。

这种家族关系有利于资源管理，父进程通过信号杀死子进程；作业控制，信号传递，系统清理，在系统关机时，init进程会负责清理所有子进程；故障排查，通过查看进程家族树，管理员可以更容易理解系统中进程的相互关系，帮助排查问题。

**讨论：**

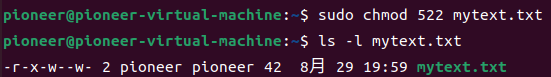
**讨论：**

1. **Linux 系统命令很多，在手头资料不全时，如何查看命令格式？**

**答：**输入command –help，或者man command等（command指具体的某个命令）查看帮助。

1. **Linux 系统用什么方式管理多个用户操作？如何管理用户文件，隔离用户空间？用命令 及结果举例说明。**

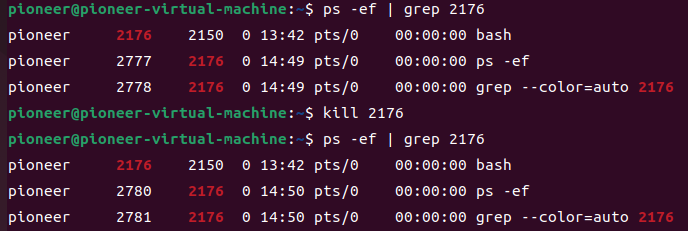
**答：**Linux中，每个用户都有一个唯一的用户账户，，包括用户名和用户ID（UID），用户可以被分配到不同的用户组，每个用户组有一个ID（GID）。文件权限（读、写、执行）和所有权（用户和组）决定了用户对文件的访问。Sudo chown NEWUSER MYFILE.txt可以修改文件所有者，sudo chmod 751 MYFILE.txt可设置文件权限，ls -l MYFILE.txt可查看文件权限



如图，所有者有读，执行权限，所属组有写权限，其他用户有写权限。

1. **用什么方式查看你的进程的管理参数？这些参数怎样体现父子关系？当结束一个父进程 后其子进程如何处理？用命令及结果举例说明。**

**答：**ps -ef查看所有进程详细信息，pstree显示以当前进程为根的进程树，top显示进程监控界面。杀死父进程，若子进程是后台进程， 将继续运行；若子进程是前台进程，将一起结束，若子进程是暂停状态，将继续暂停，直到被唤醒或结束。



2176进程应该与搜索进程相关，可见，杀死进程2176后又重新创建了。

1. **Linux 系统“文件”的含义是什么？它的文件有几种类型？如何标识的？**

**答：**Linux中的文件指任何数据存储单元，可以是文本文件、二进制文件、目录、链接、设备、套接字。文件类型可以分为以下几种：普通文件、目录文件、链接文件、块设备文件、字符设备文件、套接字文件、命名管道。文件类型的标识通常通过文件权限和文件属性区分，普通文件-开头，目录文件d开头，链接文件l开头，设备文件c或b开头，套接字文件s开头，命名管道p开头。

1. **Linux系统的可执行命令主要放在什么地方？找出你的计算机中所有存放系统的可执行命 令的目录位置。**

**答：**可执行命令通常在几个特定的目录中，被称为标准目录，

/bin:二进制命令目录，包含系统启动时需要的基本命令，ls,cp,rm,对所有用户可执行

/sbin:包含系统管理命令，ifconfig,mount,umount主要用于系统管理员

/usr/bin:包含大部分用户级命令,grep,sed,awk等

/usr/sbin:用户级系统管理命令,apachectl,ssh,sftp等

/usr/local/bin:存放本地安装命令，由本地管理员安装的第三方软件的命令

/usr/local/sbin:本地安装的系统管理命令

1. **Linux 系统的设备是如何管理的？在什么地方可以找到描述设备的信息？**

**答：**Linux使用虚拟文件系统机制来管理设备，它允许设备驱动程序以文件系统的方式与用户空间程序交互。设备文件：每个设备都有一个对应的设备文件，位于/dev目录下，驱动程序通常位于/lib/modules/目录下，sysfs提供目录树，用于存储内核管理的设备信息，包括设备的属性、状态等，位于/sys下，每个设备都有一个对应的目录，其中包含了该设备的详细信息。

1. **画出 Linux 根文件系统的框架结构。描述各目录的主要作用。你的用户主目录在哪里？**

**答：**/bin：这个目录包含系统启动时必需的基本命令，通常对所有用户都是可执行的。例如，ls, cp, rm 等。

/sbin：这个目录包含系统管理命令，如 ifconfig, mount, umount 等。这些命令主要用于系统管理员。

/usr：这个目录包含大部分用户级命令，如 grep, sed, awk 等。

/usr/bin：这个目录包含用户级命令，如文本编辑器、文件管理工具等。

/usr/sbin：这个目录包含用户级系统管理命令，如网络管理工具、服务控制命令等。

/usr/local/bin：这个目录用于存放本地安装的命令，通常是由本地管理员安装的第三方软件的命令。

/usr/local/sbin：这个目录包含本地安装的系统管理命令。

/etc：这个目录包含系统配置文件和启动脚本。它是系统启动和运行时所需的重要目录。

/var：这个目录包含可变数据文件，如日志文件、缓存文件、邮件等。

/home：这个目录包含用户的家目录，每个用户都有自己的目录，用于存储用户的数据和文件。

/dev：这个目录包含设备文件，它们是内核和设备驱动程序之间的接口。

/proc：这个虚拟文件系统包含内核和进程的信息。

/sys：这个虚拟文件系统包含内核管理的设备信息。

/tmp：这个目录包含临时文件，通常用于存储临时数据。

/run：这个目录包含运行时数据，如系统启动后生成的文件。

/mnt：这个目录用于挂载临时文件系统。

/boot：这个目录包含内核和启动加载器的配置文件。

/media：这个目录用于挂载可移动媒体，如CD-ROM。

/opt：这个目录用于安装可选的软件包。

/srv：这个目录用于存储服务相关的数据。

/selinux：这个目录包含SELinux（Security-Enhanced Linux）相关的文件。

/lost+found：这个目录包含文件系统恢复数据。

用户主目录在/home，如用户名为pioneer，则为/home/pioneer

1. **Linux 系统的 Shell 是什么？请查找这方面的资料，说明不同版本的 Shell 的特点。**

**答：**Linux系统中的Shell是一种命令行接口，它允许用户通过文本命令与操作系统进行交互。用户输入的命令首先被shell解释，然后传递给Linux内核执行，shell提供了用户与内核之间的桥梁。Shell本身是一个应用程序，不是内核的一部分，而是基于内核编写，在系统启动时立即启动并呈现给用户。

Bourne shell(sh)UNIX最初使用的shell。

C shell(csh)语法上接近C语言的shell。

Korn shell(ksh)兼容sh并包含了C shell很多特性。

Bash(Bourne Again shell)是当前最常用的一种shell，它由Bourne shell的基础上扩展和增强的，向后兼容，并增加了一些新特性。

Tcsh，csh增强版，与csh完全兼容。

Ash被用作系统的登录shell或嵌入在其他应用程序中。

1. **下面每一项说明的是哪类文件。**

(1)-rwxrw-r-- (2) /bin (3) ttyx3 (4) brw-rw-rw- (5)/etc/passwd (6) crw-rw-rw (7) /usr/lib (8) Linux

**答：**-rwxrw-r—表示普通文件，注明了所有者、所属组、其他用户的读写执行权限。

/bin系统启动时的基本命令。

Ttyx3第三个虚拟终端。

brw-rw-rw-块设备文件的权限。

/etc/passwd存储用户账户信息，包括用户名，UID,GID,用户主目录，用户shell，用户密码等。

crw-rw-rw字符设备文件权限。

/usr/lib存储系统库和可执行文件的二进制版本，用于支持系统的核心功能以及诸多应用程序的运行。

Linux内核源代码

**讨论：**

我们大多数普通人首先接触的计算机系统都是具有图形化界面的GUI，然而linux最早是一个著名的命令行界面CLI，刚开始接触可能觉得比较难用，但如果数量掌握后，命令行的形式会比图形化界面的操作效率高很多，能够节省很多性能。

通过本次实验，我熟悉了linux的一些基本的终端命令，这些有助于我从底层更好地理解操作系统内核和内部运作方式。

Linux是一个强大的开源操作系统，广泛用于服务器、超级计算机、移动设备以及嵌入式系统等。学习linux操作系统也不是一蹴而就的事情，是一个日积月累的过程，今后在学习操作系统的过程中，我需要不断熟悉命令行的操作界面，这是作为计算机专业学生的必修课。

# 实验二题目：用户界面与shell命令

**姓名：汪江豪 学号：22121630 实验日期：2024/9/12**

**实验环境：**

Linux操作系统ubuntu版本

**实验目的：**

1. 掌握图形化用户界面和字符界面下使用 Shell 命令的方法。
2. 掌握 ls、cd 等 Shell 命令的功能。
3. 掌握重定向、管道、通配符、历史记录等的使用方法。
4. 掌握手工启动图形化用户界面的设置方法。

**实验内容：**

1. 图形化用户界面下的 Shell 命令操作
2. 2. 字符界面下的 Shell 命令操作
3. 3. 通配符的使用
4. 4. 设置手工启动图形化用户界面
5. **图形化用户界面下的shell命令操作**

**操作过程1：**

【操作要求 1】显示系统时间，并将系统时间修改为 2011 年 9 月 17 日零点

【操作步骤】

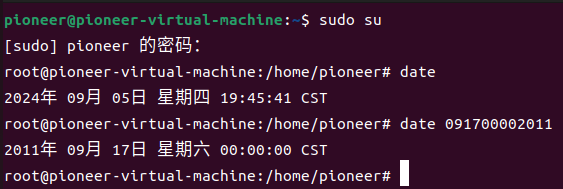
① 启动计算机，以超级用户身份登录图形化用户界面。

② 依次单击顶部面板的「应用程序」菜单=>「附件」=>「终端」，打开桌面环境下的 终端工具。

③ 输入命令“date”，显示系统的当前日期和时间。

④ 输入命令“date091700002011”，屏幕显示新修改的系统时间。在桌面环境的终端中执 行时显示中文提示信息。

**结果1：**



**操作过程2：**

【操作要求 2】切换为普通用户，查看 2011 年 9 月 17 日是星期几

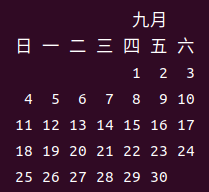
【操作步骤】

① 前一操作是以超级用户身份进行的，但通常情况下只有在必须使用超级用户权限的 时候，才以超级用户身份操作。为提高操作安全性，输入“su-helen”命令切换为普 通用户 helen。

② 输入命令“cal2011”，屏幕上显示出 2011 年的日历，由此可知 2011 年 9 月 17 日是 星期日。

**结果2：**





**操作过程3：**

【操作要求 3】查看 ls 命令的-s 选项的帮助信息

【操作步骤】

方法一：

① 输入“man ls”命令，屏幕显示出手册页中 ls 命令相关帮助信息的第一页，介绍 ls 命 令的含义、语法结构以及-a、-A、-b 和-B 等选项的意义。

② 使用 PgDn 键、PgUp 键以及上、下方向键找到-s 选项的说明信息。

③ 由此可知，ls 命令的-s 选项等同于--size 选项，以文件块为单位显示文件和目录的大 小。

④ 在屏幕上的“：”后输入“q”，退出 ls 命令的手册页帮助信息。

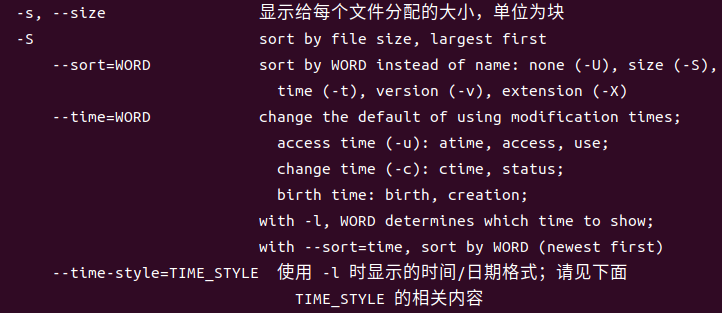
方法二：

① 输入命令“ls --help”，屏幕显示中文的帮助信息。

② 拖动滚动条，找到-s 选项的说明信息，由此可知 ls 命令的-s 选项等同于--size 选项， 以文件块为单位列出所有文件的大小。

③ 在屏幕上的“：”后输入“q”，退出 ls 命令的手册页帮助信息

**结果3：**



**操作过程4：**

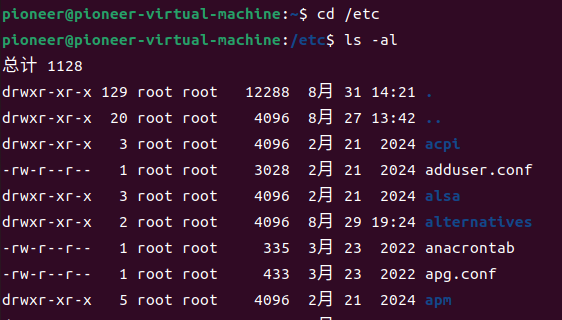
【操作要求 4】查看/etc 目录下所有文件和子目录的详细信息

【操作步骤】

① 输入命令“cd /etc”，切换到/etc 目录。

② 输入命令“ls -al”，显示/etc 目录下所有文件和子目录的详细信息。

**结果4：**



1. **字符界面下的shell命令操作**

**操作过程1：**

【操作要求 1】查看当前目录

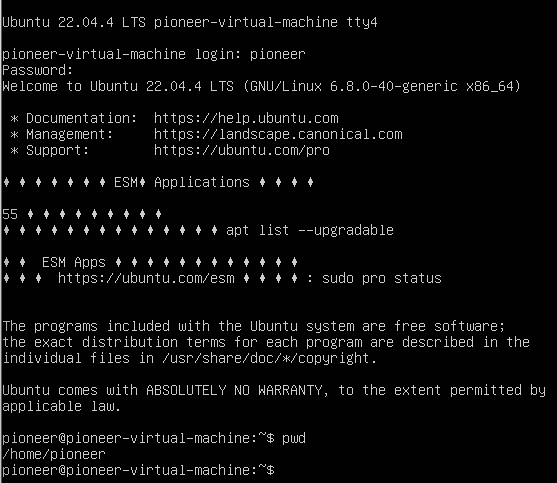
【操作步骤】

① 启动计算机后默认会启动图形化用户界面，按下 CTRL+ALT+F1 键切换到第 1 个虚 拟终端。

② 输入一个普通用户的用户名（helen）和口令，登录系统。

③ 输入命令“pwd”，显示当前目录，相关操作参见如下内容。

**结果1：**



**操作过程2：**

【操作要求 2】用 cat 命令在用户主目录下创建一名为 f1 的文本文件，内容为： Linux is useful for us all. You can never imagine how great it is.

【操作步骤】

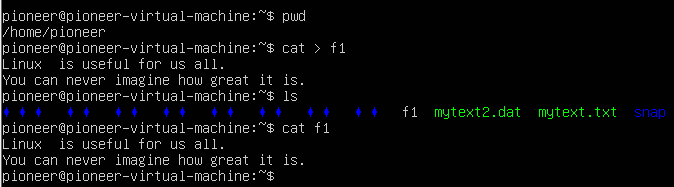
① 输入命令“cat>f1”，屏幕上输入点光标闪烁，依次输入上述内容。

② 上述内容输入后，按 Enter 键，让光标处于输入内容的下一行，按 CTRL+D 键结束 输入。

③ 要查看文件是否生成，输入命令“ls”即可。

④ 输入命令“cat f1”，查看 f1 文件的内容，相关操作参见如下内容。

**结果2：**



**操作过程3：**

【操作要求 3】向 f1 文件增加以下内容：Why not have a try?

【操作步骤】

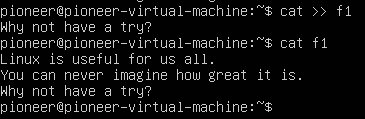
方法一：

① 输入命令“cat>>f1”，屏幕上输入点光标闪烁。

② 输入上述内容后，按 Enter 键，让光标处于输入内容的下一行，按 CTRL+D 键结束 输入。

③ 输入“cat f1”命令，查看 f1 文件的内容，会发现 f1 文件增加了一行，相关操作参见 如下内容。

**结果3：**



**操作过程4：**

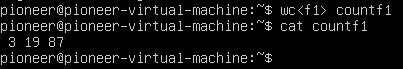
【操作要求 4】统计 f1 文件的行数，单词数和字符数，并将统计结果存放在 countf1 文件

【操作步骤】

① 输入命令“wc countf1”，屏幕上不显示任何信息。

② 输入命令“cat countf1”，查看 countf1 文件的内容，其内容是 f1 文件的行数、单词数 和字符数信息，即 f1 文件共有 3 行，19 个词和 87 个字符。

**结果4：**



**操作过程5：**

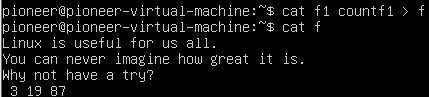
【操作要求 5】将 f1 和 countf1 文件的合并为 f 文件

【操作步骤】

① 输入命令“cat f1 countf1 >f”，将两个文件合并为一个文件。

② 输入命令“cat f”，查看 f 文件的内容，如下所示。

**结果5：**



**操作过程6：**

【操作要求 6】分页显示/etc 目录中所有文件和子目录的信息

【操作步骤】

① 输入命令“ls /etc|more”，屏幕显示出“ls/etc”命令输出结果的第一页，屏幕的最后一 行上还出现“--More--”字样，按空格键可查看下一页信息，按 Enter 键可查看下一行 信息。

② 浏览过程中按“q”键，可结束分页显示。

**结果6：**



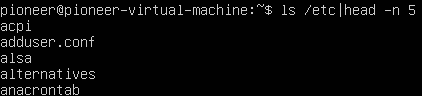
**操作过程7：**

【操作要求 7】仅显示/etc 目录中前 5 个文件和子目录

【操作步骤】

输入命令“ls /etc|head -n 5”,屏幕显示出“ls /etc”命令输出结果的前面 5 行

**结果7：**



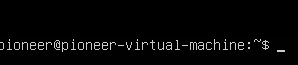
**操作过程8：**

【操作要求 8】清除屏幕内容

【操作步骤】

输入命令“clear”，则屏幕内容完全被清除，命令提示符定位在屏幕左上角。

**结果8：**



1. **通配符的使用**

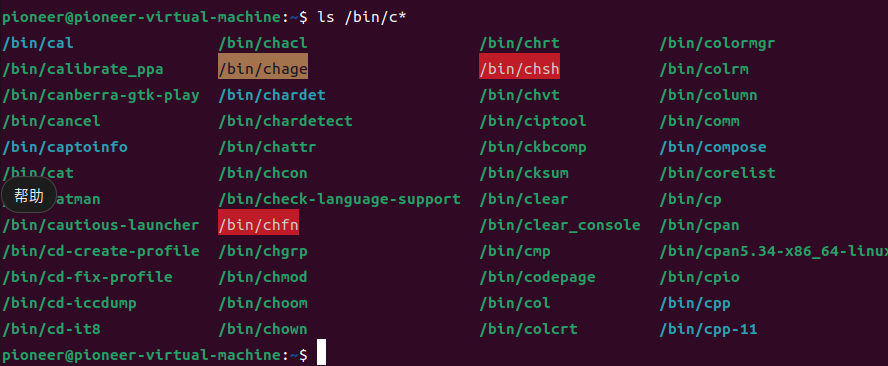
**操作过程1：**

【操作要求 1】显示/bin/目录中所有以 c 为首字母的文件和目录

【操作步骤】

输入命令“ls /bin/c\*”，屏幕将显示/bin 目录中以 c 开头的所有文件和目录。

**结果1：**



**操作过程2：**

【操作要求 2】显示/bin/目录中所有以 c 为首字母,文件名只有 3 个字符的文件和目录 【操作步骤】

① 按向上方向键，Shell 命令提示符后出现上一步操作时输入的命令“ls /bin/c\*”。

② 将其修改为“ls /bin/c??”，按下 Enter 键，屏幕显示/bin 目录中以 c 为首字母。

**结果2：**



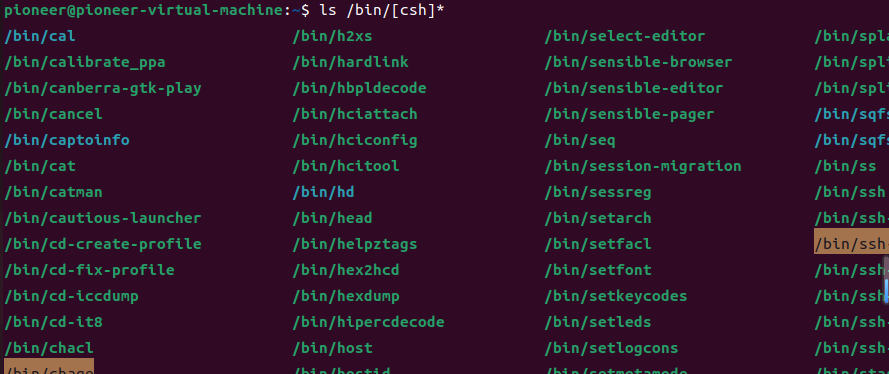
**操作过程3：**

【操作要求 3】显示/bin 目录中所有的首字母为 c 或 s 或 h 的文件和目录

【操作步骤】

输入命令“ls /bin/[csh]\*”，屏幕显示/bin 目录中首字母为 c 或 s 或 h 的文件和目录

**结果3：**



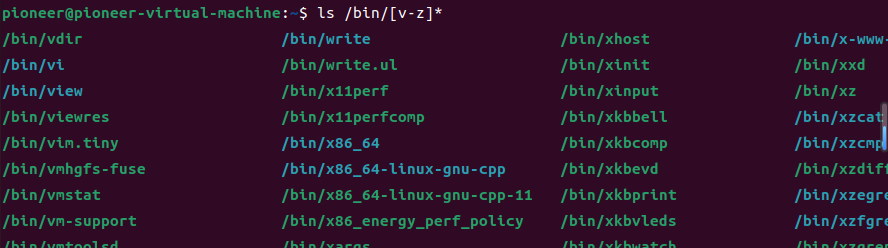
**操作过程4：**

【操作要求 4】显示/bin/目录中所有的首字母是 v、w、x、y、z 的文件和目录

【操作步骤】

输入命令“ls /bin/[!a-u]\*”，屏幕显示/bin 目录中首字母是 v~z 的文件和目录。

**结果4：**



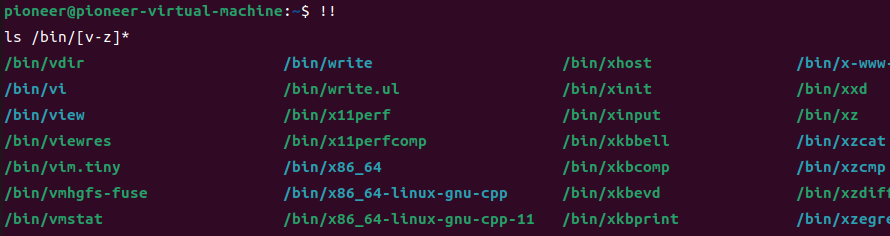
**操作过程5：**

【操作要求 5】重复上一步操作

【操作步骤】

输入命令“!!”，自动执行上一步操作中使用过的“ls /bin/[!a-u]\*”命令。

**结果5：**

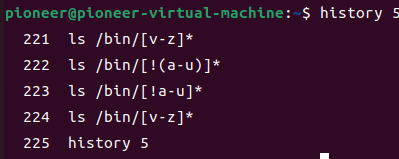


**操作过程6：**

【操作要求 6】查看刚执行过的 5 个命令

【操作步骤】 输入命令“history 5”，显示最近执行过的 5 个命令。命令编号可能不同。

**结果6：**



1. **设置手工启动图形化用户界面**

**操作过程1：**

【操作要求 1】设置开机不启动图形化用户界面

【操作步骤】

① 按下 ALT+F7 键，切换回到图形化用户界面，以超级用户身份登录。

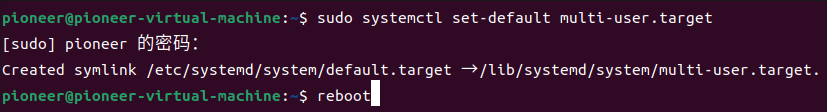
② 依次单击「应用程序」菜单=>「附件」=>「文本编辑器」，打开 gedit 文本编辑器。

③ 单击工具栏上的「打开」按钮，从「打开文件...」对话框中选择/etc 目录中的 inittab 文件。

④ 将文件中的“id：5：initdefault：”所在行的“5”修改为“3”，修改后的文件如图 2-4 所 示。

⑤ 单击工具栏上的「保存」按钮，并关闭 gedit。 ⑥ 单击顶部面板的「系统」菜单=>「关机」，弹出对话框，选择「重新启动」，重新 启动计算机。

**结果1：**



**操作过程2：**

【操作要求 2】手工启动图形化用户界面

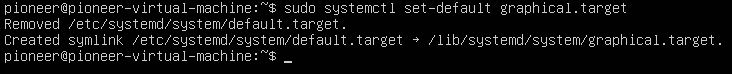
【操作步骤】

① 计算机重启后只有字符界面可用，输入用户名和相应的口令后，登录 Linux 系统。

② 输入命令“startx”，启动图形化用户界面。

③ 单击「系统」菜单=>「注销」，弹出对话框，单击「注销」按钮，返回到字符界面。

**结果2：**



**体会：**

通过这次实验，我深刻体会到图形用户界面(GUI)和命令行界面(CLI)在操作效率上的差异，虽然GUI提供了直观、易用的操作方式，但在执行批量任务和自动化脚本方面，CLI的效率远超GUI。同时，我发现了命令行的强大功能，通过简单的命令组合，就能完成复杂的数据处理和系统管理任务。刚开始接触shell命令时，可能感到有些不适应，但随着实验的深入，，发现掌握基本命令后，解决实际问题变得游刃有余。

同时，我也认识到了用户权限的重要性，和操作系统多用户的必要性，这能提高操作安全性的同时，提高工作效率。

# 实验三题目：进程管理及进程通信

**姓名：汪江豪 学号：22121630 实验日期：2024/9/19**

**实验环境：**

Linux操作系统

**实验目的：**

1. 利用 Linux 提供的系统调用设计程序，加深对进程概念的理解

2. 体会系统进程调度的方法和效果

3. 了解进程之间的通信方式以及各种通信方式的使用

**实验内容：**

用 vi 编写使用系统调用的 C 语言程序

**操作过程1：**

编写程序。显示进程的有关标识（进程标识、组标识、用户标识等）。经过 5 秒钟后， 执行另一个程序，最后按用户指示（如：Y/N）结束操作。

#include <stdio.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

int main()

{

int pid;

printf("Process ID: %d\n", getpid());

printf("Process Group ID: %d\n", getpgrp());

printf("User ID: %d\n", getuid());

printf("Group ID: %d\n", getgid());

pid = fork(); // 创建一个新的子进程

if (pid == -1) {

// fork 失败的情况

perror("Failed to fork");

return 1;

} else if (pid == 0) {

// 子进程执行的代码

if (execlp("./hello", "hello", NULL) == -1) {

perror("Failed to execute hello");

}

\_exit(0); // 确保子进程在这里结束

} else {

// 父进程执行的代码

int status;

waitpid(pid, &status, 0);

// pid返回了子进程ID,status指针，指向整数变量，存储子进程结束时的状态信息，options=0，表示waitpid将阻塞父进程，直到子进程结束。

if (WIFEXITED(status)) {

printf("Child exited with status %d\n", WEXITSTATUS(status));

}

printf("Press Y/N to continue: ");

char choice;

scanf(" %c", &choice); // 注意添加空格来跳过前面可能的换行符

if (choice == 'Y' || choice == 'y') {

printf("Operation completed.\n");

} else {

printf("Operation aborted.\n");

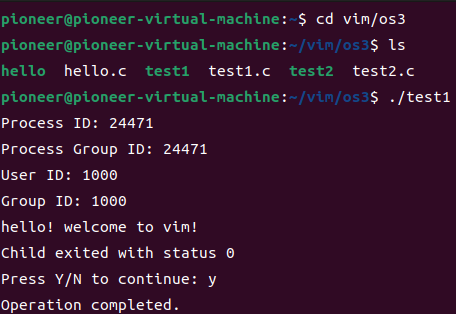
}

}

return 0;

}

**结果1：**



程序在显示完进程标识，进程组标识，用户标识，组标识后，进入子进程执行了程序hello.exe文件，显示文本后，返回父进程，并按用户指示结束了操作。

**操作过程2：**

参考例程 1，编写程序。实现父进程创建一个子进程。体会子进程与父进程分别获得不 同返回值，进而执行不同的程序段的方法。

#include<stdio.h>

#include<sys/types.h>

#include<wait.h>

#include<unistd.h>

int main()

{

int i;

if(fork())

{

printf("Parent process has been set.\n");

i=wait(NULL);

printf("It is parent process.\n");

printf("The child process,ID number %d, is finished.\n",i);

}

else{

printf("It is child process.\n");

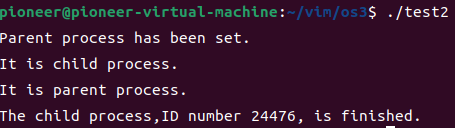
sleep(3);

}

return 0;

}

**结果2：**



**思考：子进程是如何产生的？ 又是如何结束的？子进程被创建后它的运行环境是怎样建立 的？**

**答：**子进程主要通过fork()系统调用产生，当一个进程调用fork()时，内核会创建一个新的进程，该新进程就是原进程的子进程。执行fork()时，会复制当前进程的地址空间（内存映像），为新进程分配一个唯一的进程ID(PID),复制父进程的文件描述符，设置子进程的父进程ID(PPID)为原进程的PID，复制父进程的信号处理函数和信号处置。Fork()调用后，父进程返回子进程的PID，而在子进程中返回0，若出错，返回负值。

子进程结束时，会调用exit()系统调用结束运行，关闭所有打开的文件描述符，释放用户空间分配的内存，执行终止处理程序，将子进程的退出状态传递给父进程。

子进程运行环境的建立：继承了父进程的地址空间，包括代码段、数据段、堆和栈等，但它是对这些资源的复制，后续对内存的写操作不会影响父进程。继承父进程的环境变量，这些变量定义了子进程的运行环境。继承父进程打开的文件描述符，能够访问父进程已经打开的文件和网络连接。继承父进程的权限和属性，如用户ID、组ID、当前工作目录等。继承了父进程的资源限制，如CPU时间，文件大小等。

**操作过程3：**

参考例程 2，编写程序。父进程通过循环语句创建若干子进程。探讨进程的家族树以及 子进程继承父进程的资源的关系。

修改为单分支结构后：

#include<unistd.h>

#include<stdio.h>

#include<sys/wait.h>

int main()

{

int i;

printf("My pid is %d, my father's pid is %d\n",getpid(),getppid());

for(i=0;i<3;i++)

{

//printf("before fork(), i = %d\n",i);

pid\_t pid = fork();

if(pid == 0)

{

printf(" i=%d, pid=%d, ppid=%d\n",i,getpid(),getppid());

//exit(0);

printf("i will turn to %d \n",i+1);

}

else

{

//printf("这里是父进程%d\n",getpid());

int j=-1;

j=wait(0);

printf("%d: The child %d is finished.\n",getpid(),j);

printf("i will turn to %d \n",i+1);

//printf("end fork, i = %d还未变化，即将加1\n",i);

}

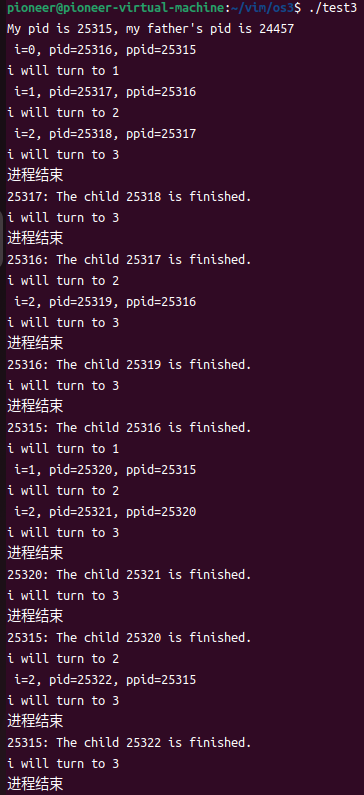
}

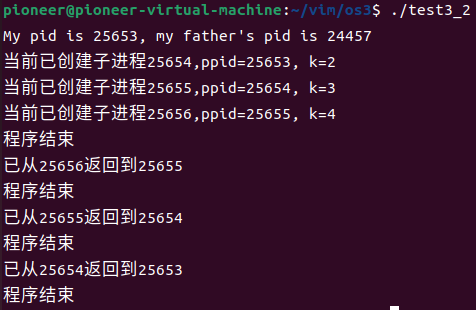
printf("进程结束\n");

return 0;

}

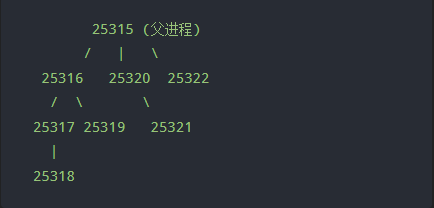
**结果3：**



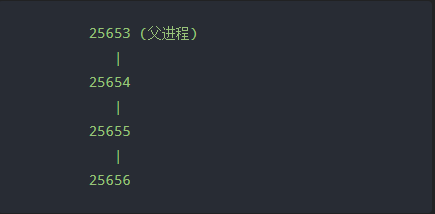


**思考：① 画出进程的家族树。子进程的运行环境是怎样建立的？反复运行此程序看会有什 么情况？解释一下。**

**答：**进程家族树：



子进程运行环境由父进程的运行环境复制而来，包括地址空间、代码段、数据段、堆和栈。反复运行此程序，会创建多个进程树，导致复杂的进程行为和资源使用情况。修改为单分支结构后的进程树：



依次创建子进程，父进程阻塞运行，累计3个后依次返回父进程，在父进程中修改k=4，防止每一层额外产生子进程，这样确保了每个父进程只有一个子进程。

**操作过程4：**

参考例程 3 编程，使用 fork()和 exec()等系统调用创建三个子进程。子进程分别启动不 同程序，并结束。反复执行该程序，观察运行结果，结束的先后，看是否有不同次序。

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

#include<stdlib.h>

int main()

{

int child\_pid1, child\_pid2,child\_pid3;

int pid, status;

setbuf(stdout,NULL);

child\_pid1=fork();

if(child\_pid1==0)

{

execlp("./hello","hello",NULL);

perror("exec1 error.\n");

exit(1);

}

child\_pid2=fork();

if(child\_pid2==0)

{

execlp("ls","ls",NULL);

perror("exec2 error.\n");

exit(2);

}

child\_pid3=fork();

if(child\_pid3==0)

{

execlp("date","date",NULL);

perror("exec3 error.\n");

exit(3);

}

puts("Parent process is waiting for child process return!");

while((pid=wait(&status))!=-1)

{

if(child\_pid1==pid)

printf("child process 1 terminated with status %d\n",(status >> 8));

else if(child\_pid2==pid)

printf("child process 2 terminated with status %d\n",(status >> 8));

else if(child\_pid3==pid)

printf("child process 3 terminated with status %d\n",(status >> 8));

}

puts("All child processes terminated.");

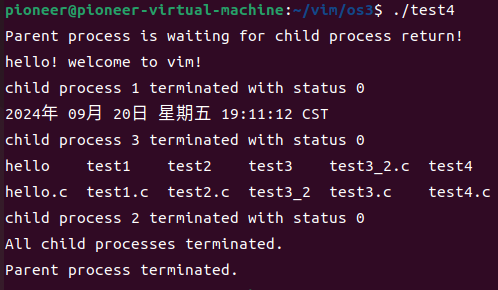
puts("Parent process terminated.");

exit(0);

return 0;

}

**结果4：**



**思考：子进程运行其它程序后，进程运行环境怎样变化的？反复运行此程序看会有 什么情况？解释一下。**

**答：**子进程运行其他程序后，会复制父进程内存空间、代码段、数据段、栈。反复运行此程序，每个子进程的打印信息的顺序可能会发生变化。因为子进程的创建顺序不可预测。Wait()函数也将随机等待子进程的结束，因此，打印信息的显示顺序也将随机。

**操作过程5：**

参考例程 4 编程，验证子进程继承父进程的程序、数据等资源。如用父、子进程修改 公共变量和私有变量的处理结果；父、子进程的程序区和数据区的位置。

#include<sys/wait.h>

#include<stdio.h>

#include<sys/types.h>

#include<unistd.h>

int global = 4;

int main()

{

pid\_t pid;

int vari = 5;

printf("before fork.\n");

printf("pid=%d, global=%d, vari=%d\n",getpid(),global,vari);

if((pid=fork()) < 0)

{

printf("fork error.\n");

\_exit(0);

}

else if(pid==0)

{

global++;

vari--;

printf("Child %d changed the vari and global.\n",getpid());

}

else

printf("Parent %d did not changed the vari and global.\n",getpid());

wait(0);

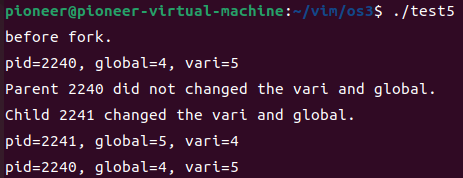
printf("pid=%d, global=%d, vari=%d\n",getpid(),global,vari);

\_exit(0);

return 0;

}

**结果5：**



**思考：子进程被创建后，对父进程的运行环境有影响吗？解释一下。**

**答：**子进程创建后，是将父进程的运行环境复制了一份，是个独立的副本。因此，对父进程的运行环境没有影响。

**操作过程6：**

参照《实验指导》第五部分中“管道操作的系统调用”。复习管道通信概念，参考例程 5，编写一个程序。父进程创建两个子进程，父子进程之间利用管道进行通信。要求能显示 父进程、子进程各自的信息，体现通信效果。

#include<stdio.h>

#include<sys/wait.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

int main()

{

int i,r,j,k,l,p1,p2,fd[2];

char buf[50],s[50];

pipe(fd);

while((p1=fork())==-1);

if(p1==0)

{

lockf(fd[1],1,0);

sprintf(buf,"Child process P1 is sending messages!\n");

printf("Childprocess P1\n");

write(fd[1],buf,50);

lockf(fd[1],0,0);

sleep(3);

j=getpid();

k=getppid();

printf("P1 %d is wake up. My parent process ID is %d.\n",j,k);

exit(0);

}

else

{

while((p2=fork())==-1);

if(p2==0)

{

lockf(fd[1],1,0);

sprintf(buf,"Child process P2 is sending messages!\n");

printf("Child process P2!\n");

write(fd[1],buf,50);

lockf(fd[1],0,0);

sleep(6);

j=getpid();

k=getppid();

printf("P2 %d is wake up. My parent process ID is %d.\n",j,k);

exit(0);

}

else

{

l=getpid();

wait(0);

if(r=read(fd[0],s,50)==-1)

printf("Can't read pipe.\n");

else

printf("Parent %d: %s \n",l,s);

wait(0);

if(r=read(fd[0],s,50)==-1)

printf("Can't read pipe. \n");

else

printf("Parent %d: %s \n",l,s);

exit(0);

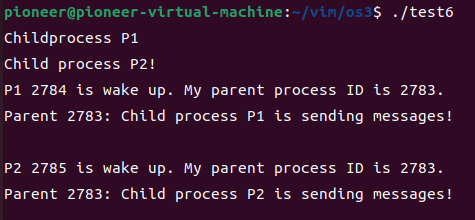
}

}

return 0;

}

**结果6：**



**思考：**

**① 什么是管道？进程如何利用它进行通信的？解释一下实现方法。**

**② 修改睡眠时机、睡眠长度，看看会有什么变化。请解释。**

**③ 加锁、解锁起什么作用？不用它行吗？**

**答：**管道是Unix和类Unix操作系统中用于进程间通信的一种机制。它允许在父子进程间或兄弟进程间进行数据传输。进程通过创建管道，fork进程，关闭不必要的管道端，通过write写数据，read读数据。完成通信。

如果修改睡眠时长，返回父进程后，打印信息的等待时间会变化。因为在子进程中的sleep会阻塞子进程的运行。

对写入端加锁，保证在同一时刻只能有一个进程访问共享资源——管道，防止其他进程干扰。解锁：释放对共享资源的锁，使得其他进程能够访问资源。如果不使用锁，会造成多个进程同时修改同一数据，导致数据不一致而崩溃。

**操作过程7：**

编程验证：实现父子进程通过管道进行通信。进一步编程，验证子进程结束，由父进程 执行撤消进程的操作。测试父进程先于子进程结束时，系统如何处理“孤儿进程”的。

#include<stdio.h>

#include<sys/wait.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

int main()

{

int i,r,j,k,l,p1,p2,fd[2],status;

char buf[50],s[50];

pipe(fd);

while((p1=fork())==-1);

if(p1==0)

{

lockf(fd[1],1,0);

sprintf(buf,"Child process P1 is sending messages!\n");

printf("Childprocess P1\n");

write(fd[1],buf,50);

lockf(fd[1],0,0);

sleep(3);

j=getpid();

k=getppid();

printf("P1 %d is wake up. My parent process ID is %d.\n",j,k);

exit(0);

}

else

{

while((p2=fork())==-1);

if(p2==0)

{

lockf(fd[1],1,0);

sprintf(buf,"Child process P2 is sending messages!\n");

printf("Child process P2!\n");

write(fd[1],buf,50);

lockf(fd[1],0,0);

sleep(6);

j=getpid();

k=getpid();

printf("P2 %d is wake up. My parent process ID is %d.\n",j,k);

exit(0);

}

else

{

l=getpid();

waitpid(-1,&status,0);

printf("Child ended with status %d\n",status);

if(r=read(fd[0],s,50)==-1)

printf("Can't read pipe.\n");

else

printf("Parent %d: %s \n",l,s);

//wait(0);

if(r=read(fd[0],s,50)==-1)

printf("Can't read pipe. \n");

else

printf("Parent %d: %s \n",l,s);

exit(0);

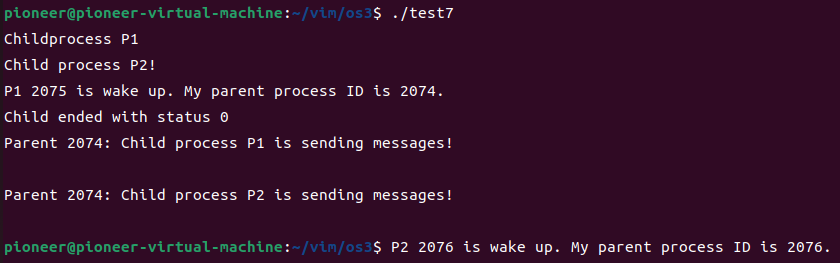
}

}

return 0;

}

**结果7：**



**思考：对此作何感想，自己动手试一试？解释一下你的实现方法。**

**答：**我通过让子进程P2进行sleep足够长时间，删除父进程中第二个wait，做到父进程先于子进程P2结束，可以看到，P2通过管道发送的信息成功达到父进程2074，但父进程2074结束后，子进程P2由init进程，也就是系统主进程接管。

**操作过程8：**

编写两个程序一个是服务者程序，一个是客户程序。执行两个进程之间通过消息机制通 信。消息标识 MSGKEY 可用常量定义，以便双方都可以利用。客户将自己的进程标识（pid） 通过消息机制发送给服务者进程。服务者进程收到消息后，将自己的进程号和父进程号发送 给客户，然后返回。客户收到后显示服务者的 pid 和 ppid，结束。以下例程 6 基本实现以上 功能。这部分内容涉及《实验指导》第五部分中“IPC 系统调用”。先熟悉一下，再调试程序。

后台启动服务器方式：

#!/bin/bash

# 启动服务器并在后台运行

./test8\_s &

for(( i = 1; i <= 3; i++))

do

echo "running client $i"

./test8\_c

sleep 3

done

kill $!

Server:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <unistd.h>

#define MSGKEY 1234

struct message {

long mtype;

pid\_t pid;

pid\_t ppid;

};

int main() {

int msgid;

struct message msg;

// 创建或获取消息队列

msgid = msgget(MSGKEY, 0666 | IPC\_CREAT);

if (msgid == -1)

{

perror("msgget failed");

exit(EXIT\_FAILURE);

}

// 接收客户端消息

while(1)

{

if (msgrcv(msgid, &msg, sizeof(msg) - sizeof(long), 0, 0) == -1)

{

perror("msgrcv failed");

exit(EXIT\_FAILURE);

}

printf("here is server,received PID from client: %ld\n", (long)msg.pid);

//键入回复消息

msg.mtype = msg.pid;

msg.pid = getpid();

msg.ppid = getppid();

// 发送消息给客户端

if (msgsnd(msgid, &msg, sizeof(msg) - sizeof(long), 0) == -1)

{

perror("msgsnd failed");

exit(EXIT\_FAILURE);

}

}

// 清理消息队列

msgctl(msgid, IPC\_RMID, NULL);

return 0;

}

Client:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <unistd.h>

#define MSGKEY 1234

struct message {

long mtype;

pid\_t pid;

pid\_t ppid;

};

int main() {

int msgid;

struct message msg;

// 创建或获取消息队列

msgid = msgget(MSGKEY, 0666 | IPC\_CREAT);

if (msgid == -1) {

perror("msgget failed");

exit(EXIT\_FAILURE);

}

// 准备发送的消息

msg.mtype = 1; // 消息类型

msg.pid = getpid();

// 发送消息到服务者

if (msgsnd(msgid, &msg, sizeof(msg) - sizeof(long), 0) == -1) {

perror("msgsnd failed");

exit(EXIT\_FAILURE);

}

// 接收服务者的响应

if (msgrcv(msgid, &msg, sizeof(msg) - sizeof(long), msg.pid, 0) == -1) {

perror("msgrcv failed");

exit(EXIT\_FAILURE);

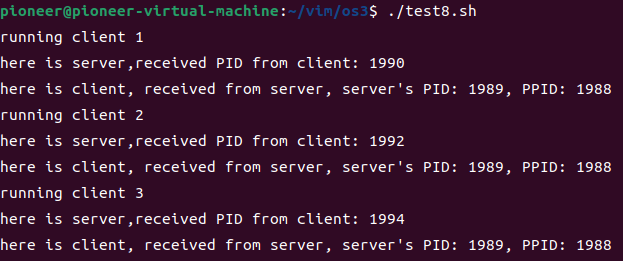
}

printf("here is client, received from server, server's PID: %ld, PPID: %ld\n", (long)msg.pid, (long)msg.ppid);

return 0;

}

**结果8：**



**思考：想一下服务者程序和客户程序的通信还有什么方法可以实现？解释一下你的设想，有 兴趣试一试吗。**

**答：**服务端和客户端的通信，可以通过管道pipes，套接字socket，消息队列message queues,共享内存shared memory,信号量semaphores，信号signals，远程过程调用RPC,HTTP/HTTPS等等。

**操作过程9：**

这部分内容涉及《实验指导》第五部分中“有关信号处理的系统调用”。编程实现软中 断信号通信。父进程设定软中断信号处理程序，向子进程发软中断信号。子进程收到信号后 执行相应处理程序。

#include<stdio.h>

#include<stdlib.h>

#include<sys/wait.h>

#include<unistd.h>

int main()

{

int i,j;

pid\_t k;

void func();

signal(18,func);

if(i = fork())

{

j = kill(i,18);

printf("parent: signal 18 has been sent to child %d, returned %d.\n",i,j);

k = wait(0);

printf("after wait %d, parent %d: finished.\n",k,getpid());

}

else

{

sleep(3);

printf("child %d : a signal from my parent is received.\n",getpid());

}

return 0;

}

void func()

{

printf("I am process %d: it is signal 18 processing function.\n",getpid());

}

**结果9：**



**思考：这就是软中断信号处理，有点儿明白了吧？讨论一下它与硬中断有什么区别？看来还 挺管用，好好利用它。**

**答：** 软中断信号是软件层面的、同步的、与进程相关联的，提供了高度的可编程性和灵活性，适合于处理复杂的进程间通信和同步问题。而硬中断信号是硬件层面的、异步的、与硬件事件相关联，提供了快速、直接且低延迟的硬件事件处理能力，各自发挥着关键作用。

**操作过程10：**

怎么样，试一下吗？用信号量机制编写一个解决生产者—消费者问题的程序，这可是受 益匪浅的事。本《实验指导》第五部分有关进程通信的系统调用中介绍了信号量机制的使用。

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <sys/types.h>

#include<sys/wait.h>

#define BUFFER\_SIZE 10

int sem\_id; // 信号量集的标识符

struct sembuf sem\_op; // 信号量操作结构体

void producer() {

for (int i = 0; i < 5; ++i) {

// P操作，减缓冲区空余空间的信号量

sem\_op.sem\_num = 0;

sem\_op.sem\_op = -1;

sem\_op.sem\_flg = 0;

semop(sem\_id, &sem\_op, 1);

// 生产数据，这里简单打印出来

printf("Producer: Produced item %d\n", i);

// V操作，增加缓冲区物品的信号量

sem\_op.sem\_num = 1;

sem\_op.sem\_op = 1;

sem\_op.sem\_flg = 0;

semop(sem\_id, &sem\_op, 1);

sleep(2); // 随机休眠一段时间

}

}

void consumer() {

for (int i = 0; i < 5; ++i) {

// P操作，减缓冲区物品的信号量

sem\_op.sem\_num = 1;

sem\_op.sem\_op = -1;

sem\_op.sem\_flg = 0;

semop(sem\_id, &sem\_op, 1);

// 消费数据，这里简单打印出来

printf("Consumer: Consumed item %d\n", i);

// V操作，增加缓冲区空余空间的信号量

sem\_op.sem\_num = 0;

sem\_op.sem\_op = 1;

sem\_op.sem\_flg = 0;

semop(sem\_id, &sem\_op, 1);

sleep(rand() % 3); // 随机休眠一段时间

}

}

int main() {

// 创建一个信号量集，包含两个信号量

sem\_id = semget(IPC\_PRIVATE, 2, 0666 | IPC\_CREAT);

// 初始化信号量，第一个为缓冲区空余空间，第二个为缓冲区物品数量

semctl(sem\_id, 0, SETVAL, BUFFER\_SIZE); // 空余空间初始为缓冲区大小

semctl(sem\_id, 1, SETVAL, 0); // 缓冲区初始为空

pid\_t pid = fork(); // 创建子进程

if (pid == 0) {

// 子进程为消费者

consumer();

} else if (pid > 0) {

// 父进程为生产者

producer();

wait(NULL); // 等待子进程结束

} else {

// 创建子进程失败

perror("fork");

exit(EXIT\_FAILURE);

}

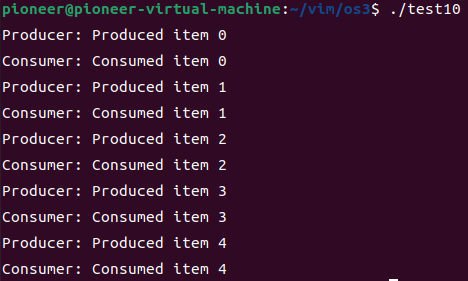
// 删除信号量集

semctl(sem\_id, 0, IPC\_RMID, 0);

return 0;

}

**结果10：**



**讨论：**

1. **讨论Linux 系统进程运行的机制和特点，系统通过什么来管理进程？**

**答：**Linux进程运行机制包括：进程控制块，包含了进程的状态信息、程序计数器、寄存器、内存限制等；进程状态，创建与终止；进程调度；上下文切换。系统通过进程表、系统调用、调度算法、进程间通信机制、资源管理等来管理进程。

1. **C 语言中是如何使用Linux 提供的功能的？用程序及运行结果举例说明。**

**答：**通过包含指定的头文件，即可使用linux相关的系统调用。如：

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include<sys/wait.h>

int main() {

int i;

printf("My pid is %d, my father's pid is %d\n", getpid(), getppid());

int k = 1;

while(k++ <= 3)

{

pid\_t pid = fork();

if(pid == 0)

{

printf("当前已创建子进程%d,ppid=%d, k=%d\n",getpid(),getppid(),k);

continue;

}

else

{

int i=-1;

i = wait(0);

printf("已从%d返回到%d\n",i,getpid());

k = 4;

}

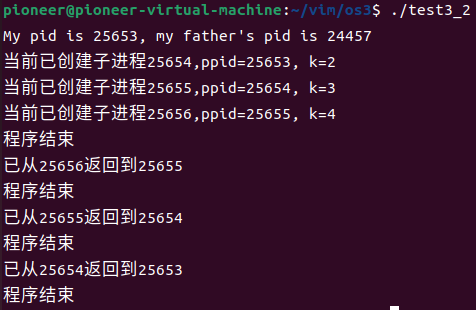
}

printf("程序结束\n");

return 0;

}

运行结果：

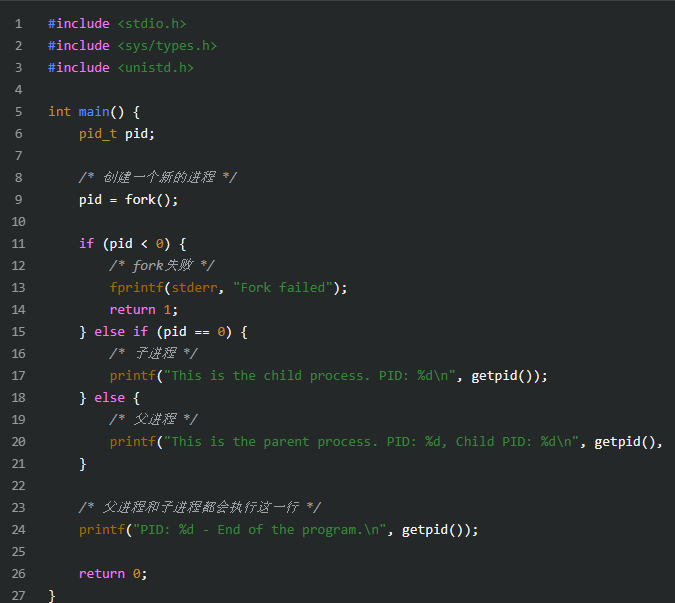


通过包含头文件unistd.h,使用了进程创建功能。

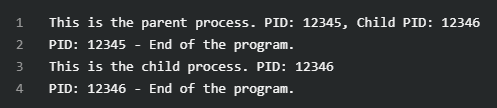
1. **什么是进程？如何产生的？举例说明。**

答：进程是计算机中程序执行的基本单元，是操作系统进行资源分配和调度的基本单位。在linux中，进程通常被视为一个正在执行的程序的实例。

进程可以通过系统初始化，程序执行，派生子进程产生。



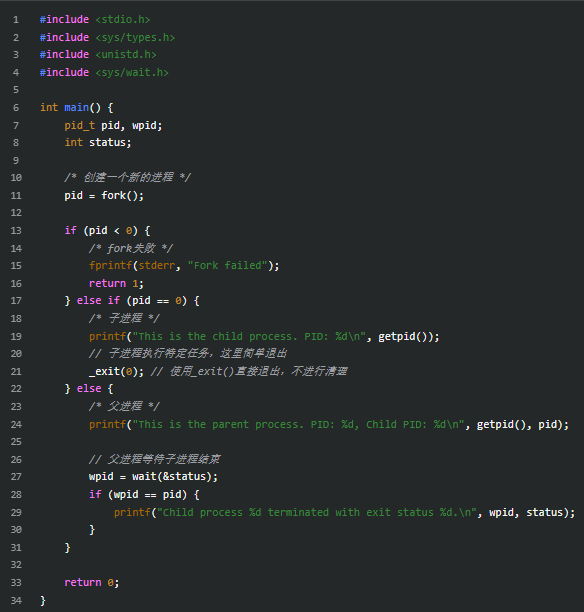
在该例子中，fork()系统调用被用来创建一个新的进程，fork()调用后，会有两个进程：父进程和子进程。编译该程序，会得到类似于以下结果：



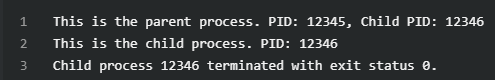
该输出显示了父进程和子进程都执行了相同的代码，但它们是不同的进程，有各自的进程ID。

1. **进程控制如何实现的？举例说明。**

**答：**进程控制可以通过系统调用、PCB进程控制块和调度算法实现。如：



在该例子中，fork()创建了一个新的子进程，\_exit()子进程完成任务后，直接退出，wait()父进程调用wait()来等待子进程结束，wait()会阻塞父进程，直到子进程结束，status参数用于获取子进程的退出状态。编译并运行该程序，会得到类似于以下结果：



1. **进程通信方式各有什么特点？用程序及运行结果举例说明。**

**答：**进程通信方式包括：管道，半双工，只能在具有亲缘关系的进程间通信，如父子进程；消息队列，可以允许一个或多个进程写入和读取数据，消息以队列形式存储；信号量，主要用于同步，确保多个进程可以安全地访问共享资源；共享内存，最快的IPC方式，不涉及数据的复制，允许多个进程访问同一块内存空间，适用于大量数据的传输，但需要同步机制（如信号量）来避免竞态条件；套接字，可以用于不同机器上的进程间通信，支持网络通信，分为流式TCP和数据报UDP。

1. **管道通信如何实现？该通信方式可以用在何处？**

**答：**创建管道，创建子进程，配置管道读写端，数据传输，关闭管道。可用于命令链、 数据转换和过滤、脚本编程、日志处理、文本处理、文件传输、进程间数据传递、系统监控等。

1. **什么是软中断？软中断信号通信如何实现？**

**答：**软中断是一种由软件产生的中断信号，它允许程序在执行途中暂停，并调用操作系统内核提供的服务。其通常用于执行系统调用或处理异步事件。

实现流程大概包括：设置信号处理程序signal(SIG,signal\_handler);

发送信号kill(pid,SIG);

处理信号,当信号被发送到相应进程时，操作系统会暂停当前执行的代码，并调用处理程序signal\_handler，信号处理程序执行完毕后，操作系统会恢复被中断的代码继续执行。

**体会：**

在本次操作系统实验中，我深入学习了进程管理及进程通信的相关知识，通过设计和实现程序，我对进程的概念有了更加深刻的理解，同时也对操作系统中的进程调度方法和进程间通信机制有了直观的认识。

首先，在进程管理方面，我通过编写简单的程序，使用了Linux提供的系统调用来创建、终止和管理进程。通过这些操作，我体会到了进程在操作系统中的基本状态转换，如创建（fork）、执行（exec）、等待（wait）和终止（exit）。这些系统调用不仅让我理解了进程的生老病死，还让我认识到了进程在操作系统中作为资源分配和调度基本单位的重要性。

在进程调度方面，我观察到不同调度算法对系统性能的影响。通过设置不同的进程优先级和调度策略，我体会到了操作系统的任务是如何在多个进程间进行分配的。例如，我尝试了使用实时调度策略和非实时调度策略，发现实时调度能够更快地响应外部事件，而非实时调度则在长任务执行时更为高效。这些实验让我对操作系统的调度策略有了更加直观的感受，也让我理解了调度器在保证系统响应性和吞吐量之间所做的权衡。

在进程通信方面，我实现了基于管道、信号和消息队列的通信机制。以下是我的一些体会：

1. 管道（Pipe）通信：管道是一种半双工的通信方式，通过它，我实现了父子进程间的数据传递。我体会到管道在进程间传递信息的便捷性，但同时也注意到了管道的局限性，比如它只能用于有亲缘关系的进程间通信。

2. 信号（Signal）通信：信号提供了一种异步的通知机制，可以用来通知接收进程某个事件已经发生。通过使用信号，我理解了信号处理函数的作用，以及如何通过信号进行简单的进程间协作。

3. 消息队列（Message Queue）通信：消息队列提供了一种在进程间传递结构化数据的机制。通过它，我实现了进程间的数据交换，并体会到了消息队列在数据传输上的有序性和可靠性。

这次实验极大地丰富了我对操作系统进程管理的认识，也让我对进程间通信的多种方式有了实际操作的经验。通过这些实验，我更加明白了操作系统在管理进程和协调进程间通信中所扮演的关键角色，这对于我未来深入学习操作系统理论和实践具有重要的指导意义。

# 实验四题目：LINUX进程调度与系统监视

**姓名：汪江豪 学号：22121630 实验日期：2024/10/10**

**实验环境：**

Linux操作系统

**实验目的：**

1. 熟练掌握手工启动前后台作业的方法。
2. 熟练掌握进程与作业管理的相关 Shell 命令。
3. 掌握 at 调度和 cron 调度的设置方法。
4. 了解进行系统性能监视的基本方法。

**实验内容：**

1. 作业和进程的基本管理
2. at进程调度
3. cron进程调度
4. 系统性能监视

一、作业和进程的基本管理

操作过程1：

【操作要求 1】先在前台启动 vi 编辑器并打开 f4 文件，然后挂起，最后在后台启动一个查 找 inittab 文件的 find 作业，find 的查找结果保存到 f5。

【操作步骤】

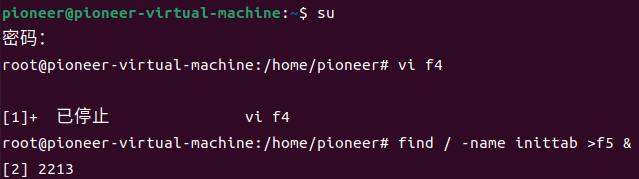
（1） 以超级用户（root）身份登录到 RHEL Server 5 字符界面。

（2） 输入命令“vi f4”，在前台启动 vi 文本编辑器并打开 f4 文件。

（3） 按下 Ctrl+Z 组合键，暂时挂起“vi f4”作业，屏幕显示该作业的作业号。

（4） 输入命令“find / -name inittab > f5 &”，启动一个后台作业，如下所示。在显 示作业号的同时还显示进程号。

结果1：



操作过程2：

【操作要求 2】查看当前作业、进程和用户信息，并对作业进行前后台切换

【操作步骤】

① 输入命令“jobs”，查看当前系统中的所有作业。

② 输入命令“fg 2”，将“find / -name inittab > f5 &”作业切换到前台。屏幕显示出“find / - name inittab > f5”命令，并执行此命令。稍等片刻，作业完成后屏幕再次出现命令提 示符。

③ 输入命令“cat f5”，查看“find / -name inittab > f5”命令的执行结果。

④ 再次输入命令“jobs”，可发现当前系统中的只有一个已停止的作业“vi f4”。

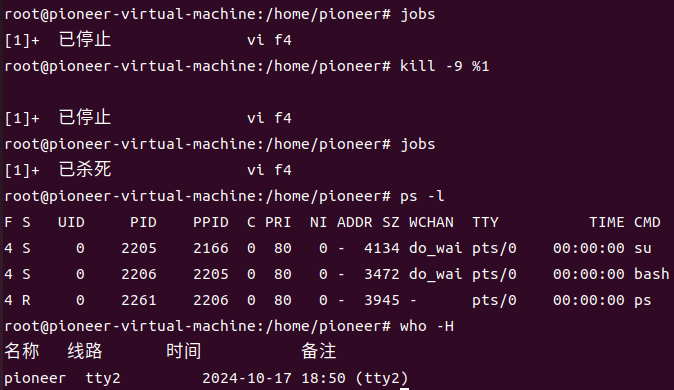
⑤ 输入命令“kill -9 %1”，终止“vi f4”作业。

⑥ 稍等片刻，输入命令“jobs”，查看到当前没有任何作业。

⑦ 输入命令“ps –l”，查看进程的相关信息，显示出的信息类似如下信息。

⑧ 输入命令“who –H”，查看用户信息。

结果2：



二、at进程调度

操作过程1：

【操作要求 1】设置一个调度，要求在 2008 年 1 月 1 日 0 时，向所有用户发送新年快乐的问候。

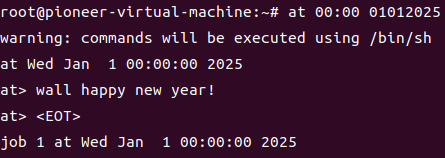
【操作步骤】

① 超级用户输入命令“at 00:00 01012008”，设置 2008 年 1 月 1 日 0 时执行的 at 调度的内容。

② 屏幕出现 at 调度的命令提示符“at>”，输入“wall Happy New Year!”，向所有用户发送消息。

③ 光标移动到“at>”提示符的第三行，按下 Ctrl+D 组合键结束输入。根据调度设置的 时间，最后显示出作业号和将要运行的时间。

结果1：



操作过程2：

【操作要求 2】设置一个调度，要求 5 分钟后向所有用户发送系统即将重启的消息，并在 2 分钟后重新启动计算机

【操作步骤】

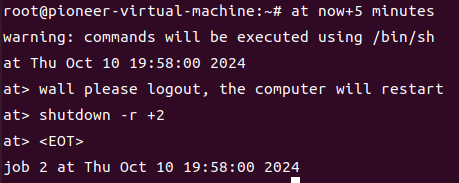
① 超级用户输入命令“at now +5 minutes”，设置 5 分钟后执行的 at 调度的内容。

② 屏幕出现 at 调度的命令提示符“at>”，输入“wall please logout; the computer will restart.”，向所有用户发送消息。

③ 在“at>”提示符的第二行输入“shutdown –r +2”，系统 2 分钟后将重新启动。

④ 光标移动到“at>”提示符的第三行，按下 Ctrl+D 组合键结束输入。最后显示作业号 和运行时间。

结果2：



操作过程3：

【操作要求 3】查看所有的 at 调度，并删除 08 年 1 月 1 日执行的调度任务

【操作步骤】

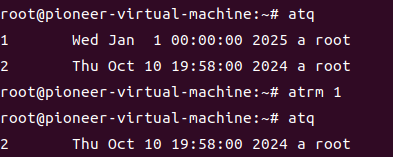
方法一：

① 输入“atq”命令，查看所有的 at 调度，显示出作业号、将在何时运行以及 at 调度的 设定者。

② 输入“atrm1”命令删除作业号为 1 的 at 调度，并再次输入“atq”命令查看剩余的所有 at 调度内容。

③ 5 分钟后系统将自动运行作业号为 2 的 at 调度内容。先向所有用户发送消息，然后， 再等 2 分钟重新启动。

结果3：



三、cron进程调度

操作过程1：

【操作要求 1】helen 用户设置 crontab 调度，要求每天上午 8 点 30 份查看系统的进程状态， 并将查看结果保存于 ps.log 文件

【操作步骤】

① 以普通用户 helen 登录，并输入命令“crontab –e”，新建一个 crontab 配置文件。

② 屏幕出现 vi 编辑器，按下“i”，进入输入模式，输入“30 8 \* \* \* ps >ps.log ”。

③ 按下 Esc 键退出 vi 的文本输入模式，并按下“：”键切换到最后行模式，输入“wq”，

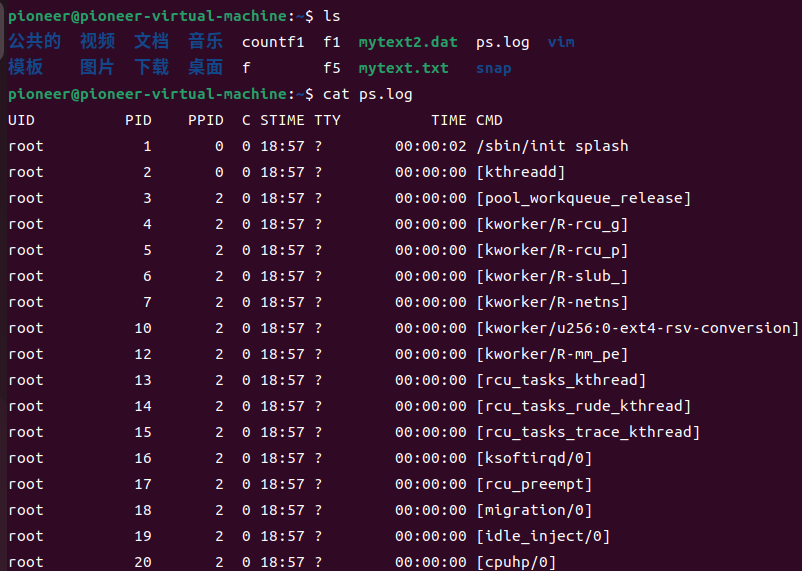
④ 保存并退出编辑器，显示“crontab: installing new crontab”信息。

⑤ 输入命令“crontab -l”，查看 helen 用户的 cron 调度内容。

⑥ 为立即查看到 crontab 调度的结果，切换为超级用户，并适当修改系统时间，如修 改为 8 点 29 分。最后退回到 helen 用户。

⑦ 等待 1 分钟后，查看 ps.log 文件的内容，如果显示出正确的内容，那么说明 crontab 调度设置成功。

结果1：



操作过程2：

【操作要求 2】helen 用户添加设置 crontab 调度，要求每三个月的 1 号零时查看正在使用的 用户列表

【操作步骤】

① 再次输入命令“crontab –e”，出现 vi 编辑器，按下“i”，屏幕进入文本输入模式。

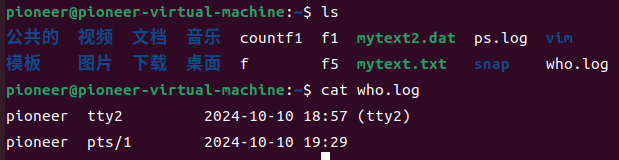
② 在原有内容之后，另起一行，输入“0 0 \* \*/3 \* who >who.log ”。

③ 最后保存并退出 vi 编辑器。

④ 为立即查看到 crontab 调度的结果，切换为超级用户，并适当修改系统时间，如修 改为 3 月 31 日 23 点 59 分。最后退回到 helen 用户。

⑤ 等待 1 分钟后，查看 who.log 文件的内容，如果显示出正确的内容，那么说明新增 加的 crontab 调度设置成功。

结果2：



操作过程3：

【操作要求 3】查看 cron 调度内容，最后删除此调度

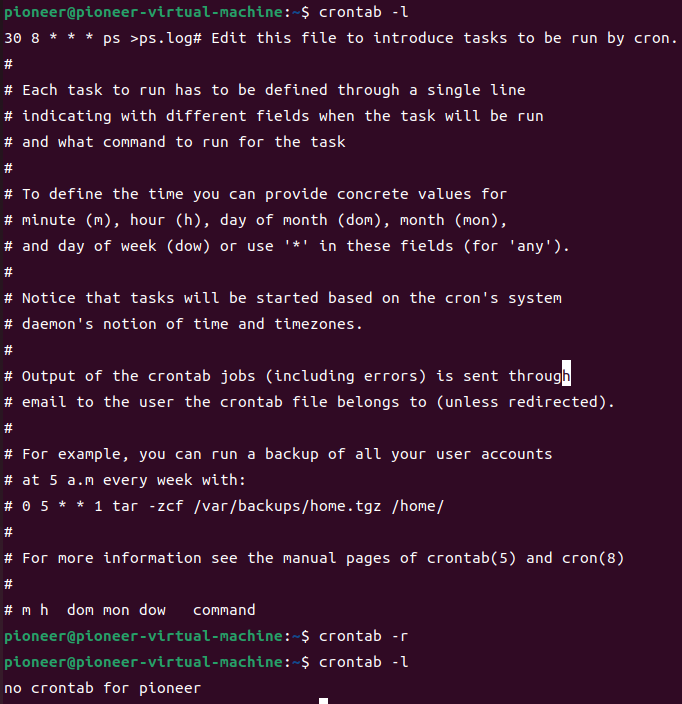
【操作步骤】

① 输入命令“crontab -l”，查看 cron 调度内容。

② 输入命令“crontab -r”，删除 cron 调度内容。

③ 再次输入命令“crontab -l”，此时无 cron 调度内容。

结果3：



四、系统性能监视

操作过程1：

【操作要求 1】利用 Shell 命令监视系统性能

【操作步骤】

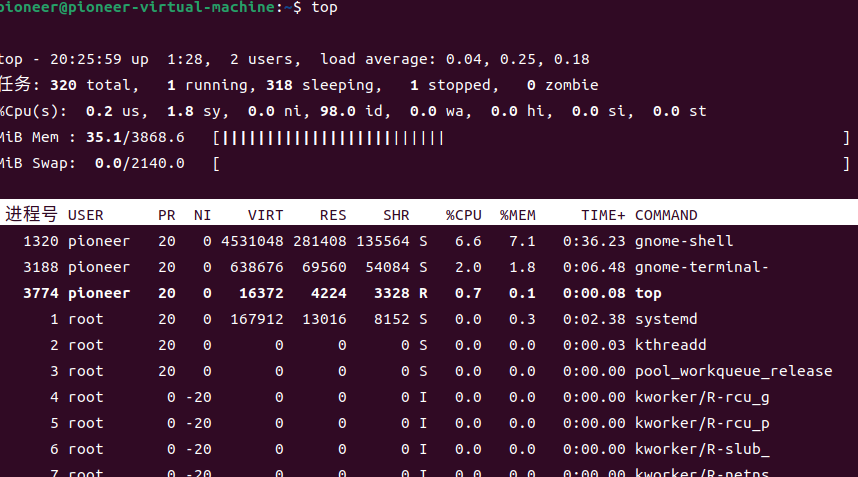
① 输入命令“top”，屏幕动态显示 CPU 利用率、内存利用率和进程状态等相关信息。

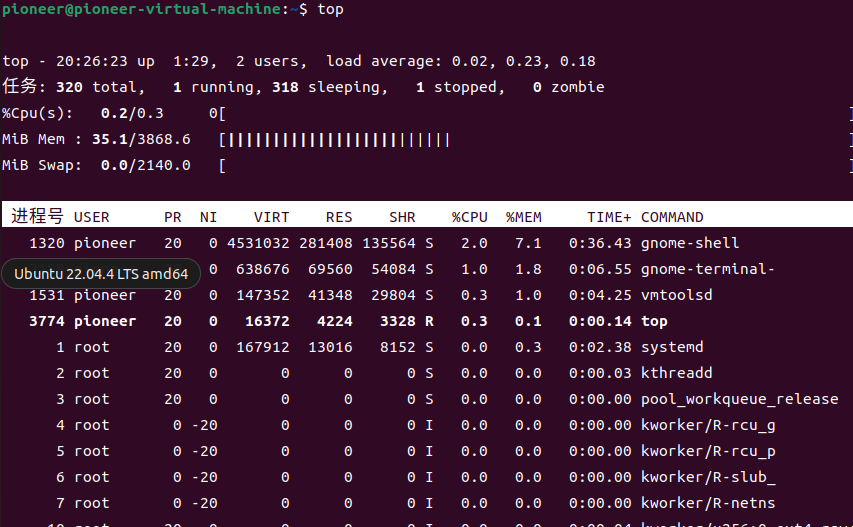
② 按下 M 键，所有进程按照内存使用率排列。

③ 按下 T 键，所有进程按照执行时间排列。

④ 最后按下 P 键，恢复按照 CPU 使用率排列所有进程。 ⑤ 按下 CTRL+C 组合键结束 top 命令。

结果1：





操作过程2：

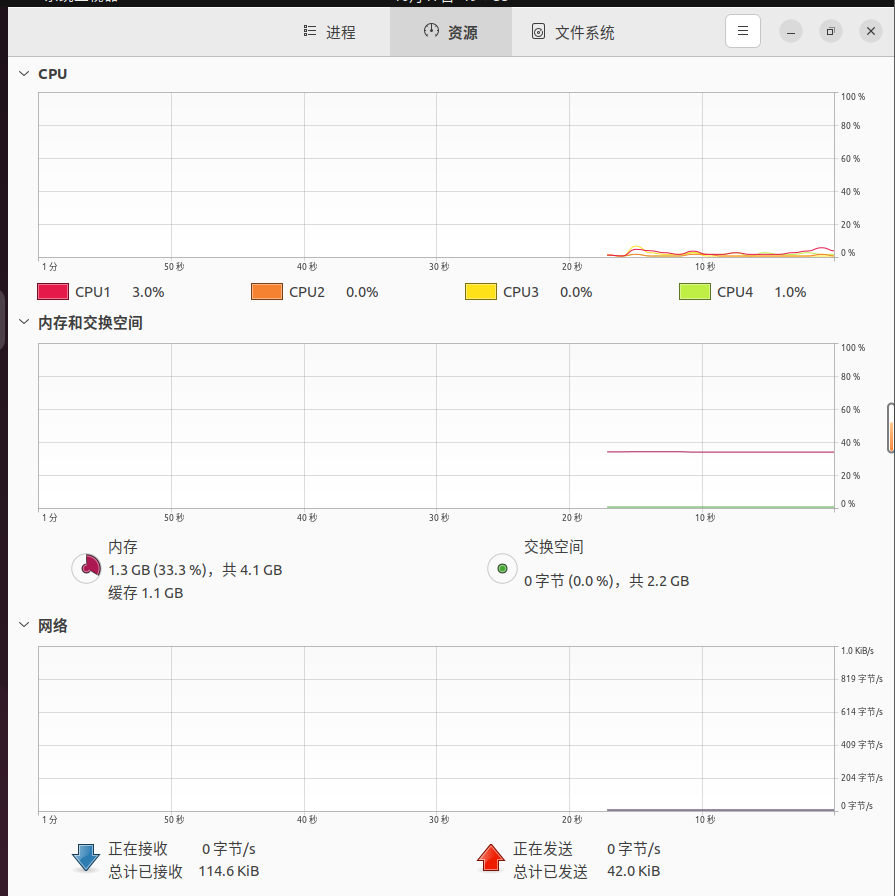
【操作要求 2】利用「系统监视器」工具监视 CPU 使用情况

【操作步骤】

① 启动 GNOME 桌面环境，依次单击「系统」菜单=>「管理」=>「系统监视器」，打 开「系统监视器」窗口。

② 自动显示「资源」选项卡，查看当前 CPU、内存和交换分区、网络历史的使用情况。

结果2：



操作过程3：

【操作要求 3】利用「系统监视器」查看当前所有的进程，要求显示出启动进程的用户 【操作步骤】

① 在「系统监视器」窗口单击「进程列表」选项卡，默认显示当前用户启动的所有进 程。单击「查看」菜单，选中「所有的进程」单选按钮，并选中「依赖关系」复选 框，则显示系统中所有的进程。

② 单击「编辑」菜单中的「首选项」，弹出「系统监视器首选项」对话框。在「进程」 选项卡，选中「进程域」栏的「用户」复选框，要求显示出启动进程的用户。单击 「关闭」按钮，显示进程的各种信息。

结果3：



操作过程4：

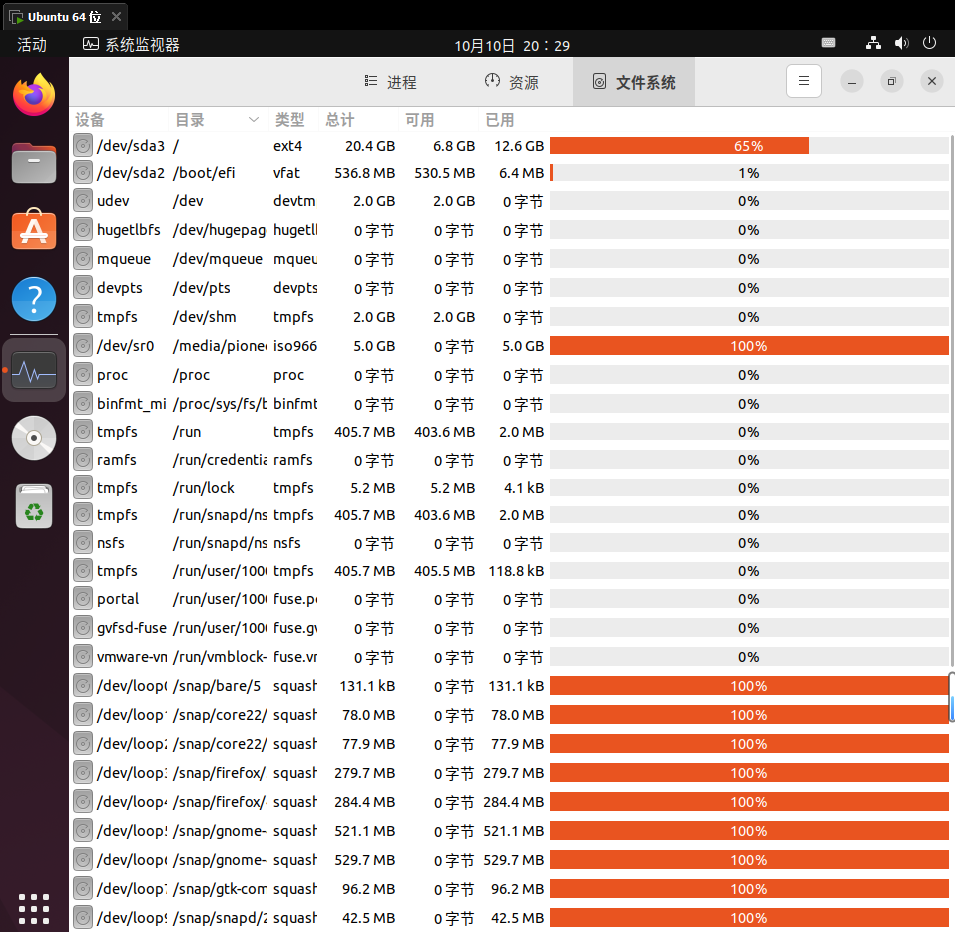
【操作要求 4】利用「系统监视器」查看所有的文件系统

【操作步骤】

① 在「系统监视器」窗口单击「文件系统」选项卡，显示当前 RHELServer5 系统中主 要的文件系统。

② 单击「编辑」菜单中的「首选项」，弹出「系统监视器首选项」对话框。在「文件系 统」选项卡，选中「显示全部文件系统」复选框，要求显示出全部的文件系统。最 后单击「关闭」按钮。

结果4：



操作过程5：

【操作要求 5】利用「系统日志」工具查看系统日志

系统日志文件都保存于/var/log 目录中，包括以下重要的日志文件：

boot.log 记录系统引导的相关信息

cron 记录 cron 调度的执行情况

dmesg 记录内核启动时的信息，主要包括硬件和文件系统的启动信息

maillog 记录邮件服务器的相关信息

messages 记录系统运行过程的相关信息，包括 I/O、网络等

rpmpkgs 记录已安装的 RPM 软件包信息

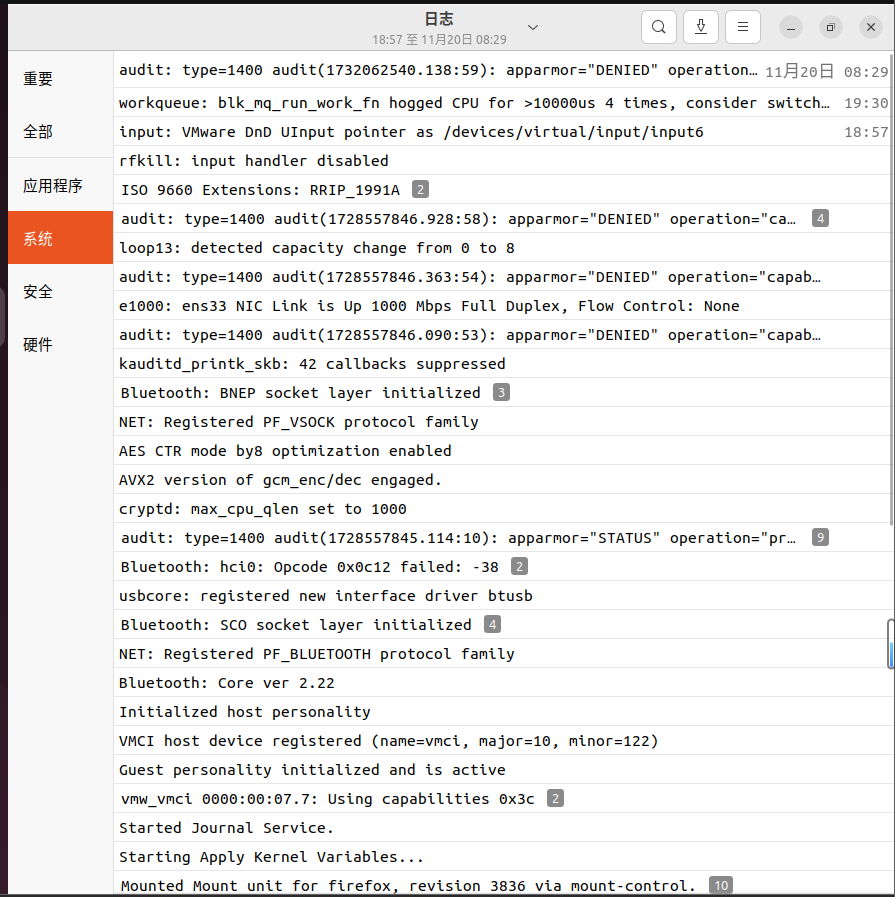
secure 记录系统安全信息

Xorg.0.log 记录图形化用户界面的 Xorg 服务器的相关信息

【操作步骤】

超级用户依次单击「系统」菜单=>「管理」=>「系统日志」，打开「系统日志」窗口。 可分别查看各类系统日志。

结果5：



实验体会：

本次linux进程调度与系统监视实验后,我对进程调度算法,优先级调度等有了更加直观和深入的理解,我认识到了不同调度算法对系统性能、响应时间、吞吐量等方面的影响.并且,系统监视及其重要,包括实时监控系统状态,CPU使用率、内存使用情况、进程状态等,对于系统管理和性能调优非常重要。

其次，我认识到了系统资源管理的复杂性，高校合理地管理计算机系统资源是一项复杂的工作，需要综合考虑多种因素，如进程优先级、系统负载、资源需求等。

操作系统是一个庞大且复杂的系统，我知道这次实验所学到的只是冰山一角，因此这也培养了我持续学习和探索的态度。

# 实验五题目：用户与群组管理

**姓名：汪江豪 学号：22121630 实验日期：2024/10/17**

**实验环境：**

Ubuntu22.04操作系统

**实验目的：**

1. 理解/etc/passwd 和/etc/group 文件的含义。

2. 掌握桌面环境下管理用户与组群的方法。

3. 掌握利用 Shell 命令管理用户与组群的方法。

4. 掌握批量新建用户帐号的步骤和方法。

**实验内容：**

1. 桌面环境下管理用户与组群

2. 利用 Shell 命令管理用户与组群

3. 批量新建多个用户帐号

**一、桌面环境下管理用户与组群**

**操作过程1：**

【操作要求 1】新建两个用户帐号，其用户名为 xuser1 和 xuser2，口令为“e12ut59er”和 “wfu1t28er”

【操作步骤】

① 以超级用户身份登录 XWindow 图形化用户界面，依次单击「系统」菜单=>「管理」 =>「用户和组群」，启动「用户管理者」窗口。

② 单击工具栏上的「添加用户」按钮，出现「创建新用户」窗口。在「用户名」文本 框中输入用户名“xuser1”，在「口令」文本框中输入口令“e12ut59er”，在「确认口令」 文本框中再次输入口令“e12ut59er”，然后单击「确定」，返回「用户管理者」窗口。

③ 用同样的方法新建用户 xuser2。

④ 依次单击顶部面板的「应用程序」=>「附件」=>「文本编辑器」，启动 gedit 文本 编辑器，打开/etc/passwd 和/etc/shadow 文件将发现文件的末尾出现表示 xuser1 和 xuser2 用户帐号的信息。打开/etc/group 和/etc/gshadow 文件将发现文件末尾出现表 示 xuser1 和 xuser2 私人组群的信息。

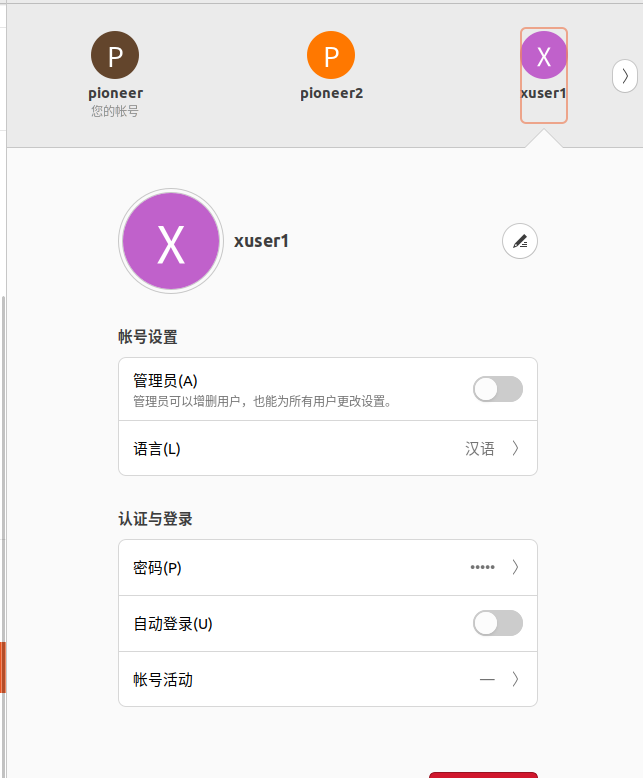
⑤ 按下 CTRL+ALT+F2 组合键切换到第 2 个虚拟终端，输入用户名 xuser2 和相应的 口令可登录 Linux 系统，说明新建用户操作已成功。

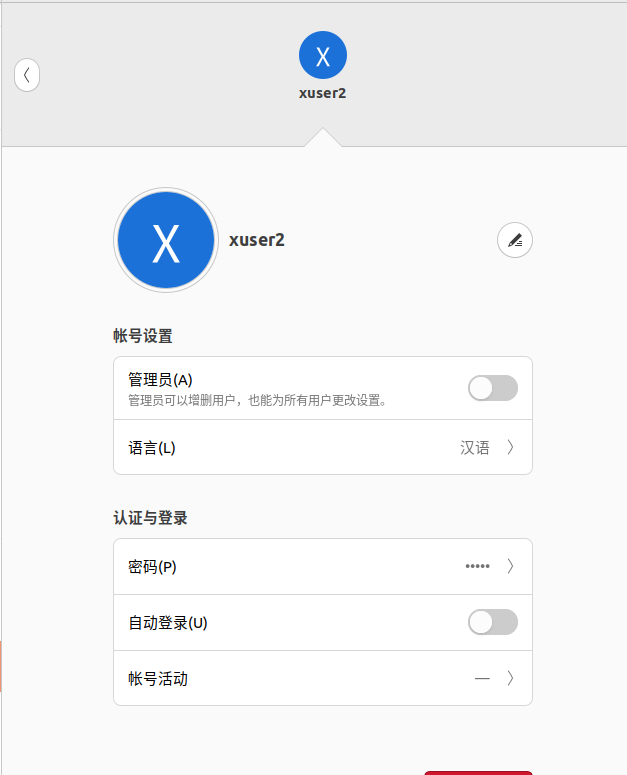
⑥ 输入“pwd”命令，屏幕显示用户登录后进入用户主目录“/home/xuser2”。

⑦ 输入“exit”命令，xuser2 用户退出登录。

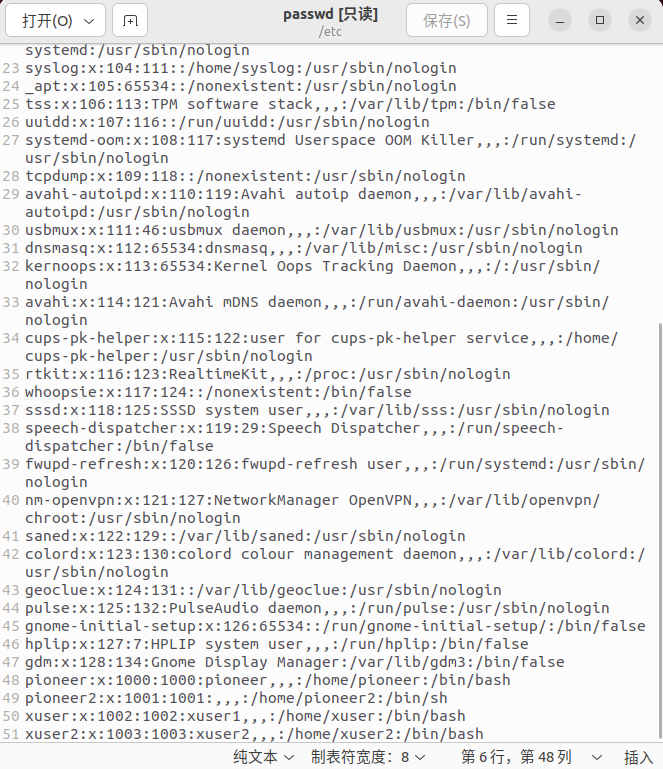
⑧ 按下 ALT+F7 组合键返回 GNOME 桌面环境。

**结果1：**

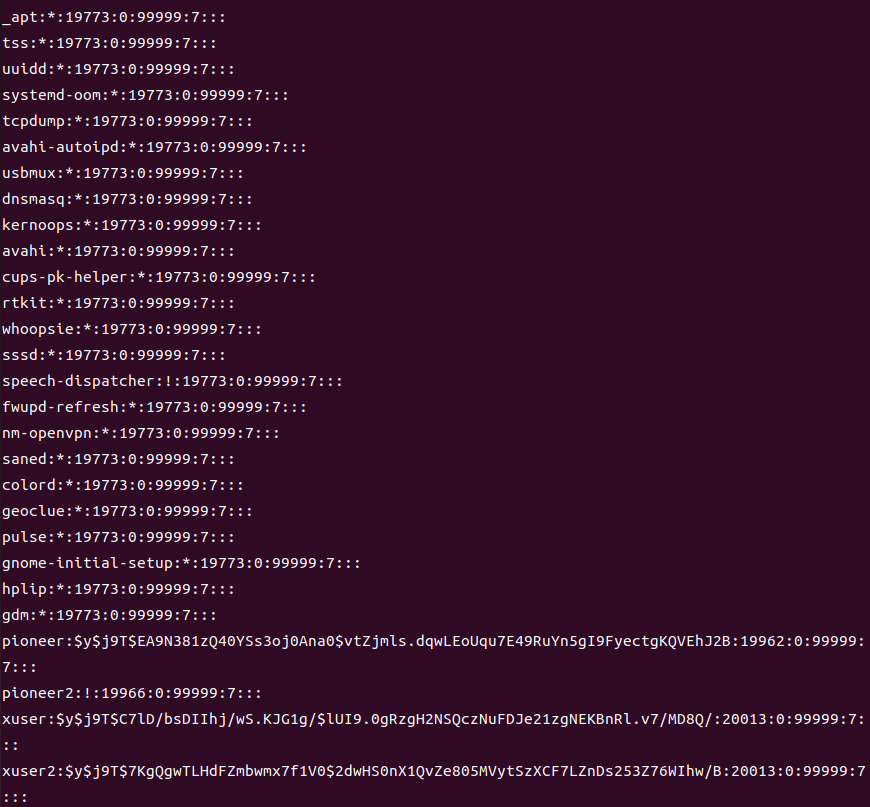




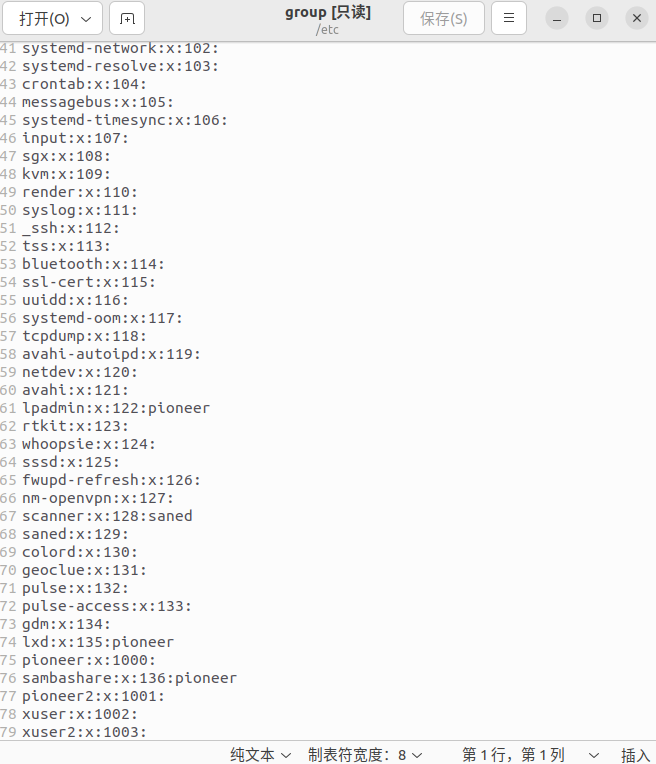
passwd:



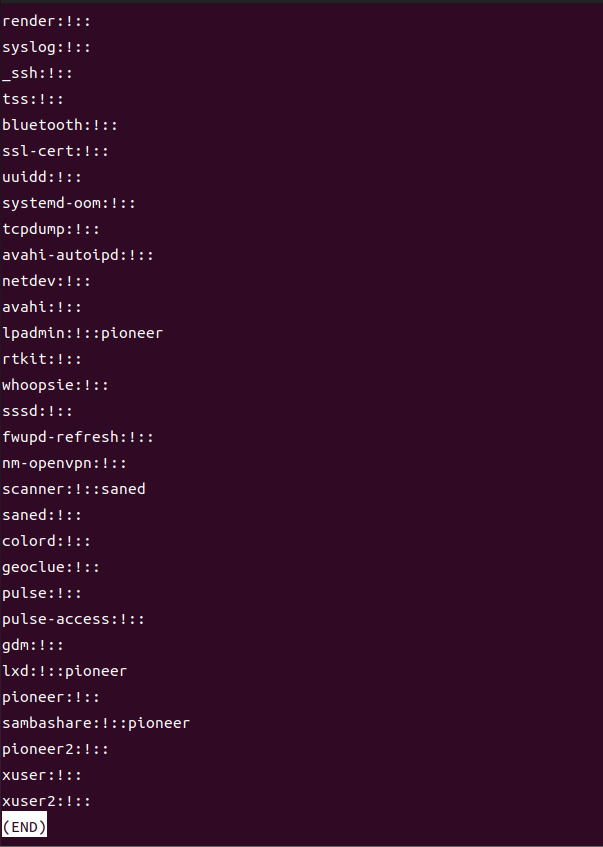
Shadow:



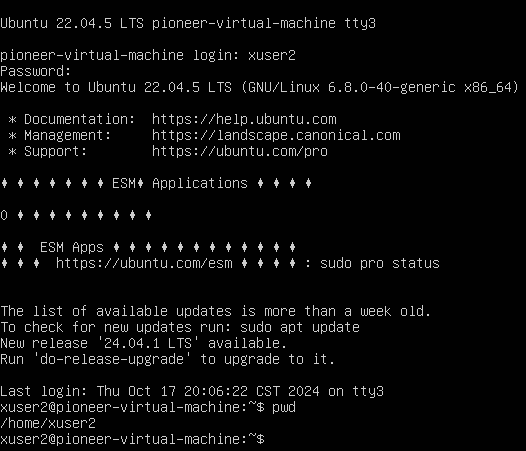
Group:



Gshadow:



切换用户：



**操作过程2：**

【操作要求 2】锁定 xuser2 用户帐号

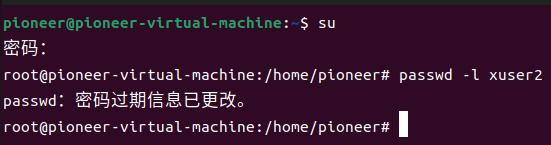
【操作步骤】

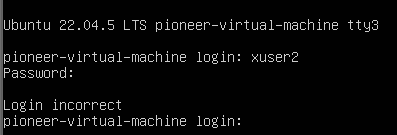
① 在「用户管理者」窗口选中 xuser2 用户帐号，单击工具栏上的「属性」按钮，打开 「用户属性」窗口。

② 选中「帐号信息」选项卡让「本地口令被锁」复选框被选中。单击「确定」按钮， 返回「用户管理者」窗口。 按下 CTRL+ALT+F2 组合键，再次切换到第 2 个虚拟终端，输入用户名 xuser2 和相 应的口令，发现 xuser2 用户无法登录 Linux 系统，说明 xuser2 用户账号的确已被锁 定。

③ 按下 ALT+F7 组合键再次返回 GNOME 桌面环境

**结果2：**





**操作过程3：**

【操作要求 3】删除 xuser2 用户

【操作步骤】

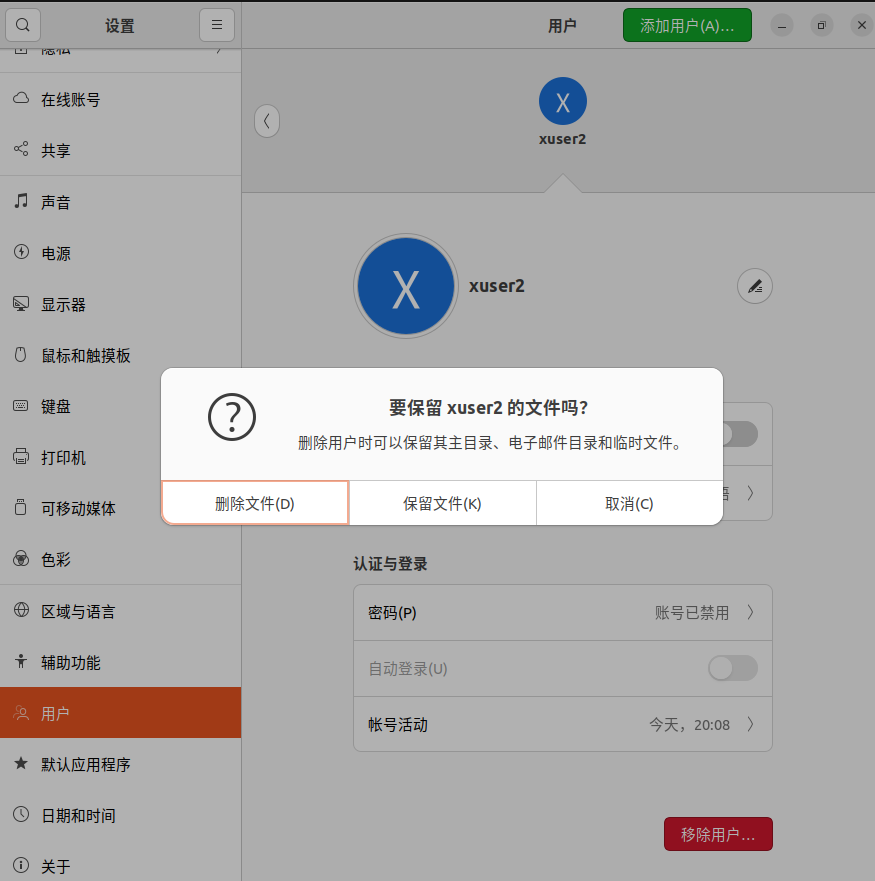
① 在「用户管理者」窗口，单击「编辑」菜单的「首选项」，弹出「首选项」对话框， 不选中「隐藏系统用户和组」复选框，最后单击「关闭」按钮。此时「用户」选项 卡中显示包括超级用户和系统用户在内的所有用户。

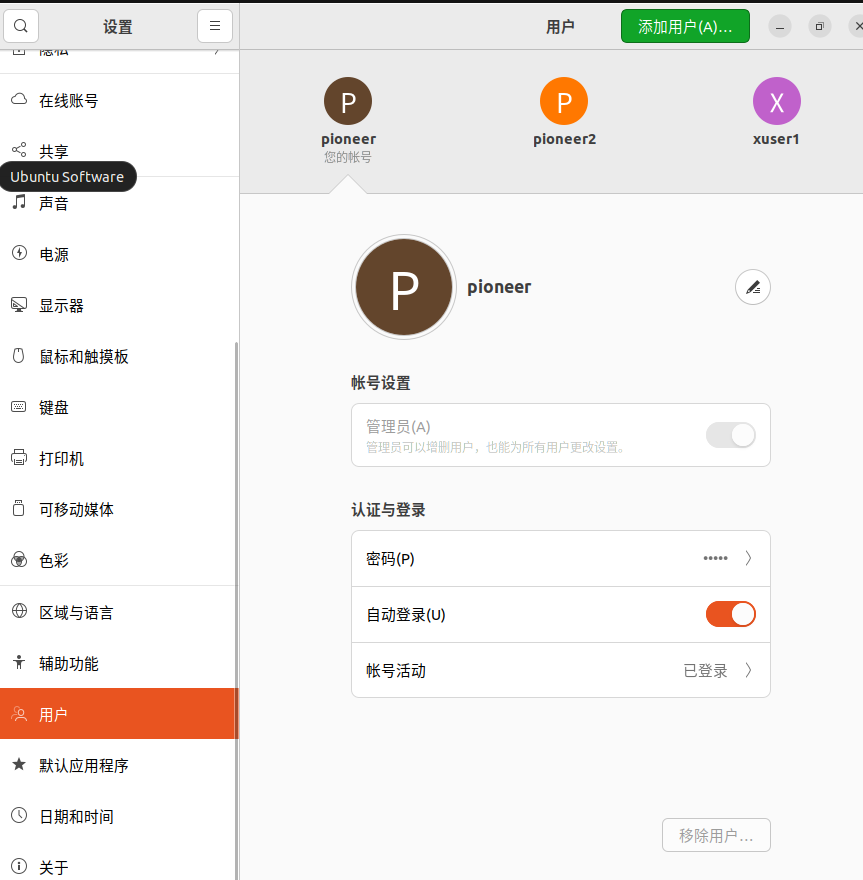
② 在「搜索过滤器」文本框中输入“x\*”并按下 Enter 键，则仅显示以 x 为首字母的用 户。

③ 选中 xuser2 用户，单击工具栏上的「删除」按钮，弹出对话框，单击「是」按钮， 返回「用户管理者」窗口，发现 xuser2 用户已被删除。

④ 在「搜索过滤器」文本框中输入“\*”并按下 Enter 键，则显示所有用户

**结果3：**





**操作过程4：**

【操作要求 4】新建两个组群，分别是 myusers 和 temp

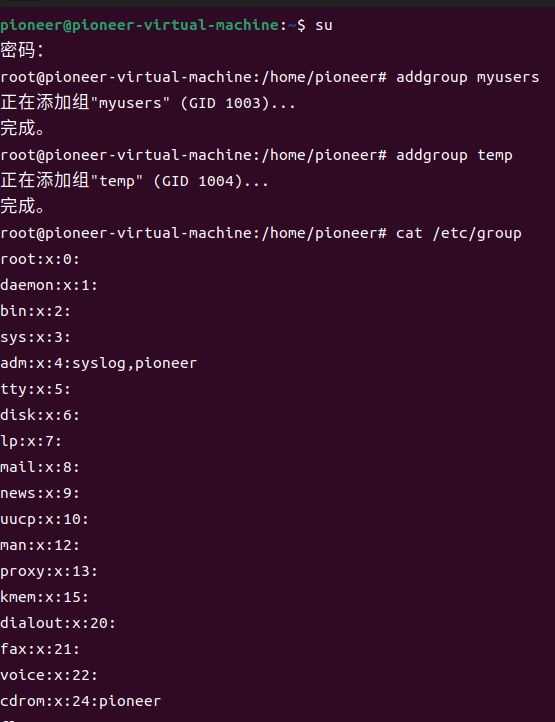
【操作步骤】

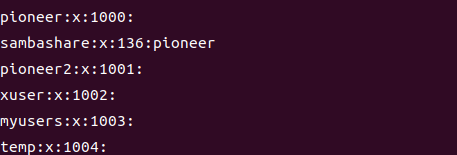
① 在「用户管理者」窗口选中「组群」选项卡，当前显示出所有组群。

② 单击工具栏上的「添加组群」按钮，出现「创建新组群」对话框。在「组群名」文 本框中输入“myusers”，单击「确定」按钮，返回「用户管理者」窗口。

③ 用相同的方法新建 temp 组群。

**结果4：**





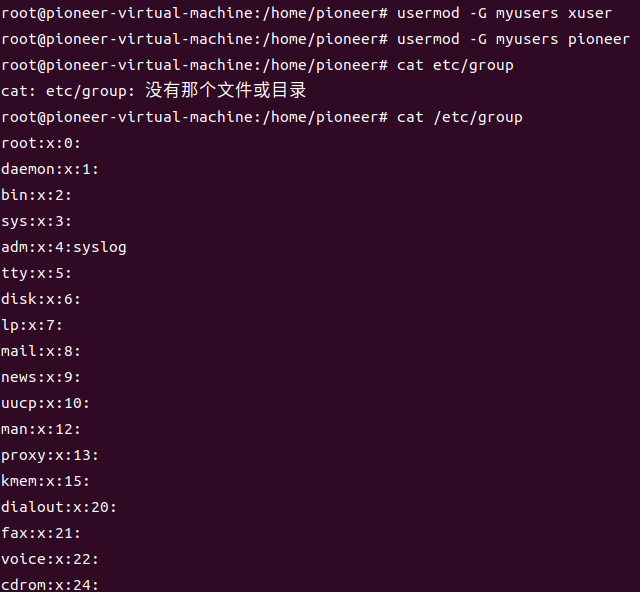
**操作过程5：**

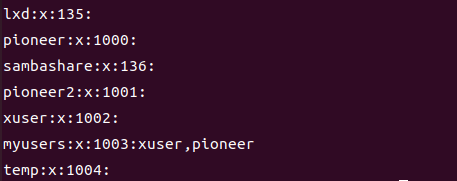
【操作要求 5】修改 myusers 组群属性，将 xuser1 和 helen 用户加入 myusers 组群 【操作步骤】

① 从「组群」选项卡中选择 myusers 组群，单击工具栏上的「属性」按钮，弹出「组 群属性」窗口。

② 选择「组群用户」选项卡，选中 helen 和 xuser1 前的复选框，设置 helen 用户和 xuser1 用户的 myusers 组群的成员。单击「确定」按钮，返回「用户管理者」窗口。

**结果5：**





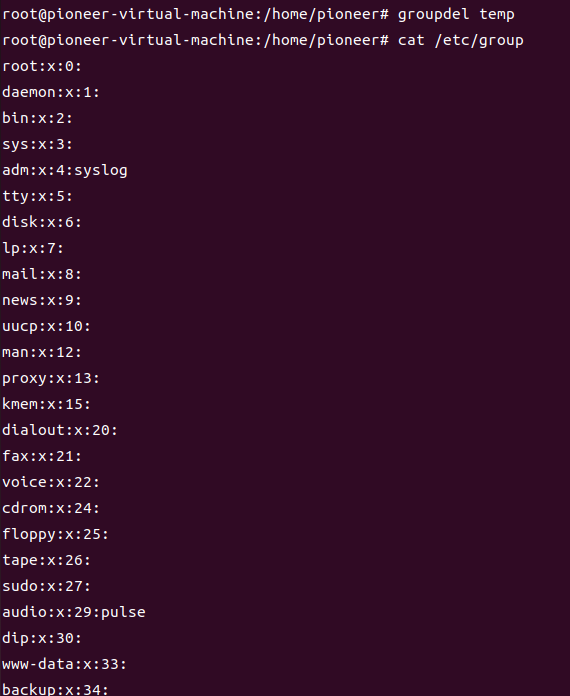
**操作过程6：**

【操作要求 6】删除 temp 组群

【操作步骤】

从「组群」选项卡中选择 temp 组群，单击工具栏上的「删除」按钮，出现确认对话框， 单击「是」按钮即可。

**结果6：**





**二、利用shell命令管理用户与组群**

**操作过程1：**

【操作要求 1】新建一名为 duser 的用户，其口令是“tdd63u2”，主要组群为 myusers 【操作步骤】

① 按下 CTRL+ALT+F3 组合键，切换到第 3 个虚拟终端，以超级用户身份登录。

② 输入命令“useradd -g myusers duser”，建立新用户 duser，其主要组群是 myusers。

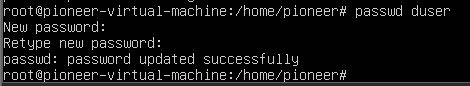
③ 为新用户设置口令，输入命令“passwd duser”，根据屏幕提示输入两次口令，最后屏 幕提示口令成功设置信息。

④ 输入命令“cat /etc/passwd”，查看/etc/passwd 文件的内容，发现文件的末尾增加 duser 用户的信息。

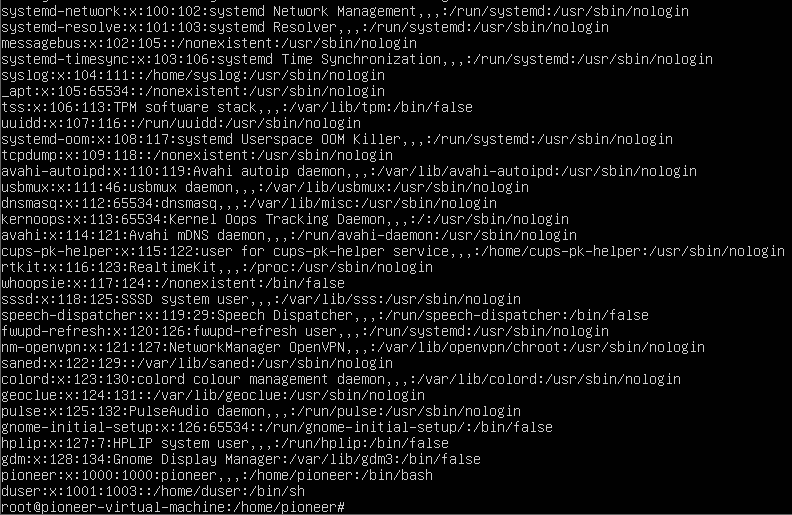
⑤ 输入命令“cat /etc/ group”，查看/etc/group 文件的内容，发现文件内容未增加。 ⑥ 按下 ALT+F4 组合键，切换到第 4 个虚拟终端，输入 duser 用户名和口令可登录 Linux 系统。

⑦ 输入“exit”命令，duser 用户退出登录。

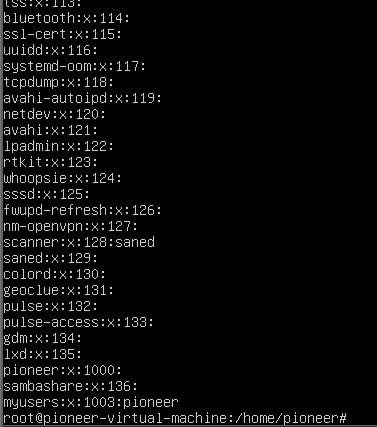
**结果1：**



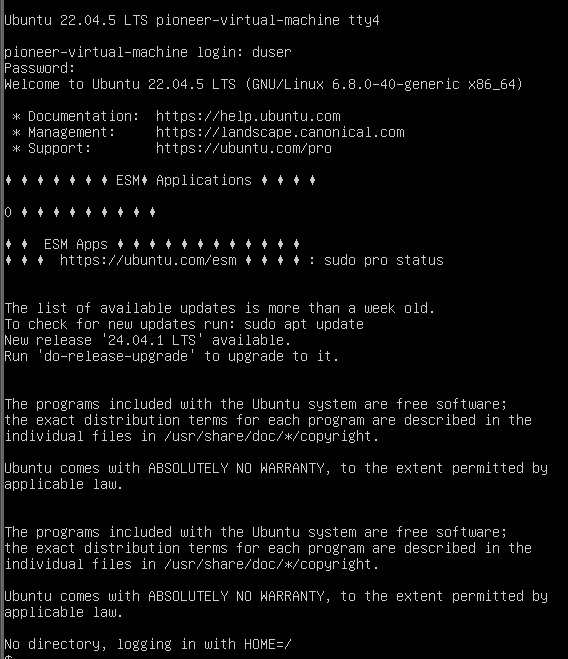
**passwd文件夹:**



**group文件夹:**



**登录duser:**



**操作过程2：**

【操作要求 2】将 duser 用户设置为不需口令就能登录

【操作步骤】

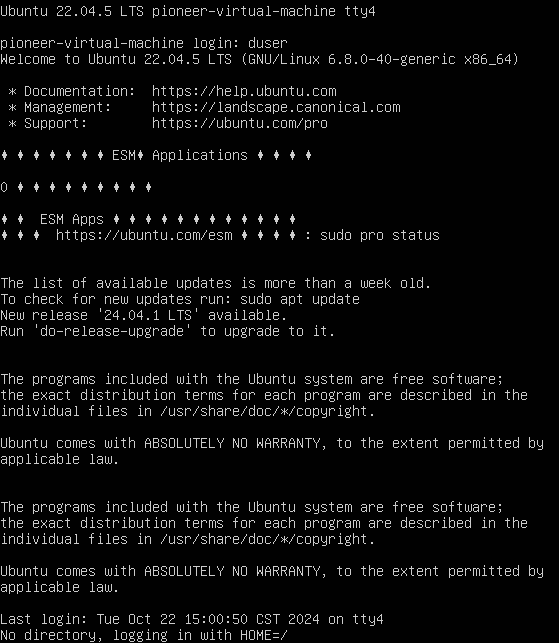
① 按下 ALT+F3 组合键，切换到正被超级用户使用的第 3 个虚拟终端。

② 输入命令“passwd -d duser”。

③ 按下ALT+F3组合键，再次切换到第3个虚拟终端，在“Login：”后输入用户名“duser”， 按下 Enter 键就直接出现 Shell 命令提示符，说明 duser 用户不需口令即可登录。

**结果2：**





**操作过程3：**

【操作要求 3】查看 duser 用户的相关信息

【操作步骤】 在第 3 个虚拟终端输入命令“idduser”，显示 duser 用户的用户 ID（UID）、主要组群 的名称和 ID（GID）。

**结果3：**



**操作过程4：**

【操作要求 4】从普通用户 duser 切换为超级用户

【操作步骤】

① 第 4 个虚拟终端当前的 Shell 命令提示符为“$”，表明当前用户是普通用户。

② 输入命令“ls /root”，屏幕上没有出现/root 目录中文件和子目录的信息，而是出现提 示信息，提示当前用户没有查看/root 目录的权限。

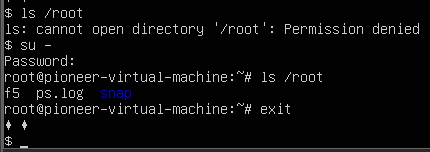
③ 输入命令“su -”或者是“su - root”，屏幕提示输入口令，此时输入超级用户的口令， 验证成功后 Shell 提示符从“$”变为“#”，说明已从普通用户转换为超级用户。

④ 再次输入命令“ls /root”，可查看/root 目录中文件和子目录的信息。

⑤ 输入“exit”命令，回到普通用户的工作状态。

⑥ 输入“exit”命令，duser 用户退出登录。

**结果4：**



**操作过程5：**

【操作要求 5】一次性删除 duser 用户及其工作目录

【操作步骤】

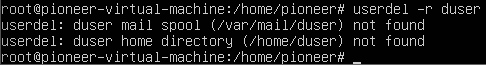
① 按下 ALT+F3 组合键，切换到正被超级用户使用的第 3 个虚拟终端。

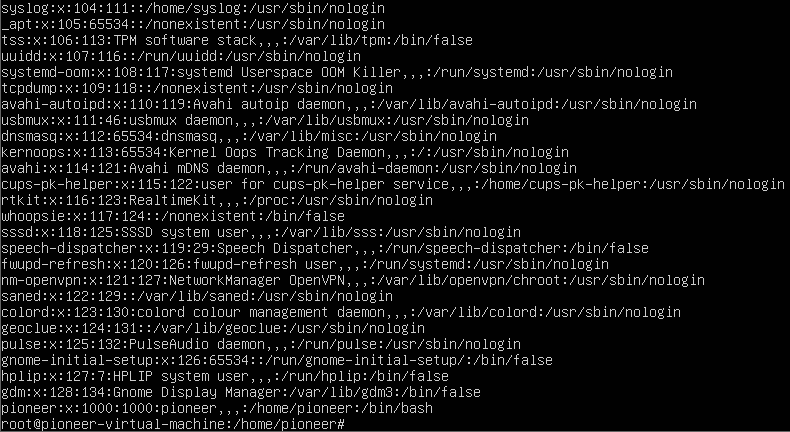
② 输入命令“userdel –r duser”，删除 duser 用户。

③ 输入命令“cat /etc/passwd”，查看/etc/passwd 文件的内容，发现 duser 的相关信息已 消失。

④ 输入命令“ls /home”，发现 duser 的主目录/home/duser 也不复存在。

**结果5：**







**操作过程6：**

【操作要求 6】新建组群 mygroup

【操作步骤】

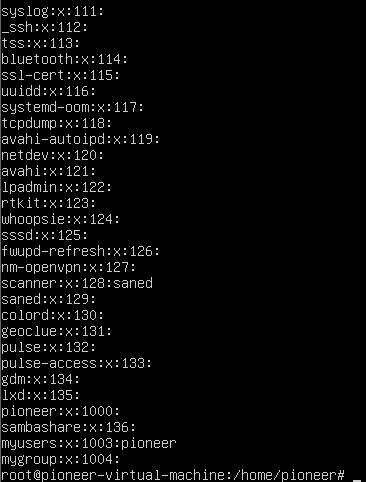
① 在超级用户的 Shell 提示符后输入命令“groupadd mygroup”，建立 mygroup 组群。

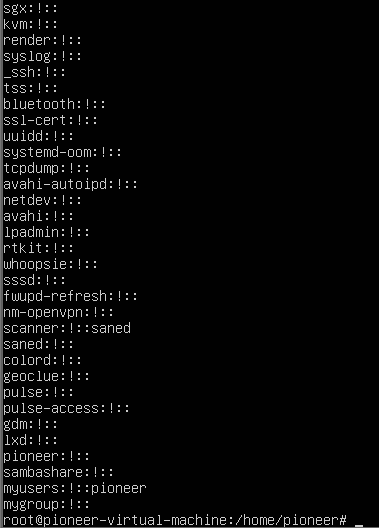
② 输入命令“cat /etc/group”，发现 group 文件的末尾出现 mygroup 组群的信息。

③ 输入命令“cat /etc/gshadow”，发现gshadow文件的末尾也出现mygroup组群的信息。

**结果6：**







**操作过程7：**

【操作要求 7】将 mygroup 组群改名为 newgroup

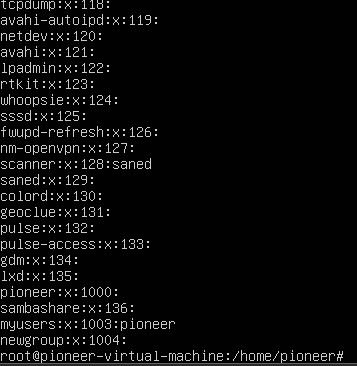
【操作步骤】

① 输入命令“groupmod –n newgroup mygroup”，其中–n 选项表示更改组群的名称。

② 输入命令“cat /etc/group”，查看组群信息，发现原来 mygroup 所在行的第一项变为 “newgroup”。

**结果7：**





**操作过程8：**

【操作要求 8】删除 newgroup 组群

【操作步骤】 超级用户输入“groupdel newgroup”命令，删除 newgroup 组群。

**结果8：**



**三、批量新建多个用户账号**

**操作过程1：**

【操作要求 1】为全班同学 20 位同学创建用户帐号，用户名为“s”+学号的组合，其中班级 名册中第一位同学的学号为 080101。所有同学都属于 class0801 组群。所有同学的初始口令 为 111111。

【操作步骤】

① 以超级用户身份登录，输入命令“groupadd -g 600 class0801”（假设值为 600 的 GID 未被 使用），新建全班同学的组群 class0801。

② 输入命令“vi student”，新建用户信息文件。

③ 按下“i”键，切换为 vi 的文本编辑模式，输入第一行信息： s080101:x:601:600::/home/s080101:/bin/bash。

④ 按下 ESC 键，切换到命令行模式，拖动鼠标，将整行选中，然后按下字母键 y 两次。 也就是将当前选中的行放到 vi 的暂存区域（类似于 Windows 的剪贴板）。

⑤ 然后按下字母键 p，就复制一行信息，重复此操作 19 次，然后部分修改每位同学用户 信息不同的地方。

⑥ 最后编辑完成的文件，为节约篇幅仅显示前 10 位同学信息。最后保存并退出 vi。

⑦ 输入命令“vi stu-passwd”，新建用户口令文件。

⑧ 按下“i”键，切换为 vi 的文本编辑模式，输入第一行信息：“s080101:111111”，即所有同 学的初始口令为 111111。按下 ESC 键，切换到命令行模式，拖动鼠标，将整行选中， 然后按下字母键 y 两次，复制行。

⑨ 连续按 p 键 19 次，就可复制出 19 行信息，然后修改成正确的用户名。

⑩ 输入命令“newusers < student”，批量新建用户帐号。

⑪ 输入命令“pwunconv”，暂时取消 shadow 加密。

⑫ 输入命令“chpasswd

⑬ 输入命令“pwconv”，进行 shadow 加密，完成批量创建用户帐号工作。

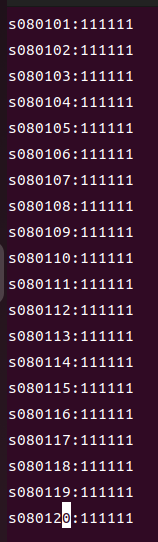
⑭ 输入命令“cat /etc/passwd”，查看/etc/passwd 文件将发现所有的用户帐号均已建立。

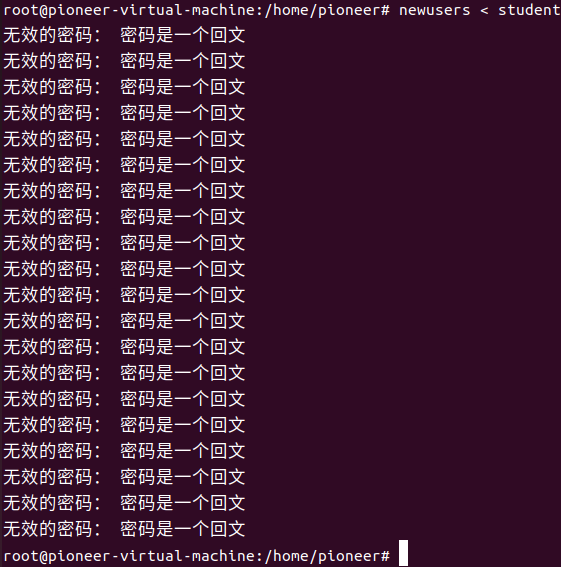
⑮ 可尝试以新建的用户名登录，并应该及时修改用户的口令。

**结果1：**

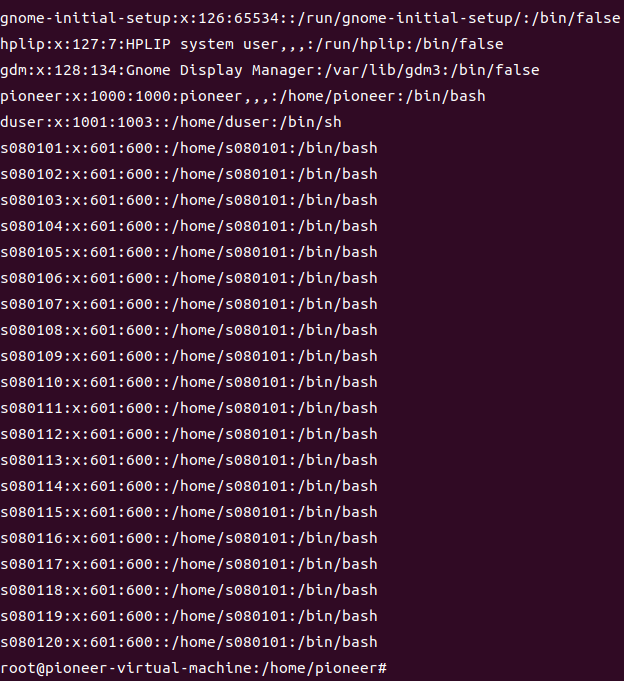




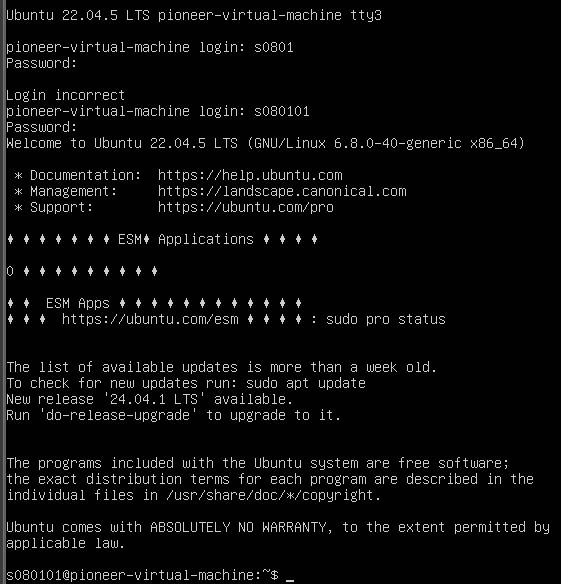








**登录成功s080101:**



**体会：**

本次实验主题是用户与组群管理，全称通过shell命令完成，由此我体验到了shell命令行语言功能的强大。我通过有条不紊地创建，管理用户和组群，体验到了linux系统管理多用户信息的方式。这极大提高了我掌握用户和多族的维护管理工作的能力，让我受益匪浅。

# 实验六题目：shell编程

**姓名：汪江豪 学号：22121630 实验日期：2024/10/22**

**实验环境：**

Ubuntu22.04操作系统

**实验目的：**

1. 掌握 vi 的三种工作方式，熟悉 vi 编辑程序的使用

2. 学习 Shell 程序设计方法。掌握编程要领

**实验内容：**

1. 学习使用 vi 编辑程序

2. 编写 Shell 程序

3. 将程序文件设置为可执行文件（用 chmod 命令）

4. 在命令行方式中运行 Shell 程序

**操作过程1：**

按本《实验指导》第三部分的内容。熟悉 vi 的三种工作方式。熟悉使用各种编辑功能

**结果1：**



**思考：**试一试 vi 的三种工作方式各用在何时？用什么命令进入插入方式？怎样退出插入方式？文件怎样存盘？注意存盘后的提示信息。

**答：**

1. Vi的三种工作方式：

（1） 插入方式：用于输入文本内容，使用退格键删除错误内容，按下ESC键，退出插入方式，进入转义命令方式。

（2） 转义命令方式：刚进入vi或退出插入方式，即为转义命令方式。此时键入的字符会转义为特殊功能，如：复制，删除，移动，替换，插入等。大多数转义命令由一个或两个字母组成，操作时没有提示符，且输入完字符不需要按回车键。

（3） 末行命令方式：在转义命令方式中，按下shift+：，会进入末行命令方式。最后一行的行首显示冒号作为命令的提示。输入命令后按下回车键执行命令。此时用户可进行文件的全局操作，如：全局查找、替换、保存、退出等。

2． 用什么命令进入插入方式：

|  |  |
| --- | --- |
| 命令 | 功能 |
| i text | 在光标前插入文本 |
| I text | 在当前行起始处插入文本 |
| a text | 在光标后输入文本 |
| A text | 在当前行末尾输入文本 |
| o text | 在当前行下产生新的一行并输入文本 |
| O text | 在当前行上产生新的一行并输入文本 |

3. 怎样退出插入方式：

在插入方式下按ESC键即可退出插入方式，进入转义命令方式。

4. 文件怎样存盘：

键入命令：w，如果需要将文件以不同名字保存，如name1,则键入命令: w name1。如果name1文件不存在，vi编辑器将存入文件并给出文件的大小。如果文件已经存在，vi会给出提示，但并不覆盖文件内容。

如果想刷新已存在的文件的内容，则可键入命令: w !name1，！的作用是放弃了vi防止覆盖文件的保护功能，强制覆盖文件。

**操作过程2：**

创建和执行 Shell 程序

用前面介绍的 Vi 或其他文本编辑器编写 Shell 程序，并将文件以文本文件方式保存在 相应的目录中。

用 chmod 将文件的权限设置为可执行模式，如若文件名为 shdemo.h，则命令如下：

$ chmod 755 shdemo.h(文件主可读、写、执行，同组人和其他人可读和执行)

在提示符后执行 Shell 程序：

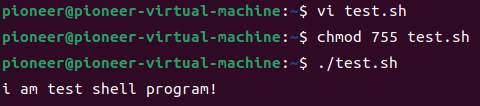
$ shdemo.h (直接键入程序文件名执行)

或 $ sh shdemo.h (执行 Shell 程序)

或 $ .shdemo.h (没有设置权限时可用点号引导)

**结果2：**





**操作过程3：**

用 vi 编写《实验指导》“第四部分 Shell 程序设计”中的例 1(假设文件名为 prog1.h)，练 习内部变量和位置参数的用法。

用 chmod 将文件的权限设置为可执行模式，并在提示符后键入命令行：

$./prog1.

或$sh prog1.

屏幕显示: Name not provided

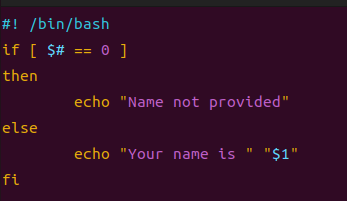
在提示符后键入命令行：

$./prog1.h Theodore #有一个参数

屏幕显示:

Your name is Theodore #引用$1 参数的效果

**结果3：**

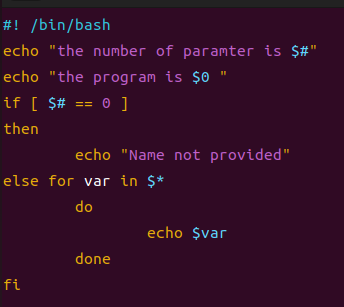


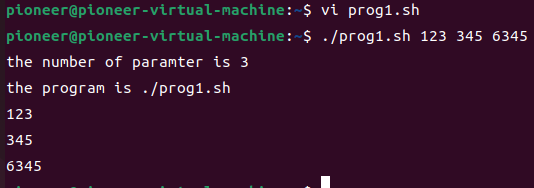


**操作过程4：**

进一步修改程序 prog1.h，要求显示参数个数、程序名字，并逐个显示参数。

**结果4：**

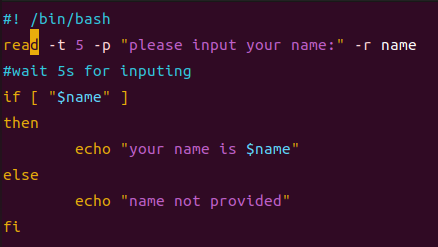


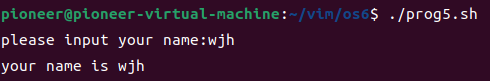


**操作过程5：**

修改例 1 程序（即上面的 prog1.h），用 read 命令接受键盘输入。若没有输入显示第一 种提示，否则第二种提示。

**结果5：**



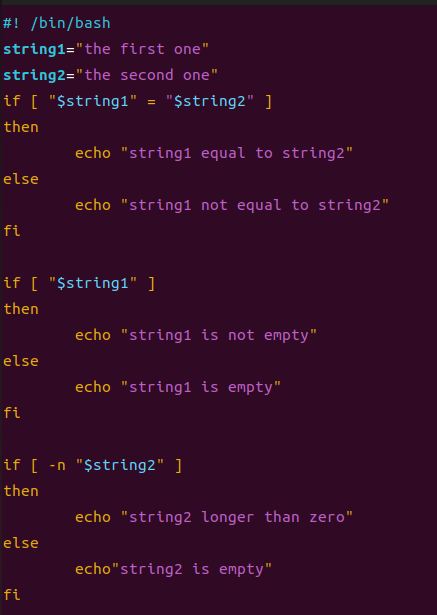


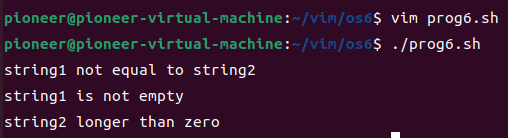
**操作过程6：**

用 vi 编写《实验指导》“第四部分 Shell 程序设计”中的例 2、例 3，练习字符串比较运 算符、数据比较运算符和文件运算符的用法，观察运行结果。

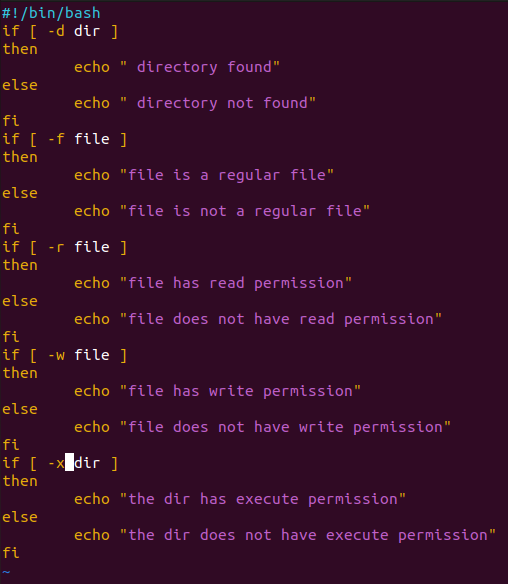
**结果6：**

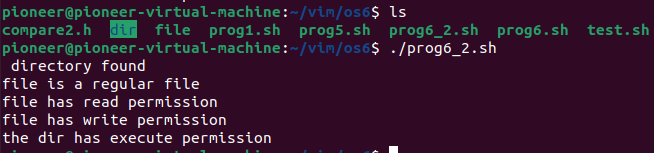
字符串比较运算练习：





数据和文件运算符练习：

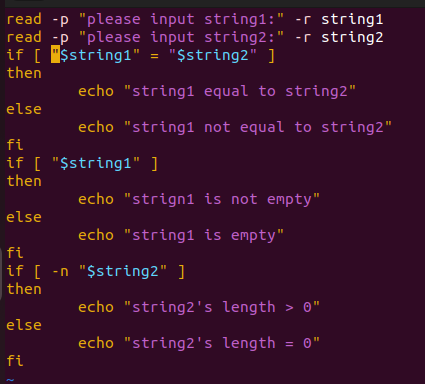


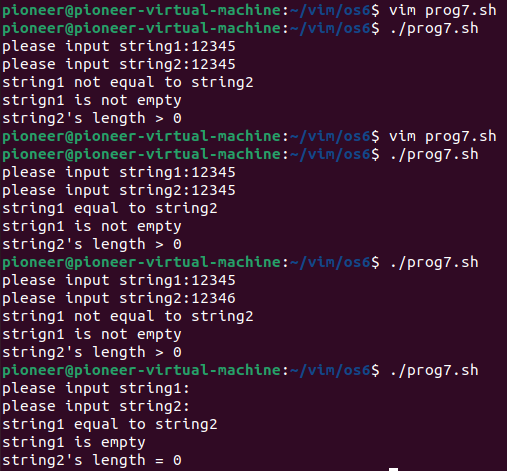


**操作过程7：**

修改例 2 程序，使在程序运行中能随机输入字符串，然后进行字符串比较。

**结果7：**

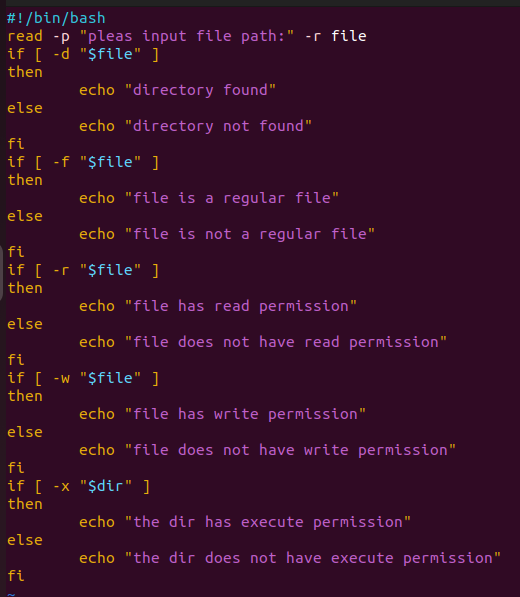


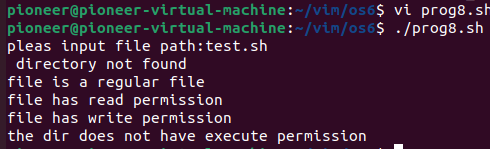


**操作过程8：**

修改例 3 程序，使在程序运行中能随机输入文件名，然后进行文件属性判断。

**结果8：**



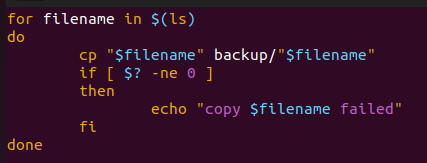


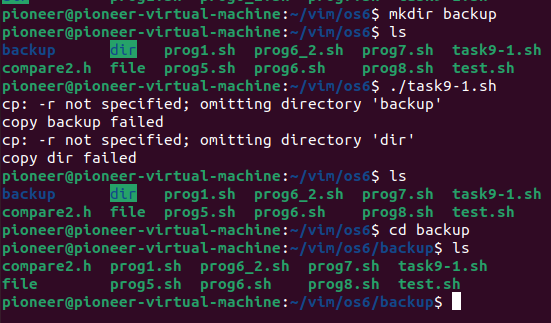
**操作过程9：**

用 vi 编写《实验指导》“第四部分 Shell 程序设计”中的例 4、例 5、例 6、例 7，掌握 控制语句的用法，观察运行结果。

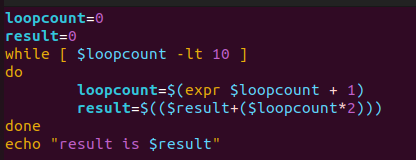
**结果9：**

例4：



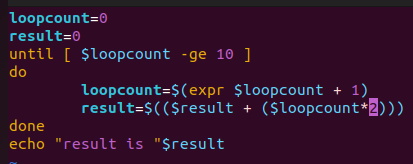


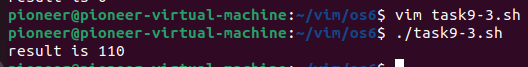
例5：



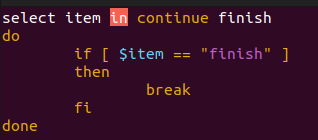


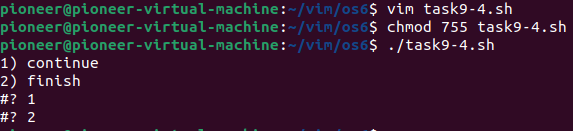
例6：





例7：

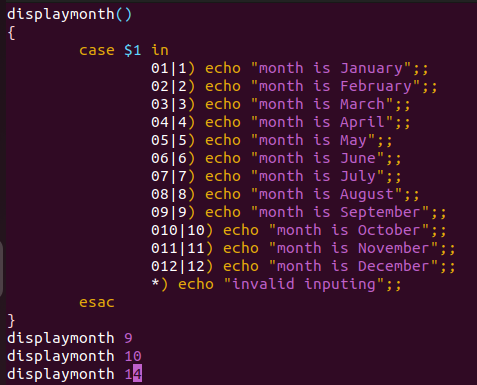


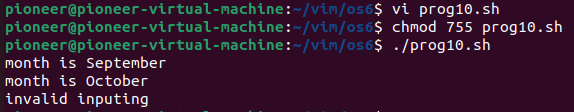


**操作过程10：**

用 vi 编写《实验指导》“第四部分 Shell 程序设计”中的例 8 及例 9 掌握条件语句的用法， 函数的用法，观察运行结果。

**结果10：**

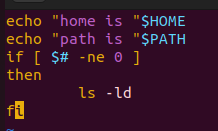


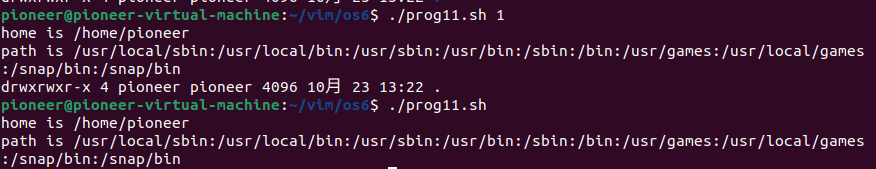


**操作过程11：**

编程，在屏幕上显示用户主目录名（HOME）、命令搜索路径（PATH），并显示由位 置参数指定的文件的类型和操作权限。

**结果11：**





**思考：**到此为止你对 Shell 有所认识了吧？怎么样？自己再编两个程序：

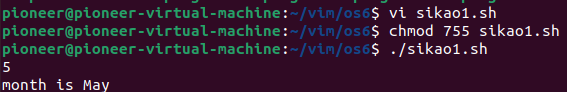
① 做个批处理程序，体会一下批处理概念。

② 做个菜单，显示系统环境参数。将此程序设置为人人可用。

**答：**① 做个批处理程序，体会一下批处理概念。

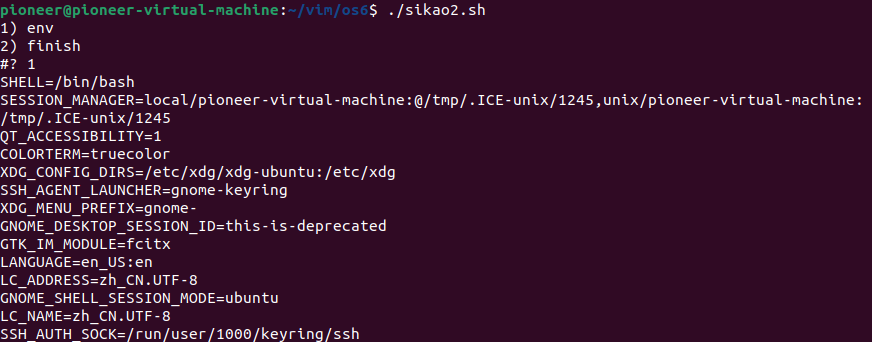
如下：输入数字，输出对应月份，即为一个简单的批处理程序。

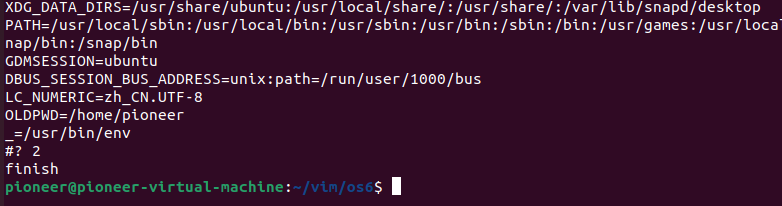




② 做个菜单，显示系统环境参数。将此程序设置为人人可用。







**讨论：**

1. **Linux 的 Shell 有什么特点？**

**答：**命令行接口：允许用户通过输入命令来与操作系统交互。

脚本编程能力：shell不仅是一个命令解释器，还是一种强大的脚本语言，可以用来编写程序，自动化任务等。

作业控制：如后台运行作业、挂起作业、恢复作业等。

管道和重定向：shell支持管道，允许将一个命令的输出作为另一个命令的输入。重定向则允许将命令的输出保存到文件中。

变量和参数：shell支持变量的使用，包括环境变量和和用户自定义变量，还支持参数传递给脚本和函数。

高级编程特性：shell脚本通常比传统编程语言简单，但也提供了循环，条件判断，函数等高级编程特性。

文件处理能力：shell具有强大的文件处理能力，包括文件创建、编辑、复制、移动和删除等。

权限和安全性：shell可以设置文件属性，执行安全检查等。

1. **怎样进行 Shell 编程？如何运行？有什么条件？**

**答：**创建shell文件，可以使用vim编写，如：vim test.sh。

编写脚本；

保存并关闭文件，在vim中切换到命令模式，输入wq保存并退出。

赋予执行权限，如：chmod 755 test.sh,为test.sh文件分配了：所有者：读写执行，组：读和执行，其他用户：读和执行。

运行脚本，例如test.sh在当前目录下，输入：./test.sh

1. **vi 编辑程序有几种工作方式？查找有关的详细资料，熟练掌握屏幕编辑方式、转移 命令方式以及末行命令的操作。学习搜索、替换字符、字和行，行的复制、移动， 以及在 vi 中执行 Shell 命令的方式。**

**答：**vi有三种工作方式。

1. 命令行模式

这是vi的默认模式，用于控制光标的移动，执行字符，字或行的删除，移动复制某区段，以及进入插入模式或末行模式。

1. 文本输入模式

按下i进入插入模式，在该模式下，用户可以输入文本。

1. 末行模式

在命令行模式下，按“：”即进入末行模式。该模式用于保存文件w，退出vi:q，强制保存w!,强制退出q！，搜索和替换：s/old/new(替换当前行的第一个old为new)、%s/old/newg(替换整个文件中的所有old为new)

1. **编写一个具有以下功能的 Shell 程序。**

**(1) 把当前目录下的文件目录信息输出到文件 filedir.txt 中；**

**(2) 在当前目录下建立一个子目录，目录名为 testdir2；**

**(3) 把当前目录下的所有扩展名为 c 的文件以原文件名复制到子目录 testdir2 中；**

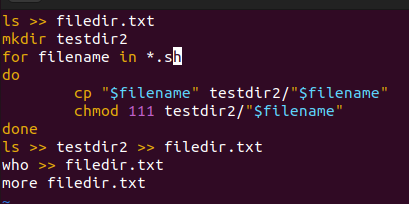
**(4) 把子目录中的所有文件的存取权限改为不可读。（提示：用 for 循环控制语句实现，循 环的控制列表用’ls’产生。）**

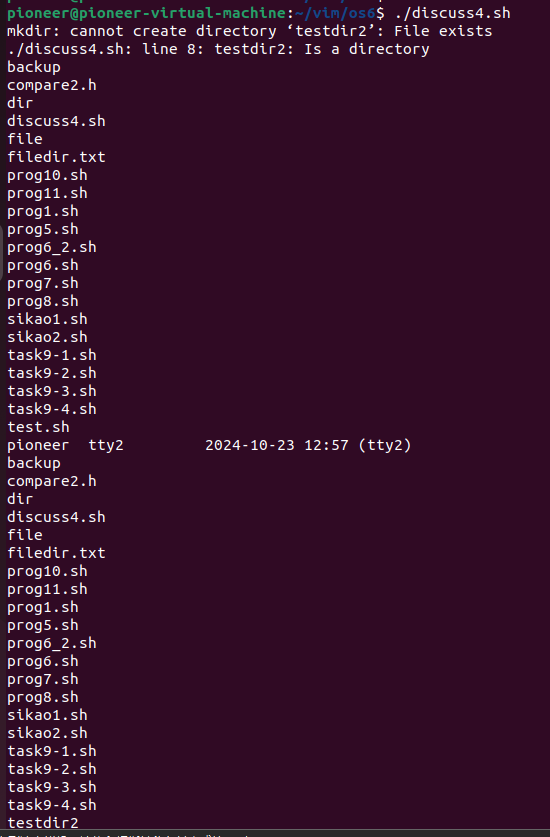
**(5) 在把子目录 testdir2 中所有文件的目录信息追加到文件 filedir.txt 中；**

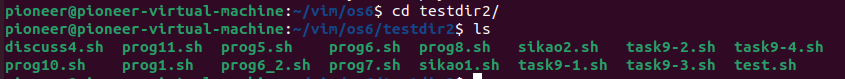
**(6) 把你的用户信息追加到文件 filedir.txt 中；**

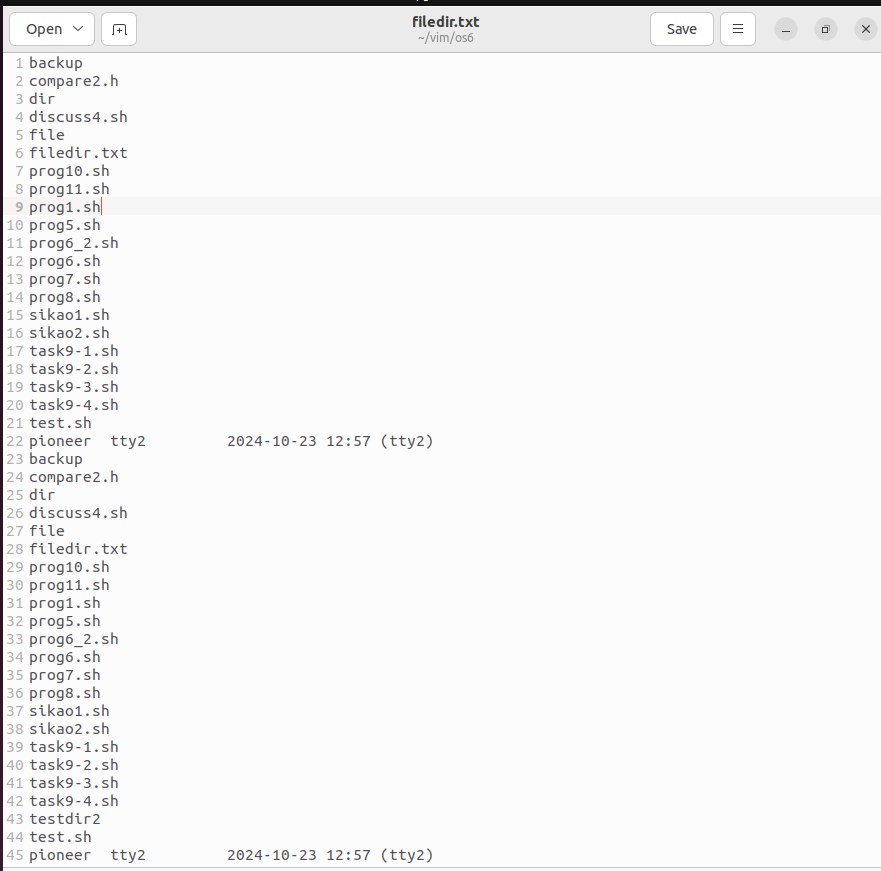
**(7) 分屏显示文件 filedir.txt**

**答：**程序如下：









**实验体会：**

本次实验主题为shell编程，基于此我编写了大量的shell程序，加深了我对于shell这一强大语言的理解。我深刻体会到了shell脚本在自动化处理和系统管理中的强大作用。实验过程中，我通过编写脚本，执行一系列命令，不仅锻炼了自己的编程能力，也对操作系统的运行机制有了更深的了解。从构思脚本逻辑，到逐步调试直至脚本完美运行，我感受到了shell编程带来的成就感。同时也意识到了shell编程在简化复杂任务，提高工作效率方面的巨大潜力。它及其适合自动化，只需要将原来手动输入多次的命令，都复制到一个文件里就行了。日后有机会我会积极尝试将shell语言融入其他编程语言中，尝试新鲜编程方式。