

实验题目：Linux 操作系统基本命令

姓名：黄晨航 学号：19120178 实验日期：2021.9.9

实验环境：Linux 环境

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。

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。

操作过程及结果：

登录进入 Linux 命令操作系统界面。

执行下列各类命令，熟悉 Linux 用户命令接口。

(1) 查看信息

- ① 执行 pwd 查看当前目录。
- ② 用 whoami 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。

```
hch@hch-virtual-machine:~/桌面$ pwd
/home/hch/桌面
hch@hch-virtual-machine:~/桌面$ whoami
hch
hch@hch-virtual-machine:~/桌面$ who
hch      :0                2021-09-16 22:23 (:0)
hch@hch-virtual-machine:~/桌面$ vmstat
procs -----memory-----swap-- -----io----- -system-- -----cpu-----
 r  b 交换 空闲 缓冲 缓存    si  so    bi    bo    in    cs us sy id wa st
 2   0      0 320208 33900 769708      0    0  3763   269  500 1128  8 15 76  1  0
```

结果：

执行 pwd 可以看到当前目录为/home/hch/桌面；

whoami 可以查看到当前用户是 hch；

who 可以看系统中只有一个用户 hch；

vmstat 可以看到当前系统的状态；

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：

```
hch@hch-virtual-machine:~/桌面$ id
用户id=1000(hch) 组id=1000(hch) 组=1000(hch),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),132(lxd),133(sambashare)
```

由 id 命令可以看到，用户名 hch，用户标识 1000，组名 hch，组表示 1000；当前处在 /home/hch/桌面目录下；hch 和 root 用户一块共享系统。

(2) 文件操作

- ① 执行 `cat>mytext.txt` 通过键盘输入一些信息，用 `ctrl+c` 结束，建立文件 `mytext.txt`。
- ② 执行 `cat mytext.txt` 显示文件内容
- ③ 执行 `ln mytext.txt mytext2.dat`
`cat mytext2.dat`
- ④ 执行 `ls -l mytext?.*`

```
hch@hch-virtual-machine:~/桌面$ cat>mytext.txt
test test hahaha
^C
hch@hch-virtual-machine:~/桌面$ cat mytext.txt
test test hahaha
hch@hch-virtual-machine:~/桌面$ ln mytext.txt mytext2.dat
hch@hch-virtual-machine:~/桌面$ cat mytext2.dat
test test hahaha
hch@hch-virtual-machine:~/桌面$ ls -l mytext?.*
mytext2.dat
```

结果：

通过 `cat` 命令新建了一个文件，并输入文件的内容为“test test hahaha”；然后用 `cat` 命令显示文件的内容之后建立连接用 `ln` 命令建立连接可以看到新建的文件 `mytext2.dat` 的内容与原来的文件 `mytext.txt` 的内容是一致的。

思考：文件链接是什么意思？有什么作用？

答：该命令在文件之间创建链接。这种操作实际上是给系统中已有的某个文件指定另外一个可用于访问它的名称。对于这个新的文件名，我们可以为之指定不同的访问权限，以控制对信息的共享和安全性的问题。如果链接指向目录，用户就可以利用该链接直接进入被链接的目录而不用打一大堆的路径名。而且，即使我们删除这个链接，也不会破坏原来的目录。

(3) 目录操作

- ① 执行 `ls -l` 看看当前目录的内容，请特别注意文件类型、文件的存取控制权限、i 节点号、文件 属主、文件属组、文件大小、建立日期等信息。

- ② 执行 `cd /lib`
`ls -l|more`

看看 `/lib` 目录的内容，这里都是系统函数。再看看 `/etc`，这里都是系统配置用的数据文件；`/bin` 中是可执行程序；`/home` 下包括了每个用户主目录。

结果：

```
hch@hch-virtual-machine:~/桌面$ ls -l
总用量 8
-rw-rw-r-- 2 hch hch 17 9月 23 19:56 mytext2.dat
-rw-rw-r-- 2 hch hch 17 9月 23 19:56 mytext.txt
```

执行 `ls -l` 可以看到文件的信息；该文件为普通文件，文件主可以读写，同组人也可以读写，其他人只能读；

查看 `/lib` 的内容：

```
hch@hch-virtual-machine:~/桌面$ cd /lib
hch@hch-virtual-machine:/lib$ ls -l|more
总用量 644
drwxr-xr-x 2 root root 4096 8月 19 18:32 accountsservice
drwxr-xr-x 2 root root 4096 8月 19 18:33 apg
drwxr-xr-x 2 root root 4096 8月 19 18:32 apparmor
drwxr-xr-x 5 root root 4096 8月 19 18:30 apt
drwxr-xr-x 3 root root 4096 8月 19 18:33 aspell
drwxr-xr-x 2 root root 4096 4月 22 2020 binfmt.d
drwxr-xr-x 2 root root 4096 8月 19 18:33 bluetooth
drwxr-xr-x 2 root root 4096 8月 19 18:33 bolt
drwxr-xr-x 2 root root 4096 8月 19 18:40 brltty
-rwxr-xr-x 1 root root 684 8月 3 2020 cnf-update-db
-rwxr-xr-x 1 root root 3565 8月 3 2020 command-not-found
drwxr-xr-x 2 root root 4096 8月 19 18:29 console-setup
lrwxrwxrwx 1 root root 21 9月 7 17:27 cpp -> /etc/alternatives/cpp
drwxr-xr-x 3 root root 4096 9月 23 20:09 crda
drwxr-xr-x 10 root root 4096 8月 19 18:33 cups
drwxr-xr-x 2 root root 4096 8月 19 18:30 dbus-1.0
drwxr-xr-x 5 root root 4096 8月 19 18:40 debug
drwxr-xr-x 3 root root 4096 8月 19 18:29 dpkg
drwxr-xr-x 2 root root 4096 8月 19 18:29 eject
drwxr-xr-x 3 root root 4096 8月 19 18:33 emacs-common
drwxr-xr-x 2 root root 4096 8月 19 18:39 environment.d
drwxr-xr-x 7 root root 4096 8月 19 18:34 evolution-data-server
drwxr-xr-x 2 root root 4096 8月 19 18:29 file
drwxr-xr-x 7 root root 4096 9月 23 20:09 firefox
drwxr-xr-x 5 root root 4096 8月 19 18:34 firefox-addons
drwxr-xr-x 90 root root 20480 9月 23 20:09 firmware
```

查看/etc 的内容:

```
hch@hch-virtual-machine:/lib$ cd /etc
hch@hch-virtual-machine:/etc$ ls -l|more
总用量 1100
drwxr-xr-x 3 root root 4096 8月 19 18:40 acpi
-rw-r--r-- 1 root root 3028 8月 19 18:29 adduser.conf
drwxr-xr-x 3 root root 4096 8月 19 18:32 alsa
drwxr-xr-x 2 root root 4096 9月 7 17:37 alternatives
-rw-r--r-- 1 root root 401 7月 17 2019 anacrontab
-rw-r--r-- 1 root root 433 10月 2 2017 apg.conf
drwxr-xr-x 5 root root 4096 8月 19 18:33 apm
drwxr-xr-x 3 root root 4096 8月 19 18:40 apparmor
drwxr-xr-x 6 root root 4096 9月 23 20:09 apparmor.d
drwxr-xr-x 4 root root 4096 9月 23 20:11 appport
-rw-r--r-- 1 root root 769 1月 19 2020 appstream.conf
drwxr-xr-x 7 root root 4096 9月 7 17:43 apt
drwxr-xr-x 3 root root 4096 8月 19 18:41 avahi
-rw-r--r-- 1 root root 2319 2月 25 2020 bash.bashrc
-rw-r--r-- 1 root root 45 1月 26 2020 bash_completion
drwxr-xr-x 2 root root 4096 9月 23 20:11 bash_completion.d
-rw-r--r-- 1 root root 367 4月 15 2020 bindresvport.blacklist
drwxr-xr-x 2 root root 4096 4月 22 2020 binfmt.d
drwxr-xr-x 2 root root 4096 8月 19 18:40 bluetooth
-rw-r----- 1 root root 33 8月 19 18:41 brlapi.key
drwxr-xr-x 7 root root 4096 8月 19 18:39 brltty
-rw-r--r-- 1 root root 26916 3月 4 2020 brltty.conf
drwxr-xr-x 3 root root 4096 8月 19 18:29 ca-certificates
-rw-r--r-- 1 root root 6569 8月 19 18:31 ca-certificates.conf
-rw-r--r-- 1 root root 5713 8月 19 18:30 ca-certificates.conf.dpkg-old
```

查看/bin 的内容:

```
hch@hch-virtual-machine:/etc$ cd /bin
hch@hch-virtual-machine:/bin$ ls -l|more
总用量 166280
-rwxr-xr-x 1 root root      59736 9月   5  2019 [
-rwxr-xr-x 1 root root      31248 5月   20  2020 aa-enabled
-rwxr-xr-x 1 root root      35344 5月   20  2020 aa-exec
-rwxr-xr-x 1 root root      22912 4月   14  21:38 aconnect
-rwxr-xr-x 1 root root      19016 11月  28  2019 acpi_listen
-rwxr-xr-x 1 root root       7415 5月    1  02:13 add-apt-repository
-rwxr-xr-x 1 root root      30952 7月   21  2020 addpart
-rwxr-xr-x 1 root root      47552 4月   14  21:38 alsabat
-rwxr-xr-x 1 root root      85296 4月   14  21:38 alsaloop
-rwxr-xr-x 1 root root      72432 4月   14  21:38 alsamixer
-rwxr-xr-x 1 root root      14720 4月   14  21:38 alsatplg
-rwxr-xr-x 1 root root      31528 4月   14  21:38 alsaucm
-rwxr-xr-x 1 root root      31112 4月   14  21:38 amidi
-rwxr-xr-x 1 root root      63952 4月   14  21:38 amixer
-rwxr-xr-x 1 root root       2668 3月   22  2020 amuFormat.sh
-rwxr-xr-x 1 root root        274 10月   2  2017 apg
-rwxr-xr-x 1 root root      26696 10月   2  2017 apgbfm
-rwxr-xr-x 1 root root      84408 4月   14  21:38 aplay
-rwxr-xr-x 1 root root      27016 4月   14  21:38 aplaymidi
-rwxr-xr-x 1 root root       2558 12月   5  2019 apport-bug
-rwxr-xr-x 1 root root      13367 8月   26  22:30 apport-cli
lrwxrwxrwx 1 root root        10 8月   26  22:30 apport-collect -> apport-bug
```

查看/home 的内容:

```
hch@hch-virtual-machine:/bin$ cd /home
hch@hch-virtual-machine:/home$ ls -l|more
总用量 4
drwxr-xr-x 15 hch hch 4096 9月   7  17:52 hch
```

思考: Linux 文件类型有几种? 文件的存取控制模式如何描述?

答:文件类型有普通文件(~)、目录文件(d)、块设备特别文件(b)、字符设备特别文件(c)、命名管道文件(p)等。

“存取控制模式”指对不同用户分配不同的操作权。Linux 文件系统将用户分成三类,即文件主、同组人、其他人。每种人可以行使的操作有三种,即读(r)、写(w)、执行(x)。

(4) 修改文件属性

① 执行 `chmod 751 mytext.txt`
`ls -l mytext.txt`

② 执行 `chown stud090 mytext.txt`

结果:

```
hch@hch-virtual-machine:~/桌面$ chmod 751 mytext.txt
hch@hch-virtual-machine:~/桌面$ ls -l mytext.txt
-rwxr-x--x 2 hch hch 17 9月 23 19:56 mytext.txt
```

通过 `chmod 751 mytext.txt` 命令设置 `mytext.txt` 文件的访问模式为,文件主可读、可写、可执行,同组人可读、可执行,其他人可执行;

```
hch@hch-virtual-machine:~/桌面$ chown stud090 mytext.txt
chown: 无效的用户: "stud090"
```

通过 `chown` 更改文件所有者为 `stud090`.

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：不能再修改文件，因为文件的所属用户不再是当前登录的用户

(5) 进程管理

① 执行 `ps -ef`

② 执行 `wait` 和 `sleep` 命令

结果：

```
hch@hch-virtual-machine:~/桌面$ ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1        0  0  19:38 ?        00:00:04 /sbin/init splash
root           2        0  0  19:38 ?        00:00:00 [kthreadd]
root           3        2  0  19:38 ?        00:00:00 [rcu_gp]
root           4        2  0  19:38 ?        00:00:00 [rcu_par_gp]
root           6        2  0  19:38 ?        00:00:00 [kworker/0:0H-events_highpri]
root           9        2  0  19:38 ?        00:00:00 [mm_percpu_wq]
root          10        2  0  19:38 ?        00:00:00 [rcu_tasks_rude_]
root          11        2  0  19:38 ?        00:00:00 [rcu_tasks_trace]
root          12        2  0  19:38 ?        00:00:01 [ksoftirqd/0]
root          13        2  0  19:38 ?        00:00:00 [rcu_sched]
root          14        2  0  19:38 ?        00:00:00 [migration/0]
root          15        2  0  19:38 ?        00:00:00 [idle_inject/0]
root          16        2  0  19:38 ?        00:00:00 [cpuhp/0]
root          17        2  0  19:38 ?        00:00:00 [cpuhp/1]
root          18        2  0  19:38 ?        00:00:00 [idle_inject/1]
root          19        2  0  19:38 ?        00:00:00 [migration/1]
root          20        2  0  19:38 ?        00:00:00 [ksoftirqd/1]
root          22        2  0  19:38 ?        00:00:00 [kworker/1:0H-events_highpri]
root          23        2  0  19:38 ?        00:00:00 [kdevtmpfs]
root          24        2  0  19:38 ?        00:00:00 [netns]
root          25        2  0  19:38 ?        00:00:00 [inet_frag_wq]
root          26        2  0  19:38 ?        00:00:00 [kauditd]
root          28        2  0  19:38 ?        00:00:00 [khungtaskd]
root          29        2  0  19:38 ?        00:00:00 [oom_reaper]
root          30        2  0  19:38 ?        00:00:00 [writeback]
root          31        2  0  19:38 ?        00:00:00 [kcompactd0]
root          32        2  0  19:38 ?        00:00:00 [ksmd]
```

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：Linux 系统为了管理进程，用 `task_struct` 数据结构表示每个进程，任务向量是一个指针数组，里面的指针指向系统中的每一个 `task_struct` 数据结构。进程的家族关系是进程家族树体现的，可以通过这种继承体系从系统的任何一个进程发现查找到任意指定的其他进程。但大多数时候，只需要通过简单的重复方式就可以遍历系统中的所有进程。

体会：

这是第一次操作系统的实验，全新的操作系统让我一时间难以适应，只能跟着实验指导书一步一步操作熟悉，但是对这一大堆命令的使用仍旧不是很熟悉，希望在以后的实验中能更进一步。

实验题目：用户界面与 Shell 命令

姓名：黄晨航 学号：19120178 实验日期：2021.9.23

实验环境： Linux 环境

实验目的：

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

实验内容：

图形化用户界面(GNOME 和 KDE)下用户操作非常简单而直观，但是到目前为止图形化用户界面还不能完成所有的操作任务。

字符界面占用资源少，启动迅速，对于有经验的管理员而言，字符界面下使用 Shell 命令更为直接高效。

Shell 命令是 Linux 操作系统的灵魂，灵活运用 Shell 命令可完成操作系统所有的工作。并且类 UNIX 的操作系统在 Shell 命令方面具有高度相似性。熟悉掌握 Shell 命令，不仅有助于掌握 RHEL Server 5，而且几乎有助于掌握各发行版本的 Linux，甚至 UNIX。RHEL Server 5 中不仅可在字符界面下使用 Shell 命令，还可以借助于桌面环境下的终端工具使用 Shell 命令。桌面的终端工具中使用 Shell 命令时可显示中文，而字符界面下显示英文。

操作过程及结果：

（一） 图形化用户界面下的 Shell 命令操作

（1） 显示系统时间，并将系统时间修改为 2011 年 9 月 17 日零点。

- 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“date 091700002011”，屏幕显示新修改的系统时间。在桌面环境的终

```
root@hch-virtual-machine:/home/hch/桌面# date
2021年 09月 23日 星期四 21:44:58 CST
root@hch-virtual-machine:/home/hch/桌面# date 091700002011
2011年 09月 17日 星期六 00:00:00 CST
```

（2） 切换为普通用户，查看 2011 年 9 月 17 日是星期几。

- 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 - hch”命令切换为普通用户 hch。
- ② 输入命令“cal 2011”，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。

```
root@hch-virtual-machine:/home/hch/桌面# su - hch
hch@hch-virtual-machine:~$ cal 2011
                2011
    一月          二月          三月
日 一 二 三 四 五 六 日 一 二 三 四 五 六 日 一 二 三 四 五 六
    1                1 2 3 4 5          1 2 3 4 5
    2 3 4 5 6 7 8    6 7 8 9 10 11 12    6 7 8 9 10 11 12
    9 10 11 12 13 14 15 13 14 15 16 17 18 19 13 14 15 16 17 18 19
   16 17 18 19 20 21 22 20 21 22 23 24 25 26 20 21 22 23 24 25 26
   23 24 25 26 27 28 29 27 28          27 28 29 30 31
   30 31

    四月          五月          六月
日 一 二 三 四 五 六 日 一 二 三 四 五 六 日 一 二 三 四 五 六
                1 2    1 2 3 4 5 6 7    1 2 3 4
    3 4 5 6 7 8 9    8 9 10 11 12 13 14    5 6 7 8 9 10 11
   10 11 12 13 14 15 16 15 16 17 18 19 20 21 12 13 14 15 16 17 18
   17 18 19 20 21 22 23 22 23 24 25 26 27 28 19 20 21 22 23 24 25
   24 25 26 27 28 29 30 29 30 31    26 27 28 29 30

    七月          八月          九月
日 一 二 三 四 五 六 日 一 二 三 四 五 六 日 一 二 三 四 五 六
                1 2    1 2 3 4 5 6    1 2 3
    3 4 5 6 7 8 9    7 8 9 10 11 12 13    4 5 6 7 8 9 10
   10 11 12 13 14 15 16 14 15 16 17 18 19 20 11 12 13 14 15 16 17
   17 18 19 20 21 22 23 21 22 23 24 25 26 27 18 19 20 21 22 23 24
   24 25 26 27 28 29 30 28 29 30 31    25 26 27 28 29 30
   31

    十月          十一月          十二月
日 一 二 三 四 五 六 日 一 二 三 四 五 六 日 一 二 三 四 五 六
                1    1 2 3 4 5    1 2 3
    2 3 4 5 6 7 8    6 7 8 9 10 11 12    4 5 6 7 8 9 10
    9 10 11 12 13 14 15 13 14 15 16 17 18 19 11 12 13 14 15 16 17
   16 17 18 19 20 21 22 20 21 22 23 24 25 26 18 19 20 21 22 23 24
   23 24 25 26 27 28 29 27 28 29 30    25 26 27 28 29 30 31
   30 31
```

(3) 查看 ls 命令的-s 选项的帮助信息。

- 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。

<pre>-r, --reverse -R, --recursive -s, --size -S --sort=WORD</pre>	<pre>逆序排列 递归显示子目录 以块数形式显示每个文件分配的尺寸 sort by file size, largest first sort by WORD instead of name: none (-</pre>
--------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------

(4) 查看/etc 目录下所有文件和子目录的详细信息。

- 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。

```

hch@hch-virtual-machine:~$ cd /etc
hch@hch-virtual-machine:/etc$ ls -al
总用量 1116
drwxr-xr-x 130 root root 12288 9月 23 20:11 .
drwxr-xr-x 20 root root 4096 9月 7 17:28 ..
drwxr-xr-x 3 root root 4096 8月 19 18:40 acpi
-rw-r--r-- 1 root root 3028 8月 19 18:29 adduser.conf
drwxr-xr-x 3 root root 4096 8月 19 18:32 alsa
drwxr-xr-x 2 root root 4096 9月 7 17:37 alternatives
-rw-r--r-- 1 root root 401 7月 17 2019 anacrontab
-rw-r--r-- 1 root root 433 10月 2 2017 apg.conf
drwxr-xr-x 5 root root 4096 8月 19 18:33 apm
drwxr-xr-x 3 root root 4096 8月 19 18:40 apparmor
drwxr-xr-x 6 root root 4096 9月 23 20:09 apparmor.d
drwxr-xr-x 4 root root 4096 9月 23 20:11 appport
-rw-r--r-- 1 root root 769 1月 19 2020 appstream.conf
drwxr-xr-x 7 root root 4096 9月 7 17:43 apt
drwxr-xr-x 3 root root 4096 8月 19 18:41 avahi
-rw-r--r-- 1 root root 2319 2月 25 2020 bash.bashrc
-rw-r--r-- 1 root root 45 1月 26 2020 bash_completion
drwxr-xr-x 2 root root 4096 9月 23 20:11 bash_completion.d
-rw-r--r-- 1 root root 367 4月 15 2020 bindresvport.blacklist
drwxr-xr-x 2 root root 4096 4月 22 2020 binfmt.d
drwxr-xr-x 2 root root 4096 8月 19 18:40 bluetooth
-rw-r----- 1 root root 33 8月 19 18:41 brlapi.key

```

（二） 字符界面下的 Shell 命令操作

（1） 查看当前目录

- ① 启动计算机后默认会启动图形化用户界面，按下 **CTRL+ALT+F3** 切换到字符界面。
- ② 输入一个普通用户的用户名(hch)和口令，登录系统。
- ③ 输入命令“pwd” 显示当前目录

```

Ubuntu 20.04.3 LTS hch-virtual-machine tty3

hch-virtual-machine login: hch
Password:
Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.11.0-36-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

23 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Your Hardware Enablement Stack (HWE) is supported until April 2025.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

hch@hch-virtual-machine:~$ pwd
/home/hch
hch@hch-virtual-machine:~$ _

```

- （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.

- 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+C** 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。

```
hch@hch-virtual-machine:~$ cat>f1
Linux is useful for us all.
You can never imagine how great it is.
^C
hch@hch-virtual-machine:~$ ls
f1                               VMwareTools-10.3.23-17030940.tar.gz
VMwareTools-10.3.23-17030940
hch@hch-virtual-machine:~$ cat f1
Linux is useful for us all.
You can never imagine how great it is.
hch@hch-virtual-machine:~$ _
```

- (3) 向 f1 文件增加以下内容：Why not have a try?

- 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+C** 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。

```
hch@hch-virtual-machine:~$ cat>>f1
Why not have a try?
^C
hch@hch-virtual-machine:~$ cat f1
Linux is useful for us all.
You can never imagine how great it is.
Why not have a try?
hch@hch-virtual-machine:~$
```

- (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。

- ① 输入命令“wc<f1>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。

```
hch@hch-virtual-machine:~$ wc<f1>countf1
hch@hch-virtual-machine:~$ cat countf1
3 19 87
```

- (5) 将 f1 和 countf1 文件的合并为 f 文件。

- 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。

```
hch@hch-virtual-machine:~$ cat f1 countf1>f
hch@hch-virtual-machine:~$ cat f
Linux is useful for us all.
You can never imagine how great it is.
Why not have a try?
3 19 87
```

- (6) 分页显示/etc 目录中所有文件和子目录的信息。

- 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，屏幕的最下一行上还会出现“--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。

```
alsa
alternatives
anacrontab
apg.conf
apm
apparmor
apparmor.d
appport
appstream.conf
apt
avahi
bash.bashrc
bash_completion
bash_completion.d
bindresvport.blacklist
binfmt.d
bluetooth
brlapi.key
brltty
brltty.conf
ca-certificates
ca-certificates.conf
ca-certificates.conf.dpkg-old
calendar
chatscripts
console-setup
cracklib
cron.d
cron.daily
cron.hourly
cron.monthly
crontab
cron.weekly
cups
cupshelpers
dbus-1
hch@hch-virtual-machine:~$ _
```

(7) 仅显示/etc 目录中前 5 个文件和子目录。

- ① 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。

```
hch@hch-virtual-machine:~$ ls /etc|head -n 5
acpi
adduser.conf
alsa
alternatives
anacrontab
```

(8) 清除屏幕内容

- ① 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。

```
hch@hch-virtual-machine:~$
```

(三) 通配符的使用

(1) 显示/bin/目录中所有以 c 为首字母的文件和目录。

① 输入命令“ls /bin/c*”, 屏幕将显示/bin 目录中以 c 开头的文件和目录。

```
hch@hch-virtual-machine:~$ ls /bin/c*
/bin/cal /bin/check-language-support /bin/colrm
/bin/calendar /bin/cheese /bin/column
/bin/calibrate_ppa /bin/chfn /bin/comm
/bin/canberra-gtk-play /bin/chgrp /bin/compose
/bin/cancel /bin/chmod /bin/corelist
/bin/captoinfo /bin/chown /bin/cp
/bin/cat /bin/chown /bin/cpan
/bin/catchsegv /bin/chrt /bin/cpan5.30-x86_64-linux-gnu
/bin/catman /bin/chsh /bin/cpio
/bin/cautious-launcher /bin/chvt /bin/cpp
/bin/cd-create-profile /bin/ciptool /bin/cpp-9
/bin/cd-fix-profile /bin/ckbcomp /bin/c_rehash
/bin/cd-iccdump /bin/cksum /bin/crontab
/bin/cd-it8 /bin/clear /bin/csplitt
/bin/chacl /bin/clear_console /bin/ctstat
/bin/chage /bin/cmp /bin/cupstestppd
/bin/chardet3 /bin/codepage /bin/cut
/bin/chardetect3 /bin/col /bin/cvt
/bin/chattr /bin/colcrt /bin/cvtsudoers
/bin/chcon /bin/colormgr
```

(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。

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,文

```
hch@hch-virtual-machine:~$ ls /bin/c??
/bin/cal /bin/cat /bin/cmp /bin/col /bin/cpp /bin/cut /bin/cvt
```

(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。

① 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。

```

/bin/colrm      /bin/select-editor  /bin/sudoedit
/bin/column     /bin/sensible-browser /bin/sudoreplay
/bin/comm       /bin/sensible-editor  /bin/sum
/bin/compose    /bin/sensible-pager   /bin/syncryptrun
/bin/corelist   /bin/seq               /bin/sync
/bin/cp         /bin/session-migration /bin/syslinux
/bin/cpan       /bin/sessreg           /bin/syslinux-legacy
/bin/cpan5.30-x86_64-linux-gnu /bin/setarch          /bin/system-config-printer
/bin/cpio       /bin/setfacl           /bin/system-config-printer-applet
/bin/cpp        /bin/setfont           /bin/systemctl
/bin/cpp-9      /bin/setkeycodes       /bin/systemd
/bin/c_rehash   /bin/setleds           /bin/systemd-analyze
/bin/crontab    /bin/setlogcons        /bin/systemd-ask-password
/bin/csplit     /bin/setmetamode       /bin/systemd-cat
/bin/ctstat     /bin/setpci            /bin/systemd-cgls
/bin/cupstestppd /bin/setpriv           /bin/systemd-cgtop
/bin/cut        /bin/setsid            /bin/systemd-delta
/bin/cvt        /bin/setterm           /bin/systemd-detect-virt
/bin/cvtsudoers /bin/setupcon          /bin/systemd-escape
/bin/h2ph       /bin/setxkbmap         /bin/systemd-hwdb
/bin/h2xs       /bin/sftp              /bin/systemd-id128
/bin/hbpldecode /bin/sg                /bin/systemd-inhibit
/bin/hciattach  /bin/sh                /bin/systemd-machine-id-setup
/bin/hcidconfig /bin/sha1sum           /bin/systemd-mount
/bin/hciitool   /bin/sha224sum         /bin/systemd-notify
/bin/hd         /bin/sha256sum         /bin/systemd-path
/bin/head       /bin/sha384sum         /bin/systemd-resolve
/bin/help2tags  /bin/sha512sum         /bin/systemd-run
/bin/hex2hcd    /bin/shasum            /bin/systemd-socket-activate
/bin/hexdump    /bin/shotwell          /bin/systemd-stdio-bridge
/bin/hipercdecode /bin/showconsolefont  /bin/systemd-sysusers
/bin/host       /bin/showkey           /bin/systemd-tmpfiles
/bin/hostid     /bin/showrgb           /bin/systemd-tty-ask-password-agent
/bin/hostname   /bin/shred             /bin/systemd-umount
/bin/hostnamectl /bin/shuf
/bin/hp-align   /bin/simple-scan

```

(4) 显示/bin 目录中所有的首字母为 v、w、x、y、z 的文件和目录。

① 输入命令“ls /bin/ll[a-u]*”，屏幕显示/bin 目录中首字母是 v~z 的文件和目录。

```

xwininfo
xwud
x-www-browser
xxd
xz
xzcat
xzcmp
xzdiff
xzegrep
xzfgrep
xzgrep
xzless
xzmore
yelp
yes
ypdomainname
zcat
zcmp
zdiff
zdump
zegrep
zenity
zfgrep
zforce
zgrep

```

(5) 重复上一步操作。

① 输入命令“!!”，自动执行上一步操作中使用过的“ls /bin/ll[a-u]*”命令。

```

xwininfo
xwud
x-www-browser
xxd
xz
xzcat
xzcmp
xzdiff
xzegrep
xzfgrep
xzgrep
xzless
xzmore
yelp
yes
ypdomainname
zcat
zcmp
zdiff
zdump
zegrep
zenity
zfgrep
zforce
zgrep

```

(6) 查看刚执行过的 5 个命令。

① 输入命令“history 5”，显示最近执行过的 5 个命令。

```

hch@hch-virtual-machine:~$ history 5
141 ls /bin/c??
142 ls /bin/[csh]*
143 ls /bin/[[a-u]*
144 history 55
145 history 5

```

(四) 设置手工启动图形化用户界面

(1) 设置开机不启动图形化用户界面。

- ① 按下 CTRL+ALT+F1 键，切换回到图形化用户界面，以超级用户身份登录。
- ② 将/etc/X11/default-display-manager 文件改为 false。
- ③ 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

```

root@hch-virtual-machine:/home/hch/桌面# cd /etc
root@hch-virtual-machine:/etc# cd /etc/X11
root@hch-virtual-machine:/etc/X11# cat >default-display-manager
false
^C
root@hch-virtual-machine:/etc/X11# cat default-display-manager
false

```

(2) 手工启动图形化用户界面。

- 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。

体会：

本次实验是关于用户界面和 **shell** 命令的，要求我们在图形化界面和字符界面进行相应的操作，一开始我不知道如何转换为字符界面，经过搜索和询问才得以解决。在不同的界面下进行操作之后，我对这两种界面更为熟悉，也摸清了其各自的优缺点。

实验题目：进程管理及进程通信

姓名：黄晨航 学号：19120178 实验日期：2021.9.30

实验环境： Linux 环境

实验目的：

7. 利用 Linux 提供的系统调用设计程序，加深对进程概念的理解。
8. 体会系统进程调度的方法和效果。
9. 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。

操作过程及结果：

- (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<stdlib.h>
#include<unistd.h>

int main(){
    printf("process id :%d\n",getpid());
    printf("process group id :%d\n",getpgrp());
    printf("calling process's real user id :%d\n",getpid());
    sleep(5);
    int child_pid;
    child_pid = fork();
    if(child_pid == 0){
        execlp("echo","echo","hello world\n",(char*)0);
        perror("execlp error\n");
        exit(1);
    }
    int i;
    i = wait(0);
    printf("if you want it stop the process(y/n)\n");
    char x;
    scanf("%c",&x);
    if(x == 'y')
        exit(0);
    else
        exit(1);
}
```

结果：

```
hch@hch-virtual-machine:~/桌面/e3$ ./test0.out
process id :2438
process group id :2438
calling process's real user id :2438
hello world

if you want it stop the process(y/n)
y
hch@hch-virtual-machine:~/桌面/e3$
```

- (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 <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>

/*例程1: 利用fork()创建子进程
  用fork()系统调用创建子进程的例子*/
int main()
{
    int i;
    if (fork()) { /*父进程执行的程序段*/
        i = wait(0); /* 等待子进程结束*/
        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(2);
        /*子进程执行的程序段*/
        exit(0); /*向父进程发出结束信号*/
    }
    exit(0);
}
```

结果:

```
hch@hch-virtual-machine:~/桌面/e3$ ./test.out
It is child process.
It is parent process.
The child process,ID number 2718, is finished.
hch@hch-virtual-machine:~/桌面/e3$
```

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: 是由父进程 `fork()` 函数创建形成的, 通过 `exit()` 函数自我结束, 子进程被创建后核心将其分配进一个进程表项和进程标识符, 检查同时运行的进程数目, 并且拷贝进程表项的数据, 由子进程继承父进程所有文件。

- (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。

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
int main()
{
    int i,j;
    printf("My pid is %d, my father's pid is
%d\n",getpid(),getppid());
    for(i=0; i<3; i++)
    {
        if(fork()==0)
        {
            printf("%d pid=%d
ppid=%d\n",i,getpid(),getppid());
        }
        else
        {
            j=wait(0);
            printf("%d:The chile %d is finished.
\n",getpid(),j);
        }
    }
    exit(0);
}
```

结果:

```
hch@hch-virtual-machine:~/桌面/e3$ ./test.out
My pid is 2780, my father's pid is 2625
0 pid=2781 ppid=2780
1 pid=2782 ppid=2781
2 pid=2783 ppid=2782
2782:The chile 2783 is finished.
2781:The chile 2782 is finished.
2 pid=2784 ppid=2781
2781:The chile 2784 is finished.
2780:The chile 2781 is finished.
1 pid=2785 ppid=2780
2 pid=2786 ppid=2785
2785:The chile 2786 is finished.
2780:The chile 2785 is finished.
2 pid=2787 ppid=2780
2780:The chile 2787 is finished.
```

- (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 <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
int main()
{
    int child_pid1,child_pid2,child_pid3;
    int pid,status;
    setbuf(stdout,NULL);
    child_pid1=fork(); /*创建子进程1*/
    if(child_pid1==0)
    {
        execlp("echo","echo","child process 1",(char *)0); /*子进程1 启动其它程序*/
        perror("exec1 error.\n ");
        exit(1);
    }
    child_pid2=fork(); /*创建子进程2*/
    if(child_pid2==0)
    {
        execlp("date","date",(char *)0); /*子进程2 启动其它程序*/
        perror("exec2 error.\n ");
        exit(2);
    }
    child_pid3=fork(); /*创建子进程3*/
    if(child_pid3==0)
    {
        execlp("ls","ls",(char *)0); /*子进程3 启动其它程序*/
        perror("exec3 error.\n ");
        exit(3);
    }
    puts("Parent process is waiting for child process return!");
    while((pid=wait(&status))!=-1) /*等待子进程结束*/
    {
        if(child_pid1==pid) /*若子进程1 结束*/
            printf("child process 1 terminated with status %d\n",(status>>8));
        else
        {
            if(child_pid2==pid) /*若子进程2 结束*/
                printf("child process 2 terminated with status %d\n",(status>>8));
            else
            {
                if(child_pid3==pid) /*若子进程3 结束*/
                    printf("child process 3 terminated with status %d\n" ,(status>>8));
            }
        }
    }
    puts("All child processes terminated.");
    puts("Parent process terminated.");
    exit(0);
}

```

结果:

```

hch@hch-virtual-machine:~/桌面/e3$ ./test.out
Parent process is waiting for child process return!
test.c test.out test.txt
child process 3 terminated with status 0
2021年 10月 14日 星期四 10:35:23 CST
child process 1
child process 2 terminated with status 0
child process 1 terminated with status 0
All child processes terminated.
Parent process terminated.

```

思考: 子进程运行其他程序后, 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。

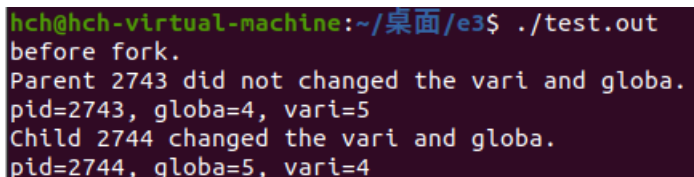
答: 子进程运行其他程序后, 这个进程就完全被新程序替代。由于并没有产生新进程, 所以进程标识号不变, 除此之外的旧进程的其他信息, 代码段, 数据段, 栈段等均被新

程序的信息所替代。新程序从自己的 `main()` 函数开始进行。反复运行此程序发现结束的先后次序是不可预知的，每次运行的结果不一样。原因是当每个子进程运行其他程序是，他们的结束随着其他程序的结束而结束，所以结束的先后次序在变化。

- (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 <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
int globa=4;
int main()
{
    pid_t pid;
    int vari=5;
    printf("before fork.\n");
    if ((pid=fork())<0) /*创建失败处理*/
    {
        printf("fork error.\n");
        exit(0);
    }
    else
    {
        if(pid==0) /*子进程执行*/
        {
            globa++;
            vari--;
            printf("Child %d changed the vari and globa.\n",getpid());
        }
        else /*父进程执行*/
        {
            printf("Parent %d did not changed the vari and globa.\n",getpid());
        }
        printf("pid=%d, globa=%d, vari=%d\n",getpid(),globa,vari); /*都执行*/
        exit(0);
    }
}
```

结果：



```
hch@hch-virtual-machine: ~/桌面/e3$ ./test.out
before fork.
Parent 2743 did not changed the vari and globa.
pid=2743, globa=4, vari=5
Child 2744 changed the vari and globa.
pid=2744, globa=5, vari=4
```

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：父进程被创建后，对父进程的运行环境无影响，因为当子进程在运行时，他有自己的代码段和数据段，这些都可以做修改。但是父进程的代码和数据段是不会随着子进程数据段和代码段的改变而改变。

- (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/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
int main()
{
    int i,r,j,k,l,p1,p2,fd[2];
    char buf[50],s[50];
    pipe(fd);/*建立一个管道fd*/
    while((p1=fork())!=-1);/*创建子进程1*/
    if(p1==0)
    {
        lockf(fd[1],1,0); /*子进程1 执行*/
        /*管道写入端加锁*/
        sprintf(buf,"Child process P1 is sending messages! \n");
        printf("Child process P1! \n");
        write(fd[1],buf,50);
        lockf(fd[1],0,0);
        /*信息写入管道*/
        /*管道写入端解锁*/
        sleep(5);
        j=getpid();
        k=getppid();
        printf("P1 %d is weakup. My parent process ID is %d.\n",j,k);
        exit(0);
    }
    else
    {
        while((p2=fork())!=-1);/*创建子进程2*/
        if(p2==0)
        {
            /*子进程2 执行*/
            lockf(fd[1],1,0);/*管道写入端加锁*/
            sprintf(buf,"Child process P2 is sending messages! \n");
            printf("Child process P2! \n");
            write(fd[1],buf,50);/*信息写入管道*/
            lockf(fd[1],0,0);/*管道写入端解锁*/
            sleep(5);
            j=getpid();
            k=getppid();
            printf("P2 %d is weakup. 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);
        }
    }
}

```

结果：

```
hch@hch-virtual-machine:~/桌面/e3$ ./test.out
Child process P1!
Child process P2!
P2 2883 is weakup. My parent process ID is 2881.
P1 2882 is weakup. My parent process ID is 2881.
Parent 2881: Child process P1 is sending messages!
Parent 2881: Child process P2 is sending messages!
```

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： ①管道是指能够连接一个写进程和一个读进程，并允许他们以生产者-消费者方式进行通信的一个共享文件。由写进程从管道的入端将数据写入管道，而读进程则从管道出端读出数据来进行通信。

②修改睡眠时间和睡眠长度都会引起进程被唤醒的时间不一，因为睡眠时间决定进程在何时睡眠，睡眠长度决定进程何时被唤醒。

③加锁、解锁是为了解决临界资源的共享问题。不用他将会引起无法有效管理数据，即数据会被修导致读错了数据。

- (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。

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：只要在父进程后加上 `wait()` 函数，然后打印“子进程已经结束”，一旦子进程结束，父进程撤销进程。当父进程先于子进程终止时，所有子进程的父进程改变为 `init` 进程，称为由 `init` 进程领养。

- (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。
- 客户端程序：

```

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#define MSGKEY 75
struct msgform
{
    long mtype;
    char mtext[256];
};
int main()
{
    struct msgform msg;
    int msgqid,pid,*pint;
    msgqid=msgget(MSGKEY,0777);/*建立消息队列*/
    pid=getpid();
    pint=(int *)msg.mtext;
    *pint=pid;
    msg.mtype=1;/*定义消息类型*/
    msgsnd(msgqid,&msg,sizeof(int),0);/*发送消息*/
    msgrcv(msgqid,&msg,256,pid,0);/*接受从服务者发来的消息*/
    printf("Client : receive from pid %d\n",* pint);
    exit(0);
}

```

服务器程序:

```

#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#define MSGKEY 75
struct msgform
{
    long mtype;
    char mtext[256];
}msg;
int msgqid;
int main()
{
    int i,pid,* pint;
    void cleanup();
    for(i=0;i<20;i++)/*设置软中断信号的处理程序*/
        signal(i,&cleanup);
    msgqid=msgget(MSGKEY,0777|IPC_CREAT); /*建立消息队列*/
    for(;;)/*等待接受消息*/
    {
        msgrcv(msgqid,&msg,256,1,0);/* 接受消息*/
        pint=(int *)msg.mtext;
        pid=*pint;
        printf("Server :receive from pid %d\n",pid);/*显示消息来源*/
        msg.mtype=pid;
        *pint=getpid();/*加入自己的进程标识*/
        msgsnd(msgqid,&msg,sizeof(int),0); /*发送消息*/
    }
    exit(0);
}
void cleanup()
{
    msgctl(msgqid,IPC_RMID,0);
    exit(0);
}

```

结果:

```
hch@hch-virtual-machine:~/桌面/e3$ gcc test.c -o test.out
hch@hch-virtual-machine:~/桌面/e3$ ./test.out
Clint : receive from pid 2599
```

```
hch@hch-virtual-machine:~/桌面/e3$ gcc server.c -o server.out
hch@hch-virtual-machine:~/桌面/e3$ ./server.out
Server :receive from pid 2623
```

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:还可以用信号量机制来实现。信号量是一个整形计数器,用来控制多个进程对共享资源的访问。或者通过消息队列信号机制,通过向消息队列发送消息、接收信息来实现进程间的通信。没兴趣。

- (9) 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 <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>

int main()
{
    int i,j,k;
    void func();
    signal(18,&func); /*设置18号信号的处理程序*/
    if(i=fork())/*创建子进程*/
    { /*父进程执行*/
        j=kill(i,18);/*向子进程发送信号*/
        printf("Parent: signal 18 has been sent to child %d,returned %d.\n",i,j);
        k=wait(0); /*父进程被唤醒*/
        printf("After wait %d,Parent %d: finished.\n",k,getpid());
    }
    else
    { /*子进程执行*/
        sleep(10);
        printf("Child %d: A signal from my parent is recived.\n",getpid());
    } /*子进程结束,向父进程发子进程结束信号*/
    //exit(0);
}

void func() /*处理程序*/
{
    int m;
    m=getpid();
    printf("I am Process %d: It is signal 18 processing function.\n",m);
}
```

结果:

```
hch@hch-virtual-machine:~/桌面/e3$ gcc test.c -o test.out
hch@hch-virtual-machine:~/桌面/e3$ ./test.out
Parent: signal 18 has been sent to child 2804,returned 0.
I am Process 2804: It is signal 18 processing function.
Child 2804: A signal from my parent is recived.
After wait 2804,Parent 2803: finished.
```


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：硬中断是由外部硬件产生的，而软化中断是 **cpu** 根据软件的某条指令或者软件对标志寄存器的某个标志位的设置而产生的。

体会：

本次实验设计到进程管理和通信的基本操作，一开始在虚拟机编译 **c** 文件都有困难，但在查询了诸多资料后解决了最基本的编译问题。在理解指导书上例程的时候发现了很多错误，虽然对于这些陌生的函数一知半解，但逐步排错的过程让我对进程的理解更为深刻了。

实验题目：Linux 进程调度与系统监视

姓名：黄晨航 学号：19120178 实验日期：2021.10.28

实验环境： Linux 环境

实验目的：

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

实验内容及操作过程：

1. 作业和进程的基本管理

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。

操作过程及结果：

- 1) 以超级用户(root)身份登录到字符界面。
- 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，在显示作业号的同时还显示进程号。

结果：

看到 f4 的作业号是 1，启动的后台作业 f5 可以看到作业号为 2，进程号为 2623。

```
root@hch-virtual-machine:/home/hch# vi f4
[1]+ 已停止                  vi f4
root@hch-virtual-machine:/home/hch# find / -name inittab > f5 &
[2] 2623
root@hch-virtual-machine:/home/hch#
```

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。

操作过程及结果：

- 1) 输入命令“jobs”，查看当前系统中的所有作业。

```
root@hch-virtual-machine:~# jobs
[1]+  已停止                  vi f4
[2]-  运行中                  find / -name inittab > f5 &
```

- 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。

```
root@hch-virtual-machine:~# fg 2
find / -name inittab > f5
```

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

```
root@hch-virtual-machine:~# cat f5
/etc/inittab
```

可以得到 f5 的文件目录为/etc/inittab

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

```
root@hch-virtual-machine:~# jobs
[1]+  已停止                  vi f4
```

- 5) 输入命令“kill -9 %1”，终止“vi f4”作业。稍后查看 jobs 时发现当前没有任何作业。

```
root@hch-virtual-machine:~# jobs
[1]+  已停止                  vi f4
root@hch-virtual-machine:~# kill -9 %1
[1]+  已杀死                  vi f4
root@hch-virtual-machine:~# jobs
root@hch-virtual-machine:~#
```

- 6) 输入命令“ps -1”，查看进程的相关信息。

```
root@hch-virtual-machine:~# ps -1
  PID TTY          STAT       TIME COMMAND
    1 ?           Ss         0:08 /sbin/init splash
```

- 7) 输入命令“who -H”，查看用户信息。

```
root@hch-virtual-machine:~# who -H
名称  线路      时间      备注
hch   :0        2021-10-28 09:53 (:0)
```

2. at 进程调度

操作要求 1：

设置一个调度，要求在 2038 年 1 月 1 日 0 时，向所有用户发送新年快乐的问候。

操作过程及结果：

- 1) 超级用户输入命令“at 00:00 01012038”，设置 2038 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>”调度提示符的第三行，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:

```
root@hch-virtual-machine:~# at 00:00 01012038
warning: commands will be executed using /bin/sh
at> wall Happy New Year!
at> <EOT>
job 1 at Fri Jan 1 00:00:00 2038
```

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。

操作过程及结果:

- 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。“shutdown -r +2” 命令与 “reboot +2” 命令效果相同, 都是在 2 分钟后重新启动。
- 4) 光标移动到 “at>” 调度提示符的第三行, 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:

```
root@hch-virtual-machine:~# at now +5 minutes
warning: commands will be executed using /bin/sh
at> wall please logout;the computer will restart.
at> shutdown -r +2
at> <EOT>
job 3 at Thu Oct 28 10:46:00 2021
```

操作要求 3:

查看所有的 at 调度, 并删除 2038 年 1 月 1 日执行的调度任务。

操作过程及结果:

- 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 “atrm 1” 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 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 分钟后重新启动。

结果:

```
root@hch-virtual-machine:~# atq
5          Thu Oct 28 11:02:00 2021 a root
1          Fri Jan 1 00:00:00 2038 a root
root@hch-virtual-machine:~# atrm 1
root@hch-virtual-machine:~# atq
5          Thu Oct 28 11:02:00 2021 a root
```

3. cron 进程调度

操作要求 1：

hch 用户设置 crontab 调度，要求每天上午 8 点 30 分查看系统的进程状态，并将查看结果保存于 ps.log 文件。

操作过程及结果：

- 1) 以普通用户 hch 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”，保存并退出编辑器，显示 “crontab: installing new crontab” 信息。
- 4) 输入命令 “crontab -l”，查看 hch 用户的 cron 调度内容。
- 5) 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 hch 用户。
- 6) 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：

```
hch@hch-virtual-machine:~$ crontab -e
crontab: installing new crontab
hch@hch-virtual-machine:~$ crontab -l
30 8 * * * ps >ps.log

root@hch-virtual-machine:~# date 11200829
2021年 11月 20日 星期六 08:29:00 CST
root@hch-virtual-machine:~# exit
注销
hch@hch-virtual-machine:~$ crontab -l
30 8 * * * ps >ps.log
```

操作要求 2：

hch 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。

操作过程及结果：

- 1) 再次输入命令 “crontab -e”，出现 vi 编辑器，按下 “i”，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) 保存并退出编辑器。
- 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 hch 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：


```
hch@hch-virtual-machine:~$ crontab -e
crontab: installing new crontab
hch@hch-virtual-machine:~$ su -
密码:
root@hch-virtual-machine:~# date 03312359
2021年 03月 31日 星期三 23:59:00 CST
root@hch-virtual-machine:~# exit
注销
hch@hch-virtual-machine:~$ crontab -l
30 8 * * * ps >ps.log
0 0 * */3 *who >who.log
```

操作要求 3:

查看 cron 调度内容，最后删除此调度。

操作过程及结果:

- 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) 再次输入输入命令 “crontab -l”，此时无 cron 调度内容。

结果:

```
hch@hch-virtual-machine:~$ crontab -l
30 8 * * * ps >ps.log
0 0 * */3 *who >who.log
hch@hch-virtual-machine:~$ crontab -r
hch@hch-virtual-machine:~$ crontab -l
no crontab for hch
```

4. 系统性能监视

操作要求 1:

利用 Shell 命令监视系统性能。

操作过程及结果:

- 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。

```
top - 13:03:34 up 14 min, 1 user, load average: 0.05, 0.23, 0.25
任务: 284 total, 1 running, 283 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.3 us, 0.7 sy, 0.0 ni, 99.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0
MiB Mem : 1952.9 total, 446.6 free, 792.3 used, 713.9 buff/cach
MiB Swap: 923.3 total, 923.3 free, 0.0 used. 996.7 avail Mem
```

进程号	USER	PR	NI	VIRT	RES	SHR	%CPU	%MEM	TIME+
2517	hch	20	0	20668	4084	3320 R	0.7	0.2	0:00.28
1259	root	20	0	156872	8848	7816 S	0.3	0.4	0:03.04
1296	kernoops	20	0	11264	448	0 S	0.3	0.0	0:00.04
2071	hch	20	0	129332	19176	17032 S	0.3	1.0	0:01.34
2308	root	20	0	0	0	0 I	0.3	0.0	0:00.38
1	root	20	0	102064	11308	8224 S	0.0	0.6	0:04.06
2	root	20	0	0	0	0 S	0.0	0.0	0:00.03
3	root	0	-20	0	0	0 I	0.0	0.0	0:00.00
4	root	0	-20	0	0	0 I	0.0	0.0	0:00.00
6	root	0	-20	0	0	0 I	0.0	0.0	0:00.00
9	root	0	-20	0	0	0 I	0.0	0.0	0:00.00
10	root	20	0	0	0	0 S	0.0	0.0	0:00.00
11	root	20	0	0	0	0 S	0.0	0.0	0:00.00
12	root	20	0	0	0	0 S	0.0	0.0	0:00.17
13	root	20	0	0	0	0 I	0.0	0.0	0:00.46
14	root	rt	0	0	0	0 S	0.0	0.0	0:00.01
15	root	-51	0	0	0	0 S	0.0	0.0	0:00.00

- 2) 按下 M 键，所有进程按照内存使用率排列。

```
top - 13:02:40 up 13 min, 1 user, load average: 0.12, 0.28, 0.27
任务: 284 total, 1 running, 283 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.7/1.0 2[|
```

进程号	USER	PR	NI	VIRT	RES	SHR	%CPU	%MEM	TIME+
1867	hch	20	0	3650324	242680	113492 S	1.3	12.1	0:16.92
1674	hch	20	0	278828	64564	40124 S	1.3	3.2	0:06.47
2496	hch	20	0	1056120	64064	47784 S	1.0	3.2	0:01.97
2019	hch	20	0	721220	62416	46468 S	0.0	3.1	0:01.15
1696	hch	20	0	554128	36528	30592 S	0.0	1.8	0:00.26
2333	hch	20	0	452048	34736	24980 S	0.0	1.7	0:00.55
1890	hch	20	0	287396	33708	19916 S	0.0	1.7	0:03.92
1949	hch	20	0	682820	33608	27708 S	0.0	1.7	0:00.61
2009	hch	20	0	699908	33280	22572 S	0.0	1.7	0:00.88
2064	hch	20	0	358424	31984	21164 S	0.0	1.6	0:00.95
2013	hch	20	0	431808	31548	21196 S	0.0	1.6	0:00.86
1988	hch	20	0	579708	31396	21068 S	0.0	1.6	0:00.68
1931	hch	20	0	847864	31256	27088 S	0.0	1.6	0:00.42
736	root	20	0	858040	30988	16280 S	0.0	1.5	0:05.30
1999	hch	20	0	357672	30904	20696 S	0.0	1.5	0:00.83
2054	hch	20	0	357220	30600	20296 S	0.0	1.5	0:00.88
1892	hch	20	0	209664	30524	20464 S	0.0	1.5	0:00.80
1964	hch	20	0	2607668	26904	21872 S	0.0	1.3	0:00.30
1922	hch	20	0	399504	26796	22960 S	0.0	1.3	0:00.27

- 3) 按下 T 键，所有进程按照执行时间排列。

```
top - 13:03:10 up 14 min, 1 user, load average: 0.07, 0.25, 0.26
任务: 284 total, 1 running, 283 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.3 us, 0.3 sy, 0.0 ni, 99.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0
MiB Mem : 1952.9 total, 446.7 free, 792.3 used, 713.9 buff/cach
MiB Swap: 923.3 total, 923.3 free, 0.0 used. 996.7 avail Mem
```

进程号	USER	PR	NI	VIRT	RES	SHR	%CPU	%MEM	TIME+
1867	hch	20	0	3650324	242680	113492 S	0.0	12.1	0:17.12
1674	hch	20	0	278828	64564	40124 S	0.0	3.2	0:06.64
736	root	20	0	858040	30988	16280 S	0.0	1.5	0:05.30
1	root	20	0	102064	11308	8224 S	0.0	0.6	0:04.06
1890	hch	20	0	287396	33708	19916 S	0.0	1.7	0:03.93
1630	hch	9	-11	1155460	19644	15220 S	0.0	1.0	0:03.80
723	root	20	0	253960	13152	9616 S	0.0	0.7	0:03.31
1259	root	20	0	156872	8848	7816 S	0.0	0.4	0:03.00
708	root	20	0	347760	22340	19120 S	0.0	1.1	0:02.44
2496	hch	20	0	1056120	64064	47784 S	0.3	3.2	0:02.12
707	message+	20	0	9868	6420	3960 S	0.0	0.3	0:01.94
178	root	0	-20	0	0	0 I	0.0	0.0	0:01.79
2071	hch	20	0	129332	19176	17032 S	0.3	1.0	0:01.30
359	root	19	-1	51900	18436	16624 S	0.0	0.9	0:01.21
2019	hch	20	0	721220	62416	46468 S	0.0	3.1	0:01.16
1885	hch	20	0	397356	10620	8980 S	0.0	0.5	0:01.15
1624	hch	20	0	19092	10340	8076 S	0.0	0.5	0:01.01

4) 最后按下 P 键，恢复按照 CPU 使用率排列所有进程。

```
top - 13:03:34 up 14 min, 1 user, load average: 0.05, 0.23, 0.25
任务: 284 total, 1 running, 283 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.3 us, 0.7 sy, 0.0 ni, 99.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0
MiB Mem : 1952.9 total, 446.6 free, 792.3 used, 713.9 buff/cach
MiB Swap: 923.3 total, 923.3 free, 0.0 used. 996.7 avail Mem
```

进程号	USER	PR	NI	VIRT	RES	SHR	%CPU	%MEM	TIME+
2517	hch	20	0	20668	4084	3320 R	0.7	0.2	0:00.28
1259	root	20	0	156872	8848	7816 S	0.3	0.4	0:03.04
1296	kernoops	20	0	11264	448	0 S	0.3	0.0	0:00.04
2071	hch	20	0	129332	19176	17032 S	0.3	1.0	0:01.34
2308	root	20	0	0	0	0 I	0.3	0.0	0:00.38
1	root	20	0	102064	11308	8224 S	0.0	0.6	0:04.06
2	root	20	0	0	0	0 S	0.0	0.0	0:00.03
3	root	0	-20	0	0	0 I	0.0	0.0	0:00.00
4	root	0	-20	0	0	0 I	0.0	0.0	0:00.00
6	root	0	-20	0	0	0 I	0.0	0.0	0:00.00
9	root	0	-20	0	0	0 I	0.0	0.0	0:00.00
10	root	20	0	0	0	0 S	0.0	0.0	0:00.00
11	root	20	0	0	0	0 S	0.0	0.0	0:00.00
12	root	20	0	0	0	0 S	0.0	0.0	0:00.17
13	root	20	0	0	0	0 I	0.0	0.0	0:00.46
14	root	rt	0	0	0	0 S	0.0	0.0	0:00.01
15	root	-51	0	0	0	0 S	0.0	0.0	0:00.00

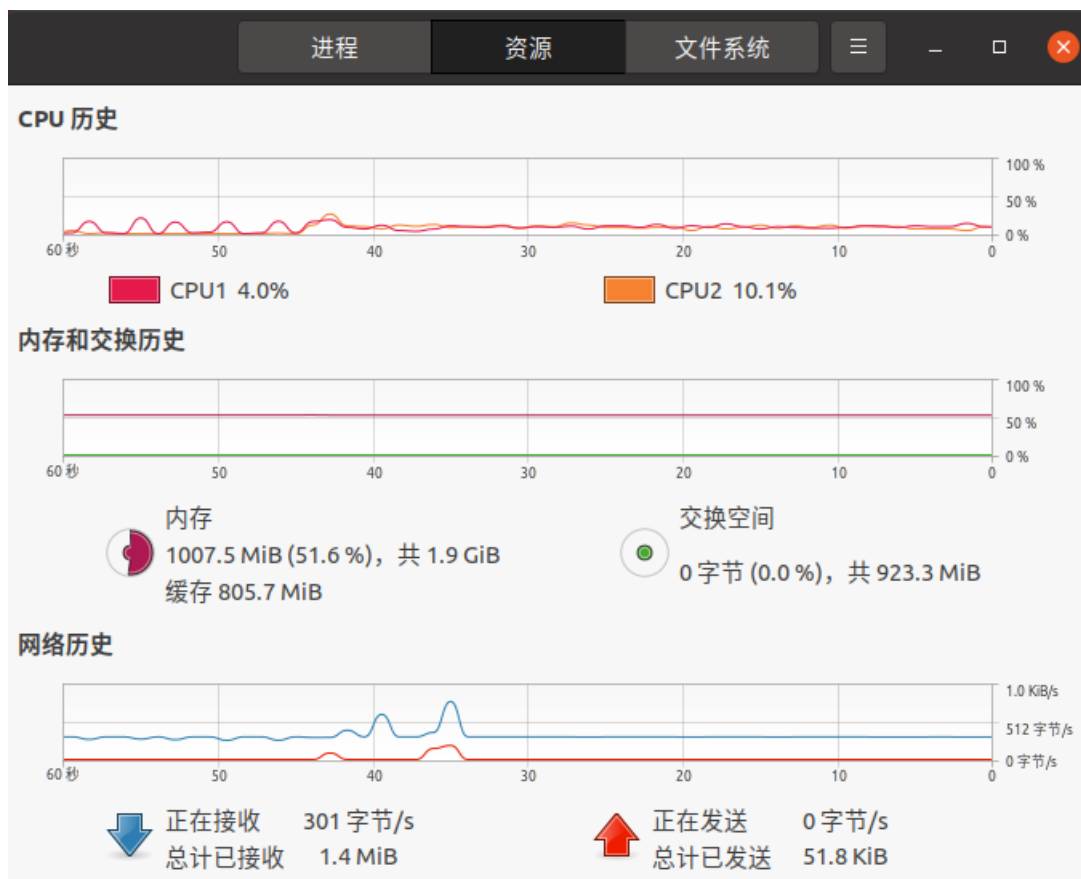
5) 按下 CTRL+C 组合键结束 top 命令。

操作要求 2:

利用系统监视器工具监视 CPU 使用情况。

操作过程及结果:

- 1) 启动 Ubuntu 图形化界面，依次单击显示应用程序、系统监视器，打开系统监视器窗口。
- 2) 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。



操作要求 3:

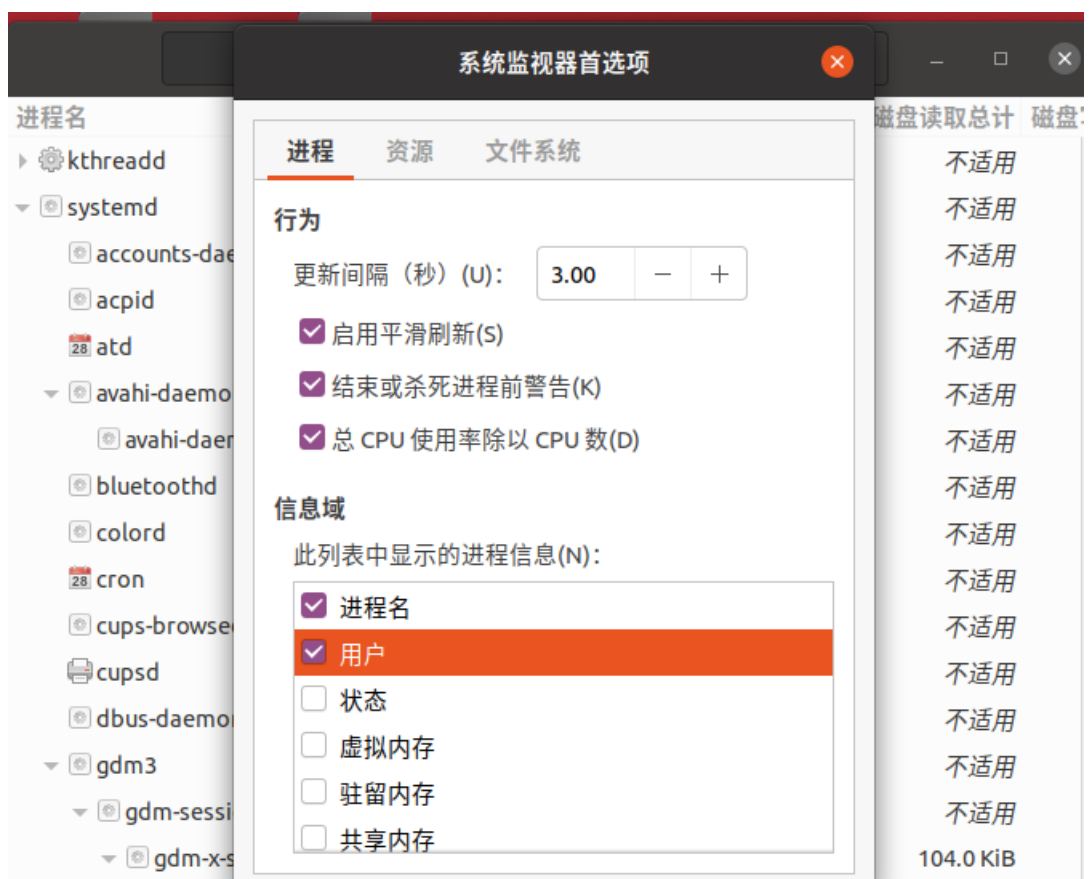
利用系统监视器查看当前所有的进程，要求显示出启动进程的用户。

操作过程及结果:

- 1) 在系统监视器窗口单机进程列表选项卡，默认显示出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。在进程选项卡中，选中进程与栏的用户复选框，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。



进程						
资源						
文件系统						
进程名	用户	% CPU	ID	内存	磁盘读取总计	磁盘写入总计
▶ kthreadd	root	0	2	不适用	不适用	不适用
▼ systemd	root	0	1	3.0 MiB	不适用	不适用
accounts-daemon	root	0	694	1.0 MiB	不适用	不适用
acpid	root	0	695	72.0 KiB	不适用	不适用
atd	daemon	0	752	172.0 KiB	不适用	不适用
▼ avahi-daemon: running [h	avahi	0	699	328.0 KiB	不适用	不适用
avahi-daemon: chroot	avahi	0	764	328.0 KiB	不适用	不适用
bluetoothd	root	0	702	396.0 KiB	不适用	不适用
colord	colord	0	1530	5.2 MiB	不适用	不适用
cron	root	0	703	204.0 KiB	不适用	不适用
cups-browsed	root	0	805	1.6 MiB	不适用	不适用
cupsd	root	0	792	1.3 MiB	不适用	不适用
dbus-daemon	messagebus	0	707	2.4 MiB	不适用	不适用
▼ gdm3	root	0	847	1.2 MiB	不适用	不适用
▼ gdm-session-wor	root	0	1611	1.4 MiB	不适用	不适用
gdm-x-session	hch	0	1671	652.0 KiB	104.0 KiB	104.0 KiB

操作要求 4：

利用系统监视器查看所有的文件系统。

操作过程及结果：

- 1) 在系统监视器窗口单机进程文件系统选项卡，显示出当前 Ubuntu 系统中主要的文件系统。

文件系统						
设备	目录	类型	总计	可用	已用	
/dev/sda5	/	ext4	20.5 GB	9.9 GB	9.5 GB	43%
/dev/sda1	/boot/efi	vfat	535.8 MB	535.8 MB	4.1 kB	0%
/dev/sr0	/media/hch/U	iso9660	3.1 GB	0 字节	3.1 GB	100%

- 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。在文件系统选项卡中，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。单击关闭按钮，显示全部的文件系统的信息。

系统监视器首选项

进程 资源 文件系统

行为

更新间隔 (秒) (U): 5.00 - +

☒ 显示全部文件系统(A)

信息域

列表中显示的文件系统信息(N):

☒ 设备

☒ 目录

☒ 类型

☒ 总计

☐ 空闲

☒ 可用

☒ 已用

/dev/sda5	/	ext4	20.5 GB	9.9 GB	9.5 GB	43%
/dev/sda1	/boot	vfat	535.8 MB	535.8 MB	4.1 kB	0%
udev	/dev					0%
hugetlbfs	/dev/					0%
mqueue	/dev/					0%
devpts	/dev/					0%
tmpfs	/dev/					0%
/dev/sr0	/medi	iso9660	3.1 GB	0 字节	3.1 GB	100%
proc	/proc					0%
binfmt_mi	/proc					0%
tmpfs	/run					0%
tmpfs	/run/					0%
tmpfs	/run/					0%
gvfsd-fuse	/run/					0%
vmware-vn	/run/					0%
/dev/loop	/snap					100%
/dev/loop	/snap					100%

文件系统						
设备	目录	类型	总计	可用	已用	
/dev/sda5	/	ext4	20.5 GB	9.9 GB	9.5 GB	48%
/dev/sda1	/boot/efi	vfat	535.8 MB	535.8 MB	4.1 kB	0%
udev	/dev	devtmpfs	993.5 MB	993.5 MB	0 字节	0%
hugetlbfs	/dev/hugepages	hugetlbfs	0 字节	0 字节	0 字节	0%
mqueue	/dev/mqueue	mqueue	0 字节	0 字节	0 字节	0%
devpts	/dev/pts	devpts	0 字节	0 字节	0 字节	0%
tmpfs	/dev/shm	tmpfs	1.0 GB	1.0 GB	0 字节	0%
/dev/sr0	/media/hch/U	iso9660	3.1 GB	0 字节	3.1 GB	100%
proc	/proc	proc	0 字节	0 字节	0 字节	0%
binfmt_misc	/proc/sys/fs/binfmt_misc	binfmt_misc	0 字节	0 字节	0 字节	0%
tmpfs	/run	tmpfs	204.8 MB	203.2 MB	1.6 MB	0%
tmpfs	/run/lock	tmpfs	5.2 MB	5.2 MB	4.1 kB	0%
tmpfs	/run/user/1000	tmpfs	204.8 MB	204.7 MB	49.2 kB	0%
gvfsd-fuse	/run/user/1000/gvfs	gvfsd-fuse	0 字节	0 字节	0 字节	0%
vmware-vmblock	/run/vmblock-fuse	vmware-vmblock-fuse	0 字节	0 字节	0 字节	0%
/dev/loop7	/snap/bare/5	squashfs	131.1 kB	0 字节	131.1 kB	100%
/dev/loop6	/snap/core18/1000	squashfs	58.2 MB	0 字节	58.2 MB	100%

操作要求 5:

利用系统日志工具查看系统日志。

操作过程及结果:

- 1) 依次单击显示应用程序、日志，打开系统日志窗口。可分别查看各类系统日志。

日志		
12:48 至 13:25		
重要	rfkill: input handler disabled	12:53
全部	ISO 9660 Extensions: RRIP_1991A	
	Bluetooth: RFCOMM ver 1.11	
应用程序	rfkill: input handler enabled	
	NET: Registered protocol family 40	
系统	loop9: detected capacity change from 0 to 8	
	sched: RT throttling activated	
安全	IPv6: ADDRCONF(NETDEV_CHANGE): ens33: link becomes ready	
	e1000: ens33 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: None	
硬件	Bluetooth: BNEP socket layer initialized	12:48
	audit: type=1400 audit(1635828537.326:11): apparmor="STATUS" operation="profile_load" profile="unconfi...	
	AES CTR mode by8 optimization enabled	
	AVX2 version of gcm_enc/dec engaged.	
	cryptd: max_cpu_qlen set to 1000	
	RAPL PMU: API unit is 2^-32 Joules, 0 fixed counters, 10737418240 ms ovfl timer	
	[drm] Initialized vmwgfx 2.18.0 20200114 for 0000:00:0f.0 on minor 0	
	usbcore: registered new interface driver btusb	
	Console: switching to colour frame buffer device 100x37	
	fbcon: svgadrmfb (fb0) is primary device	
	[drm] SM5 support available.	
	[TTM] Zone kernel: Available graphics memory: 999880 KiB	

体会：

本次实验主要是熟悉了进程的调度，大体上比较简单，跟着指导书一步步走就顺利完成了。

实验题目：用户与组群管理

姓名：黄晨航 学号：19120178 实验日期：2021.11.4

实验环境： Linux 环境

实验目的：

14. 理解/etc/passwd 和/etc/group 文件的含义。
15. 掌握桌面环境下管理用户与组群的方法。
16. 掌握利用 Shell 命令管理用户与组群的方法。
17. 掌握批量新建用户账号的步骤和方法。

实验内容及操作过程：

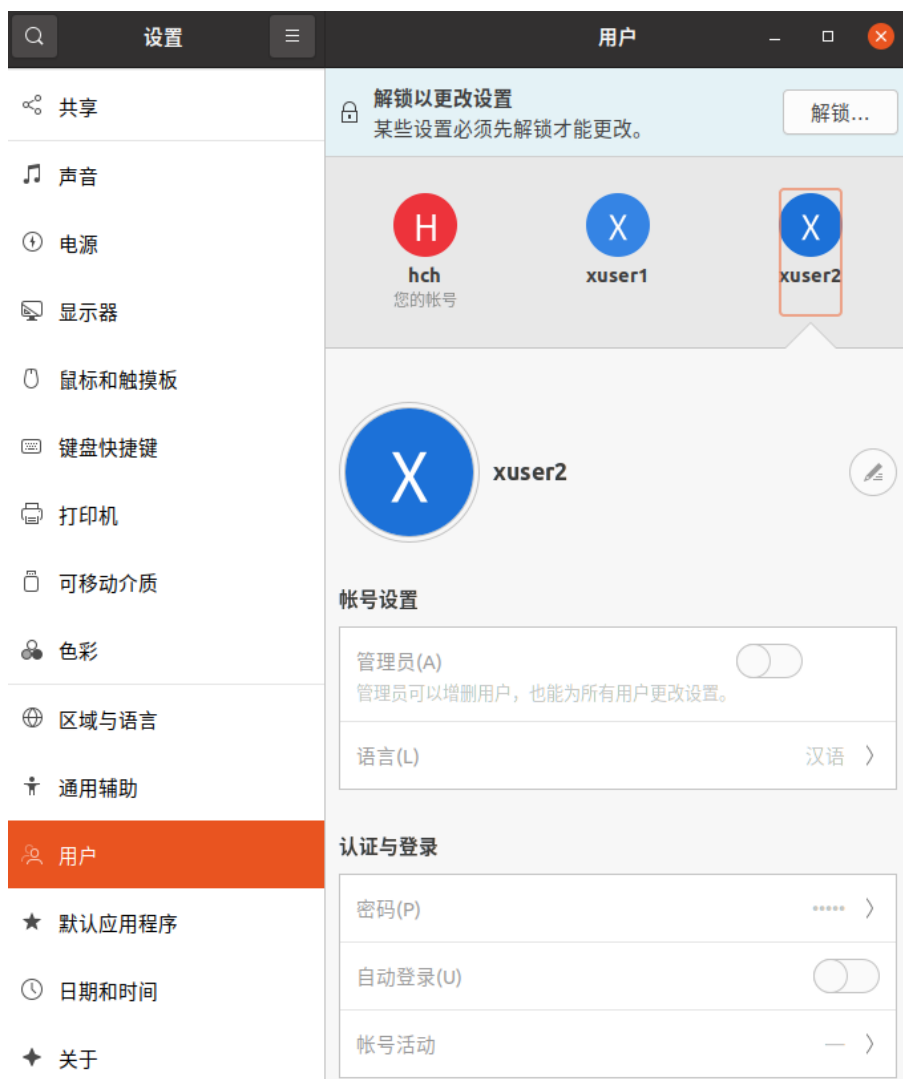
5. 桌面环境下管理用户与组群

操作要求 1：

新建两个用户账号，其用户名为 xuser1 和 xuser2，口令为“e12ut59er”和“wfult28er”。

操作过程及结果：

- 5) 以超级用户身份登录到图形界面，依次单击设置、用户，启动用户管理者窗口。
- 6) 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”，在密码文本框中输入口令“e12ut59er”，在确认密码文本框中再次输入口令“e12ut59er”，然后单击确定按钮。
- 7) 用同样的方法新建用户 xuser2，完成后窗口如下所示。



- 8) 启动文本编辑器，打开 `/etc/passwd` 和 `/etc/passwd`-文件将发现文件的末尾出现表示 `xuser1` 和 `xuser2` 用户账户的信息。打开 `/etc/group` 和 `/etc/group`-文件将发现文件末尾出现表示 `xuser1` 和 `xuser2` 私人组群的信息。

```
48 xuser1:x:1001:1001:xuser1,,,:/home/xuser1:/bin/bash
49 xuser2:x:1002:1002:xuser2,,,:/home/xuser2:/bin/bash

75 xuser1:x:1001:xuser1
76 xuser2:x:1002:xuser2
```

- 9) 按下 `CTRL+ALT+F2` 组合键切换到第 2 个虚拟终端，输入用户名 `xuser2` 和相应的密码可登录 Linux 系统，说明新建用户操作已成功。
- 10) 输入 `pwd` 命令，屏幕显示用户登录后进入用户主目录 `/home/xuser2`。

```
xuser2@hch-virtual-machine:~$ pwd
/home/xuser2
xuser2@hch-virtual-machine:~$
```

- 11) 输入 `exit` 命令，`xuser2` 用户退出登录。
- 12) 返回 Ubuntu 桌面环境。

操作要求 2：

锁定 xuser2 用户账号。

操作过程及结果：

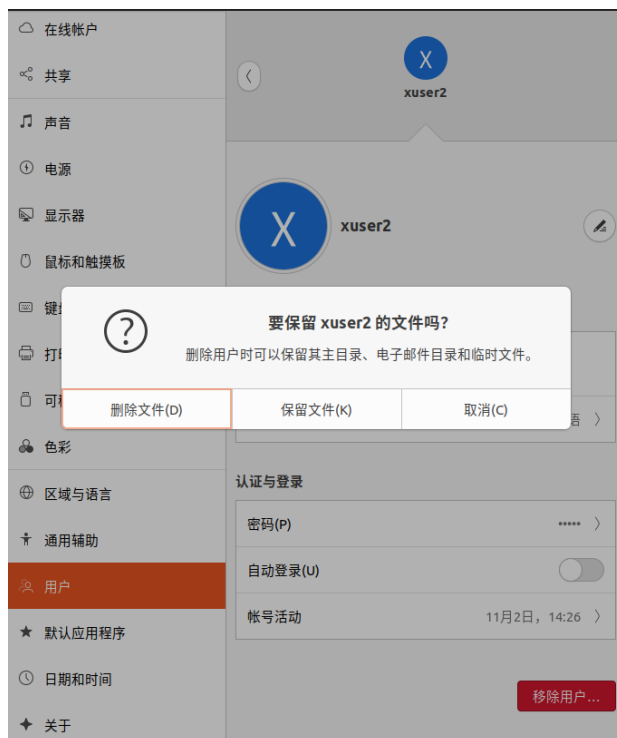
- 1) 在用户管理者窗口选中 **xuser2**，单击工具栏上的属性按钮，打开用户属性窗口。
- 2) 选中账户信息选项卡让本地口令被锁复选框被选中，单击确定按钮，返回用户管理者窗口。
- 3) 按下 CTRL+ALT+F2 组合键，再次切换到第 2 个虚拟终端，输入用户名 **xuser2** 和对应密码，发现 **xuser2** 用户无法登录 Linux 系统，说明 **xuser2** 用户账号的确已被锁定。
- 4) 返回 Ubuntu 桌面环境。

操作要求 3：

删除 xuser2 用户。

操作过程及结果：

- 1) 在用户管理者窗口选中 **xuser2**，单击移除用户按钮。



操作要求 4：

新建两个组群，分别是 myusers 和 temp。

操作过程及结果：

- 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) 用相同方法新建 temp 组群。

操作要求 5：

修改 myusers 组群属性，将 xuser1 和 hch 用户加入 myusers 组群。

操作过程及结果：

- 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 hch 和 xuser1 前的复选框，设置 hch 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。

操作要求 6：

删除 temp 组群。

操作过程及结果：

- 1) 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，单击是即可。

6. 编辑用户配置文件

操作要求 1：

删除 temp 组群。

操作过程及结果：

- 1) 依次单击系统菜单、管理、用户配置文件编辑器，打开 User Profile Editor 窗口。
- 2) 单击添加按钮，弹出 Add Profile 窗口，在 Profile name 文本框中输入用户配置文件名配置文件名“myusers-profile”。单击添加按钮，回到 User Profile Editor。

操作要求 2：

设置 myusers-profile 用户配置文件的内容：应用程序的默认字体为中易宋体 18030，桌面背景为花园。

操作过程及结果：

- 1) 在 User Profile Editor 窗口选中“myusers-profile”文件，单击编辑按钮，出现编辑配置文件 myusers-profile 窗口。
- 2) 在编辑配置用户窗口中依次单击系统菜单、首选项、字体，打开字体首选项对话框，如图所示。单击应用程序字体的字体列表，出现拾取字体对话框，从字体族选择“中易宋体 18030”，并单击确定按钮。
- 3) 回到字体首选项对话框，此时窗口中的字体发生变化，单击窗口右上角的关闭按钮，关闭此对话框。

- 4) 在编辑配置用户窗口中依次单击系统菜单、首选项、桌面背景，打开桌面背景首选项对话框，选择“花园”。此时编辑配置用户窗口的桌面也发生变化，最后单击关闭按钮。
- 5) 单击编辑配置用户窗口配置文件菜单的保存项，保存用户配置文件的修改内容。最后单击编辑配置用户窗口右上角的关闭按钮，回到 **User Profile Editor** 窗口。

操作要求 3：

设置 **xuser1** 的用户配置文件为 **myusers-profile**。

操作过程及结果：

- 1) 在 **User Profile Editor** 窗口选中“**myusers-profile**”文件，单击 **Users** 按钮，出现配置文件 **myusers-profile** 窗口。
- 2) 选中 **xuser1** 用户的复选框，最后单击关闭按钮。
- 3) 单击系统菜单的注销项，超级用户退出 **Ubuntu** 桌面环境。
- 4) 以 **xuser1** 用户登录，并启动 **GNOME** 桌面环境，查看应用程序的字体和桌面环境。

7. 利用 Shell 命令管理用户与组群

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**。。

操作过程及结果：

- 1) 以超级用户身份登录虚拟终端。
- 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。

```
root@hch-virtual-machine:~# useradd -g myusers duser
root@hch-virtual-machine:~# passwd duser
新的 密码:
重新输入新的 密码:
passwd: 已成功更新密码
```

- 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。

```
duser:x:1004:1004:~/home/duser:/bin/sh
```

- 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。

```
myusers:x:1004:
```

- 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。

操作要求 2：

将 **duser** 用户设置为不需口令就能登录。

操作过程及结果：

- 1) 按下 ALT+F3 组合键，切换到正被超级用户使用的虚拟终端。
- 2) 输入命令 “passwd -d duser”。

```
root@hch-virtual-machine:~# passwd -d duser
passwd: 密码过期信息已更改。
```

- 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。

操作要求 3:

查看 duser 用户的相关信息。

操作过程及结果:

- 1) 在第 3 个虚拟终端输入命令 “id duser”，显示 duser 用户的用户 ID(UID)、主要组群的名称和 ID(GID)。

```
root@hch-virtual-machine:~# id duser
用户id=1004(duser) 组id=1004(myusers) 组=1004(myusers)
```

操作要求 4:

从普通用户 duser 切换为超级用户。

操作过程及结果:

- 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。

操作要求 5:

一次性删除 duser 用户及其工作目录。

操作过程及结果:

- 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。

```
rrr:x:1003:1003:rrr,,,:/home/rrr:/bin/bash
root@hch-virtual-machine:~#
```

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

```
root@hch-virtual-machine:~# ls /home
hch  rrr  xuser1  xuser2
```

操作要求 6:

新建组群 mygroup。

操作过程及结果:

- 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。

```
mygroup:x:1005:
```

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

```
mygroup:x:1005:
```

操作要求 7:

将 mygroup 组群改名为 newgroup。

操作过程及结果:

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

```
root@hch-virtual-machine:~# groupadd mygroup
root@hch-virtual-machine:~# cat /etc/group
```

- 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”。

```
newgroup:x:1005:
```

操作要求 8:

删除 newgroup 组群。

操作过程及结果:

- 1) 超级用户输入 “groupdel newgroup” 命令，删除 newgroup 组群。

```
root@hch-virtual-machine:~# groupdel newgroup
root@hch-virtual-machine:~#
```

8. 批量新建多个用户账号

操作要求:

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。

操作过程及结果:

- 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。修改完成后保存并退出。

```
s080101:x:601:600::/home/s080101:/bin/bash
s080102:x:601:600::/home/s080102:/bin/bash
s080103:x:601:600::/home/s080103:/bin/bash
s080104:x:601:600::/home/s080104:/bin/bash
s080105:x:601:600::/home/s080105:/bin/bash
s080106:x:601:600::/home/s080106:/bin/bash
s080107:x:601:600::/home/s080107:/bin/bash
s080108:x:601:600::/home/s080108:/bin/bash
s080109:x:601:600::/home/s080109:/bin/bash
s080110:x:601:600::/home/s080110:/bin/bash
s080111:x:601:600::/home/s080111:/bin/bash
s080112:x:601:600::/home/s080112:/bin/bash
s080113:x:601:600::/home/s080113:/bin/bash
s080114:x:601:600::/home/s080114:/bin/bash
s080115:x:601:600::/home/s080115:/bin/bash
s080116:x:601:600::/home/s080116:/bin/bash
s080117:x:601:600::/home/s080117:/bin/bash
s080118:x:601:600::/home/s080118:/bin/bash
s080119:x:601:600::/home/s080119:/bin/bash
s080120:x:601:600::/home/s080120:/bin/bash
```

- 6) 输入命令“vi stu-passwd”，新建用户口令文件。
- 7) 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。
- 8) 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。

```
s080101:111111
s080102:111111
s080103:111111
s080104:111111
s080105:111111
s080106:111111
s080107:111111
s080108:111111
s080109:111111
s080110:111111
s080111:111111
s080112:111111
s080113:111111
s080114:111111
s080115:111111
s080116:111111
s080117:111111
s080118:111111
s080119:111111
s080120:111111
```

- 9) 输入命令“newusers < student”，批量新建用户帐号。
- 10) 输入命令“pwunconv”，暂时取消 shadow 加密。
- 11) 输入命令“chpasswd <stu-passwd”，批量新建用户的口令。
- 12) 输入命令“pwconv”，进行 shadow 加密，完成批量创建用户帐号工作。

13) 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。

```
s080101:x:601:600::/home/s080101:/bin/bash
s080102:x:601:600::/home/s080101:/bin/bash
s080103:x:601:600::/home/s080101:/bin/bash
s080104:x:601:600::/home/s080101:/bin/bash
s080105:x:601:600::/home/s080101:/bin/bash
s080106:x:601:600::/home/s080101:/bin/bash
s080107:x:601:600::/home/s080101:/bin/bash
s080108:x:601:600::/home/s080101:/bin/bash
s080109:x:601:600::/home/s080101:/bin/bash
s080110:x:601:600::/home/s080101:/bin/bash
s080111:x:601:600::/home/s080101:/bin/bash
s080112:x:601:600::/home/s080101:/bin/bash
s080113:x:601:600::/home/s080101:/bin/bash
s080114:x:601:600::/home/s080101:/bin/bash
s080115:x:601:600::/home/s080101:/bin/bash
s080116:x:601:600::/home/s080101:/bin/bash
s080117:x:601:600::/home/s080101:/bin/bash
s080118:x:601:600::/home/s080101:/bin/bash
s080119:x:601:600::/home/s080101:/bin/bash
s080120:x:601:600::/home/s080101:/bin/bash
```

体会：

本次实验进行了用户与组群的管理，图形化界面和 Shell 字符界面的操作都比较简单好操作，主要是碰到了 vi 不便操作的问题，我将 vim-tiny 卸载换成了 vim-full 以解决了这个问题。

实验题目：SHELL 编程

姓名：黄晨航 学号：19120178 实验日期：2021.11.9

实验环境：Linux 环境

实验目的：

18. 掌握 vi 的三种工作方式，熟悉 vi 编辑程序的使用。
19. 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。

操作过程及结果：

- (10) 熟悉 vi 的三种工作方式。熟悉使用各种编辑功能。并思考：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 工作方式的使用时机：
插入方式：进入插入方式，屏幕下方有一行“----Insert----”字样表示。需要输入文本的内容时用到插入方式，可以用退格键来纠正错误。在插入方式中按一下 ESC 即退出插入方式，进入转义命令方式。

转义命令方式：刚进入 vi 或退出插入方式，即为转移命令方式。这时键入的任何字符转义为特殊功能，如：移动、删除、替换等。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，而且输入命令不需要按 ENTER。

末行命令方式：在转义命令方式中，按冒号“:”进入末行命令方式。屏幕最末一行的行首显示冒号作为命令提示。命令行输入后按 ENTER 开始执行。此时用户可进行文件的全局操作，如：全局查找、替换、文件读、写等。

- ② 用什么命令进入插入方式：
- | 用户输入命令 | 功能描述 |
|--------------|-----------------------|
| i text <ESC> | 在光标前插入新文本，ENTER 可重启一行 |
| I text <ESC> | 在当前行起始处插入新文本 |
| a text <ESC> | 在光标后输入新的文本 |
| A text<ESC> | 在当前行末尾输入新的文本 |
| o text <ESC> | 在当前行下产生新的一行并输入文本 |
| O text<ESC> | 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：
在插入方式中按一下<ESC>，即退出插入方式，进入转义命令方式。

④ 文件怎样存盘; 存盘命令	功能描述
<code>:w</code>	命令 w 将文件以当前名字存入磁盘，并覆盖了文件先前的副本。但 w 命令不影响缓冲区的内容。
<code>:w newfile</code>	将文件以不同名字保存，例如： newfile 。如果不存在名为 newfile 的文件， vi 编辑器将存入文件并给出文件的大小。如果这个文件已经存在，那么 vi 会通知你，但不刷新文件的内容。
<code>:w ! newfile</code>	“!”放弃了标准的 vi 防止覆盖文件的保护功能，强制刷新文件内容。

- (11) 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。

```
if [ $# == 0 ]
then
    echo "Name not provided"
else
    echo "Your name is " $1
fi
```

运行结果：

```
hch@hch-virtual-machine:~/桌面/e3/EX06/02$ chmod 755 2.sh
hch@hch-virtual-machine:~/桌面/e3/EX06/02$ ./2.sh
"Name not provided"
```

- (12) 用 **vi** 编写《实验指导》“第四部分 Shell 程序设计”中的例 1， 练习内部变量和位置参数的用法。

```
if [ $# == 0 ]
then
    echo "Name not provided"
else
    echo "Your name is " $1
fi
```

运行结果：

```
hch@hch-virtual-machine:~/桌面/e3/EX06/03$ chmod 755 3.sh
hch@hch-virtual-machine:~/桌面/e3/EX06/03$ ./3.sh
Name not provided
hch@hch-virtual-machine:~/桌面/e3/EX06/03$ ./3.sh Theodore
Your name is Theodore
```

- (13) 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。

```
#!/bin/bash
#Name Display program
if [ $# == 0 ]; then
    echo "未输入参数"
else
    echo $0 "有" $# "个参数"
    for i in $*; do
        echo $i
    done
fi
```

运行结果:

```
hch@hch-virtual-machine:~/桌面/e3/EX06/04$ chmod 755 4.sh
hch@hch-virtual-machine:~/桌面/e3/EX06/04$ ./4.sh
未输入参数
hch@hch-virtual-machine:~/桌面/e3/EX06/04$ ./4.sh Theodore
./4.sh 有 1 个参数
Theodore
```

- (14) 修改例 1 程序, 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。

```
#!/bin/bash
#Name Display program
echo input text:
read parameter
if [ "$parameter" = "" ]; then
    echo 未输入文本
else
    echo 输入文本为: $parameter
fi
```

运行结果:

```
hch@hch-virtual-machine:~/桌面/e3/EX06/05$ chmod 755 5.sh
hch@hch-virtual-machine:~/桌面/e3/EX06/05$ ./5.sh
input text:

未输入文本
hch@hch-virtual-machine:~/桌面/e3/EX06/05$ ./5.sh
input text:
asdfghjkl
输入文本为: asdfghjkl
```

- (15) 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。

例 2:

```
string1="The First one"
string2="The second one"

if [ "$string1" = "$string2" ]
then
    echo string1 equal to string2
else
    echo "string1 not equal to string2"
fi

if [ string1 ]
then
    echo "string1 is not empty"
else
    echo "string1 is empty"
fi

if [ -n string2 ]
then
    echo "string2 has a length greater than zero"
else
    echo "string2 has a length equal to zero"
fi
```

运行结果:

```
hch@hch-virtual-machine:~/桌面/e3/EX06/06$ chmod 755 example02.sh
hch@hch-virtual-machine:~/桌面/e3/EX06/06$ ./example02.sh
"string1 not equal to string2"
"string1 is not empty"
"string2 has a length greater than zero"
```

例 3:

```
if [ -d cppdir ]
then
    echo "cppdir is a directory"
else
    echo "cppdir is not a directory"
fi

if [ -f filea ]
then
    echo "filea is a regular file"
else
    echo "filea is not a regular file"
fi

if [ -r filea ]
then
    echo "filea has read permissione"
else
    echo "filea dose not have read permissione"
fi

if [ -w filea ]
then
```

运行结果:

```
hch@hch-virtual-machine:~/桌面/e3/EX06/06$ chmod 755 example03.sh
hch@hch-virtual-machine:~/桌面/e3/EX06/06$ ./example03.sh
"cppdir is a directory"
"filea is a regular file"
"filea has read permissione"
"filea has write permissione"
"cppdir has execute permissione"
```

(16) 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。


```

echo please input string1
read string1
echo please input string2
read string2

if [ "$string1" = "$string2" ]
then
    echo string1 equal to string2
else
    echo "string1 not equal to string2"
fi

if [ $string1 ]
then
    echo "string1 is not empty"
else
    echo "string1 is empty"
fi

if [ -n "$string2" ]
then
    echo "string2 has a length greater than zero"
else
    echo "string2 has a length equal to zero"
fi

```

运行结果:

```

hch@hch-virtual-machine:~/桌面/e3/EX06/07$ chmod 755 7.sh
hch@hch-virtual-machine:~/桌面/e3/EX06/07$ ./7.sh
please input string1
abcdefg
please input string2
asdfgh
"string1 not equal to string2"
"string1 is not empty"
"string2 has a length greater than zero"

```

(17) 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。

```
#!/bin/bash
while ((1)); do
    echo -e "请输入文件名: \c"
    read file
    if test -e $file; then
        if test -d $file; then
            echo "$file是一个目录"
        else
            echo -e "$file\c"
            if test -r $file; then
                echo -e "可读\c"
            else
                echo -e "不可读\c"
            fi
            if test -w $file; then
                echo -e "可写\c"
            else
                echo -e "不可写\c"
            fi
            if test -x $file; then
                echo -e "可执行\c"
            else
                echo -e "不可执行\c"
            fi
            echo ""
        fi
    else
        echo "No such file" $file
    fi
done
```

运行结果:

```
hch@hch-virtual-machine:~/桌面/e3/EX06/08$ chmod 755 prog8.sh
hch@hch-virtual-machine:~/桌面/e3/EX06/08$ ./prog8.sh
请输入文件名: r
r可读可写不可执行
请输入文件名: rw
rw可读可写不可执行
请输入文件名: rwx
rwx可读可写不可执行
请输入文件名: rx
rx可读可写不可执行
请输入文件名: rrrr
No such file rrrr
请输入文件名:
```

- (18) 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。

例 4:

```

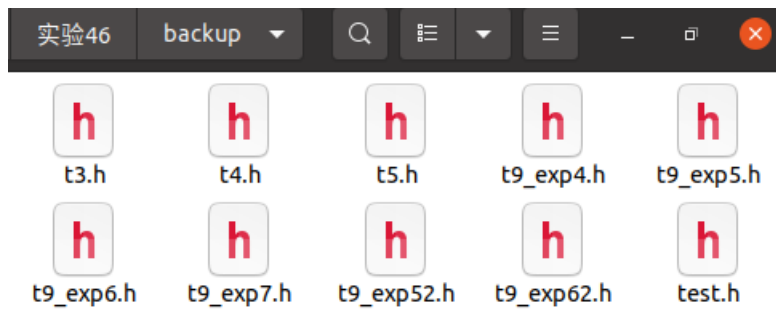
for filename in `ls`
do
    if [ "$filename" != "backup" ]
    then
        cp $filename backup/$filename
        if [ $? -ne 0 ]
        then
            echo "copy $filename failed"
        fi
    fi
done

```

```

hch@hch-virtual-machine:~/桌面/e3/实验46$ chmod 755 t9_exp4.h
hch@hch-virtual-machine:~/桌面/e3/实验46$ ./t9_exp4.h

```



可以看到文件夹中文件被复制到 backup 文件夹中。

例 5:

```

1 loopcount=0
2 result=0
3 while [ $loopcount -lt 10 ]
4 do
5     let loopcount++
6     let result=result+$loopcount*2
7 done
8 echo "result is $result"

```

运行结果:

```

hch@hch-virtual-machine:~/桌面/e3/实验46$ chmod 755 t9_exp5.h
hch@hch-virtual-machine:~/桌面/e3/实验46$ ./t9_exp5.h
result is 110

```

例 6:

```

1 loopcount=0
2 result=0
3 until [ $loopcount -ge 10 ]
4 do
5     let loopcount++
6     let result=result+$loopcount*2
7 done
8 echo "result is $result"

```

运行结果:

```

hch@hch-virtual-machine:~/桌面/e3/实验46$ chmod 755 t9_exp6.h
hch@hch-virtual-machine:~/桌面/e3/实验46$ ./t9_exp6.h
result is 110

```

例 7:

```
1 select item in continue finish
2 do
3     if [ $item = "finish" ]
4     then
5         break
6     fi
7 done
```

运行结果:

```
hch@hch-virtual-machine:~/桌面/e3/实验46$ chmod 755 t9_exp7.h
hch@hch-virtual-machine:~/桌面/e3/实验46$ ./t9_exp7.h
1) continue
2) finish
#? 1
#? 2
```

(19) 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。

例 8:

```
case $1 in
01|1) echo "Month is January";;
02|2) echo "Month is February";;
03|3) echo "Month is March";;
04|4) echo "Month is April";;
05|5) echo "Month is May";;
06|6) echo "Month is June";;
07|7) echo "Month is July";;
08|8) echo "Month is August";;
09|9) echo "Month is September";;
10|10) echo "Month is October";;
11|11) echo "Month is November";;
12|12) echo "Month is December";;
*) echo "Invalid parameter";;
esac
```

运行结果:

```

hch@hch-virtual-machine:~/桌面/e3/EX06/10$ chmod 755 ex08.sh
hch@hch-virtual-machine:~/桌面/e3/EX06/10$ ./ex08.sh
"Invalid parameter"
hch@hch-virtual-machine:~/桌面/e3/EX06/10$ ./ex08.sh 1
"Month is January"
hch@hch-virtual-machine:~/桌面/e3/EX06/10$ ./ex08.sh 2
"Month is February"
hch@hch-virtual-machine:~/桌面/e3/EX06/10$ ./ex08.sh 3
"Month is March"
hch@hch-virtual-machine:~/桌面/e3/EX06/10$ ./ex08.sh 4
"Month is April"
hch@hch-virtual-machine:~/桌面/e3/EX06/10$ ./ex08.sh 5
"Month is May"
hch@hch-virtual-machine:~/桌面/e3/EX06/10$ ./ex08.sh 6
"Month is June"
hch@hch-virtual-machine:~/桌面/e3/EX06/10$ ./ex08.sh 7
"Month is July"
hch@hch-virtual-machine:~/桌面/e3/EX06/10$ ./ex08.sh 8
"Month is August"
hch@hch-virtual-machine:~/桌面/e3/EX06/10$ ./ex08.sh 9
"Month is September"
hch@hch-virtual-machine:~/桌面/e3/EX06/10$ ./ex08.sh 10
"Month is October"

```

例 9:

```

displaymonth ( ) {
case $1 in
01|1) echo "Month is January";;
02|2) echo "Month is February";;
03|3) echo "Month is March";;
04|4) echo "Month is April";;
05|5) echo "Month is May";;
06|6) echo "Month is June";;
07|7) echo "Month is July";;
08|8) echo "Month is August";;
09|9) echo "Month is September";;
10|10) echo "Month is October";;
11|11) echo "Month is November";;
12|12) echo "Month is December";;
*) echo "Invalid parameter";;
esac
}
displaymonth 8 #调用函数
displaymonth 12

```

运行结果:

```

hch@hch-virtual-machine:~/桌面/e3/EX06/10$ chmod 755 ex09.sh
hch@hch-virtual-machine:~/桌面/e3/EX06/10$ ./ex09.sh
"Month is August"
"Month is December"

```

- (20) 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。

```

echo "home is :$HOME"
echo "path is :$PATH"
if [ $# -ne 0 ]
then
    ls -ld $1
fi

```

运行结果:

```
hch@hch-virtual-machine:~/桌面/e3$ chmod 755 prog11.sh
hch@hch-virtual-machine:~/桌面/e3$ ./prog11.sh
home is :/home/hch
path is :/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr
/games:/usr/local/games:/snap/bin
./prog11.sh: 行 7: /home/hch: 是一个目录
```

体会:

本次实验是对实验四的进阶,通过使用 vi 进行编程,让我对 Linux 的 SHELL 编程指令更加熟悉,有了更充分的了解,一开始的编译过程很坎坷,一直报错,经过了大量时间的试错才顺利完成本次实验。