

# 实验题目: Linux 操作系统基本命令

姓名: 邢志昂

学号: 19122110

实验日期: 2021 年 10 月 7 日

## Linux 操作系统基本命令

### 实验环境:

Centos8

### 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, 管道命令等命令的操作要点。

### 操作过程

#### 1. man 命令和-help 命令

man 命令的使用见图 1, -help 命令的使用见图 2。

```
[root@VM-0-3-centos ~]# man rm
rm(1)                                User Commands                                rm(1)

NAME
  rm - remove files or directories

SYNOPSIS
  rm [OPTION]... FILE...

DESCRIPTION
  This manual page documents the GNU version of rm. rm removes each specified file. By default, it does not remove directories.

  If the -i or --interactive option is given, and there are more than three files or the -r, -B, or --recursive are given, then rm prompts the user for whether to proceed with the entire operation. If the response is not affirmative, the entire command is aborted.

  Otherwise, if a file is unwritable, standard input is a terminal, and the -f or --force option is not given, or the -i or --interactive option is given, rm prompts the user for whether to remove the file. If the response is not affirmative, the file is skipped.

OPTIONS
  Remove (unlink) the FILE(s).
  -f, --force          ignore nonexistent files and arguments, never prompt
  -i                  prompt before every removal
  -I                  prompt once before removing more than three files, or
                    when recursing; less intrusive than -i,
                    while still giving protection against most mistakes
  --interactive[=MODE] prompt according to MODE: never, once (-i), or
                    always (-I); without MODE, prompt always
  --one-file-system   when removing a hierarchy recursively, skip any
                    directory that is on a file system different from
                    that of the corresponding command line argument
  --no-preserve-root  do not treat '/' specially
  --preserve-root     do not remove '/' (default)
  -r, -R, --recursive remove directories and their contents recursively
  -d, --dir           remove empty directories
  -v, --verbose       explain what is being done
  --help             display this help and exit
  --version          output version information and exit

By default, rm does not remove directories. Use the --recursive (-r or -R)
option to remove each listed directory, too, along with all of its contents.

To remove a file whose name starts with a '.', for example './foo',
use one of these commands:
  rm -- ./foo
  rm ./foo
```

图 1: man 命令

```
[root@VM-0-3-centos ~]# rm -help
Usage: rm [OPTION]... FILE...
Remove (unlink) the FILE(s).
  -f, --force          ignore nonexistent files and arguments, never prompt
  -i                  prompt before every removal
  -I                  prompt once before removing more than three files, or
                    when recursing; less intrusive than -i,
                    while still giving protection against most mistakes
  --interactive[=MODE] prompt according to MODE: never, once (-i), or
                    always (-I); without MODE, prompt always
  --one-file-system   when removing a hierarchy recursively, skip any
                    directory that is on a file system different from
                    that of the corresponding command line argument
  --no-preserve-root  do not treat '/' specially
  --preserve-root     do not remove '/' (default)
  -r, -R, --recursive remove directories and their contents recursively
  -d, --dir           remove empty directories
  -v, --verbose       explain what is being done
  --help             display this help and exit
  --version          output version information and exit

By default, rm does not remove directories. Use the --recursive (-r or -R)
option to remove each listed directory, too, along with all of its contents.

To remove a file whose name starts with a '.', for example './foo',
use one of these commands:
  rm -- ./foo
  rm ./foo
```

图 2: -help 命令



```
[root@VM-0-3-centos lab]# cat>test.txt
this is a test for cat
^C
[root@VM-0-3-centos lab]# cat test.txt
this is a test for cat
[root@VM-0-3-centos lab]# ln test.txt test2.dat
[root@VM-0-3-centos lab]# cat test2.dat
this is a test for cat
[root@VM-0-3-centos lab]# ls -l test?.*
-rw-r--r-- 2 root root 23 Oct  9 10:24 test2.dat
```

图 6: cat 和 ln 命令

文件链接的作用: 当我们需要在不同的目录, 用到相同的文件时, 我们不需要在每一个需要的目录下都放一个必须相同的文件, 我们只要在某个固定的目录, 放上该文件, 然后在其它的目录下用 ln 命令链接 (link) 它就可以, 不必重复的占用磁盘空间。

## 5. cd,ls,mkdir,rmdir 命令

cd,ls,mkdir,rmdir 命令的使用见图 7

```
[root@VM-0-3-centos bin]# cd ~
[root@VM-0-3-centos ~]# ls
init-test-data.sql  mysql80-community-release-el7-3.noarch.rpm  rpm
[root@VM-0-3-centos ~]# ll
total 32
-rw-r--r-- 1 root root 1900 Sep  8 10:40 init-test-data.sql
-rw-r--r-- 1 root root 26024 Apr 25  2019 mysql80-community-release-el7-3.noarch.rpm
-rw-r--r-- 1 root root  0 Sep  7 22:04 rpm
[root@VM-0-3-centos ~]# cd /home/
[root@VM-0-3-centos home]# ls
lab
[root@VM-0-3-centos home]# mkdir test
[root@VM-0-3-centos home]# ls
lab  test
[root@VM-0-3-centos home]# rmdir test/
[root@VM-0-3-centos home]# ls
lab
```

图 7: cd,ls 等命令

linux 下的文件类型有如下七种:

### (1) 普通文件类型

Linux 中最多的一种文件类型, 包括纯文本文件 (ASCII); 二进制文件 (binary); 数据格式的文件 (data); 各种压缩文件. 第一个属性为 [-]

### (2) 目录文件

就是目录, 能用 cd 命令进入的。第一个属性为 [d], 例如 [drwxrwxrwx]

### (3) 块设备文件

就是存储数据以供系统存取的接口设备, 简单而言就是硬盘。例如一号硬盘的代码是 /dev/hda1 等文件。第一个属性为 [b]

### (4) 字符设备

字符设备文件: 即串行端口的接口设备, 例如键盘、鼠标等等。第一个属性为 [c]

## (5) 套接字文件

这类文件通常用在网络数据连接。可以启动一个程序来监听客户端的要求, 客户端就可以通过套接字来进行数据通信。第一个属性为 [s], 最常在 /var/run 目录中看到这种文件类型

## (6) 管道文件

FIFO 也是一种特殊的文件类型, 它主要的目的是, 解决多个程序同时存取一个文件所造成的错误。FIFO 是 first-in-first-out(先进先出) 的缩写。第一个属性为 [p]

## (7) 链接文件

类似 Windows 下面的快捷方式。第一个属性为 [l], 例如 [lrwxrwxrwx]

Linux 的文件, 有个 16 位的字来表示文件的类型和属性信息, 其中 4 位表示文件的类型信息, 剩下的 12 位表示文件的模式。在传统的 unix 和 linux 文件系统模型中, 每个文件都有一个 9 个权限位用来控制谁能够读写和执行该文件内容, 还有一个 3 个权限位来影响可执行程序运行, 这 12 个位就共同构成了该文件的“模式”。

## 6. chmod 和 chown 命令

chmod 和 chown 命令的使用见图 8

```
[root@VM-0-3-centos lab]# ll
total 4
-rw-r--r-- 1 root root 23 Oct  9 10:24 test.txt
[root@VM-0-3-centos lab]# chmod 751 test.txt
[root@VM-0-3-centos lab]# ll
total 4
-rwxr-x--x 1 root root 23 Oct  9 10:24 test.txt
[root@VM-0-3-centos lab]# useradd -m violet
[root@VM-0-3-centos lab]# passwd violet
Changing password for user violet.
New password:
BAD PASSWORD: The password contains the user name in some form
Retype new password:
Sorry, passwords do not match.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
[root@VM-0-3-centos lab]# chown violet test.txt
[root@VM-0-3-centos lab]# ll
total 4
-rwxr-x--x 1 violet root 23 Oct  9 10:24 test.txt
```

图 8: chmod 和 chown 等命令

修改权限后再如果用其他用户不能修改该文件, 因为其他用户仅有执行权限。

## 7. ps 等命令

ps 命令的使用如图 9

```
[root@VM-0-3-centos lab]# ps -ef
UID          PID  PPID  C  SIME  TTY          TIME CMD
root           1      0  0  Sep07  ?        00:04:29 /usr/lib/systemd/systemd
root           2      0  0  Sep07  ?        00:00:00 [kthreadd]
root           4      2  0  Sep07  ?        00:00:00 [kworker/0:0]
root           6      2  0  Sep07  ?        00:00:00 [kworker/0:0]
root           7      2  0  Sep07  ?        00:00:00 [migration/0]
root           8      2  0  Sep07  ?        00:00:00 [rcu_bh]
root           9      2  0  Sep07  ?        00:01:33 [rcu_sched]
root          10      2  0  Sep07  ?        00:00:00 [lru-add-drain]
root          11      2  0  Sep07  ?        00:00:00 [watchdog/0]
root          13      2  0  Sep07  ?        00:00:00 [kswapd0]
root          14      2  0  Sep07  ?        00:00:00 [netns]
root          15      2  0  Sep07  ?        00:00:00 [khungtaskd]
root          16      2  0  Sep07  ?        00:00:00 [writeback]
root          17      2  0  Sep07  ?        00:00:00 [kintegrityd]
root          18      2  0  Sep07  ?        00:00:00 [bioset]
root          19      2  0  Sep07  ?        00:00:00 [bioset]
root          20      2  0  Sep07  ?        00:00:00 [bioset]
root          21      2  0  Sep07  ?        00:00:00 [kblockd]
root          22      2  0  Sep07  ?        00:00:00 [kfs]
root          23      2  0  Sep07  ?        00:00:00 [ndac-poller]
root          24      2  0  Sep07  ?        00:00:00 [watchdogd]
root          30      2  0  Sep07  ?        00:00:00 [kswapd1]
root          31      2  0  Sep07  ?        00:00:00 [kswapd]
root          32      2  0  Sep07  ?        00:00:02 [khugetaged]
root          33      2  0  Sep07  ?        00:00:00 [cryptd]
root          41      2  0  Sep07  ?        00:00:00 [kthrottld]
root          43      2  0  Sep07  ?        00:00:00 [nmpath_rdacd]
root          44      2  0  Sep07  ?        00:00:00 [kblockd]
root          45      2  0  Sep07  ?        00:00:00 [kswapd1]
```

图 9: ps 等命令



bin 是 Binaries (二进制文件) 的缩写, 这个目录存放着最经常使用的命令。

/boot:

这里存放的是启动 Linux 时使用的一些核心文件, 包括一些连接文件以及镜像文件。

/dev :

dev 是 Device(设备) 的缩写, 该目录下存放的是 Linux 的外部设备, 在 Linux 中访问设备的方式和访问文件的方式是相同的。

/etc:

etc 是 Etcetera(等等) 的缩写, 这个目录用来存放所有的系统管理所需要的配置文件和子目录。

/home:

用户的主目录, 在 Linux 中, 每个用户都有一个自己的目录, 一般该目录名是以用户的账号命名的, 如上图中的 alice、bob 和 eve。

/lib:

lib 是 Library(库) 的缩写这个目录里存放着系统最基本的动态连接共享库, 其作用类似于 Windows 里的 DLL 文件。几乎所有的应用程序都需要用到这些共享库。

/lost+found:

这个目录一般情况下是空的, 当系统非法关机后, 这里就存放了一些文件。

/media:

linux 系统会自动识别一些设备, 例如 U 盘、光驱等等, 当识别后, Linux 会把识别的设备挂载到这个目录下。

/mnt:

系统提供该目录是为了让用户临时挂载别的文件系统的, 我们可以将光驱挂载在 /mnt/ 上, 然后进入该目录就可以查看光驱里的内容了。

/opt:

opt 是 optional(可选) 的缩写, 这是给主机额外安装软件所摆放的目录。比如你安装一个 ORACLE 数据库则就可以放到这个目录下。默认是空的。

/proc:

proc 是 Processes(进程) 的缩写, /proc 是一种伪文件系统 (也即虚拟文件系统), 存储的是当前内核运行状态的一系列特殊文件, 这个目录是一个虚拟的目录, 它是系统内存的映射, 我们可以通过直接访问这个目录来获取系统信息。这个目录的内容不在硬盘上而是在内存里, 我们也可以直接修改里面的某些文件, 比如可以通过下面的命令来屏蔽主机的 ping 命令, 使别人无法 ping 你的机器:

/sbin:

s 就是 Super User 的意思, 是 Superuser Binaries (超级用户的二进制文件) 的缩写, 这里存放的是系统管理员使用的系统管理程序。

/srv:

该目录存放一些服务启动之后需要提取的数据。

/tmp:

tmp 是 temporary(临时) 的缩写这个目录是用来存放一些临时文件的。

/usr:

usr 是 unix shared resources(共享资源) 的缩写, 这是一个非常重要的目录, 用户的很多应用程序和文件都放在这个目录下, 类似于 windows 下的 program files 目录。

/usr/bin:

系统用户使用的应用程序。

/usr/sbin:

超级用户使用的比较高级的管理程序和系统守护程序。

/usr/src:

内核源代码默认的放置目录。

/var:

var 是 variable(变量) 的缩写, 这个目录中存放着在不断扩充着的东西, 我们习惯将那些经常被修改的目录放在这个目录下。包括各种日志文件。

/run:

是一个临时文件系统, 存储系统启动以来的信息。当系统重启时, 这个目录下的文件应该被删掉或清除。如果你的系统上有 /var/run 目录, 应该让它指向 run。

8. Shell 是系统的用户界面, 提供了用户与内核进行交互操作的一种接口。它接收用户输入的命令并把它送入内核去执行。Linux 中的 shell 有多种类型, 其中最常用的几种是 Bourne shell (sh)、C shell (csh) 和 Korn shell (ksh)。三种 shell 各有优缺点。Bourne shell 是 UN 最初使用的 shell, 并且在每种 UN 上都可以使用。Bourne shell 在 shell 编程方面相当优秀, 但在处理与用户的交互方面做得不如其他几种 shell。Linux 操作系统缺省的 shell 是 Bourne Again shell, 它是 Bourne shell 的扩展, 简称 Bash, 与 Bourne shell 完全向后兼容, 并且在 Bourne shell 的基础上增加、增强了很多特性。Bash 放在 /bin/bash 中, 它有许多特色, 可以提供如命令补全、命令编辑和命令历史表等功能, 它还包含了很多 C shell 和 Korn shell 中的优点, 有灵活和强大的编程接口, 同时又有很友好的用户界面。

9.

- (1) 压缩文件
- (2) 目录文件
- (3) 二进制文件
- (4) 块文件
- (5) 文本文件

- (6) 字符文件
- (7) 目录文件
- (8) 系统文件



# 实验题目：用户界面与 Shell 命令

姓名：邢志昂

学号：19122110

实验日期:2021 年 10 月 14 日

## 用户界面与 Shell 命令

### 实验环境:

ubuntu server 20.02 on arm

由于本人使用的是 ubuntu server, 该系统是不带有图形化界面的, 所以本次实验一些操作可能无法完成。

### 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。

### 操作过程

#### 1. 图形化用户界面下的 Shell 命令操作

日期操作

- (1) 打开 shell
- (2) 输入 date
- (3) 输入 date 091700002011 更改系统时间
- (4) 输入 date 查看更新后的系统时间, 如图 1
- (5) 切换至普通用户
- (6) 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, 如图 2

文件操作

- (1) 输入 ls - -help 命令
- (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. 如图 3
- (3) 输入 cd /etc 进入/etc 目录
- (4) 输入 ls -al 显示/etc 目录下所有文件和子目录的详细信息。如图 4

```
[root@root ~]# date
Mon Oct 25 10:27:50 CST 2021
[root@root ~]# date 091700002011
Sat Sep 17 00:00:00 CST 2011
[root@root ~]# date
Sat Sep 17 00:00:01 CST 2011
```

图 1: 将系统时间修改为 2011 年 9 月 17 日零点

```
[root@root ~]# su violet
[violet@root root]$ cal 2011

                2011

   January           February           March
Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa
                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

   April             May              June
Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa
                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

   July             August           September
Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa
                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

   October          November          December
Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa
                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
```

图 2: 查看 2011 年 9 月 17 日是星期几

```

-s, --size                print the allocated size of each file, in blocks
-S                        sort by file size, largest first
--sort=WORD               sort by WORD instead of name: none (-U), size (-S),
                           time (-t), version (-v), extension (-X)
--time=WORD               with -l, show time as WORD instead of default
                           modification time: atime or access or use (-u);
                           ctime or status (-c); also use specified time
                           as sort key if --sort=time (newest first)
--time-style=TIME_STYLE  time/date format with -l; see TIME_STYLE below
-t                        sort by modification time, newest first
-T, --tabsize=COLS        assume tab stops at each COLS instead of 8
-u                        with -lt: sort by, and show, access time;
                           with -l: show access time and sort by name;

```

图 3: 查看 ls 命令的-s 选项的帮助信息

```

root@root:~# cd /etc
root@root:/etc# ls -al
total 1280
drwxr-xr-x 157 root root    12288 Oct 22 06:14 .
drwxr-xr-x  21 root root    4096 Oct 22 14:08 ..
-rw-r--r--   1 root root    3028 Feb  1 2021 adduser.conf
drwxr-xr-x   3 root root    4096 Mar 30 2021 alsa
drwxr-xr-x   2 root root   12288 Oct 19 16:08 alternatives
-rw-r--r--   1 root root    401 Jul 17 2019 anacrontab
-rw-r--r--   1 root root    433 Oct  2 2017 app.config
drwxr-xr-x   5 root root    4096 Mar 30 2021 apm
drwxr-xr-x   3 root root    4096 Feb  1 2021 apparmor
drwxr-xr-x   7 root root    4096 Oct 19 16:07 apparmor.d
drwxr-xr-x   5 root root    4096 Sep 15 13:31 appport
-rw-r--r--   1 root root    769 Jan 19 2020 appstream.conf
drwxr-xr-x   7 root root    4096 Sep 15 20:49 apt
-rw-r-----   1 root daemon  144 Nov 13 2018 at.deny
drwxr-xr-x   3 root root    4096 Jul  9 14:14 avahi
-rw-r--r--   1 root root   2319 Feb 25 2020 bash.bashrc
-rw-r--r--   1 root root    45 Jan 26 2020 bash_completion
drwxr-xr-x   2 root root    4096 Sep 15 13:31 bash_completion.d
-rw-r--r--   1 root root    367 Apr 15 2020 bindresvport.blacklist
drwxr-xr-x   2 root root    4096 Apr 22 2020 binfmt.d
drwxr-xr-x   2 root root    4096 Sep 15 20:59 bluetooth
-rw-r-----   1 root root    33 Mar 30 2021 brlapi.key
drwxr-xr-x   7 root root    4096 Mar 30 2021 brltty
-rw-r--r--   1 root root   26916 Mar  4 2020 brltty.conf
drwxr-xr-x   2 root root    4096 Feb  1 2021 byobu
drwxr-xr-x   3 root root    4096 Feb  1 2021 ca-certificates
-rw-r--r--   1 root root   6570 Sep 24 14:11 ca-certificates.conf
-rw-r--r--   1 root root   6569 Mar 24 2021 ca-certificates.conf.dpkg-old
drwxr-xr-x   2 root root    4096 Feb  1 2021 calendar
drwxr-s---   2 root dip    4096 Mar 30 2021 chatscripts
drwxr-xr-x   4 root root    4096 Oct 18 14:33 cloud
drwx-----   3 root root    4096 May 26 09:57 cni
drwxr-xr-x   2 root root    4096 Feb  1 2021 console-setup
drwxr-xr-x   2 root root    4096 Mar 30 2021 cracklib
drwxr-xr-x   2 root root    4096 Sep 15 20:59 cron.d
drwxr-xr-x   2 root root    4096 Sep 21 16:18 cron.daily

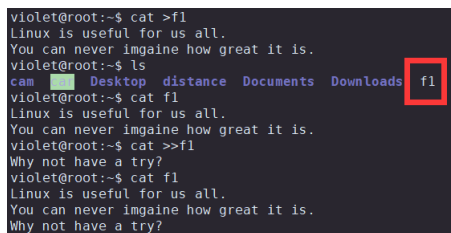
```

图 4: 查看/etc 目录下所有文件和子目录的详细信息

## 2. 字符界面下的 Shell 命令操作

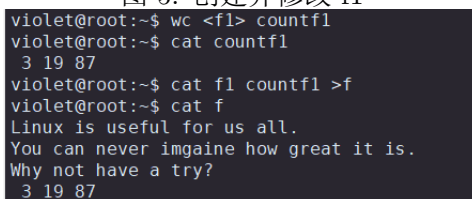
用户登录

- (1) 启动 xshell, 连接服务器
- (2) 输入 pwd 命令, 如图 ??
- (3) 输入命令 “cat >f1”, 输入字符, 按 CTRL+D 键结束输入。
- (4) 输入 ls 查看文件是否生成。
- (5) 输入命令 “cat f1”, 查看 f1 文件的内容。
- (6) 输入命令 “cat »f1”, 屏幕上输入点光标闪烁。
- (7) 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。
- (8) 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, 相关操作如图 5。
- (9) 输入命令 “wc <f1> countf1”, 屏幕上不显示任何信息。
- (10) 输入命令 “cat countf1”, 查看 countf1 文件的内容。
- (11) 输入命令 “cat f1 countf1 >f”, 将两个文件合并为一个文件
- (12) 输入命令 “cat f”, 查看 f 文件的内容, 结果如图 6



```
violet@root:~$ cat >f1
Linux is useful for us all.
You can never imagine how great it is.
violet@root:~$ ls
cam Desktop distance Documents Downloads f1
violet@root:~$ cat f1
Linux is useful for us all.
You can never imagine how great it is.
violet@root:~$ cat »f1
Why not have a try?
violet@root:~$ cat f1
Linux is useful for us all.
You can never imagine how great it is.
Why not have a try?
```

图 5: 创建并修改 f1



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

图 6: 创建并合并 countf1

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
- (2) 浏览过程中按 “q” 键, 可结束分页显示。结果如图 7
- (3) “ls /etc |head -n 5”, 屏幕显示出 “ls /etc” 命令输出结果的前面 5 行。结果如图 8
- (4) 输入命令 “clear”

```
adduser.conf
alsa
alternatives
anacrontab
apg.conf
apm
apparmor
apparmor.d
appport
appstream.conf
apt
at.deny
avahi
bash.bashrc
bash_completion
bash_completion.d
bindresvport.blacklist
binfmt.d
bluetooth
brlapi.key
brltty
brltty.conf
byobu
ca-certificates
ca-certificates.conf
ca-certificates.conf.dpkg-old
calendar
chatscripts
cloud
cni
console-setup
cracklib
cron.d
cron.daily
cron.hourly
cron.monthly
crontab
cron.weekly
cryptsetup-initramfs
crypttab
cups
--More--
```

图 7: 分页显示/etc

```
violet@root:/etc$ ls /etc|head -n5
adduser.conf
alsa
alternatives
anacrontab
apg.conf
```

图 8: 输出结果的前面 5 行

### 3. 通配符的使用

显示/bin/目录中符合要求的文件

- (1) 输入命令 “ls /bin/c\*”
- (2) 输入命令 “ls /bin/c??”
- (3) 输入命令 “ls /bin/[csh]\*”, 结果如图 9
- (4) 输入命令 “ls /bin/[!a-u]\*”
- (5) 输入命令 “!!”, 结果如图 10
- (6) 输入命令 “history 5”, 显示最近执行过的 5 个命令. 结果如图 11

### 4. 设置手工启动图形化用户界面

由于本系统无图形化界面, 所以这里就不在演示。下面给出对于 ubuntu 系统打开和关闭图形界面的操作

- (1) 关闭用户图形界面, 使用 tty 登录。  
sudo systemctl set-default multi-user.target  
sudo reboot
- (2) 开启用户图形界面。  
sudo systemctl set-default graphical.target  
sudo reboot

```

violet@root:~$ ls /bin/c*
/bin/c++          /bin/cd-it8      /bin/chvt        /bin/cmp         /bin/compose     /bin/cpp-8
/bin/c89          /bin/c++filt     /bin/cifsioatat  /bin/codepage    /bin/conch3      /bin/cpp-9
/bin/c89-gcc      /bin/cftp3       /bin/ciptool     /bin/col         /bin/containerd  /bin/crc32
/bin/c99          /bin/chacl       /bin/ckbcomp     /bin/col1        /bin/containerd-shim /bin/c_rehash
/bin/c99-gcc      /bin/chage       /bin/ckeygen3    /bin/col2        /bin/containerd-shim-runc-v1 /bin/crontab
/bin/cal          /bin/chartest3   /bin/cksum       /bin/col3        /bin/containerd-shim-runc-v2 /bin/csplit
/bin/calendar     /bin/chartdetect3 /bin/clang       /bin/col4        /bin/containerd-stress /bin/ctags
/bin/cancel       /bin/chatrr      /bin/clang++     /bin/col5        /bin/convert-dtso0   /bin/ctags-exuberant
/bin/captoinfo    /bin/chcon       /bin/clang++-10  /bin/col6        /bin/corelist       /bin/ctail
/bin/cat          /bin/check-language-support /bin/clang-10   /bin/col7        /bin/count-10       /bin/ctest
/bin/catchsegv    /bin/chff        /bin/clang-cpp-10 /bin/col8        /bin/cp             /bin/ctr
/bin/catman       /bin/chgrp       /bin/clear       /bin/col9        /bin/cpack          /bin/ctstat
/bin/cautious-launcher /bin/chmod      /bin/clear console /bin/colcrt      /bin/cpan           /bin/cupstestppd
/bin/cc           /bin/choom       /bin/cloud-id    /bin/colormgr    /bin/cpan5.30-aarch64-linux-gnu /bin/curl
/bin/cd-create-profile /bin/chown      /bin/cloud-init  /bin/colrm       /bin/cpio           /bin/cut
/bin/cd-fix-profile /bin/chrt       /bin/cloud-init-per /bin/column      /bin/cpio-filter    /bin/cvt
/bin/cd-icdump    /bin/chsh       /bin/cmake       /bin/comm        /bin/cpp            /bin/cvtsudoers
violet@root:~$ ls /bin/c??
/bin/c++ /bin/c89 /bin/c99 /bin/cal /bin/cat /bin/cmp /bin/col /bin/cpp /bin/ctr /bin/cut /bin/cvt
violet@root:~$ ls /bin/[csh]*
/bin/c++          /bin/corelist      /bin/hydra_nameserver /bin/sg_map26     /bin/soelim
/bin/c89          /bin/count-10      /bin/hydra_persist   /bin/sgm_dd       /bin/soffice
/bin/c89-gcc      /bin/cp            /bin/hydra_pmi_proxy /bin/sg_modes     /bin/software-properties-gtk
/bin/c99          /bin/cpack         /bin/sadf            /bin/sg_opcodes   /bin/sort
/bin/c99-gcc      /bin/cpan          /bin/sane-find-scanner /bin/sgp_dd       /bin/sos
/bin/cal          /bin/cpan5.30-aarch64-linux-gnu /bin/sanstats-10    /bin/sg_persist   /bin/sos-collector
/bin/calendar     /bin/cpio          /bin/sar             /bin/sg_prevent   /bin/sosreport
/bin/cancel       /bin/cpio-filter   /bin/sar.sysstat    /bin/sg_raw       /bin/sotrust
/bin/captoinfo    /bin/cpp           /bin/savetlog       /bin/sg_rbuf      /bin/speaker-test
/bin/cat          /bin/cpp-8         /bin/sbattach       /bin/sg_rdcac     /bin/speedtest
/bin/catchsegv    /bin/cpp-9         /bin/sbkeysync      /bin/sg_read      /bin/speedtest-cli
/bin/catman       /bin/crc32         /bin/sbsiglist      /bin/sg_read_attr /bin/splain
/bin/cautious-launcher /bin/c_rehash     /bin/sbsign         /bin/sg_read_block_limits /bin/split
/bin/cc           /bin/crontab      /bin/sbvarsign      /bin/sg_read_buffer /bin/splitfont
/bin/cd-create-profile /bin/csplit       /bin/sbverify       /bin/sg_readcap   /bin/sproff
/bin/cd-fix-profile /bin/ctags        /bin/scanimimage    /bin/sg_read_long /bin/ss
/bin/cd-icdump    /bin/ctags-exuberant /bin/scp            /bin/sg_reassign  /bin/ssh
/bin/cd-it8       /bin/ctail        /bin/scp-dbus-service /bin/sg_referrals /bin/ssh-add
/bin/c++filt      /bin/ctest        /bin/screen         /bin/sg_rep_zones /bin/ssh-agent
/bin/cftp3        /bin/ctr          /bin/screendump     /bin/sg_requests  /bin/ssh-argv0
/bin/chacl        /bin/ctstat       /bin/screenfetch    /bin/sg_reset     /bin/ssh-copy-id

```

图 9: 显示/bin/目录中符合要求的文件

```

ls /bin/[!a-u]*
/bin/!          /bin/wpa_passphrase /bin/xfce4-keyboard-settings /bin/xkbwatch     /bin/Xwayland
/bin/2to3-2.7   /bin/w.procps       /bin/xfce4-mime-settings     /bin/xkeystone    /bin/xwd
/bin/FileCheck-10 /bin/write         /bin/xfce4-mouse-settings   /bin/xkill        /bin/x-window-manager
/bin/GET        /bin/wsdump        /bin/xfce4-notified-config  /bin/xload        /bin/xwininfo
/bin/HEAD       /bin/X             /bin/xfce4-power-manager    /bin/xlogo        /bin/xwwd
/bin/NF         /bin/xl1perf       /bin/xfce4-power-manager-settings /bin/xlsatoms     /bin/x-www-browser
/bin/POST       /bin/xl1perfcomp   /bin/xfce4-screensaver      /bin/xslclients   /bin/xxd
/bin/vcs-run    /bin/xargs         /bin/xfce4-screensaver-command /bin/xlsfonts     /bin/xz
/bin/vdir       /bin/xauth         /bin/xfce4-screensaver-configure /bin/xmag         /bin/xzcat
/bin/verify-uselistorder-10 /bin/xbiff        /bin/xfce4-screensaver-preferences /bin/xman         /bin/xzcmp
/bin/vi         /bin/xcalc         /bin/xfce4-session         /bin/xmessage     /bin/xzdiff
/bin/view       /bin/xclipboard    /bin/xfce4-session-logout   /bin/xmllint      /bin/xzegrep
/bin/viewres    /bin/xclock        /bin/xfce4-session-settings /bin/xmodmap      /bin/xzfgrep
/bin/viopp      /bin/xcmsdb        /bin/xfce4-settings-editor  /bin/xmore        /bin/xzgrep
/bin/vim        /bin/xconsole     /bin/xfce4-settings-manager /bin/xmrm         /bin/xzless
/bin/vim.basic  /bin/xcursorgen    /bin/xfce4-terminal        /bin/Xorg         /bin/xzmore
/bin/vimdiff    /bin/xcutsel       /bin/xfce4-terminal.wrapper /bin/xprop        /bin/yaml2obj-10
/bin/vim.tiny   /bin/xdg-dbus-proxy /bin/xfconf-query          /bin/xrandr       /bin/yaml-bench-10
/bin/vimtutor   /bin/xdg-desktop-icon /bin/xfd                 /bin/xrdb         /bin/yelp
/bin/vmstat     /bin/xdg-desktop-menu /bin/xfdesktop           /bin/xrdb-dis     /bin/yes
/bin/vncconnect /bin/xdg-email     /bin/xfdesktop-settings   /bin/xrdp-genkeymap /bin/ypdomainname
/bin/vncpasswd  /bin/xdg-icon-resource /bin/xflock4             /bin/xrdp-keygen  /bin/zcat
/bin/vncserver  /bin/xdg-mime      /bin/xfonstsel           /bin/xrdp-sesadmin /bin/zcmp
/bin/volname    /bin/xdg-open      /bin/xfsettingsd         /bin/xrdp-sesrun  /bin/zdiff
/bin/w          /bin/xdg-screensaver /bin/xfwm4              /bin/xrefresh     /bin/zdump
/bin/wall       /bin/xdg-settings  /bin/xfwm4-settings      /bin/x-session-manager /bin/zegrep
/bin/watch      /bin/xdg-user-dir   /bin/xfwm4-tweaks-settings /bin/xset         /bin/zenity
/bin/watchgnupg /bin/xdg-user-dirs-update /bin/xfwm4-workspace-settings /bin/xsetmode     /bin/zfgrep
/bin/wc         /bin/xditview      /bin/xgamma             /bin/xsetpointer  /bin/zforce
/bin/wdctl      /bin/xdpyinfo      /bin/xgc                /bin/xsetroot     /bin/zgrep
/bin/wget       /bin/xdriinfo      /bin/xgettext           /bin/xsetwacom    /bin/zip
/bin/whatis     /bin/xedit          /bin/xhost              /bin/xsm          /bin/zipcloak
/bin/whereis    /bin/Xephyr        /bin/xicc               /bin/xstdcmap     /bin/zipdetails
/bin/which      /bin/xev           /bin/xinitt             /bin/xsubpp       /bin/zipgrep
/bin/whiptail   /bin/xeyes         /bin/xinput            /bin/xterm        /bin/zipinfo
/bin/who        /bin/xfce4-accessibility-settings /bin/xkbcomp          /bin/x-terminal-emulator /bin/zipnote
/bin/whoami     /bin/xfce4-appearance-settings /bin/xkbcomp          /bin/Xtightvnc     /bin/zipsplit
/bin/whoopsie-preferences /bin/xfce4-color-settings /bin/xkbcomp          /bin/xvidtune      /bin/zless
/bin/wifi-status /bin/xfce4-display-settings /bin/xkbcomp          /bin/xvinfo        /bin/zmore
/bin/word-list-compress /bin/xfce4-find-cursor /bin/xkbcomp          /bin/Xvnc         /bin/znew

```

图 10: 屏幕显示/bin 目录中首字母是 v z 的文件和目录

```

violet@root:~$ history 5
537 clear
538 ls /bin/[!a-u]*
539 ls /bin/[!a-u]*clear
540 clear
541 history 5

```

图 11: 显示最近执行过的 5 个命令

# 实验题目：进程管理及进程通信

姓名：邢志昂

学号：19122110

实验日期:2021 年 10 月 21 日

## 进程管理及进程通信

### 实验环境:

ubuntu server 20.04 on arm

本系统没有安装桌面，所以以下实验均在 xshell 终端下进行。

### 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。

### 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。

代码如下，根据内容，其将会依次输出：进程标识、父进程标识、进程真实用户标识、进程真实组标识、进程有效用户标识、进程有效组标识。其对应的函数分别为 getpid; getppid; getuid; getgid; geteuid; getegid。运行的结果如图 1

Listing 1: 进程标识

```
1 #include <iostream>
2 #include <unistd.h>
3 #include <sys/wait.h>
4 using namespace std;
5 int main(){
6     cout<<"进程标识 pid = "<<getpid()<<std::endl;
7     cout<<"父进程标识 ppid = "<<getppid()<<std::endl;
8     cout<<"进程真实用户标识 uid = "<<getuid()<<std::endl;
9     cout<<"进程真实组标识 gid = "<<getgid()<<std::endl;
10    cout<<"进程有效用户标识 euid = "<<geteuid()<<std::endl;
11    cout<<"进程有效组标识 egid = "<<getegid()<<std::endl;
12    cout<<"等待5秒钟"<<std::endl;
13    int pid;
14    pid = fork();
```



```
15     if(pid == 0){
16         execlp("date","date",(char*)0);
17         perror("exec error\n");
18         exit(1);
19     }
20     wait(0);
21     char c ;
22     cout<<"Quit?"<<endl;
23     while(cin>>c){
24         if(c == 'y') return 0;
25     }
26
27 }
```

```
root@root:/home/code/proc# ./par
进程标识 pid = 1450036
父进程标识 ppid = 1446933
进程真实用户标识 uid = 0
进程真实组标识 gid = 0
进程有效用户标识 euid = 0
进程有效组标识 egid = 0
等待5秒钟
Wed 10 Nov 2021 11:02:49 AM UTC
Quit?
y
```

图 1: 显示进程的有关标识（进程标识、组标识、用户标识等）

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。

代码见下，可以看到首先执行了 `fork()` 创建了子进程，之后根据进程的状态进行了不同的输出，运行的结果如图 2

Listing 2: 创建子进程

```
1  #include <iostream>
2  #include <unistd.h>
3  #include <sys/wait.h>
4
5  int main(){
6      pid_t id = fork();
7      int status = -1;
8      std::cout<<id<<std::endl;
9      if(id==-1){
10         std::cout<<"fork failed"<<std::endl;
11     }else if(id == 0){
12         std::cout<<"this is the child proc, id = "<<getpid()<<std::endl;
13     }else if(id>0){
14         std::cout<<"this is the parent proc, id = "<<getpid()<<std::endl;
15         int wait_id = wait(&status);
16         std::cout<<"this wait ind is "<<wait_id<<std::endl;
17     }
18 }
```

```
19     return 0;
20 }
```

```
root@root:/home/code/proc# ./main
1451032
this is the parent proc, id = 1451031
0
this is the child proc, id = 1451032
this wait ind is 1451032
```

图 2: 创建子进程

在执行程序之后，系统创建了子进程，首先子进程进行运行并输出，父进程等待子进程的状态，在子进程运行结束后父进程开始运行，并输出了子进程的进程标识。

**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？**

从上面的运行结果来看，我们使用 `fork()` 函数时产生子进程，在其遇到 `exit()` 函数时运行结束，这两个都是系统调用。子进程被创建之后，其会获得父进程完全相同内存拷贝，且与父进程同样在 `fork()` 后继续执行。但是，对于一个计算机而言，拷贝所有数据属实有点浪费，因而，在 Linux 中系统采用了写时复制 (copy-on-write) 技术，即只有在部分内存需要写入时，数据才会被复制；而在此之前，资源都以只读方式共享。此外，`fork()` 系统调用的开销是将父进程的页表拷贝给子进程，并且给子进程创建唯一的 PCB。子进程只需要通过页表获取相应的资源地址，在需要再在相应的位置进行拷贝。而对于每一个进程，必须有唯一的 PCB 与之对应，从而使得系统能够描述进程的基本情况和活动过程，进而控制和管理进程。

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。

代码如下，执行结果见图 3，进程家族树见图 4

Listing 3: 创建多个进程

```
1  #include <iostream>
2  #include <unistd.h>
3  #include <sys/wait.h>
4
5  int main(){
6  //    std::cout<<"id ="<<id<<std::endl;
7      for(int i = 0; i < 3; ++i){
8          int id = fork();
9          if(id==0){
10             std::cout<<"this is child proc, id = "<<getpid()<<" parent is ppid =
                "<<getppid()<<std::endl;
11         }else if(id == -1){
12             std::cout<<"create fork failed"<<std::endl;
13         }else if(id > 0){
14             int status = -1;
15             std::cout<<"this is parent proc, id = "<<getpid()<<" parent is ppid =
                "<<getppid()<<std::endl;
16             int wait_id = wait(&status);
17             std::cout<<"the wait_id = "<<wait_id<<std::endl;
18         }
19     }
```

```
19  
20     }  
21     return 0;  
22 }
```

```
root@root:/home/code/proc# ./main  
1451032  
this is the parent proc, id = 1451031  
0  
this is the child proc, id = 1451032  
this wait ind is 1451032
```

图 3: 子进程

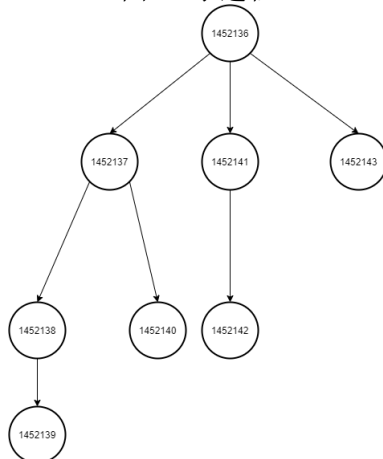


图 4: 进程家族树

思考：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。修改程序，使运行结果呈单分支结构，即每个父进程只产生一个子进程。画出进程树，解释该程序。

a. 家族树已经画出来了，效果见上。反复运行程序的唯一变化就是每次生成的进程号有所不同，但是程序运行的结构不发生改变，即仍然为上面的树形结构，且运行顺序也不发生改变

b. 要想程序运行没有那么多分支，其实只需要令每个子程序不接着进行多次 for 循环即可，即在 else 中加入 break 使其中断。具体查看图 5

```
std::cout<<"create fork failed"<<std::endl;  
}else if(id > 0){  
    int status = -1;  
    std::cout<<"this is parent proc, id = "<<getpid()<<" parent is ppid = "<<getppid()<<std::endl;  
    int wait_id = wait(&status);  
    std::cout<<"the wait_id = "<<wait_id<<std::endl;  
    break;  
}  
}
```

```
root@root:/home/code/proc# ./many_proc  
this is parent proc, id = 1456903 parent is ppid = 1456498  
this is child proc, id = 1456904 parent is ppid = 1456903  
this is parent proc, id = 1456904 parent is ppid = 1456903  
this is child proc, id = 1456905 parent is ppid = 1456904  
this is parent proc, id = 1456905 parent is ppid = 1456904  
this is child proc, id = 1456906 parent is ppid = 1456905  
the wait_id = 1456906  
the wait_id = 1456905  
the wait_id = 1456904  
root@root:/home/code/proc#
```

图 5: 单分支

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。

程序执行两遍后结果如图 6。程序相同，但运行结果有所不同 (见标红位置)，子程序的运行和终止顺序发生了变化。`execlp` 是系统调用函数，即会根据环境变量中的文件执行对应的操作，但是由于程序中父进程在创建三个子进程，三个子进程是并行执行的，无法知道运行结束的先后顺序，也不具备可再现性。

```
root@root:/home/code/proc# ./exec_demo
parent proc is wait for child proc return
Wed 10 Nov 2021 12:00:59 PM UTC
child proc 3
child proc 2 end with status = 0 id= 1457804
child proc 3 end with status = 0 id= 1457805
channel.cpp exec_demo exec_demo.cpp main main.cpp many_proc many_proc.cpp par par.cpp val
child proc 1 end with status = 0 id= 1457803
all child proc end
parent proc end
root@root:/home/code/proc# ./exec_demo
parent proc is wait for child proc return
Wed 10 Nov 2021 12:01:40 PM UTC
child proc 3
child proc 3 end with status = 0 id= 1457869
child proc 2 end with status = 0 id= 1457868
channel.cpp exec_demo exec_demo.cpp main main.cpp many_proc many_proc.cpp par par.cpp val
child proc 1 end with status = 0 id= 1457867
all child proc end
parent proc end
```

图 6: 三个子进程

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。

子进程在运行其他程序后，进程将会被新程序完全取代，但是标识号不变（未创建新进程），除此之外的信息均有源程序取代。而新程序的内容将会从自己的部分进行运行，先后次序不可预知，即子进程的结束只会随着程序的结束而结束，从而不可预知。

5. 参考例程 4 编程，验证子进程继承父进程的变量、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。

从图 7中可以看到，子进程中尝试修改了局部变量和全局变量的值，在子进程中，全局变量的值从 5 减 1 得到 4，而 `val` 变量从 4 加一得到 5，但是在父进程中可以看到其值并未发生变化，这实际与前面 2 的的结论是一致的，因为子程序在使用时会获得父进程的数据拷贝，因而不会影响到父进程。

```
root@root:/home/code/proc# ./val
before fork globa = 4 val = 5
not change in parent proc id= 1458500
change in child proc globa = 5 val = 4 id = 1458501
globa = 5val = 4 id =1458501
globa = 4val = 5 id =1458500
```

图 7: val 值变化

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。

从上述实验结果可以看出，子进程被创建之后，其代码段和数据段都是独立的、经过拷贝的，对父进程的运行环境是没有影响的。

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。要能显示父进程、子进程各自的信息，体现通信效果。

运行结果如图 8，在程序中，首先创建了两个子进程，绑定管道的写入端，并先后写入数据，父进程在读端接收数据。首先创建了两个子进程，在发送完数据后停止。父进程获取数据并打印在屏幕上。

```
root@root:/home/code/proc# ./6
Child process P1!
Child process P2!
P1 1460977 is wakeup. My parent process ID is 1460976.
P2 1460978 is wakeup. My parent process ID is 1460976.
Parent 1460976: Child process P1 is sending messages!
Parent 1460976: Child process P2 is sending messages!
```

图 8: 管道通信

思考：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。修改睡眠时机、睡眠长度，看看会有什么变化。请解释。加锁、解锁起什么作用？不用它行吗？

管道是一个文件，其具有写入端和读出端，分别可以连接两个进程，以生产者-消费者模型进行通讯。在运行过程中，写进程从管道的写入端输入数据，读进程从管道的读出端获取数据。其主要改变了唤醒时间的，对实验的结果不会产生太大影响。锁是为了解决临界资源的共享问题，如不使用，可能会导致数据的混杂和读端数据的错误读取。

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。

运行结果如图 9，这次父进程在运行的时候没有等待、阻塞而自己结束了，导致子进程仍然还在运行状态：

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。

执行进程树指令查看运行情况，可以看到的是，在父进程结束之后，孤儿进程会认 systemd 为父进程，即被 systemd 进程所领养。

```
root@root:/home/code/proc# ./7
1462541
this is the parent proc, id = 1462540
0
this is the child proc, id = 1462541
root@root:/home/code/proc# this is child proc, id = 1462541 now I'm parent id is: 1
```

```
root@root:/home/code/proc# pstree
systemd--7
└─ModemManager--2*[{ModemManager}]
```

图 9: 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。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。

程序运行结果如图 10

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。

在计算机网络的 Socket 通讯，两端都开多线程，一个线程监听，接受信息，另一个线程发送信息。

```

root@root:/home/code/proc# ./6_server
Server :receive from pid 1465814
Server :receive from pid 1465818
Server :receive from pid 1465820
Server :receive from pid 1465822

root@root:/usr# cd /home/code/proc/
root@root:/home/code/proc# ./6_client
Clint : receive from pid 1465664
root@root:/home/code/proc# ./6_client
Clint : receive from pid 1465664
root@root:/home/code/proc# ./6_client
Clint : receive from pid 1465664
root@root:/home/code/proc# ./6_client
Clint : receive from pid 1465664
root@root:/home/code/proc# ./6_client
Clint : receive from pid 1465664
root@root:/home/code/proc#

```

图 10: 进程通信

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。

程序结果见图 11.signal 函数将 18 号处理程序与 func 函数绑定；在父进程调用 `j = kill(i, 18);` 表示向子进程发送 18 号处理程序的中断信号，子进程响应信号，直接打断 `sleep()` 并执行相应 func 函数。

```

root@root:/home/code/proc# ./9
Parent: signal 18 has been sent to child 1466956, returned 0.
I am Process 1466956. It is signal 18 processing function.
Child 1466956: A signal from my parent is received.
After wait 1466956, Parent 1466955: finished.
root@root:/home/code/proc#

```

图 11: 信号处理的系统调用

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。

硬中断由外部硬件产生；软中断由执行中断指令产生的，硬中断可以直接中断 CPU，它会引起内核中相关的代码被触发；软中断并不会直接的中断 CPU，也只有当前正在运行的代码（或者是进程）才会产生软中断，硬中断可以被屏蔽；软中断不可被屏蔽。Linux 内核将中断函数需要处理的任务分为两部分，一部分在中断处理函数中执行，此时系统关闭中断；另外一部分在软件中断中执行，此时开启中断，允许系统响应外部中断。当中断发生时，使用硬中断处理那些短时间就可以完成的工作，而将那些处理事件比较长的工作，放到中断之后来完成，也就是软中断来完成。

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。

代码如下：

Listing 4: 创建多个进程

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <pthread.h>
5  #include <semaphore.h>
6  #include <signal.h>
7  #define N 5    // 消费者或者生产者的数目
8  #define M 10  // 缓冲数目
9  int in = 0; // 生产者放置产品的位置
10 int out = 0; // 消费者取产品的位置
11
12 int buff[M] = { 0 }; // 缓冲初始化为 0，开始时没有产品

```

```
13
14 sem_t empty_sem; // 同步信号量，当满了时阻止生产者放产品
15 sem_t full_sem; // 同步信号量，当没产品时阻止消费者消费
16 pthread_mutex_t mutex; // 互斥信号量，一次只有一个线程访问缓冲
17
18 int product_id = 0; //生产者 id int prochase_id = 0; //消费者 id
19 //信号处理函数
20 void Handlesignal(int signo){
21     printf("程序退出%d\n",signo);
22     exit(0);
23 }
24 /* 打印缓冲情况 */ void print() {
25     int i;
26     printf("产品队列为");
27     for(i = 0; i < M; i++)
28         printf("%d", buff[i]); printf("\n");
29 }
30
31 /* 生产者方法 */
32 void *product() {
33     int id = ++product_id;
34     while(1) { //重复进行
35         //用sleep 的数量可以调节生产和消费的速度，便于观察
36         sleep(2);
37         sem_wait(&empty_sem);
38         pthread_mutex_lock(&mutex);
39
40         in= in % M;
41         printf("生产者%d 在产品队列中放入第%d 个产品\t",id, in);
42
43         buff[in]= 1; print();
44         ++in;
45
46         pthread_mutex_unlock(&mutex);
47         sem_post(&full_sem);
48     }
49 }
50
51
52
53
54 /* 消费者方法 */
55 void *prochase() {
56     int id = ++prochase_id;
57     while(1) { //重复进行
58         //用sleep 的数量可以调节生产和消费的速度，便于观察
59         sleep(5);
60
61         sem_wait(&full_sem);
62         pthread_mutex_lock(&mutex);
63
64         out= out % M;
```

```
65     printf("消费者%d 从产品队列中取出第%d 个产品\t",id, out);
66
67     buff[out]= 0; print();
68     ++out;
69
70     pthread_mutex_unlock(&mutex);
71     sem_post(&empty_sem);
72 }
73 }
74
75 int main() {
76     printf("生产者和消费者数目都为 5,产品缓冲为 10,生产者每 2 秒生产一个产品，消费者每 5
77         秒消费一个产品,Ctrl+退出程序\n");
78
79     pthread_t id1[N];
80     pthread_t id2[N];
81     int i;
82     int ret[N];
83     //结束程序 if(signal(SIGINT,Handlesignal)==SIG_ERR){//按 ctrl+C 产生
84     if(signal(SIGINT,Handlesignal)==SIG_ERR){//按 ctrl+C 产生SIGINT 信号
85     printf("信号安装出错\n");
86     }
87     // 初始化同步信号量
88     int ini1 = sem_init(&empty_sem, 0, M);//产品队列缓冲同步
89     int ini2 = sem_init(&full_sem, 0, 0);//线程运行同步
90     if(ini1 && ini2 != 0) {
91         printf("信号量初始化失败! \n"); exit(1);
92     }
93     //初始化互斥信号量
94     int ini3 = pthread_mutex_init(&mutex, NULL);
95     if(ini3 != 0) {
96         printf("线程同步初始化失败! \n"); exit(1);
97     }
98     // 创建N 个生产者线程
99     for(i = 0; i < N; i++) {
100         ret[i]= pthread_create(&id1[i], NULL, product, (void *) (&i));
101         if(ret[i] != 0) {
102             printf("生产者%d 线程创建失败! \n", i); exit(1);
103         }
104     }
105     //创建 N 个消费者线程
106     for(i = 0; i < N; i++) {
107         ret[i]= pthread_create(&id2[i], NULL, prochase,NULL);
108         if(ret[i] != 0) {
109             printf("消费者%d 线程创建失败! \n", i); exit(1);
110         }
111     }
112     //等待线程销毁
113     for(i = 0; i < N; i++) {
114         pthread_join(id1[i], NULL); pthread_join(id2[i],NULL);
115     }
116     exit(0);
117 }
```



## 一、 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？

Linux 一切皆文件，进程也不例外。Linux 中将进程抽象为文件，一个进程被描述为一类数据结构；系统通过优先级决定进程运行顺序，通过 pid 标记所有进程，提供 ps、top 等命令接口供用户修改进程状态，使用用户堆栈和系统堆栈控制用户态和核心态之间的互相转换，使用管道、信号量等传统的进程通信方式。

### 2. 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。

有一系列系统调用函数。Linux 为 C 提供了一系列调用接口来方便用户进行开发，在本次实验中已经充分见到了，诸如：fork,wait,exit,execlp 等函数均为其提供的接口。

### 3. 什么是进程？如何产生的？举例说明。

进程是进程实体的运行过程，是系统进行资源分配和调度的一个独立单位。执行程序时，会产生相应的进程。很简单的例子，每次运行一个程序，执行 ps，都可以看到系统为该程序运行启动了进程，且若程序在运行过程中若生成了子进程，则同样会产生进程。

### 4. 进程控制如何实现的？举例说明。

PCB。系统为了控制进程，设计出一种数据结构——进程控制块 PCB，其中记录了操作系统所需的、用于描述进程当前情况以及控制进程运行的全部信息。

### 5. 进程通信方式各有什么特点？用程序及运行结果举例说明。

共享存储器系统	基于共享数据结构、共享存储区的通信方式， 仅适用于传递相对少量的数据，通信效率低，属于低级通信)
管道通信系统	互斥，同步
消息传递系统	进程间的数据交换以格式化的消息为单位
客户机服务器系统	基于 socket

### 6. 管道通信如何实现？该通信方式可以用在何处？

管道是一个文件。进程以先进先出的方式从管道存取数据：管道一端的进程顺序地将进程数据写入管道，另一端的进程则顺序地读取数据。管道通信通常用于两个进程之间的相互通信。

### 7. 什么是软中断？软中断信号通信如何实现？

软中断是 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，其不会直接中断 CPU。当谈到软中断通信的时候，就有很多细节了，通常在中断程序的时候，我们都会使用 ctrl+c

SIGHUP	挂起
SIGINT	Ctrl+C
SIGQUIT	Ctrl+\
SIGILL	非法指令

中断，除此之外，还有一些软中断信号：实际上有十九种，可以在系统文件中查看到。这些信号被系统 C 接口的 `signal` 函数定义，其有点像其它语言的异常处理，表示捕捉到中断信号之后执行的操作。

# 实验题目: Linux 进程调度与系统监视

姓名: 邢志昂

学号: 19122110

实验日期: 2021 年 10 月 25 日

---

## Linux 进程调度与系统监视

---

### 实验环境:

ubuntu server 20.04 on arm

### 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。

### 操作过程

#### 1. 作业和进程的基本管理

先在前台启动 vim 编辑器并打开 f4 文件, 然后挂起, 最后在后台启动一个查找.vimrc 文件的 find 作业, find 的查找结果保存到 f5。

- (1) 以超级用户登录系统
- (2) 输入命令 vim f4
- (3) 按下 Ctrl+Z 组合键, 暂时挂起 “vim f4” 作业。
- (4) 输入命令 “find / -name .vimrc >f5 &”, 启动一个后台作业。
- (5) 输入命令 “jobs”, 查看当前系统中的所有作业。
- (6) 输入命令 “fg 2”, 将 “find / -name .vimrc >f5 &” 作业切换到前台。
- (7) 输入命令 “cat f5”
- (8) 再次输入命令 “jobs”
- (9) 输入命令 “kill -9 %1”, 终止 “vim f4” 作业。
- (10) 稍等片刻, 输入命令 “jobs”. 上述操作如图 1

(11) 输入命令 “ps -l”, 查看进程的相关信息

(12) 输入命令 “who -H”, 查看用户信息。上述操作如图 2

```

root@root:/home/code/oslab# jobs
[1]+  Stopped                  vim f4
root@root:/home/code/oslab# find / -name .vimrc >f5 &
[2] 1078579
root@root:/home/code/oslab# jobs
[1]+  Stopped                  vim f4
[2]-  Running                  find / -name .vimrc > f5 &
root@root:/home/code/oslab# fg 2
find / -name .vimrc > f5
find: '/proc/1078583': No such file or directory
root@root:/home/code/oslab# cat f5
/root/.vimrc
/home/.vimrc
root@root:/home/code/oslab# jobs
[1]+  Stopped                  vim f4
root@root:/home/code/oslab# kill -9 %1

[1]+  Stopped                  vim f4
root@root:/home/code/oslab# jobs
[1]+  Killed                   vim f4

```

图 1: 作业和进程的基本管理

```

root@root:/home/code/oslab# ps -l
F S  UID      PID     PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S   0 1076414 1076338 0  80   0 - 2873 do_wai pts/1        00:00:00 bash
4 R   0 1078996 1076414 0  80   0 - 2850 -      pts/1        00:00:00 ps
root@root:/home/code/oslab# who -H
NAME      LINE      TIME          COMMENT
root      pts/0      2021-10-21 01:54 (180.168.188.100)
root      pts/1      2021-10-21 02:01 (101.82.59.41)

```

图 2: 用户信息

## 2. at 进程调度

at 进程调度的创建和删除。

(1) 超级用户输入命令 “at 00:00 01012008”

(2) 屏幕出现 at 调度的命令提示符 “at>”, 输入 “Hello World!”, 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。如图 3

(4) 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。

(5) 屏幕出现 at 调度的命令提示符 “at>”, 输入 “fouces on the screen!”, 向所有用户发送消息。回车

(6) 在 “at>” 提示符的第二行输入 echo ”hello world” +2.

(7) 按下 Ctrl+D 组合键结束输入。如图 4

(8) 输入 “atq” 命令, 查看所有的 at 调度

(9) 输入 “atrm 1” 命令删除作业号为 1 的 at 调度, 并再次输入 “atq” 命令查看剩余的所有 at 调度内容。如图 5

```
root@root:/home/code/oslab# at 00:00 01012022
warning: commands will be executed using /bin/sh
at> Hello Worls^Hd^H!
at> <EOT>
job 1 at Sat Jan 1 00:00:00 2022
```

图 3: 指定时间调度

```
root@root:/home/code/oslab# at now+5 minutes
warning: commands will be executed using /bin/sh
at> wall please focus on screnn
at> echo "Hello world!" +2
at> ^Z
[1]+  Stopped                  at now+5 minutes
root@root:/home/code/oslab# at now+5 minutes
warning: commands will be executed using /bin/sh
at> wall please fouces on screen!
at> echo "Hello World!" +2
at> <EOT>
job 3 at Thu Oct 21 02:32:00 2021
```

图 4: 定时调度

```
root@root:/home/code/oslab# atq
1      Sat Jan 1 00:00:00 2022 a root
2      Thu Oct 21 02:31:00 2021 a root
root@root:/home/code/oslab# atrm 1
root@root:/home/code/oslab# atq
2      Thu Oct 21 02:31:00 2021 a root
```

图 5: 查看并删除 at 调度

### 3. cron 进程调度

用户设置 crontab 调度.

- (1) 输入命令 “crontab -e”
- (2) 输入 “30 8 \* \* \* ps >ps.log”。
- (3) 保存, 输入命令 “crontab -l”, 查看 helen 用户的 cron 调度内容。
- (4) 修改系统时间, 调度结果如图 6
- (5) 再次输入命令 “crontab -e”
- (6) 输入 “0 0 \* \*/3 \* who >who.log”
- (7) 保存, 修改系统时间为 3 月 31 日 23 点 59 分。
- (8) 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。结果如图 7
- (9) 输入命令 “crontab -l”, 查看 cron 调度内容。
- (10) 输入命令 “crontab -r”, 删除 cron 调度内容。
- (11) 再次输入命令 “crontab -l”, 此时无 cron 调度内容. 结果如图 8

```
violet@root:/home/code/oslab$ date
Thu 21 Oct 2021 02:47:47 AM UTC
violet@root:/home/code/oslab$ crontab -e
crontab: installing new crontab
violet@root:/home/code/oslab$ crontab -l
49 2 * * * ps >/home/code/oslab/ps.log
```

PID	TTY	TIME	CMD
1076414	pts/1	00:00:00	bash
1079728	pts/1	00:00:00	at
1081322	pts/1	00:00:00	su
1083681	pts/1	00:00:00	su
1083687	pts/1	00:00:00	bash
1083723	pts/1	00:00:00	ps

图 6: 每天上午 8 点 30 分查看系统的进程状态

```
violet@root:/home/code/oslab$ crontab -e
crontab: installing new crontab
violet@root:/home/code/oslab$ su root
Password:
root@root:/home/code/oslab# date 03312359
Wed 31 Mar 2021 11:59:00 PM UTC
root@root:/home/code/oslab# exit
exit
```

```
violet@root:/home/code/oslab$ cat who.log
root pts/1 2021-04-01 00:00 (101.82.59.41)
```

图 7: 每三个月的 1 号零时查看正在使用的用户列表

```
violet@root:/home/code/oslab$ crontab -l
49 2 * * * ps >/home/code/oslab/ps.log
0 0 * */3 * who >/home/code/oslab/who.log
violet@root:/home/code/oslab$ crontab -r
violet@root:/home/code/oslab$ crontab -l
no crontab for violet
violet@root:/home/code/oslab$
```

图 8: 查看并删除 crontab 调度

- (1) 输入命令“top”，屏幕动态显示 CPU 利用率，如图 9
- (2) 按下 M 键，所有进程按照内存使用率排列，如图 10
- (3) 按下 T 键，所有进程按照执行时间排列，如图 11
- (4) 最后按下 P 键，恢复按照 CPU 使用率排列所有进程。

5

# 实验题目：用户与组群管理

姓名：邢志昂

学号：19122110

实验日期:2021 年 11 月 6 日

---

## 用户与组群管理

---

### 实验环境:

ubuntu server 20.04 on arm

本系统没有安装桌面，所以以下实验均在 xshell 终端下进行。

### 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。

### 操作过程

#### 1. 终端环境下管理用户与组群

新建用户

- (1) 以超级用户登录系统
- (2) 输入命令 useradd xuser1
- (3) 输入命令 passwd xuser1, 为 xuser1 设置密码
- (4) 输入命令 useradd xuser2
- (5) 输入命令 passwd xuser2, 为 xuser2 设置密码, 结果如图 1
- (6) 输入命令 cat /etc/passwd, 结果如图 2
- (7) 输入命令 cat /etc/shadow, 结果如图 3
- (8) 输入命令 cat /etc/group, 结果如图 4
- (9) 输入命令 cat /etc/gshadow, 结果如图 5
- (10) 输入命令 su xuser2 -l, 切换至 xuser2 用户



(11) 输入命令 `pwd`, 查看当前目录

(12) 输入命令 `exit`, 退出登录。结果如图 6

```
[root@root ~]# useradd xuser1
useradd: user 'xuser1' already exists
[root@root ~]# passwd xuser1
Changing password for user xuser1.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
[root@root ~]# useradd xuser2
[root@root ~]# passwd xuser2
Changing password for user xuser2.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
[root@root ~]#
```

图 1: 新建用户

```
violet:x:1003:1003::/home/violet:/bin/bash
xuser1:x:1004:1004::/home/xuser1:/bin/bash
xuser2:x:1005:1005::/home/xuser2:/bin/bash
```

图 2: passwd 文件

```
xuser1:$6$yB.bZyFY00mBIkVo$vfCQKQ2hueW9iD5SmuEWCW11shJFFh2WD6FirZW350U1bIH
:7:::
xuser2:$6$ZxmopCRe1TQP0qF1$BK7eQKZTS9iPwgejPrJIsujer2zhBgJafXw1zPF7zHZXqZc
:7:::
```

图 3: shadow 文件

```
violet:x:1003:
xuser1:x:1004:
xuser2:x:1005:
```

图 4: group 文件

```
hadoop:!:
violet:!:
xuser1:!:
xuser2:!:
```

图 5: gshadow 文件

```
[root@root ~]# su xuser2 -l
Last login: Tue Sep 20 06:02:16 CST 2011 on pts/0
-bash: /home/xuser2/.nvm/nvm.sh: No such file or directory
[xuser2@root ~]$ pwd
/home/xuser2
[xuser2@root ~]$ exit
logout
```

图 6: xuser2 login

锁定并删除用户

- (1) 输入命令 `passwd -l xuser2`, 锁定 `xuser2`
- (2) 输入命令 `passwd -S xuser2`, 查看用户状态, 如图 7
- (3) 输入命令 `userdel xuser2`, 删除用户 `xuser2`
- (4) 输入命令 `cat /etc/passwd`, 查看用户信息, 如图 8

```
[root@root ~]# passwd -l xuser2
Locking password for user xuser2.
passwd: Success
[root@root ~]# passwd -S xuser2
xuser2 LK 2011-09-19 0 99999 7 -1 (Password locked.)
```

图 7: 锁定用户

```
mysql:x:27:27:MySQL Server:/var/lib/mysql:/sbin
hadoop:x:1002:1002::/home/hadoop:/bin/bash
violet:x:1003:1003::/home/violet:/bin/bash
xuser1:x:1004:1004::/home/xuser1:/bin/bash
[root@root ~]#
```

图 8: passwd 文件

#### 用户组群管理

- (1) 输入命令 `groupadd myusers`
- (2) 输入命令 `groupadd temp`
- (3) 输入命令 `usermod -a -G myusers xuser1`, 将 `xuser1` 加入 `myusers`
- (4) 输入命令 `usermod -a -G myusers violet`, 将 `violet` 加入 `myusers`, 如图 9
- (5) 输入命令 `groupdel temp` 删除群组 `temp`, 如图 10

```
[root@root ~]# groupadd myusers
[root@root ~]# groupadd temp
[root@root ~]# usermod -a -G myusers xuser1
[root@root ~]# usermod -a -G myusers violet
```

图 9: 新建用户组

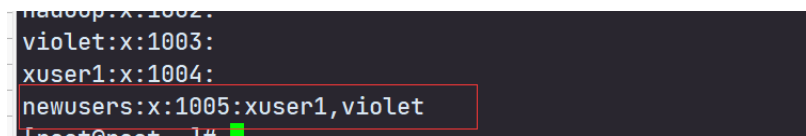
```
[root@root ~]# groupdel temp
[root@root ~]#
```

图 10: 删除用户组

## 2. 利用 Shell 命令管理用户与组群

本小节实验与上一小节实验内容基本一致, 以下仅对上一小节未涉及到的实验内容进行补充.  
将 `myusers` 用户组更名为 `newusers`

- (1) 输入命令 `groupmod -n newusers myusers`
- (2) 输入命令 `cat /etc/group`, 结果如图 11



```
hadoop:x:1002:
violet:x:1003:
xuser1:x:1004:
newusers:x:1005:xuser1,violet
```

图 11: 更改用户组名

批量处理多个用户

- (1) 输入命令 `groupadd -g 600 class`
- (2) 输入命令 `vim student`
- (3) 在 `student` 文件中添加“`s080101:x:601:600::/home/s080101:/bin/bash`”等 8 个用户，保存退出
- (4) 输入命令 `vim stu-passwd`
- (5) 在 `stu-passwd` 文件中添加“`s080101:111111`”等八项，用于配置上述用户的密码
- (6) 输入命令“`newusers < student`”，批量新建用户帐号。
- (7) 输入命令“`pwunconv`”，暂时取消 shadow 加密。
- (8) 输入命令“`chpasswd <stu-passwd`”，批量新建用户的口令。
- (9) 输入命令“`pwconv`”，进行 shadow 加密，完成批量创建用户帐号工作. 如图 12
- (10) 输入 `cat /etc/passwd` 命令查看已添加用户，如图 13

```
[root@root code]# vim student
[root@root code]# vim stu-passwd
[root@root code]# ls
mapred  proc  student  stu-passwd
[root@root code]# newusers < student
[root@root code]# pwunconv
[root@root code]# chpasswd <stu-passwd
[root@root code]# pwconv
[root@root code]#
```

图 12: 批量新建用户

```
violet:x:1003:1003:./home/violet:/bin/bash
xuser1:x:1004:1004:./home/xuser1:/bin/bash
s080101:x:601:600:./home/s080101:/bin/bash
s080102:x:602:600:./home/s080102:/bin/bash
s080103:x:603:600:./home/s080103:/bin/bash
s080104:x:604:600:./home/s080104:/bin/bash
s080105:x:605:600:./home/s080105:/bin/bash
s080106:x:606:600:./home/s080106:/bin/bash
s080107:x:607:600:./home/s080107:/bin/bash
s080108:x:608:600:./home/s080108:/bin/bash
```

图 13: 查看批量新建的用户

# 实验题目: Shell 编程

姓名: 邢志昂

学号: 19122110

实验日期: 2021 年 11 月 10 日

## Shell 编程

### 实验环境:

ubuntu server 20.04 on arm

### 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。

### 操作过程

#### 1. VIM

vim 编辑程序有三种操作模式, 分别称为编辑模式、插入模式和命令模式, 当运行 Vim 时, 首先进入编辑模式。

vim 编辑模式的主要用途是在被编辑的文件中移动光标的位置。一旦光标移到到所要的位置, 就可以进行剪切和粘贴正文块, 删除正文和插入新的正文。

vim 插入模式的主要用途就是对写文本

vim 命令模式的主要作用为执行相关命令, 进行如查找, 替换, 等常用操作。同时, 在命令模式下还可以对 Vim 本身进行配置, 如增加行号的功能。图 1 展示了配置后的 vim

vim 有多种方式进入插入模式, 如按 i 在光标后插入, 按 a 在光标前插入, 按 s 替换并插入等, 详细内容见 vim 键位图, 如图 2。按下 esc 键可以退出插入模式, 键入:w 命令可将文件存盘。

#### 2. 创建和执行 Shell 程序

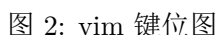
##### (1) 基本输入

编写例 1 程序如下, 给予 sh 文件可执行权限后, 执行结果如图 3

Listing 1: 读取输入参数, prog1.sh

```
1 if [ $# == 0 ]
2 then
3     echo "Name not provided"
4 else
5     echo "You name is" $1
6 fi
```

图 1: 配置后的 vim



进一步修改程序 prog1.sh, 显示参数个数、程序名字, 并逐个显示参数。修改后的代码如下。执行结果如图 4。

Listing 2: prog2.sh

```
1 echo "执行的文件名: $0";
2 echo "参数个数为: $#";
3 for i in "$@"; do
4     echo $i
5 done
```

进一步修改程序 prog1.sh, 用 read 命令接受键盘输入。若没有输入显示 no input, 如果有输入显示输入的值。代码如下。执行结果如图 5

Listing 3: prog3.sh

```
1 read demo
2 if [[ $demo == '' ]]
3 then
4     echo "no input!!!!!"
5 else
6     echo "value is $demo"
7 fi
```

```
root@root:/home/shell_test# ./myproc1.sh
Name not provided
root@root:/home/shell_test# ./myproc1.sh violet
You name is violet
root@root:/home/shell_test#
```

图 3: 读取参数

```
root@root:/home/shell_test# ./myproc2.sh 1 2 3 4
执行的文件名: ./myproc2.sh
参数个数为: 4
1
2
3
4
```

图 4: 显示参数个数

```
root@root:/home/shell_test# ./myproc3.sh
no input!!!!
root@root:/home/shell_test# ./myproc3.sh
apple
value is apple
```

图 5: read 命令检测输入

## (2) 比较运算符

字符串比较运算: -z string 如果 string 长度为零; -n string 如果 string 长度非零, 则为真; string1 = string2 如果 string1 与 string2 相同, 则为真; string1 != string2 如果 string1 与 string2 不同, 则为真; 实验代码如下, 执行结果如图 7

Listing 4: 字符串比较

```
1 string1="The First one"
2 string2="The second one"
3 if [ string1 = string2 ]
4 then
5 echo "string1 equal to string2"
6 else echo "string1 not equal to string2"
7 fi
8 if [ string1 ]
9 then
10 echo "string1 is not empty"
11 else echo "string1 is empty"
12 fi
13 if [ -n string2 ]
14 then
15 echo "string2 has a length greater than zero"
16 else echo "string2 has a length equal to zero"
17 fi
```

文件比较代码如下, 执行结果如图 ??

Listing 5: 文件比较

```
1 if [ -d cppdir ]
2 then
3 echo "cppdir is a directory"
4 else echo "cppdir is not a directory"
5 fi
6 if [ -f filea ]
7 then
8 echo "filea is a regular file"
9 else echo "filea is not a regular file"
10 fi
11 if [ -r filea ]
12 then
13 echo "filea has read permissione"
14 else echo "filea dose not have read permissione"
15 fi
16 if [ -w filea ]
17 then
18 echo "filea has write permissione"
19 else echo "filea dose not have write permissione"
20 fi
21 if [ -x cppdir ]
22 then
23 echo "cppdir has execute permissione"
24 else echo "cppdir dose not have execute permissione"
25 fi
```

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。修改后代码如下, 执行结果如图 9

Listing 6: 文件比较



```
1 read -p "input string1:" string1
2 read -p "input string2:" string2
3 if [ string1 = string2 ]
4 then
5 echo "string1 equal to string2"
6 else echo "string1 not equal to string2"
7 fi
8 if [ string1 ]
9 then
10 echo "string1 is not empty"
11 else echo "string1 is empty"
12 fi
13 if [ -n string2 ]
14 then
15 echo "string2 has a length greater than zero"
16 else echo "string2 has a length equal to zero"
17 fi
```

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。修改后代码如下, 执行结果如图 ??

Listing 7: 文件比较

```
1 read -e -p "input file name:" filename
2 if [ -d $filename ]
3 then
4 echo "$filename is a directory"
5 else echo "$filename is not a directory"
6 fi
7 if [ -f $filename ]
8 then
9 echo "$filename is a regular file"
10 else echo "$filename is not a regular file"
11 fi
12 if [ -r $filename ]
13 then
14 echo "$filename has read permission"
15 else echo "$filename dose not have read permission"
16 fi
17 if [ -w $filename ]
18 then
19 echo "$filename has write permission"
20 else echo "$filename dose not have write permission"
21 fi
22 if [ -x $filename ]
23 then
24 echo "$filename has execute permission"
25 else echo "$filename dose not have execute permission"
26 fi
```

```
root@root:/home/shell_test# ./str.sh
string1 not equal to string2
string1 is not empty
string2 has a length greater than zero
```

图 6: 字符串比较

```
drwxr-xr-x  4 root root 4096 Nov 10 08:05 ./
drwxr-xr-x 11 root root 4096 Nov  4 02:57 ../
drwxr-xr-x  2 root root 4096 Nov 10 08:05 cppdir/
-rwxrwxr-x  1 root root  530 Nov 10 08:07 dir.sh*
drwxr-xr-x  2 root root 4096 Nov 10 08:05 filea/
-rwxrwxr-x  1 root root   80 Nov  4 03:05 myproc1.sh*
-rwxrwxr-x  1 root root   96 Nov  4 03:22 myproc2.sh*
-rwxrwxr-x  1 root root   91 Nov 10 07:43 myproc3.sh*
-rwxrwxr-x  1 root root  363 Nov 10 07:55 str.sh*
root@root:/home/shell_test# ./dir.sh
cppdir is a directory
filea is not a regular file
filea has read permissione
filea has write permissione
cppdir has execute permissione
```

图 7: 文件比较

```
root@root:/home/shell_test# ./str2.sh
input string1:apple
input string2:banana
string1 not equal to string2
string1 is not empty
string2 has a length greater than zero
root@root:/home/shell_test#
```

图 8: 随机输入字符并比较

```
root@root:/home/shell_test# ./dir2.sh
input file name:str2.sh
str2.sh is not a directory
str2.sh is a regular file
str2.sh has read permissione
str2.sh has write permissione
str2.sh has execute permissione
```

图 9: 随机文件名, 判断文件属性

## (3) 指导书示例程序

例 4 执行结果图 10, 我们可以发现, prog4.sh 执行后, 当前文件夹里的文件已经被复制到了 backup 文件夹里。例 5 和例 6 实验指导书中 bug 较多, 对代码进行修改后执行效果如图 11。

```

root@root:/home/shell_test# ls backup/
root@root:/home/shell_test# ./prog4.sh
backup
cp: -r not specified; omitting directory 'backup'
copy backup failed
cppdir
cp: -r not specified; omitting directory 'cppdir'
copy cppdir failed
dir2.sh
dir.sh
filea
cp: -r not specified; omitting directory 'filea'
copy filea failed
myproc1.sh
myproc2.sh
myproc3.sh
prog4.sh
prog5.sh
prog6.sh
prog7.sh
prog8.sh
prog9.sh
str2.sh
str.sh
root@root:/home/shell_test# ls backup/
dir2.sh dir.sh myproc1.sh myproc2.sh myproc3.sh prog4.sh prog5.sh prog6.sh prog7.sh prog8.sh prog9.sh str2.sh str.sh

```

图 10: 将当前文件里的文件复制到 backup

```

root@root:/home/shell_test# ./prog5.sh
result is 110

```

```

root@root:/home/shell_test# ./prog6.sh
result is 110

```

图 11: 进行 10 个偶数的相加

例 7 执行结果如图 12。

```

root@root:/home/shell_test# ./prog7.sh
1) continue
2) finish
#? 1
#? 2

```

图 12: 功能选择

例 8 和例 9 执行结果如图 13

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。其中 HOME 目录和 PATH 目录的列表主要使用 \$HOME 和 \$PATH 两个环境变量。而文件的属性则参考例三的代码。代码如下。执行效果如图 14。

Listing 8: 显示用户主目录名 (HOME)

```

1  #!/bin/bash
2  echo "The path of HOME: ${HOME}"
3  echo "The path of PATH: ${PATH}"
4  if [ -d $1 ]
5  then echo "$1 is a directory"
6  else echo "$1 is not a directory"

```

```
root@root:/home/shell_test# ./prog8.sh
"Invalid parameter"
root@root:/home/shell_test# ./prog8.sh 2
"Month is February"
root@root:/home/shell_test# ./prog8.sh 6
"Month is June"
```

```
root@root:/home/shell_test# ./prog9.sh
"Month is August"
"Month is December"
```

图 13: 例 8 和例 9 的执行结果

```
7  fi
8  if [ -f $1 ]
9  then echo "$1 is a regular file"
10 else echo "$1 is not a regular file"
11 fi
12 if [ -r $1 ]
13 then echo "$1 has read permissione"
14 else echo "$1 dose not have read permissione"
15 fi
16 if [ -w $1 ]
17 then echo "$1 has write permissione"
18 else echo "$1 dose not have write permissione"
19 fi
20 if [ -x $1 ]
21 then echo "$1 has execute permissione"
22 else echo "$1 dose not have execute permissione"
23 fi
```

```
root@root:/home/shell_test# ./myprog.sh
The path of HOME: /root
The path of PATH: /usr/hadoop-3.3.0/bin:/usr/local/sbin:/usr/local/bin:/usr/jdk1.8.0_301/bin:/root/.vim/kg/bin
is a directory
is a regular file
has read permissione
has write permissione
has execute permissione
```

图 14: 例 10 的执行结果

## 一、 讨论

### 1. Linux 的 Shell 有什么特点?

- (1) 把已有命令进行适当组合构成新的命令。
- (2) 提供了文件名扩展字符 (通配符, 如 \*、?、[ ]), 使得用单一的字符串可以匹配多个文件名, 省去键入一长串文件名的麻烦。
- (3) 可以直接使用 Shell 的内置命令, 而不需创建新的进程, 如 Shell 中提供的 cd、echo、exit、pwd、kill 等命令。为防止因某些 Shell 不支持这类命令而出现麻烦, 许多命令都提供了对应的二进制代码, 从而也可以在新进程中运行。
- (4) Shell 允许灵活地使用数据流, 提供通配符、输入/输出重定向、管道线等机制, 方便了模式匹配、I/O 处理和数据传输。

- (5) 结构化的程序模块, 提供了顺序流程控制、条件控制、循环控制等。
- (6) Shell 提供了在后台执行命令的能力。
- (7) Shell 提供了可配置的环境, 允许创建和修改命令、命令提示符和其它的系统行为。
- (8) Shell 提供了一个高级的命令语言, 能够创建从简单到复杂的程序。这些 Shell 程序称为 Shell 脚本, 利用 Shell 脚本, 可把用户编写的可执行程序与 Unix 命令结合在一起, 当作新的命令使用, 从而便于用户开发新的命令。

## 2. 怎样进行 Shell 编程? 如何运行? 有什么条件?

进行 shell 编程, 首先需要有一个 shell, 当然, 默认的系统都安装有 shell。之后创建一个文件, 默认的文件显然是无法执行的, 需要给予文件可执行权限, 之后编辑文件, 编写 shell 程序即可。

## 3. vim 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 vim 中执行 Shell 命令的方式。

vim 相关内容见以上报告。

## 4. 编写一个具有以下功能的 Shell 程序。

编写程序如下, 程序执行结果见图 15

Listing 9: 讨论题

```
1 for filename in `ls`
2 do
3 echo "$filename"
4 echo "$filename" >> filedir.txt
5 done
6 `mkdir testdir2`
7 for file in `ls`
8 do
9 if [ "${file##*.}"x = "c"x ]
10 then
11 cp "$file" "testdir2/$file"
12 fi
13 done
14 for file in `ls`
15 do
16 `chmod 333 "$file"`
17 done
18 for file in `ls testdir2`
19 do
20 echo "$file" >> filedir.txt
21 done
22 echo `who am i`
```

```
root@root:/home/shell_test# ./myprog2.sh
backup
cpkdir
dir2.sh
dir.sh
filea
myproc1.sh
myproc2.sh
myproc3.sh
myprog2.sh
myprog.sh
prog4.sh
prog5.sh
prog6.sh
prog7.sh
prog8.sh
prog9.sh
str2.sh
str.sh
root pts/0 2021-11-10 09:31 (101.82.238.44)
```

```
root@root:/home/shell_test# cat filedir.txt
backup
cpkdir
dir2.sh
dir.sh
filea
myproc1.sh
myproc2.sh
myproc3.sh
myprog2.sh
myprog.sh
prog4.sh
prog5.sh
prog6.sh
prog7.sh
prog8.sh
prog9.sh
str2.sh
str.sh
```

图 15: 讨论题 4 的执行结果