Linux

张海宁[1]

May 17, 2018

[1]hnzhang1@gzu.edu.cn

Contents

Chapter 1

Basic Opertation

1.1 目的

1. 熟悉 linux 下的常用命令
2. 熟悉 linux 下的文本编辑工具
3. 熟悉重定向和管道的使用

1.2 目标

1. 掌握 pwd, cd, ls, cat, cp, mv, rm, head, tail, find, man 等命令的使用方法
2. 掌握 vim 的基本使用方法
3. 掌握重定向和管道的使用方法

1.3 实验过程

1.3.1 常用命令的使用

pwd

pwd—Print Working Directory

```
$ pwd
/Users/hainingzhang
```

cd

Change Directory - change the current working directory to a specific Folder

```
$ cd /var/log/
$ pwd
/var/log
$ cd
$ pwd
/Users/hainingzhang
```

```
$ cd ~
$ pwd
/Users/hainingzhang
$ cd .
$ pwd
/Users/hainingzhang
$ cd ..
$ pwd
/Users
```

注意 ~ . .. 这三个符号的特殊意义以及 cd 后面不接参数的用法。

ls

List directory contents.

```
$ ls
Desktop Documents Downloads IdeaProjects Library Movies Music Pictures Public tensorf
$ ls -a
. Downloads Music .viminfo IdeaProjects Pictures
.bash_history Desktop Library Public
.bash_sessions Documents Movies tensorflow
$ ls -l
total 0
drwx------@ 6 hainingzhang staff 192 Mar 22 23:09 Desktop
drwx------@ 7 hainingzhang staff 224 Mar 23 23:53 Documents
drwx------+ 10 hainingzhang staff 320 Mar 22 21:12 Downloads
drwxr-xr-x 3 hainingzhang staff 96 Mar 22 00:14 IdeaProjects
drwx------@ 61 hainingzhang staff 1952 Mar 19 08:35 Library
drwx------+ 3 hainingzhang staff 96 Mar 18 10:14 Movies
drwx------+ 3 hainingzhang staff 96 Mar 18 10:14 Music
drwx------+ 5 hainingzhang staff 160 Mar 19 00:09 Pictures
drwxr-xr-x+ 5 hainingzhang staff 160 Mar 18 10:14 Public
drwxr-xr-x 8 hainingzhang staff 256 Mar 20 08:17 tensorflow
HainingdeMacBook-Pro:~ hainingzhang$ ls -al
total 88
drwxr-xr-x+ 24 hainingzhang staff 768 Mar 24 00:04 .
drwxr-xr-x 6 root admin 192 Mar 18 12:06 ..
-rw------- 1 hainingzhang staff 0 Mar 19 00:44 .bash_history
drwx------ 10 hainingzhang staff 320 Mar 19 22:30 .bash_sessions
-rw-r--r-- 1 hainingzhang staff 49 Mar 22 00:22 .gitconfig
-rw------- 1 hainingzhang staff 17529 Mar 22 23:14 .viminfo
drwx------@ 6 hainingzhang staff 192 Mar 22 23:09 Desktop
drwx------@ 7 hainingzhang staff 224 Mar 23 23:53 Documents
drwx------+ 10 hainingzhang staff 320 Mar 22 21:12 Downloads
drwxr-xr-x 3 hainingzhang staff 96 Mar 22 00:14 IdeaProjects
drwx------@ 61 hainingzhang staff 1952 Mar 19 08:35 Library
drwx------+ 3 hainingzhang staff 96 Mar 18 10:14 Movies
drwx------+ 3 hainingzhang staff 96 Mar 18 10:14 Music
```

cat

Concatenate and print (display) the content of files.

```
$ cat wifi.log
Mar 24 00:30:00 HainingdeMBP newsyslog[5697]: logfile turned over
$ cat -n wifi.log
1 Mar 24 00:30:00 HainingdeMBP newsyslog[5697]: logfile turned over
$ cat wifi.log wifi.log
Mar 24 00:30:00 HainingdeMBP newsyslog[5697]: logfile turned over
Mar 24 00:30:00 HainingdeMBP newsyslog[5697]: logfile turned over
$ cat -n wifi.log wifi.log
1 Mar 24 00:30:00 HainingdeMBP newsyslog[5697]: logfile turned over
1 Mar 24 00:30:00 HainingdeMBP newsyslog[5697]: logfile turned over
```

cp

Copy files and directories.

```
$ ls -al | grep cat
-rw-rw-r--. 1 zhanghaining zhanghaining 46 Mar 20 14:49 catErr.txt
-rw-rw-r--. 1 zhanghaining zhanghaining 0 Mar 20 14:49 cat.txt
$ cp cat.txt catCP.txt
$ ls -al | grep cat
-rw-rw-r--. 1 zhanghaining zhanghaining 0 Mar 24 11:33 catCP.txt
-rw-rw-r--. 1 zhanghaining zhanghaining 46 Mar 20 14:49 catErr.txt
-rw-rw-r--. 1 zhanghaining zhanghaining 0 Mar 20 14:49 cat.txt

$ ls -al|grep linux
drwxrwxr-x. 2 zhanghaining zhanghaining 4096 Mar 15 10:29 linux
$ cp -r linux/ linuxbak
$ ls -al|grep linux
drwxrwxr-x. 2 zhanghaining zhanghaining 4096 Mar 15 10:29 linux
drwxrwxr-x. 2 zhanghaining zhanghaining 4096 Mar 24 11:38 linuxbak
```

mv

Move or rename files or directories.

```
$ ls -al|grep linux
drwxrwxr-x. 2 zhanghaining zhanghaining 4096 Mar 15 10:29 linux
drwxrwxr-x. 2 zhanghaining zhanghaining 4096 Mar 24 11:38 linuxbak
drwxrwxr-x. 2 zhanghaining zhanghaining 4096 Mar 24 11:38 linuxbakR
$ mv linuxbakR linuxbakr
$ ls -al|grep linux
drwxrwxr-x. 2 zhanghaining zhanghaining 4096 Mar 15 10:29 linux
drwxrwxr-x. 2 zhanghaining zhanghaining 4096 Mar 24 11:38 linuxbak
drwxrwxr-x. 2 zhanghaining zhanghaining 4096 Mar 24 11:38 linuxbakr

$ ls -al|grep list
-rw-rw-r--. 1 zhanghaining zhanghaining 339 Mar 24 11:38 list
$ mv list listMV
```

```
$ ls -al|grep list
-rw-rw-r--. 1 zhanghaining zhanghaining 339 Mar 24 11:38 listMV
$ mv listMV ../
$ ls -al|grep list
$ ls -al ../ | grep list
-rw-rw-r--. 1 zhanghaining zhanghaining 339 Mar 24 11:38 listMV
```

rm

Remove files or directories.

```
$ ls -al|grep list
-rw-rw-r--. 1 zhanghaining zhanghaining 339 Mar 24 11:38 listMV
$ rm listMV
$ ls -al|grep list


$ ls -al|grep linux
drwxrwxr-x. 2 zhanghaining zhanghaining 4096 Mar 15 10:29 linux
drwxrwxr-x. 2 zhanghaining zhanghaining 4096 Mar 24 11:38 linuxbak
drwxrwxr-x. 2 zhanghaining zhanghaining 4096 Mar 24 11:45 linuxbakr
$ rm -r linuxbakr
$ ls -al|grep linux
drwxrwxr-x. 2 zhanghaining zhanghaining 4096 Mar 15 10:29 linux
drwxrwxr-x. 2 zhanghaining zhanghaining 4096 Mar 24 11:38 linuxbak

$ ls
adduser.sh a.out edTst.txt hw1.c jk151-add jk151.dat jk151-passwd
$ rm -i a.out
rm: remove regular file  ‘a.out’ ? n
```

head

Output the first part of files, prints the first part (10 lines by default) of each file.

```
$ wc -l adduser.sh
67 adduser.sh
$ head adduser.sh
#! /bin/bash
# This is to add users in batch mode
# Written by Wang
# wang
# 2015.03.13
#
# MAKE SURE THE $group.dat IS IN UNIX FORMAT,
# INSTEAD OF DOS FORMAT
# OTHERWISE, THERE MAY INDUCE 'INVALID USER NAME' ERROR
# 2018.3.6

$ head -2 adduser.sh
```

```
#! /bin/bash
# This is to add users in batch mode

$ head -2 adduser.sh jk151-passwd
==> adduser.sh <==
#! /bin/bash
# This is to add users in batch mode

==> jk151-passwd <==
150017014:PcCMdhm
150017015:5i222Uy
```

tail

Output the last part of files, print the last part (10 lines by default) of each FILE.

```
$ tail -2 adduser.sh

echo "Done."
```

find

Search for files in a directory hierarchy.

```
$ find ~ -name "*.c"
/home/zhanghaining/linux/hw1.c
/home/zhanghaining/linuxbak/hw1.c
```

man

man 是一个获取某个命令使用方法的工具。

```
$ man wc
```

man 打开的页面，可以按 q 键退出，其他光标移动方式和 vim 相同。

1.3.2 vim 的使用

打开或新建文件

```
$ vim afile
```

如果 afile 不存在则新建 afile, 否则直接打开文件。如 Figure 1.1所示。在 Figure 1.1中：最左侧的 ~ 代表当前行是空行，最下面是状态栏（显示了文件名，当前光标所在位置等信息）。vim 的这种状态称为一般模式。

插入文本

在一般模式下，可以按 iIoOaA 这几个字符中的一个进行插入字符或行的操作。如果按下了 i，则进入了插入状态，如 Figure 1.2所示。可以看到 Figure 1.2中左下状态栏中有 insert 字样，此时可以进行文件输入，直到按下 esc 键，按下 esc 键后，vim 又会重返一般模式。

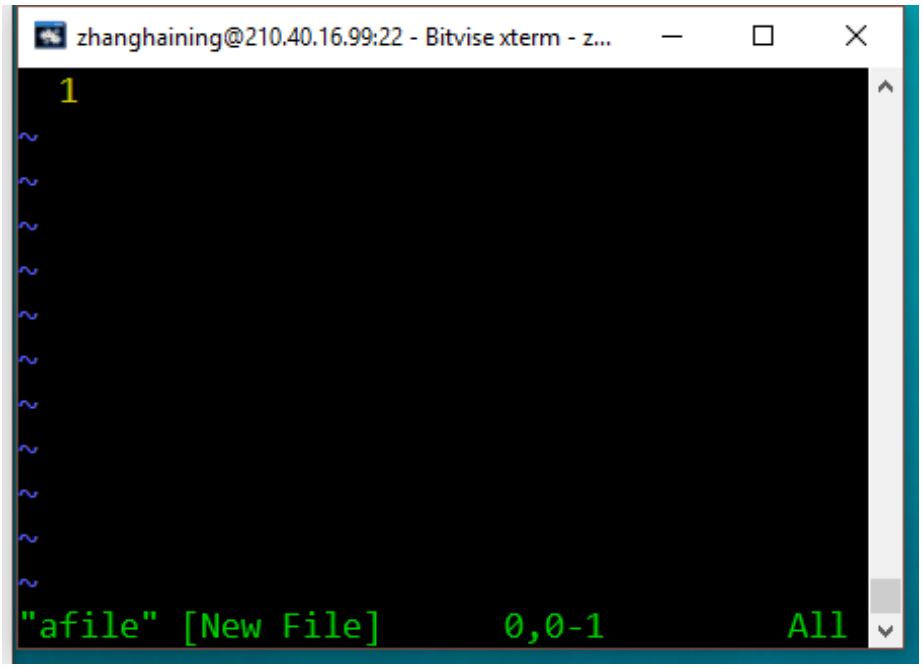

Figure 1.1: vim open a file

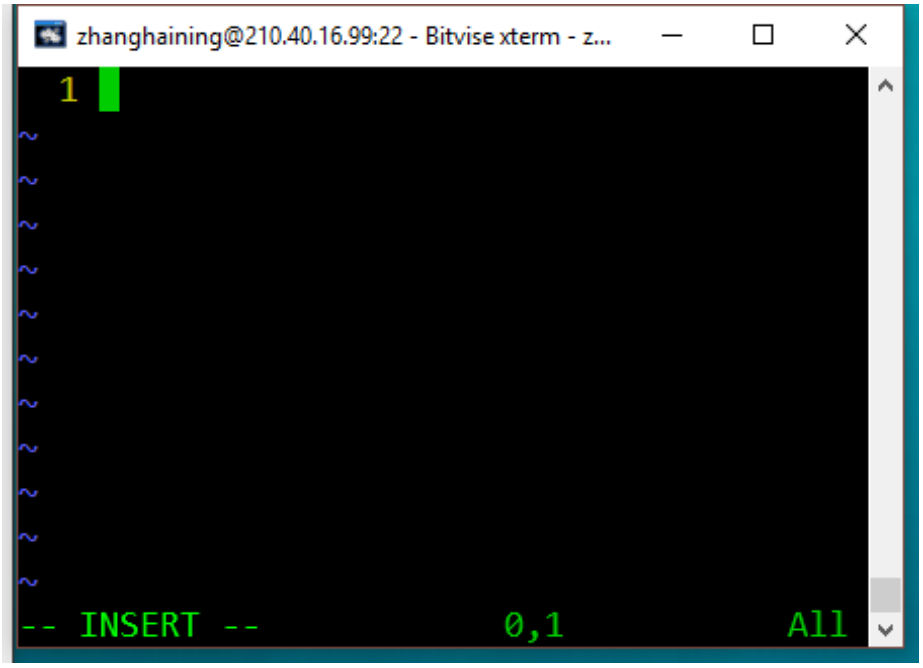

Figure 1.2: insert in vim

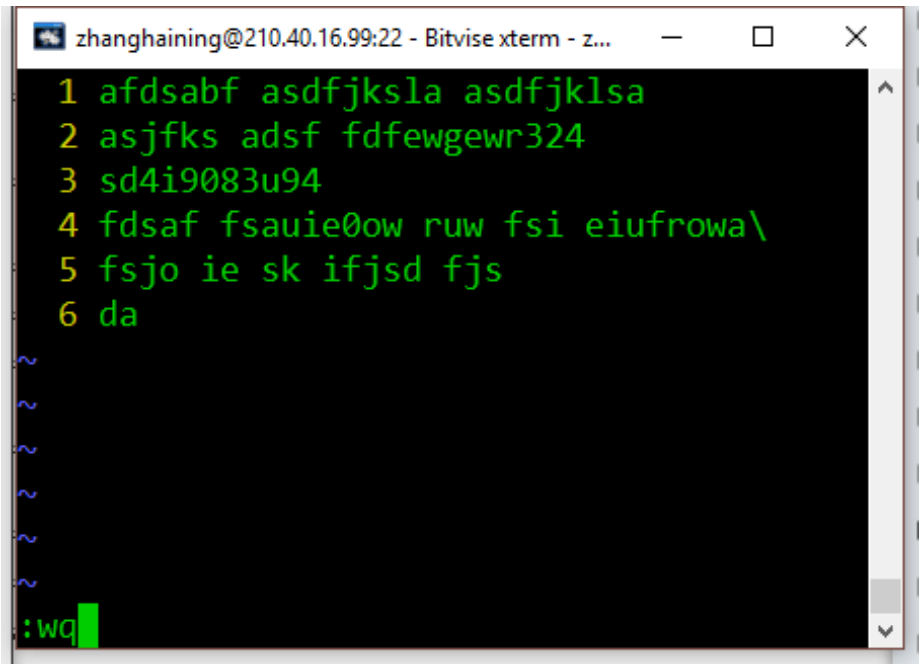

Figure 1.3: save and quit

删除文本和移动光标

在一般模式下，可以按下 hjkl 这 4 个键其中之一进行光标的移动（一次移动一个字符或一行），也可以按下 w 键将光标移动到下一个单词的词首，或 b 键移动到上一个单词的词首。

如要删除某个字符可将光标移动到该字符上，然后按 x 键，如要删除一行，则将光标移动到那一行，然后按 dd(d 键连按两次)。

保存文件和退出 vim

在一般模式下，按下：键，输入 w 即可完成文件的保存，输入 q 键即可退出 vim，输入 wq 则可以完成保存并退出。如 Figure 1.3所示。

1.3.3 重定向和管道

重定向就是将数据流的目的地进行改变的一种机制。

```
$ cat afile > bfile
$ cat afile >> bfile
$ find / -name "shadow" 2> findErr
/usr/share/texlive/texmf-dist/tex/latex/shadow
/usr/share/texlive/texmf-dist/doc/latex/shadow
/etc/shadow
$ find / -name "shadow" > findLOG 2>&1
$ wc -l afile bfile findErr findLOG
6 afile
12 bfile
```

```
2969 findErr
2981 findLOG
5968 total
```

管道是将数据流在程序（命令）间进行传递的一种机制，即将一个命令的输出做为另一个命令的输入。

```
$ cat findErr | head -5
find:  ‘/lost+found’ : Permission denied
find:  ‘/mnt/public/lost+found’ : Permission denied
find:  ‘/mnt/share/temp’ : Permission denied
find:  ‘/run/sudo’ : Permission denied
find:  ‘/run/svnserve’ : Permission denied
$ cat findErr | head -5| tail -2
find:  ‘/run/sudo’ : Permission denied
find:  ‘/run/svnserve’ : Permission denied
$ cat findLOG | grep -v "denied"
/usr/src/kernels/4.1.13-100.fc21.x86_64/include/config/kasan/shadow
/usr/src/debug/glibc-2.20/shadow
/usr/share/texlive/texmf-dist/tex/latex/shadow
/usr/share/texlive/texmf-dist/doc/latex/shadow
/etc/shadow
$ cat findLOG | grep "denied"|head -2
find:  ‘/lost+found’ : Permission denied
find:  ‘/mnt/public/lost+found’ : Permission denied
```

Chapter 2

ShellScript

2.1 目的

1. 理解 shell script 的意义
2. 掌握 shell script 的写法

2.2 目标

1. 写一脚本，计算你下一次的生日距离今天还有多少天。
2. 写一脚本，将服务器上所有的用户名显示出来。(按如下格式显示)

```
the 802 line is:1400170159

the 803 line is:1500170146

the 804 line is:1500170150
```

2.3 实验过程

2.3.1 计算生日

分析

本题目需要使用到 date 命令来获取当前的时间，并且需要输入生日的日期。

难点

日期差的计算。

解决方法

计算机中的时间表示是从 1970-01-01 00:00:00 开始计算，到目前为止经过了多长时间，可以借助这个知识来计算。

```
#!bin/bash
#calculate the day to next birthday
read -p"please input your birthday:(mmdd/0102)" birthday
year=`date +%Y`
yb=${year}${birthday}
now=`date +%Y%m%d`
nowSecond=`date +%s`
birthdaySecond=`date +%s -d "${yb}"`
bts=$((${birthdaySecond}/60/60/24))
nts=$((${nowSecond}/60/60/24))
echo "birthday to 19700101 is ${bts} days."
echo "today to 19700101 is ${nts} days."
if [ "${bts}" == "${nts}" ]; then
echo "Today is your birthday.Happy birthday to you!"
elif [ "${bts}" -gt "${nts}" ]; then
day=$((${bts}-${nts}))
echo "Your birthday is after ${day} days."
else
day=$((365-$((${nts}-${bts}))))
echo "Your birthday is after about ${day} days."
fi
```

Figure 2.1: 计算生日

代码

此问题的脚本如 Figure 2.1所示。

2.3.2 输出系统中的用户

分析

要完成任务，需要知道当前系统中的用户名存储在哪个文件中，还要知道如何按要求输出。

难点

获取特定字段。

解决方法

可用 cut 命令截取要求的字段输出。(awk 也可以。)

代码

此问题的脚本如 Figure 2.2所示。

```
#!/bin/bash
#list all the user of this system.
i=1
for user in `cat /etc/passwd|cut -d: -f1`
do
echo "the ${i} line is:${user}"
i=$((i+1))
done
```

Figure 2.2: 计算生日

Chapter 3

Makefile

3.1 目的

1. 熟悉 make 的使用
2. 熟悉 makefile 文件的写法

3.2 目标

按以下要求进行编程练习：

1. 写 2 个 cpp 文件：prog.cpp，aux.cpp。
2. 写 1 个 h 文件：aux.h。
3. aux.h 头文件定义函数 Max 和 Min，它们分别计算四个数（参数）的最大值和最小值；aux.cpp 实现这两个函数。
4. prog.c 中定义主函数，循环输入 4 个随机数，输出他们的最大和最小值。
5. 编译该项目，调试、跟踪程序执行过程；并在控制台界面运行编译的可执行文件。
6. 编写类似功能的 c 语言程序也可。

3.3 实验过程

3.3.1 定义头文件

头文件 aux.h 代码如 Figure 3.1所示。

3.3.2 实现头文件

实现头文件的 aux.c 代码如 Figure 3.2所示。

3.3.3 编写主程序

主程序 prog.c 代码如 Figure 3.3所示。

```
#ifndef _AUX_H_
#define _AUX_H_

int Max(int a,int b,int c,int d);
int Min(int a,int b,int c,int d);

#endif
```

Figure 3.1: 定义头文件

```
#include <stdio.h>
#include "aux.h"

int Mx(int a, int b){
 if(a>b){
  return a;
 }else{
  return b;
 }
}
int Max(int a,int b,int c,int d){
 return Mx(Mx(a,b),Mx(c,d));
}
int Mn(int a, int b){
 if(a>b){
  return b;
 }else{
  return a;
 }
}
int Min(int a,int b,int c,int d){
 return Mn(Mn(a,b),Mn(c,d));
}
```

Figure 3.2: 实现头文件

```
#include <stdio.h>
#include <stdlib.h>

#include "aux.h"

int main(){
 int i=0;
 for(i=0;i<4;i++){
  int a=rand();
  int b=rand();
  int c=rand();
  int d=rand();
  printf("the max number of %d,%d,%d,%d, is:%d\n",a,b,c,d,Max(a,b,c,d));
  printf("the min number of %d,%d,%d,%d, is:%d\n",a,b,c,d,Min(a,b,c,d));
 }
}
```

Figure 3.3: 主程序

```
prog:prog.c aux.c
  gcc -o prog prog.c aux.c -I.
```

Figure 3.4: simple makefile

3.3.4 编写 makefile 文件

一个简单的 makefile 文件如 Figure 3.4所示。Figure 3.4所示是最简单的 makefile 文件写法，但是存在一个问题，就是头文件编辑后，执行 make 不会重新编译。为解决这个问题，重新编写如 Figure 3.5所示的 makefile 文件。

参考资料

1. http://www.gnu.org/software/make/manual/make.html
2. http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/

```
CC=gcc
CFLAGS=-I.
DEPS=aux.h
OBJ=prog.o aux.o

%.o:%.c $(DEPS)
  $(CC) -c -o $@ $< $(CFLAGS)
prog:$(OBJ)
  $(CC) -o $@ $^ $(CFLAGS)
clean:
  rm *.o
```

Figure 3.5: a smart makefile

Chapter 4

File

4.1 目的

熟悉 linux 中与文件操作相关的系统调用和标准 I/O 库。

4.2 目标

编写两个程序，分别使用系统调用和标准 I/O 库实现文件的拷贝。

4.3 实验过程

4.3.1 准备知识

用户程序、库函数、系统调用与内核之间的关系

用户程序、库函数、系统调用与内核之间的关系如 Figure 4.1所示。

相关的系统调用

open

```
OPEN(2)                    BSD System Calls Manual

NAME
     open, openat — open or create a file for reading or writing

SYNOPSIS
     #include <fcntl.h>

     int
     open(const char *path, int oflag, ...);

     The flags specified for the oflag argument are formed by or'ing
     the following values:

          O_RDONLY          open for reading only
```

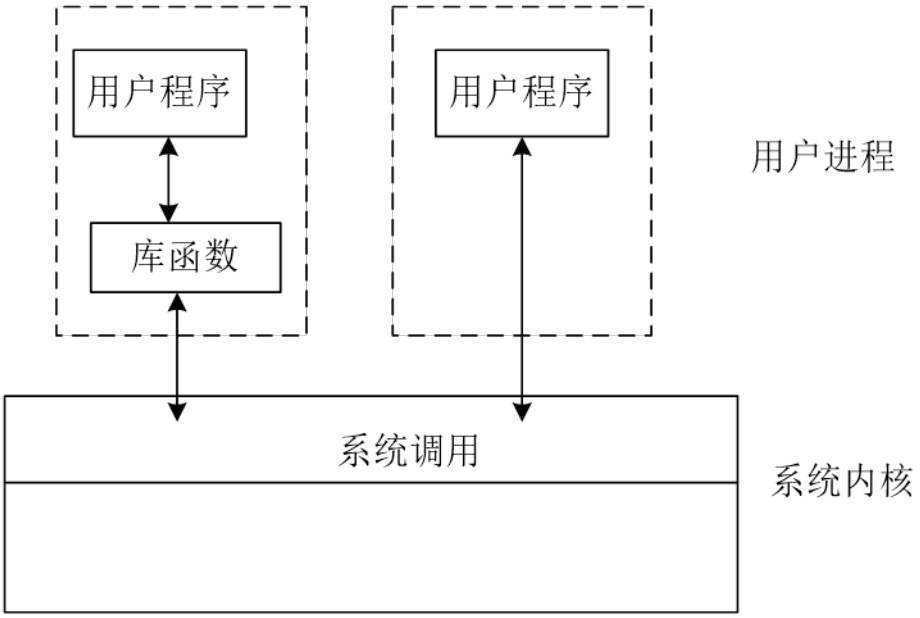

Figure 4.1: 用户程序、库函数、系统调用与内核之间的关系

```
O_WRONLY        open for writing only
O_RDWR          open for reading and writing
O_NONBLOCK      do not block on open or for data
                        to become available
O_APPEND        append on each write
O_CREAT         create file if it does not exist
O_TRUNC         truncate size to 0
O_EXCL          error if O_CREAT and the file exists
```

read

```
READ(2)                  BSD System Calls Manual

NAME
     pread, read, readv -- read input

LIBRARY
     Standard C Library (libc, -lc)

SYNOPSIS
     #include <sys/types.h>
     #include <sys/uio.h>
     #include <unistd.h>

     ssize_t
     read(int fildes, void *buf, size_t nbyte);
```

write

```
WRITE(2)                    BSD System Calls Manual

NAME
     pwrite, write, writev -- write output

LIBRARY
     Standard C Library (libc, -lc)

SYNOPSIS
     #include <unistd.h>



     ssize_t
     write(int fildes, const void *buf, size_t nbyte);
```

close

```
CLOSE(2)                    BSD System Calls Manual

NAME
     close -- delete a descriptor

SYNOPSIS
     #include <unistd.h>

     int
     close(int fildes);
```

相关的库函数

fopen

```
FOPEN(3)                BSD Library Functions Manual

NAME
     fopen, fdopen, freopen, fmemopen -- stream open functions

LIBRARY
     Standard C Library (libc, -lc)

SYNOPSIS
     #include <stdio.h>

     FILE *
     fopen(const char * restrict path, const char * restrict mode);

The argument mode points to a string beginning with
one of the following letters:

     ``r''   Open for reading.  The stream is positioned at
```

```
        the beginning of the file.  Fail if the file does not exist.

'w'     Open for writing.  The stream is positioned at
        the beginning of the file.  Create the file if it does not ex

'a'     Open for writing.  The stream is positioned at
        the end of the file.  Subsequent writes to the file
        will always end up at the then current end of file,
        irrespective of any intervening fseek(3) or similar.
        Create the file if it does not exist.

An optional '+' following 'r', 'w', or 'a' opens the file
for both reading and writing.
```

fread、fwrite

```
FREAD(3)                BSD Library Functions Manual

NAME
     fread, fwrite -- binary stream input/output

LIBRARY
     Standard C Library (libc, -lc)

SYNOPSIS
     #include <stdio.h>

     size_t
     fread(void *restrict ptr, size_t size, size_t nitems,
           FILE *restrict stream);

     size_t
     fwrite(const void *restrict ptr, size_t size, size_t nitems,
           FILE *restrict stream);

DESCRIPTION
     The function fread() reads nitems objects, each size bytes long,
     from the stream pointed to by stream, storing them at the location
     given by ptr.

     The function fwrite() writes nitems objects, each size bytes long,
     to the stream pointed to by stream, obtaining them from the location
     given by ptr.
```

fclose

```
FCLOSE(3)               BSD Library Functions Manual

NAME
     fclose, fcloseall -- close a stream

LIBRARY
```

```
     Standard C Library (libc, -lc)

SYNOPSIS
     #include <stdio.h>

     int
     fclose(FILE *stream);
```

fgetc

```
GETC(3)                  BSD Library Functions Manual

NAME
     fgetc, getc, getc_unlocked, getchar, getchar_unlocked, getw
       -- get next character or word from input stream

LIBRARY
     Standard C Library (libc, -lc)

SYNOPSIS
     #include <stdio.h>

     int
     fgetc(FILE *stream);

     int
     getc(FILE *stream);

     int
     getc_unlocked(FILE *stream);

     int
     getchar(void);

     int
     getchar_unlocked(void);

     int
     getw(FILE *stream);

DESCRIPTION
     The fgetc() function obtains the next input character (if present) from
          the stream pointed at by stream, or the next character
          pushed back on the stream via ungetc(3).

     The getc() function acts essentially identically to fgetc(),
         but is a macro that expands in-line.

     The getchar() function is equivalent to getc(stdin).
```

fputc

```
PUTC(3)                    BSD Library Functions Manual

NAME
     fputc, putc, putc_unlocked, putchar, putchar_unlocked, putw
      -- output a character or word to a stream

LIBRARY
     Standard C Library (libc, -lc)

SYNOPSIS
     #include <stdio.h>

     int
     fputc(int c, FILE *stream);

     int
     putc(int c, FILE *stream);

     int
     putc_unlocked(int c, FILE *stream);

     int
     putchar(int c);

     int
     putchar_unlocked(int c);

     int
     putw(int w, FILE *stream);

DESCRIPTION
     The fputc() function writes the character c
       (converted to an ``unsigned char'') to the output stream
        pointed to by stream.

     The putc() macro acts essentially identically to fputc(),
     but is a macro that expands in-line.  It may evaluate stream
     more than once, so arguments given to putc() should not be
      expressions with potential side effects.

     The putchar() function is identical to putc() with an output
        stream of stdout.
```

4.3.2 使用系统调用实现文件拷贝

使用系统调用逐个字符地拷贝文件，代码如 Figure 4.2所示。

```
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>

int main(){
        char c;
        int in,out;

        in = open("simple_read.c",O_RDONLY);
        out = open("copy_example",O_WRONLY|O_CREAT,S_IRWXU);
        while(read(in,&c,1) ==1 ){
                write(out,&c,1);
                write(1,&c,1);
        }
        close(in);
        close(out);
        exit(0);
}
```

Figure 4.2: 系统调用逐个字符拷贝

4.3.3 使用标准 I/O 库实现文件拷贝

使用 fread 和 fwrite 函数实现

使用 fread 和 fwrite 函数实现文件的拷贝，代码如 Figure 4.3所示，注意本程序有 bug，请尝试发现并改正[1]。

使用 getc 和 putc 函数实现

使用 getc 和 putc 函数实现文件拷贝的代码如 Figure 4.4所示。

[1]参考 fseek 和 ftell 函数

```
#include <stdio.h>
#include <stdlib.h>
//fread fwrite have problems,
//the final part cannot be
//written to the file.
int main(){
        int c;
        FILE *in, *out;
        char s[20];

        in = fopen("simple_read.c","r");
        out = fopen("io_copy","w+");
        c = fread(s,10,1,in);

        while(c>0){
                printf("%s",s);
                fwrite(s,10,c,out);
                c = fread(s,10,1,in);

        }
        fclose(in);
        fclose(out);
        exit(0);
}
```

Figure 4.3: 使用 fread 和 fwrite 函数拷贝文件

```
#include <stdio.h>
#include <stdlib.h>

int main(){
        int c;
        FILE *in;
        FILE *out;

        in=fopen("simple_read.c","r");
        out=fopen("io_copy_getcputc","w");
        while((c=fgetc(in))!=EOF){
                fputc(c,out);
        }
        fclose(in);
        fclose(out);
        exit(0);
}
```

Figure 4.4: 使用 getc 和 putc 函数拷贝文件

Chapter 5

Curses

5.1 目的

熟悉 linux 中Curses库的使用，巩固标准 I/O 库的读文件操作。

5.2 目标

编写一个 vimlike 程序，要求至少能够读取并显示文件内容。

5.3 实验过程

5.3.1 准备知识

相关的函数

本次实验可能要用到的 curses 库函数有：

- 窗口相关
  initscr, subwin, newwin, newpad, subpad, box, delwin, endwin
- 字符串相关
  getch, addch, getstr, addstr

对于这些函数的具体用法，请使用 *man 3 xxx* 来获取帮助。

5.3.2 实现

构造主界面

构造用户打开程序后的界面，代码如 Figure 5.1所示，这段代码做了这些事情：

1. 构造了一个父窗口 fwin
2. 为文件名 fileName 开辟了一块内存空间
3. 为本程序的所有窗口设置了背景色和前景色

```
#include<unistd.h>
#include<curses.h>
#include<stdlib.h>

void readAfile(char * fileName, WINDOW * win);
int main(){
WINDOW * fwin;
WINDOW * swin;
char * fileName;
fileName = (char *) malloc(32);
fwin=initscr();
//add some colors
if(has_colors()){
 start_color();
 init_pair(1,COLOR_GREEN,COLOR_BLACK);
 attron(COLOR_PAIR(1));
}
//give some tips
addstr("choose an option:\nO for open a file\nQ for quit");
```

Figure 5.1: 构造主界面

判断用户按键并作出相应的操作

判断用户按键，代码如 Figure 5.2所示，这段代码做了这些事情:

1. 如果用户输入了 Q 键，直接退出程序
2. 如果用户输入了 O 键，则打开一个子窗口，并接收用户输入
3. 打开用户在子窗口输入的文件名，并将文件内容显示到父窗口中

函数—读取指定的文件并显示到父窗口

使用 readAfile 函数读取指定文件，并将文件内容显示到父窗口，代码如 Figure 5.3所示。

```
//detect the keypress
while(1){
 int key=getch();
 char key_press=(char)key;
 switch(key_press){
  case 'Q':
   endwin();
   exit(0);
  case 'O':
   swin = subwin(fwin,3,15,20,30);
   box(swin,'#','#');
   wmove(swin,1,2);
   wgetstr(swin,fileName);
   wrefresh(swin);
   readAfile(fileName,fwin);
   delwin(swin);
 }
}
```

Figure 5.2: 判断用户按键并执行相应的操作

```
void readAfile(char * fileName,WINDOW * win){
        FILE * file = fopen(fileName,"r");
        int ch= fgetc(file);
        while(ch!=EOF){
                waddch(win,(char)ch);
                wrefresh(win);
                ch= fgetc(file);
        }
        fclose(file);
}
```

Figure 5.3: 读取文件并显示

Chapter 6

Process

6.1 目的

掌握在一个进程中创建另一个进程的方法。

6.2 目标

编写一个程序，并在这个程序中创建一个新的进程，在父进程中每秒输出一个字符，同时子进程也每秒输出一个字符。

6.3 实验过程

6.3.1 准备知识

创建子进程的方式

在一个进程中创建一个子进程可以有三种方式：

1. system
   其执行过程如 Figure 6.1所示。
2. exec
   其执行过程如 Figure 6.2所示。
3. exec+fork
   其执行过程如 Figure 6.3所示。

6.3.2 使用 fork 的方式完成本实验

请过 man 3 fork 来查看相关函数原型。本实验的一个参考代码如 Figure 6.4所示。

6.3.3 DEBUG

What!? Figure 6.4所示代码与我们期待的结果竟然不一样？这是为什么呢？请大家一起来查找问题的原因所在吧。

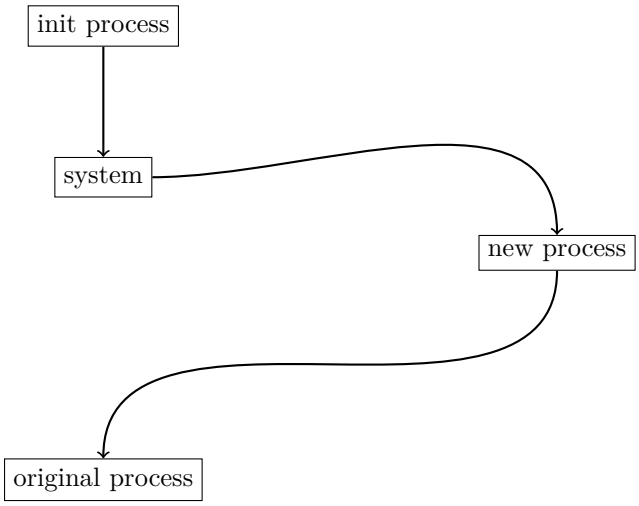

Figure 6.1: system 图示

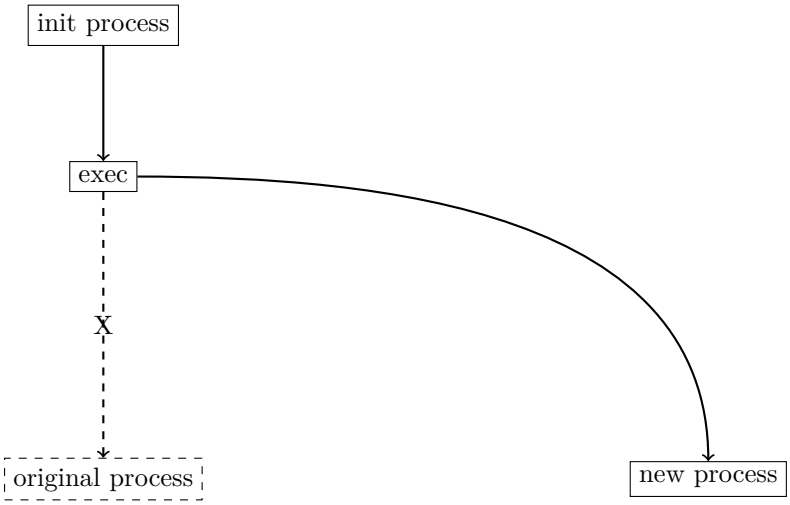

Figure 6.2: exec 图示

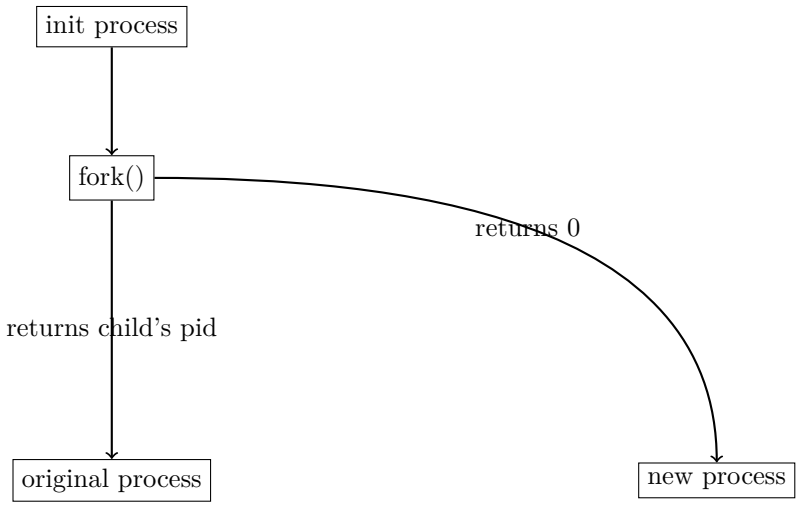

Figure 6.3: fork 图示

```
#include<unistd.h>
#include<stdlib.h>
#include<stdio.h>

void fmt(char st, char ed);
int main(){

printf("I will fork a progress.\n");
pid_t pid=fork();
switch(pid){
case -1:
printf("error!");
break;
case 0:
//printf("I am child.\n");
fmt('a','z');
break;
default:
//printf("I am parent.\n");
fmt('A','Z');
break;

}
if(pid!=0){
int stat;
pid_t child;
child=wait(&stat);
printf("\nthe child(pid=%d) has finished.\n",child);
if(WIFEXITED(stat)!=0){
printf("child exit with code:%d\n",WEXITSTATUS(stat));
}else{
printf("child terminated abnormally.\n");
}
}
exit(0);
}

void fmt(char st, char ed){
for(char a=st; a<=ed;a++){
fputc(a,stdout);
//
sleep(1);
}
}
```

Figure 6.4: 非严格意义的交替输出