第十章：文件系统

生命科学学院

一、文件输入/输出函数

- 1、基本I/O 函数

一些I/O 函数在前面的章节中已有讲述， 如

- open： 允许程序访问文件
- close： 终止文件访问
- print： 文件写入字符串
- write： 向文件写入格式化信息
- printf： 格式化字符串并输出到文件

1)open 函数

open 函数将文件变量与某文件联系起来，提供访问文件的接口，例如：

open(MYVAR, "/u/file")；如果文件打开成功，则返回非零值，否则返回零。缺省地，open 打开文件用以读取其内容，若想打开文件以写入内容，则在文件名前加个大于号：open(MYVAR, ">/u/file")；向已有的文件末尾添加内容用两个大于号：open(MYVAR, ">>/u/file")；若想打开文件作为数据导向的命令，则在命令前加上管道符(|)：open(MAIL, "|mail dave");

2)用open 重定向输入

可以把打开的文件句柄用作向程序输入数据的命令，方法是在命令后加管道符(|)，如:

```
open(CAT, "cat file*|");
```

对open 的调用运行命令cat file* ，此命令创建一个临时文件，这个文件的内容是所有以file 打头的文件的内容连接而成，此文件看作输入文件，可用文件变量CAT 访问，如:

```
$input = ;
```

下面的例子使用命令w 的输出来列出当前登录的所有用户名。

```
1 : # ! /usr/local /bin/perl
2 :
3 : open (WOUT, "w|");
4 : $time = <WOUT>;
5 : $time = ~ s / ^ * / / ;
6 : $time = ~ s / .* / / ;
7 : ; # skip headings line
8 : @users = ;
9 : close (WOUT);
10: foreach $user (@users) {
11: $user = ~ s / .* / / ;
12: }
13: print ("Current time: $time " ) ;
14: print ("Users logged on:\n");
15: $prevuser = "";
16: foreach $user (sort @users) {
17: if ($user ne $prevuser) {
18: print ("\t$user");
19: $prevuser = $user;
20: }
21: }
```

结果输出如下：

```
Current time: 4:25pm
Users logged on:
dave
kilroy
root
zarquon
```

w 命令列出当前时间、系统负载和登录的用户，以及每个用户的作业时间和当前运行的命令，如：

```
          4:25pm  up 1 day,  6:37,  6 users,  load
average: 0.79, 0.36, 0.28
          User      tty         login@  idle    JCPU
PCPU what
          dave      ttyp0       2:26pm             27
3 w
          kilroy    ttyp1       9:01am  2:27    1:04
11 -csh
          kilroy    ttyp2       9:02am    43    1:46
27 rn
          root      ttyp3       4:22pm     2
-csh
          zarquon   ttyp4       1:26pm     4       43
16 cc myprog.c
          kilroy    ttyp5       9:03am           2:14
48 /usr/games/hack
```

上例中从w 命令的输出中取出所需的信息：当前时间和登录的用户名。
第3 行运行w 命令，此处对open 的调用指定w 的输出用作程序的输入，用文件变量WOUT 来访问该输入。第4 行读取第一行信息，即：
4:25pm up 1 day, 6:37, 6 users, load average: 0.79, 0.36, 0.28
接下来的两行从这行中抽取出时间。首先，第5 行删除起始的空格，然后第6行删去除时间和结尾换行符之间的所有字符，存入变量$time。
第7 行从WOUT 读取第二行，这行中无有用信息，故不作处理。第8 行把剩下的行赋给数组@users，然后第9 行关闭WOUT，终止运行w 命令的进程。@users 中的每个元素都是一行用户信息，因为本程序只需要每行的第一个单词，即用户名，故10~12 行去掉除换行符外的其它字符，这一循环结束后，@users中只剩下用户名的列表。
第13 行输出存贮在$time 中的时间，注意这时print 不需要加上换行符，因为$time 中有。16~21 行对@users 中的用户名排序并输出。因为同一个用户可以多次登录，所以用$preuser 存贮输出的最后一个用户名，下次输出数组元素$user时，如果其与$preser 相等，则不输出。

3)文件重定向

许多UNIX shell 可以把标准输出文件(STDOUT)和标准错误文件(STDERR)都重定向到同一个文件，例如在Bourne Shell (sh) 中，命令

```
$ foo > file1 2>&1
```

运行命令foo 并把输出到标准输出文件和标准错误文件的内容存贮到文件file1中。下面是用Perl 实现这一功能的例子：

```
1: #!/usr /local/bin/ p e r l
2:
3: open (STDOUT, ">file1") || die ("open STDOUT failed");
4: open (STDERR, ">&STDOUT") || die ("open STDERR f a i l e d " ) ;
5 : print STDOUT ("line 1\n") ;
6 : print STDERR ("line 2\n");
7: close (STDOUT);
8: close (STDERR);
```

运行后，文件file1 中的内容为：

line 2
line 1

可以看到，这两行并未按我们想象的顺序存贮，为什么呢？我们来分析一下这段程序。

第3 行重定向标准输出文件，方法是打开文件file1 将它与文件变量STDOUT关联，这也关闭了标准输出文件。第4 行重定向标准错误文件，参数>&STDOUT告诉Perl 解释器使用已打开并与STDOUT 关联的文件，即文件变量STDERR 指向与STDOUT 相同的文件。第5、6 行分别向STDOUT 和STDERR 写入数据，因为这两个文件变量指向同一个文件，故两行字符串均写到文件file1 中，但顺序却是错误的，怎么回事呢？

问题在于UNIX 对输出的处理上。当使用print（或其它函数）写入STDOUT等文件时，UNIX 操作系统真正所做的是把数据拷贝到一片特殊的内存即缓冲区中，接下来的输出操作继续写入缓冲区直到写满，当缓冲区满了，就把全部数据实际输出。象这样先写入缓冲区再把整个缓冲区的内容输出比每次都实际输出所花费的时间要少得多，因为一般来说，I/O 比内存操作慢得多。
程序结束时，任何非空的缓冲区都被输出，然而，系统为STDOUT 和STDERR分别维护一片缓冲区，并且先输出STDERR 的内容，因此存贮在STDERR 的缓冲区中的内容line 2 出现在存贮在STDOUT 的缓冲区中的内容line 1 之前。
为了解决这个问题，可以告诉Perl 解释器不对文件使用缓冲，方法为：
1、用select 函数选择文件
2、把值1 赋给系统变量$|

系统变量$|指定文件是否进行缓冲而不管其是否应该使用缓冲。如果$|为非零值则不使用缓冲。$|与系统变量$~和$^协同工作，当未调用select 函数时，$|影响当前缺省文件。下例保证了输出的次序：

```
1 : # ! /usr/local / bin/perl
2 :
3 : open (STDOUT, ">file1") || die ("open STDOUT failed");
4 : open (STDERR, ">&STDOUT") || die ("open STDERR failed");
5 : $ | = 1 ;
6 : select (STDERR);
7 : $ | = 1 ;
8 : print STDOUT ("line 1\n");
9 : print STDERR ("line 2\n");
10: close (STDOUT);
11: close (STDERR);
```

程序运行后， 文件file 1 中内容为：

line 1

line 2

第5 行将$ |赋成1 ，告诉Perl 解释器当前缺省文件不进行缓冲，因为未调用select， 当前的缺省文件为重定向到文件file1 的STDOUT。第6 行将当前缺省文件设为STDERR， 第7 行又设置$ | 为1， 关掉了重定向到file1 的标准错误文件的缓冲。由于STDOUT 和STDERR 的缓冲均被关掉，向其的输出立刻被写到文件中， 因此line 1 出现在第一行。

4)指定读写权限

打开一个既可读又可写的文件方法是在文件名前加上"+>"，如下：

```
open (READWRITE, "+>file1");
```

此语句打开既可读又可写的文件file1， 即可以重写其中的内容。文件读写操作最好与库函数seek 和tell 一起使用， 这样可以跳到文件任何一点。

注： 也可用前缀"+<"指定可读写权限。

5)close 函数

用于关闭打开的文件。当用close 关闭管道，即重定向的命令时， 程序等待重定向的命令结束， 如:

```
open (MYPIPE, "cat file*|");
close (MYPIPE);
```

当关闭此文件变量时，程序暂停运行，直到命令cat file *运行完毕。

6)print , printf 和write 函数

- print是这三个函数中最简单的， 它向指定的文件输出， 如果未指定， 则输出到当前缺省文件中， 如:
- print ("Hello, there!\n");
- print OUTFILE ("Hello, there !\n") ;
- 第一句输出到当前缺省文件中， 若未调用select，则为STDOUT。第二句输出到由文件变量OUTFILE 指定的文件中。

printf函数先格式化字符串再输出到指定文件或当前缺省文
件中， 如：
printf OUTFILE (“You owe me %8.2f", $owing);
此语句取出变量$owing 的值并替换掉串中的%8.2f， %8.2f 是域格式的例子， 把$owing 的值看作浮点数。
write函数使用输出格式把信息输出到文件中， 如：
select (OUTFILE);
$~ = "MYFORMAT";
write;
关于printf和write， 详见《第x 章 格式化输出》.

7)select 函数

- select 函数将通过参数传递的文件变量指定为新的当前缺省文件， 如:
- select (MYFILE) ;
- 这样，MYFILE 就成了当前缺省文件，当对print 、write 和printf的调用未指定文件时， 就输出到MYFILE 中。

8)eof 函数

- eof 函数查看最后一次读文件操作是否为文件最后一个记录，
- 如果是， 则返回非零值， 如果文件还有内容， 返回零。
- 一般情况下，对eof 的调用不加括号，因为eof 和eof()是等效
- 的，但与<>操作符一起使用时，eof 和eof()就不同了。现在我们来创建两个文件， 分别叫做file 1 和file2。file1 的内容为:
- This is a line from the first file .
- Here is the last line of the first file.
- file2的内容为:
- This is a line from the second and last file.
- Here is the last line of the last file .

下面就来看一下eof 和eof( )的区别， 第一个程序为：

```
1: #!/usr/local/bin/perl
2:
3: while ($line = <>) {
4:     print ($line);
5:     if (eof) {
6:         print ("-- end of current file --\n");
7:     }
8: }
```

运行结果如下：

```
$ program file1 file2
This is a line from the first file.
Here is the last line of the first file.
-- end of current file --
This is a line from the second and last file.
Here is the last line of the last file.
-- end of current file --
```

下面把eof 改为eof()， 第二个程序为：

```
1: #!/usr/local/bin/perl
2:
3: while ($line = <>) {
4:     print ($line);
5:     if (eof()) {
6:         print ("-- end of output --\n");
7:     }
8: }
```

这时， 只有所有文件都读过了， eof()才返回真，如果只是多个文件中前几个的末尾，返回值为假，因为还有要读取的输入。

运行结果如下：

```
$ program file1 file2
This is a line from the first file.
Here is the last line of the first file.
This is a line from the second and last file.
Here is the last line of the last file.
-- end of output --$
```

9)间接文件变量

- 对于上述各函数open ， close ， print ， printf ， write ， select 和eof，都可以用简单变量来代替文件变量，这时，简单变量中所存贮的字符串就被看作文件变量名，下面就是这样一个例子，此例很简单，就不解释了。需要指出的是， 函数open，close，write，select和eof 还允许用表达式来替代文件变量，表达式的值必须是字符串， 被用作文件变量名。

```
1: #!/usr /local/bin/ p e r l
3: &open_file("INFILE", "", "file1");
4: &open_file("OUTFILE", ">", "file2");
5: while ($line = &read_from_fi l e ( " INFILE")) {
6: &print_to_file("OUTFILE", $line);
7 : }
8:
9: sub open_file {
10: local ($filevar, $filemode, $filename) = @_;
11:
12: open ($filevar, $filemode . $filename) | |
13: die ("Can't open $filename" ) ;
1 4 : }
15: sub read_from_file {
16: local ($filevar) = @_;
17:
18: <$filevar>;
19: }
20: sub print_to_file {
21: local ($filevar, $line) = @_;
22:
23: p r i n t $ f i l e v a r ( $ l i n e ) ;
24: }
```