第5章 进程、重定向和管道

本章内容

- 描述进程的概念和单CPU计算机系统执行多个进程的过程。
- 说明shell如何执行命令
- 讨论进程的属性
- 解释前台和后台进程的概念并解释什么是守护进程
- 描述命令的串行执行和并行执行过程
- 讨论LINUX下进程和作业的控制：前台和后台进程，串行和并行进程，进程挂起，前台和后台进程的相互转换，以及进程终止。
- 描述LINUX进程的层次关系
- 介绍一些命令和简单操作<Ctrl-C> 、 <Ctrl-D>、<Ctrl-Z>、 、 ; 、&、( )、bg、jobs、kill、ps、pstree、sleep、top

本章内容

- 描述标准文件的概念——标准输入、标准输出和标准出错文件，以及文件描述符
- 描述标准文件的输入输出重定向
- 讨论出错重定向和追加输出文件的概念
- 解释LINUX中管道的概念
- 描述管道、文件描述符、重定向原语所实现的强大功能
- 涉及下面的命令和原语：&，|，<，>，>>，cat，diff，grep，lp，more，pr，sort，stderr，stdin，stdout，tee，tr，uniq，wc

5.1 进程

什么是进程

- An executing program = a *process*
- 每当你执行一个外部命令时，Linux系统就会为之创建一个进程，
- 命令执行完毕后系统就撤销它。
- 进程创建和终止是Linux系统处理外部命令所采用的唯一机制。

Shell命令的执行

什么是内部、外部命令？

- Shell 命令可以是内部或者外部命令。
- 内部（内置）命令（*internal (built-in) command*）的代码本身就是shell进程的一部分。LINUX shell中的一些内部命令如.（点命令）、alias、bg、cd 、continue、echo 、exec 、exit 、fg 、jobs 、pwd 、set 、shift 、test 、time 、umask 、unset和wait。
- 外部命令是(*external command*)命令代码以文件的形式出现的称为；文件内容可以是二进制代码或者shell脚本。通常使用的一些外部命令如grep、more 、cat 、mkdir 、rmdir 、ls 、sort 、ftp 、telnet 、lp和ps 。
- LINUX进程通过系统调用fork创建另外一个进程，该操作建立原进程内存的完全拷贝。这两个进程接着fork后的语句执行。
- The forking process is known as the *parent process*(*父进程*)
- The created (forked) process is called the *child process*(*子进程*)

shell命令的执行(cont.)

- 为了执行一个外部二进制命令，需要一种机制使子进程成为要执行的命令。
- LINUX系统调用exec完成这项工作，进程可以用另外一个命令的可执行代码来覆盖自身。
- shell先后使用fork和exec系统调用来执行一个外部二进制命令。
- *例：fork.c*

ps 命令

shell命令的执行(cont.)

```
% ps
PID    TTY      TIME     CMD
12675  pts/0    0:03     tcsh
12695  pts/0    0:03     ps
% bash
$ echo This is Bourne shell
This is Bourne shell.
$ zsh
[~]-% echo This is Korn shell.
This is Korn shell.
[~]-% ps
PID    TTY      TIME CMD
12675  pts/0    0:03     tcsh
15772  pts/0    0:00     bash
15773  pts/0    0:00     zsh
12695  pts/0    0:03     ps
[~]-% ^D
[~]-% ^D
%
```

tcsh提示符

bash提示符

zsh提示符

进程属性

- Linux系统每一个进程都具有一些属性，包括:
  - 所有者的ID
  - 进程名
  - 进程ID（PID）
  - 进程状态
  - 父进程的ID
  - 进程已执行的时间
  - …

进程属性(cont.)

- ps命令用来查看系统中运行的进程的各种属性
- Linux系统中使用的GNU ps命令支持3种不同类型的命令行参数：
  - Unix风格的参数，**前面加单破折线；**
  - BSD风格的参数，**前面不加破折线；**
  - GNU风格的长参数，**前面加双破折线。**

Unix风格的ps命令参数

参数	描述
-A	显示所有进程
-N	显示与指定参数不符的所有进程
-a	显示除控制进程（session leader①）和无终端进程外的所有进程
-d	显示除控制进程外的所有进程
-e	显示所有进程
-C *cmdlist*	显示包含在*cmdlist*列表中的进程
-G *grplist*	显示组ID在*grplist*列表中的进程
-U *userlist*	显示属主的用户ID在*userlist*列表中的进程
-g *grplist*	显示会话或组ID在*grplist*列表中的进程②
-p *pidlist*	显示PID在*pidlist*列表中的进程
-s sesslist	显示会话ID在sesslist列表中的进程
-t ttylist	显示终端ID在ttylist列表中的进程
-u userlist	显示有效用户ID在userlist列表中的进程
-F	显示更多额外输出（相对-f参数而言）
-O format	显示默认的输出列以及format列表指定的特定列
……	

BSD风格的ps命令参数

参数	描述
T	显示跟当前终端关联的所有进程
a	显示跟任意终端关联的所有进程
g	显示所有的进程，包括控制进程
r	仅显示运行中的进程
x	显示所有的进程，甚至包括未分配任何终端的进程
U userlist	显示归userlist列表中某用户ID所有的进程
p pidlist	显示PID在pidlist列表中的进程
t ttylist	显示所关联的终端在ttylist列表中的进程
O format	除了默认输出的列之外，还输出由format指定的列
X	按过去的Linux i386寄存器格式显示
Z	将安全信息添加到输出中
j	显示任务信息
l	采用长模式
o format	仅显示由format指定的列
s	采用信号格式显示
u	采用基于用户的格式显示

GNU风格的ps命令参数

参数	描 述
--deselect	显示所有进程，命令行中列出的进程
--Group grplist	显示组ID在grplist列表中的进程
--User userlist	显示用户ID在userlist列表中的进程
--group grplist	显示有效组ID在grplist列表中的进程
--pid pidlist	显示PID在pidlist列表中的进程
--ppid pidlist	显示父PID在pidlist列表中的进程
--sid sidlist	显示会话ID在sidlist列表中的进程
--tty ttylist	显示终端设备号在ttylist列表中的进程
--user userlist	显示有效用户ID在userlist列表中的进程
--format format	仅显示由format指定的列
--context	显示额外的安全信息
--cols n	将屏幕宽度设置为n列
--columns n	将屏幕宽度设置为n列

.....

进程属性(cont.)

- Examples:
  - ps -l (Display in long format)
  - ps -A |more (Display all processes running on that system)
  - ps -j (Display the parent PID, group ID, session ID
  - ps -u *userlist*(Display processes owned by specific users)
  - ps r (Show processes that are in runnable state)
  - ps f (Display process hierarchy in tree view)

进程属性(cont.)

- $ ps

```
PID      TTY        TIME       CMD
19440  pts/1       0:02       -ksh
18786  pts/1       0:00        ps
20668  pts/1       0:10        vi
```

- $ ps –al //下面是部分内容

```
PID  TTY     STAT     TIME    COMMAND
  7627  pts/1     S        0:00    login – ji
  7628  pts/1     S        0:00    -tcsh
  7666  pts/1     S        0:00    bash
  7831  pts/1     R        0:00    ps a
```

- $ ps --forest

```
 PID TTY          TIME CMD
2087 pts/18   00:00:00 bash
2909 pts/18   00:00:00  \_ ps
```

进程属性(cont.)

- $ ps -u 147

```
UID   PID      TTY      TIME     CMD
147   18802    pts/1    0:00      ps
147   19440    pts/1    0:02     ksh
147   20668    pts/1    0:10     vi
$
```

- $ ps –l

```
 F   UID   PID   PPID  PRI  NI  VSZ   RSS WCHAN  STAT TTY  TIME  COMMAND
100 121  7628  7627   0    0   2136  1228 rt_sig   S   pts/2 0:00  -tcsh
000 121  7666  7628   10   0   2060  1244 wait4    S   pts/2 0:00  bash
000 121  7828  7666   0    0   4044  1192 do_sig   T   pts/2 0:00  pine
000 121  7829  7666   18   0   1104  292  _        R   pts/2 49:22 banner
000 121  8007  7666   17   0   2740  1012  _       R   pts/2 0:00   ps l
```

进程属性(cont.)

表	ps 1命令的各种输出域
域	含义
F	标志：与进程相关的标志。它用来指示：进程是否用户进程或者内核进程，进程为什么停止或进入休眠，等等
UID	用户ID：进程所有者的用户ID
PID	进程ID：进程的ID
PPID	父PID：父进程的PID
PRI	优先权：进程调度时要参考该优先权
NI	Nice值：进程的nice值；用于计算进程优先权的另外一个参数。
VSZ	虚拟大小：这个域中的值是按照块计算的进程（代码+数据+栈）内存映像的大小。
RSS	驻留集的大小：物理内存的大小，用千字节表示(K)；它不包括页表所占用的空间和为该进程耗费的内核任务数据结构所用的空间。
WCHAN	等待管道：对于正在运行的进程或者进程处于就绪状态并等待CPU，该域为空。对于等待或者休眠的进程，这个域显示该进程所等待的事件—进程等待在其上的内核函数。
STAT	进程状态：参考表13.1和表13.2
TTY	终端:进程属于哪个终端。
TIME	时间：到当前为止进程已经运行的时间（按照分钟和秒的格式），或者在休眠和停止之前已经运行的时间。
COMMAND	命令：显示启动该进程的命令行。使用-f选项可以看到全部的命令行；否则只有路径名的最后一项可以显示出来。

前台、后台、作业等命令

进程和作业控制

- 当键入命令并敲<Enter>后，
  - shell执行命令
  - 在当前命令执行结束即shell返回前，你不能执行任何命令。
- 这种执行命令的方式，称之为在前台foreground执行
- 需要运行一个要花费很长时间才能完成的LINUX命令(或者任一这样的程序)，当命令执行时，你无法做其它工作。
- LINUX可以让你那样做，在执行命令的同时，你可以看到shell提示符出现，然后做其他工作。这种能力称为在后台background执行命令。

进程和作业控制(cont.)

- 在命令后面加上一个“与”操作符号（&），使该命令在后台操作。例：sleep 1000&
- While a command is executed in the background,
  - The shell prompt comes back
  - Allows the user to type and run another command in the foreground
- Executing a command in the background allows the user to continue his/her work without having to wait for the shell prompt
- 前台、后台运行命令的语法：
  - 命令　（对于前台执行）
  - 命令&　（对于后台执行）

进程和作业控制(cont.)

- Example:

  $ find / -name foo –print > foo.paths 2> /dev/null &

  [1] 23467

  - 括号中shell返回的数字是该进程的作业号（job number）；另外一个数字是进程PID。
  - find 命令的作业号是 1，其PID为23467。
  - 作业是一个不运行于前台的进程，并且只能在关联的终端上访问。这样的进程通常在后台执行或者成为被挂起的进程。

进程和作业控制(cont.)

- fg命令可以使后台进程转到前台。

句法：	fg[%jobid]
用途：	使作业号为jobid的前台进程继续执行，或者把后台进程转到前台执行
%jobid经常要使用的值：	%或者%+ 当前的作业 %- 以前的作业 %N 作业号为N %Name 作业的开头名字为Name %?Name 命令中含有Name

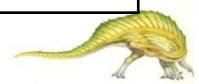
进程和作业控制(cont.)

```
$ find / -inum 23456 –print > pathnames 2 > /dev/null &
[1] 13590
$ find / -name foo –print > foo.paths 2> /dev/null &
[2] 13951
$ find / -name foobar –print > foobar.paths 2> /dev/null &
[3] 13596
$ ps
  PID    TTY     TIME    CMD
13495   pts/0    00:00:03   bash
13583   pts/0    00:00:11   find
13586   pts/0    00:00:05   find
13587   pts/0    00:00:03   find
$ fg
find / -name foobar –print > foobar.paths 2 > /dev/null
<Ctrl-C>
$ fg %2
find / -name foo –print > foo.paths 2 > /dev/null
<Ctrl-Z>     #用<Ctrl-Z>挂起一个前台进程。
$ fg %"find / -inum"
find / -inum 23456 –print > pathnames 2 > /dev/null
```

进程和作业控制(cont.)

- 使用<Ctrl-Z>挂起一个前台进程。
- 使用bg命令把被挂起的进程转到后台。
- 使用jobs命令显示所有挂起的（停止的）和后台进程的作业号，确定哪一个是当前的进程。
- 命令suspend 可以挂起当前shell进程

句法：	bg[%jobid-list]
用途：	恢复执行作业号在jobid-list中给出的那些被挂起的进程/作业
%jobid经常要使用的值：	%或者%+ 当前的作业 %- 以前的作业 %N 作业号为N %Name 作业以Name开头 %?Name 命令中含有Name



进程和作业控制(cont.)

句法：	jobs[option][%jobid-list]
用途：	显示所有在jobid-list中指明的被挂起的和后台进程的状态；如果没有列表，则显示当前进程的状态。
常用选项/功能：	-l　　也显示作业的PID

进程和作业控制(cont.)

```
$ find / -inum 23456 –print 2 > /dev/null 1 >&2
[1]+ Stopped find / -inum 23456 –print 2 >/dev/null 1>&2
$ find / -name foo > foo.paths 2 > /dev/null &
[2] 13586
$ find / -name foobar > foobar.paths 2> /dev/null &
[3] 13587
$ vi chapter13
~
~
. . .
~
“chapter13” [New File]
<Ctrl-Z>
[4]+ Stopped vi chapter13
$
$ jobs
[1] – Stopped        find / -inum 23456  -print 2 > /dev/null 1 >&2
[2]  Running        find / -name foo > foo.paths 2 > /dev/null&
[3]  Running        find / -name foobar > foobar.paths 2 > /dev/null&
[4] + Stopped        vi chapter13
```

进程和作业控制(cont.)

```
$ jobs –l
[1] – 13583 Stopped        find / -inum 23456  -print 2 > /dev/null 1 >&2
[2]  13586 Running        find / -name foo > foo.paths 2 > /dev/null&
[3]  13587 Running        find / -name foobar > foobar.paths 2 > /dev/null&
[4] + 13589 Stopped        vi chapter13
$ bg %1
[1] find /-inum 23456 –print &
$ jobs
[1] – Running        find / -inum 23456  -print 2 > /dev/null 1 >&2
[2]  Running        find / -name foo > foo.paths 2 > /dev/null&
[3]  Running        find / -name foobar > foobar.paths 2 > /dev/null&
[4] + Stopped        vi chapter13
$ bg
[4] vi chapter13&
$ jobs
[1] – Running        find / -inum 23456  -print 2 > /dev/null 1 >&2
[2]  Running        find / -name foo > foo.paths 2 > /dev/null&
[3]  Running        find / -name foobar > foobar.paths 2 > /dev/null&
[4] + Stopped        vi chapter13
```

进程和作业控制(cont.)

- at命令：在指定时刻执行指定的命令序列。
  - 命令语法：

  at [-V] [-q 队列] [-f 文件名] [-mldbv] 时间
  - 例：at -f p1 4pm + 3 days

    在三天后下午4点执行p1作业
- batch命令：指定作业在系统负载较轻的时候运行。
  - 命令格式为：

    batch -f fname

Linux守护进程(Daemons)

- 守护进程（daemon,内核线程）是运行于后台的系统进程。
- 守护进程在LINUX中经常用于向用户提供各种类型的服务和执行系统管理任务。
- Some example Linux daemons:
  - smtpd (e-mail service)
  - httpd (web browsing)
  - inetd (internet related services)

命令的顺序和并行执行

- 在一个命令行中键入多个命令来顺序或者并发执行它们。
- 在同一命令行中多个命令顺序执行
  - 句法：命令1；命令2；…；
  - 用途：顺序执行‘命令1’，‘命令2’，…‘命令N’
  - Example:

    $ date ; echo Hello,World;

    Fri Jun 18 23:43:39 PDT 1999

    Hello,World
- 每一个命令后面加上一个“与”符号（&）来使同一命令行中的命令并发执行
  - 句法：命令1&命令2&…命令N&
  - 用途：并发执行 ‘命令1’，‘命令2’，…‘命令N’，每一个作为单独的进程

命令的顺序和并行执行

- 并发执行例：

```
$ date & echo Hello, World! & uname; who
[1] 15575
[2] 15576
Sat Dec 23 23:22:37 PST 2000
Hello, World!
Linux
[1] – Done                    date
[2]+ Done                     echo Hello, World!
ji    pts/0        Dec 23 21:14
```

  - date和echo命令并行执行，然后顺序执行uname和who命令。

命令的顺序和并行执行

- 在最后一个“&”和之前的“&”之间的所有命令使用一个进程来完成。
- 下面命令行里，date命令使用一个进程执行，而所有命令who;whoami;uname;echo Hello,World!使用另外一个进程来执行。

  $ date & who; whoami ; uname; echo Hello, World! &
- LINUX可以使用括号把用分号隔开的一组命令编成一组，然后在当前shell的子shell中执行它们。这就是命令组（command grouping）
- 命令组中的所有命令都在一个进程中执行，

  $ (date;echo Hello,World!)

  Sat Dec 16 09:27:38 PST 2000

  Hello , World!

终止进程

命令和进程的异常终止

- 按<Ctrl-C>来终止一个前台进程。
- 终止后台进程可用两种方法中的一种：
  - 使用kill命令
  - 先使用fg命令把进程转向前台，然后按<Ctrl-C>。
- kill命令的主要目的是向进程发送信号（也称为软中断）。
- 进程接收到信号后，可以采取以下三种行为之一：
  - 接受内核规定的默认动作
  - 忽略该信号
  - 截获该信号并且执行用户定义的动作
- 缺省的动作将导致进程终止

命令和进程的异常终止

句法：	kill [-signal_number]proc-list kill –l
用途：	发送’signal_number’信号到PID或者jobID在‘proc-list’中的进程；jobID必须以“%”号开始。命令kill -l返回所有信号的号码以及名字的列表
通常使用的信号号码：	1 挂断（退出系统，或者在用调制解调器使用系统时挂断电话） 2 中断<Ctrl-C> 3 退出<Ctrl-\> 9 强制终止 15 终止进程（默认的信号号码）

- 为了终止一个忽略15号信号或者其它信号的进程，需要使用9号信号，即强制终止信号
- 进程号0可以指代所有在当前登录期间创建的进程。因此 kill –9 0命令可以终止*所有*登录时产生的进程（即，当前会话中的所有进程），这样，你自己就不得不退出系统。

命令和进程的异常终止

```
$ ps
  PID  TTY          TIME  CMD
1322   pts/0      00:00:00  bash
1831   pts/0      00:34:17  sort
1837   pts/0      00:12:42  a.out
1850   pts/0      00:00:10  find
1851   pts/0      00:00:03  find
1852   pts/0      00:00:02  find
1853   pts/0      00:00:00  ps
$ kill 1850
$ kill –2 1851
[1] Terminated     find /-inum 23456 –print > inumpaths 2 > /dev/null
$ kill –9 1837
[2] – Interrupt      find / -name foo –print > foopaths 2 > /dev/null
$ <Enter>
[3] + Killed       a.out
$
```

pgrep，pkill

- pgrep, pkill − look up or signal processes based on name and other attributes
- pgrep [−cflvx] [−d *delimiter*] [−n|−o] [−P *ppid*,...] [−g *pgrp*,...] [−s *sid*,...] [−u *euid*,...] [−U *uid*,...] [−G *gid*,...] [−t *term*,...] [*pattern*]
- pkill [−*signal*] [−fvx] [−n|−o] [−P *ppid*,...] [−g *pgrp*,...] [−s *sid*,...] [−u *euid*,...] [−U *uid*,...] [−G *gid*,...] [−t *term*,...] [*pattern*]

进程的睡眠、顺序、并发命令

进程睡眠

- sleep 命令，使当前正在执行的进程在规定的时间内处于睡眠状态。
- 句语：

  sleep *n*

  睡眠n秒

命令的有条件执行

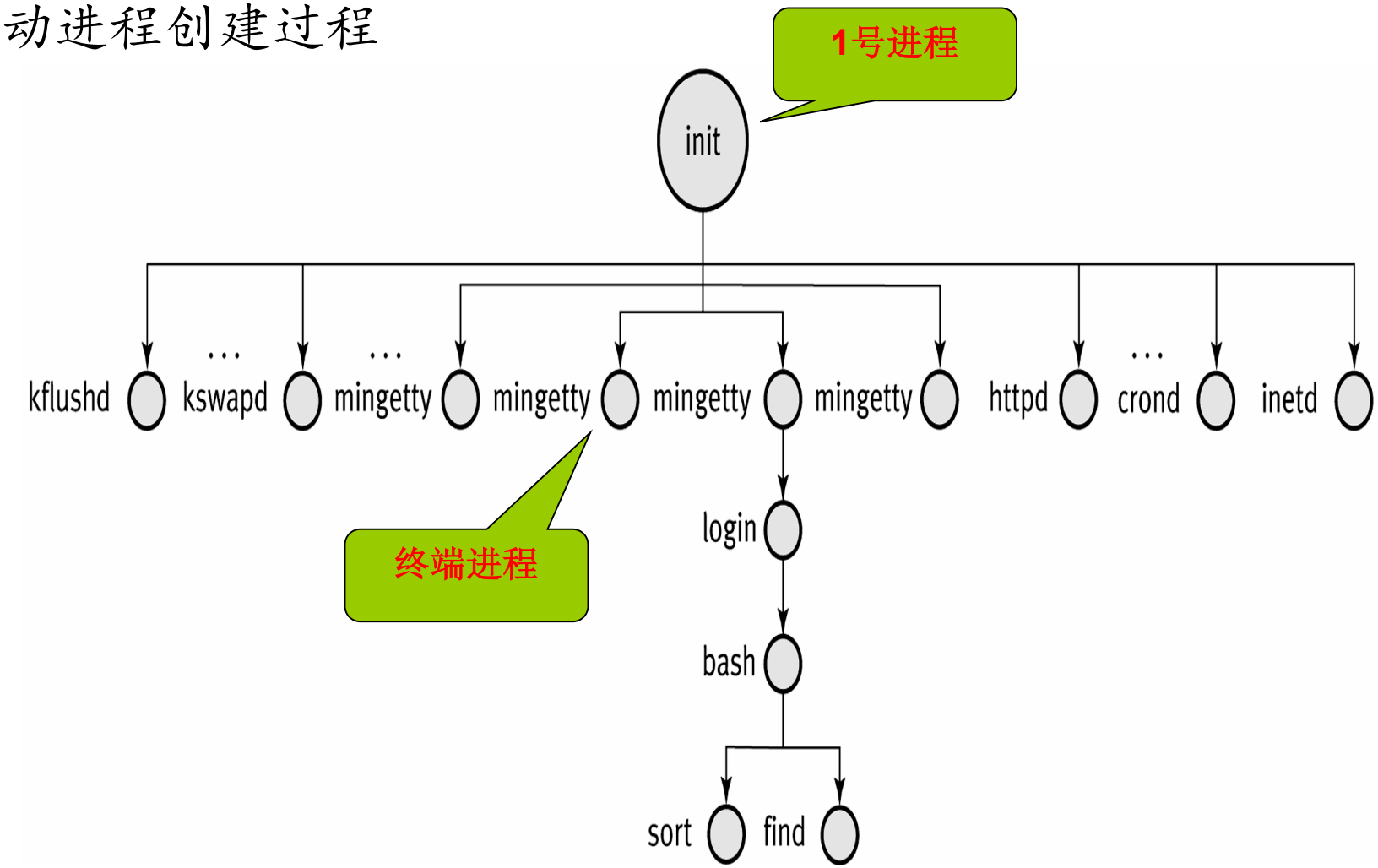
- 使用逻辑与操作符“&&”和逻辑或操作符“||”来有条件地执行命令。
- 命令语法：

  command1&&command2

  command1 | | command2
- 用途：
  - 第一个句法：如果’command1’成功则执行’command2’
  - 第二个句法：如果’command1’失败则执行’command2’

LINUX进程的层次关系

- Linux启动进程创建过程



LINUX进程的层次关系

- pstree命令可以用图的形式显示当前系统中执行进程的进程树，勾勒出进程间的父子关系。
  - $ pstree –a | more
- bash shell下可以用ulimit显示用户可以同时执行的最大进程个数。
  - $ ulimit –u

  12489 #ubuntu 18.10

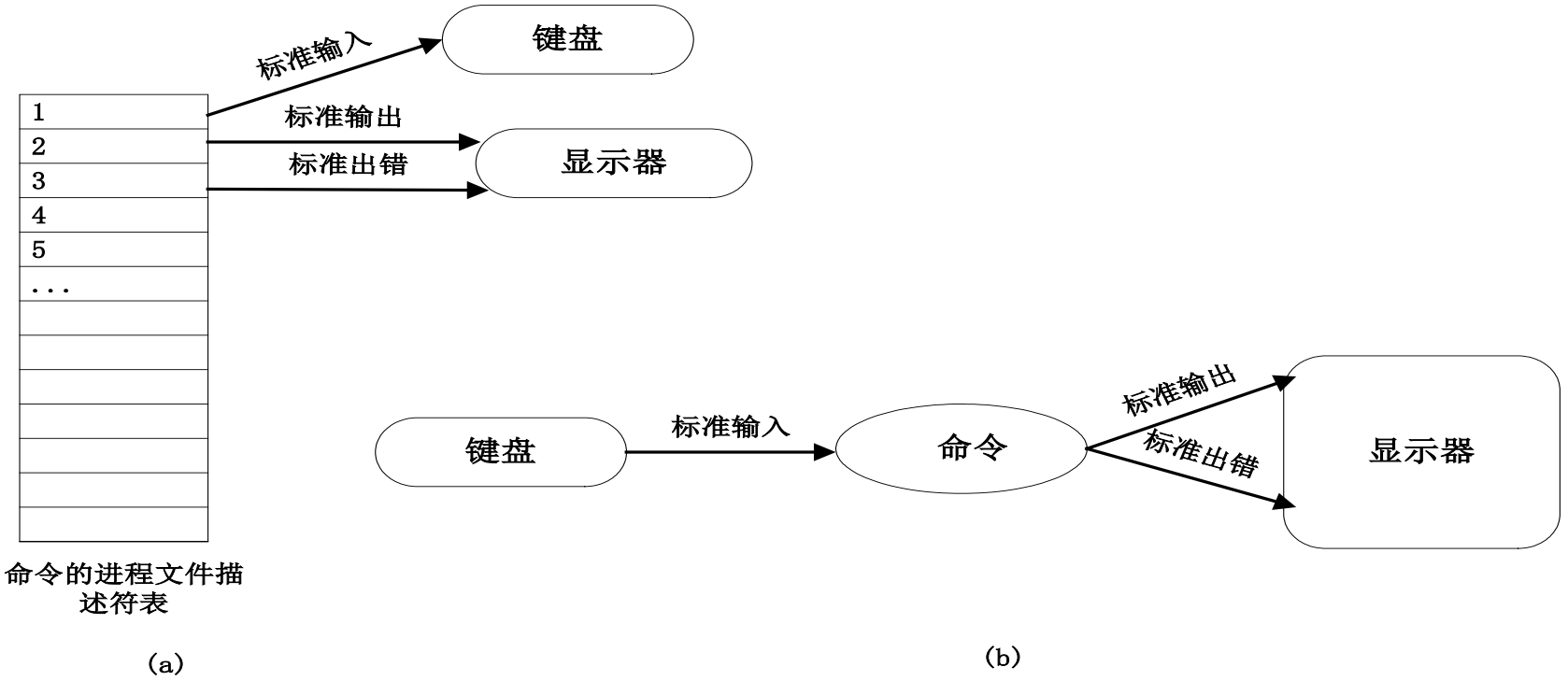
5.2 重定向

重定向

- 所有计算机软件（命令）无非是执行一条或者多条下列的操作：输入、处理和输出。
- 默认的输入、输出和错误输出文件
  - Standard Input (stdin)
  - Standard Output (stdout)
  - Standard Error (stderr)
- 在LINUX中，内核为每个命令自动打开三个文件，他们分别是命令的输入、输出和错误信息文件。
- 在LINUX中，命令的输入、输出和错误信息可以利用文件的重定向功能重定向到其他文件。

标准文件

- By default
  - stdin is associated with keyboard
  - stdout is associated with display screen
  - stderr is associated with display screen



输入重定向(Input Redirection)

- 输入重定向用 ‘<‘符号表示
- 语法：command < input-file
  - 命令的输入来自 ‘input-file’而不是键盘
- Example:

  cat < tempfile

输入重定向（续）

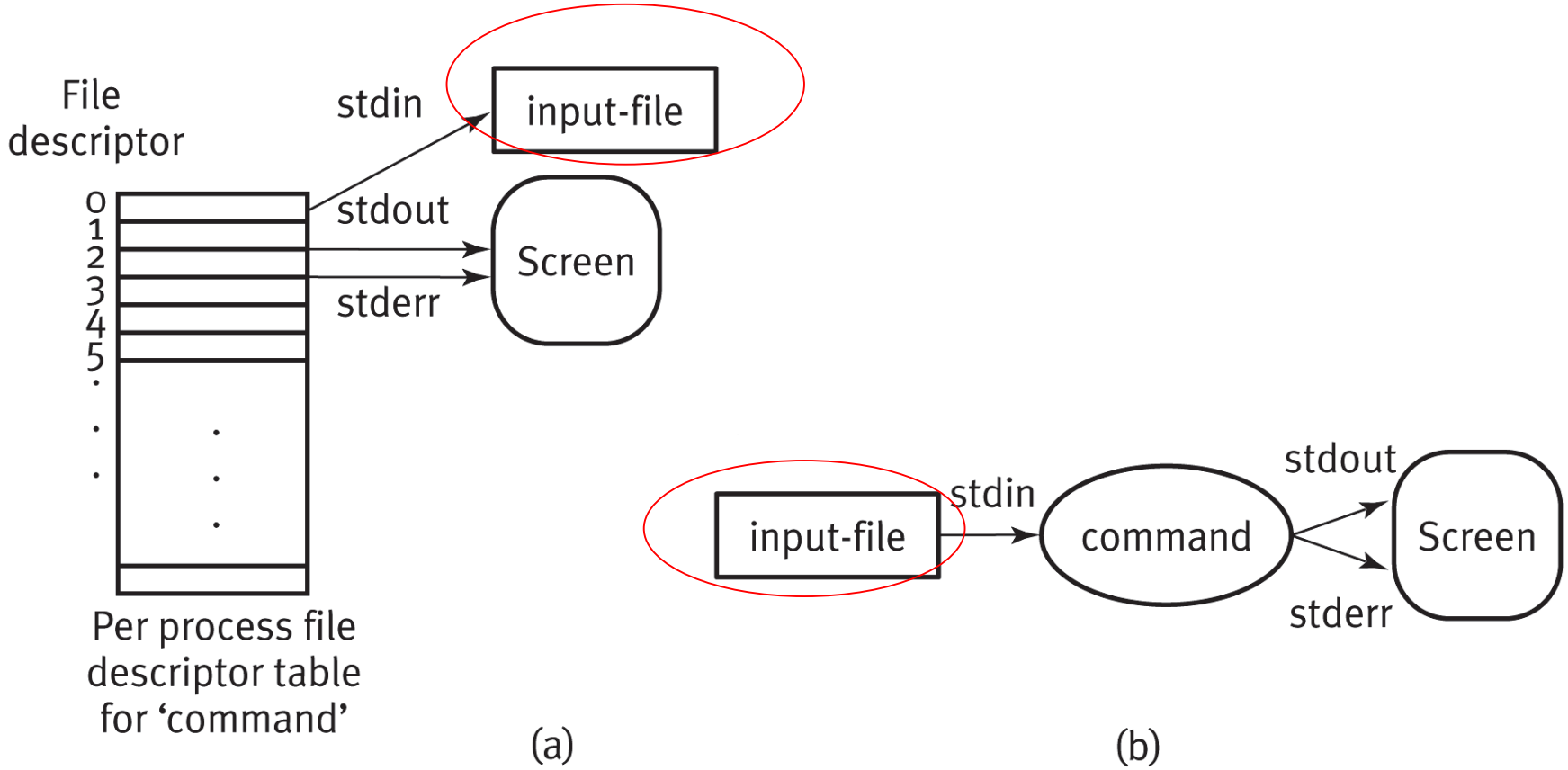
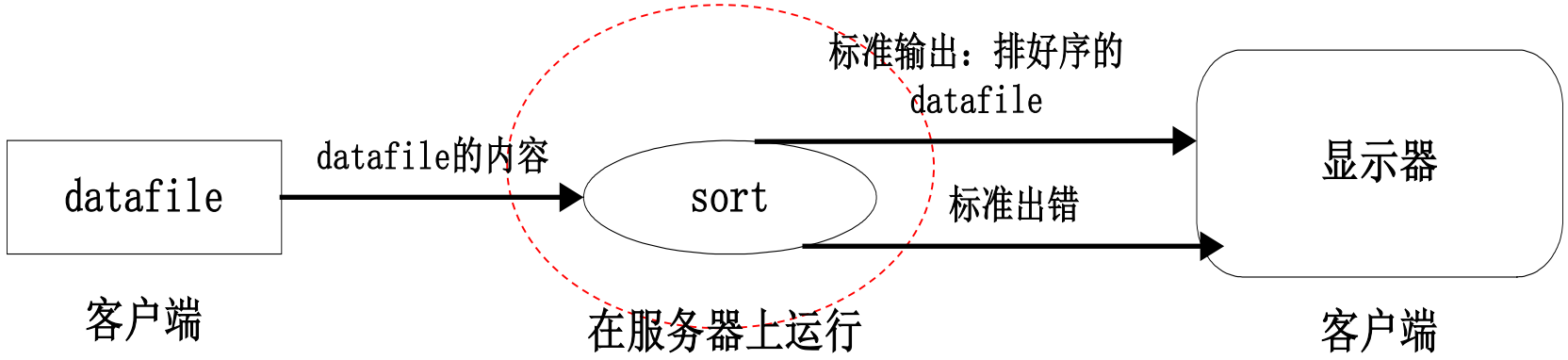


Figure 12.2 Input redirection: (a) file descriptors and standard files for 'command'; (b) semantics of input redirection

输入重定向（续）

- 在网络环境中，下面的命令被用作排序数据文件，这些文件保存在客户端计算机中，而客户端可登录到服务器上运行sort命令:

  $ rsh *server* sort < datafile



内联输入重定向

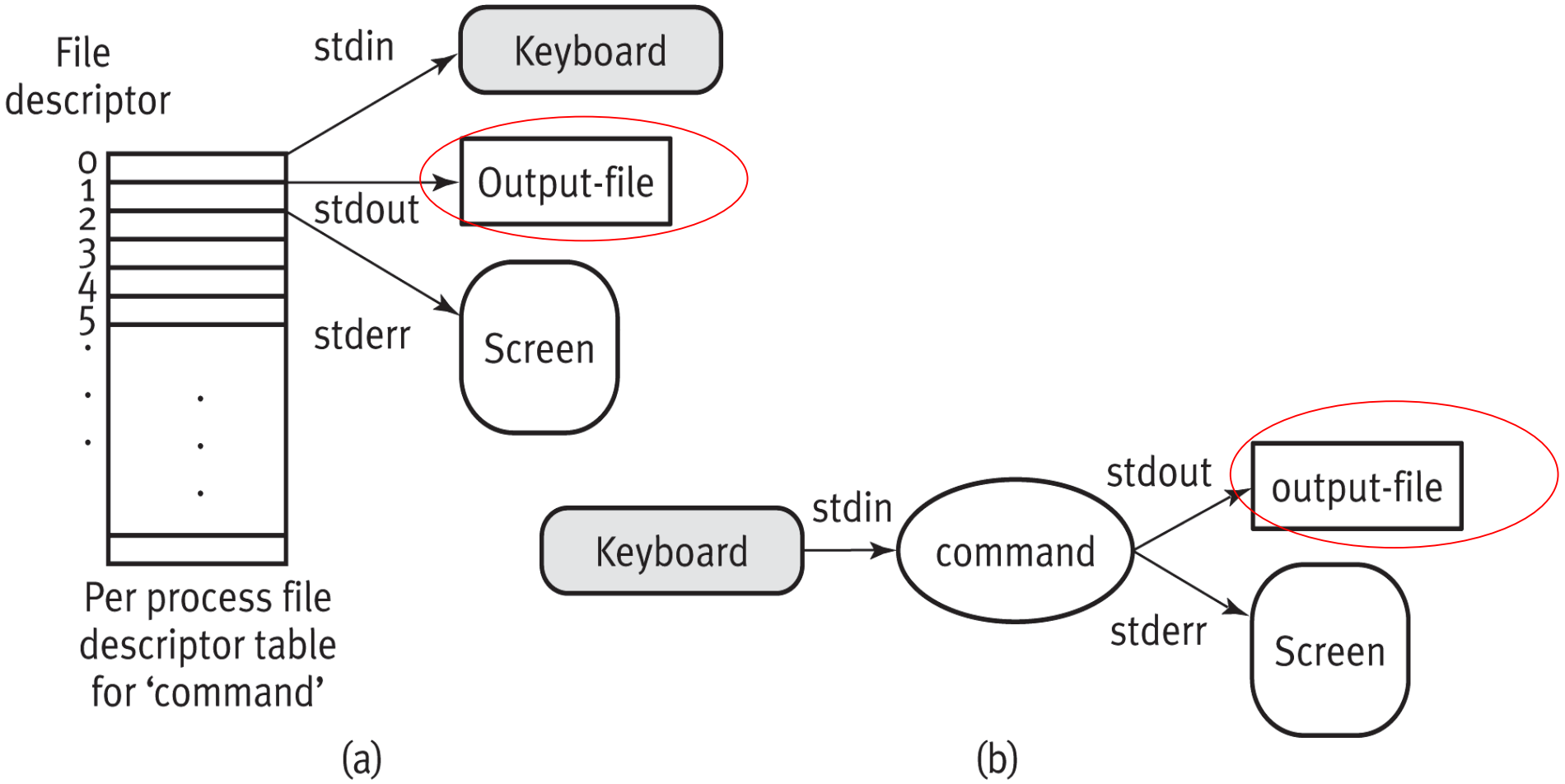
- 内联输入重定向符号是远小于号（<<）。除了这个符号，你必须指定一个文本标记来划分输入数据的开始和结尾。任何字符串都可作为文本标记，但在数据的开始和结尾文本标记必须一致。
- 在命令行上使用内联输入重定向时，shell会用PS2环境变量中定义的次提示符来提示输入数据。

```
$ wc << EOF
> test string 1
> test string 2
> test string 3
> EOF
3 9 42
```

输出重定向(Output Redirection)

- 输出重定向用 ‘>‘符号表示
- 语法：command >output-file
  - 将命令的输出送到文件‘output-file’而不是显示器
- Example: cat > newfile

输出重定向（续）


File
descriptor
0
1
2
3
4
5
stdin
stdout
stderr
Keyboard
Output-file
Screen
Per process file
descriptor table
for 'command'
(a)
Keyboard
stdin
command
stdout
stderr
output-file
Screen
(b)

输出重定向和输入重定向的合并

- command < input-file > output-file
- command > output-file < input-file
- 命令从 ‘input-file’读取输入而不是键盘，输出结果送到 ‘output-file’而不是屏幕

- Example:
  - $ cat < lab1 > lab2
  - cat命令的输入来自文件lab1，然后将输出送到文件lab2。这条命令的真正作用就是建立一个lab1的拷贝——lab2

带文件描述符的I/O重定向

- LINUX的内核将一系列小的整数关联到每个已打开的文件上，称作文件描述符。

  standard input (sdin) — 0

  standard output (stout) — 1

  standard error (sderr) — 2

- 通过使用文件描述符，在Bash Shell和POSIX Shell中标准输入和标准输出能够分别用 0< 和 1>操作符来重定向。
- Example: $ grep “John” 0< tempfile

  == grep “John” < tempfile

标准出错重定向

- 通过使用2>操作符（用 >操作符将标准错误与文件描述符关联起来），一条命令的标准出错能够被重定向。
- command 2> error-file
- Error message generated by ‘command’ and sent to stderr are redirected to error-file
- Example:
  - $ ls –l foo 2> error.log

标准出错重定向（续）

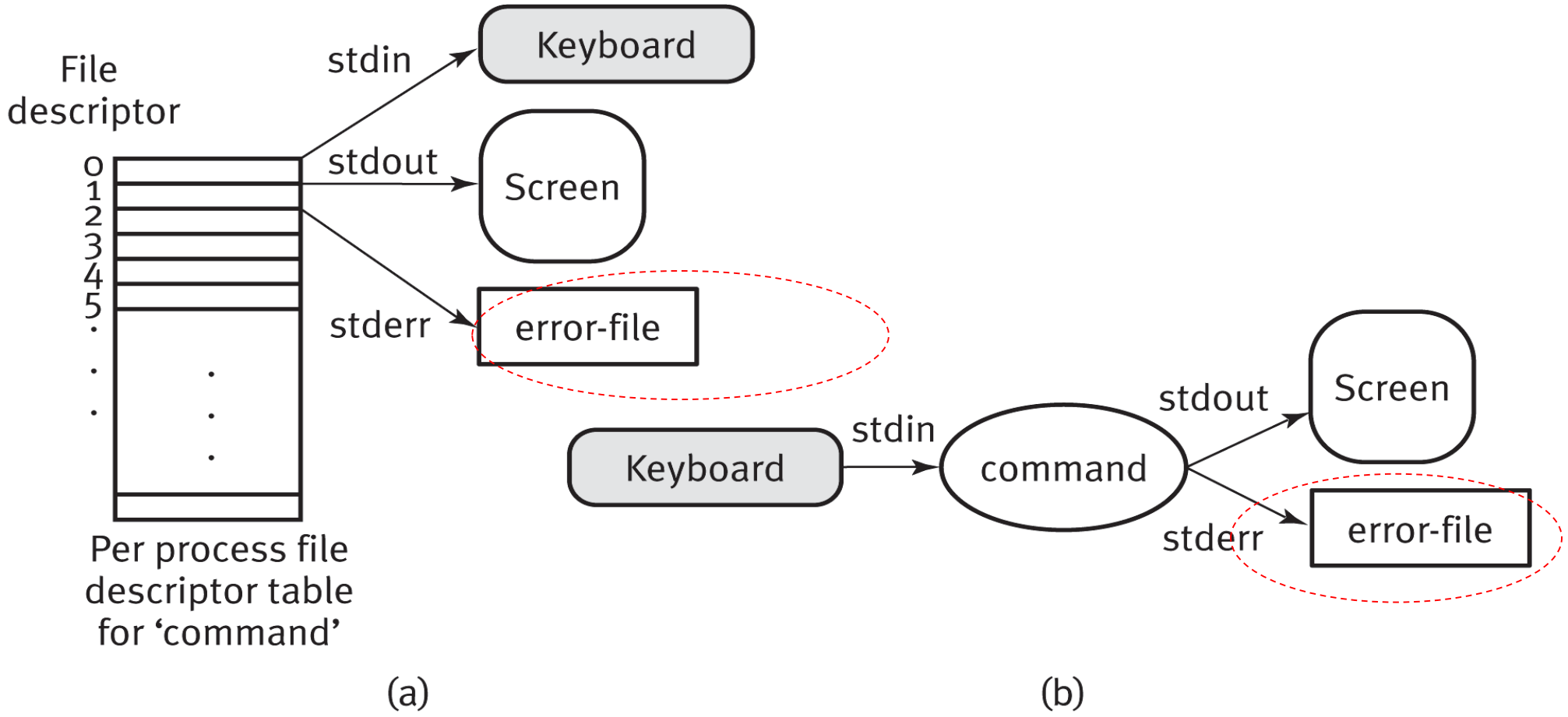
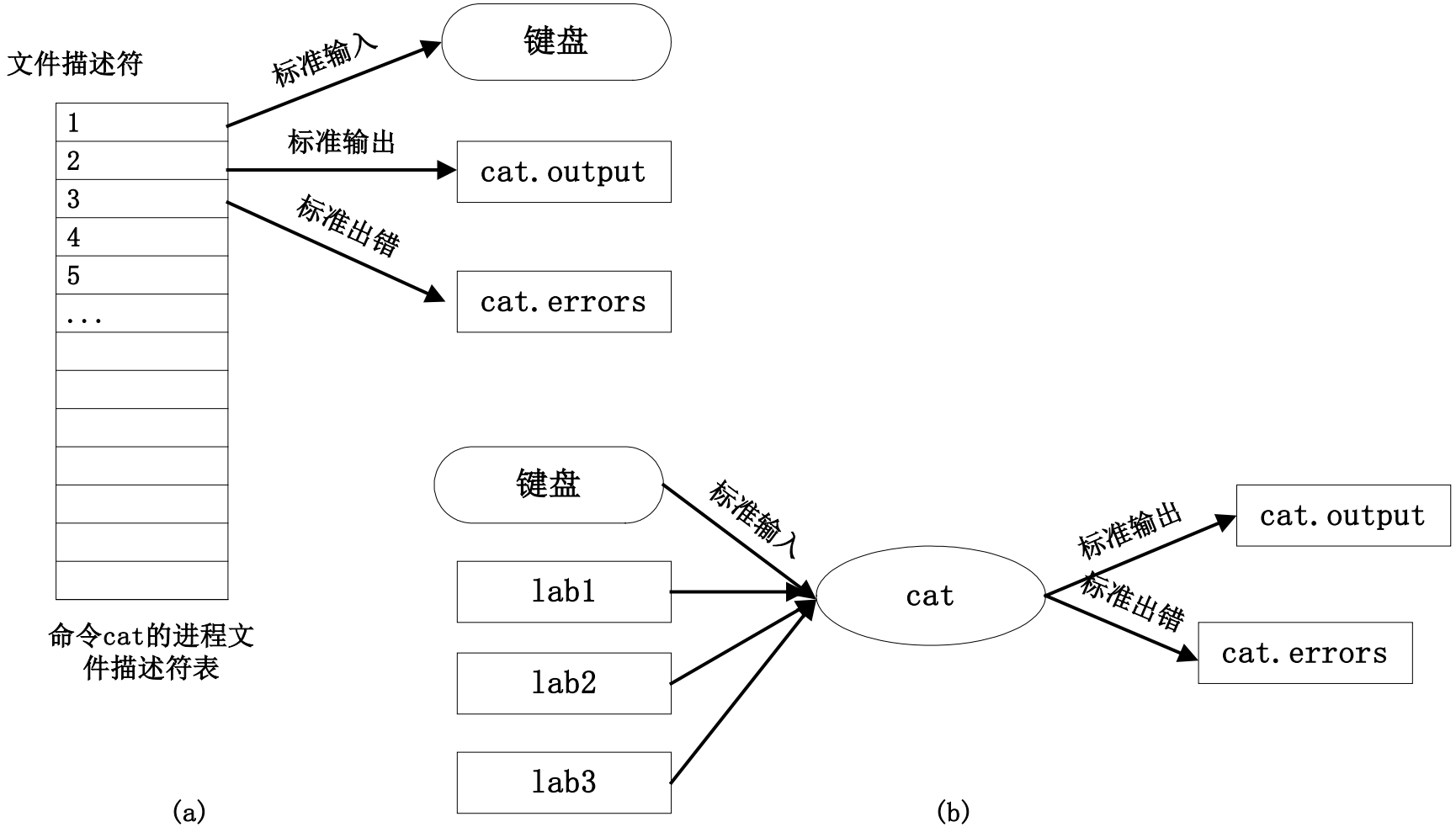


Figure 12.6 Error redirection: (a) file descriptors and standard files for 'command'; (b) semantics of error redirection

用一条命令实现标准输出和标准出错的重定向

- 用文件描述符带上 > 符号将标准输出和标准出错重定向到同一个文件。

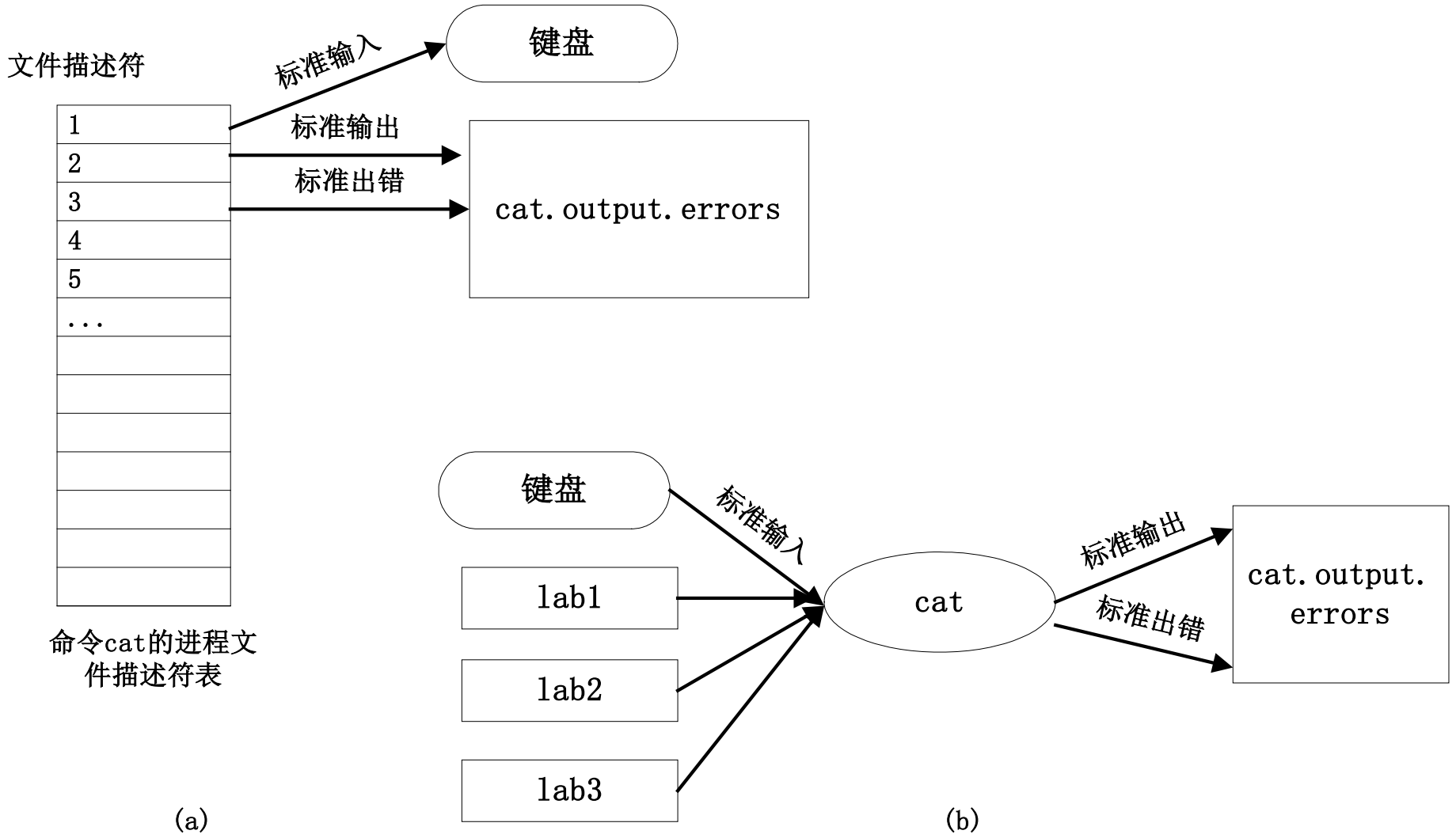
  $cat lab1 lab2 lab3 1> cat.output 2>cat.errors



用一条命令实现标准输出和标准出错的重定向（续）

- 下面的命令将cat 命令的标准输出和标准错误都重定向到文件cat.output.errors。
  - $cat lab1 lab2 lab3 1> cat.output.errors 2>&1
  - $cat lab1 lab2 lab3 2> cat.output.errors 1>&2
- 字符串2> &1告诉Shell，使文件描述符2为文件描述符1的拷贝，导致错误信息送往和该命令输出相同的地方。
- 字符串1>&2告诉Shell，使文件描述符1为文件描述符2的一个拷贝。

用一条命令实现标准输出和标准出错的重定向（续）



用一条命令实现标准输出和标准出错的重定向（续）

- 在命令行的解析中，文件的重定向顺序是从左到右
- $ cat lab1 lab2 lab3 2>&1 1> cat.output.errors
  - 标准出错先设置成标准输出的拷贝（这个时候标准输出还是显示器），然后标准输出才改为cat.output.errors。

用一条命令重定向标准输入、标准输出和标准出错

- command 0<input-file 1>output-file 2> error-file
- “命令”的输入来自输入文件而不是键盘，输出结果送到输出文件而不是显示器，“命令”产生的错误信息送到出错文件。
- 文件描述符0和文件描述符1并不需要，是由于它们的值是默认的

$sort 0< students 1> stuents.sorted 2> sort.error

$sort 2>sort.error 0< students 1> students.sorted

不覆盖文件内容的重定向(追加)

- 默认情况下，输出和错误重定向会覆盖目标文件的内容。
- 如果需要在文件的末尾追加输出结果或者错误信息而不是覆盖它，那么，只要将>操作符换成>>操作符即可。
- 默认情况下， >>符号的文件描述符是1，但是文件描述符2也可以在文件的后面追加出错内容。
- Examples:

$ ls –l 1>> output.dat 2>> error.log

$ cat memo letter >>stuff 2 > error.log

exec命令和文件I/O

- exec命令是命令级的Linux载入器。
- 通常exec用来执行新的命令以替换当前的shell进程，而不是生成一个新的进程（用被运行命令的代码覆盖当前的shell）。
- exec命令有下面两个独特的用途：
  - 1．执行一个命令或程序来取代当前的进程（运行exec的进程，一般是shell）。
  - 2．打开和关闭文件描述符。
- 当exec命令与重定向符号一起使用时，允许命令或脚本读写任何类型的文件，包括设备。

不创建新的进程执行命令

- exec命令可以用来执行命令替换当前运行这个命令的进程（通常是shell）。
- 语法：exec command
  - 用途：将command的代码覆盖到运行exec命令的进程（调用进程）上，command 替换调用进程而不生成新的进程。
- 例：

  $exec date

  Thu Feb 10 23:08:29 PST 2001

  login:

  - 当exec date结束时，控制不返回shell进程而是返回mingetty进程（shell进程的父进程）

不创建新的进程执行命令

```
$ ps
  PID TTY          TIME CMD
30680 pts/1    00:00:00 bash
30835 pts/1    00:00:00 ps
$ bash
% ps
  PID TTY          TIME CMD
30680 pts/1    00:00:00 bash
30836 pts/1    00:00:00 bash
30856 pts/1    00:00:00 ps
% exec date
Sat Feb 10 21:06:10 PST 2001
$ ps
  PID TTY          TIME CMD
30680 pts/1    00:00:00 bash
30857 pts/1    00:00:00 ps
$
```

通过exec命令的文件I/O

- Bash shell最多允许同时使用10个文件描述符。其中三个是保留的，标准输入（0），标准输出（1），标准错误（2）。
- 用exec命令及重定向操作符可以用这10个描述符进行文件I/O。
- 当从命令行运行时，exec < sample命令将sample文件中的每一行作为命令，由当前shell执行。
- 当从命令行执行exec > data 命令时，将使此shell后面的所有命令输出到文件data中（通常是输出到显示屏）。

通过exec命令的文件I/O

表	操作文件I/O时exec命令的句法

句法	含义
exec < file	把这个进程的标准输入重定向到file，从file中读入
exec > file	把这个进程的标准输出重定向到file，向file中写
exec >> file	把这个进程的标准输出重定向到file，向file末尾写入输出（添加）
exec n< file	打开文件file读，并把文件描述符n赋值给这个文件
exec n> file	打开文件file写，并把文件描述符n赋值给这个文件
exec n<< tag ... tag	打开一个here文档（即，在<<tag和tag之间的数据）用来读，把文件描述符n赋值给这个文件。
exec n>> file	打开文件file，并把文件描述符n赋值给这个文件，输出内容添加到file末尾
exec n>& m	把m复制到n中，即，进入到文件描述符m中的内容也同样进入到文件描述符n中。
exec </dey/tty	恢复重定向为标准输入
exec >/dev/tty	恢复重定向为标准输出

通过exec命令的文件I/O

- exec < 实例：

```
$ cat p1.sh
pwd
date
who
$ ps
  PID TTY          TIME CMD
 2113 pts/0    00:00:00 bash
 2133 pts/0    00:00:00 ps
$ bash
$ ps
  PID TTY          TIME CMD
 2113 pts/0    00:00:00 bash
 2134 pts/0    00:00:00 bash
 2140 pts/0    00:00:00 ps
```

```
$ exec<p1.sh
$ pwd
/home/ji
$ date
2020年 04月 20日 星期一 14:57:35 CST
$who
ji       :0           2020-04-20 14:48 (:0)
exit
$ ps
  PID TTY          TIME CMD
 2113 pts/0    00:00:00 bash
 2143 pts/0    00:00:00 ps
```

通过exec命令的文件I/O

```
$ exec > data
$ date
$ echo Hello, World!
$ uname -a
$ exec > /dev/tty
$ date
Tue Jun 29 23:57:30 PDT 2004
$ cat data
Tue Jun 29 23:56:03 PDT 2004
Hello, World!
AIX pccaix 3 4 000001336700
$
```

通过exec命令的文件I/O

```
$ cat exec.demo1
cat > file1
exec < file2
cat > file3
exec < /dev/tty
cat > file4
$ chmod 755 exec.demo1
$ exec.demo1

Hello, world!
<Ctrl-D>
This is great!
<Ctrl-D>
$
```

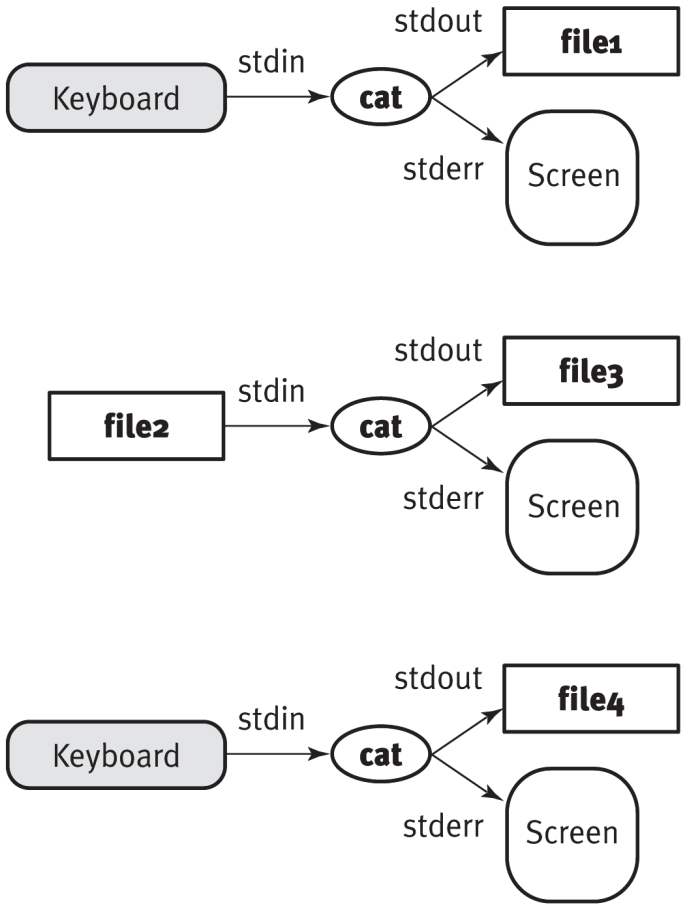


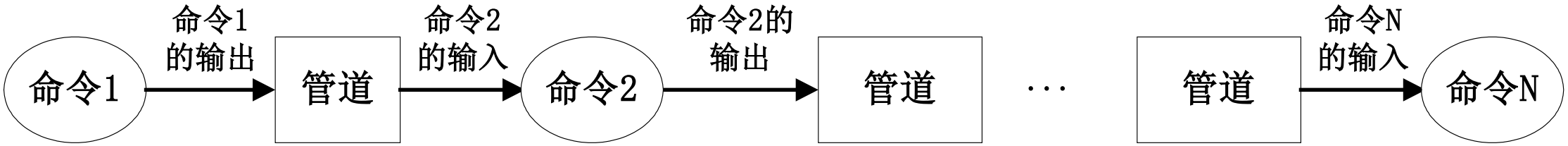
Figure 16.4 Detachment and reattachment of stdin and stdout inside a shell script

5.3 管道

Linux管道(Pipes |)

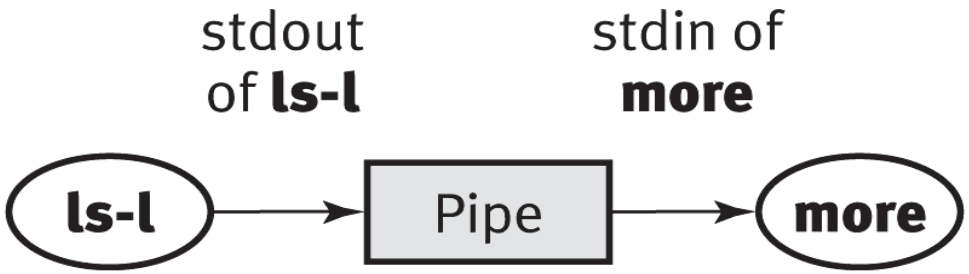
- Linux允许一条命令的标准输出成为另外一条命令的标准输入
- 语法:command1 | command2 | … | commandN

  ‘command1’的标准输出作为command2的标准输入,…, ‘command N-1’的标准输出作为‘command N’的标准输入
- 用管道连接的那些命令叫做过滤器（filter）。一个过滤器(Filters)是一组Linux的命令，从标准输入经过处理送到标准输出。
- 一些经常用到的过滤器是：cat、compress、crypt、grep、gzip、lp、pr、sort、tr、uniq和wc。

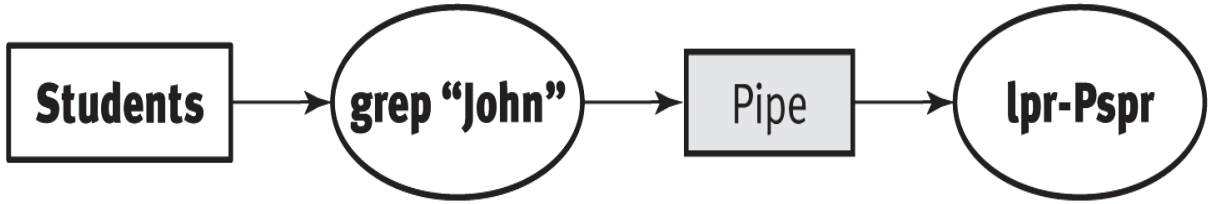


Linux Pipes （续）

- $ ls –l | more



- I/O 重定向和管道用在同一条命令中:
  - $grep “join” < Student | lpr -Pspr



Linux Pipes （续）

- $ who | sort | grep “John” | mail –S “John’s Terminal” doe@coldmail.com
  - 这条命令将who命令的输出排序，然后找到其中包含“John”的行（如果存在的话），连带着“John’s Terminal”发送到doe@coldmail.com。这行命令和下列命令序列等效。

```
$ who > temp1
$ sort < temp1 > temp2
$ grep “John” temp2 > temp3
$ mail –S “John’s Terminal” doe@coldmail.com < temp3
$ rm temp1 temp2 temp3
$
```

结合管道使用重定向

- 在同一条命令中，你不可能单独使用重定向或管道将这条命令的标准输出重定向到一个文件并且连接到另一条命令的标准输入。但是可以用tee工具达到上面的要求。
- tee命令
  - 语法：tee [options] file-list
  - 用途：从标准输入中得到输入然后送到标准输出和file-list中
- command1| tee file1…fileN|command2
  - ‘command1’标准输出作为tee的标准输入， tee输出送到文件 ‘file1’到 ‘fileN’中，同时作为 ‘command2’的标准输入
- Example:

  $ cat names stuents | grep “John Doe” | tee file1 file2 | wc –l

结合管道使用重定向（续）

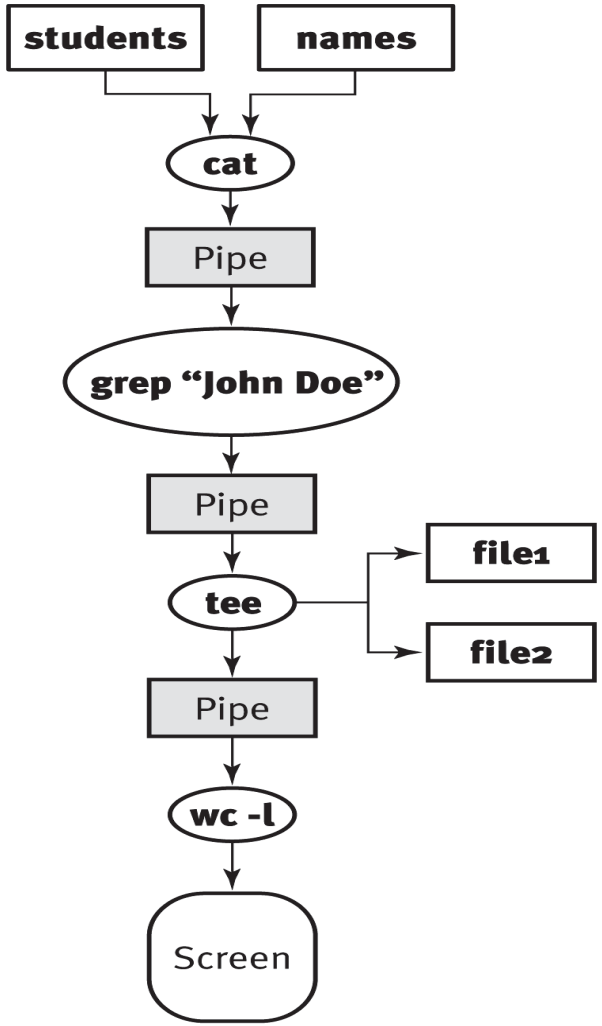


Figure 12.16 The semantics of the `cat names students | grep “John Doe” | tee file1 file2 | wc –l` command

管道应用

- xargs命令：将参数列表转换成小块分段传递
  - 用途：构造参数列表并运行命令

- 例：统计指定目录下所有.c、.h、.s文件的代码行数

```
$ find ~  -name " *.[chS] " | xargs cat | wc –l
```

小结

- 本章需要掌握：
  - 知识：进程概念、串行执行和并行执行过程、前台和后台。重定向，管道
  - 命令：ps、kill、sleep、jobs、bg、fg、；、&、（）、<Ctrl-D>、xargs。在命令中使用I/O重定向（<、>、<<、>>），在命令中使用管道（|），完成单个命令不可能实现的复杂任务
- 本章需要了解：
  - 知识：守护进程、进程创建、文件描述符0、1、2
  - 命令：pstree、sleep、top、tee

Q&A