<http://book.51cto.com/art/200711/59762.htm>

# 守護進程

**第8章 守護進程**

Linux系統啟動時往往需要啟動很多的系統服務程序，例如著名的apache、telnet、ftp等服務程序，這些系統服務程序往往是運行在後台的，人們將這些服務程序稱為守護進程（daemon）。本章將講解守護進程的基本概念，並介紹如何實現一個守護進程。由於守護進程運行在後台中，不可能向終端輸出相關的運行信息，因此，日誌系統是守護進程用於記錄信息的重要手段。本章還將介紹如何在守護進程中使用日誌系統來記錄進程的運行信息。

**8.1 守護進程（daemon）介紹**

守護進程，是一種運行在後台的特殊進程，它不存在控制終端，並週期性地執行某項任務或等待處理某項任務。

圖8.1所示為使用ps命令查看Linux系統進程的情況。從圖中可以看到守護進程的一些特點：所有的守護進程都是以超級用戶啟動的（UID為0）；沒有控制終端（TTY為？）；終端進程組ID為-1（TPGID表示終端進程組ID，該值表示與控制終端相關的前台進程組，如果未和任何終端相關，其值為−1）。所有的守護進程的父進程都為init進程（PID為1的進程），如圖8.2所示。可以使用pstree命令列出Linux系統中進程樹的結構來進行驗證。

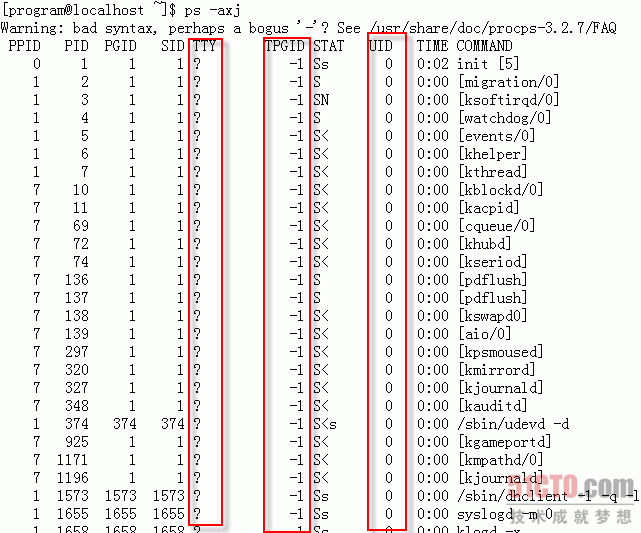


圖8.1 ps命令顯示的系統進程

**8.2 創建守護進程**

由於守護進程的特點，編寫守護進程程序必須遵守一定的規則。本節將闡述這些規則的要點，並給出相關代碼。

**8.2.1實現守護進程的步驟**在Linux系統中，要編程實現一個守護進程必須遵守如下的步驟。

**1．讓init進程成為新產生進程的父進程。**  
調用fork函數創建子進程後，使父進程立即退出。這樣，產生的子進程將變成孤兒進程，並被init進程接管，同時，所產生的新進程將變為在後台運行。

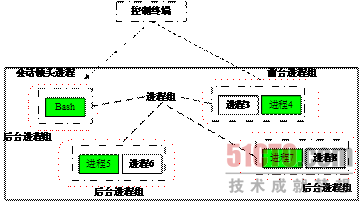
**2．調用setsid函數**  
通過調用setsid函數，使得新創建的進程脫離控制終端，同時創建新的進程組，並成為該進程組的首進程。

在Linux系統中，所有的進程都屬於各自的進程組。進程組是一個或多個進程的集合。可以認為某個班級是一個進程組，而其中成員就是進程。一個班級至少有一個成員。當一個班級的最後一個成員不存在的時候，這個班級也就不存在了，也就是進程組消亡了。每個進程組都有類似於進程號的標識，稱為進程組ID。進程組ID是由領頭進程的進程號決定的，每個進程組都存在一個領頭進程。進程組的存在與否與領頭進程是否存在沒有關係。

會話是一個或多個進程組的集合。與進程組類似，每個會話都存在一個領頭進程。Linux是一個多用戶的操作系統，在同一時刻系統中會存在屬於不同用戶的多個進程。如果用戶在某個終端上發送了某個信號，例如，按下“Ctrl+C”發送SIGINT信號，如何確保信號被正確地發送到對應的進程，同時不會影響使用其他終端的用戶的進程？

會話和進程組是Linux內核用於管理多用戶情況下用戶進程的方法。每個進程都屬於一個進程組，而進程組又屬於某個會話。當用戶從終端登錄系統（不管是終端還是偽終端），系統會創建一個新的會話。在該終端上啟動的進程都會被系統劃歸到會話的進程組中。會話中的進程通過該會話中的領頭進程（常稱其為控制進程）與一個終端相連。該終端是會話的控制終端。一個會話只能有一個控制終端，反之一樣。如果會話存在一個控制終端，則它必然擁有一個前台進程組。屬於該組的進程可以從控制終端獲得輸入。這時，其他的進程組都為後台進程組。圖8.3所示為會話、進程組、進程與控制終端之間的關係。

圖8.3 會話、進程組、進程與控制終端的關係



由於守護進程沒有控制終端，而使用fork函數創建的子進程繼承了父進程的控制終端、會話和進程組，因此，必須創建新的會話，以脫離父進程的影響。Linux系統提供了setsid函數用於創建新的會話。setsid函數的信息如表8.1所示。

表8.1 setsid函數

|  |  |  |  |
| --- | --- | --- | --- |
| 頭文件 | <unistd.h> | | |
| 函數形式 | pid\_t setsid(void); | | |
| 返回值 | 成功 | 失敗 | 是否設置errno |
| 調用進程的會話ID | −1 | 是 |

setsid函數將創建新的會話，並使得調用setsid函數的進程成為新會話的領頭進程。調用setsid函數的進程是新創建會話中的惟一的進程組，進程組ID為調用進程的進程號。setsid函數產生這一結果還有個條件，即調用進程不為一個進程的領頭進程。由於在第一步中調用fork的父進程退出，使得子進程不可能是進程組的領頭進程。該會話的領頭進程沒有控制終端與其相連。至此，滿足了守護進程沒有控制終端的要求。

**3．更改當前工作目錄**  
使用fork函數產生的子進程將繼承父進程的當前工作目錄。當進程沒有結束時，其工作目錄是不能被卸載的。為了防止這種問題發生，守護進程一般會將其工作目錄更改到根目錄下（/目錄）。更改工作目錄使用的函數是chdir。

**4．關閉文件描述符，並重定向標準輸入、輸出和錯誤輸出**新產生的進程從父進程繼承了某些打開的文件描述符，如果不使用這些文件描述符，則需要關閉它們。守護進程是運行在系統後台的，不應該在終端有任何的輸出信息。使用dup函數將標準輸入、輸出和錯誤輸出重定向到/dev/null設備上(/dev/null是一個空設備，向其寫入數據不會有任何輸出)。下面給出具體的代碼：

|  |
| --- |
| …  *int fd;  //將標準輸入輸出重定向到空設備 fd = open ("/dev/null", O\_RDWR, 0);  if (fd != -1) {   dup2 (fd, STDIN\_FILENO);   dup2 (fd , STDOUT\_FILENO);   dup2 (fd, STDERR\_FILENO);   if (fd > 2) close (fd);  }  …* |

**5．設置守護進程的文件權限創建掩碼**很多情況下，守護進程會創建一些臨時文件。出於安全性的考慮，往往不希望這些文件被別的用戶查看。這時，可以使用umask函數修改文件權限，創建掩碼的取值，以滿足守護進程的要求

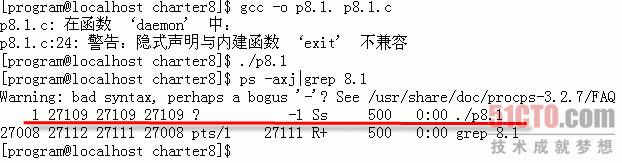
**8.2.2  守护进程具体实现**

本节给出一个守护进程创建的实例。程序p8.1.c中定义了daemon函数，用于实现对守护进程的创建。其创建思想在8.2.1中有详细的介绍，程序的具体代码如下：

|  |
| --- |
| #include <stdio.h> #include <unistd.h> #include <sys/types.h> #include <sys/stat.h> #include <fcntl.h>  /\* daemon函数用于将调用函数的进程转化为守护进程 \*/ int daemon (int nochdir, int noclose) {  pid\_t pid = fork (); /\* 如果创建进程失败 \*/  if (pid < 0) {  perror ("fork");  return -1;  }  if (pid != 0) exit (0);/\* 父进程退出运行 \*/  pid = setsid();/\* 成为会话领头进程 \*/  if (pid < -1) { perror ("setsid"); return -1;  }  if (! nochdir) chdir ("/"); /\* 将工作目录修改成根目录 \*/ if (! noclose) { /\* 将标准输入输出重定向到空设备 \*/ int fd;  fd = open ("/dev/null", O\_RDWR, 0); if (fd != -1) { dup2 (fd, STDIN\_FILENO); dup2 (fd, STDOUT\_FILENO); dup2 (fd, STDERR\_FILENO); if (fd > 2) close (fd); } }  umask (0027);  return 0; }  int main(void) { daemon(0,0); sleep(1000);  return 0; } |

使用gcc编译p8.1.c，得到名为p8.1的可执行文件。执行该程序，程序将以守护进程的状态运行，如图8.4所示。

图8.4  守护进程的实现



**8.3  守护进程的日志实现**

由于守护进程并不拥有控制终端，因此无法将进程运行信息输出显示。但有时候需要根据进程提供的信息来进行系统管理和维护工作。为此，Linux系统提供了一种特殊的机制来解决守护进程的日志问题。syslogd守护进程通过接收其他守护进程的信息，并将这些信息记录在指定位置来解决日志记录问题。syslogd守护进程会根据消息级别来判断是将消息记录在日志文件，还是显示在用户终端上。本节将首先介绍syslogd的基本知识，然后介绍如何编程实现守护进程的日志。

**8.3.1  syslogd守护进程**

syslogd守护进程负责记录、发送系统或工具所产生的信息，其配置文件是/etc/syslog.conf。当系统内核或工具产生信息时，通过调用相关函数，将信息发送到syslogd守护进程。syslogd守护进程会根据/etc/syslog.conf中的配置信息，对消息的去向做出处理。例如，将消息记录在系统日志中、输出到控制台或转发给某个指定的用户，甚至可以让syslogd将信息通过网络转发给网络上其他主机的syslogd守护进程实现系统的集中管理。

**8.3.2  syslogd守护进程配置文件说明**

syslogd守护进程根据/etc/syslog.conf中的配置信息实现消息转发和处理。下面是该配置文件的具体信息：

|  |
| --- |
| # Log all kernel messages to the console. # Logging much else clutters up the screen. #kern.\*                                                 /dev/console  # Log anything (except mail) of level info or higher. # Don't log private authentication messages! \*.info;mail.none;authpriv.none;cron.none                /var/log/messages  # The authpriv file has restricted access. authpriv.\*                                              /var/log/secure  # Log all the mail messages in one place. mail.\*                                                  -/var/log/maillog  # Log cron stuff cron.\*                                                  /var/log/cron  # Everybody gets emergency messages \*.emerg                                                 \*  # Save news errors of level crit and higher in a special file. uucp,news.crit                                          /var/log/spooler  # Save boot messages also to boot.log local7.\*                                                /var/log/boot.log |

在syslog.conf文件中以#开头的是注释行，配置行的语法格式为：[消息类型][TAB分隔符][处理方案]。

**1．消息类型**  
消息类型包括facility和level两个部分，这两个部分以“.”隔开，如：  
mail.\*                                                  -/var/log/maillog

表示facility为mail，而level是所有级别。处理方案是将消息记录在/var/log/maillog日志文件中。表8.2所示为facility可以使用的字段。  
表8.2   facility字段

|  |  |  |
| --- | --- | --- |
| facility取值 | 值 | 说    明 |
| kern | 0 | 内核日志信息 |
| user | 1 | 用户日志信息 |
| mail | 2 | 邮件系统日志信息 |
| daemon | 3 | 系统守护进程日志信息 |
| auth | 4 | 安全管理日志信息 |
| syslog | 5 | syslogd守护进程的日志信息 |
| lpr | 6 | 打印服务日志信息 |
| news | 7 | 新闻组服务日志信息 |
| uucp | 8 | uucp系统日志信息 |
| cron | 9 | 守护进程cron的日志信息 |
| authpriv | 10 | 私有的安全管理日志信息 |
| ftp | 11 | ftp守护进程 |

|  |  |  |
| --- | --- | --- |
|  | 12～15 | 系统保留 |
| local0～local7 | 16～23 | 保留给系统本地使用 |

level给出的级别代表着信息的重要性，表8.3所示为level中的级别信息。

表8.3   level字段

|  |  |  |
| --- | --- | --- |
| level取值 | 值 | 说    明 |
| emerg | 0 | 系统不可用，出现这一情况的话，应该通知所有用户，因为这表示系统出现了严重的问题 |
| alert | 1 | 警告事件，必须立即采取行动纠正的事件 |
| crit | 2 | 关键事件，例如硬件出现故障 |
| err | 3 | 其他错误事件 |
| warning | 4 | 警告事件 |
| notice | 5 | 应该注意的事件，需要特别处理 |
| info | 6 | 通报信息 |
| debug | 7 | 调试程序时产生的信息 |

在syslog.conf中，可以使用\*表示任何facility和任意的level。通过指定facility和level的取值，可以使facility在大于或等于该优先级的时候才记录相关日志信息。对于level，还可以使用none参数，表示不包含任何优先级别。例如，\*.\*表示任何facility的任何优先级的信息。\*.emerg表示任何facility的大于或高于emerg优先级的信息。

**8.3.3  守护进程日志的实现**

syslogd守护进程用于解决守护进程的日志记录问题，而日志信息保存的位置和记录的信息级别是在syslogd守护进程的配置文件中设定的。守护进程日志主要涉及3个函数，分别是openlog、syslog和closelog函数。表8.4所示为这3个函数的具体信息。

表8.4   openlog、syslog和closelog函数

|  |  |  |  |
| --- | --- | --- | --- |
| 头文件 | <syslog.h> | | |
| 函数形式 | void openlog(const char \*ident, int option, int facility);  void syslog(int priority, const char \*format, ...);  void closelog(void); | | |
| 返回值 | 成功 | 失败 | 是否设置errno |
| − | − | − |

**说明：**openlog函数用于打开系统日志连接。只有在打开连接后，才能使用syslog函数向日志文件中添加日志信息。closelog函数用于关闭打开的系统日志连接，该函数的调用在实现中是可选择的。

openlog函数中的第1个参数为ident，该参数常用来表示信息的来源。ident指向的字符信息会被固定地添加在每行日志的前面。第2个参数option用于指定openlog函数和接下来调用的syslog函数的控制标志。option的取值情况如表8.5所示，可以单独取其中的某个值，也可以通过与运算来获得多种特性。第3个参数为facility，这个要与syslogd守护进程的配置文件对应，日志信息会写入syslog.conf文件指定的位置。

表8.5   openlong函数中的option取值表

|  |  |
| --- | --- |
| 参    数 | 说    明 |
| LOG\_CONS | 如果将信息发送给syslogd守护进程时发生错误，直接将相关信息输出到终端 |
| LOG\_NDELAY | 立即打开与系统日志的连接（通常情况下，只有产生第一条日志信息的情况下才会打开与日志系统的连接） |
| LOG\_NOWAIT | 在记录日志信息时，不等待可能的子进程的创建 |
| LOG\_ODELAY | 类似于LOG\_NDELAY参数，与系统日志的连接只有在syslog函数调用时才会创建 |

|  |  |
| --- | --- |
| LOG\_PERROR | 在将信息写入日志的同时，将信息发送到标准错误输出（POSIX.1-2001不支持该参数） |
| LOG\_PID | 每条日志信息中都包括进程号 |

参数facility的取值如表8.6所示。这些值与前面syslog.conf中的facility是存在对应关系的。

表8.6   openlog函数参数facility取值

|  |  |
| --- | --- |
| facility参数 | syslog.conf中对应的facility取值 |
| LOG\_KERN | kern |
| LOG\_USER | user |
| LOG\_MAIL | mail |
| LOG\_DAEMON | daemon |
| LOG\_AUTH | auth |
| LOG\_SYSLOG | syslog |
| LOG\_LPR | lpr |
| LOG\_NEWS | news |
| LOG\_UUCP | uucp |
| LOG\_CRON | cron |
| LOG\_AUTHPRIV | authpriv |
| LOG\_FTP | ftp |
| LOG\_LOCAL0～LOG\_LOCAL7 | local0～local7 |

syslog函数中的第一个参数priority表示消息的级别。与openlog函数中的facility参数类似，priority参数与level也存在对应的关系。priority取值和对应关系如表8.7所示。

表8.7   syslog函数参数priority取值

|  |  |
| --- | --- |
| priority参数 | syslog.conf中对应的level取值 |
| LOG\_EMERG | emerg |
| LOG\_ALERT | alert |
| LOG\_CRIT | crit |
| LOG\_ERR | err |
| LOG\_WARNING | warning |
| LOG\_NOTICE | notice |
| LOG\_INFO | info |
| LOG\_DEBUG | debug |

syslog函数的第二个参数为消息的格式，之后是格式对应的参数。函数的使用类似于printf函数。

程序p8.2.c给出了在程序中实现日志的简单实例。首先，程序调用openlog函数打开与系统日志间的连接；然后，通过调用syslog函数产生5条日志信息；最后，关闭与系统日志间的连接。

|  |
| --- |
| #include <stdio.h> #include <syslog.h>  int main(int argc,char\* argv[]) { openlog(argv[0],LOG\_CONS | LOG\_PID, LOG\_USER);  int count=0; while(count<5) { syslog(LOG\_INFO,"%d, log info test...",count); count++; }  closelog();  return 0; } |

如何查看执行结果呢？

前面介绍了syslogd守护进程会根据syslog.conf文件中的配置，将信息写入指定的日志文件中。调用syslog函数时，指定的priority为LOG\_INFO，对应于syslog.conf中的info优先级。而调用openlog函数的facility为LOG\_USER，对应于syslog.conf中facility取user的情况。查找syslog.conf中的facility.level为user.info的规则，如图8.7所示。可知，日志文件为/var/log/message。

由于查看/var/log下的日志需要root权限。使用tail命令查看message文件中的日志信息，发现相关信息已经通过syslogd守护进程记录在了/var/log/message日志文件中，如图8.8所示。

图8.7  syslog.conf中对应的规则

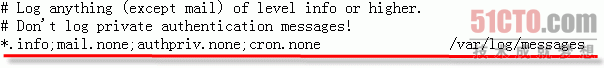
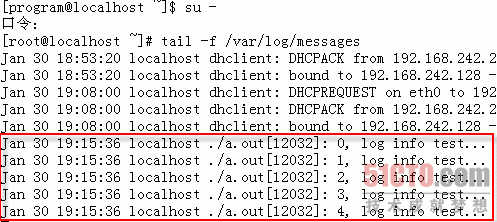


图8.8  程序执行结果



**8.4  Client/Server工作模式**

由于守护进程的特点，守护进程往往作为服务器（Server）进程。例如，syslogd守护进程用来接收其他守护进程发送的消息。其他的守护进程是作为客户机（Client）存在的。

一般情况下，服务器是一个为客户机服务的进程。客户机给服务器发送服务请求，服务器相应这种请求。这一工作模式存在于很多场合，例如，常见的ftp服务器、telnet服务器等。

**8.5  综合实例：定时执行任务的守护进程**

这说明如何编写守护进程。在实际中，处于这一位置的代码往往是一个无限的while循环。这样，守护进程直到系统关闭时才会终止，下面给出相关的伪代码。

|  |
| --- |
| … int main(void) { daemon(0,0);  while(1){ do something } return 0; }  … |

当while循环中的代码为查询时间，并根据时间完成一定的任务时，该程序就转换成类似于Linux系统中的crond守护进程。例如，可以在while循环中添加读写系统当前时间的代码，并判断两次读取的时间差。当到达某个时刻时（如30s）就在日志中写入相关记录。