第13章 精灵进程

13.1 引言

精灵进程（daemon）是生存期长的一种进程。它们常常在系统引导装入时起动，在系统关闭时终止。因为它们没有控制终端，所以说它们是在后台运行的。 UNIX系统有很多精灵进程，它们执行日常事物活动。

本章说明精灵的进程结构，以及如何编写精灵进程程序，因为精灵没有控制终端，我们需要了解在有关事物出问题时，精灵进程如何报告出错情况。

13.2 精灵进程的特征

先来察看一些常用的系统精灵进程，以及它们怎样和第 9章中所叙述的概念：进程组、控制终端和对话期相关联。ps(1)命令打印系统中各个进程的状态。该命令有多个选择项，有关细节请参考系统手册。为了察看本节讨论中所需的信息，在4.3+BSD或SunOS 系统下执行：

```
ps -axj
```

选择项-a显示由其他用户所拥有的进程的状态。 -x显示没有控制终端的进程的状态。 -j显示与作业有关的信息：对话期ID、进程组ID、控制终端以及终端进程组ID。在SVR4之下，与此相类似的命令是ps -efjc（在某些符合美国国防部安全性准则要求的UNIX系统中，只能使用ps 查看自己所拥有的进程）。ps的输出大致是：

```
 PPID   PID    PGID   SID    TT   TPGID   UID     COMMAND
    0     0      0     0     ?     -1      0      swapper
    0     1      0     0     ?     -1      0      /sbin/init -
    0     2      0     0     ?     -1      0      pagedaemon
    1    80     80    80     ?     -1      0      syslogd
    1    88     88    88     ?     -1      0      /usr/lib/sendmail -bd -qlh
    1   105     37    37     ?     -1      0      update
    1   108    108   108     ?     -1      0      cron
    1   114    114   114     ?     -1      0      inetd
    1   117    117   117     ?     -1      0      /usr/lib/lpd
```

其中，已移去了一些我们并无兴趣的列，例如累计 CPU时间。按照顺序，各列标题的意义是：父进程ID、进程ID、进程组ID、终端名称、终端进程组ID（与该控制终端相关的前台进程组）、用户ID以及实际命令字符串。

> 这些ps命令在支持对话期ID的系统（SunOS）上运行，9.5节的setsid函数中曾提及对话期ID。它是对话期首进程的进程ID。但是，4.3+BSD系统将打印与本进程所属进程组对应的session结构的地址（见9.11节）。

进程0、1以及2是8.2节中所述的进程。这些进程非常特殊，存在于系统的整个生命期中。它们没有父进程ID，没有组进程ID，也没有对话期ID。syslogd精灵进程可用于任何为操作人

员记录系统消息的程序中。可以在一台实际的控制台上打印这些消息，也可将它们写到一个文件中（13.4.2节将对syslog设施进行说明）。sendmail是标准邮递精灵进程。update程序定期将内核缓存中的内容写到硬盘上（通常是每隔 30秒）。为了做到这一点，该程序每隔 30秒调用sync（2）函数一次（4.24节已对sync进行了说明）。cron精灵进程在指定的日期和时间执行指定的命令。许多系统管理任务是由 cron定期地使相关程序执行而得以实现的。我们已在 9.3节中提到inetd精灵进程。它监听系统的网络界面，以输入对各种网络服务器的请求。最后一个精灵进程，lpd处理对系统提出的各个打印请求。

注意，所有精灵进程都以超级用户（用户 ID为0）的优先权运行。没有一个精灵进程具有控制终端 —— 终端名称设置为问号（？）、终端前台进程组ID设置为－1。缺少控制终端可能是精灵进程调用了setsid的结果。除update以外的所有精灵进程都是进程组的首进程，对话期的首进程，而且是这些进程组和对话期中的唯一进程。update是它所在进程组（37）和对话期（37）中的唯一进程，但是该进程组的首进程（可能也是该对话期的首进程）已经终止。最后，应当引起注意的是所有这些精灵进程的父进程都是 init进程。

13.3 编程规则

在编写精灵进程程序时需遵循一些基本规则，以便防止产生并不希望的交互作用。下面先说明这些规则，然后是一个按照规则编写的函数 daemon_init。

(1) 首先做的是调用fork，然后使父进程exit。这样做实现了下面几点：第一，如果该精灵进程是由一条简单 shell命令起动的，那么使父进程终止使得 shell认为这条命令已经执行完成。第二，子进程继承了父进程的进程组ID，但具有一个新的进程ID，这就保证了子进程不是一个进程组的首进程。这对于下面就要做的setsid调用是必要的前提条件。

(2) 调用setsid以创建一个新对话期。于是执行 9.5节中列举的三个操作，使调用进程：（a）成为新对话期的首进程，（b）成为一个新进程组的首进程，（c）没有控制终端。

> 在SVR之下，有些人建议在此时再调用 fork，并使父进程终止。第二个子进程作为精灵进程继续运行。这样就保证了该精灵进程不是对话期首进程，于是按照SVR4规则（见9.6节）可以防止它取得控制终端。另一方面，为了避免取得控制终端，无论何时打开一个中断设备都要指定O_NOCTTY。

(3) 将当前工作目录更改为根目录。从父进程继承过来的当前工作目录可能在一个装配的文件系统中。因为精灵进程通常在系统再引导之前是一直存在的，所以如果精灵进程的当前工作目录在一个装配文件系统中，那么该文件系统就不能被拆卸。

另外，某些精灵进程可能会把当前工作目录更改到某个指定位置，在此位置做它们的工作。例如，行式打印机假脱机精灵进程常常将其工作目录更改到它们的 spool目录上。

(4) 将文件方式创建屏蔽字设置为0。由继承得来的文件方式创建屏蔽字可能会拒绝设置某些许可权。例如，若精灵进程要创建一个组可读、写的文件，而继承的文件方式创建屏蔽字，屏蔽了这两种许可权，则所要求的组可读、写就不能起作用。

(5) 关闭不再需要的文件描述符。这样使精灵进程就不再持有从其父进程继承来的某些文件描述符（父进程可能是 shell进程，或某个其他进程）。但是，究竟关闭哪些描述符则与具体的精灵进程有关，所以在下面的例子中不包含此步骤。可以使用程序 2-3中的open_max函数来决定最高文件描述符值，并关闭直到该值的所有描述符。

实例

程序13-1是个函数，可由想初始化成为一个精灵进程的程序调用。

程序13-1 初始化一个精灵进程

```
#include    <sys/types.h>
#include    <sys/stat.h>
#include    <fcntl.h>
#include    "ourhdr.h"

int
daemon_init(void)
{
    pid_t   pid;

    if ( (pid = fork()) < 0)
        return(-1);
    else if (pid != 0)
        exit(0);    /* parent goes bye-bye */

    /* child continues */
    setsid();       /* become session leader */

    chdir("/");     /* change working directory */

    umask(0);       /* clear our file mode creation mask */

    return(0);
}
```

若daemon_init函数由main函数调用，然后进入睡眠状态，那么可以用ps命令检查该精灵进程的状态：

```
$ a.out
$ ps_axj
PPID    PID  PGID   SID    TT   TPGID    UID     COMMAND
   1    735   735   735     ?      -1    224     a.out
```

从中可以看到，该精灵进程已被正确地初始化。

13.4 出错记录

与精灵进程有关的一个问题是如何处理出错消息。因为它没有控制终端，所以不能只是写到标准出错输出上。在很多工作站上，控制台设备运行一个窗口系统，所以我们不希望所有精灵进程都写到控制台设备上。我们也不希望每个精灵进程将它自己的出错消息写到一个单独的文件中。对系统管理人员而言，如果要关心哪一个精灵进程写到哪一个记录文件中，并定期地检查这些文件，那么一定会使他感到头痛。所以，需要有一个集中的精灵进程出错记录设施。

伯克利开发了BSD syslog 设施，并广泛应用于4.2BSD。从4.xBSD导出的很多系统都支持syslog。13.4.2节将说明该设施。

系统V中从来没有一个集中的精灵进程记录设施。 SVR4支持BSD风格的syslog设施，SVR4之下的inetd精灵进程使用syslog。在SVR中，syslog的基础是/dev/log流设备驱动程序，下一节将对此进行说明。

13.4.1 SVR4流log驱动程序

SVR4提供了一种流设备驱动程序，其界面具有流出错记录，流事件跟踪以及控制台记录功能。有关文献包含在〔AT&T 1990d〕的log(7) 中。图13-1详细给出了这种设施的整个结构。

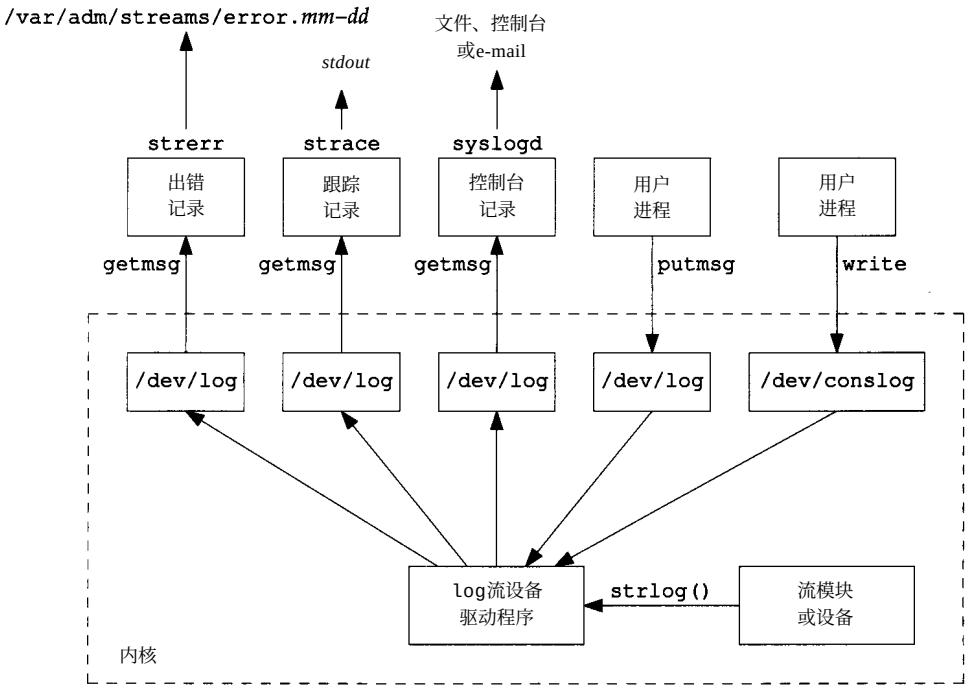

图13-1 SVR4 log 设施

有三个记录进程（logger）：出错记录进程、跟踪记录进程以及控制台记录进程。每一条记录消息可以送给其中之一。

下面介绍三种产生记录消息的方法以及三种读记录消息的方法。

• 产生记录消息。

(1) 内核中的例程可以调用strlog以产生记录消息。这种方法通常由流模块和流设备驱动程序用于出错消息或跟踪消息（跟踪消息常用在新的流模块或驱动程序的排错中）。因为我们无意编写内核中的例程，所以不详细说明这种消息产生方法。

(2) 一个用户进程（例如一个精灵进程）可以用 putmsg将消息送到/dev/log。这种消息可被送到三个记录进程中的任意一个。

(3) 一个用户进程（例如一个精灵进程）可以用 write将消息写到/dev/conslog。这种消息只能送向控制台记录进程。

• 读记录消息。

(4) 标准的出错记录进程是strerr（1M）。它将记录消息增写到在目录/var/adm/stream下的一个文件中。该文件名是 error.*mm-dd*，其中，*mm*是月份，*dd*是天数。strerr本身是个精灵进程，通常在后台运行,它将记录消息增写到该文件中。

(5) 标准的跟踪记录进程是strace(1M)。它能有选择地将一套指定的跟踪消息写至其标准输出。

(6) 标准控制台记录进程是 syslogd，这是一个BSD导出程序，下一节将对此进行叙述。此进程是个精灵进程，它读一个配置文件，然后将记录消息写至一个指定的文件（控制台是一个

文件）或登录用户，或将该消息发送给在另一台主机上的syslog精灵进程。
虽然上面没有提及，但用户也可以用自己的进程替换任意一个系统提供的标准精灵进程。我们可以提供自己的出错记录进程、跟踪记录进程或控制台记录进程。

每则log消息除消息本身外，还包含有一些其他信息。例如，由 log驱动程序沿逆流方向发送的消息，还包含有下列消息：哪个模块产生此消息（如果该消息是由内核中的一个流模块产生的）、级别、优先级、某些标志以及消息产生的时间。有关细节请参阅手册中的 log（7）。如果使用putmsg产生一则log消息，则可以设置这些字段中的几个。如果调用 write将一则消息发送至控制台记录进程（通过/dev/conslog），则只能发送消息字符串。

图13-1中没有显示的另一种可能性是：由一个 SVR4精灵进程调用BSD syslog（3）函数。用这种方法可将消息发送至控制台记录进程，这与用 putmsg向/dev/log发送消息类似。使用syslog，可以设置消息的优先权字段。下一节将讨论此函数。

当产生了某种类型的记录消息，但是相应类型的记录进程却不在运行时， log驱动程序丢弃该消息。

> 不幸的是，在SVR4中，使用这种log设施带有随意性。一些精灵进程使用它，而大多数由系统提供的精灵进程则编写成直接写向控制台。
>
> syslog（3）函数和syslogd(1M)精灵进程的有关文档在 BSD兼容库文档部分〔AT&T 1990c〕，但是它们本身并不在此库中，而是在所有用户进程（精灵进程）都可使用的标准C库中。

13.4.2 4.3+BSD syslog 设施

自4.2BSD以来，广泛地应用了BSD syslog设施。大多数精灵进程使用这一设施。图 13-2显示了syslog设施的详细组织结构。

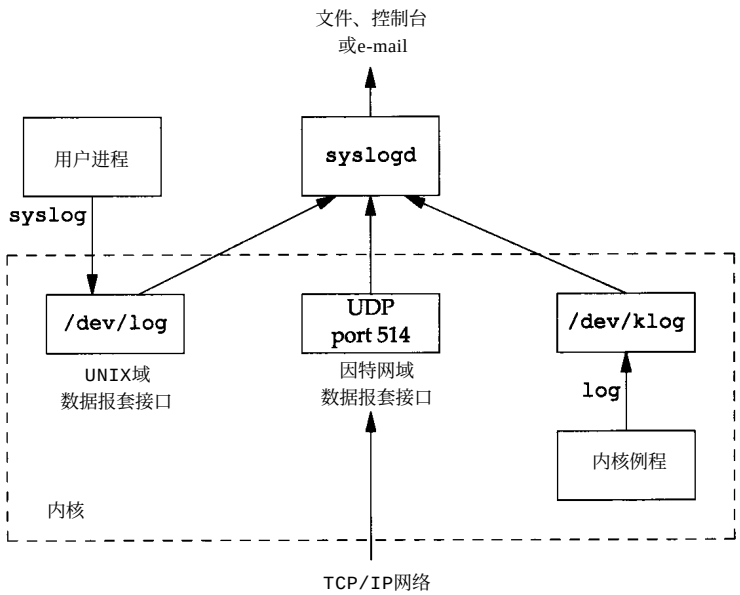

图13-2 4.3+BSD syslog设施

有三种方法产生记录消息：

(1) 内核例程可以调用log函数。任何一个用户进程通过打开和读/dev/klog设备就可以读取这些消息。因为我们无意编写内核中的例程，所以不再进一步说明此函数。

(2) 大多数用户进程（精灵进程）调用syslog（3）函数以产生记录消息。我们将在下面说明其调用序列。这使消息发送至UNIX域数据报套接口/dev/log。

(3) 在此主机上，或通过TCP/IP网络连接到此主机的某一其他主机上的一个用户进程可将记录消息发向UDP端口514。注意：syslog函数并不产生这些UDP数据报——它们要求产生此记录消息的进程具有显式的网络编程。

关于UNIX域套接口以及UDP套接口的细节，请参阅stevens〔1990〕。

通常，syslogd精灵进程读取三种格式的记录消息。此精灵进程在起动时读一个配置文件。一般，其文件名为/etc/syslog.conf，该文件决定了不同种类的消息应送向何处。例如，紧急消息可被送向系统管理员（若已登录），并在控制台上显示，而警告消息则可记录到一个文件中。

该设施的界面是syslog函数。

```
#include  <syslog.h>
void  openlog(char *ident, int option, int facility);
void  syslog(int priority, char *format, ...);
void  closelog(void);
```

调用openlog是可选择的。如果不调用 openlog，则在第一次调用 syslog时，自动调用openlog。调用closelog也是可选择的 —— 它只是关闭被用于与syslogd精灵进程通信的描述符。

调用openlog使我们可以指定一个*ident*，以后，此*ident*将被加至每则记录消息中。*ident*一般是程序的名称（例如，cron、inetd等）。表13-1说明了4种可能的*option*。

表13-1 openlog的*option*参数

option	说 明
LOG_CONS	若日志消息，不能通过UNIX域数据报发送至syslogd，则将该消息写至控制台
LOG_NDELAY1	立即打开UNIX域数据报套接口至syslsgd精灵进程 —— 不要等到记录第一条消息。通常，在记录第一条消息之前，该套接口不打开
LOG_PERROR	除将日志消息发送给syslog外，还将它写至标准出错。此选项仅由4.3BSD Reno及以后版本支持
LOG_PID	每条消息都包含进程ID此选择项可供对每个请求都fork一个子进程的精灵进程使用

openlog中的参数*facility*可以选取表13-2中列举的值。设置*facility*参数的目的是让配置文件可以说明，来自不同设施的消息以不同的方式进行处理。如果不调用 openlog，或者以*facility*为0来调用它，那么在调用syslog时，可将*facility*作为*priority*参数的一个部分进行说明。

调用syslog产生一个记录消息。其*priority*参数是*facility*和*level*的组合，它们可选取的值分别列于*facility*（见表13-2）和*level*（见表13-3）中。*level*值按优先级从最高到最低按序排列。

*format*参数以及其他参数传至vsprintf函数以便进行格式化。在*format*中，每个%m都被代换成对应于errno值的出错消息字符串（strerror）。

SVR4和4.3+BSD都提供logger(1)程序，以其作为向syslog设施发送出错消息的方法。送至该程序的可选择参数可以指定*facility*、*level*以及*ident*。logger的意图是用于以非交互方式运行，又要产生记录消息的shell过程。

表13-2 openlog的*facility*参数

facility	说 明
LOG_AUTH	授权程序:login.su,getty,…
LOG_CRON	cron和at
LOG_DAEMON	系统精灵进程:ftpd,routed,…
LOG_KERN	内核产生的消息
LOG_LOCAL0	保留由本地使用
LOG_LOCAL1	保留由本地使用
LOG_LOCAL2	保留由本地使用
LOG_LOCAL3	保留由本地使用
LOG_LOCAL4	保留由本地使用
LOG_LOCAL5	保留由本地使用
LOG_LOCAL6	保留由本地使用
LOG_LOCAL7	保留由本地使用
LOG_LPR	行打系统:lpd, lpc,…
LOG_MAIL	邮件系统
LOG_NEWS	Usenet网络新闻系统
LOG_SYSLOG	syslogd精灵进程本身
LOG_USER	来自其他用户进程的消息
LOG_UUCP	UUCP系统

表13-3 syslog中的*levels*(按序排列)

level	说 明
LOG_EMERG	紧急(系统不可使用)(最高优先级)
LOG_ALERT	必须立即修复的条件
LOG_CRIT	临界条件(例如,硬设备出错)
LOG_ERR	出错条件
LOG_WARNING	警告条件
LOG_NOTICE	正常,但重要的条件
LOG_INFO	信息性消息
LOG_DEBUG	调试排错消息(最低优先级)

logger命令的格式正由POSIX.2标准化。

实例

第17章的PostScript打印机精灵进程中,包含有下面的调用序列:

```
openlog("lprps", LOG_PID, LOG_LPR);
syslog(LOG_ERR, "open error for %s: %m", filename);
```

第一个调用将*ident*字符串设置为程序名,指定打印该进程ID,并且将系统默认的*facility*设定为行式打印机系统。对syslog的实际调用指定一个出错条件和一个消息字符串。如若不调用openlog,则第二个调用的形式可能是:

```
syslog(LOG_ERR | LOG_LPR, "open error for %s: %m", filename);
```

其中,将*priority*参数指定为*level*和*facility*的组合。

13.5 客户机-服务器模型

精灵进程常常用作为服务器进程。确实，图13-2中，可以称syslogd进程为服务器，用户进程（客户机）用UNIX域数据报套接口向其发送消息。

一般而言，服务器是一个进程，它等待客户机与其联系，提出某种类型的服务要求。图13-2中，由syslogd服务器提供的服务是记录出错消息。

图13-2中，客户机和服务器之间的通信是单向的。客户机向服务器发送其服务要求，服务器则不向客户机回送任何消息。在下面有关进程通信的几章中，有大量实例，其中有客户机和服务器之间的双向通信。客户机向服务器发送要求，服务器则向客户机回送回答。

13.6 小结

在大多数UNIX系统中，精灵进程是一直运行的。为了初始化我们自己的精灵进程，需要一些审慎的思索并理解第9章中说明过的进程之间的关系。本章开发了一个可由精灵进程调用，对其自身正确地进行初始化的函数。

本章还讨论了精灵进程记录出错消息的几种方法，因为精灵进程通常没有控制终端。在SVR4下，可以使用流记录驱动程序，在4.3+BSD之下，提供了syslog设施。因为SVR4也提供BSD syslog设施，所以在下面的章节中，当精灵进程需要记录出错消息时，将调用syslog函数。第17章中，PostScript打印机精灵进程就包含有这种情况。

习题

13.1 从图13-2可以看出，直接调用openlog或第一次调用syslog都可以初始化syslog，此时一定要打开用于UNIX域的数据报套接口的特殊设备文件/dev/log。如果调用openlog前，用户进程(精灵进程)先调用了chroot，结果如何？

13.2 列出你的系统中所有的精灵进程，并说明它们的功能。

13.3 编写一段调用程序13-1中daemon_init函数的程序。调用该函数后调用getlogin（见8.14节）查看该精灵进程是否有登录名。若程序带有 3 > /tmp/name1运行时（Bourne shell或KornShell），则将登录名打印到文件描述符3，并重定向到一个临时文件。在调用daemon_init和getlogin之间关闭描述符1、2和3，此时再运行该程序会有什么不同？

13.4 编写一个SVR4精灵进程，将其作为一个控制台记录进程。细节可参阅〔 AT&T 1990d〕的log(7)。每次接收一则消息，并打印相关信息。再编写一个测试程序，将控制台记录消息发送给/dev/log以测试该精灵进程。

13.5 根据13.3节中提到的规则(2)，通过调用两次fork修改程序13-1，以使它在SVR4下不能取得控制终端。测试新的函数，并验证它不是一个对话期首进程。