Linux 控制预定作业的持续时间

参考 ： https://www.ibm.com/developerworks/cn/linux/l-job-terminating/index.html

Linux® 和 UNIX® 系统允许您对作业进行预先调度，可以只调度一次，也可以反复调度。阅读过最近一篇技巧性文章 Linux 提示: 用 cron 和 at 调度作业 的读者可能想知道如何录制广播和 TV 节目并在节目结束后停止录制。我想起了 Ettore Bugatti，这个意大利人在 Alsace-Lorraine 生产非常好的汽车。当其他汽车制造商很早就改用液压式刹车时，他仍然使用缆索操作式刹车，当客户询问时，Bugatti 这样回答，“先生，我生产的汽车是为了行驶，而不是为了刹车。” 因此，本文的技巧就是为您的作业调度需求提供“刹车”功能。

想了解更多 Linux 技巧吗？

请查阅这些已整理过的技巧性文章：

- 使用 cron 和 at 调度作业
- Bash 参数和参数扩展
- Bash test 和 comparison 函数
- 使用 CUPS 打印 DVI 文件
- 所有 Linux 技巧文章

还没有找到需要的资料吗？请 对本文作出评价，让我们了解您的需求以提供所需的技巧！

在一段时间（或满足其他条件）后终止某个作业，通常涉及两个进程，其中一个进程运行作业而另一个进程监视作业完成条件。通过阅读本文介绍的技巧，您将了解进程如何管理作业实时运行的时间。还将了解在其中一个进程提前结束时如何使用信号和 trap 实用程序终止另一个进程。

定时器进程

shell 脚本中执行定时的基本工具是 `sleep` 命令，该命令将使运行中的 shell 在指定的一段时间内暂停执行。默认的暂停时间为数秒，但是可以使用 `s`、`m` 或 `h` 命令分别将时间值设置为秒、分钟或小时级别。这将延缓 shell 的执行，因此需要在另一个 shell 中运行实时任务，通过将任务放到使用 `&` 字符的后台中运行即可达到此目的。

首先，假定您需要使某个命令持续运行 10 分钟。您将编写清单 1 所示的 bash shell 脚本来按照指定的时间段运行 `xclock` 命令。请在您自己的系统中尝试运行。

清单 1. 首次尝试使用 runclock1.sh

```
#!/bin/bash
runtime=${1:-10m}
# Run xclock in background
xclock&
#Sleep for the specified time.
sleep $runtime
echo "All done"
```

您应该能看到类似图 1 所示的时钟。

图 1. 一个简单的 xclock

这种方法惟一的缺陷就是：虽然脚本停止了运行，但是时钟仍然继续运行。

父进程，子进程和孤儿进程（orphan）

清单 2 展示了一个增强的 `runclock2.sh` 脚本，它在 shell 完成后捕获 shell 和 `xclock` 进程的进程 id 信息，以及脚本输出和 `ps` 命令输出，并显示 `xclock` 的进程状态。

清单 2. 收集诊断信息 runclock2.sh

```
[ian@attic4 ~]$ cat runclock2.sh
#!/bin/bash
runtime=${1:-10m}
mypid=$$
# Run xclock in background
xclock&
clockpid=$!
echo "My PID=$mypid. Clock's PID=$clockpid"
ps -f $clockpid
#Sleep for the specified time.
sleep $runtime
echo "All done"
[ian@attic4 ~]$ ./runclock2.sh 10s
My PID=8619. Clock's PID=8620
UID        PID  PPID  C STIME TTY      STAT   TIME CMD
ian       8620  8619  0 19:57 pts/1    S+     0:00 xclock
All done
[ian@attic4 ~]$ ps -f 8620
UID        PID  PPID  C STIME TTY      STAT   TIME CMD
ian       8620     1  0 19:57 pts/1    S      0:00 xclock
```

注意：ps 的第一个输出中的父进程 id（PPID）为 8619，这是脚本的进程 id（PID）。当脚本终止后，这个时钟进程将变成孤儿进程并被指定为 init 进程（进程 1）的子进程。当父进程终止时，子进程不会立即终止，但是它将在用户注销系统后终止。

终止子进程

终止子进程的解决方法就是使用 kill 命令进行显式终止。它将向进程发送一个*信号*，这通常都会使进程终止。稍后您将看到进程如何 trap 信号而使终止失败，但是这里我们将使用终止信号（SIGINT）来终止时钟。

要查看系统可用的信号列表，请使用 kill 命令和 -l 选项，如清单 3 所示。注意：有些信号可在所有 Linux 系统中通用，而有些信号只能用于特定的机器架构。其中一些信号是由系统生成，例如 floating point exceptions (SIGFPE) 或 segment violation (SIGSEGV)，而另一些信号则由应用程序发送，例如 interrupt (SIGINT)、user signals (SIGUSR1 或 SIGUSR2) 或 unconditional terminate (SIGKILL)。

清单 3. Fedora Core 5 系统中的信号

```
[ian@attic4 ~]$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS      34) SIGRTMIN
35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4
39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

在清单 2 中，我们使用 $! shell 变量捕获到 xclock 进程的 PID。清单 4 展示了如何使用这些信息向 xclock 进程发送终止信号情报（SIGTERM），从而终止该进程。

清单 4. 使用 runclock3.sh 终止子进程 runclock3.sh

```
[ian@attic4 ~]$ cat ./runclock3.sh
#!/bin/bash
runtime=${1:-10m}
mypid=$$
# Run xclock in background
xclock&
clockpid=$!
echo "My PID=$mypid. Clock's PID=$clockpid"
ps -f $clockpid
#Sleep for the specified time.
sleep $runtime
kill -s SIGTERM $clockpid
echo "All done"
```

清单 5 展示了在执行 runclock3.sh 时发生的操作。最后的 kill 命令确定 xclock 进程（PID 9285）确实已终止。

清单 5. 验证子进程是否终止

```
[ian@attic4 ~]$ ./runclock3.sh 5s
My PID=9284. Clock's PID=9285
UID        PID  PPID  C STIME TTY      STAT   TIME CMD
ian       9285  9284  0 22:14 pts/1    S+     0:00 xclock
All done
[ian@attic4 ~]$ kill -0 9285
bash: kill: (9285) - No such process
```

如果没有查看信号规范，那么这里稍作说明：SIGTERM 是默认信号。信号名称中的 SIG 部分是可选的。无需使用 -s 和信号名，只需在信号数之前加上 - 前缀，因此清单 6 中显示的四种方法都能够终止进程 9285。注意特殊值 -0，正如清单 4 中的用法一样，它将测试是否可以将某个信号发送到进程。

清单 6. 使用 kill 命令指定信号的方法

```
kill -s SIGTERM 9285
kill -s TERM 9285
kill -15 9285
kill 9285
```

其他终止条件

现在您已经具备了基本的工具来控制进程运行的时间。在继续深入了解信号处理之前，让我们首先查看一下如何处理其他终止需求，例如在有限的时间内不断捕获信息，直到文件大小达到某个值时终止作业，或者当文件包含特定字符串时终止作业。循环可以很好地执行这类任务，例如 for、while 或 until，同时使用 sleep 命令提供的内置延迟反复执行循环。如果您需要使用低于秒的时间粒度，还可以使用 usleep 命令。

您还可以向时钟添加秒针，并自定义颜色。使用 showrgb 命令可以查看可用的颜色名称。假设使用xclock -bg Thistle -update 1& 命令启动背景为蓟色的具有秒针的时钟。

现在您可以使用循环结合已学到的知识捕获每秒的时钟外观图像，然后组合这些图像来制作动画 GIF 图像。清单 7 展示了如何使用 xwininfo 命令查找 xclock 命令的窗口 id。然后使用 ImageMagick 命令行工具捕获 60 个时钟外观图像（时间间隔为 1 秒）（参阅 参考资料 中有关 ImageMagick 的详细内容）。最后，将这些图像组合成无限循环的 GIF 文件，其大小是原始时钟的一半。

清单 7. 每秒捕获一次图像

```
[ian@attic4 ~]$ cat getclock.sh
#!/bin/bash
windowid=$(xwininfo -name "xclock"| grep '"xclock"' | awk '{ print $4 }
')
sleep 5
for n in `seq 10 69`; do
  import -frame  -window $windowid clock$n.gif&
  sleep 1s
#  usleep 998000
done
convert -resize 50% -loop 0 -delay 100 clock?[0-9].gif clocktick.gif
[ian@attic4 ~]$ ./getclock.sh
[ian@attic4 ~]$ file clocktick.gif
clocktick.gif: GIF image data, version 89a, 87 x 96
```

这种类型的计时总是会发生变化，因此将获取时钟图像的 import 命令放在后台中运行，从而避免使用主要的 shell 脚本跟踪时间。但是，这个过程会发生一些时间偏差，因为需要花费一些时间启动每个子 shell 脚本来进行后台处理。本例中存在一个 5 秒的延迟，即从开始启动 shell 脚本，然后单击时钟使其显示在桌面上，这一过程需要花费 5 秒钟的时间。即使注意了这些事项，时钟运行时仍然丢失了一个滴答，并额外复制了一个起始的滴答，因为脚本的运行略微超过了 60 秒。解决这种问题的一种方法是使用时间粒度为微秒级的 usleep 命令，这种时间粒度远远小于 1 秒，从而解决延迟问题。如脚本中的注释行所示。如果一切如预期运行，那么输出图像应该类似于图 2 所示。

图 2. 滴答中的 xclock

这个例子展示了如何以固定时间间隔获取一定数量的系统条件快照。使用本文所述的技巧，您可以获取其他系统条件的快照。您可能希望检查输出文件的大小以确保没有超出限制，或者检查某个文件是否包含了特定的消息，或者使用 vmstat 这样的命令检查系统状态。您可以尽情发挥自己的想象力。

信号和 trap

如果亲自运行清单 7 的 getclock.sh 脚本，并在运行脚本时关闭时钟窗口，脚本会继续运行但是会在每次尝试获取时钟窗口快照时打印错误信息。类似地，如果运行清单 4 中的 runclock3.sh 脚本，并在运行脚本的终端窗口中按下 **Ctrl-c**，脚本会立即终止，但是没有关闭时钟。要解决这类问题，脚本需要能够捕获或 *trap*[终止子进程](#) 一节讨论的信号。

如果在后台中执行 runclock3.sh 并同时运行 ps -f 命令，将看到类似清单 8 中的输出。

清单 8. runclock3.sh 的进程信息

```
[ian@attic4 ~]$ ./runclock3.sh 20s&
[1] 10101
[ian@attic4 ~]$ My PID=10101. Clock's PID=10102
UID        PID  PPID  C STIME TTY      STAT   TIME CMD
ian      10102 10101  0 06:37 pts/1    S      0:00 xclock
ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
ian       4598 12455  0 Jul29 pts/1    00:00:00 bash
ian      10101  4598  0 06:37 pts/1    00:00:00 /bin/bash ./runclock3.s
h 20s
ian      10102 10101  0 06:37 pts/1    00:00:00 xclock
ian      10104 10101  0 06:37 pts/1    00:00:00 sleep 20s
ian      10105  4598  0 06:37 pts/1    00:00:00 ps -f
[ian@attic4 ~]$ All done

[1]+  Done                    ./runclock3.sh 20s
```

注意：`ps -f` 输出具有三个和 runclock3.sh 进程（PID 10101）相关的条目。特别是，`sleep` 命令是作为独立进程运行的。处理 xclock 进程的过早终止或使用 **Ctrl-c** 终止运行脚本的方法是捕获这些信号并使用 `kill` 命令终止 `sleep` 命令。

有多种方法可以确定使用 sleep 命令的进程。清单 9 展示了最新的脚本，`runclock4.sh`。注意以下几点：

- `sleep` 命令在后台显式运行。
- `wait` 命令用于等待 `sleep` 命令的终止。
- 第一个 `trap` 命令将使 `stopsleep` 函数在接受到 SIGCHLD、SIGINT 或 SIGTERM 信号时运行。sleeper 进程的 PID 将作为参数传递。
- `stopsleep` 函数将作为信号结果运行。它将打印状态信息并将 `sleep` 命令发送给 SIGINT 信号。
- 当 `sleep` 命令终止后，将完成 `wait` 命令。然后清除 trap，终止 `xclock` 命令。

清单 9. 使用 runclock4.sh 以 trap 信号

```
[ian@attic4 ~]$ cat runclock4.sh
#!/bin/bash

stopsleep() {
  sleeppid=$1
  echo "$(date +'%T') Awaken $sleeppid!"
  kill -s SIGINT $sleeppid >/dev/null 2>&1
}

runtime=${1:-10m}
mypid=$$
# Enable immediate notification of SIGCHLD
set -bm
# Run xclock in background
xclock&
clockpid=$!
#Sleep for the specified time.
sleep $runtime&
sleeppid=$!
echo "$(date +'%T') My PID=$mypid. Clock's PID=$clockpid sleep PID=$sle
eppid"
# Set a trap
trap 'stopsleep $sleeppid' CHLD INT TERM
# Wait for sleeper to awaken
wait $sleeppid
# Disable traps
trap SIGCHLD
trap SIGINT
trap SIGTERM
# Clean up child (if still running)
echo "$(date +'%T') terminating"
kill -s SIGTERM $clockpid >/dev/null 2>&1 && echo "$(date +'%T') Stoppi
ng $clockpid"
echo "$(date +'%T') All done"
```

清单 10 展示了三次运行 `runclock4.sh` 的输出。第一次，一切正常完成。第二次，`xclock` 提前终止。第三次，使用 **Ctrl-c** 终止 shell 脚本。

清单 10. 使用不同方法停止 runclock4.sh

```
[ian@attic4 ~]$ ./runclock4.sh 20s
09:09:39 My PID=11637. Clock's PID=11638 sleep PID=11639
09:09:59 Awaken 11639!
09:09:59 terminating
09:09:59 Stopping 11638
09:09:59 All done
[ian@attic4 ~]$ ./runclock4.sh 20s
09:10:08 My PID=11648. Clock's PID=11649 sleep PID=11650
09:10:12 Awaken 11650!
09:10:12 Awaken 11650!
[2]+  Interrupt               sleep $runtime
09:10:12 terminating
09:10:12 All done
[ian@attic4 ~]$ ./runclock4.sh 20s
09:10:19 My PID=11659. Clock's PID=11660 sleep PID=11661
09:10:22 Awaken 11661!
09:10:22 Awaken 11661!
09:10:22 Awaken 11661!
[2]+  Interrupt               sleep $runtime
09:10:22 terminating
09:10:22 Stopping 11660
./runclock4.sh: line 31: 11660 Terminated              xclock
09:10:22 All done
```

注意，“Awaken” 消息显示了 `stopsleep` 函数的调用次数。如果想细究其原因，可以尝试针对每种终止类型获得该函数的单独副本，研究产生额外调用的原因。

您还将注意到一些业务控制信息将通知 `xclock` 命令和 `sleep` 命令的终止。当使用默认的 bash 终端设置在后台运行作业时，bash 通常会捕获 SIGCHLD 信号并在打印下一个终端输出行*之后* 打印消息。脚本中的 `set -bm` 命令将通知 bash 立即报告 SIGCHLD 信号并启用业务控制监视。下一小节的闹钟示例将展示如何禁用这些消息。

闹钟

我们最后一个应用将返回到本文最初的问题上：如何录制广播节目。我们将实际构建一个闹钟。如果当地法律允许录制这类内容，您可以构建一个录制程序而不需添加 `vsound` 这样的程序。

对于这个示例，我们将使用 GNOME rhythmbox 应用程序进行演示。即使您使用的是其他媒体播放器，仍然可从该示例获益。

闹钟应该能够发出您希望的任何声音，包括播放自己的 CD、MP3 文件。在北卡罗莱纳州中部，我们具有一个 广播电台 WCPE，该电台全天播放古典音乐。除了广播外，WCPE 还以不同的格式连入互联网，包括 Ogg Vorbis。您可以选择自己喜欢的流媒体源。

要从 X Windows 终端会话中启动 rhythmbox，播放 WCPE Ogg Vorbis 流，需要使用清单 11 所示的命令。

清单 11. 使用 WCPE Ogg Vorbis 流启动 rhythmbox

```
rhythmbox --play http://audio-ogg.ibiblio.org:8000/wcpe.ogg
```

`rhythmbox` 的第一个有趣特性是运行中的程序可以对命令作出响应，包括表示终止的命令。因此，这里不需要使用 `kill` 命令终止应用程序，但是您可以根据需要使用该命令。

第二个有趣特性是，和我们在前面示例中使用的时钟一样，大多数媒体播放器都需要使用图形化表示。通常，当您不在计算机附近时，一般使用 `cron` 和 `at` 实用程序运行命令，这些程序通常会认为预定的作业*不能* 访问播放器的图形化表示。`rhythmbox` 命令允许您指定要使用的图形化表示。您很可能需要进行登录，即使屏幕被锁定，但是您可以亲自研究这些不同变化。清单 12 展示了 `alarmclock.sh` 脚本，可用作闹钟工作的基础。其中包含指定运行时间的单个参数，默认值为一小时。

清单 12. 闹钟 - alarmclock.sh

```
[ian@attic4 ~]$ cat alarmclock.sh
#!/bin/bash

cleanup () {
  mypid=$1
  echo "$(date +'%T') Finding child pids"
  ps -eo ppid=,pid=,cmd= --no-heading | grep "^ *$mypid"
  ps $playerpid >/dev/null 2>&1 && {
    echo "$(date +'%T') Killing rhythmbox";
    rhythmbox --display :0.0 -quit;
    echo "$(date +'%T') Killing rhythmbox done";
  }
}

stopsleep() {
  sleeppid=$1
  echo "$(date +'%T') stopping $sleeppid"
  set +bm
  kill $sleeppid >/dev/null 2>&1
}

runtime=${1:-1h}
mypid=$$
set -bm
rhythmbox --display :0.0 --play http://audio-ogg.ibiblio.org:8000/wcpe.
ogg&
playerpid=$!
sleep $runtime& >/dev/null 2>&1
sleeppid=$!
echo "$(date +'%T') mypid=$mypid player pid=$playerpid sleeppid=$sleepp
id"
trap 'stopsleep $sleeppid' CHLD INT TERM
wait $sleeppid
echo "$(date +'%T') terminating"
trap SIGCHLD
trap SIGINT
trap SIGTERM
cleanup $mypid final
wait
```

注意 `stopsleep` 函数中 `set +bm` 的用法，它将重置作业控制设置并禁止显示 `runclock4.sh` 中出现的消息。

清单 13 展示了一个示例 `crontab`，它将在每星期周一到周五的上午六点到七点、每个周六上午七点到九点、以及每个周日上午八点半到十点之间运行闹钟程序。

清单 13. 运行闹钟程序的示例 crontab

```
0 6 * * 1-6 /home/ian/alarmclock.sh 1h
0 7 * * 7 /home/ian/alarmclock.sh 2h
30 8 * * 0 /home/ian/alarmclock.sh 90m
```

请参考前一篇技巧文章 Job scheduling with cron and at，学习如何针对新的闹钟程序设置 crontab。

在更复杂任务中，可能具有多个子线程。cleanup 例程将展示如何使用 `ps` 命令查找脚本进程的子进程。您可以在此应用之上进行扩展，循环遍历任意的子进程集合并一一终止它们。

结束语

如果希望了解更多 Linux 管理任务，请参阅教程 “LPI 102 考试准备：管理任务” 或参考以下 参考资料。不要忘记 对本文作出评价，告诉我们您感兴趣的技巧。