系统管理就像日常生活一样。就像刷牙和吃蔬菜一样，日常的维护能保持机器的良好状态。您必须定期清空废物，比如临时文件或无用的日志文件，以及花时间填写表单、回复电话、下载更新和监控进程等。幸好自动化 shell 脚本、使用 Nagios 等工具进行监控、通过常见的 cron 进行任务调度可以减轻这个负担。

但奇怪的是，这些工具没有一个具有*响应性*。当然，您可以安排一个频繁运行的 cron 任务来监控条件，但这样繁忙的轮询 — 消耗大量资源并且具有不确定性 — 并不是很理想。例如，如果您必须监控输入数据的几个 Transfer Protocol（FTP）收存箱，您可能要通过 find 命令扫描每个目标目录，列举新的内容。然而，尽管这个操作看起来并没有什么害处，但每个调用都产生一个新的 shell 和 find 命令，这需要许多系统调用来打开目录，然后扫描目录，等等。这会造成过于频繁的或大量的轮询任务（更糟糕的是，繁忙的轮询并不总是很好。想象一下一个文件系统浏览器，比如 Mac OS X 的 Finder，轮询更新时需要的大量资源及其复杂性）。

那么，管理员应该怎么办呢？令人高兴的是，您可以再次求助于可以信赖的计算机。

了解 inotify

Inotify 是一个 Linux 内核特性，它监控文件系统，并且及时向专门的应用程序发出相关的事件警告，比如删除、读、写和卸载操作等。您还可以跟踪活动的源头和目标等细节。

使用 inotify 很简单：创建一个文件描述符，附加一个或多个监视器（一个*监视器* 是一个路径和一组事件），然后使用 read() 方法从描述符获取事件信息。read() 并不会用光整个周期，它在事件发生之前是被阻塞的。

更好的是，因为 inotify 通过传统的文件描述符工作，您可以利用传统的 select() 系统调用来被动地监控监视器和许多其他输入源。两种方法 — 阻塞文件描述符和使用 select()— 都避免了繁忙轮询。

现在，让我们深入了解 inotify，写一些 C 代码，然后看看一组命令行工具，您可以构建并使用它们将命令和脚本附加到文件系统事件。Inotify 不会在中途失去控制，但它可以运行 cat 和 wget，并且在必要时严格执行。

要使用 inotify，您必须具备一台带有 2.6.13 或更新内核的 Linux 机器（以前的 Linux 内核版本使用更低级的文件监控器 *dnotify*）。如果您不知道内核的版本，请转到 shell，输入 uname -a：

```
% uname -a
Linux ubuntu-desktop 2.6.24-19-generic #1 SMP ... i686 GNU/Linux
```

如果列出的内核版本不低于 2.6.13，您的系统就支持 inotify。您还可以检查机器的 /usr/include/sys/inotify.h 文件。如果它存在，表明您的内核支持 inotify。

注意：FreeBSD 和 Mac OS X 提供一个类似于 inotify 的 *kqueue*。在 FreeBSD 机器上输入 man 2 kqueue 获取更多信息。

本文基于 Ubuntu Desktop version 8.04.1（即 *Hardy*），它运行在 Mac OS X version 10.5 Leopard 的 Parallels Desktop version 3.0。

回页首

inotify C API

Inotify 提供 3 个系统调用，它们可以构建各种各样的文件系统监控器：

- inotify_init() 在内核中创建 inotify 子系统的一个实例，成功的话将返回一个文件描述符，失败则返回 -1。就像其他系统调用一样，如果 inotify_init() 失败，请检查 errno 以获得诊断信息。
- 顾名思义，inotify_add_watch() 用于添加监视器。每个监视器必须提供一个路径名和相关事件的列表（每个事件由一个常量指定，比如 IN_MODIFY）。要监控多个事件，只需在事件之间使用逻辑操作符*或* — C 语言中的管道线（|）操作符。如果 inotify_add_watch() 成功，该调用会为已注册的监视器返回一个惟一的标识符；否则，返回 -1。使用这个标识符更改或删除相关的监视器。

- inotify_rm_watch() 删除一个监视器。

此外，还需要 read() 和 close() 系统调用。如果描述符由 inotify_init() 生成，则调用 read() 等待警告。假设有一个典型的文件描述符，应用程序将阻塞对事件的接收，这些事件在流中表现为数据。文件描述符上的由 inotify_init() 生成的通用 close() 删除所有活动监视器，并释放与 inotify 实例相关联的所有内存（这里也用到典型的引用计数警告。与实例相关联的所有文件描述符必须在监视器和 inotify 消耗的内存被释放之前关闭）。

这个强大的工具提供 3 个应用程序编程接口（API）调用，以及简单、熟悉的范例 “所有内容都是文件”。现在，我们看看示例应用程序。

示例应用程序：事件监控

清单 1 是一个监控两个事件的目录的简短 C 程序：文件的创建和删除。

清单 1. 简单的 inotify 应用程序，它监控创建、删除和修改事件的目录

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/inotify.h>

#define EVENT_SIZE  ( sizeof (struct inotify_event) )
#define BUF_LEN     ( 1024 * ( EVENT_SIZE + 16 ) )

int main( int argc, char **argv )
{
  int length, i = 0;
  int fd;
  int wd;
  char buffer[BUF_LEN];

  fd = inotify_init();

  if ( fd < 0 ) {
    perror( "inotify_init" );
  }

  wd = inotify_add_watch( fd, "/home/strike",
                          IN_MODIFY | IN_CREATE | IN_DELETE );
  length = read( fd, buffer, BUF_LEN );

  if ( length < 0 ) {
    perror( "read" );
  }
```

```
  while ( i < length ) {
    struct inotify_event *event = ( struct inotify_event * ) &buffer[ i ];
    if ( event->len ) {
      if ( event->mask & IN_CREATE ) {
        if ( event->mask & IN_ISDIR ) {
          printf( "The directory %s was created.\n", event->name );
        }
        else {
          printf( "The file %s was created.\n", event->name );
        }
      }
      else if ( event->mask & IN_DELETE ) {
        if ( event->mask & IN_ISDIR ) {
          printf( "The directory %s was deleted.\n", event->name );
        }
        else {
          printf( "The file %s was deleted.\n", event->name );
        }
      }
      else if ( event->mask & IN_MODIFY ) {
        if ( event->mask & IN_ISDIR ) {
          printf( "The directory %s was modified.\n", event->name );
        }
        else {
          printf( "The file %s was modified.\n", event->name );
        }
      }
    }
    i += EVENT_SIZE + event->len;
  }

  ( void ) inotify_rm_watch( fd, wd );
  ( void ) close( fd );

  exit( 0 );
}
```

这个应用程序通过 fd = inotify_init(); 创建一个 inotify 实例，并添加一个监视器来监控修改、新文件和 /home/strike 中的损坏文件（由 wd = inotify_add_watch(...) 指定）。read() 方法在一个或多个警告到达之前是被阻塞的。警告的详细内容 — 每个文件、每个事件 — 是以字节流的形式发送的；因此，应用程序中的循环将字节流转换成一系列事件结构。

在文件 /usr/include/sys/inotify.h. 中，您可以找到事件结构的定义，它是一种 C 结构，如清单 2 所示。

清单 2. 事件结构的定义

```
struct inotify_event
{
  int wd;           /* The watch descriptor */
  uint32_t mask;      /* Watch mask */
  uint32_t cookie;   /* A cookie to tie two events together */
  uint32_t len;       /* The length of the filename found in the name field
*/
  char name __flexarr;   /* The name of the file, padding to the end with
NULs */
}
```

wd 字段是指与事件相关联的监视器。如果每个 inotify 有一个以上的实例，您可以使用这个字段确定如何继续以后的处理过程。mask 字段由几个部分组成，它说明发生的事情。分别测试每个部分。

当把一个文件从一个目录移动到另一个目录时，您可以使用 cookie 将两个事件绑在一起。仅当您监视源和目标目录时，inotify 才生成两个移动事件 — 分别针对源和目标 —，并通过设置 cookie 将它们绑定在一起。要监视一个移动操作，必须指定 IN_MOVED_FROM 或 IN_MOVED_TO，或使用简短的 IN_MOVE，它可以监视两个操作。使用 IN_MOVED_FROM 和 IN_MOVED_TO 来测试事件类型。

最后，name 和 len 包含文件的名称（但不包括路径）和受影响文件的名称的长度。

构建示例应用程序代码

要构建这些代码，请将目录 /home/strike 更改到您的主目录，即将这些代码保存到一个文件中，然后调用 C 编译器 — 在大部分 Linux 系统中为 gcc。然后，运行这个可执行文件，如清单 3 所示。

清单 3. 运行可执行文件

```
% cc -o watcher watcher.c
% ./watcher
```

在监视程序运行时，打开第二个终端窗口并使用 touch、cat 和 rm 来更改主目录的内容，如清单 4 所示。完成之后，重新启动您的新应用程序。

清单 4. 使用 touch、cat 和 rm

```
% cd $HOME
% touch a b c
The file a was created.
The file b was created.
The file c was created.

% ./watcher  &
% rm a b c
The file a was deleted.
The file b was deleted.
The file c was deleted.

% ./watcher  &
% touch a b c
The file a was created.
The file b was created.
The file c was created.

% ./watcher  &
% cat /etc/passwd >> a
The file a was modified.

% ./watcher  &
% mkdir d
The directory d was created.
```

试用其他可用的监视标志。要捕捉权限的更改，请将 IN_ATTRIB 添加到 mask。

使用 inotify 的技巧

您还可以使用 select()、pselect()、poll() 和 epoll() 来避免阻塞。如果您想将监视器的监控作为图形应用程序的主事件处理循环的一部分，或作为监视其他输入连接的守护进程的一部分，这是很有用的。将该 inotify 描述符添加到这组描述符中，进行并发监控。清单 5 展示了 select() 的标准形式。

清单 5. select() 的标准形式

```
int return_value;
fd_set descriptors;
struct timeval time_to_wait;
```

```
FD_ZERO ( &descriptors );
FD_SET( ..., &descriptors );
FD_SET ( fd, &descriptors );

...

time_to_wait.tv_sec = 3;
time.to_waittv_usec = 0;

return_value = select ( fd + 1, &descriptors, NULL, NULL, &time_to_wait);

if ( return_value < 0 ) {
    /* Error */
}

else if ( ! return_value ) {
    /* Timeout */
}

else if ( FD_ISSET ( fd, &descriptors ) ) {
    /* Process the inotify events */
    ...
}

else if ...
```

select() 方法在 time_to_wait 期间暂停程序。然而，如果在这个延迟期间这组描述符的任意一个文件描述符发生活动，将立即恢复执行程序。否则，调用就会超时，允许应用程序执行其他进程，比如在图形用户界面（GUI）工具中响应鼠标或键盘事件。

下面是使用 inotify 的其他技巧：

- 如果监视中的文件或目录被删除，它的监视器也会被自动删除（在删除事件发出之后）。
- 如果在已卸载的文件系统上监控文件或目录，监视器将在删除所有受影响的监视之前收到一个卸载事件。
- 将 IN_ONESHOT 标志添加到监视器标记中，设置一个一次性警告。警告在发送之后将被删除。
- 要修改一个事件，必须提供相同的路径名和不同的标记。新监视器将取代老监视器。
- 考虑到实用性，不可能耗尽任何一个 inotify 实例的监视器。然而，您可能会耗尽事件队列的空间，这取决于处理事件的频率。队列溢出会引起 IN_Q_OVERFLOW 事件。
- close() 方法毁坏 inotify 实例和所有相关联的监视器，并清空队列中的所有等待事件。

回页首

安装 inotify 工具套件

inotify 编程界面很容易使用，但如果您不想编写自己的工具，可以使用一种开源的灵活的代替方法。Inotify 工具库（参见下面的 参考资料 获得链接）提供一对监控文件系统活动的命令行实用程序：

- inotifywait 仅执行阻塞，等待 inotify 事件。您可以监控任何一组文件和目录，或监控整个目录树（目录、子目录、子目录的子目录等等）。在 shell 脚本中使用 inotifywait。
- inotifywatch 收集关于被监视的文件系统的统计数据，包括每个 inotify 事件发生多少次。

在撰写本文时，最新版本的 inotify 库是 version 3.13，于 2008 年 1 月发布。安装 inotify 工具有两种方法：可以下载并亲自构建该软件，或使用 Linux 发布版的包管理器安装一组二进制文件（如果已知库包含 inotify 工具）。要在基于 Debian 的发布版上使用后一种方法，请运行 apt-cache search inotify，并查找匹配的工具，如清单 6 所示。在本文的示例系统 Ubuntu Desktop version 8.04 上，这些工具已经可用。

清单 6. 搜索 inotify 工具

```
% apt-cache search inotify
incron - cron-like daemon which handles filesystem events
inotail - tail replacement using inotify
inoticoming - trigger actions when files hit an incoming directory
inotify-tools - command-line programs providing a simple interface to inotify
iwatch - realtime filesystem monitoring program using inotify
libinotify-ruby - Ruby interface to Linux's inotify system
libinotify-ruby1.8 - Ruby interface to Linux's inotify system
libinotify-ruby1.9 - Ruby interface to Linux's inotify system
libinotifytools0 - utility wrapper around inotify
libinotifytools0-dev - Development library and header files for libinotifytools0
liblinux-inotify2-perl - scalable directory/file change notification
muine-plugin-inotify - INotify Plugin for the Muine music player
python-kaa-base - Base Kaa Framework for all Kaa Modules
python-pyinotify - Simple Linux inotify Python bindings
python-pyinotify-doc - Simple Linux inotify Python bindings
% sudo apt-get install inotify-tools
...
Setting up inotify-tools.
```

但是构建代码也是很容易的。下载并解压缩源文件；然后配置、编译和安装它，如清单 7 所示。整个过程可能需要 3 分钟。

清单 7. 构建代码

```
% wget \
```

```
    http://internap.dl.sourceforge.net/sourceforge/inotify-tools/inotify-tools-3.13.tar.gz
% tar zxvf inotify-tools-3.13.tar.gz
inotify-tools-3.13/
inotify-tools-3.13/missing
inotify-tools-3.13/src/
inotify-tools-3.13/src/Makefile.in
...
inotify-tools-3.13/ltmain.sh

% cd inotify-tools.3.13
% ./configure
% make
% make install
```

现在，您可以使用这个工具了。例如，如果您想监控整个主目录的更改，请运行 inotifywait。最简单的调用是 inotifywait -r -m，它循环监控参数（-r），并使该实用程序在每个事件（-m）之后保持运行：

```
% inotifywait -r  -m $HOME
Watches established.
```

运行另一个终端窗口，并修改您的主目录。有趣的是，即使一个简单的通过 Is 列出的目录也生成一个事件：

```
/home/strike OPEN,ISDIR
```

阅读 inotifywait 手册页获得将事件限制到特定列表（反复地使用 -e *event_name* 选项来创建列表）的选项，并从循环的监视器中排除匹配的文件（--exclude *pattern*）。

回页首

继续使用 inotify

就像上面揭示的 apt-cache ，您还可以考虑使用许多其他基于 inotify 的实用程序。incron 实用程序源自于 cron，但它响应 inotify 事件，而不是调度。inoticoming 实用程序专门用于监控收存箱。如果您是 Perl、Ruby 或 Python 开发人员，您可以找到从您喜欢的脚本语言调用 inotify 的模块和库。

例如，Perl 编程人员可以使用 Linux::Inotify2（参见 参考资料 获得详细信息）来将 inotify 特性嵌入到任何 Perl 应用程序中。这些取自 Linux::Inotify2 README 文件的代码演示了监控事件的回调接口，如清单 8 所示。

清单 8. 监控事件的回调接口

```
use Linux::Inotify2;

my $inotify = new Linux::Inotify2
    or die "Unable to create new inotify object: $!";

 # for Event:
 Event->io (fd =>$inotify->fileno, poll => 'r', cb => sub
{ $inotify->poll });

 # for Glib:
 add_watch Glib::IO $inotify->fileno, in => sub { $inotify->poll };

 # manually:
 1 while $inotify->poll;

 # add watchers
 $inotify->watch ("/etc/passwd", IN_ACCESS, sub {
    my $e = shift;
    my $name = $e->fullname;
    print "$name was accessed\n" if $e->IN_ACCESS;
    print "$name is no longer mounted\n" if $e->IN_UNMOUNT;
    print "$name is gone\n" if $e->IN_IGNORED;
    print "events for $name have been lost\n" if $e->IN_Q_OVERFLOW;

    # cancel this watcher: remove no further events
    $e->w->cancel;
 });
```

因为 Linux 中的所有东西都是一个文件，所以您将发现 inotify 有大量的用法。