Administer Linux on the fly

Use the /proc filesystem to get a handle on your system

Level: Intermediate

Graham White (gwhite@uk.ibm.com), IT specialist, Hursley, IBM

14 May 2003

The /proc filesystem is one of Linux's great features, and this article gives you a thorough grounding in some of its most useful aspects. With it, you can administer many details of the operating system without ever having to shut down and reboot the machine, which is a boon for those who need to keep their systems as available as possible.

Anyone who has administered a system of commercial importance knows the value of uptime -- or, conversely, knows the headaches you get from users because of downtime. One of the main reasons a company will run a UNIX server is because of its reliability and stability. If managed carefully, there's usually no need to restart these servers for long periods of time. And to improve matters further, there are administrative tasks -- even at the kernel level -- that you can perform on the fly, keeping your servers available. While you may still need to restart a system to upgrade hardware or if someone trips over the power cord, it's good to know that many administrative tasks can be performed without disrupting service.

This article includes hints and tips for performing various administrative tasks and changing your system *without rebooting*. Linux provides various ways to change underlying operating system values and settings while keeping the system up and running. These come in two basic forms, those that are general to all Linux systems and are provided in the Linux kernel (you can find more information about the Linux kernel and download kernel source at the Linux Kernel Archives; see Resources for a link), and those that are distribution specific and provided by the vendor. This article deals with both types.

Note: This article is written for a 2.4-level kernel. Some of the options and features may be different for other kernel versions.

Changing running kernel parameters

Linux provides a really neat way for administrators to change the kernel while the system is running and without the need to reboot the kernel/system. This is done with a virtual filesystem called `/proc`. Linux Gazette provides one of the simplest and easiest references on `/proc` I have seen. (See Resources for a link.) Very basically, the `/proc` filesystem gives you a view into the running kernel, which can be useful for monitoring performance, discovering system information, finding out how the system is configured, and changing that configuration. This filesystem is called a *virtual filesystem*, because it is not really a filesystem at all. It's just a map provided by the kernel that is attached to your usual filesystem structure to give you access to it.

The fact that we have some way of changing the running kernel parameters while the system is up and running gives the system administrator great power and flexibility in changing kernel settings. This sort of implementation was an inspired idea on the part of the Linux kernel developers. But can too much power be a bad thing? Sometimes. If you are going to change anything in the `/proc` filesystem, you **must** make sure that you know what you are changing and what effect this will have on the system. These are really useful techniques, but a wrong move can give you some rather undesired consequences. If you are new to this sort of thing or are not sure what effect one of your changes will have, practice on a machine that is not important to you or your business.

How to make changes

First, think about how *not* to make changes to the kernel. There are two good reasons why you should not just jump into the `/proc` filesystem, open a file in your text editor, make a bunch of changes, and save the file back out again. These are:

- Data integrity: All of these files represent the running system, and since the kernel may change any of these files at any time, if you open an editor and change some data while the system is changing it underneath you, whatever you save back is unlikely to be what the kernel is expecting.
- Virtual files: All of these files do not actually exist. How would the saved data be synchronized, etc.?

The answer to making changes to any of these files, therefore, is not to use an editor. When making changes to anything at all in the `/proc` filesystem, you should use the `echo` command and redirect the output from the command line into your chosen files under `/proc`. For example:

```
echo "Your-New-Kernel-Value" > /proc/your/file
```

Similarly, if you wish to view information from `/proc`, you should either use a command that is designed for the purpose or use the command line `cat` command.

What to change

You do not need to be a kernel hacker to get good use out of `/proc`, and a basic understanding of the structure of this filesystem will aid you greatly. You may find that you don't need to know about anything in `/proc`, until the day a user asks you for a certain bit of functionality that makes you glad you bothered to learn where to look to make changes. The `/proc` filesystem helps the system administrator in this respect via its structure and file permissions.

Each file in `/proc` has a very particular set of file permissions assigned to it and will be owned by a particular user ID. This is very carefully done so that the correct functionality is presented to the administrator and to the users. The following list summarizes what particular permissions may do on individual files:

- Read-only: File is not changeable by any user; used for presenting system information
- Root-write: If a file is writeable in `/proc`, it is usually writeable only by the root user
- Root-read: Some files may not be viewable to normal system users, only to root
- Other: You may find combinations other than the common three, above, for various reasons

A very broad generalization about `/proc` is that you will find most of it read-only except for the `/proc/sys` directory. This directory is the one that holds most kernel parameters (rather than information) and is the one that is designed to be changed while the system is running. As a result, this is the directory that this article will look mainly at.

The last thing to know about learning what to change in `/proc` is what you should actually be writing to these files. You will notice as you look at various files in `/proc` that some of them are human readable and some are data files. The data files can still be read by using specific utilities such as `top`, `lspci`, and `free`. You will also notice that the human-readable files take two different formats: some are binary switches and others contain more information. The binary switch files only contain a zero (off) or a one (on) for that particular kernel feature.

Making changes

Detailing the exact information and usage of each file in `/proc` is outside the scope of this article. For more information about any `/proc` files not discussed in this article, one of the best sources is the Linux kernel source itself, which contains some very good documentation. The following files in `/proc` are more useful to a system administrator. This is not meant to be an exhaustive treatment but an easy-access reference for day-to-day use.

/proc/scsi

/proc/scsi/scsi
One of the most useful things to learn as a system administrator is how to add more disk space if you have hot-swap drives available to you, without rebooting the system. Without using `/proc`, you could insert your drive, but you would then have to reboot in order to get the system to recognize the new disk. Here, you can get the system to recognize the new drive with the following command:

```
echo "scsi add-single-device w x y z" > /proc/scsi/scsi
```

For this command to work properly, you must get the parameter values w, x, y, and z correct, as follows:

- w is the host adapter ID, where the first adapter is zero (0)
- x is the SCSI channel on the host adaptor, where the first channel is zero (0)
- y is the SCSI ID of the device
- z is the LUN number, where the first LUN is zero (0)

Once your disk has been added to the system, you can mount any previously formatted filesystems or you can start formatting it, and so on. If you are not sure about what device the disk will be, or you want to check any pre-existing partitions, for example, you can use a command such as `fdisk -l`, which will report this information back to you.

Conversely, the command to remove a device from your system without a reboot would be:

```
echo "scsi remove-single-device w x y z" > /proc/scsi/scsi
```

Before you enter this command and remove your hot-swap SCSI disk from your system, make sure you have unmounted any filesystems from this disk first.

/proc/sys/fs/

/proc/sys/fs/file-max
This specifies the maximum number of file handles that can be allocated. You may need to increase this value if users get error messages stating that they cannot open more files because the maximum number of open files has been reached. This can be set to any number of files and can be changed by writing a new numeric value to the file.

Default setting: 4096

/proc/sys/fs/file-nr
This file is related to file-max and holds three values:

- Number of allocated file handles
- Number of used file handles
- Maximum number of file handles

This file is read-only and for informational purposes only.

/proc/sys/fs/inode-*
Any files starting with the name "inode" will perform the same operation as files starting with the name "file" as above, but perform their operation relative to inodes instead of file handles.

/proc/sys/fs/overflowuid and **/proc/sys/fs/overflowgid**
This holds the User ID (UID) and Group ID (GID) for any filesystems that support 16-bit user and group IDs. These values can be changed, but if you really do find the need to do this, you might find it easier to change your group and password file entries instead.

Default Setting: 65534

/proc/sys/fs/super-max
This specifies the maximum number of super block handlers. Any filesystem you mount needs to use a super block, so you could possibly run out if you mount a lot of filesystems.

Default setting: 256

/proc/sys/fs/super-nr
This shows the currently allocated number of super blocks. This file is read-only and for informational purposes only.

/proc/sys/kernel

/proc/sys/kernel/acct
This holds three configurable values that control when process accounting takes place based on the amount of free space (as a percentage) on the filesystem that contains the log:

1. If free space goes below this percentage value then process accounting stops
2. If free space goes above this percentage value then process accounting starts
3. The frequency (in seconds) at which the other two values will be checked

To change a value in this file you should echo a space separated list of numbers.

Default setting: 2 4 30

These values will stop accounting if there is less than 2 percent free space on the filesystem that contains the log and starts it again if there is 4 or more percent free space. Checks are made every 30 seconds.

/proc/sys/kernel/ctrl-alt-del
This file holds a binary value that controls how the system reacts when it receives the ctrl+alt+delete key combination. The two values represent:

1. A zero (0) value means the ctrl+alt+delete is trapped and sent to the init program. This will allow the system to have a graceful shutdown and restart, as if you typed the shutdown command.
2. A one (1) value means the ctrl+alt+delete is not trapped and no clean shutdown will be performed, as if you just turned the power off.

Default setting: 0

/proc/sys/kernel/domainname
This allows you to configure your network domain name. This has no default value and may or may not already be set.

/proc/sys/kernel/hostname
This allows you to configure your network host name. This has no default value and may or may not already be set.

/proc/sys/kernel/msgmax
This specifies the maximum size of a message that can be sent from one process to another process. Messages

are passed between processes in kernel memory that is not swapped out to disk, so if you increase this value, you will increase the amount of memory used by the operating system.

Default setting: 8192

/proc/sys/kernel/msgmnb
This specifies the maximum number of bytes in a single message queue.

Default setting: 16384

/proc/sys/kernel/msgmni
This specifies the maximum number of message queue identifiers.

Default setting: 16

/proc/sys/kernel/panic
This represents the amount of time (in seconds) the kernel will wait before rebooting if it reaches a "kernel panic." A setting of zero (0) seconds will disable rebooting on kernel panic.

Default setting: 0

/proc/sys/kernel/printk
This holds four numeric values that define where logging messages are sent, depending on their importance. For more information on different log levels, read the manpage for syslog(2). The four values of the file are:

1. Console Log Level: messages with a higher priority than this value will be printed to the console
2. Default Message Log Level: messages without a priority will be printed with this priority
3. Minimum Console Log Level: minimum (highest priority) value that the Console Log Level can be set to
4. Default Console Log Level: default value for Console Log Level

Default setting: 6 4 1 7

/proc/sys/kernel/shmall
This is the total amount of shared memory (in bytes) that can be used on the system at any given point.

Default setting: 2097152

/proc/sys/kernel/shmax
This specifies the largest shared memory segment size (in bytes) allowed by the kernel.

Default setting: 33554432

/proc/sys/kernel/shmmni
This represents the maximum number of shared memory segments for the whole system.

Default setting: 4096

/proc/sys/kernel/sysrq
This activates the System Request Key, if non-zero.

Default setting: 0

/proc/sys/kernel/threads-max
This is the maximum number of threads that can be used by the kernel.

Default setting: 2048

/proc/sys/net

/proc/sys/net/core/message_burst
This is the time required (in 1/10 seconds) to write a new warning message; other warning messages received during this time will be dropped. This is used to prevent Denial of Service attacks by someone attempting to flood your system with messages.

Default setting: 50 (5 seconds)

/proc/sys/net/core/message_cost
This holds a cost value associated with every warning message. The higher the value, the more likely the warning message is to be ignored.

Default setting: 5

/proc/sys/net/core/netdev_max_backlog
This gives the maximum number of packets allowed to queue when an interface receives packets faster than the kernel can process them.

Default setting: 300

/proc/sys/net/core/optmem_max
This specifies the maximum buffer size allowed per socket.

/proc/sys/net/core/rmem_default
This is the receive socket buffer's default size (in bytes).

/proc/sys/net/core/rmem_max
This is the receive socket buffer's maximum size (in bytes).

/proc/sys/net/core/wmem_default
This is the send socket buffer's default size (in bytes).

/proc/sys/net/core/wmem_max
This is the send socket buffer's maximum size (in bytes).

/proc/sys/net/ipv4
All of the IPv4 and IPv6 parameters are fully documented in the kernel source documentation. See the file `/usr/src/linux/Documentation/networking/ip-sysctl.txt`.

/proc/sys/net/ipv6
Same as IPv4.

/proc/sys/vm

/proc/sys/vm/buffermem
This controls the amount of the total system memory (as a percent) that will be used for buffer memory. It holds three values that can be set by writing a space-separated list to the file:

1. Minimum percentage of memory that should be used for buffers
2. The system will try and maintain this amount of buffer memory when system memory is being pruned in the event of a low amount of system memory remaining
3. Maximum percentage of memory that should be used for buffers

Default setting: 2 10 60

/proc/sys/vm/freepages
This controls how the system reacts to different levels of free memory. It holds three values that can be set by writing a space-separated list to the file:

1. If the number of free pages in the system reaches this minimum limit, only the kernel will be permitted to allocate any more memory.
2. If the number of free pages in the system falls below this limit, the kernel will start swapping more aggressively to free memory and maintain system performance.
3. The kernel will try to keep this amount of system memory free. Falling below this value will start the kernel swapping.

Default setting: 512 768 1024

/proc/sys/vm/kswapd
This controls how the kernel is allowed to swap memory. It holds three values that can be set by writing a space separated list to the file:

1. Maximum number of pages the kernel tries to free at one time. If you want to increase bandwidth to/from swap, you will need to increase this number.
2. Minimum number of times the kernel tries to free a page on each swap.
3. The number of pages the kernel can write in one swap. This has the greatest impact on system performance. The larger the value, the more data can be swapped and the less time is spent disk seeking. However, a value that is too large will adversely affect system performance by flooding the request queue.

Default setting: 512 32 8

/proc/sys/vm/pagecache
This does the same job as `/proc/sys/vm/buffermem`, but it does it for memory mapping and generic caching of files.

Making your kernel settings persistent

A handy utility is provided for making changes to any kernel parameters under the `/proc/sys` directory. It allows you to make changes to the running kernel (similarly to the echo and redirection method used above), but it also has a configuration file that is executed on system boot. This lets you make changes to the running kernel and add them to the configuration file so that any changes you make will remain after a system reboot.

The utility is called `sysctl` and is fully documented in the man pages at sysctl(8). The configuration file for `sysctl` is `/etc/sysctl.conf`, which can be edited and is documented under sysctl.conf(8). `Sysctl` treats the files under `/proc/sys` as individual variables that can be changed. So, for example, the file under `/proc/sys` that represents the maximum number of file handles allowed on the system, `/proc/sys/fs/file-max`, is represented as `fs.file-max`.

This example reveals some oddities in `sysctl` notation. Since `sysctl` can only change variables under the `/proc/sys` directory, that part of the variable name is missing as the variables are always assumed to be under that directory. The next change to note is that the directory separators (slash, /) have changed to periods (dot, .).

There are two simple rules for converting between files in `/proc/sys` and variables in `sysctl`:

- Drop the `/proc/sys` from the beginning.
- Swap slashes for dots in the filenames.

These two rules will let you swap any file name in `/proc/sys` for any variable name in `sysctl`. The general file to variable conversion is:

```
/proc/sys/dir/file --> dir.file
dir1.dir2.file --> /proc/sys/dir1/dir2/file
```

You can view all the variables that are available to be changed, along with their current setting, using the command `sysctl -a`.

Variables can also be changed using `sysctl`, which does exactly the same job as the echo method used above. This notation is as follows:

```
sysctl -w dir.file="value"
```

Using the file-max example again, we could change this value to 16384 using one of two methods as follows:

```
sysctl -w fs.file-max="16384"
```

Or:

```
echo "16384" > /proc/sys/fs/file-max
```

Don't forget that `sysctl` does not add changes made to the configuration file; this is left for you to do manually. If you want your changes to persist after a reboot, you must maintain this file.

Note: Not all distributions provide `sysctl` support. If this is the case for your particular system, then you can use the echo and redirect method described above and add these commands to a start-up script so they are executed every time the system boots.

Commands for setting the system

It is possible to change other non-kernel system parameters while the system is running and also get these settings to take effect without rebooting. These can mainly be classified as services, daemons, and servers that will be listed in the `/etc/init.d` directory. Since there is an increasingly wide range of scripts that can be listed in this directory, it is not possible to go through all the different configurations here. However, below are a few examples of how the `/etc/init.d` scripts can be manipulated on different distributions of Linux. Examples of where changes to a daemon and a reload of the configuration without rebooting might be useful are:

- Changing your Web server configuration and reloading Apache
- Removing an inetd login service that you don't require
- Manipulating your network settings
- Exporting new filesystems via NFS
- Starting/stopping your firewall

First, the generic way of manipulating system services is directly, via the scripts in `/etc/init.d`. These scripts take parameters to manipulate the services that they control; you can type the script name without any parameters to see what the valid options are. Common parameters are:

- start: Starts a stopped service
- stop: Stops a running service
- restart: Stops and then starts a running service; will start a stopped service
- reload: Reloads service configuration without breaking any connection(s)
- status: Outputs whether the service is running or not

As an example, the following command would reload your xinetd configuration without terminating any connected user's sessions (useful if you make a change to /etc/xinetd.conf):

```
/etc/init.d/xinetd reload
```

Red Hat provides a command, `service`, that will manipulate services for you. The `service` command provides the same functionality as typing the script name itself. The syntax is as follows:

```
service script-name [parameter]
```

For example:

```
service xinetd reload
```

SuSE also provide a command called `rc`. This is similar to the `service` command above, but has no space between the command and the script name. The syntax is as follows:

```
rc{script-name} parameter
```

For example:

```
rcapache start
```

Similarly to changing kernel parameters, once you reboot your system, any changes made to services will be lost. More and more distributions are adopting the use of the `chkconfig` command, which manages the services that are started at various run levels (including on boot). At the time of this writing, the `chkconfig` command syntax differs slightly on different versions of Linux, but if you enter the command `chkconfig` without any parameters, you will get a list of how to use it. More information about `chkconfig` can also be found via the man pages at chkconfig(8).

Conclusion

Configuring the Linux kernel on the fly using the `/proc` filesystem isn't to be taken lightly, but once you understand its structure and how to manipulate the various files and parameters, you've gained the use of a powerful tool for keeping your servers available around the clock.

Acknowledgment

I would like to thank Mr. Adrian Fewings for proofreading this article.

Resources

- Download, get involved with, or just learn about the Linux kernel at the Linux Kernel Archives.
- Refer to the kernel documentation in the "Documentation" directory where you installed the kernel source.
- Learn more about the /proc virtual filesystem from Linux Gazette.
- Read the man pages for sysctl(8) and sysctl.conf(8).

- Find more Linux documents at the Linux Documentation Project homepage.
- For a guide to troubleshooting hardware problems, read "Linux hardware stability guide, Part 1" (*developerWorks*, March 2001) and "Linux hardware stability guide, Part 2" (*developerWorks*, July 2001).
- "Understanding Linux configuration files" (*developerWorks*, December 2001) gives an overview of files that control permissions, system applications, daemons, and more.
- If you want to know what "high availability" means from a mainframe perspective, read the IBM Redpaper "Linux on IBM zSeries and S/390: High Availability for z/VM and Linux."
- Find more resources for Linux developers in the *developerWorks* Linux zone.

About the author

Graham graduated from The University of Exeter with a B.Sc. (Honors) degree in Computer Science with Management Science in July 2000. He joined IBM as an IT support worker in September 2000 with no previous experience and started learning Linux. One year later, in September 2001, he was certified as a Red Hat Certified Engineer. His work and personal interests gave him experience with many different versions of Linux running on a wide range of platforms to support the development community at the IBM Hursley Laboratory in the UK. Recently, he has taken up writing articles about Linux, his first and only other publication being a guide for the Linux Documentation Project. Contact Graham at gwhite at uk.ibm.com.