procfs

The **proc filesystem** (**procfs**) is a special filesystem in Unix-like operating systems that presents information about processes and other system information in a hierarchical file-like structure, providing a more convenient and standardized method for dynamically accessing process data held in the kernel than traditional tracing methods or direct access to kernel memory. Typically, it is mapped to a mount point named */proc* at boot time. The proc file system acts as an interface to internal data structures in the kernel. It can be used to obtain information about the system and to change certain kernel parameters at runtime (sysctl).

Many Unix-like operating systems support the proc filesystem, including Solaris, IRIX, Tru64 UNIX, BSD, Linux, IBM AIX, QNX, and Plan 9 from Bell Labs. The Linux kernel extends it to non–process-related data.

The proc filesystem provides a method of communication between kernel space and user space. For example, the GNU version of the process reporting utility ps uses the proc file system to obtain its data, without using any specialized system calls.

Contents

History

UNIX 8th Edition

Tom J. Killian implemented the UNIX 8th Edition (V8) version of */proc*: he presented a paper titled "Processes as Files" at USENIX in June 1984. The design of procfs aimed to replace the *ptrace* system call used for process tracing. Detailed documentation can be found in the proc(4) manual page.

SVR4

Roger Faulkner and Ron Gomes ported V8 */proc* to SVR4, and published a paper called "The Process File System and Process Model in UNIX System V" at USENIX in January 1991. This kind of procfs supported the creation of *ps*, but the files could only be accessed with functions *read()*, *write()*, and *ioctl()*. Between 1995 and 1996, Roger Faulkner created the procfs-2 interface for Solaris-2.6 that offers a structured /proc filesystem with sub-directories.

Plan 9

Plan 9 implemented a process file system, but went further than V8. V8's process file system implemented a single file per process. Plan 9 created a hierarchy of separate files to provide those functions, and made /proc a real part of the file system.

4.4BSD

4.4BSD cloned its implementation of `/proc` from Plan 9. As of February 2011, procfs is gradually becoming phased out in FreeBSD.[1] It was removed from OpenBSD in version 5.7, which was released in May 2015, because it "always suffered from race conditions and is now unused".[2]

Solaris

/proc in Solaris 2.6 was finished in 1996; the developers also cloned Plan 9.

Linux

The Linux implementation of `/proc` also clones that of Plan 9. Under Linux, `/proc` includes a directory for each running process, including kernel processes, in directories named `/proc/PID`, where `PID` is the process number. Each directory contains information about one process, including:

- `/proc/PID/cmdline`, the command that originally started the process.
- `/proc/PID/cwd`, a symlink to the current working directory of the process.
- `/proc/PID/environ` contains the names and values of the environment variables that affect the process.
- `/proc/PID/exe`, a symlink to the original executable file, if it still exists (a process may continue running after its original executable has been deleted or replaced).
- `/proc/PID/fd`, a directory containing a symbolic link for each open file descriptor.
- `/proc/PID/fdinfo`, a directory containing entries which describe the position and flags for each open file descriptor.
- `/proc/PID/maps`, a text file containing information about mapped files and blocks (like heap and stack).
- `/proc/PID/mem`, a binary image representing the process's virtual memory, can only be accessed by a ptrace'ing process.
- `/proc/PID/root`, a symlink to the root path as seen by the process. For most processes this will be a link to / unless the process is running in a chroot jail.
- `/proc/PID/status` contains basic information about a process including its run state and memory usage.
- `/proc/PID/task`, a directory containing hard links to any tasks that have been started by this (i.e.: the parent) process.

(Users may obtain the PID with a utility such as pgrep, pidof or ps:

```
$ ls -l /proc/$(pgrep -n python)/fd        # List all file descriptors of the most recently started `python
samtala 0
lrwx------ 1 baldur baldur 64 2011-03-18 12:31 0 -> /dev/pts/3
lrwx------ 1 baldur baldur 64 2011-03-18 12:31 1 -> /dev/pts/3
lrwx------ 1 baldur baldur 64 2011-03-18 12:31 2 -> /dev/pts/3
$ readlink /proc/$(pgrep -n python)/exe    # List executable used to launch the most recently started `pyth
/usr/bin/python3.1
```

)

`/proc` also includes non-process-related system information, although in the 2.6 kernel much of that information moved to a separate pseudo-file system, sysfs, mounted under `/sys`:

- depending on the mode of power management (if at all), either directory `/proc/acpi` or `/proc/apm`, which predate sysfs and contain various bits of information about the state of power management.
- `/proc/buddyinfo`, information about the buddy algorithm that handles memory fragmentation.[3]
- `/proc/bus`, containing directories representing various buses on the computer, such as input/PCI/USB. This has been largely superseded by sysfs under /sys/bus which is far more informative.
- `/proc/fb`, a list of the available framebuffers

- `/proc/cmdline`, giving the boot options passed to the kernel
- `/proc/cpuinfo`, containing information about the CPU, such as its vendor (and CPU family, model and model names which should allow users to identify the CPU) and its speed (CPU clockspeed), cache size, number of siblings, cores, and CPU flags. `/proc/cpuinfo` includes a value for "bogomips", frequently misconstrued as a measure of CPU speed, like a benchmark, but it does not actually measure any sensible (for end-users) value at all. It occurs as a side-effect of kernel timer calibration and yields highly varying values depending on CPU type, even at equal clock speeds.

On multi-core CPUs, /proc/cpuinfo contains the fields for "siblings" and "cpu cores" which represent the following calculation is applied:[4]

```
"siblings" = (HT per CPU package) * (# of cores per CPU package)
"cpu cores" = (# of cores per CPU package)
```

A CPU package means physical CPU which can have multiple cores (*single core* for one, *dual core* for two, *quad core* for four). This allows a distinction between hyper-threading and dual-core, i.e. the number of hyper-threads per CPU package can be calculated by *siblings / CPU cores*. If both values for a CPU package are the same, then hyper-threading is not supported.[5] For instance, a CPU package with siblings=2 and "cpu cores"=2 is a dual-core CPU but does not support hyper-threading.

- `/proc/crypto`, a list of available cryptographic modules
- `/proc/devices`, a list of character and block devices sorted by device ID but giving the major part of the `/dev` name too
- `/proc/diskstats`, giving some information (including device numbers) for each of the logical disk devices
- `/proc/filesystems`, a list of the file systems supported by the kernel at the time of listing
- `/proc/interrupts`, `/proc/iomem`, `/proc/ioports` and the directory `/proc/irq`, giving some self-explanatory details about the devices (physical or logical) using the various system resources
- `/proc/kmsg`, holding messages output by the kernel[6]
- `/proc/meminfo`, containing a summary of how the kernel is managing its memory
- `/proc/modules`, one of the most important files in `/proc`, containing a list of the kernel modules currently loaded . It gives some indication (not always entirely correct) of dependencies.
- `/proc/mounts`, a symlink to self/mounts which contains a list of the currently mounted devices and their mount points (and which file system is in use and what mount options are in use).
- `/proc/net/`, a directory containing useful information about the network stack, in particular `/proc/net/nf_conntrack`, which lists existing network connections (particularly useful for tracking routing when iptables FORWARD is used to redirect network connections)
- `/proc/partitions`, a list of the device-numbers, their size and `/dev` names which the kernel has identified as existing partitions
- `/proc/scsi`, giving information about any devices connected via a SCSI or RAID controller
- a symbolic link to the current (traversing) process at `/proc/self` (i.e. `/proc/PID/` where PID is that of the current process).
- `/proc/slabinfo`, listing statistics on the caches for frequently-used objects in the Linux kernel
- `/proc/swaps`, a list of the active swap partitions, their various sizes and priorities
- Access to dynamically-configurable kernel options under `/proc/sys`. Under `/proc/sys` appear directories representing the areas of kernel, containing readable and writable virtual files.
  For example, a commonly referenced virtual file is `/proc/sys/net/ipv4/ip_forward`, because it is necessary for routing firewalls or tunnels. The file contains either a '1' or a '0': if it is 1, the IPv4 stack forwards packets not meant for the local host, if it is 0 then it does not.
- `/proc/sysvipc`, containing memory-sharing and inter-process communication (IPC) information.
- `/proc/tty`, containing information about the current terminals; `/proc/tty/driver` looks to be a list of the different types of tty available - each of which is a list of those of each type
- `/proc/uptime`, the length of time the kernel has been running since boot and spent in idle mode (both in seconds)
- `/proc/version`, containing the Linux kernel version, distribution number, gcc version number (used to build the kernel) and any other pertinent information relating to the version of the kernel currently running
- other files depending on various hardware, module configurations, and changes to the kernel.

The basic utilities that use /proc under Linux come in the procps (`/proc` processes) package, and only function in conjunction with a mounted `/proc`.

Cobalt

Cobalt Networks added additional functions to */proc* for their systems:

- */proc/cobalt*, a directory containing Cobalt-specific data such as the system type and serial number
- */proc/lcd*, a file containing the contents of the front-panel LCD screen. Text written to this file would be displayed on the screen.

References

1. "Why is procfs deprecated in favor of procstat?" (http://lists.freebsd.org/pipermail/freebsd-fs/2011-February/010760.html). *freebsd.org*.
2. "Detailed changes between OpenBSD 5.6 and 5.7" (http://www.openbsd.org/plus57.html) *openbsd.org*.
3. "3.2.2. /proc/buddyinfo" (http://www.centos.org/docs/5/html/5.2/Deployment_Guide/s2-proc-buddyinfo.html) *centos.org*.
4. Baron, Jason. "HT vs. dual-core" (http://www.redhat.com/archives/nahant-list/2006-January/msg00176.html)
5. "Understanding Linux /proc/cpuinfo" (https://web.archive.org/web/20120403230159/http://www.richweb.com/cpu_info). *richweb.com*. Archived from the original (http://www.richweb.com/cpu_info) on 2012-04-03. Retrieved 2015-04-21.
6. Nguyen, Binh (2004-07-30). "Linux Filesystem Hierarchy" (https://books.google.com/books?id=wLJWBQAAQBAJ) Binh Nguyen. p. 63. Retrieved 2016-07-18. "/proc/kmsg[:] Messages output by the kernel. These are also routed to syslog."

- Unix 8th Edition proc(2) manual page - Description of the original procfs.
- Plan 9 procfs manual page - Plan 9 greatly expanded the procfs concept, providing a much expanded interface to control and manipulate processes.
- Linux Manual Pages Proc(5) Linux manual documentation for procfs
- Documentation/filesystems/proc.txt Linux kernel documentation for procfs

External links

- A brief history of /proc Eric Schrock's Weblog
- Access the Linux kernel using the Procfs An IBM developerWorks article by M. Tim Jones
- Linux-Filesystem-Hierarchy Linux Documentation Project
- Discover the possibilities of the /proc directory by Federico Kereki