Touring the processes on a Linux workstation

The Process List

Each process on your Linux system has a job to do. This issue of Admin Workshop examines some of the processes running on a typical Linux workstation. BY MARC ANDRÉ SELIG

The last issue of this column [1] looked at methods that allow admins to check the processes currently running on a system. This month I'll describe some of the processes you'll find when you generate a process list for a Linux system using the *ps ax* command. The results of this command may vary depending on your configuration, but you'll have no trouble identifying some of the important processes I'll describe. Understanding these basic Linux processes will give you a head start in troubleshooting process problems. A strong background in the important daemons will also make it easier to identify unauthorized and potentially dangerous software that may appear unexpectedly on the process list.

Figure 1 shows an example of the processes running on a typical Linux system. This example is the partial output of the *ps ax* command for a fairly ancient SuSE 8.2 installation just after booting. Most of the network services on the system in our example have been disabled – a full-fledged server would have many more daemons running.

The processes shown in Figure 1 can be assigned to two completely different groups: kernel threads and user processes. Kernel threads run in kernelspace – the protected area occupied by Linux itself. Kernel threads are typically created when modules are initialized. User processes, on the other hand, are created either by launching a program or by executing a *fork()*.

Init, the Great Grandma of All Processes

The *init* process (which always has process ID 1) is the parent of all other processes on the system. As described at [2], init starts (and stops) other programs as defined in the */etc/inittab* configuration file.

Kernel threads [4] handle critical kernel tasks. You can recognize a kernel thread by the square brackets surrounding the process name. Table 1 gives examples of the kernel threads for the Linux 2.4 kernel.

Another set of kernel threads handles logical devices (*mdrecoveryd* or *raid*d*), or special hardware(*ahd** and *scsi** or *khubd*).

Kernel threads are typically generated when a specific kernel segment initializes, and thus, a kernel thread usually begins at an early stage of the boot process. The exceptions to this rule are modules, which can be loaded at runtime. For example, your system might load a module to handle a USB or cardbus device inserted into a

Table 1: Kernel Threads

Thread	Explanation
keventd	Helps to maintain kernel threads themselves.
kapmd, kacpid	Handle interaction with the computers power management features (Advanced Power Management or Advanced Configuration and Power Interface).
ksoftirqd	Provides high-performance interrupt handling (using soft IRQs). There may be multiple instances of this process with different names for multiprocessor systems.
pdflush, bdflush, kupdated, kjournald	Ensures that any data written to disk really does end up on the disk.
kswapd, kscand	Swaps unneeded data from main memory out to disk.

machine. User processes are generated as required by the program cascade that *init* calls. The complete startup process can take a few minutes, which means that a user process typically has a higher process ID than a kernel thread.

On the Network

One of the first things a computer on a network has to do is set up network access. The configuration files needed for this task may not be stored on the device but instead may be centrally managed, using a mechanism like DHCP (the Dynamic Host Configuration Protocol). In this case, the DHCP client daemon, *dhcpcd*, is one of the first user processes to appear. On our lab system, *dhcpcd* is PID 615. Note that the second "c" in front of the last "d" distinguishes the client daemon from the server daemon *dhcpd*.

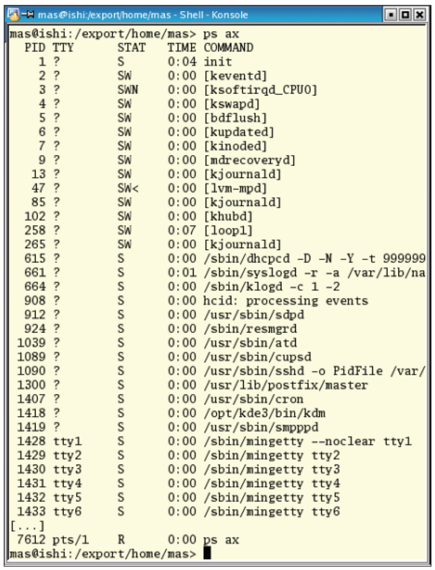
The DHCP client retrieves essential data, such as the computer's IP address, netmask, and gateway, and adds this information to the kernel-based routing table. The client also modifies this information when required to do so by the DHCP server. Some systems may still have *pump* instead of *dhcpcd*; *pump* is an older version of the DHCP client with an almost identical role.

Stand-alone devices that have a separate router for DSL or cable-based Internet access also need *dhcpcd*. If you use static addresses on your LAN, there is no need for the DHCP process.

Today's modem or ISDN connections, which many users may use to save money, mostly use the Point-to-Point protocol with a daemon called *pppd*. Direct DSL connections, where the Linux computer is connected to the DSL modem via an Ethernet card, either use *dhcpcd* or *pppd* in connection with *pppoe* (PPP over Ethernet). dhclient is also a popular DHCP alternative.

Logging

The next pair of daemons shown in Figure 1 are the admin's best friends: *syslogd* and *klogd* [3] provide central management of critical logging messages. *syslogd* accepts syslog messages from user processes, retrieves the messages from */etc/syslogd.conf* (or */etc/syslog.conf*), and, depending on the configuration, stores the messages in files or



```
mas@ishi:/export/home/mas> ps ax
  PID TTY      STAT   TIME COMMAND
    1 ?        S      0:04 init
    2 ?        SW     0:00 [keventd]
    3 ?        SWN    0:00 [ksoftirqd_CPU0]
    4 ?        SW     0:00 [kswapd]
    5 ?        SW     0:00 [bdflush]
    6 ?        SW     0:00 [kupdated]
    7 ?        SW     0:00 [kinoded]
    9 ?        SW     0:00 [mdrecoveryd]
   13 ?        SW     0:00 [kjournald]
   47 ?        SW<    0:00 [lvm-mpd]
   85 ?        SW     0:00 [kjournald]
  102 ?        SW     0:00 [khubd]
  258 ?        SW     0:07 [loop1]
  265 ?        SW     0:00 [kjournald]
  615 ?        S      0:00 /sbin/dhcpcd -D -N -Y -t 999999
  661 ?        S      0:01 /sbin/syslogd -r -a /var/lib/na
  664 ?        S      0:00 /sbin/klogd -c 1 -2
  908 ?        S      0:00 hcid: processing events
  912 ?        S      0:00 /usr/sbin/sdpd
  924 ?        S      0:00 /sbin/resmgrd
 1039 ?        S      0:00 /usr/sbin/atd
 1089 ?        S      0:00 /usr/sbin/cupsd
 1090 ?        S      0:00 /usr/sbin/sshd -o PidFile /var/
 1300 ?        S      0:00 /usr/lib/postfix/master
 1407 ?        S      0:00 /usr/sbin/cron
 1418 ?        S      0:00 /opt/kde3/bin/kdm
 1419 ?        S      0:00 /usr/sbin/smpppd
 1428 tty1     S      0:00 /sbin/mingetty --noclear tty1
 1429 tty2     S      0:00 /sbin/mingetty tty2
 1430 tty3     S      0:00 /sbin/mingetty tty3
 1431 tty4     S      0:00 /sbin/mingetty tty4
 1432 tty5     S      0:00 /sbin/mingetty tty5
 1433 tty6     S      0:00 /sbin/mingetty tty6
[...]
 7612 pts/1    R      0:00 ps ax
mas@ishi:/export/home/mas>
```

Figure 1: The *ps ax* command shows a variety of processes running on a SuSE 8.2 box just after booting. (The list has been shortened.)

displays them to the screen or console window. The kernel logging daemon, *klogd*, converts kernel messages into logging entries that syslog can understand.

Local Access

Another group of daemons handles local devices. Unix is a multi-user operating system by design. Modern Linux distributions have mechanisms that allow you to restrict access privileges for local devices to users logged on to the local console. Remote users who log in via the network are not permitted to use these devices. This approach protects the privacy of the console users and ensures that the console user will be able to access the attached devices when he or she needs to do so. Examples of local devices are sound cards, CD recorders, scanners, some USB devices, the Bluetooth PAN (Personal Area Networking), and sometimes the modem.

Listing 1: Excerpt from *resmgr.conf*

```
# Desktop-User:
class desktop
# Audio:
add /dev/audio      desktop
add /dev/mixer      desktop
add /dev/dsp        desktop
add /dev/sequencer  desktop
add /dev/video      desktop
# Modem:
add /dev/modem      desktop
<ééI>[...]<ééI>
# Allow local users access:
allow desktop
  tty=/dev/tty[1-9]* ||
  tty=tty[1-9]* || tty=:0
```

The Resource Manager Daemon *resmgrd* (PID 924 in Figure 1) updates access privileges for device files to reflect the current user. The configuration is stored in */etc/resmgr.conf* (Listing 1). The */etc/resmgr.conf* file is where admins define device classes and specify under what conditions a user should be permitted access.

The Host Controller Interface Daemon *hcid* and *sdpd* (Service Discovery Protocol) support interaction with Bluetooth devices. *smpppd* (SuSE Meta PPP Daemon) is a SuSE invention that allows console users to access modem, ISDN, and DSL networks without requiring the native PPP daemon (*pppd*). This makes it easier for non-privileged users to work with the GUI.

Scheduling

cron belongs to another class of daemons that support automatic, scheduled execution of re-occurring tasks. *cron* uses the global tables stored below */etc/cron** and user-specific tables in */var/spool/cron*. The batch daemon, *atd*, is a close relative of *cron* that facilitates the execution of individual jobs at an arbitrary time.

Basic Network Services

Most network daemons are beyond the scope of this article; and in fact, most simple workstations will not run many of them. There are, however, some critical network services that we should examine.

The spooler daemon *cupsd* (Common Unix Printing System), or the slightly ancient *lpd* (Line Printer Daemon), are responsible for interacting with the printers attached to the local network. Both accept printing jobs and convert them to a printable format, adding a banner if required, before passing the print jobs to the printer.

The SSH daemon, *sshd*, is responsible for remote access and allows secure remote logins. The Secure Shell encrypts data transmissions and supports multiple cryptographic authentication methods.

The Unix philosophy states that no system is complete without a working email server. But this does not necessarily mean that every Unix system needs to be accessible to external SMTP (Simple Mail Transfer Protocol) communications. In fact, the opposite is true: you should disable this service, unless you have good reason to enable it. However, most people will need a program that accepts email messages and forwards them to the local user, LAN, or Internet-based maildrop. The daemons that handle this job are either *sendmail*, *master* (Postfix), or *qmail-**, depending on your environment and preferences.

The Console

The daemons described thus far ensure that the kernel and the Internet connection will run smoothly. The only thing missing is user interaction. The Linux console supports both GUI and character-based environments. Virtual consoles, which can be toggled by pressing [Ctrl] + [Alt] + [F1] through [F10], allow both environments to coexist peacefully.

Users who prefer a character-based approach use *getty* (get TTY), which shows them the */etc/issue* file and a login prompt. When a user types an entry, *getty* hands control over to */bin/login*, which launches a shell, assuming that the user supplied the correct password.

Traditional gettys are complex programs designed to work with any modem (TTY is short for teletype). As the Linux virtual console is easy on resources, most people use the smaller footprint *agetty* (Alternative Linux Getty) or *mingetty* (Minimal Getty).

The GUI-based login needs more in the line of resources, using a display manager such as the traditional X Display Manager *xdm*, or an equivalent program, such as *gdm* for Gnome, or *kdm* for KDE

.

Keeping On Top

Understanding the critical services running on a machine is an important prerequisite to troubleshooting. Be inquisitive: investigate the process list. ■

INFO

[1] Marc André Selig, "Locating and disabling network services": Linux Magazine #47, p. 58

[2] Marc André Selig, "Starting Lineup – The Sys V Init Process": Linux Magazine #44, p. 61

[3] Marc André Selig, "Computer Logbook – Syslog": Linux Magazine #40, p. 64

[4] Tigran Aivazian, "Linux Kernel 2.4 Internals": *http://www.faqs.org/docs/kernel_2_4/lki.html*

[5] Roberto Arcomano, "Linux Kernel Analysis Howto": *http://www.linux.org/docs/ldp/howto/KernelAnalysis-HOWTO.html*

THE AUTHOR

Marc André Selig spends half of his time working as a scientific assistant at the University of Trier and as an ongoing medical doctor in the Schramberg hospital. If he happens to find time for it, his currenty preoccupation is programing web based databases on various Unix platforms.