Processes and Daemons

Fundamentally, kernels provide a few logical constructs that mediate access to either real or virtual resources. The two most important in Unix are processes and filesystems.

You can view the characteristics of processes on a Unix machine with a variety of programs, including ps, top, lsof, and even ls.
What Unix/Linux system administrators see – ps

```
[root@localhost root]# cat /etc/redhat-release
Fedora release 8 (Werewolf)
[root@localhost root]# ps -elf  # This is SYSV; Berkeley = 'ps axlww'
F  S  UID    PID  PPID  C PRI NI TTY  TIME   CMD
4  S   root   1   0   0   75  0 ?   00:00:08   init
4  S   root  1573 1384  0   75  0 tty  00:00:00   -bash
5  S  root  7492   1   0   75  0 ?   00:01:08   sendmail: accepting
1  S  smmsp  7497   1   0   75  0 ?   00:00:00   sendmail: Queue run
5  S  apache 25079 1321  0   75  0 ?   00:00:00   /usr/sbin/httpd
5  S  apache 25080 1321  0   75  0 ?   00:00:00   /usr/sbin/httpd
5  S  apache 25085 1321  0   75  0 ?   00:00:00   /usr/sbin/httpd
5  S  apache 25086 1321  0   75  0 ?   00:00:00   /usr/sbin/httpd
```
What system administrators see – ps

5 S root 13137 7492 0 76 0 ? 00:00:00 sendmail: server [10.1.
5 S root 16572 7492 0 75 0 ? 00:00:00 sendmail: k0CBPF4I01657
5 S root 18574 7492 0 75 0 ? 00:00:00 sendmail: k0CBcKUk01857
5 S root 20824 7492 0 75 0 ? 00:00:00 sendmail: k0CBs9CZ02082
5 S root 22950 7523 6 75 0 ? 00:04:14 /usr/bin/perl
5 S root 23050 7523 6 78 0 ? 00:03:58 /usr/bin/perl
5 S root 32112 1151 0 75 0 ? 00:00:00 sshd: root@pts/0
4 S root 32142 32112 0 75 0 pts/0 00:00:00 -bash
5 S root 32286 1 0 83 0 ? 00:00:00 sendmail: ./k0CD8sHV032
5 S root 32317 7492 0 75 0 ? 00:00:00 sendmail: k0CD96Jh03231
What Unix/Linux system administrators see – top

[root@localhost root]# top -b -n1  # run in batch mode for one iteration
08:17:41 up 1 day, 18:12, 2 users, load average: 9.69, 9.14, 8.89
115 processes: 114 sleeping, 1 running, 0 zombie, 0 stopped
CPU states: cpu user nice system irq softirq iowait idle
    total 0.0% 0.0% 0.9% 0.0% 0.9% 0.0% 98.0%
Mem: 510344k av, 392504k used, 117840k free, 0k shrd, 17208k buff
        240368k actv, 55488k in_d, 4760k in_c
Swap: 522104k av, 90392k used, 431712k free
                 72852k cached

           PID USER  PRI  NI  SIZE  RSS  SHARE  STAT  %CPU  %MEM   TIME  CPU  COMMAND
         1090 root   20   0  1088  1088   832  R   0.9  0.2  0:00  0.0  top
          1 root    15   0   492   456   432  S   0.0  0.0  0:08  0.0  init
          3 root    15   0   0   0   0  SW   0.0  0.0  0:00  0.0  keventd

CNT 4603
What Unix/Linux system administrators see - `lsof`

```
[root@localhost root]# lsof  # heavily redacted to fit on page

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>PID</th>
<th>USER</th>
<th>NODE</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>sendmail</td>
<td>20824</td>
<td>root</td>
<td>159526</td>
<td>/lib/libcrypt-2.3.2.so</td>
</tr>
<tr>
<td>sendmail</td>
<td>20824</td>
<td>root</td>
<td>159568</td>
<td>/lib/libcrypto.so.0.9.7a</td>
</tr>
<tr>
<td>sendmail</td>
<td>20824</td>
<td>root</td>
<td>319023</td>
<td>/usr/lib/libldap.so.2.0.17</td>
</tr>
<tr>
<td>sendmail</td>
<td>20824</td>
<td>root</td>
<td>32286</td>
<td>/usr/lib/sasl/libcrammd5.so.1.0.19</td>
</tr>
<tr>
<td>sendmail</td>
<td>20824</td>
<td>root</td>
<td>32104</td>
<td>/usr/kerberos/lib/libk5crypto.so.3.0</td>
</tr>
<tr>
<td>sendmail</td>
<td>20824</td>
<td>root</td>
<td>32095</td>
<td>/lib/tls/libdb-4.2.so</td>
</tr>
</tbody>
</table>
```
### What system administrators see - `ls/of`

<table>
<thead>
<tr>
<th>Program</th>
<th>PID</th>
<th>User</th>
<th>FD</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>sendmail</td>
<td>20824</td>
<td>root</td>
<td>318943</td>
<td>/usr/lib/libz.so.1.1.4</td>
</tr>
<tr>
<td>sendmail</td>
<td>20824</td>
<td>root</td>
<td>65611</td>
<td>/dev/null</td>
</tr>
<tr>
<td>sendmail</td>
<td>20824</td>
<td>root</td>
<td>TCP anothermachine.com:smtp-&gt;10.1.1.20:</td>
<td></td>
</tr>
<tr>
<td>sendmail</td>
<td>20824</td>
<td>root</td>
<td>65611</td>
<td>/dev/null</td>
</tr>
<tr>
<td>sendmail</td>
<td>20824</td>
<td>root</td>
<td>16220</td>
<td>socket</td>
</tr>
<tr>
<td>sendmail</td>
<td>20824</td>
<td>root</td>
<td>TCP anothermachine.com:smtp-&gt;10.1.1.20:</td>
<td></td>
</tr>
<tr>
<td>sendmail</td>
<td>20824</td>
<td>root</td>
<td>TCP localhost.localdomain:48512-&gt;localhost:</td>
<td></td>
</tr>
<tr>
<td>sendmail</td>
<td>20824</td>
<td>root</td>
<td>TCP anothermachine.com:smtp-&gt;10.1.1.20:</td>
<td></td>
</tr>
</tbody>
</table>
Processes and Daemons: `fork(2)` and `clone(2)`

Fundamentally, kernels provide some logical constructs that mediate access to either real or virtual resources. The two most important in Unix are processes and filesystems.

A new process is created by `fork(2)`; or, alternatively, in Linux with `clone(2)` since processes and threads are both just `task_struct` in Linux.
Processes and Daemons: `fork(2)` and `clone(2)`

- With `clone(2)`, memory, file descriptors and signal handlers are still shared between parent and child.
- With `fork(2)`, these are copied, not shared.
Starting a Unix/Linux process

exec*() instantiates a new executable:

- Usually, when doing an exec*() the named file is loaded into the current process’s memory space.
Starting a Unix/Linux process

Unless the first two characters of the file are `#!` and the following characters name a valid pathname to an executable file, in which that file is instead loaded.

If the executable is dynamically linked, then the dynamic loader maps in the necessary bits (not done if the binary is statically linked.)
Starting a Unix/Linux process

Then code in the initial "text" section is then executed. (There are three main types of sections: "text" sections for executable code, "data" sections (including read-only "rodata" sections), and "bss" sections (Blocks Started by Symbol) which contains "uninitialized" data.
Some Typical Assembly Code

.file   "syslog.c" ; the file name this originated in
.data   ; a data section
.align  4 ; put PC on 4 (or 16) byte alignment
.type   LogFile,@object ; create a reference of type object
.size   LogFile,4 ; and give it 4 bytes in size
Some Typical Assembly Code

LogFile:  ; address for object
  .long  -1  ; initialize to a value of -1
  .align  4  ; align . to 4 (16) byte
  .type   LogStat,@object  ; a new object reference is created
  .size   LogStat,4       ; give it 4 bytes also

LogStat:  ; here’s its address in memory
  .long   0               ; and initialized it to a value zero
  .section .rodata       ; here’s a ‘‘read-only’’ section
Some Typical Assembly Code

.LC0: ; local label for a string  
.string "syslog" ; initialized to "syslog"
[ ... ]

.text ; now we have some executable code
.globl syslog ; and it is a global symbol for
.type syslog,@function ; a function syslog()
Some Typical Assembly Code

syslog:

    pushl %ebp                    ; and away we go...
    movl %esp, %ebp
    subl $8, %esp
Daemon processes

When we refer to a daemon process, we are referring to a process with these characteristics:

- Generally persistent (though it may spawn temporary helper processes like xinetd does)
Daemon processes

- No controlling terminal (and the controlling tty process group (tpgid) is shown as -1 in ps)
- Parent process is generally init (process 1)
- Generally has its own process group id and session id;
Daemon processes

- Generally a daemon provides a service. So why not put such services in the kernel?
- Another level of modularity that is easy to control
- Let’s keep from growing the already largish kernel
Daemon processes

- Ease (and safety) of killing and restarting processes
- Logically, daemons generally share the characteristics one expects of ordinary user processes (except for the lack of controlling terminal.)
BSD-ish: Kernel and user daemons:

swapper

-all UNIX processes have a unique process ID (pid).

An increasing number of daemons execute in kernel mode; (pagedaemon and swapper are two early examples from the BSD world); the rest still execute in user mode.
BSD-ISH: Kernel and user daemons:

swapper

BSD swapper (pid 0) daemon

The BSD swapper is a kernel daemon. swapper moves whole processes between main memory and secondary storage (swapping out and swapping in) as part of the operating system’s virtual memory system.
BSD-ish: Kernel and user daemons:

swapper

SA RELEVANCE: In BSD-land, the swapper is the first process to start after the kernel is loaded. (If the machine crashes immediately after the kernel is loaded then you may not have your swap space configured correctly.)
BSD-ish: Kernel and user daemons: swapper

The swapper is described as a separate kernel process in other non-BSD UNIXes. It appears in the Linux process table as kswapd. It does appear on AIX, HP-UX, IRIX; for example it appears in the Solaris process table as sched (the SysV swapper was sometimes called the scheduler because it 'scheduled' the allocation of memory and thus influences the CPU scheduler).
BSD: Kernel and user daemons: pagedaemon

BSD pagedaemon. In days gone by, the third process created by the kernel was always the pagedaemon and always had pid 2. These days, it’s just another in the rapidly proliferating “kernel processes” in BSD. The pagedaemon as a kernel process originated with BSD systems (demand paging was initially a BSD feature) which was adopted by AT&T. The pageout process
(still pid 2) in Solaris provides the same function with a different name.
BSD: Kernel and user daemons:
pagedaemon

SA RELEVANCE: This is all automatic – not much for the SA to do, except monitor system behavior to make sure the system isn’t thrashing (you would expect to see this process taking up a lot of cpu time if there were thrashing.)
Kernel and user daemons: init

init (pid 1) daemon: The first “user” process started by the kernel; its userid is 0. All other “normal” processes are children of init. Depending on the boot parameters init either:

- Spawns a single-user shell at the console
Kernel and user daemons: init

or begins the multi-user start-up scripts (which are, unfortunately, not standardized across UNIXes; see section 2.4 (starts on page 24) in USAH). There is a lot of flux in this area; we are seeing, for instance, in Fedora 10 replacement of the old SysV init with upstart; hopefully we can get better dependency resolution than we have had previously and faster boot times. (Take a look at /etc/event.d on Fedora 10 for instance.)
Kernel and user daemons: update (aka bdflush/kupdate and fsflush)

update daemons: An update daemon executes the sync() system call every 30 seconds or so. The sync() system call flushes the system buffer cache; it is needed because UNIX uses delayed write when buffering file I/O to and from disk.
Kernel and user daemons: \texttt{update} (aka \texttt{bdflush/kupdate} and \texttt{fsflush})

\begin{itemize}
  \item SA RELEVANCE: It’s best not to just turn off a UNIX machine without flushing the buffer cache. It is better to halt the system using \texttt{/etc/shutdown}, \texttt{/etc/halt}, or \texttt{poweroff}; these commands attempt to put the system in a quiescent state (including calling \texttt{sync()}).
\end{itemize}
Kernel and user daemons: update (aka bdflush/kupdate and fsflush)

I like to do something like `sync ; sync ; poweroff` or `sync ; sync ; reboot` just to make sure a few manual synchronizations are made. When I am removing a USB drive, I like to do something like `sync ; umount /media/disk ; sync`.

The update daemon goes by many names (see CNT 4603).
bdflush, bdflush(2), and kupdate in Linux and fsflush in Solaris).
Even though well-written daemons consume little CPU time they do take up virtual memory and process table entries.

Years ago, as people created new services, the idea of a super-daemon inetd was created to manage the class of network daemons.
Many network servers were mediated by the inetd daemon at connect time, though some, such as sendmail, postfix, qmail, and sshd were not typically under inetd.
Kernel and user daemons: `inetd` and `xinetd`

The original `inetd` listened for requests for connections on behalf of the various network services and then started the appropriate daemon, handing off the network connection pointers to the daemon.
Kernel and user daemons: `inetd` and `xinetd`

Some examples are `pserver`, `rlogin`, `telnet`, `ftp`, `talk`, and `finger`.

The configuration file that told `inetd` which servers to manage was `/etc/inetd.conf`. 
Kernel and user daemons: inetd and xinetd

The /etc/services file: This file maps TCP and UDP protocol server names to port numbers.

The /etc/inetd.conf file This file has the following format (page 824 in USAH and “man inetd.conf”):
Kernel and user daemons: \texttt{inetd} and \texttt{xinetd}

\begin{itemize}
\item 1st column is the name of the service (must match an entry in \texttt{/etc/services} (or be in the services NIS map))
\item 2nd column designates the type of socket to be used with the service (stream or datagram)
\end{itemize}
Kernel and user daemons: inetd and xinetd

- 3rd column designates the communication protocol (tcp is paired with stream sockets and udp is paired with datagram sockets)
- 4th column applies only to datagram sockets - if the daemon can process multiple requests then put 'wait' here so that inetd doesn't keep forking new daemons
Kernel and user daemons: `inetd` and `xinetd`

- 5th column specifies the username that the daemon should run under (for example - let’s have `fingerd` run as ‘nobody’)
- remaining columns give the pathname and arguments of the daemons (here’s where TCP wrappers are typically installed).
The successor to/inetd was/xinetd, which combined standard/inetd functions with other useful features, such as logging and access control.
Kernel and user daemons: inetd and xinetd

The configuration file structure for xinetd is also different: /etc/xinetd.conf is used to modify general behavior of the daemon and the directory /etc/xinetd.d contains separate files per service. Your CentOS machines use xinetd instead of inetd.
Kernel and user daemons: *inetd* and *xinetd*

⚠️ **SA RELEVANCE:** When installing new software packages you may have to modify `/etc/inetd.conf`, `/etc/xinetd.d/` files, and/or `/etc/services`. A hangup signal (`kill -HUP SOMEPID`) will get the `inetd/xinetd` to re-read its config file. Or you might be able to use a startup script, such as `"/etc/init.d/inetd restart"` or `"service inetd`
restart".
Kernel and user daemons: portmap and rpcbind

portmap/rpcbind: portmap (rpcbind on OpenSolaris and BSD) maps Sun Remote Procedure Call (RPC) services to ports (/etc/rpc). Typically, /etc/rpc looks something like:
Kernel and user daemons: portmap

[root@vm5 etc]# more /etc/rpc
#ident '©(#)rpc 1.11 95/07/14 SMI' /* SVr4.0
#
#
rpc
#
portmapper 100000 portmap sunrpc rpcbind
rstatd 100001 rstat rup perfmeter rstat_svc
rusersd 100002 rusers
nfs 100003 nfsprog
ypserv 100004 ypprog
mountd 100005 mount showmount
ypbind 100007
walld 100008 rwall shutdown
ypasswdd 100009 yppasswd
Kernel and user daemons:
portmap/rpcbind

Sun RPC is a backbone protocol used by other services, such as NFS and NIS. RPC servers register with this daemon and RPC clients get the port number for a service from the daemon. You can find operational information using `rpcinfo`. For example, `rpcinfo -p` will list the RPC services on the local machine.
Kernel and user daemons: portmap/rpcbind

-SA RELEVANCE: Some daemons may fail if portmap isn’t running. Most UNIXes these days automatically start up portmap after installation, so it’s usually not a problem. Also, there are subtle points that have oddly crept in from the old tcpwrappers package that can affect the portmapper. See for example /etc/hosts.deny.
Kernel and user daemons: syslogd

syslogd: syslogd is a daemon whose function is to handle logging requests from

- the kernel
- other user processes, primarily daemon processes
- processes on other machines, since syslogd can listen for logging requests across a network
A process can make a logging request to the syslogd by using the function `syslog(3)`. syslogd determines what to do with logging requests according to the configuration file `/etc/syslog.conf`

`/etc/syslog.conf` generally looks something like:
Kernel and user daemons: syslogd

*.info;mail.none;news.none;authpriv.none;cron.none  /var/log/messages
authpriv.*  /var/log/secure
mail.*  /var/log/maillog
cron.*  /var/log/cron
*.emerg  *
uucp,news.crit  /var/log/spooler
local7.*  /var/log/boot.log
Kernel and user daemons: syslogd

-SA RELEVANCE: For a single UNIX machine, the default /etc/syslog.conf will suffice. Also, you should note that Linux distributions have been moving to rsyslogd, which provides expanded capabilities (such as logging directly to a database) and still tries to preserve the capabilities of the original syslogd.

-You should read the file and figure out where the most common error messages end up (/var/adm/messages
or `/var/log/messages` are typical default locations).
Kernel and user daemons: syslogd

If you are going to manage a number of UNIX machines, consider learning how to modify /etc/syslog.conf on the machines so all the syslog messages are routed to a single “LOGHOST”.