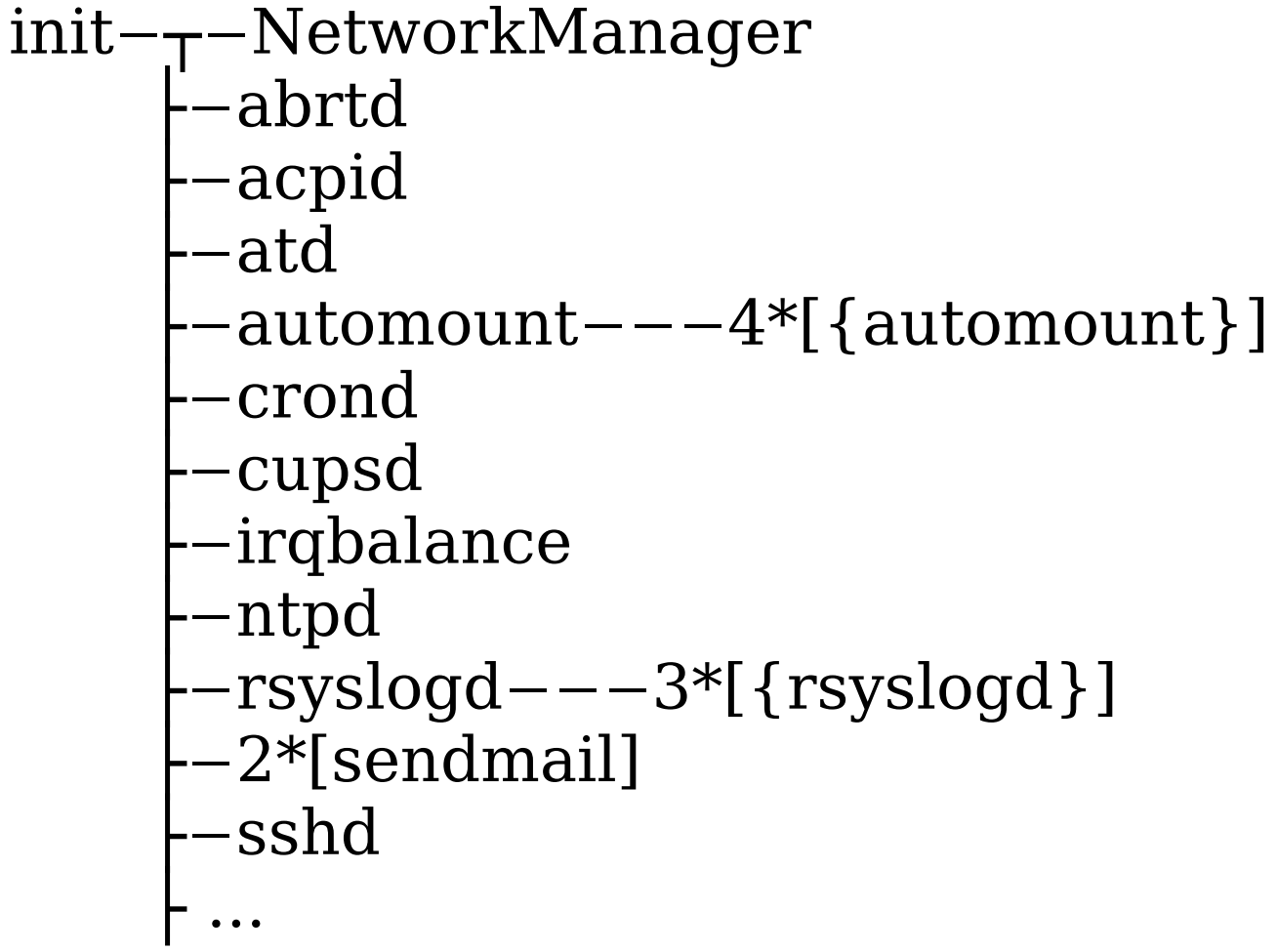
CST8177

Services, Daemons, and Logs

Services

A service is just that – a facility that provides something useful in a Linux/Unix system. I suppose even Windows has services, but that's not within our scope today.

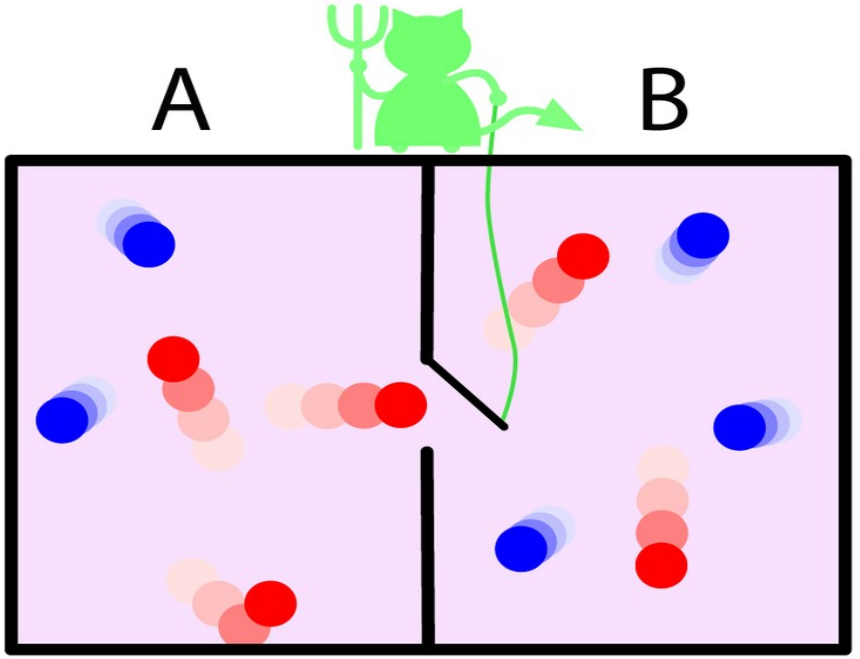
A service is provided by a daemon.



Daemon?

Yes, pronounced "demon" but spelled daemon. Blame James Clerk Maxwell (1831 - 1879)), he of Maxwell's equations for electromagnetic theory.

He spoke of a "thought experiment" in entropy, using a sealed box divided by a partition with a small hole in it.



Maxwell's Daemon

Maxwell's daemon sorts air molecules by speed, fast ones (red) to the right, and slower ones to the left (blue). Since their speed and their temperature are related, this has the effect of cooling the left-hand side – a refrigerator.

Maxwell's daemon (not an unusual term, with fewer religious connotations in the 19th century than it seems to have today) was a mindless automaton that did one thing and did it well: detected air molecules and sorted them by their speed/temperature.

We might call this refrigerating service the "cool" service, and give the daemon the name "coold".

Hence our daemons are automatic processes that provide a service. Max's was preprogrammed, as are ours, but we may want to check on what our daemons are doing and exercise some level of control.

Init and its processes

We've already seen how init manages various services' programs through the runlevels by the scripts in **/etc/rc.d/init.d** and the soft links in **/etc/rc.d/rc?.d** (where **?** represents a runlevel number from **0** to **6**).

We've also seen the chkconfig command and used it to see which service is started at which runlevels:

```
System Prompt$ chkconfig --list rsyslog
rsyslog 0:off  1:off  2:on  3:on  4:on  5:on 6:off
```

Which is fine if we remember the service name. Of course, grep can come to the rescue:

```
System Prompt$ chkconfig --list | grep log
rsyslog   0:off 1:off 2:on  3:on  4:on  5:on  6:off
syslog-ng 0:off 1:off 2:off 3:off 4:off 5:off 6:off
```

Whatever **syslog-ng** (next generation?) may be, it's always off so we can ignore it.

chkconfig vs service

The **chkconfig** command allows us to manage the runlevels in which the services are started. It does not do anything to any running process, the daemons that actually provide the services.

To manage the service daemons themselves (as **root**, naturally), we use the rather oddly-named **service** command. Oddly because while it does control the services that are offered, it does so by controlling the daemon.

```
service daemon command optional-options
```

It's done by running the script in **/etc/rc.d/init.d** with the matching name. For example, say you wanted to stop the **crond** daemon to end the **cron** service. The command is

```
service crond stop
```

What actually runs is this:

```
/etc/rc.d/init.d/crond stop
```

Inside the script

Well, isn't that neat! The command used in **service** is passed directly into the script for **crond**. Here's what it looks like inside (heavily edited):

```
case "$1" in
    start)
        ...
    stop)
        ...
    restart)
        ...
    reload)
        ...
    status)
        ...
    *)
        echo $"Usage: ...
        exit 2
esac
```

Notifying Daemons

Now we've tied together the **init** scripts, the runlevels and **chkconfig** to manage them, plus the **service** command to run the scripts under our control to manage the daemons.

As we've seen with **crond**, each daemon has (or is likely to have) a configuration file of some sort to tell it what to do. For **crond**, it's a list of times and associated actions.

It's easy to notify **crond** that there's a change - it checks every minute anyway so there's nothing to do. With most other daemons, we have to tell it to re-read its config file.

One way is to restart the daemon:

```
service rsyslogd restart
```

Since **restart** is one of the commands these scripts must support, that's fine. We could even do a **stop** and a **start**.

Note, though, that these mean that an in-use service has to either drop all users or at least block new users from the service while it does its restart.

Signalling a daemon

It turns out that there's another way.

Recall that a daemon has no terminal (this comes up again in a few minutes). Therefore, its terminal can't actually do anything including provide **`stdin`**, **`stdout`**, or **`stderr`**.

It also can't lose its connection. In the olden days when dinosaurs walked the earth, a typical Unix user connected to the system using a dumb terminal over wires or a phone line. The call can drop, the user can pull the cable, the terminal can lose power. As a result, the **`SIGHUP`** signal was devised to notify a process that its terminal had gone away (HUP is short for Hang-UP).

For daemons, with no terminal, **`SIGHUP`** was re-purposed to cause that daemon to re-read its config file:

```
kill -SIGHUP rsyslogd
```

No users have to be deprived of the service, merely delayed a bit (perhaps). Much better.

Messages from a Daemon

No **`stdout`** or **`stderr`**, eh? So how does a daemon notify us, the admins, of its on-going status or any problems?

In those same olden days, each daemon logged its own output to some sort of log file on disk. Each one (naturally) did it differently, and each one handled the growing size of a log file in its own unique manner.

Some clever people came up with a system-wide logging facility, a way to get to it, and a way to manage log file growth. Today every daemon and even the kernel use the system log. Look in **`/var/log`** to see some of the files.

We'll look at each of these in turn.

Logging: syslog and rsyslog

Originally the service was called **syslog**, and you will hear the service called that even today amongst Old Geeks. But it was revised a few years ago and is now called **rsyslog** with the daemon named **rsyslogd**. The revised service has also subsumed the **sysklog** kernel logging facility.

It has the config file (see rsyslog.conf(5) for full details) **/etc/rsyslog.conf** that contains:

- Global directives
- Templates
- Output channels
- Rules

It should also contain lots of comments (which start with a hash character **#**) and blank lines for readability. Both are ignored, so you can use them freely.

We will only consider the Rules section in this course.

Logging Rules

They are deceptively simple. Each rule takes a single line, with a selector on the left and an action on the right, separated by white space (blanks and tabs).

The selector is in two parts, the facility and the priority, joined by a dot.

```
selector                                        action
facility.priority                               action
```

There are a whole host of possible actions, from the typical one of "put the message into this log file" to "tell everyone logged-in at once!!!" and lots in-between including email.

The most typical action would be, for example, for **cron** which has **/var/log/cron**. It's a requirement to give the absolute path.

To tell everyone logged in, the action is just *, an asterisk. It uses the wall(1) command facility.

Facilities

There is a fixed list of facilities, pre-defined for a variety of purposes:

auth	authorization
authpriv	security
cron	**crond**
daemon	other unnamed daemons
kern	the system kernel (was **sysklog**)
lpr	printers
mail	the email service
news	the Usenet news service
syslog	**syslog** (**rsyslog**) itself
user	user related
uucp	inter-system communication; obsolete
local?	where **?** Can be from **0** through **7**; local use

Priorities

The priorities are the level of importance of the message, and are fixed. They are listed here in ascending order, so **debug** has the lowest priority and **emerg** the highest. This is important, since messages are sent to a range of priorities.

debug	for system debugging activities
info	informational messages
notice	general notices
warning	normal, but significant, condition
err	actual errors
crit	a critical error
alert	take immediate action
emerg	(once called **panic**) system unusable

Messages are logged according to the rules for the specified priority and all lower priorities (higher in the list). A **crit** error (for example) is also logged to **err**, **warning**, **notice**, **info**, and **debug**.

Selectors

A selector is a facility, a dot, and a priority level.

A simple selector might be **mail.info** to log **mail** issues to **info** and **debug** priorities to whatever action is defined.

Selectors can be combined in certain ways. For example, **news,mail.info** logs both **news** and **mail** issues. ***.emerg** will handle all facilities **emerg** priority messages the same way. And **news.warning;mail.err** will use the same action for each as if it were a separate selector.

,	combine facilities
***.**	all facilities
.*	all priorities
;	combine selectors
.none	no priority
=priority	only this priority
!priority	not this priority, only higher

How to log a message

By using the logger(1) command, of course. It is a shell command interface to the **rsyslog** system log daemon.

logger [-isd] [-f file] [-p pri] [-t tag]
[-u socket] [message ...]

In its simplest form, it requires a priority and some text. It's like a simple echo, except for the priority (really a complete **facility.priority**) and no output appears on the screen.

logger -p mail.info incoming mail is blocked

The rules are examined and the message (which might be quoted) is sent according to the selector and its action.

-i	Log the process id of the process
-s	Log the message to **stderr** as well as **rsyslog**
-t tag	Mark every line in the log with the specified **tag**
-f file	Log the specified file contents
--	last argument; the message can start with a **-**

Log management

Logs grow. Every time a message is sent to a log file, the log file gets bigger. Over time, log files could eat up even the largest disk. What to do?

The first step is log rotation. The **logrotate** script is issued by **crond** from **cron.daily**. It in turn runs the **logrotate** command, which gets its orders from **/etc/logrotate.conf** where the default log rotation rules are (see logrotate(8) for information on both the command and its config file).

The second step is to age the log files, and keep only the most recent messages. Combined with rotation, it means keeping some fixed number of old log files.

Logrotate renames the current log file, generally adding the date of rotation or a sequence number, and then creates a new empty log file of the correct name. If there are now too many old log files, the oldest is deleted.

logrotate.conf

Here's the first part of my file, lightly edited. The key items are **weekly** rotation for **4** old log files using the date extension style **dateext**.

```
# see "man logrotate" for details
# rotate log files weekly
weekly
# keep 4 weeks worth of backlogs
rotate 4
# create new (empty) log files after rotating
create
# use date as a suffix of the rotated file
dateext
# RPM packages drop log rotation info here
include /etc/logrotate.d
```

But what about that reference to **/etc/logrotate.d**? It contains a whole lot of other **logrotate** config files, named for the log user each relates to.

Here's the file for process accounting (comments removed):

```
/var/account/pacct {
    compress
    delaycompress
    notifempty
    daily
    rotate 31
    create 0600 root root
    postrotate
        /usr/sbin/accton /var/account/pacct
    endscript
}
```

It contains a separate set of rules just for this process. It runs **daily** with **31** backup copies. Note how it is careful to **create** new log files with owner **root** and group **root** with **0600** permissions. It takes money seriously!