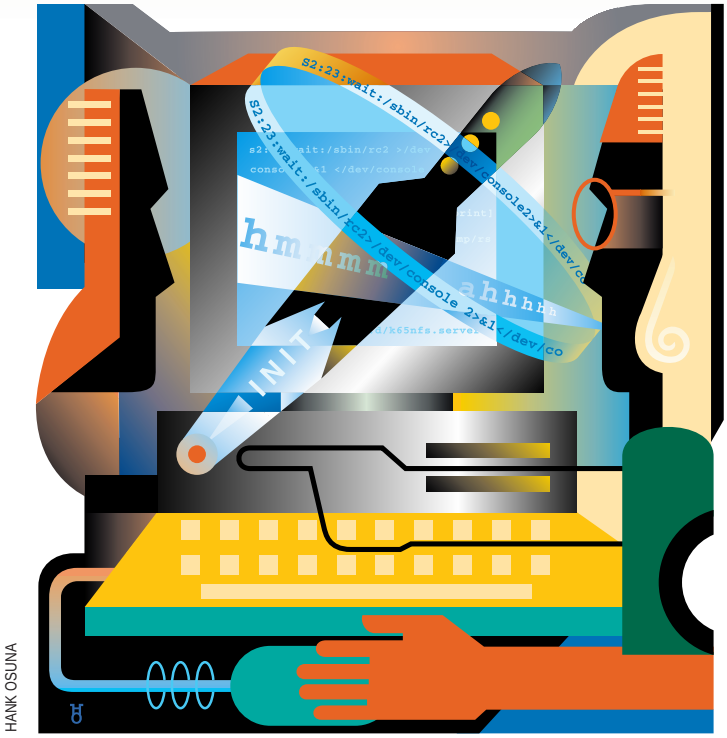
by S. Lee Henry



HANK OSUNA

Inside Run States

One way that I can always impress computer neophytes with my system wizardry is to allow them to stand behind me as I boot my system. While the messages run down the screen, I murmur dispassionately "hmm" and then in a slightly higher tone "ahhh." Immediately, they think, "Wow, look at all that complicated stuff and she understands it." Little do they suspect that the booting process is really quite straightforward. After all, anything a person doesn't understand appears complicated and magical, especially if you throw in a bunch of numbers along with unpronounceable strings of characters.

If I then proceed to explain to my admirers that this computer doesn't only know "on" and "off" but has eight possible run states, they will likely smile weakly and nod their heads at roughly half the normal speed. Little do they know that run states can simply be thought of as particular configurations of running processes.

The init Process

Of course, I know that the `init` process, which runs as long as the system is running, is responsible for ensuring that all the necessary processes are running when they are needed. All the messages filling up my screen are just the result of a very predictable sequence of things that must happen before my system is ready for use. Any wizardry evident during this process belongs to the good design and not any particular prowess on my part.

The `init` process is itself controlled through commands provided in a typical colon-separated initialization file (e.g., `/etc/inittab`). The four fields in `/etc/inittab` are the entry name, the `init` states in which the command–in the fourth field–should be active, one of the ways in which the command can be run (e.g., wait for it to finish or not) and the command itself.

My favorite line in the file is this:

```
s2:23:wait:/sbin/rc2 >/dev/console 2>&1 </dev/console
```

I like this line because (a), run state 2 involves a lot of familiar processes starting and stopping, and (b), the entry "23" causes me to pause while I sort out its meaning. "Ahhh," I say (this time with genuine feeling), "this line is tells me that the processes required in run state 2 [invoked with the script `/sbin/rc2`] must also be running in run state 3!"

The options for the third field are also rather interesting. If the field says "wait," it means that `init` should wait for the command to complete before moving on. This is the most commonly used option. The `sysinit` option means that the command is run before `init` ever tries to access the console.

Notice the absence of run states when `sysinit` is specified. Some commands in the `inittab` file are specified with the option `respawn`. This means just what it suggests–the process is started

when the system enters the particular run state and is respawned if it stops running. This ensures that some processes, like the one that keeps your console in touch with reality, continue running despite whatever else happens on the system.

Less often used options include:

`Off`	Kill the named command.
`Once`	Execute the command when entering the run state, but don't wait for completion.
`Ondemand`	Equivalent to `respawn`.
`Powerfail`	Execute if `init` receives a power fail signal.
`Boot`	Only run at boot time, and don't wait.
`Bootwait`	Only run at boot time, and wait.
`Initdefault`	The state to enter at startup (usually "3" for multiuser).

The rc? Scripts

The `/sbin/rc?` (i.e., `rc0`, `rc1`...) scripts are higher level scripts run only by `init`. Whenever the system boots or a privileged user issues the `init` command (e.g., `init 2`), the `init` process runs the appropriate script(s) as defined in the `/etc/inittab` file. These higher level scripts, in turn, invoke any number of `start` and `kill` scripts in the corresponding directories (e.g., `/etc/rc2.d`).

Could you modify the `/etc/inittab` file to do "interesting" things with your system? Well, yes, but most of the time, you'll change the processes that run in any particular run state by creating and installing run scripts in the `/etc/rc?.d` directories. These scripts control the operations of the system at a much finer granularity than the `/sbin/rc?` scripts. Each starts or stops a particular process or set of related processes.

The proper way to add run scripts to the system is to add them to the `/etc/init.d` directory and create hard links in the directories that correspond to the run state in which they should be invoked. These commands, therefore, relate the files that have the same associated inode by generating a list of run scripts in `/etc/init.d` (where they all should reside) along with the files in the `/etc/rc?.d` directories that are hard-linked to them. Let's see what scripts are set up in these directories on a typical system. To display the scripts in a useful format, I ran the commands shown below:

Figure 1. Start/Kill Script Data Sample

```
/etc/init.d/autofs
/etc/rc2.d/S74autofs
/etc/rc0.d/K69autofs
/etc/rc1.d/K68autofs
-----
/etc/init.d/nfs.client
/etc/rc2.d/K65nfs.client
/etc/rc2.d/S73nfs.client
/etc/rc0.d/K75nfs.client
/etc/rc1.d/K80nfs.client
-----
/etc/init.d/nfs.server
/etc/rc2.d/K60nfs.server
/etc/rc3.d/S15nfs.server
/etc/rc0.d/K66nfs.server
/etc/rc1.d/K65nfs.server
-----
```

```
psycho> foreach runscript (`ls /etc/init.d/*`)
? echo $runscript ? set INODE=`ls -i $runscript | awk '{print $1}'`
? find /etc -inum $INODE -print >> /tmp/rs
? echo "-----" >> /tmp/rs
? end
psycho> cat /tmp/rs
```

Figure 2. mktab awk Script

```
BEGIN { FS = "/" } { if (NR == 1) {
    print "Proc              0                1                2               3"
    print "==============    ==============    ==============    ==============   ==============="
    }
if ($0 == "-----") {
    print COL[-1] COL[0] COL[1] COL[2] COL[3]
    for (RUNSTATE = -1; RUNSTATE <= 3; RUNSTATE++)
        COL[RUNSTATE] =    "             "
    }
else {
        RUNSTATE = substr($0,8,1)
        if (RUNSTATE == "i")
                RUNSTATE = -1
        PROC    = $4
        BLANKS  = substr("                          ",1,15-length(PROC))
        COL[RUNSTATE] = PROC cat BLANKS
    }
}
```

The file names in the /etc/rc?.d directories start with an "S" or "K" (for start or kill) followed by a two-digit number that controls the relative sequence number for running the script. Whenever the system enters a state (e.g., 0 or 2) the rc? script(s) specified in /etc/inittab will execute the scripts in the corresponding /etc/rc?.d directory that start with "K" (stop argument) and then those that start with "S" (start argument). Example data collected with the commands I listed earlier is shown in Figure 1.

The "K" scripts kill processes that should not be running in that particular run state. The "S" scripts start those processes that should be running.

Each group of linked files in Figure 1 is separated by a line of dashes. I processed this file through the script shown in Figure 2 (i.e., awk -f mktabs /tmp/rs), which results in the table shown in Figure 3.

Notice that processes are killed but none are started in state 0 (firmware). Many start processes are invoked in run

Figure 3. Run States and Scripts

Proc	0	1	2	3
ANNOUNCE	K00ANNOUNCE	K00ANNOUNCE		
MOUNTFSYS		S01MOUNTFSYS	S01MOUNTFSYS	
PRESERVE			S80PRESERVE	
README				
RMTMPFILES			S05RMTMPFILES	
acct				
asppp	K47asppp	K47asppp	S47asppp	
audit	K42audit	K42audit	S99audit	
autofs	K69autofs	K68autofs	S74	
autofs autoinstall			S72	
autoinstall				
buildmnttab				
buttons_n_dials-setup				
cron	K70cron	K70cron	S75cron	
devlinks				
drvconfig				
gsconfig			S91gsconfig	
gtconfig			S91gtconfig	
inetinit			S69inet	
inetsvc			S72inetsvc	
leoconfig			S91leoconfig	
lp	K20lp		S80lp	
mkdtab				
nfs.client	K75nfs.client	K80nfs.client	S73nfs.client	
nfs.server	K66nfs.server	K65nfs.server	K60nfs.server	S15nfs.server
perf			S21perf	
rootusr				
rpc	K85rpc	K67rpc	S71rpc	
rtvc-config			S92rtvc-config	
sendmail	K57sendmail	K57sendmail	S88sendmail	
standardmounts				
sxcmem				
sysetup			S20sysetup	
sysid.net			S30sysid.net	
sysid.sys		S71sysid.sys		
syslog	K55syslog	K55syslog	S74syslog	
ufs_quota				
utmpd	K50utmpd	K50utmpd	S88utmpd	
uucp			S70uucp	
volmgt			S92volmgt	

state 2 (multiuser without shared file systems). However, almost no processes are started in run state 3 (multiuser); this is because both the `/sbin/rc2` and `/sbin/rc3` scripts are run when the system enters run state 3. Also notice that most of the services started in run state 2 are killed in run states 0 and 1. Whenever there is a start and kill script for the same process in the same `/etc/rc?.d` directory, the process is killed and then restarted.

Another interesting feature of the `rc1`, `rc2` and `rc3` scripts is their use of the `who -r` command to determine the current and previous run states. An excerpt from `/sbin/rc2` is shown in Figure 4. Notice how the actions taken depend on the response from `who -r`. The kill scripts are only invoked when the previous state was not "S" or "1" and the state being moved into is "2" (keep in mind that `/sbin/rc2` is also run when entering run state 3). The output from this command will look roughly like this:

```
psycho% /usr/bin/who -r
   .    run-level 3  May 18 10:13    3  0  S
```

The `x$9` value (ninth field in this output) is "S" and the `x$7` value is "3."

Whether or not you're trying to dazzle your friends, the proper configuration of run state scripts will allow you to best take advantage of `init`.

S. Lee Henry *is on the board of directors of the Sun User Group and works as a security services engineer for Infonet (where no one else necessarily shares any of her opinions). Her current running state is California. Like her readers, she gets lots of email. Send her more by addressing it to* `slee@cpg.com`.

Figure 4. Excerpt from /sbin/rc2

```
set `/usr/bin/who -r`
if [ x$9 = "xS" -o x$9 = "x1" ] then
         echo 'The system is coming up. Please wait.'
         BOOT=yes
elif [ x$7 = "x2" ] then
         echo 'Changing to state 2.'
```