

## Back to the Basics: Solaris Default Processes and init.d Pt. II

Hal Flynn 2000-06-11

### Back to the Basics: Solaris Default Processes and init.d Pt. II

By Hal Flynn <hmflynn@earthlink.net>

last updated Monday, June 11, 2000

---

#### • Introduction

This article has been written to provide insight into a stock installation of Solaris 8, and the services started by default. Out of the box, Solaris 8 by runs many services. This article was written using Solaris 8, the latest version available, and a Sparcstation 20. Most of this document will apply to releases of Solaris prior to 8, and to both the Sparc and Intel architectures. For documentation purposes, a full OEM install was done. Many topics discussed will be familiar to seasoned administrators. However, this document will benefit all parties involved in the administration and security aspects of Solaris.

#### • Review

In the last article, we discussed which services are started by default, and gave a `ps -ef` of our host Thebes. If you haven't read the first part or don't recall, you can find it [here](#).

#### • Initialization

At bootstrap time, `init` is the first process started. It's the parent process of everything that will later run on the system. To touch lightly on this, after `init` is started and it inherits its environment, it begins running through the initialization scripts. These are what's known as the "Run Command" scripts, or `rc` scripts. `rc` scripts are nothing more than executable bourne shell scripts.

Solaris has a different directory for each run level. Each run level has a number or letter attached to it, and a different set of scripts in it, such as run level three (normal multiuser mode on a Solaris system), which keeps the initialization scripts in `/etc/rc3.d`, to be executed when run level 3 is entered.

Within these directories are scripts prefixed first with a letter (either K or S) a number (which determines chronologically when they are started in the sequence) and a name (which is usually a hint as to what they do, although this isn't always true). The files in these directories are hard links, which are linked to files in the master directory, `/etc/init.d`.

Below is a partial listing of what `/etc/init.d` looks like:

```
$ pwd
/etc/init.d
$ ls -li
total 250
 81666 -rwxr--r--  3 root    sys      171 Jan  5 18:55 ANNOUNCE
 81667 -rwxr--r--  3 root    sys     1881 Jan  5 18:55 MOUNTFSYS
 81668 -rwxr--r--  2 root    sys      256 Jan  5 19:27 PRESERVE
 81669 -rw-r--r--  1 root    sys     2681 Jan  5 18:55 README
 81670 -rwxr--r--  2 root    sys     2004 Jan  5 18:55 RMTMPFILES
 81720 -rwxr--r--  1 root    sys      833 Jan  5 18:52 acct
```

```

81703 -rwxr-xr-x  2 root    sys      1707 May  4  1998 afbinit
81722 -rwxr--r--  6 root    sys      572 Jan  5  19:27 apache
81723 -rwxr--r--  5 root    sys     1365 Jan  5  18:56 asppp
81671 -rwxr--r--  5 root    sys      447 Jan  5  18:55 audit
81704 -rwxr--r--  5 root    sys      364 Jan  5  18:55 autofs
81700 -rwxr-xr-x  2 root    other   1558 Dec 16  17:19 autoinstall
81672 -rwxr--r--  2 root    sys     1153 Jan  5  18:55 buildmmttab
81726 -rwxr--r--  1 root    sys      271 Dec  7  1999 buttons_n_dials-setup
81673 -rwxr--r--  2 root    sys     1101 Jan  5  18:55 cachefs.daemon
81674 -rwxr--r--  2 root    sys      392 Jan  5  18:55 cachefs.root

```

<--- Rest of output snipped --->

As you can see, our current working directory is /etc/init.d. In this directory, we have a multitude of files, all of which are executable. These files are the master rc scripts.

For documentation purposes, we'll also show a partial list of the scripts in /etc/rc2.d. Here is the output:

```

$ ls -li
total 158
 81714 -rwxr--r--  6 root    sys      861 Jan  5  19:13 K07dmi
 81715 -rwxr--r--  6 root    sys      404 Jan  5  19:13 K07snmpdx
 81722 -rwxr--r--  6 root    sys      572 Jan  5  19:27 K16apache
 81690 -rwxr--r--  6 root    sys     3080 Jan  5  18:55 K28nfs.server
126209 -rw-r--r--  1 root    sys     1369 Jan  5  18:55 README
 81667 -rwxr--r--  3 root    sys     1881 Jan  5  18:55 S01MOUNTFSYS
 81670 -rwxr--r--  2 root    sys     2004 Jan  5  18:55 S05RMTMPFILES
 81696 -rwxr--r--  2 root    sys      611 Jan  5  18:55 S20syssetup
 81721 -rwxr--r--  2 root    sys      989 Jan  5  19:01 S21perf
 81701 -rwxr-xr-x  2 root    other   1995 Dec 16  17:19 S30sysid.net
 81706 -rwxr--r--  5 root    sys      359 Jan  5  18:57 S401lc2
 81723 -rwxr--r--  5 root    sys     1365 Jan  5  18:56 S47asppp
 81683 -rwxr--r--  5 root    sys    11201 Jan  5  18:52 S69inet
 81724 -rwxr--r--  2 root    sys      327 Jan  5  18:55 S70uucp
 81686 -rwxr--r--  5 root    sys      413 Jan  5  18:55 S71ldap.client
 81693 -rwxr--r--  5 root    sys     2839 Jan  5  18:55 S71rpc
 81702 -rwxr-xr-x  2 root    other   1498 Dec 16  17:19 S71sysid.sys
 81700 -rwxr-xr-x  2 root    other   1558 Dec 16  17:19 S72autoinstall
 81684 -rwxr--r--  5 root    sys     7134 Jan  5  18:52 S72inetsvc
 81716 -rwxr--r--  5 root    sys      525 Jan  5  17:46 S72slpd
 81673 -rwxr--r--  2 root    sys     1101 Jan  5  18:55 S73cachefs.daemon

```

<---Output snipped--->

Were more output displayed in each of the lists, you'd start to see matching inodes in each output. This reasserts the fact that the scripts in each run level directory are hard links to the master files in /etc/init.d.

#### o The Functions of rc scripts

rc scripts serve two purposes. The first one is execution of the process for which they are responsible. The other is stopping the process(es) they start. With rc scripts in each run level, a script prefixed with K denotes "Kill", and S denotes "Start".

On the command line, these scripts can be executed and post-appended with a start or stop, allowing command like execution or killing of the respective processes. Here's a sample of source of one of these scripts, taken from S71rpc (listed above). We're only going to show the sections pertinent to our discussion:

```
#!/sbin/sh
#
# Copyright (c) 1997-1999 by Sun Microsystems, Inc.
# All rights reserved.
#
#ident  "@(#)rpc          1.46      99/09/22 SMI"

[ ! -d /usr/bin ] && exit

case "$1" in
'start'|'rpcstart')
    if [ "$1" = start ]; then
        if [ -z "$_INIT_PREV_LEVEL" ]; then
            set -- ` /usr/bin/who -r `
            _INIT_PREV_LEVEL="$9"
        fi
    <---Lines snipped--->
'stop')
        # Bring all of the RPC service daemons to a halt.  Note that the
        # daemons are stopped in a particular order.  Further note that rpcbind
        # is special in that it needs to be killed with -9 to prevent it from
        # saving its state and sending a message to syslog.

        for daemon in rpc.nisd nis_cachemgr keyserv rpc.nispasswd; do
            /usr/bin/pkill -x -u 0 $daemon
        done

        if [ -x /usr/lib/netsvc/yp/ypstop ]; then
            /usr/lib/netsvc/yp/ypstop
        fi

        /usr/bin/pkill -9 -x -u 0 rpcbind
        /usr/bin/rm -rf /var/run/rpc_door
        ;;

*)
    echo "Usage: $0 { rpcstart | start | stop }"
    exit 1

```

```

        ;;
esac
exit 0

```

As you can see, there's a subsection 'start'|'rpcstart', a subsection 'stop', and one last subsection '\*'. These sections contain the commands executed for each argument specified with script.

Another thing we should mention here while on the topic of shell scripts is choice of shell. As you can see, the scripts are being written with the Bourne shell, however, not /bin/sh (actually, /usr/bin/sh. /bin is a symbolic link to /usr/bin). The executable being used here is /sbin/sh. There's a valid reason for this. We'll give some output below to show why.

This is the Bourne shell executable in /usr/bin/sh:

```

$ pwd
/usr/bin
$ ls -l sh
-r-xr-xr-x  4 root  root    95308 Jan  5 19:02 sh
$ file sh
sh:          ELF 32-bit MSB executable SPARC Version 1, dynamically linked,
            stripped

```

Here is the Bourne shell executable in /sbin:

```

$ pwd
/sbin
$ ls -l sh
-r-xr-xr-x  2 root  root    275692 Jan  5 19:02 sh
$ file sh
sh:          ELF 32-bit MSB executable SPARC Version 1, statically linked,
            stripped

```

As you can see here, one is dynamically linked, and one statically linked. The reason for this is availability of libraries. When the system is bootstrapping, and init is executing, the libraries in /usr/lib may not be readily available, creating a situation where the shell can't execute. To avoid this, a statically linked binary is available, and should be used for all rc scripts.

While we're on the topic, it's also worth mentioning /sbin/sh as a login shell. It's no coincidence that this is the default login shell for root. As mentioned previously, if /usr/lib is not mounted, the shell will not execute properly. This can be displayed by changing the default shell to /bin/sh and booting into single user mode. However I don't recommend this unless you have time to fix the problem. You've been warned. If you do so, it is at your own risk. Changing the default shell (unless you've statically compiled a new shell from source to replace it with) is a bad idea.

#### o Run Levels

We'll review run levels as well, and talk about how the system initializes different things at different run levels. The first thing to do is describe the run levels (or init states...the difference is purely dogmatic).

Run level 0 is Open Boot Prom mode. This is the most basic operation level of the machine. At run level 0, there is no kernel in memory, and it's safe to shut down the system.

Run level s (or S, they're synonymous), is single user mode. This run level does not allow remote logins of other users, and makes an attempt to mount all the file systems for the performance of administrative upkeep. Few things are running at this level.

Run level 1 is similar to run level s. Run level 1 is an administrative run level, however, it also allows remote logins. Again, this administrative mode makes an effort to mount all available file systems.

Run level 2 is the first multiuser mode. This run level by default runs the standard UNIX services (telnetd, ftpd, smtp, etc).

Run level 3 is the default run level of a given Solaris system. This run level, like 2, is a multiuser mode. The key difference between run level 2 and run level three is NFS. At run level three, all NFS shared resources are available.

Run level 4 is not yet implemented, but is the third multiuser mode.

Run level 5 is a shutdown mode. This mode will take the system from its present state, shut it down, and power off the system, respectively. A note, this is not the same with Intel Solaris. This applies only to Solaris Systems running on various Sparc-style platforms.

Run level 6 is the final mode. This run level takes the system from its present state, shuts down the operating system, and reboots the system.

As the superuser on a system, any of these run levels can be reached at nearly any given time. Execution on the command line of init (i.e. init 0, init 5, etc) will take the system to the desired run level.

## • Conclusion

In this article, we built a solid understanding of the initialization of the system. We talked about init, the location of the initialization scripts, and how they're started. We also discussed the content of the startup scripts, the shell used for them, an important detail about the root shell, and run levels.

To further this understanding, in our next article, we'll look back at what services are started on a stock install of Solaris 8, and examine the scripts that make them run. From there, we'll talk about the necessity of these services, their applications, and the shutting down the unnecessary ones.

To read **Back to the Basics: Solaris Default Processes and init.d Pt. III**, click [here](#).