The Design and Implementation of the NetBSD rc.d system

Luke Mewburn

Wasabi Systems, Inc.

`lukem@wasabisystems.com`

Abstract

In this paper I cover the design and implementation of the rc.d system start-up mechanism in NetBSD 1.5, which replaced the monolithic */etc/rc* start-up file inherited from 4.4BSD. Topics covered include a history of various UNIX start-up mechanisms (including NetBSD prior to 1.5), design considerations that evolved over six years of discussions, implementation details, an examination of the human issues that occurred during the design and implementation, as well as future directions for the system.

1. Introduction

NetBSD recently converted from the traditional 4.4BSD monolithic */etc/rc* start-up script to an */etc/rc.d* mechanism, where there is a separate script to manage each service or daemon, and these scripts are executed in a specific order at system boot.

This paper covers the motivation, design and implementation of the *rc.d* system; from the history of what NetBSD had before to the system that NetBSD 1.5 shipped with in December 2000, as well as future directions.

The changes were contentious and generated some of the liveliest discussions about any feature change ever made in NetBSD. Parts of those discussions will be covered to provide insight into some of the design and implementation decisions.

2. History

There is great diversity in the system start-up mechanisms used by various UNIX variants. A few of the more pertinent schemes are detailed below. As NetBSD is derived from 4.4BSD, it follows that a description of the latter's method is relevant. Solaris' start-up method is also detailed, as it is the most common System V UNIX variant.

2.1. 4.4BSD

4.4BSD has a rather simple start-up sequence.

When booting multi-user, the kernel runs `init` (located in */sbin/init*), which spawns a shell (*/bin/sh*) to run */etc/rc*, which contains commands to check the consistency of the file-systems, mount the disks, start up system processes, etc. */etc/rc* invokes */etc/netstart* to configure the network and any associated services, and */etc/rc.local* (if it exists) for locally added services. After */etc/rc* has successfully completed, `init` forks a copy of itself for each terminal in */etc/ttys*, usually running */usr/libexec/getty* on them. [1]

Administrative configuration of system services is controlled by editing the scripts (*/etc/rc*, */etc/rc.local*, */etc/netstart*). In some instances, only shell variables need to be changed, in others commands are added, changed, or removed. [2]

4.4BSD has no specific shut down procedure. After `init` receives a `SIGTERM` signal it sends a `SIGHUP` signal to each process with a controlling terminal, which the process was expected to catch and handle appropriately. Ten seconds later, this is repeated with `SIGTERM` instead of the `SIGHUP`, and another ten seconds after that `SIGKILL` is sent. After all processes have exited or when thirty seconds had elapsed, `init` then drops to single user mode, reboots, or shuts down, as appropriate.

2.2. Solaris 7

Solaris is the most common System V variant, and serves as a good reference implementation of the System V init.d mechanism, as implemented by System V Release 4 (SVR4).

When running, the system can be in one of eight distinct run levels [3], which are distinct states in which selected groups of processes may run. The run level may be changed at any time by a privileged user running the `init` with the run level as the argument, and the run level may be determined at any time with the “`who -r`” command.

When the system is booted, the kernel runs `init` (located in */sbin/init*), whose purpose is to spawn processes defined in */etc/inittab* [4]. For each configuration line in */etc/inittab* that has a run level field (‘rlevel’) which matches the current run level, `init` starts the process defined on that line as per the given ‘action’ field. The different run levels are:

`0`	Shut down the operating system so that it’s safe to turn off the power.
`s or S`	Single user mode, with all file systems mounted.
`1`	Single user mode, with all file systems mounted and user logins allowed.
`2`	Multi user mode, with all services running except NFS server daemons.
`3`	Multi-user mode with all services running. This is usually the default.
`4`	Currently unused.
`5`	Shut down the system and attempt to turn off the power.
`6`	Shut down the system to level 0, and reboot.

For a given run level *X*, an shell script */sbin/rcX* exists to control the run level change, and */etc/rcX.d* contains scripts to be executed at the change. */sbin/rcX* stops the services in the files matching */etc/rcX.d/K** in lexicographical order, and then starts the services matching */etc/rcX.d/S** in order.

To add a new service *foo* requires adding */etc/rcX.d/S*foo* in the appropriate run level to start the service, and then */etc/rcY.d/K*foo* in all the other run levels *Y* where the service is not to be run. Usually these files are actually links to the appropriate script in */etc/init.d* which implements the start up and shut down procedures for a given service.

To disable or remove a service *foo*, any files matching */etc/rc?.d/[KS]*foo* need to be removed.

2.3. NetBSD prior to 1.5

Prior to the release of NetBSD 1.3, NetBSD’s start-up mechanism was similar to 4.4BSD’s, with relatively minor changes, as described below.

2.3.1. NetBSD 1.3

In NetBSD 1.3 (released in January 1998), two major user-visible additions were made to the start-up system; */etc/rc.conf* and */etc/rc.lkm*.

/etc/rc.conf contains variables to control which services are started by */etc/rc* and */etc/netstart*. For each service *foo*, two variables may be provided:

`$`*`foo`*	Can be “`yes`” or “`no`” (or various other boolean equivalents). If set to “`yes`”, the service or action relating to *foo* is started.
`$`*`foo`*`_flags`	Optional flags to invoke *foo* with.

The aim of */etc/rc.conf* was to separate the scripts that start services from the configuration information about the services. This allows updating of the start-up scripts in an operating system upgrade with less chance of losing site-specific configuration.

Similar */etc/rc.conf* functionality has been implemented in commercial UNIX and BSD derived systems, including current systems such as FreeBSD. By the time this change was considered for NetBSD, it had a reasonable number of users of the prior art to help justify its implementation.

/etc/rc.lkm was added to provide control over how loadable kernel modules (LKMs) are loaded at boot time. */etc/rc.lkm* is invoked at three separate stages during the boot process; before networking is started, before non-critical file systems (i.e., file systems other than */*, */usr*, */var*) are mounted, and after all file-systems are mounted. This complexity is required because an LKM may be located on a local or remote file system. The configuration file */etc/lkm.conf* controls behavior of */etc/rc.lkm*.

2.3.2. NetBSD 1.4

In NetBSD 1.4 (released in May 1999), two more additions were made; */etc/rc.shutdown* and */etc/rc.wscons*.

/etc/rc.shutdown is run at shut down time by `shutdown`. This occurs before the global `SIGHUP` is sent (as described in section 2.1). This is useful because there

are some services that should be shut down in order (e.g., database-using applications before their databases) and some services that require more than `SIGHUP` for a clean shutdown.

/etc/rc.wscons was added to control how the *wscons* console driver was configured at boot time, and to allow manual reconfiguration. */etc/wscons.conf* controls this behavior.

2.3.3. Summary prior to NetBSD 1.5

At multiuser boot, `init` calls */etc/rc* to initialize the system. */etc/rc* calls */etc/netstart* to setup network services, */etc/rc.local* for local services, */etc/rc.lkm* to initialize load-able kernel modules, and */etc/rc.wscons* to configure the *wscons* console driver. The start-up of services is controlled by variables in */etc/rc.conf*.

At system shutdown time, `shutdown` calls */etc/rc.shutdown* to shut down specific services which have to be shut down before the global `SIGHUP` that `init` sends.

3. Design considerations

Over a six year period, various ideas on how to enhance the start-up system were floated on the public NetBSD mailing lists 'current-users' and 'tech-userlevel', as well as on the NetBSD developer-only mailing list.

There was no consensus on '*One True Design*'; there was too much contention for that. What is described below is an amalgamation of what a few developers felt was a reasonable analysis of the problems and feedback as well as the most reasonable solution to support the widest variety of circumstances.

3.1. Problems with the old system

The old system was perceived to suffer from the following problems:

- There was no control over the dependency ordering, except by manually editing */etc/rc* (and other scripts) and moving parts around.

  This caused problems at various times, in situations such as workstations with remotely mounted */usr* partitions, and these problems weren't completely resolved as was seen by observing various mailing discussions and a flurry of CVS commits to the source tree.

- It was difficult to manually control an individual service after the system booted (e.g., restart `dhcpd`, shut down a database, etc).

  Whilst some people suggested that a system administrator who couldn't manually restart a service was incompetent, this doesn't resolve the issue that typing "`/etc/rc.d/amd restart`" is significantly easier that finding the process identifier of `amd`, killing it, examining */etc/rc* for the syntax that `amd` is invoked with, searching */etc/rc.conf* for any site-specific options, and manually typing in the resulting command.

  Unfortunately, there was a slight tendency during some of the mailing list discussions to resort to attacks on people's competency in this manner. I consider this a form of computer based intellectual snobbery, and an unreasonable justification for why that person disliked a feature.

- It didn't easily cater for addition of local or third party start-up mechanisms, especially addition into arbitrary points in the boot sequence, including those installed by (semi-)automated procedures such as the NetBSD 'pkg' tools.

3.2. Requirements of the new system

Given the problems in the old system, and observations of what other systems have done, including those described in section 2, the following design considerations were defined.

Some of these considerations were not determined during discussion prior to implementation, but were identified once users were actively using the implementation.

3.2.1. Dependency ordering

Dependency ordering is a strong requirement.

The following dependency ordering requirements were determined:

- Independence from lexicographical ordering of filenames.

  Some other systems (e.g., System V init.d) use an existing lexicographical ordering of filenames in a given directory, such as */etc/rc2.d/S69inet* occurring before */etc/rc2.d/S70uucp*, but experience has shown that this doesn't necessarily scale cleanly when adding local or third-party services into the order; often you end up with a lot of convoluted names around '`S99`'.

- Ability to insert local or third-party scripts anywhere into the sequence.

  Some people proposed running */etc/rc.d/** out of */etc/rc.local*, and retaining the existing */etc/rc* semantics. This doesn't easily cater to a user who requires the ability to insert their own start-up items anywhere in the boot sequence (such as a cable modem authentication daemon required for networking).

- Not bloating */bin* and */sbin* on machines with small root (*/*) file-systems. The use of tools from */usr/bin* has to be avoided because */usr* might not be available early in the boot sequence.

- Use a dynamic dependency ordering.

  A lot of debate occurred regarding whether the dependency ordering is predetermined (e.g., by creating links to filenames or building a configuration file), or dynamically generated.

  A predetermined order may be more obvious to determine the order (using '`ls`' or examining the configuration file instead of invoking a special command), but it can be difficult to add a service in at a given point on a system because generally ordering is not based on services provided.

  A dynamic order may slow down boot slightly, but provides the flexibility of specifying start-up order in terms of dependencies.

  For example, if service *C* depends on *B* which depends on *A*, and I have a new service *D* to install that must start after *A* and before *C* then I want to specify it in these terms, without having to worry about whether it starts before *B*, after *B*, or simultaneously with *B*.

  There was some discussion about various methods in which to determine the dynamic ordering:

  - Using `make` and a *Makefile*.
  - Using `tsort`, `awk`, and a few shell commands
  - Providing a dedicated ordering tool which parsed the scripts for command directives in special comments to determine the order. If a script did not have a directive, it would be ordered last.

After various discussions and implementation tests, it was decided that a dedicated dynamic ordering tool, `rcorder` (see section 4.2.6), was the most appropriate mechanism; using `make` or `tsort` and `awk` would require moving those programs to */bin* ('bloating' the root file-system for machines with limited resources), and a dedicated tool could provide better feedback in certain error situations.

3.2.2. Manipulation of individual services

Most people seem to agree that the ability to manipulate an individual service (via a script) is one of the benefits of the System V init.d start-up mechanism. Having a script that allows direct starting, stopping, and restarting of a service, as well as other per-service options like 'reloading configuration files', significantly reduces system administrator overheads.

Having the same script be used by the start-up sequence is also highly desirable, as opposed to using a monolithic */etc/rc* for booting and separate */etc/rc.d* scripts for manual control (which had been suggested).

It is interesting to note that some System V init.d implementations often start multiple services in the one file, which defeats the purpose of providing per-service control files. An example is Solaris' */etc/init.d/inetsvc*, which configures network interfaces, starts `named` and starts `inetd`.

3.2.3. Support third-party scripts

An important requirement is the ability to support third-party scripts, especially by allowing them to be inserted at any place in the boot sequence order.

The current system does support third-party scripts if they are installed into */etc/rc.d*. There has been discussion about allowing for different directories to be used for local and third-party scripts, in order to provide a separate 'name-space' to prevent possible conflicts with a local script and a future base system script, but so far none of the suggestions has been considered sufficiently complete to provide in the default system. This, however, does not prevent a site from implementing their own method.

3.2.4. Maintain /etc/rc.conf

/etc/rc.conf was introduced in NetBSD 1.3, and most users seem fairly happy with the concept.

One of the concerns about a traditional System V init.d style mechanism is that the control of service start-up is managed by the existing of a link (or symbolic link) from */etc/rc2.d/S69inet* to */etc/init.d/inetinit*, which is difficult to manage in a traditional configuration change management environment (such as RCS). Similar concerns exist regarding the suggestion of using mode bits on files in */etc/rc.d* to control start-up.

/etc/rc.conf was further enhanced as described in section 3.3.

3.2.5. Promote code re-use

Traditional System V init.d implementations do not appear to re-use any code between scripts. From experience, maintaining local scripts in a traditional *init.d* environment is a maintenance nightmare. We achieved code re-use with common functions in */etc/rc.subr* which results in the average */etc/rc.d* script being a small (5-10 line) file. There were some concerns raised about using these common functions, but they weren't considered to be serious issues. (We have a C library and common Makefile fragments, so why not common shell functions?)

3.2.6. Service shut down

The ability to shut down certain services at system shutdown time with */etc/rc.shutdown* was a useful feature of the previous system and of other systems, and it makes sense to retain this feature.

In the initial implementation, we reverse the dependency order, and shut down any services which are tagged with a '`shutdown`' keyword (see section 4.2.6) within the script. We may modify or enhance this behavior if observation of in-field use reveals a more complicated scheme is required.

3.2.7. Avoid mandatory run levels

We avoided the use of System V run levels (also known as run states or init states) and */etc/inittab*. This was the result of many discussions about the design, which can be summarized to:

- They're just too contentious; the */etc/inittab* concept had the least number of advocates. Many people expressed the opinion (both during the design phase and post implementation) that they don't mind the */etc/rc.d* idea but don't think an */etc/inittab*, run-levels or */etc/rcN.d* directories would improve things.

- There doesn't seem to be consistency between what each run-level means on various System V init.d implementations, or the exact semantics of what occurs at state change. Thus, using the argument of compatibility for system administrator ease of use isn't as relevant. Some systems (such as HP/UX 10.x) treat these as levels, where a transition from level 4 to level 2 executes the shut down scripts in level 3 and then level 2. Other systems (such as Solaris) treat these as separate run states, where a transition to a level runs all the stop scripts in that level and then all the start scripts. This can be confusing to an administrator, as well as not necessarily providing the optimal behavior.

- We currently support single-user mode (s), multiuser mode (2 or 3, depending on whether NFS serving is configured in */etc/rc.conf*), shutting down the system to single user mode (1), halting the system (0), rebooting the system (6), and powering off the system (5) (with the equivalent Solaris init state in parenthesis).

- If the ability to take the system from a given point in the order to another point in the order, then I feel that most people's requirements for what run-levels are touted to provide would be met. This is currently a work in progress.

- Whilst */etc/inittab* provides for re-spawning of daemons, in practice very few daemons are actually started that way, and it's trivial to implement that feature in a few lines of shell script as a 'wrapper' to the start of the daemon.

3.2.8. Other issues

After various discussions, we settled on the name */etc/rc.d* instead of */etc/init.d*, because the implementation was different enough from the System V /etc/init.d mechanism that we decided not to confuse people expecting the exact System V semantics. Many system administrators may be used to referring directly to */etc/init.d/foo* or */sbin/init.d/bar* when manipulating a service; a symbolic link from */etc/init.d* or */sbin/init.d* to */etc/rc.d* on their systems could help retain their sanity.

The first implementation of */etc/rc.d* that I released for evaluation supported all three start-up schemes; the original monolithic */etc/rc*, a System V init.d (without run-levels), and the current */etc/rc.d*. These were all built from the same sources, and a command was provided to generate the style that an administrator preferred. After feedback and discussion, this functionality was abandoned, because:

- It is very difficult to support multiple ways of starting the system when users have problems or questions, especially so in a volunteer project.

- Two of the methods (*/etc/rc*, and System V init.d) do not have the ability to dynamically order the dependency list. In those situations, an administrator (or automatic application) would have to perform the extra step of 'rebuild order' upon installation.

- The source scripts had various constraints to ensure that they could work as part of */etc/rc* as well as acting as a stand-alone script in */etc/rc.d* or */etc/init.d*.

As architects of the NetBSD operating system, we have the responsibility to provide useful solutions to problems. In general, those solutions should be as flexible as possible, without introducing *unnecessary* flexibility, which will only cause confusion. Therefore, the alternative mechanisms were dropped.

That said, the current system is flexible enough that if a site decided to use a System V init.d approach, it is fairly trivial to populate */etc/rcN.d* with a symbolic link farm to files in */etc/rc.d* (using `rcorder` to build the dependency list), and modify */etc/rc* to run the scripts in */etc/rcN.d/* in lexicographical order, or to even implement a System V */etc/inittab* and run states.

Unfortunately, there is no easy solution for people who want to retain */etc/rc*. However, as NetBSD is an Open Source project and allows for public access to the CVS source code repository (via anonymous CVS as well as via a WWW front-end [6]), nothing prevents users from reverting to the old style */etc/rc*.

It is interesting that the people who argued the most to retain */etc/rc* are probably those who are skilled enough to maintain this, and during the various discussions some even offered (some might say "threatened") to maintain their own copy of */etc/rc* in their own public CVS server for those who wished to retain this functionality. Interestingly, over a year has passed since the implementation of this work and there is no evidence that any */etc/rc* splinter work has actually occurred.

3.3. Configuration improvements

The */etc/rc.conf* mechanism was enhanced in two ways:

- The default configuration settings were moved from */etc/rc.conf* to */etc/defaults/rc.conf*, and */etc/rc.conf* sources the former. Site specific configuration overrides are placed in */etc/rc.conf*. This enables easier upgrades (both manual and automatic) of the default settings in */etc/defaults/rc.conf* for new or changed services.

  There was debate about this change, but a significant majority of users agreed with the change. Also, FreeBSD had made a similar change some time before, with a similar debate and outcome, and subsequent upgrade benefits observed which helped the case supporting the change.

- An optional per-service configuration file in */etc/rc.conf.d/SERVICE* was added. This configuration file (if it exists) is read after */etc/rc.conf*, to allow per-service overrides. This optional functionality was added to allow automated third-party installation mechanisms to easily add configuration data.

  Migrating entirely away from */etc/rc.conf* to a multitude of */etc/rc.conf.d/SERVICE* files was considered, but no consensus was reached, and after a local trial, we decided that providing for the latter but retaining the former satisfies proponents of either side.

  Recently (post NetBSD 1.5), a */sbin/chkconfig* command has been added (similar to the equivalent command in IRIX) to manage */etc/rc.conf.d* by displaying a setting or changing its value.

Thus, the order that configuration information for a given service *foo* is read in is as follows:

- *foo* sources */etc/rc.conf*.

- */etc/rc.conf* sources in */etc/defaults/rc.conf* (if it exists), and machine specific overrides of the defaults are added at the end of */etc/rc.conf*.

- A per-service configuration file in */etc/rc.conf.d/foo* (if it exists) will be loaded. This allows for automated maintenance of */etc/rc.conf.d* configuration files, whilst retaining the popular */etc/rc.conf* semantics.

4. Implementation & aftermath

The system was implemented as described above in the design section, although the design was slightly fluid and did change as feedback was incorporated.

There are two elements to the post-implementation analysis; the human issues, and the technical details.

4.1. The human issues

There was a lot of feedback, debate, angst, flames, and hate-mail. The change has been one of the most contentious in the history of the project.

The first commits to the source code repository were made with the intention of providing a mostly complete implementation which was to be incrementally improved over a few months before the release of NetBSD 1.5.

Unfortunately, we made one of our largest implementation mistakes at this point; we didn't warn the user-base that this was our intention, and the commits were seen as a 'stealth attack'. This was partly because we felt that there had been enough debate and announcing our intentions would have delayed the project another few months for a rehash of the same debate (which had been going on for five years at that point).

After the initial implementation, various technical and 'religious' complaints were raised about the system. A summary of these is:

- "*The use of 'magic' functions [from /etc/rc.subr] is bad.*"

  It was felt that the code re-use that */etc/rc.subr* promotes was sufficiently worthy to justify its continued use, as described in section 3.2.5.

- "*Switching from /etc/rc is not the BSD way, …*"

  This particular objection was expected; it's a religious argument and the change was bound to annoy a certain section of the community.

  Robert Elz, a long time user and contributor to BSD, had a good point to make about 'the BSD way': "*[the BSD way is to] find something that looks to be better (in the opinion of the small group deciding such things), implement it, and ship it.*"[5]

  In this case, the 'small group' was the NetBSD core team, who voted in unanimous agreement for the work, with the proviso that it would be tweaked and improved as necessary, which is what occurred.

- "*Why wasn't a System V init.d implemented?*"

  This was covered in section 3.2.7.

Because some of the detractors were quite vocal in the complaints, there was a perception for a time that the work was against a majority decision. This was far from the truth; many users and developers had become jaded with the discussion over the years and did not bother to argue in support of the change, since they agreed with it in principle, if not in implementation particulars. This was borne out by the level of support for the change in the time since implementation.

4.2. The technical details

The *rc.d* system comprises of the following components:

/etc/rc	System start-up script.
/etc/rc.shutdown	System shutdown script.
*/etc/rc.d/**	Individual start-up scripts.
/etc/rc.subr	Common shell code used by various scripts.
/etc/defaults/rc.conf	Default system configuration.
/etc/rc.conf	System configuration file.
*/etc/rc.conf.d/**	Per service configuration file.

4.2.1. /etc/rc

On system start-up, */etc/rc* is executed by `init`.

/etc/rc then calls `rcorder` to order the scripts in */etc/rc.d* that do not have a '`nostart`' `rcorder` keyword to obtain a dependency list of script names. */etc/rc* then invokes each script in turn with the argument of '`start`' to start the service.

The purpose of the '`nostart`' support it to allow (primarily third-party) scripts which are only to be manipulated manually (and not started automatically) to be installed into */etc/rc.d*. No scripts in the standard NetBSD distribution use this feature as yet.

4.2.2. /etc/rc.shutdown

At system shutdown, */etc/rc.shutdown* is executed by `shutdown`. `halt`, `reboot` and `poweroff` do not call this script.

/etc/rc.shutdown then calls `rcorder` to order the scripts in */etc/rc.d* that have a '`shutdown`' `rcorder` keyword to obtain a dependency list of script names. This dependency list is then reversed, and */etc/rc.shutdown* then invokes each script in turn with the argument of '`stop`' to stop the service.

The rationale for this is that only a few services (such as databases) actually require a shutdown mechanism more complicated than the `SIGHUP` sent by */sbin/init* at shutdown time. Also, having every script perform '`stop`' slows down system shutdown as well as causing problems in other areas (such as cleanly un-mounting a 'busy' NFS mount once the networking services have been stopped).

4.2.3. /etc/rc.d/* scripts

The scripts in */etc/rc.d* are invoked by */etc/rc* (with an argument of '`start`') and */etc/rc.shutdown* (with an argument of '`stop`') in the order specified by `rcorder` to start and stop (respectively) a given service.

The */etc/rc.d* scripts can be invoked manually by a system administrator to manipulate a given service (such as starting, reloading configuration, stopping, etc.)

Each script should support the following (mutually exclusive) arguments:

`start`	Start the service. This should check that the service is to be started as controlled by */etc/rc.conf*. Also checks if the service is already running and refuses to start if it is. This latter check is not performed by standard NetBSD scripts if the system is starting directly to multi-user mode, to speed up the boot process.
`stop`	Stop the service if */etc/rc.conf* specifies that it should have been started. This should check that the service is running and complain if it is not.

Other arguments which are supported in the standard NetBSD */etc/rc.d* scripts include:

`restart`	Effectively perform 'stop' then 'start'.
`status`	If the script starts a process (rather than performing a one-off operation), show the status of the process. Otherwise, it's not necessary to support this argument. Defaults to displaying the process ID of the service (if running).
`rcvar`	Display which */etc/rc.conf* variables are used to control the start-up of the service (if any).

If the argument is prefixed by '`force`', then tell the script to as if the */etc/rc.conf* variable which controls that service's start-up has been set to '`yes`'. This allows a system administrator to manually control a service disabled by */etc/rc.conf* without editing the latter to enable it. This does not skip the check which determines if the service is already running.

Other arguments for manual use by a system administrator (such as '`reload`', etc) can be added on a per service basis. For example, */etc/rc.d/named* supports '`reload`' to reload `named`'s configuration files without interrupting service.

There are some 'placeholder' services which can be required by a service to ensure that it is started before or after certain operations have been performed. These scripts generally have a name that is all upper case, and in the order found in the default boot sequence are:

`NETWORK`	Ensure basic network services are running, including general network configuration (`network`), and `dhclient`.
`SERVERS`	Ensure basic services (such as `NETWORK`, `ppp`, `syslogd`, and `kdc`) exist for services that start early (such as `named`), because they're required by `DAEMON` below.
`DAEMON`	Before all general purpose daemons such as `dhcpd`, `lpd`, and `ntpd`.
`LOGIN`	Before user login services (`inetd`, `telnetd`, `rshd`, `sshd`, and `xdm`), as well as before services which might run commands as users (`cron`, `postfix`, and `sendmail`).

4.2.4. /etc/defaults/rc.conf, /etc/rc.conf, /etc/rc.conf.d/*

/etc/defaults/rc.conf contains the default settings for the standard system services, and is provided to facilitate easier system upgrades. End users should not edit this file.

/etc/rc.conf is the primary system start-up configuration file. It reads in */etc/defaults/rc.conf* (if it exists), and the end-user puts site-local overrides of these settings at the end of the */etc/rc.conf*. This makes it more obvious to between what is a system default and what is a site-local change, and provides similar functionality to FreeBSD's */etc/defaults* mechanism.

For a given service *foo*, it is possible to have a per-service configuration file in */etc/rc.conf.d/foo*, which is read after */etc/rc.conf*. This was provided to allow third-party installation tools to install a default configuration without requiring them to in-line edit */etc/rc.conf*.

Example */etc/rc.conf* entries for `dhclient` are:

```
dhclient=YES
dhclient_flags="-q tlp0"
```

To ensure that the system doesn't start into multi-user mode without the system administrator actually checking the configuration of the system, the variable '`rc_configured`' is set to '`no`' by default, and must be set to '`yes`' by the system administrator. If this is not set, the system will not boot into multi-user mode, and instead remain in single-user mode. The system installation tool '`sysinst`' makes this change for you when configuring a newly-installed system, but users performing manual installations or upgrades need to be aware of this.

As */etc/rc.conf* is a shell script, it is possible to put various shell commands into the script to conditionally set flags if necessary. Be aware, however, that if the script exits then any script that sources */etc/rc.conf* (such as the system boot scripts) will exit too. As */etc/rc.conf* may be loaded early in the boot sequence (possibly before */usr* is mounted), not all commands may not be available for use.

4.2.5. /etc/rc.subr

/etc/rc.subr is a shell script that's sourced by the */etc/rc.d* scripts. It contains 'helper' shell functions for commonly used operations:

- `checkyesno` *`var`*

  Determine if the given variable *`var`* is set to 'yes' or 'no' (or a variant), and return with an exit code of 0 if yes, 1 if no.

- `check_pidfile` *`pidfile procname`*

  Parse the first line of the specified file *`pidfile`* for a PID, and print the PID if that process is running and matches the given process name *`procname`*.

- `check_process` *`procname`*

  Print a list of PIDs that match the given process name *`procname`*.

- `load_rc_config` *`command`*

  Load in the rc.conf configuration for *`command`*, first from */etc/rc.conf*, and then from */etc/rc.conf.d/*`command` (if it exists.)

- `run_rc_command` *`arg`*

  Implement the 'guts' of an rc.d script. This is highly flexible, and supports many different service types. *`arg`* is argument describing the operation to perform (e.g., 'start' or 'stop'). The behavior of `run_rc_command` is controlled by shell variables defined before invoking the function.

In traditional System V init.d systems (e.g., Solaris), each script contains the code to determine if a script should be started or shut down, and often re-implemented the checks for a running process, etc. These scripts become difficult to maintain, and are often 1-2 pages long.

By using the functions in */etc/rc.subr*, the standard NetBSD rc.d scripts are quite small in comparison.

For example, the */etc/rc.d/dhclient* script (sans comments which aren't used by rcorder) is:

```
#!/bin/sh
#
# PROVIDE: dhclient
# REQUIRE: network mountcritlocal

. /etc/rc.subr

name="dhclient"
rcvar=$name
command="/sbin/${name}"
pidfile="/var/run/${name}.pid"
load_rc_config $name
run_rc_command "$1"
```

It is not mandatory for scripts to use these functions. An ordinary shell script (with the appropriate rcorder control comment lines) which supports the arguments 'start' and 'stop' should work at system start-up and shutdown without modification. In order to be consistent with the existing rc.d scripts, in may help to also support 'restart', 'status', 'rcvar' (if appropriate), as well as the 'force' prefix.

4.2.6. rcorder

The ordering of the scripts in */etc/rc.d* is performed by rcorder (located in */sbin/rcorder*), which prints a dependency ordering of a set of interdependent scripts. rcorder reads each script for special comment lines which describe how the script is dependent upon other services, and what services this script provides.

Example rcorder comment lines for */etc/rc.d/dhclient* follow:

```
# PROVIDE: dhclient
# REQUIRE: network mountcritlocal
```

In this case, dhclient requires the services 'network' (to configure basic network services) and 'mountcritlocal' (to mount critical file-systems required for early in the boot sequence, usually */var*), and provides the service 'dhclient' (which happens to be depended upon by the placeholder script */etc/rc.d/NETWORK*).

It is possible to tag a script with a keyword which can be used to conditionally include or exclude the script from being returned by rcorder in the result. */etc/rc* uses this to exclude scripts that have a 'nostart' keyword, and */etc/rc.shutdown* uses this to only include scripts that have a 'shutdown' keyword. For example, as xdm needs to be shut down cleanly on some platforms, */etc/rc.d/xdm* contains:

```
# KEYWORD: shutdown
```

The rcorder dependency mechanism enables third-party scripts to be installed into */etc/rc.d* and therefore added into the dependency tree at the appropriate start-up point without difficulty.

5. Future Work

I'd like to implement functionality to allow you to start up (or shut down) services from service *A* to service *B*. This would allow you to start in single user mode, and then start up enough to get the network running, or start all services until just before 'multi user login', or just those between 'network running' to 'database start', etc.

This could be a fairly simple system, and would provide most of the functionality that most people seem to want run states for.

I encourage other systems that are still using a monolithic /etc/rc and who would like to resolve some of the similar issues NetBSD had, to consider this work. I would like to liaise with the maintainers of those systems to ensure as much code re-use as possible.

6. Conclusion

NetBSD 1.5 has a start-up system which implements useful functionality such the ability to control the dependency ordering of services at system boot and manipulate individual services, as well as retaining useful features of previous releases such as */etc/rc.conf*.

This work was extremely contentious and difficult to implement because of this contentious nature. The implementation phase did provide valuable insight into some of the difficulties involved in the design and development of large open source projects.

I the long run I believe that this work will have benefits for a majority of users, both in day-to-day operation of the system as well as during future upgrades from NetBSD 1.5.

Availability

This work first appeared in NetBSD 1.5, which was released in December, 2000 [7]. The CVSweb interface [6] can be used to browse the work and its CVS history.

Acknowledgements

Many people contributed to the discussions and design of the current system.

However, some people in particular provided some of the important elements: Matthew Green for finishing rcorder and providing the initial attempt at splitting */etc/rc* into */etc/rc.d*, and Perry Metzger for the idea of providing dependencies using a 'PROVIDE' and 'REQUIRE' mechanism, and for the initial `rcorder` implementation.

References

[1] M. K. McKusick, K. Bostic, M. J. Karels, & J. S. Quarterman, *The Design and Implementation of the 4.4BSD Operating System*, Addison-Wesley, Reading, MA, 1996.

[2] M. K. McKusick, K. Bostic, M. J. Karels, & S. J. Leffler, "Installing and Operating 4.4BSD UNIX", in *4.4BSD System Managers Manual*, pp. 1:44-53, O'Reilly & Associates, Sebastopol, CA, 1994.

[3] Sun Microsystems, *System Administration Guide, Volume I*, Sun Microsystems, Palo Alto, CA, 1998.

[4] Sun Microsystems, "init(1M)", in *man Page(1M) System Administration Commands*, Sun Microsystems, Palo Alto, CA, 1998.

[5] Robert Elz, in email to tech-userlevel@netbsd.org: *http://mail-index.netbsd.org/tech-userlevel/2000/03/17/0010.html*

[6] The NetBSD Project, "CVS Repository", *http://cvsweb.netbsd.org/*

[7] The NetBSD Project, "Information about NetBSD 1.5", *http://www.netbsd.org/Releases/formal-1.5/*