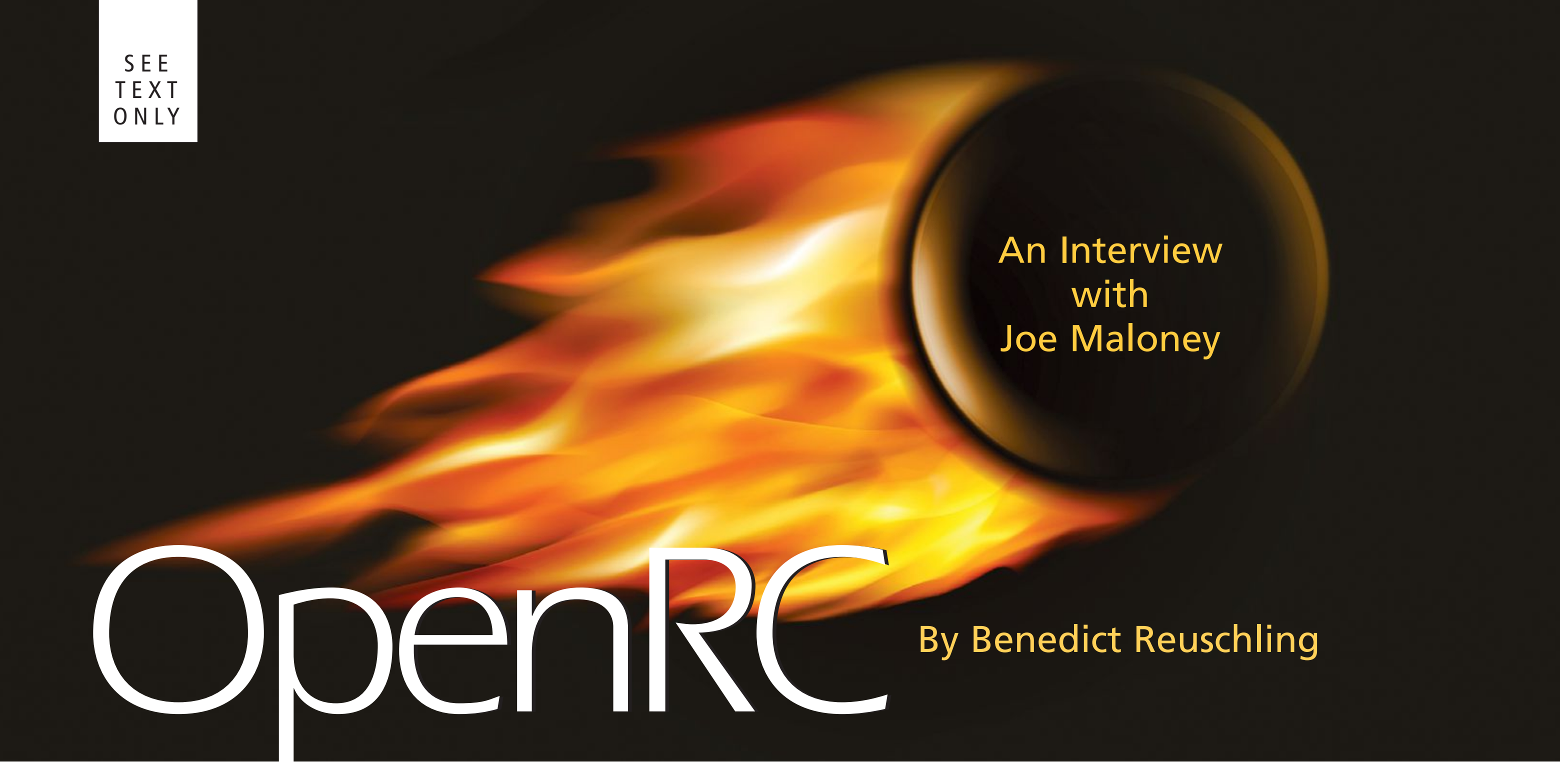
SEE TEXT ONLY



OpenRC

An Interview with Joe Maloney

By Benedict Reuschling

Tell us a little bit about yourself, and how you got involved with OpenRC.

● My name is Joe Maloney, and I am the Lead Systems Architect for the TrueOS project. My primary responsibilities include helping with the overall direction of the project, support, and Q&A for the project. I can partially be blamed for the name change away from PC-BSD as well as the rolling release model concept that drives our development pace. I am even more to blame for the integration of OpenRC into the TrueOS project.

A couple of years ago while attending my first conference at vBSDcon 2015, I was inspired to help pursue the portability use case. Replacing `init` would not have provided the gradual approach we desired, so we landed on OpenRC, which allows us to evolve our project in phases.

Before talking about replacing the `rc` system, can you give us a brief overview of the FreeBSD boot process?

● Within the FreeBSD boot process, the kernel loads any drivers specified to run early using `loader.conf`. This means that drivers are loaded as modules before the kernel boots. The kernel itself then boots within about 6 seconds. The system `V init` program `/sbin/init` starts as `pid 1`, and executes a shell file called `/etc/rc`.

What exactly is `rc`?

● The `/etc/rc` file is nothing more than a shell script that is executed by `/sbin/init`, which exists to execute processes at bootup. Prior to `rc.d`, all processes had to be hardcoded into `/etc/rc`.

This approach had the cons of hanging the boot process if a mistake was made while the user was editing `/etc/rc`. Users would have to keep track of service ordering when using this method. They would have to daemonize processes by hand, and of course this would provide no easy means of checking service status.

What is `rc.d`?

● To solve these issues, and other problems, FreeBSD adopted `rc.d` from the NetBSD community. The `rc.d` system is a collection of shell scripts that often source other shell scripts. This slows down the system considerably when you look at something like `netif`, which, including the sourcing, processes over 4,000 lines of shell as a one-time action at bootup.

The `rc.d` system allows for arguments startup order, service status, and daemonizing within each script. Base default configuration for defaults is done in `/etc/defaults/rc.conf`. User configuration for `rc.d` is typically done with `/etc/rc.conf`. This is where a service is enabled, disabled, or additional flags for the service are set.

What were prior efforts to improve rc.d?

● Over many years the PC-BSD project tested DHCP vs SYNCDHCP for wireless. The difference between the two is that DHCP would background, making the boot process a little faster, and SYNCDHCP would foreground. Unfortunately, `wpa_supplicant` was unreliable for many users in our community without foregrounding with the SYNCDHCP flag, and this hung the boot process without the backgrounding functionality.

Briefly we looked at patches for `rcorder`. There were a series of patches to enable parallel startup for `rc.d`. Unfortunately, over the years, these patches were neglected, and, as many changes were made, they were no longer able to apply cleanly without a great amount of effort.

Kris Moore, the founder of PC-BSD, also previously wrote patches to add an `rc_delay` function. This would have allowed the networking, or other services delaying boot, to run delayed. However, it was suggested within reviews that parallel startup be used instead, and these commits unfortunately died in the FreeBSD reviews process as well. Still, even if that work had been accepted, it would not have been the ideal solution. For example, think about the case where login over ldap needs to happen, and network needs to be up for that. Delaying the network would only break that use case.

Why was OpenRC developed and where does it originate?

● OpenRC is an evolution of `rc.d`. It is not an `init` system. It works with the system-provided init system. Is OpenRC a drop in replacement for `rc.d`? Well, almost. Yes, there are thousands of init scripts in ports that are being converted to a simpler format with less shell. OpenRC is mostly written in C. The primary objective to using OpenRC is to be able to reduce the amount of shell by using built-in C functions, and to simplify services.

OpenRC was written by Roy Marples, a NetBSD developer. Gentoo, PacBSD, UbuntuBSD, and others use OpenRC. Work on OpenRC originally started in the Gentoo alt fork which uses the BSD kernel. When it came time for release, the Gentoo project gave Roy the blessing to release a portable version under the BSD license.

What was needed to integrate OpenRC into the TrueOS base?

● For TrueOS we removed code for Linux portability. We found this was the primary requirement for `gmake`. To integrate into our FreeBSD fork, and build with world, this had to change. Our OpenRC implementation uses BSD make now, and is packaged along with the rest of base. We added hooks to `/etc/rc, /etc/rc.shutdown`, `/etc/rc.devd`. We did not replace the scripts entirely. We just simply added the hooks to start OpenRC as the upstream project would have if installed using `gmake`. We reworked many `rc.d` scripts to be `init.d` compatible. The `/etc/init.d/` location for base scripts replaces `/etc/rc.d`. Both `/etc/defaults/rc.conf` and `/etc/rc.conf` no longer start, or stop, services but can still be used for flags. The `/etc/conf.d/` directory replaces the `/etc/conf.d/` directory, which is not as commonly used by some users.

The end result looks something like this in our fork:

Modifications to hook in OpenRC:
https://github.com/trueos/freebsd/blob/drm-next/etc/rc
https://github.com/trueos/freebsd/blob/drm-next/etc/rc.devd
https://github.com/trueos/freebsd/blob/drm-next/etc/rc.shutdown

New additions with only top level Makefiles in FreeBSD modified to build these directories:
https://github.com/trueos/freebsd/tree/drm-next/etc/init.d
https://github.com/trueos/freebsd/tree/drm-next/lib/libeinfo
https://github.com/trueos/freebsd/tree/drm-next/libexec/rc
https://github.com/trueos/freebsd/tree/drm-next/bin/rc-status
https://github.com/trueos/freebsd/tree/drm-next/sbin/openrc
https://github.com/trueos/freebsd/tree/drm-next/sbin/rc-service
https://github.com/trueos/freebsd/tree/drm-next/sbin/rc-update
https://github.com/trueos/freebsd/tree/drm-next/sbin/start-stop-daemon
https://github.com/trueos/freebsd/tree/drm-next/sbin/supervise-daemon

Were there any roadblocks during the migration?

● During the migration process, it became clear that neither parallel startup or `rc_delay` would ever have truly fixed much at all. Especially when processes foreground rather than background. I believe `rc.d` itself could be improved by simply modernizing many of the scripts, and trimming the fat where possible. The proof for me was running a smaller, and simpler network script in shell, and seeing the difference for myself.

Our first effort can be seen here:
https://github.com/trueos/freebsd/blob/fa10ac7ceb7048baad84fd3455a77edeb118c0a2/etc/init.d/network

That script works with parallel startup, and `netif` does not.

The FreeBSD `netif` script itself is fairly minimal:
https://github.com/freebsd/freebsd/blob/master/etc/rc.d/netif

However, the files it sources for functions are where the performance problems begin:
https://github.com/freebsd/freebsd/blob/master/etc/network.subr
https://github.com/freebsd/freebsd/blob/master/etc/rc.subr

What was missing was `ip alias`, `lagg`, and some other edge case functionality. Therefore it was decided to move back to `netif` for the time being for overall compatibility. This has somewhat hurt our performance, and prevented parallel from operating properly under the circumstances. However, we cannot justify keeping users from that functionality.

We can of course remove `netif` from the boot runlevel, and gain better performance by simply backgrounding networking by allowing `devd` to start it later. In TrueOS we need something to apply only specific configuration when needed rather than just a one-time evaluation at startup. Again, we are back to the issue of not really solving anything here for someone who needs network at the login manager. Some of us think that the way forward may be to write a network manager that works more efficiently in event-driven scenarios with `devd`.

Another consideration is that we currently use `dhcpcd` as our default `dhcpcd` client:
https://github.com/trueos/dhcpcd

We have not yet accomplished the task of integrating directly into the tree as NetBSD has:
http://cvsweb.netbsd.org/bsdweb.cgi/src/external/bsd/dhcpcd/

However, this is on the roadmap to complete integration for TrueOS as well as offering `dhclient` integration in base for those who wish to use it. We found in the particular case of parallelization efforts that `dhcpcd` worked better for that use case, so we chose it as our default `dhcp` client. We will have to revisit dhclient in the future.

To summarize on our to-do list:

- Import `dhcpcd` into base
- Add `dhclient` support
- Try to fix `netif` parallelization
- Possibly write a network manager down the road and make it the default option

What improvements did you gain from the migration to OpenRC?

● Even without parallel startup, TrueOS averages a 10- to 15-second startup in most cases, a 127% improvement. Some would argue they have a 9-second startup without OpenRC but obviously they are not running all of the many services we are in TrueOS out of box to support a more general use case for the average user. Of course, no login manager, and direct startup to fluxbox without `cups`, `avahi`, and many services will be faster. However, if you start 100 services, you will see a slowdown when `rc.d` is used. That is where parallel startup is useful.

We also have service supervision capability directly integrated into OpenRC. We are currently using it for sysadm to monitor the service, and restart it in the case of crashes. Other supervision backends such as `s6` can also be used even without replacement of the `init`. This can provide socket activation functionality. However, the built-in supervisor has proven functional enough for our use case.

We now have simpler service files due to `openrc-run` built-in functions. Some services can simply start with name and command specified. All services now report service status properly. With FreeBSD's `rc.d` services, we were finding that many services were not reporting service status properly. We have found the same services fixed during the migration to OpenRC.

OpenRC also provides a nice representation of the startup of services. It makes it easy to see failures, with a warning by color code. Services with errors will show up in red, services with warnings are yellow. In the case of parallel startup, blue is the general theme, and in the case of normal startup, green is the general theme. Of course, for a more traditional `rc` look, color can be turned off with a configuration parameter in `rc.conf`.

Finally we gain the ability to provide runlevels. These runlevels are not the same as Linux SysV runlevels but allow us to order services by grouping them into a runlevel.To summarize on the improvements:

- Parallel startup capability
- Service supervision capability
- Simple startup scripts
- Improved reporting of service status
- Improved display of service startup
- Runlevels, and stacked runlevels

Lastly our most recent version including OpenRC 26.2 includes the ability to show service uptime for supervised services, and we are now monitoring services.

Can you provide us with a project-hosted `init.d` script example?

● Here is an example that uses the new supervise-daemon:
https://github.com/trueos/sysadm/blob/master/src/init.d/sysadm

The `sysadm` service will use the supervise daemon to keep the service running even if it crashes. The process will only stop if the user stops the service cleanly.

What about an integrated script example for the ports tree?

● For ports the following overlay structure is used:
https://github.com/trueos/freebsd-ports/blob/trueos-master/devel/dbus/Makefile.trueos
https://github.com/trueos/freebsd-ports/blob/trueos-master/devel/dbus/files/openrc-dbus.in

The original rc.d scripts from upstream are preserved, and can still coexist on the system:
https://github.com/trueos/freebsd-ports/blob/trueos-master/devel/dbus/files/dbus.in

How difficult is it to convert an `rc.d` script to OpenRC?

● An rc.d script can be simply converted in many cases. Most of the time it just involves removing a few simple lines. A few common offenders:

- `./etc/rc.subr`
- `-rcvar=`
- `-exit_code=0-`
- `-load_rc_config ${name}`

A more in-depth guide is available on the TrueOS blog: https://www.trueos.org/blog/openrc-update-simplifying-openrc-scripts/. We also highly recommend using `man openrc-run` to view all of the available options for writing `init.d`-compatible scripts. The TrueOS project is 100% complete now with the process of adding init.d compatible scripts to ports.

Do you think FreeBSD should consider migrating to OpenRC too?

● I see FreeBSD as an excellent vehicle for appliance development. I currently hold the view that TrueOS should be seen as just another appliance based on FreeBSD.

There is more to be done but we hope to continue the work toward OpenRC to make it an even more attractive option in the future. Of course, the TrueOS project considers the OpenRC licensing model to be just right, and aside from the netif parallelization bottleneck, there is not much work to be done aside from general script conversion.

BENEDICT REUSCHLING joined the FreeBSD Project in 2009. After receiving his full documentation commit bit in 2010, he actively began mentoring other people to become FreeBSD committers. He is a proctor for the BSD Certification Group and joined the FreeBSD Foundation in 2015, where he is currently serving as vice president. Benedict has a Master of Science degree in Computer Science and is teaching a UNIX for software developers class at the University of Applied Sciences, Darmstadt, Germany.