Advanced systemd

for the embedded use-case

Jérémy Rosen

Introduction

- This is just a feature list.
- Based on the industrial/embedde use-case
- From the point of view of someone that knows and teaches systemd

... but watches as other use it.

This talk will sound a lot like advertisement.

Test were done on a minimal buildroot

- qemu_x86-64, glibc, udev/eudev
- No usefull software
- 9.4 Mo vs 17.4 Mo
- Boot time couldn't be measured

Plan

1. Headline features
2. Hidden gems
3. Features I usually disable

Mastering the daemon's environment

Systemd makes it easy to control/restrict your daemon's execution environment

- Environment variables
- Standard file descriptors
- User, groups, User namespaces
- Chroot, bind mounts, filesystem masking, etc.
- Scheduler configuration, control groups
- Device/Network access
- Capabilities, Syscall filtering
- SELinux/Smack aware
- Security analysis tool.

Daemons don't need to set their own environment.
This can be done by the integrator...
And checked system-wide.

A note on systemd and security

Systemd secures the system, not the applications

- Minimize application privileges
- Ensure applications do not perform forbidden action
- Control communication channels between applications.

Systemd does not perform any security check itself

- Systemd configures the kernel security mechanisms.
- All mechanisms are configurable using the command line…
  But good luck with that.

Systemd automates frequent cases, which decreases the risk of errors

Mastering the daemon's lifecycle

systemd makes sure your daemon starts, runs and shuts down correctly

- Robust startup logic with timeouts, readiness detection, pre-start scripts, post-start scripts
- Software watchdog with a single API call.
- Robust cleanup, including IPC, post-exit scripts, dependency aware
- Configurable restarts, including grace-period and burst protection
- Well defined dependency/ordering rules
  - Just remember it's a partial order, not a strict order

Writing a bullet-proof startup script is hard.
With systemd, you don't even need to fork anymore.

Boot-related features

- Standardized boot-blessing
  - Easy to add your own tests in a OTA/Distro neutral way
  - Easily to integrate into custom OTA update systems
- Multiple boot targets
  - Production/Developement/Factory-test modes
  - Able to switch mode on a live system
- Boot-time analysis tool
  - No guessing anymore…
- Generators
  - Hardware-based boot targets (GPIO)
  - Easily convert XML config files into system configuration

Why does systemd boot faster

systemd boots more efficiently for various reasons

Paralelization Many services are booted in parallel, saturating both CPU and disks

Socket-based dependencies Data-providers can be started before data-consumers

Less services On demand startup means some daemons are not started at all

Less processes Systemd sets the environment itself.
No shells, no subshells, much less commands.

- 72 vs 155 for the pid of the first shell.

Plan

1. Headline features
2. Hidden gems
3. Features I usually disable

Journald

The one everybody loves to hate

- Exhaustive data collection
  - including stdouts, kernel and containers
- Exhaustive metadata collection
  - Including reliable timestamps, boot ID and process information
- Exhaustive API
  - Poll aware, with custom filters.
  - Binary data in the journal
- Network protocol
  - HTTPS Based, push and pull protocols
  - Integrated web-server for visualisation
- File rotation/suppression
  - Handle all the cases you can throw at it.

Logs are a hard problem, especially for isolated systems.
Journald is a solid brick on which to build.

Dbus piloting

Everything `systemctl`[1] can do, you can do via dbus
Any info `systemctl` can get, you can get via dbus (including dbus signals)

- system monitoring applications : just monitor the unit state
- on demand restart of services : just trigger a unit-state change
- dynamically change unit properties, including cgroup settings

Embedded applications need to interact with the system.
All embedded applications need to use Dbus anyway.

systemd's Dbus API is exhaustive and well documented

1. And other systemd command-line tools

Filesystem/partition management

- systemd can guess what partion goes where (GPT based)
- systemd can create missing partition (systemd-parted)
- systemd can format blank partitions
- systemd can populate empty directories

`fstab` is complicated to handle.

systemd makes it easy for a system to "fill up the empty space" on first boot

Portable services

Portable services are applications packaged in an image

- Easily buildable with buildroot/Yocto
- Single file to install/upgrade/remove
- Contain their own dependencies and configuration
- Integrated in the host system (dbus, journal, dependencies)

A poor man's packaging system that fits perfectly the embedded philosophy

Plan

Features for non-embedde use-cases

- **networkd** Engineered for the datacenter use-case.
- **logind, homed, per-user systemd** Only usefull for human users
- **nspawn** very little need for containers in the embedded world
- **systemd-boot** Only usefull on EFI systems
- **systemd in initrd** our initrd are usually too trivial

Conclusion

Why I use systemd as much as possible on embedded systems

- Writing a daemon is easy
- Mastering the environment is easy
- Securing a daemon is easy

- Interacting with the system is easy
- Understanding system interactions is easy
- Debugging the system is easy

For embedded systems, learning systemd is definitely worth your time.