# demystifying systemd for embedded systems

OpenIoT & ELC Europe 2016

ProFUSION
embedded systems

# Agenda

- Who am I?
- Embedded Systems?
- Background
- Systemd for Embedded Systems Myths
- Baseline
- Scaling Up
- Super-tiny Systems

# Who am I?

Gustavo Sverzut Barbieri
Computer Engineer
ProFUSION embedded systems

- Brazilian
- Software Developer since 9yo
- Working with Embedded since 2005
- Software development services
- Passionate about efficiency
- Fast boot enthusiast
- Hacked many init systems
- Doing systemd since it was public

# Embedded Systems?

ProFUSION
embedded systems

# Embedded Systems?

- Underpowered hardware
- Low memory
- Simple applications
- Single purpose
- Long development cycles
- Long deployment

# Embedded Systems?

- Underpowered hardware
- Low memory
- Simple applications
- Single purpose
- Long development cycles
- Long deployment

?

- Medical Equipment is beefy
- Smartphones are multi purpose and far from simple
- IoT expects faster cycles than Smartphones

# Embedded Systems?

- Underpowered hardware
- Low memory
- Simple applications
- Single purpose
- Long development cycles
- Long deployment

?

- Medical Equipment is beefy
- Smartphones are multi purpose and far from simple
- IoT expects faster cycles than Smartphones

it’s not a server or a laptop/desktop

# Embedded Systems in this talk

- runs regular GNU/Linux
- more than one persistent process running
- reasonable hardware

# Background

ProFUSION
embedded systems

# Background

- Recurrent requests for efficient boot
- Proper babysitting various kinds of processes is not trivial
- Security concerns raise need for proper isolation
- Growing awareness that systems are dynamic

# Background: Ostro Project

- Yocto Project based OS for Internet of Things (IoT)
- Pre-built
- Pre-configured
- Pre-secured

https://ostroproject.org/

# Background: Ostro Project is Pre-Built

- IoT and traditional Embedded Systems scopes are too broad
- One choice that nicely covers a wide spectrum is essential
- Time to market and quick development cycles over manual fine tuning

# Background: Ostro Project is Pre-Configured

- Stateless is important
- Dynamic behavior is essential
- Uniform file format helps a lot
- Drop-in configuration fragments
- Well documented configuration files

# Background: Ostro Project is Pre-Secured

- Least privilege rule for services is essential
- Namespaces are useful
- Multi-purpose systems based on 3rd party software benefit from containers

# Background: Ostro Project

Possibilities:

- systemd
- upstart
- openrc
- sysvinit
- busybox / toybox

# Systemd for Embedded Systems Myths

ProFUSION embedded systems

# Systemd for Embedded Systems Myths

- too big
- too complex
- uses DBus and I don’t need XML
- is done by Lennart and he did PulseAudio, will break my system

# Baseline

what does a
**minimal systemd**
looks like?

Most people get GIT or a pre-built package and are scared by the amount of files and the resulting size.

- 3M /usr/bin
- 15M /usr/lib

Is ~18M the baseline?

How to compare apples-to-apples?

* x86_64bits using glibc

# Baseline considerations on /usr/bin

- *ctl, systemd-{escape,path}: 648K of useful tools
- systemd-{analyze,cgls,cgtop,delta}: 1.1M of useful debug tool
- systemd-{ask-password,tty-ask-password}: should be done in your application
- systemd-sysusers is 44K… but shadow is 3M!
- udevadm and systemd-hwdb are 512K
- ...

All useful but not required or provided by competition, apples-to-apples…

**HINT:** to boot a system you need none of these if you remove the ".service" that may use them.

# Baseline considerations on /usr/lib

- libsystemd.so 548K, systemd/libsystemd-shared.so 2.1M, systemd/systemd 1.1M
- 6.9M udev (libudev.so 128K, udev/ 5.8M, systemd/systemd-udevd 452K...)
- libnss_*.so: 904K of optional improvements and convenience for name server
- security/pam_systemd.so 276K for PAM
- ...

# Baseline: step 1 - easy diet

- Compiled with -Os (previous numbers were -O2)
- Disabled all features listed by ./configure –help
- 7.4 M of systemd software (previously 18M)
- still lots of /usr/bin/ utils that could be removed (2M)
- udev (1.2M) and journal (104K) still present

# Baseline: step 2 - manual inspection

- Based on step 1 - easy-diet (7.4M of systemd files)
- Manually removing useful but not essential (./initramfs.sh): 5.4M
- No journal: 5.0M
- No journal, no udev: 3.9M

**NOTE:** timers, socket activation, process babysitting, service dependencies, namespaces, capabilities… all there!

# Baseline: what about the kernel?

| Build | Size | Comments |
|---|---:|---|
| x86_64_defconfig | 6.3M | Recommended config for 64-bits x86 |
| minimal | 668K | allnoconfig<br>+ printk + tty + /proc + /sys + /dev + serial |
| systemd | 1256K<br>+88% | minimal<br>+ systemd/README (IPv6, SECCOMP, Namespaces…) |
| systemd-minimal | 820K<br>+25% | minimal<br>+ systemd/README essentials (no network, block devices…) |

# Scaling Up

You know systemd scales up, but how other solutions do?

How to scale up busybox?

# Scaling Up Busybox

| | |
|---|---|
| Journal/Log | klogd and syslogd (builtins) or rsyslog |
| Service babysit and restart | inittab and inetd (builtins) + shell script |
| Networking<br>systemd-networkd | udhcpc and udhcpc6 (builtins) + shell script |
| Dynamic Name Resolver<br>systemd-resolved | Shell script |
| Hotplug | mdev (builtin) + shell script |
| Automount | mdev (builtin) + shell script |
| Module loading | mdev (builtin) + shell script |

# Scaling Up Busybox

| | |
|---|---|
| System Users | adduser and addgroup (builtins) + shell script |
| Locale Setup | Shell script |
| Boot loader | Shell script |
| Socket Activation | Inetd (builtin) |
| Timers | crond (builtin) |
| Cleanup<br>systemd-tmpfiles | Shell script |
| Containers<br>systemd-nspawn | Not covered |

# Scaling Up Busybox

- Only basic blocks are provided
- User is left with the task to glue with shell script
- Based on traditional tools file formats – all different
- Very simple functionality

Busybox focus on disk footprint…
…so you can “focus” on doing everything on your own.

# Super-tiny Systems

Baseline is too big?
Want to go very small?

Busybox / Toybox are cumbersome, could we have some systemd-like utility that is small?

# Super-tiny systems

Talking to Marcel Holtmann he shared his view:

> Really constrained embedded systems shouldn't even have userspace! They should be a single binary that does everything… Statically linked PID1 applications! Built as initramfs inside the kernel, signed and handled as a single entity.
>
> I'm using that to test BlueZ, you should try that.

This drove the linux-micro implementation of Soletta Project, a framework for making IoT devices which provides an API to the whole system: network, sensors, actuators and… system init!

https://github.com/solettaproject/soletta

# Soletta Project

- Developed primarily on GNU/Linux with systemd
- Port to various Small OSes (MCU-class), such as RIoT, Contiki and Zephyr
- Linux-micro port allows systemd-like behavior as PID1
- Mounts filesystems, including automount and fstab reading
- Setups hostname and networking (IPv6 autoconfig)
- Watchdog
- Module autoloading using kmod
- Applies sysctl
- Spawns and babysit dbus-daemon and bluetoothd
- Configures machine-id
- Spawns console for debug

https://github.com/solettaproject/soletta

# Soletta Project - Linux-Micro

- no busybox, no shell, no scripts
- statically linked binaries using musl-libc
- network-up and watchdog modules
- Flow-Based-Programming (FBP) runtime with:
  GPIO
  Timer and
  OpenInterConnect (OIC - now OCF): ~400Kb total userspace

# Thank You!

# Questions?

Gustavo Sverzut Barbieri
<barbieri@profusion.mobi>

scripts available at:
https://github.com/profusion/demystifying-systemd-for-embedded-systems