How to keep your embedded Linux up and running?

Krzysztof Opasiak

SAMSUNG
Samsung R&D Institute Poland

FOSDEM '18
Brussels 3 & 4 February 2018
Agenda

What’s the problem?

How do server guys do this?

How do we do this?

Summary

Q & A
What’s the problem?
California
Houses in California
Drought
So why not just water the lawn?

- It’s costly
- It takes some time
- You cannot leave it unattended
Let’s paint it green!
Your product

- UI
- Performance
- Reliability
Your product

- UI
- Performance
- Reliability

User Experience
Our case

- Minimal Tizen OS version
- Images customizable via web server
- Dedicated for small IoT devices (Artik, RPI)
- A base for many different products
- Anyone can say “My product runs TizenOS”
Your own code/Good open source code

- Code Review
- Tests
- Continuous Integration
- Static analysis

It's still imperfect! …but please remember to do this
Your own code/Good open source code

• Code Review
• Tests
• Continuous Integration
• Static analysis

It’s still imperfect!
Your own code/Good open source code

- Code Review
- Tests
- Continuous Integration
- Static analysis

It’s still imperfect!
…but please remember to do this
Tons of foreign code

• Has it been reviewed properly?
• Has it been well tested?
• Has CI practices been used?
• Has static analysis been used?
Tons of foreign code

- Has it been reviewed properly?
- Has it been well tested?
- Has CI practices been used?
- Has static analysis been used?

No one knows:(

FOSDEM 2018 | Brussels 3 & 4 February 2018
Typical problems

- Memory leaks
- FD leaks
- Bugs (service failures)
- Boot loops
- Other which we don’t know now (extensibility required)
How to fix them?

• Service restarting
• Fix scripts
• Recovery mode
• Report to developer
• Other methods which we don’t know now (extensibility required)
How do server guys do this?
systemd

Restart=

Restart the service based on exit method:

<table>
<thead>
<tr>
<th>Restart settings/Exit causes</th>
<th>no always</th>
<th>on-success</th>
<th>on-failure</th>
<th>on-abnormal</th>
<th>on-abort</th>
<th>on-watchdog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clean exit code or signal</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unclean exit code</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unclean signal</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Timeout</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Watchdog</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

OnFailure=

A space-separated list of one or more units that are activated when this unit enters the "failed" state.
Nagios

- Nagios Core
  - Scheduler
  - Web Interface
- Plugins (checkers)
- Passive checks
- Event Handlers
  - Shell scripts
  - Lack of global state
- Heavy
Icinga

- Fork of Nagios
- More modular design
- Mobile client
- Quite similar functionality
- Heavy
Zabbix

- Built from scratch
- Only single DB deployment
- Graphs out of the box
- Events
  - Trigger events
  - Discovery events
  - Auto registration events
  - Internal Events
- Rules engine
- Heavy
In general

- Web interface
- Periodic checks
- Shell scripts
- Some passive check also
- Dependencies
- Delays

They are all…

Web Scale!
Couldn’t we just fit this into YOUR pockets?

- We don’t want central decision server (Less we know, the better we sleep)
- Focus on passive checks (power consumption)
- Single machine monitoring
- No Web interface
- Low delay
- Light even with dependencies
How do we do this?
faultd architecture
Listeners

- **systemd listener**
  - listen dbus notification from systemd
  - uses private bus
  - reports suitable event when some service failed
- **audit listener**
  - Every service declares max resource usage
  - Limits are enforced using rlimits
  - Audit syscall is used to notify about reaching the limit (-EMFILE for example)
  - There may be more service failures:(


There is no free lunch

- **audit**
  - Measured overhead for 40 000 open() syscall:
    - 33% for cold file
    - 45% for hot file

- **rlimit-events**
  - RFC posted on LKML
  - Measured overhead for 40 000 open() calls:
    - 5.6% for hot file
    - 1.6% for cold file
Decision Makers

• VIP process handler
• Standard recovery
  • N times recover the service
  • M times reboot the platform
  • Enter recovery
• Resource violation
Actions

- **Recover service**
  - Run recovery unit (if defined)
  - Restart service

- **Restart service**

- **Reboot**
  - Forced reboot
  - Reboot using systemd
  - Reboot using deviced (tizen specific)

- **Reboot to recovery**
  - Reboot with param
Database

- Every event that goes through the core is stored in database
- This gives us a nice chain:
  - trigger
  - decision
  - action
- Initially we’ve chosen EJDB
- Now we are switching to SQLite
faultd

Listeners
- systemd
- audit

Core

Actions
- Recover service
- Restart service
- Reboot
- Reboot recovery

Decision Makers
- VIP handler
- Standard
- Resource violation

Database
- SQLite
- EJDB

DB

Userspace

Kernel

Audit

systemd

deviced
Summary
Summary

• Server monitoring tools are useful
• Unfortunately too big for our devices
• Audit syscall is not free
• EJDB is fast but overweight in terms of storage:(
• faultd is very extensible so try it
• Maybe it fits also your needs
Thank you!

Krzysztof Opasiak
Samsung R&D Institute Poland

+48 605 125 174
k.opasiak@samsung.com