Introducing serel

Revision History

Revision 1.0 21 May 2002 Revised by: LM
Initial revision
Revision 1.1 11 Aug 2002 Revised by: LM
Minor editing and clarification

serel's primary purpose is to help general-purpose operating systems boot faster. This document provides an overview of how serel works.

Table of Contents

Introduction

serel provides synchronisation primitives to **init** [1] that allow a computer's services to start and run in parallel. This reduces boot time because service startup is often the most time-consuming phase of the boot process, and because dependencies between services seem to be relatively rare.

serel is designed to be useful on a wide array of operating system platforms, even though there is considerable diversity in the way that operating systems start services [me00]. Aiming to support diversity, serel is designed so as not to be responsible for starting services. **Init** starts services; platforms are free to use the serel framework in diverse ways.

In order for serel to be usable as early as possible within user-space, the implementation does not depend on any services started by **init**.

Table 1. Phases of boot

Phase	Description
Bios	The bios locates and initialises hardware. A boot loader may run. Control passes to the kernel.
Kernel	The kernel starts and passes control to **init** in user space.
User-space - "basic" services	The "basic" services may include: filesystem check and mount , load and initialise device drivers, and initialise system time. .
User-space - "optional" services	The "optional" services may include: logging, networking, mail, network filesystems, web server, the windowing system etc.

The name "serel" derives from "SErvices and RELationships". The name hints at the core idea - that explicit service relationships allows operating systems to boot faster.

serel is open-source software, licensed under the GNU General Public Licence, Version 2 (http://www.gnu.org/licenses/gpl.html).

More information on serel is available at http://www.fastboot.org.

Dependencies

When services start sequentially, dependencies are implicit in the order of startup. Parallelism benefits from explicit dependencies.

serel allows dependencies to be expressed as:

- xml - serialisation of the W3C's Resource Description Format [wc00]
- special comments inside boot scripts, as per the Linux Standard Base Specification [lsb]
- synchronisation primitives usable from within boot scripts

Examples are shown in Appendix A.

These options offer comparable expressive power to each of the actors with an interest in dependencies: system administrators, OS vendors, and service developers.

Integrity of the dependencies is critical to a healthy boot. serel aggregates dependencies into graphs, which may correspond to a runlevel, a switch between runlevels, or other granularities such as "network" or "windows". Graphs are tested for integrity prior to committing to parallel boot. [2] [3]

It is not assumed that dependencies for all services are known at boot time. Best-effort handling of "unknown" services involves starting such services after all parallel services have finished and retaining the order of startup from the non-serel boot. Other approaches are possible.

Expressing Dependencies

Dependencies expressed via xml and boot script comments are enforced before a script executes. In addition, scripts can call the synchronisation primitives directly. This allows scripts to get work done before blocking, resulting in superior run-time performance. [4]

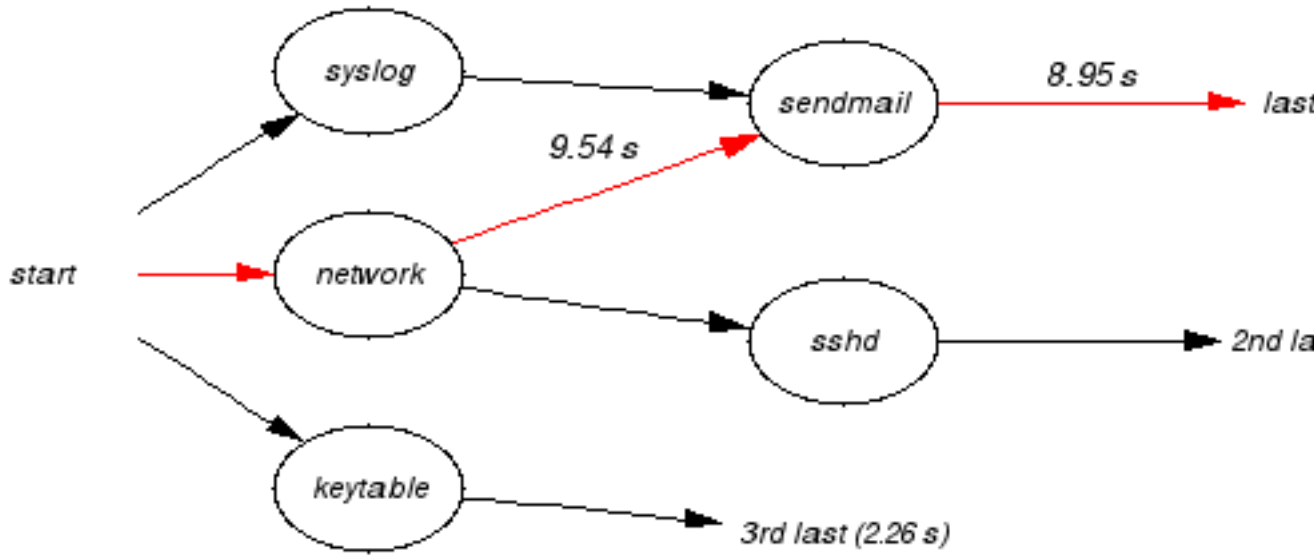
In choosing whether to express dependencies via xml and boot script comments or make direct calls to the synchronisation primitives, developers face a tradeoff: improved performance vs. increased maintenance costs. For resource-constrained platforms such as embedded systems, serel can be rebuilt so as to entirely exclude the use of xml or boot-script infrastructure, resulting in a daemon with a reduced memory footprint.

Dependencies expressed via xml and boot scripts are compiled into a database in order to optimise run-time queries. This database only gets rebuilt when something changes, which explains why second and subsequent boots can be faster than the first.

Visualisation

When services start sequentially, responsibility for slow boot times is shared among many services. When services start in parallel, boot time is a function of those few services that lie on the critical path.

Example 1. Visualisation example



The image above shows services, relationships, timings, and the critical path. The image is a visualisation (http://www.fastboot.org/visual.html) of the log file shown in Appendix B.

The three leftmost services have no dependencies. Once started, they run to completion in parallel.

sendmail blocks until both *syslog* and *network* have completed.

network ran for 9.54 seconds, *sendmail* ran 8.95 seconds, not counting the time it spent waiting for *network* and *syslog* to finish.

Red edges show the critical path of *network* followed by *sendmail*, running for a total of 18.49 seconds.

Can this system boot faster?

For this system to boot faster, either dependencies along the critical path must be untangled, or the individual services must be made to boot faster. Removing *sendmail* altogether would change the critical path to *network* followed by *sshd*, reducing boot time by about 5.8 seconds.

Visualisation credits

serel visualisation was inspired by the W3C RDF validation service (http://www.w3.org/RDF/Validator/). Analysis

of the RDF is built on top of the Redland (http://www.redland.opensource.ac.uk/) platform. The image is generated by AT+T's excellent GraphViz (http://www.research.att.com/sw/tools/graphviz/).

How fast is it?

serel 0.3.2 has been measured as reducing the boot time of the last phase of boot by as much as 38%. The reduction in boot time due to parallelism is highly dependent upon configuration, the smallest reduction measured to date is 8%.

The patches for `rc` under Red Hat Linux and Debian GNU/Linux have a similar function. When booting to a multiuser runlevel, `rc` starts the serel daemon, confirms the integrity of the dependencies, and runs the S??* scripts in parallel. The default configuration generates the visualisation log file whenever the history database changes.

The present serel implementation aims for robust delivery of a few core features, and leaves ample scope for future boot time improvement.

References

[go00] Gooch, Richard. *Linux Boot Scripts*, http://www.atnf.csiro.au/people/rgooch/linux/boot-scripts/index.html

[lsb] Linux Standard Base Specification, Section 19. System Initialization, Comment conventions for init scripts, http://www.linuxbase.org/spec/refspecs/LSB_1.1.0/gLSB/initscrcomconv.html

[me00] Mewburn, Luke. *The Design and Implementation of the NetBSD rc.d system*, http://www.mewburn.net/luke/papers/rc.d.pdf

[sv00] Sanchez, Wilfredo, and Van Vechten, Kevin. *SystemStarter and the Mac OS/X Startup Process*, http://cory.eecs.berkeley.edu/~kevinvv/SystemStarter.html

[wc00] *Resource Description Framework*, W3C , http://www.w3.org/RDF

A. Example dependencies

Here are three alternative ways of expressing "*sendmail* depends on *network* and *syslog*".

Example A-1. expressing dependencies via xml/rdf

```
<service rdf:ID="sendmail" >
  <edges>
    <rdf:Bag>
      <rdf:li rdf:resource="#edge-network-sendmail" />
      <rdf:li rdf:resource="#edge-syslog-sendmail" />
```

```
    </rdf:Bag>
  </edges>
</service>

<edge rdf:ID="edge-network-sendmail" >
  <source rdf:resource="#network"/>
  <target rdf:resource="#sendmail"/>
</edge>

<edge rdf:ID="edge-syslog-sendmail" >
  <source rdf:resource="#syslog"/>
  <target rdf:resource="#sendmail"/>
</edge>
```

Example A-2. special comments in boot scripts

```
#!/bin/bash
#
# sendmail      This shell script takes care of starting and stopping sendmail.
#
### BEGIN INIT INFO
# Provides: sendmail
# Required-Start: syslog network
### END INIT INFO
#
# ...
```

Example A-3. Direct use of synchronisation primitives

```
#!/bin/bash
#
# sendmail      This shell script takes care of starting and stopping sendmail.
# ...
#
if [ $1 = "start" ]; then
    serelc --provide sendmail
    serelc --need syslog
    serelc --need network
   ...
fi
...
```

B. Sample log file for visualisation

The image shown in Example 1 is generated from the following log file.

```
<?xml version="1.0" encoding="utf-8"?>

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:serel="http://www.fastboot.org/2002/03/serel-schema#"
  xmlns:graph="http://www.fastboot.org/2002/03/graph-schema#"
  xmlns="http://www.fastboot.org/2002/03/graph-schema#"
>
<graph rdf:ID="graph2" graph:directed="#true" serel:version="0.3.1"
                       serel:created="2002-04-25 15:30:00" >
  <nodes>
    <rdf:Bag>
      <rdf:li rdf:resource="#keytable" />
      <rdf:li rdf:resource="#sendmail" />
      <rdf:li rdf:resource="#sshd" />
      <rdf:li rdf:resource="#syslog" />
      <rdf:li rdf:resource="#network" />
    </rdf:Bag>
  </nodes>

  <edges>
    <rdf:Bag>
      <rdf:li rdf:resource="#edge-syslog-sendmail" />
      <rdf:li rdf:resource="#edge-network-sendmail" />
      <rdf:li rdf:resource="#edge-network-sshd" />
    </rdf:Bag>
  </edges>
</graph>

<node rdf:ID="keytable" graph:weight="2.26" />
<node rdf:ID="sendmail" graph:weight="8.95" />
<node rdf:ID="sshd" graph:weight="3.12" />
<node rdf:ID="syslog" graph:weight="1.46" />
<node rdf:ID="network" graph:weight="9.54" />

<edge rdf:ID="edge-syslog-sendmail" >
  <source rdf:resource="#syslog"/>
  <target rdf:resource="#sendmail"/>
</edge>
<edge rdf:ID="edge-network-sendmail" >
  <source rdf:resource="#network"/>
  <target rdf:resource="#sendmail"/>
</edge>
<edge rdf:ID="edge-network-sshd" >
```

```
  <source rdf:resource="#network"/>
  <target rdf:resource="#sshd"/>
</edge>

<rdf:Description about="graph2"
  serel:base_os_release="Red Hat Linux release 7.1 (Seawolf)"
  serel:package_owning_rc="initscripts-5.83-1"
/>
</rdf:RDF>
```

Notes

1. On many *nix systems, the first user-space process is named **init**. **init** (or a child like **rc**) is responsible for starting system services.
2. Early releases of serel distributed default dependencies by inserting special comments within boot scripts. This approach carries a number of undesirable properties, including the burden of maintaining dependencies scattered across dozens of files. serel now distributes default dependencies in a single xml configuration file.
3. Apple's OS/X 10 starts it's services in parallel [sv00]. Dependencies are expressed via xml configuration files, one per service.
4. See also: **simpleinit** [go00].