Improve the Debian Boot Process
Google Summer of Code 2006

State-of-the-Art in the Boot Process
First Deliverable

student: Carlos Villegas (Carlos.Villegas at nuim.ie)
mentor: Petter Reinholdtsen (pere at hungry.com)

June 20, 2006

1 Introduction

The boot process of unix-like operating systems has been based, until recently, on the init script system used in System 5 [20]. This process is commonly called sysvinit is dominated by the rc script which calls all the processes required for a determined runlevel one after the other in a prespecified order. The order given to this scripts is conservative as there is not information of what scripts are actually needed to run before. This whole process doesn't tend to use all the CPU nor disk access capacity as just one script run at the same time.

The Debian linux distribution uses currently the sysvinit process although several approaces tried by several developers have shown that its boot process can be faster. One good example is given by the results presented by Margarita Manterola at DebConf6 [6]. Distributions like Ubuntu seem to be already on their way to faster booting with Sun SMF self-heal services and parallel execution in NexentaOS [12] and SUSE Linux already implementing LSB-compliant parallel execution of init scripts [18]. Thus, the need to improve the current process has been identified and different approaches are being taken to solve it. Parallel processing is an option that has been considered but the dependencies between the programs is a constraint. With dependencies taken into account like by making init scripts LSB-compliant, parallel execution could be implemented in Debian. At the end of the day, a 60% faster booting time could be obtained according to Petter Reinholdtsen [17], which would make a big difference for high availability systems.

In order to do this, we aim at detecting *hotspots* and implementing the most promising ones. With this first deliverable, the first "*hotspots*" are identified" together with the current state-of-the-art in the boot process. Afterwards, the

hotspots will be iteratively applied with consideration on LSB (Linux Standard Base) compliance and backwards compatibility.

Furthermore, *initscripts-ng.alioth.debian.org/soc2006-bootsystem* is the project webpage published to promote the project and track its development. Additionally, a blog at *bootdebian.blogspot.com/* to provide information of the day-to-day progress. Besides, discussionis held through the *alioth initscripts-ng* mailing list [2] and the debian's IRC in the sysvinit channel [7].

In the first section of this document, we outline some of the identified *hotspots*. They are ordered by their possible effect to the decrease the boot time. The second section has as an objective to describe the state-of-the art in the boot process in unix-like distributions. They ordered by the program/algorithm used with subsections for its implementation in particular distributions.

2 Hotspots

The following hotspots have been identified so far:

1. **Use dash or internal functions in the init scripts** – The use of dash can reduce the boot time around 6 seconds[6] as it is a smaller program than bash. Rewritting slow shell scripts to use internal function instead of external programs is another option.
2. **LSB-compliance and reordering bootscripts**– LSB compliance in the init scripts can be used to reorder the scripts as shown in the SUSE boot with insserv [18]. Lintian rules should be considered as well during the implementation. Just by reordering some processes around 2 seconds could be gained[6].
3. **Parallel execution of boot scripts** – currently can be activated setting CONCURRENCY=startpar in /etc/default/rcS but there is still the need to efficiently reorganize the scripts;
4. **Preloading** – having parts of the program already in memory can improve execution speed. Nevertheless, with parallel execution, preloading should be adjusted not to hinder other programs memory requirements [19] (solved with dynamic preloading – readahead),
5. **Set up the hardware clock in the background** – around 6 seconds could be gained[6].
6. **Set up the network in the background** – around 2 seconds could be gained[6].
7. **Remove depmod from the boot process** – around 2 seconds could be gained[6].
8. **Improve CPU use when starting the desktop manager**,
9. **Make the boot less verbose**.

3 State-of-the-Art in boot processing

Several linux distributions are trying to make their boot process faster and they have taken different approaches. Additionally, there is a group of enthusiast that make code to improve the boot process. In this section, we'll mention the different approaches found for improving the boot process and consider the implementation in some unix-like distributions.

3.1 sysvinit (Debian/Fedora)

Many unix-like distributions use the System V init (sysvinit) script architecture as it is quite powerful and adaptable due to is modular design. It works mostly well with the software packages concept although it can be messy to keep the execution of the scripts on the right order[20].

It is composed of three modules:

/sbin/init – the System V init,

/sbin/rc – with the symbolic link farms in */etc/rc?.d/** and

/etc/init.d/* – the System V-like system init scripts pool.

Besides the sysvinit concept has manages different system profiles, which are called runlevels. The runlevels are:

0 – system halt,

1 – single-user mode,

2-5 – normal modes,

6 – reboot system and

S – system startup.

The runlevel S is only used at startup and then it switches to another runlevel from 1 to 5. The use of runlevels 2 to 5 varie from vendor to vendor, Debian using runlevel 2 as default[14]. On the other hand, SUSE linux makes a difference between runlevels being:

2 – local multiuser mode without remote network (NFS, etc),

3 – full multiuser mode with network,

4 – not used and

5 – full multiuser mode with network and X display manager[18].

Debian

Debian uses sysvinit for the boot process with *update-rc.d* to manipulate the symbolic link farm. Besides, the program *invoke-rc.d* may be used when a script has to be run outside the bootprocess, e.g. package installation or removal performed by maintainer scripts. This avoids services to be started just when they should (like when they are not in the appropiate runlevel).

Nevertheless, *update-rc.d* cannot perform incremental changes such that, in order to add or remove a script in a certain runlevel, it is necessary to specify the complete setup.

Fedora Core 5 and future plans

Fedora seems to have a standard System V like init process but with added LSB support. The latter is included via a program that can parse LSB standard headers for start and stop levels called chkconfig. These dependencies are used add them in the right order in the /etc/rc<X>.d directories and maintain them, although priorities are not recomputed if other dependencies are added or changed. [3]. The chkconfig command was inspired in the one present in the IRIX operating system [1] and, in constrast with Debian's *update-rc.d*, it can be used as well as a runlevel editor.

There are plans to improve Fedora boot process by implementing a new one based on D-BUS. The objectives are to:

- proper runtime dependency support,
- full backwards compatibility,
- full LSB support,
- service exposure via D-BUS and
- support for respawning services.

3.2 readahead (Ubuntu Dapper/ Knoppix)

readahead is a program used to improve the boot time when the RAM memory is larger than 256 Mb [8] based on reading in advance one or more pages of a file within a page cache [9]. It is used in some distributions like Ubuntu Dapper and Knoppix.

3.3 startpar/insserv (SUSE 10)

The combination of startpar and insserv intends to provide parallel execution during boot time by considering the dependencies. Startpar task is to start

runlevel scripts in parallel [10] while serializing the output. Insserv is used to reorder the init scripts by considering comment headers on the scripts (LSB-compliant) and calculating the dependencies between the scripts in the specified runlevel [5]. It uses the concept of System Facilities (SF) and associates dependencies to each one of them. The SF in the script is followed in the same line by the SF or scripts it depends on. For example:

$remote_fs $local_fs nfs

indicates that the SF *remote_fs* requires the SF *local_fs* and the script *nfs*. It may be noticed that a SF name is preceded by a *$*. Innserv writes down a file that may be used with startpar to perform a parallel init script execution while considering the dependencies.

The SuSE boot concept

The SuSE distribution implements the *startpar/insserv* combination to provide parallel execution during the boot process. Besides, it also uses a preload script **which doesn't seem to be active**. A short description of the SuSE boot concept is presented next.

The SuSE boot concept seems to be LSB-compliant such that the init scripts have been moved to /etc/init.d. The boot process seems to be divided at least in two phases being:

1. from boot time and
2. after system startup.

At boot time, the first boot level master script is */etc/init.d/boot*. It has the task to initialise the system: like doing a filesystem check and running hardware init scripts specified in the */etc/init.d/boot.d* directory. Afterwards, the local commands are executed with the script */etc/init.d/boot.local*.

After the system startup, the master level script */etc/init.d/rc* takes control and starts the scripts for the specified run-level.

Besides, with the *differential link scheme*,switching between run-levels is considered. It uses the start and stop links of init scripts inside the run-level directories (*/etc/init.d/rc<X>.d*). With the *differential link scheme*, just those scripts that were not present in the previous run-level are started while only those that are not present in the new run-level are removed. As a result, repetitive script starts and stops are avoided.

If parallel execution is enabled, the init scripts are executed in parallel using startpar. In order to know which programs should be executed first, the insserv command creates some files with the dependencies before starting or stopping the related process. This information is used by startpar to execute the scripts in parallel while considering dependencies[11].

The default arguments for startpar in SuSE [10] at boot to

- (*-p 4*) have a degree 4 of parallelism,
- (*-t 30*) an individual buffer timeout of 30,
- (*-T 3*) a global buffer timeout of 3 and
- (*-M*) to have a *make*-like behaviour.

where the individual buffer timeout refers to the time for the buffer to be flushed, while the global timout will flush the buffer of the script with the oldest output. With the *make*-like behaviour, the appropiate dependency file will be used: boot, start or stop.

On the other hand, insserv in SuSE writes the dependency information taken from the scripts into the files: */etc/init.d/.depend.boot, /etc/init.d/.depend.start, /etc/init.d/.depend.stop* for the boot, start and stop dependencies, respectively [5]. Besides, different to other ordering scripts, the scripts do not require an extra flag to enable parallel execution [22].

3.4 pinit/prcsys(Mandriva)

Program to implement LSB compliance without mixing startup script output and not modifying much the current startup script. It retrieves the services that a determined init script enables and disables while considers if the program should be executed in parallel. It doesn't use the typical sysvinit SXX ordering and, in order to keep a clean screen output, it stocks temporarily the output in a file which is afterwards printed when the service startup is over [15]. It has been implemented in Mandriva Linux [23]

3.5 rcorder (NetBSD/FreeBSD)

NetBSD init script system has an advanced init script system that was later adopted by FreeBSD as well. Its */sbin/init* is mostly the same as the one inherited from 4.4BSD, and all the intelligence of the init script system is in the */sbin/rcorder* script [20].

What *rcorder* does, is to print out dependency ordering of a set of interdependent files. It is specifically used for the init scripts by reading special keywords (inserted inside shell comments) indicating their dependencies [13]. This program will be run every boot time by the */etc/rc* script twice to consider the scripts in filesystems mounted at a later stage like the ones in network filesystems [16]. Thus, insted of calculating dependencies just when a new program is installed like other scripts, rcorder does the ordering every boot time.

Moreover, rcorder dependency mechanism make it possible for third-party scripts to be installed and added to the dependency tree at the appropiate start-up point without difficulty [21]

On the other side, *rcorder* approach doesn't make it easy to implement parallel execution such that the NetBSD init script system doesn't even attempt to do that. Although the information for parallel execution is there a different approach is required for parallel execution [20].

3.6 file-rc

This approach is similar to the standard sysvinit although, instead of using a symbolic link farm (*/etc/rc<X>.d/*) for each runlevel, it uses a single configuration file [23]. This file is in an easy-to-parse, tabular format. [20]

3.7 Solaris SMF (NexentaOS)

SMF stands for Service Management Framework and consists of a daemon that takes care of managing the services (starting/stopping) while leaving *init* to take care of the other tasks of booting the system [23]. The services are described by an XML file and can include dependencies such that a service would bring up all other services it needs. Nevertheless, as it doesn't replacethe *init*, it requires that everything be modified to run in the foreground and not daemonise [23].

Moreover, it's licence is CDDL and not GPL-compatible.

3.8 Apple launchd

Apple's approach is to replace the *init* and uses XML configuration files for the services. It is not dependency based which means that services are started on demand and kept running as long as they are needed. This means that if a service requires another, it will wait until that other service starts.

The license is APSL and not GPL-compatible. It has even clauses that cause issues if you even read the source code [23]

3.9 init-ng

Full replacement of sysvinit tool created by Jimmy Wennlund and designed to increase the booting speed of unix-compatible systems by starting processes asynchronously. [4]

It has a plugin architecture such that almost all the functionality is provided by loadable *.so* modules. The plugins have features like restarting services should they fail, setting resource limits or communications over dbus [23].

3.10 serel

Leni Mayo's serel aims at reducing boot time by implementing a dependency based init script capable of parallel execution. For dependencies, it uses the need(8) concept (dinamically and statically) and should work out of the box with RedHat 7 systems. [20]

The project seems to have been abandoned by the author in 2002.

4 Conclusions and Future Work

The boot process can be faster and promising solutions have been explored by several groups. From the *hotspots* in Section 2 we will implement the most promising ones and reorder the hotspots. The results will be presented in the next deliverable together with a boot time benchmark of the debian releases from *woody* to *sid*. With a better perspective of the problem and possible solutions given by the brief state-of-the-art in Section 3 we should be in a better position to improve Debian's boot process.

References

[1] Chkconfig(8) man page in fedora core 5.

[2] Debian next generation initscripts project at alioth. http://alioth.debian.org/projects/initscripts-ng.

[3] Fedora analysis. http://fedoraproject.org/wiki/FCNewInit.

[4] initng – next generation init system. http://www.initng.org/wiki.

[5] insserv(8) manual. http://man-wiki.net/index.php/8:insserv.

[6] Marga's blog – parallel booting. www.marga.com.ar/blog/index.cgi/debian/Parallel_booting.html.

[7] #pkg-sysvinit at irc.debian.org.

[8] Readahead in knoppix. http://unit.aist.go.jp/itri/knoppix/readahead/index-en.html.

[9] Readahead(2) – linux man page. http://www.die.net/doc/linux/man/man2/readahead.2.html.

[10] startpar manual. www.math.ucla.edu/computing/docindex/sysvinit-man-25.html.

[11] (suse 10)/etc/init.d/readme – the suse boot concept.

[12] Ubuntu and smf.
http://www.calivia.com/blog/mike/solaris_smf_for_ubuntu_packages_on_opensolaris.

[13] Rcorder(8) in the freebsd system manager's manual. July 2000.
http://www.freebsd.org/cgi/man.cgi?query=rcorder.

[14] rcs(5) man page in debian administrator's manual. November 2003.

[15] Mandrake linux archives: cooker@mandrivalinux.org. October 2005.
http://archives.mandrivalinux.com/cooker/2005-10/msg00256.php.

[16] (freebsd/src/etc/rc script from freebsd. February 2006.
http://www.freebsd.org/cgi/cvsweb.cgi/src/etc/rc.

[17] Ideas for speeding up the debian boot process. January 2006.
http://lists.alioth.debian.org/pipermail/pkg-sysvinit-devel/2006-January/000542.html.

[18] Suse linux reference 10.1. April 2006.
http://ftp.opensuse.org/pub/opensuse/distribution/SL-10.1/inst-source/docu/en/reference_en.pdf.

[19] Eric Brasseur. Linux optimization. May 2005.
http://www.4p8.com/eric.brasseur/linux_optimization.html.

[20] Henrique de Moraes Holschuh. System init scripts and the debian o.s. In *3rd Debian Conference*, June 2002.
alioth.debian.org/docman/view.php/30730/38/debconf2-initscripts-bkg.pdf.

[21] Luke Mewburn. The design and implementation of the netbsd rc.d system. June 2001.
http://www.mewburn.net/luke/papers/rc.d.pdf.

[22] Petter Reinholdtsen. Ubuntu plans for init. February 2006.
http://lists.alioth.debian.org/pipermail/initscripts-ng-devel/2006-February/000251.html.

[23] Jane Weideman. Ubuntu plans for init.
https://wiki.ubuntu.com/ReplacementInit.