Improve the Debian Boot Process
Google Summer of Code 2006

Implementation of hotspots
Third Deliverable

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

September 25, 2006

1 Introduction

In this deliverable we intend to show the results of testing the remaining hotspots and to explore briefly the interactions between them.

Several approaches have been tried along the past two months of the Summer of Code together with some patches, scripts and bug reports. Now it is time to review the effect of the remaining hotspots and try them together to see the total time improvement in the boot time.

First, in section 2, we will ennumerate the different hotspots, present their testing procedure and their individual effect on boot time. Afterwards, in section 3, we will try them together in some combinatios and show the total time improvement.

Discussion on the project may be followed in the initscripts-ng-devel mailing list and the channel #pkg-sysvinit in *irc.debian.org*.

2 Individual Hotspots

As any changes resulting from this project may be implemented in Etch+, we've used the Debian unstable release – Sid –, to test the hotspots. According to the discussion in [6], we may consider two different metrics:

startup time - from the point of view of the user from turning on the computer until KDE becomes usable, and

sysvinit time - from the moment /sbin/init takes control of boot process until KDE becomes usable.

We will use the second metric because it is the one that can be influenced by modifying the sysvinit boot process and it is a subset of the first metric.

Order	*Hotspot*	Marga's time[2]	our time
1	dash	6	4
2	reorder	2	4
3	preloading	–	2
4	hwclock	6	2
5	parallel	–	2
6	depmod	2	2
7	network	2	0
8	discover	–	0

Table 1: *hotspot* time reduction

2.1 dash

Dash is a smaller shell and by using it instead of the default /textitbash we get a reduction of the boot time. The Ubuntu distribution already implements it by linking /bin/sh to /bin/dash [3]. Besides, during Debconf6, Margarita Manterola presented a time reduction of 6 seconds by using dash in Debiani [2]. In our previous deliverable 2 [6] we present results of up to 4 seconds reduction of the boot time. Therefore, using dash during the boot process seems to be the *hotspot* with the bigger influence.

Dash may be implemented during boot in two ways:

1. by linking /bin/sh to /bin/dash, and
2. by changing the init-scripts to use dash.

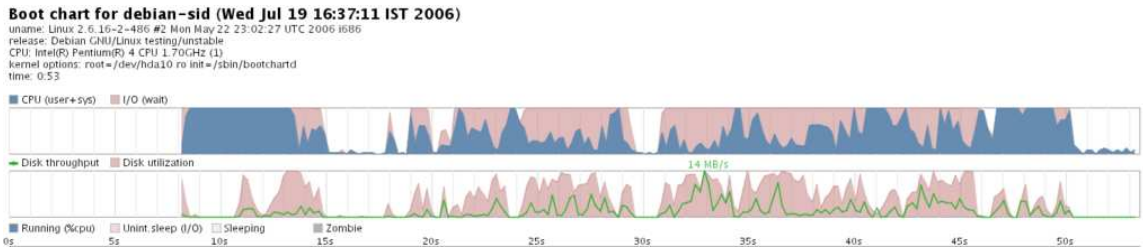

Figure 1: Bootchart of reference system

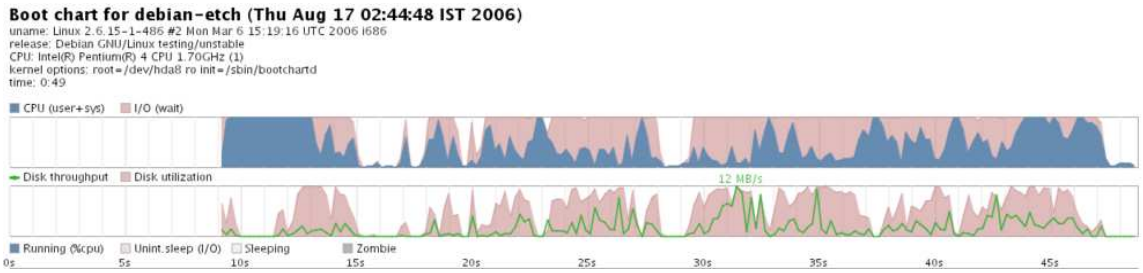

Figure 2: Bootchart for Dash as /bin/sh (partial)

The first approach caused a 4 second improvement while the second approach a 2 second improvement in our test system. The first approach is implemented as follows:

apt-get install dash
dpkg-divert –add /bin/sh
mv /bin/sh /bin/sh_old

cd /bin
ln -s dash sh

and removed as:

dpkg-divert –remove /bin/sh
rm /bin/sh
mv /bin/sh_old /bin/sh
apt-get remove –purge dash
apt-get install –reinstall bash

For the second approach, the test consisted of modifying the first line of the init scripts in /etc/init.d from #!/bin/sh to #!/bin/dash.

The first approach caused a 4 second improvement while the second approach a 2 second improvement in our test system.

2.2 Reordering the init-scripts

Normally, the init-scripts are executed one-by-one during the boot process according to the order in /etc/rcS.d and /etc/rc2.d. The order is important to satisfy runtime dependencies, i.e., to satisfy the requirements for an init script to run properly like *udev* requires *mountvirtfs*. Besides, the order has shown to be influential on the boot time.

Some distributions like SUSE have taken runtime dependencies seriously and implemented a system to keep the init scripts ordered[4]. In Debian, the script order still depends on the number assigned by the maintainer. This approach is prone to errors and normally requires somebody to file a bug report for them to get corrected.

Margarita Manterola already showed in Debconf6 a boot time improvement of 2 seconds by reordering the scripts manually[2]. These satisfactory results lead to consider this hotspot as important but the results presented in this deliverable made us consider it as one of the main *hotspots*.

There are mainly two ways to order the init-scripts:

1. trial-and-error based on experience and/or a deep understanding of each of the init-scripts,
2. using explicit run-time dependencies written by the experts on each init-script.

The first approach is an experimental optimization problem with a different solution for each system configuration and therefore not feasible in the long-run. The second approach requires the init-script maintainers to provide explicit run-time dependencies and would adapt to each possible system configuration. The latter has been already implemented in the SUSE distribution mainly through the insserv and update-rc programs [4].

The program insserv reorders the init script symbolic pools (/etc/rcX.d) and uses LSB-compliant headers in the scripts. Besides, it contains some run-time dependencies for some init scripts that don't provide them yet. Nevertheless, the current results are rather poor and /etc/rcS.d is not modified. Besides, in /etc/rc2.d many scripts from /etc/rcS.d are repeated.

Using insserv without modifications, the boot time increases by 2 seconds, i.e., 2 seconds slower. On the other hand, with a slightly different order and removing some repeated scripts, a 2 second faster boot time than the original is obtained. Furthermore, by deleting all scripts repeated from /etc/rcS.d, a total of 4 seconds time reduction is obtained.

...rcS.D symbolic farm...
S01glibc.sh

S02mountkernfs.sh
S03udev
S04mountdevsubfs.sh
S05bootlogd
S05keymap.sh
S10checkroot.sh
S18hwclockfirst.sh
S18ifupdown-clean
S20module-init-tools
S20modutils
S22hwclock.sh
S25libdevmapper1.02
S30checkfs.sh
S30procps.sh
S35mountall.sh
S36discover
S36mountall-bootclean.sh
S36mtab.sh
S36udev-mtab
S38pppd-dns
S39ifupdown
S40hostname.sh
S40networking
S43portmap
S45mountnfs.sh
S46mountnfs-bootclean.sh
S48console-screen.sh
S50alsa-utils
S55bootmisc.sh
S55urandom
S70nviboot
S70x11-common
S75sudo
S80installation-report
S99stop-bootlogd-single

...rcS.2 symbolic farm...
S08gdm
S08kdm
S08makedev
S08stopbootlogd
S09portmap
S10nfs-common
S11sysklogd
S12klogd
S13acpid
S13atd
S13cron
S13dbus
S13dirmgr
S13exim4
S13lpd
S13rmnologin
S14hotkey-setup
S19ssh
S20inetd

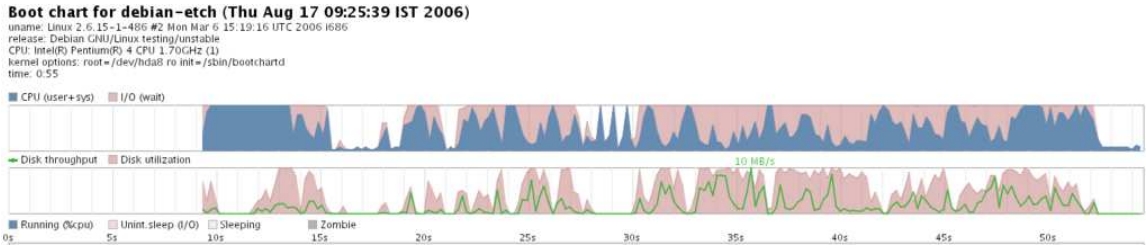

Figure 3: Bootchart for ordered scripts using insserv (partial)

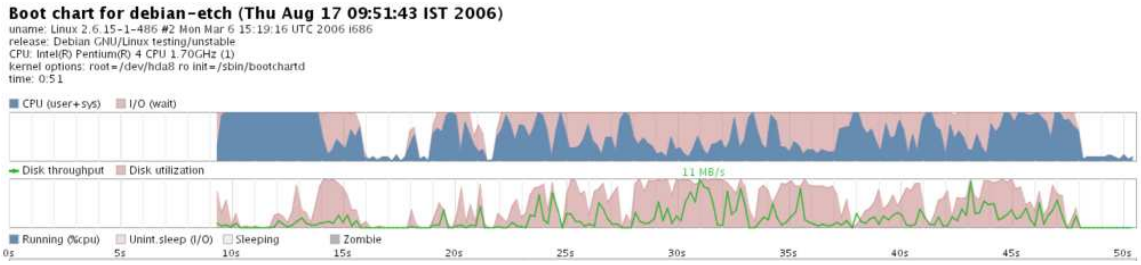

Figure 4: Bootchart for manual re-ordered scripts (partial)

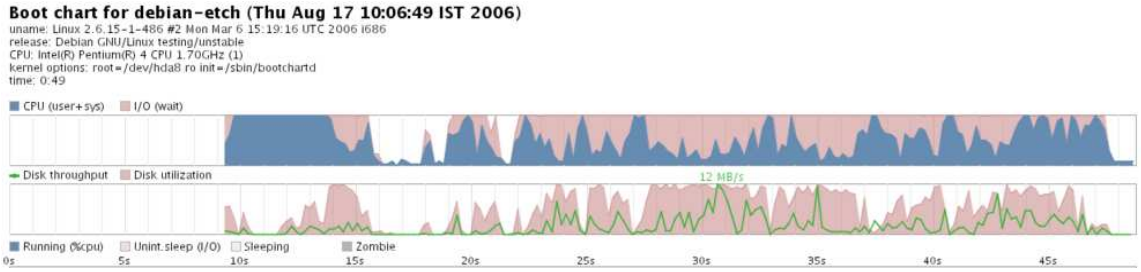

Figure 5: Bootchart for manual re-order scipts and no repetitions (partial)

We implement insserv as:

apt-get install insserv
update-bootsystem-insserv

and removed as:

update-bootsystem-insserv restore
apt-get remove –purge insserv

An approach towards using explicit run-time dependencies is provided by the Linux Standard Base(LSB) through a convention of the form:

BEGIN INIT INFO
 # Provides: x-display-manager gdm
 # Should-Start: console-screen
 # Required-Start: $local_fs $remote_fs
 # Required-Stop: $local_fs $remote_fs
 # Default-Start: 2 3 4 5
 # Default-Stop: S 0 1 6
 # Short-Description: GNOME Display Manager
 # Description: Debian init script for the GNOME Display Manager
END INIT INFO

In order to make it easier for Debian maintainters to provide init-scripts with LSB-compliance a maintainers guide was published as part of this project [1]. Besides, a patch has been provided for lintian to check LSB-compliance.

2.3 Preloading

There are different programs for preloading. For this project we've considered:

1. preload from Behdad Esfahbod and
2. Ubuntu's readahead.

There is a debian package of Behdad's preload. The latter is a result from the google summer of code 2005 (SoC2005) and claims to be a dynamic readahead program, i.e., it adjusts the files to be preloaded according to the files used during the previous boot.

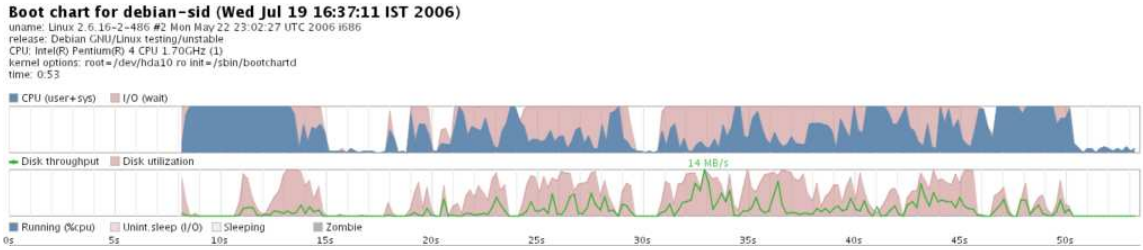

Figure 6: Bootchart for reference system

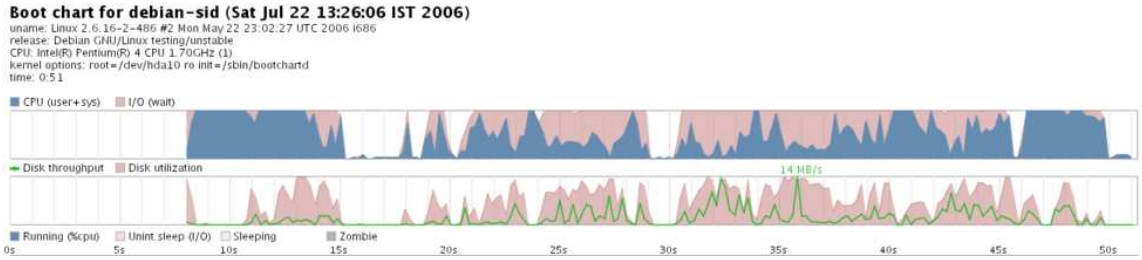

Figure 7: Bootchart for preload with default configuration

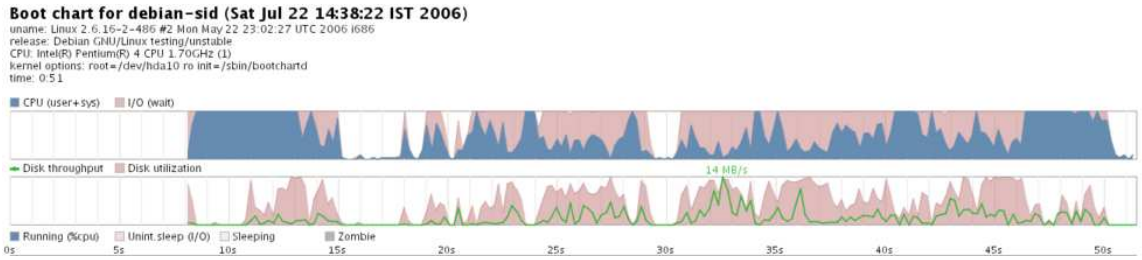

Figure 8: Bootchart for preload with modified configuration

Using preload 0.2, we had a 2 second improvement using the default configuration: mapprefix and exeprefix were set to empty such that all files would be accepted. The results were later compared with mapprefix being set to hte directories used by some of the scripts at boot time (found with the command "strace -f *init-script*"), adjusting the quantum time of preload (cycle parameter) and changing the position of the initscript to start just after hwclockfirst.sh. The results were the same: a 2 second time improvent.

The main change in preload.conf was: *mapprefix = /lib; /lib/lsb/; /lib/terminfo/x; /lib/tls; /usr/lib; /usr/lib/gconv; /usr/lib/locale*
exeprefix = !/usr/sbin;/usr;!/

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 [5] (possibly solved with dynamic preloading).

On the other hand, using Ubuntu's readahead we found no time improvement with version 1.0.1-2 and a 1 second longer boot time using version 0.20050517.0220-0. The reason for this may be that a much different readahead file list has to be found. The previous is not only difficult but would have to be repeated every time for each system (like with dynamic readahead).

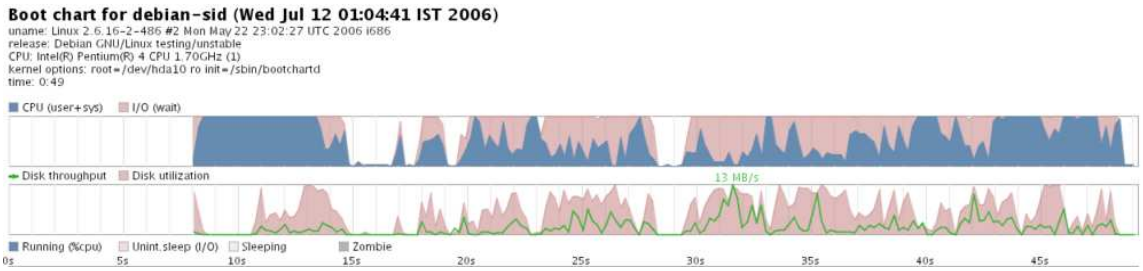

Figure 9: Bootchart for reference system

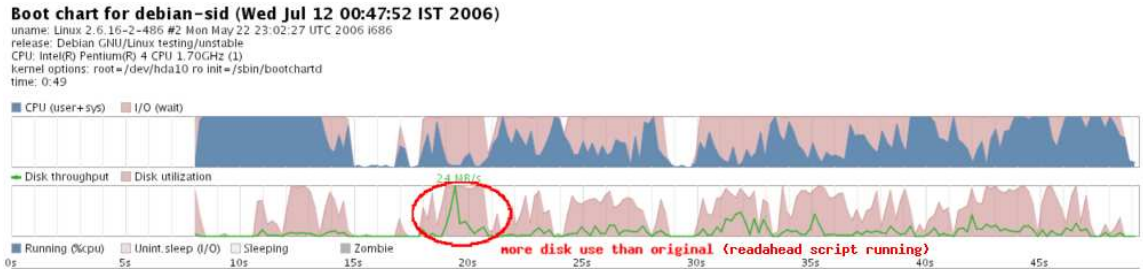

Figure 10: Bootchart for readahead 1.0.1-2

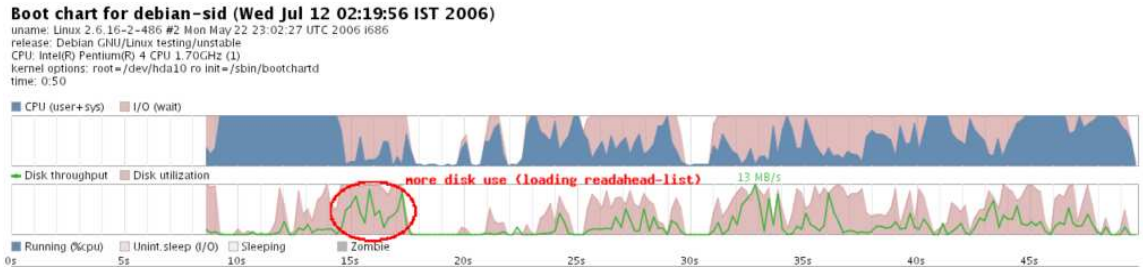

Figure 11: Bootchart for readahead 0.20050517.0220-0

After making changing the readahead list, we got a very small time reduction. Firstly, the script readahead-desktop was removed from the symbolic link pool (/etc/rcS.d) and the file containing the files to preload (/etc/readahead/boot) was adjusted to use some of the files required by kde (strace -f kdm). We use readahead in /etc/rcS.d position S42. The files included can be found in the appendix A.

2.4 hwclock

Setting up the hardware clock improves the boot time by setting it in the background. This approach was tested and presented in Debconf6 by Margarita Mangerola with a 6 second reduction[2]. During the project we've tried hwclock in the background with a 2 second time reduction [6]. Nevertheless, running hwclock on the background may cause problems with processes that can get confused if the time changes while they are running. Therefore, just hwclock.sh is set on the background while hwclockfirst.sh is executed as usual.

The changes on the hwclock.sh init-script consist of sending to the background by adding **&** at the end of every line containing */sbin/hwclock*.

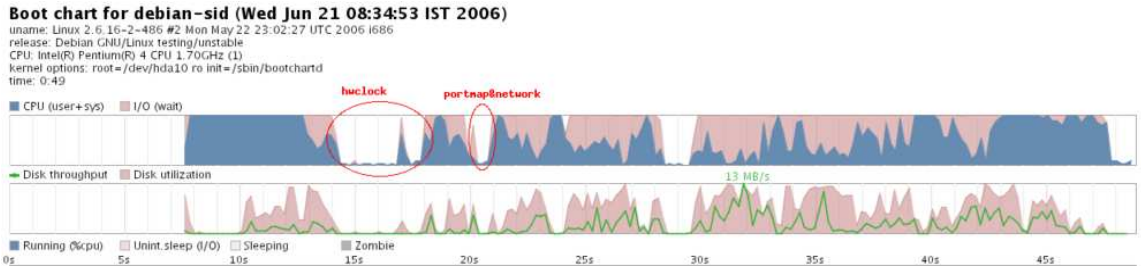

Figure 12: Bootchart of reference system

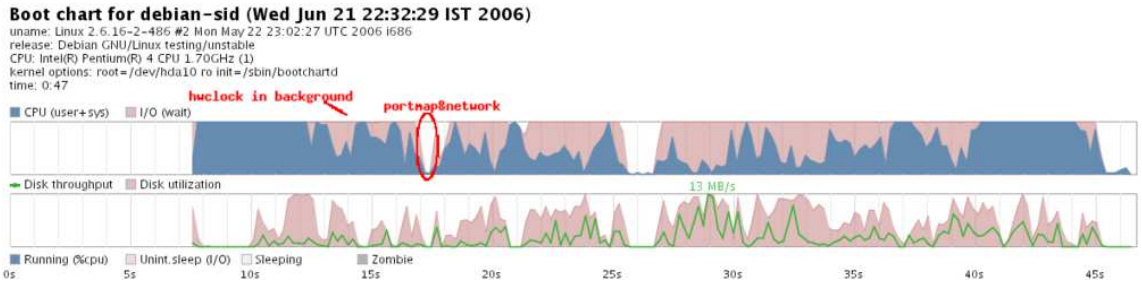

Figure 13: Bootchart with hwclock in the background

2.5 Parallel execution of boot scripts

We tried parallel execution by setting the CONCURRENCY variable in /etc/default/rcS to both shell and startpar. In order to avoid unmet dependencies during the boot time, the system was reordered using insserv and afterwards modifying the order manually. Insserv currently doesn't change the order in /etc/rcS.d and the new order of /etc/rc2.d is:

S08gdm
S08kdm
S08makedev
S08stopbootlogd
S09portmap
S10nfs-common
S11sysklogd
S12klogd
S13acpid
S13atd
S13cron
S13dbus

S13dirmgr
S13exim4
S13lpd
S13rmnologin
S14hotkey-setup
S19ssh
S20inetd

The boot time difference between parallel execution using CONCURRENCY=startpar and CONCURRENCY=shell in /etc/default/rcS was tested. CONCURRENCY=startpar shows no improvement from a modified script order with insserv while there is a further 2 second improvement with CONCURRENCY=shell. Thus, the total time improvement with parallel execution and reordering was of 4 seconds. The reason for this behaviour may be due to a bug found in startpar.

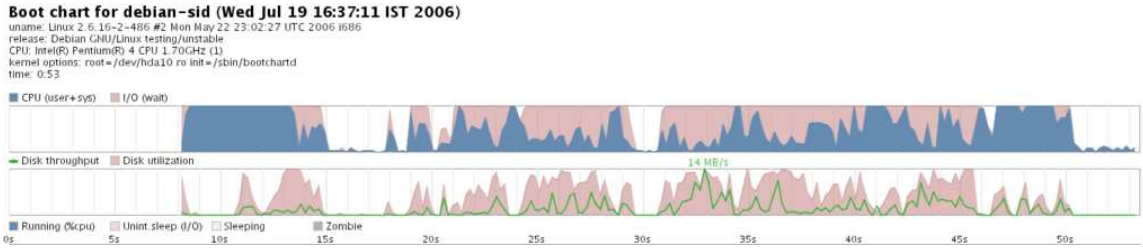

Figure 14: Bootchart for reference system

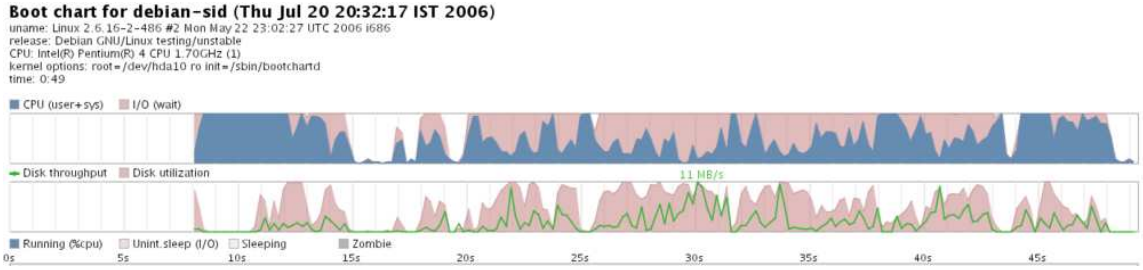

Figure 15: Bootchart for parallel execution with CONCURRENCY=shell

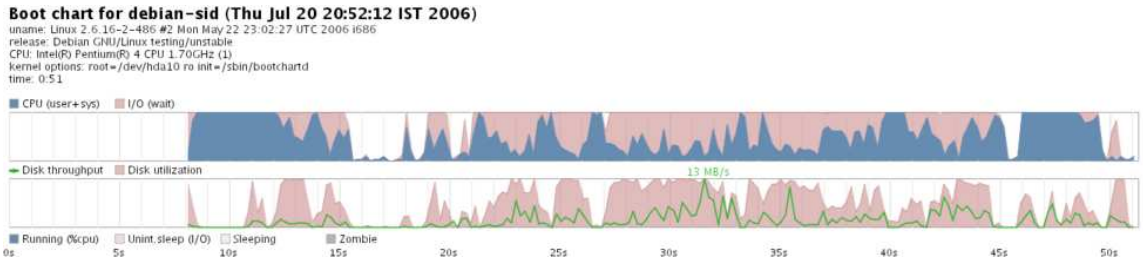

Figure 16: Bootchart for parallel execution with CONCURRENCY=startpar

2.6 Remove depmod from the boot process

Depmod is not used any more in sid's module-init-tools and this is one of the hotspots. This was removed as a result from a discussion started by Margarita Manterola in debian-devel. Just by adding it again from an old script the boot time increases by 2 seconds. Similarly, 2 seconds were gained by Margarita Manterola[2].

2.7 Set up the network in the background

Setting up the network in the background was proposed as a promising hotspot based on the results from Margarita Manterola [2] with a 2 second reduction. This hotspot was tried by modifying the networking init script but no time difference was noticed.

Setting up the network is sent to the background by sending most of the commands to the background:

```
(
log_action_begin_msg "Configuring network interfaces"
if ifup -a; then
log_action_end_msg $?
else
log_action_end_msg $?
fi )&
```

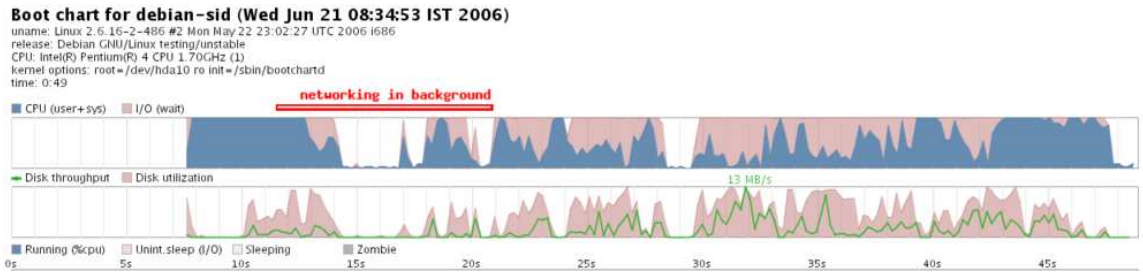

Figure 17: Bootchart of reference system

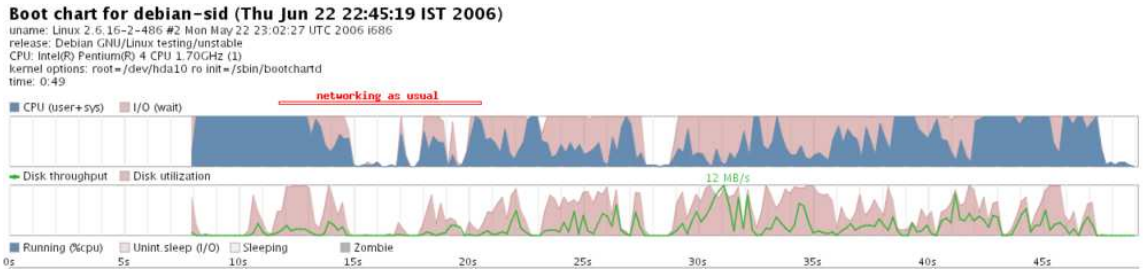

Figure 18: Bootchart with networking in the background

2.8 Remove discover

The discover script was removed as udev is already doing the same job. This change is valid for 2.6 kernels as older kernels don't support udev. We noticed no change on the boot time by removing discover. This should be hardware dependent.

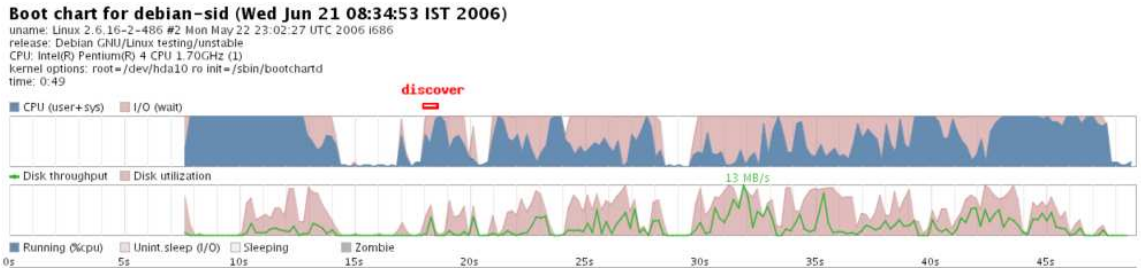

Figure 19: Bootchart of reference system

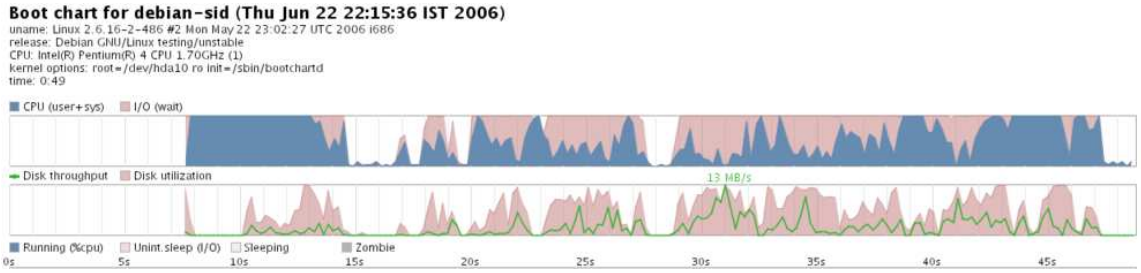

Figure 20: Bootchart with discover in the background

In some newer tests we've removed discover with even a decrease in the time. This was performed in an upgraded sid system.

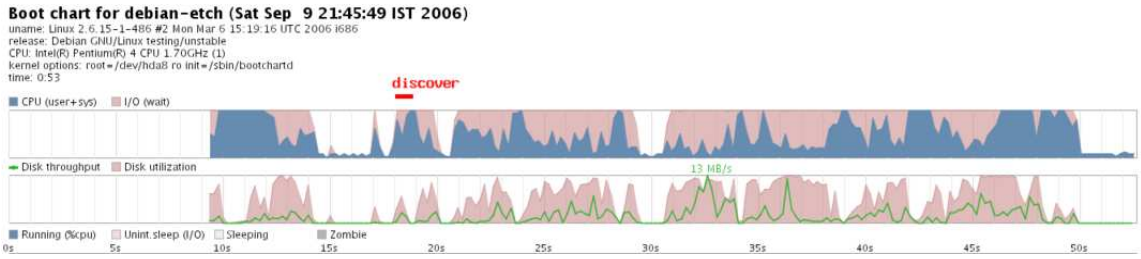

Figure 21: Bootchart of reference system

2.9 Make the boot less verbose

.

Making the boot print less messages to the screen should decrease the boot time. This approach probably requires to modify most scripts although the issue of deciding the information that should be showed would remain open.

The verbose option from the file /etc/default/rcS was changed to "VERBOSE=no" and "VERBOSE=quiet" but the effect was a one second longer boot time for both cases. This means that a different way to test it has to be found and that the current VERBOSE variable doesn't work properly. Nevertheless, by taking a look at the bootcharts, one may notice that kde starts half a second earlier although it takes longer to load ktip and stop bootlogging.

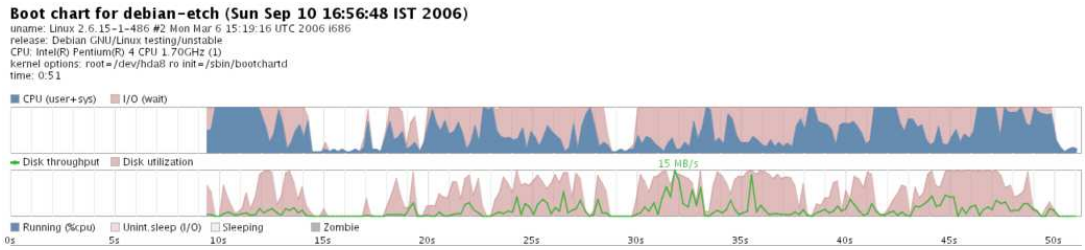

Figure 22: Bootchart of newer system with discover in the background

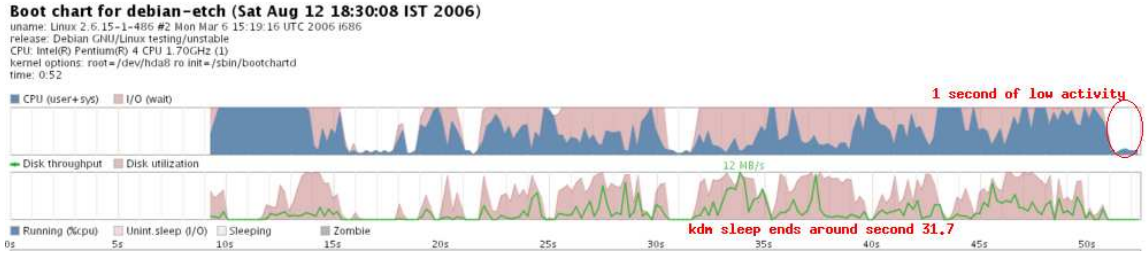

Figure 23: Bootchart for reference system

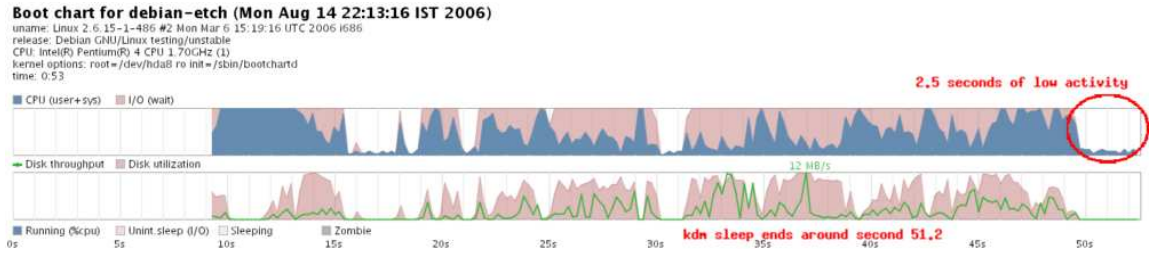

Figure 24: Bootchart for less verbose boot with VERBOSE=no

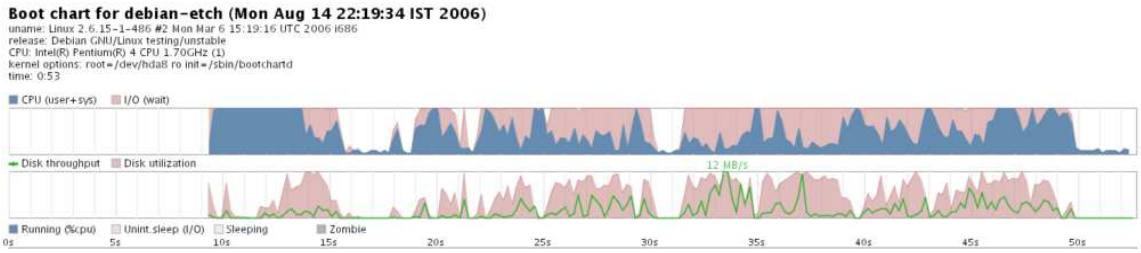

Figure 25: Bootchart for less verbose boot with VERBOSE=quiet

2.10 Use ELF prelinking

The use of ELF prelinking could help with the programs that need to link to many libraries. We tried the unstable package of prelink for Debian.It was added after readahead in a custom script run during the boot in /etc/rcS.d/S70prelink. What the script does is to prelink: kdeinit, Xorg, kwin, kdesktop, kicker, artsd, kaccess, ktip, klipper and korgac.

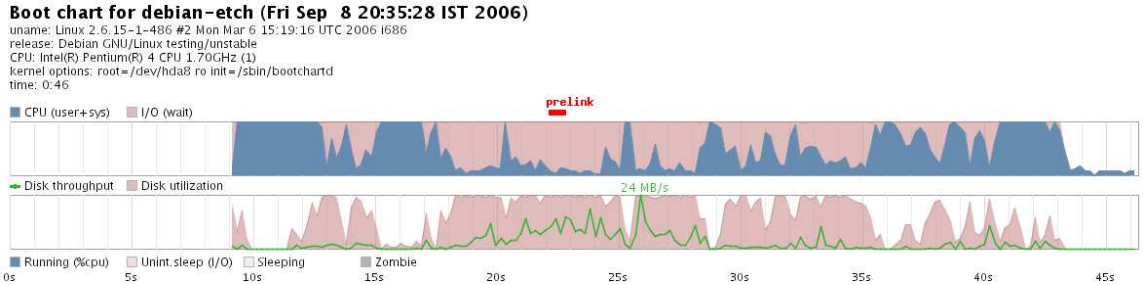

Figure 26: Bootchart of reference system

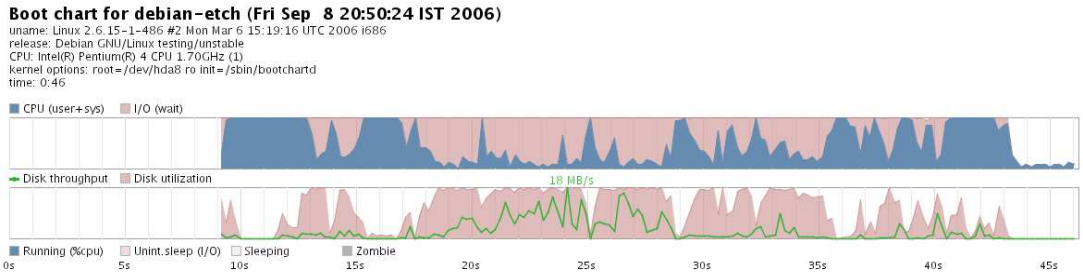

Figure 27: Bootchart system with prelinking

Hotspot	Comb.1	Comb.2	Comb.3	Comb.4	Comb.5	Comb. 6
dash	X	X	X	X	X	
hwclock	X		X	X		
network	X		X	X		
reorder	X	X	X	X	X	X
parallel		X	X		X	X
preloading		X	X	X	X	X
time red.	4	6	6	6	6	4

Table 2: Time reduction with *hotspot* combination

3 Hotspot combination

The different *hotspots* were tested in combination with others to try to find both, the interactions among them and the best probable combination. Currently, dash, preloading and reorder together the bootscripts seem to have the biggest effect (Combinations 2 to 5).

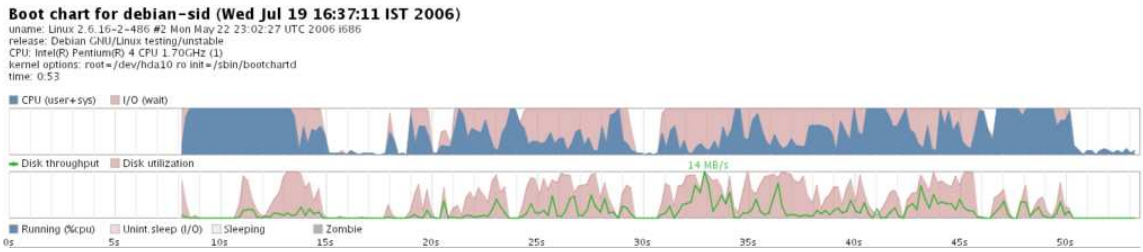

Figure 28: Bootchart for reference system

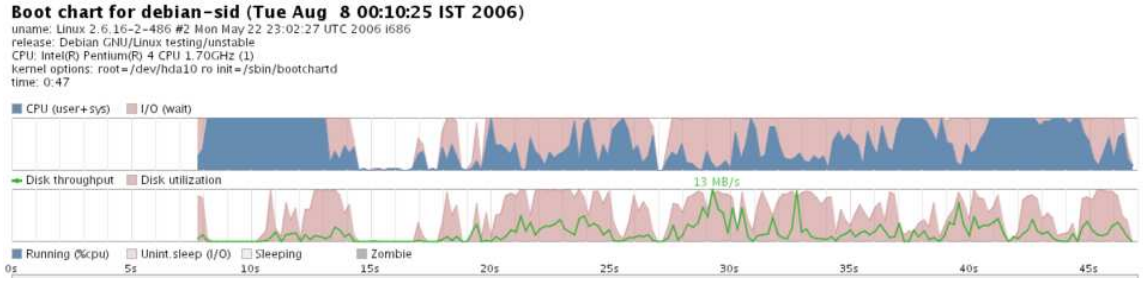

Figure 29: Bootchart for Combination 2: dash+shell+insserv+preload

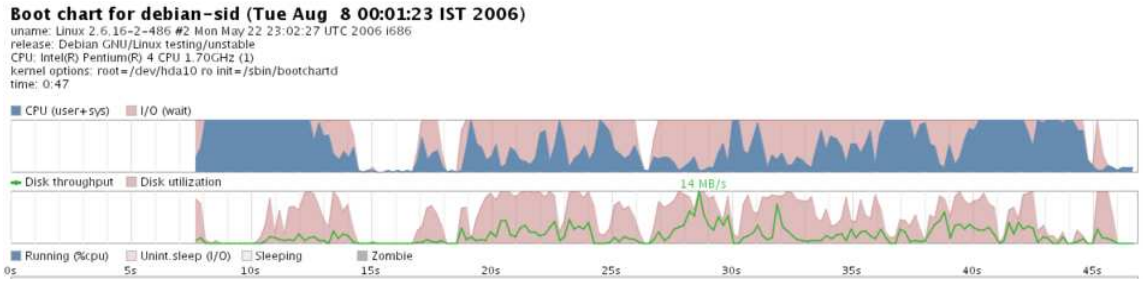

Figure 30: Bootchart for Combination 3: dash+shell+insserv+preload+hwclock+networking

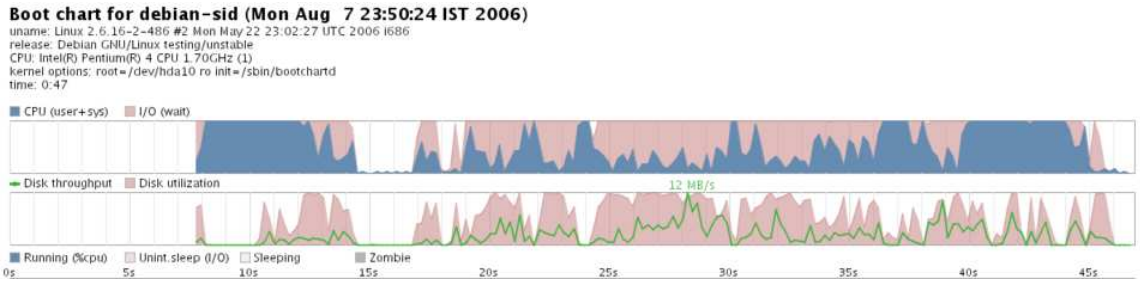

Figure 31: Bootchart for Combination 4: dash+insserv+preload+hwclock+networking

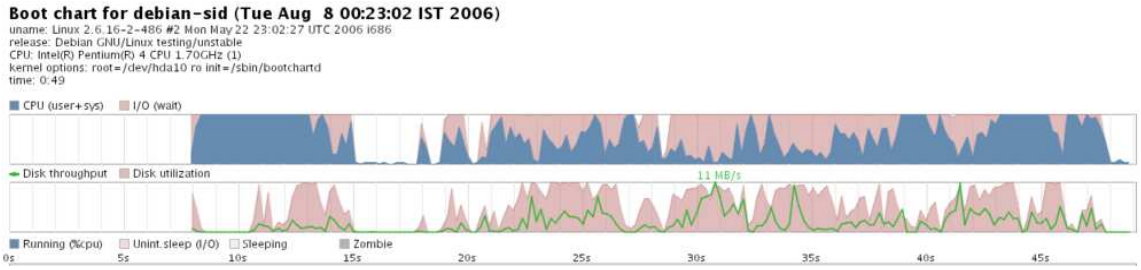

Figure 32: Bootchart for Combination 5: shell+insserv+preload

4 Conclusions and Future Work

Discussion on the project may be followed in the initscripts-ng-devel mailing list and the channel #pkg-sysvinit in *irc.debian.org*.

A readahead-list in /etc/readahead/boot

/etc/default/rcS
/etc/hotplug/blacklist.d
/etc/hotplug/blacklist.d/alsa-base
/etc/hotplug/blacklist.d/libsane
/etc/hotplug/blacklist.d/linux-sound-base_noOSS
/etc/init.d/module-init-tools
/etc/ld.so.cache
/etc/modprobe.conf
/etc/modprobe.d
/etc/modprobe.d/aliases
/etc/modprobe.d/alsa-base
/etc/modprobe.d/alsa-base-blacklist
/etc/modprobe.d/arch-aliases
/etc/modprobe.d/blacklist
/etc/modprobe.d/display_class
/etc/modprobe.d/libsane
/etc/modprobe.d/linux-sound-base_noOSS
/etc/modprobe.d/pnp-hotplug
/etc/modules
/lib/lsb/init-functions
/lib/modules/2.6.15-1-486/kernel/drivers/cdrom/cdrom.ko

/lib/modules/2.6.15-1-486/kernel/drivers/ide/ide-cd.ko
/lib/modules/2.6.15-1-486/kernel/drivers/ide/ide-core.ko
/lib/modules/2.6.15-1-486/kernel/drivers/ide/ide-disk.ko
/lib/modules/2.6.15-1-486/kernel/drivers/ide/ide-generic.ko
/lib/modules/2.6.15-1-486/kernel/drivers/input/mouse/psmouse.ko
/lib/modules/2.6.15-1-486/modules.dep
/lib/tls/libc.so.6
/usr/lib/gconv/gconv-modules.cache
/usr/lib/locale/locale-archive
/usr/share/locale/en_GB/LC_MESSAGES/grep.mo
/usr/share/locale/en_IE/LC_MESSAGES/grep.mo
/usr/share/locale/en/LC_MESSAGES/grep.mo
/usr/share/locale/en_US/LC_MESSAGES/grep.mo
/usr/share/locale/locale.alias
/etc/blkid.tab
/etc/fstab
/etc/init.d/mountall.sh
/etc/ld.so.cache
/etc/mtab
/lib/init/vars.sh
/lib/libblkid.so.1
/lib/libdevmapper.so.1.02
/lib/libselinux.so.1
/lib/libsepol.so.1
/lib/libuuid.so.1
/lib/lsb/init-functions
/lib/tls/libc.so.6
/lib/tls/libdl.so.2
/usr/lib/locale/locale-archive
/usr/share/locale/en_GB/LC_MESSAGES/util-linux.mo
/usr/share/locale/en_IE/LC_MESSAGES/util-linux.mo
/usr/share/locale/en/LC_MESSAGES/util-linux.mo
/usr/share/locale/en_US/LC_MESSAGES/util-linux.mo
/usr/share/locale/locale.alias
/etc/default/portmap
/etc/init.d/portmap
/etc/ld.so.cache
/lib/libncurses.so.5
/lib/lsb/init-functions
/lib/terminfo/l/linux
/lib/tls/libc.so.6
/etc/console-tools/config
/etc/console-tools/remap
/etc/default/locale
/etc/init.d/console-screen.sh
/etc/ld.so.cache
/lib/libcfont.so.0
/lib/libconsole.so.0
/lib/libctutils.so.0
/lib/libncurses.so.5
/lib/terminfo/l/linux
/lib/tls/libc.so.6
/lib/tls/libdl.so.2
/lib/tls/libm.so.6
/usr/lib/gconv/gconv-modules.cache
/usr/lib/locale/locale-archive

/usr/share/consolefonts/lat0-sun16.psf.gz
/usr/share/locale/en_GB/LC_MESSAGES/grep.mo
/usr/share/locale/en_IE/LC_MESSAGES/grep.mo
/usr/share/locale/en/LC_MESSAGES/grep.mo
/usr/share/locale/en_US/LC_MESSAGES/grep.mo
/usr/share/locale/locale.alias
/etc/init.d/alsa-utils
/etc/ld.so.cache
/lib/lsb/init-functions
/lib/tls/libc.so.6
/lib/tls/libdl.so.2
/lib/tls/libm.so.6
/lib/tls/libpthread.so.0
/usr/lib/gconv/gconv-modules.cache
/usr/lib/libasound.so.2
/usr/lib/locale/locale-archive
/usr/share/alsa/alsa.conf

References

[1] How to lsbize an init script. http://wiki.debian.org/LSBInitScripts.

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

[3] Ubuntu feature specification: Dash as /bin/sh. https://launchpad.net/distros/ubuntu/+spec/dash-as-bin-sh.

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

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

[6] Carlos Villegas and Petter Reinholdtsen. Benchmarking debian and first hotspots. In *Improve the Debian Boot Process Project of Google Summer of Code 2006*, June 2006.
http://initscripts-ng.alioth.debian.org/soc2006-bootsystem/deliverable2.html.