Quick booting with Upstart, a replacement for the legacy Sys V Init

JUMP START

The slow Linux boot has troubled users for years. Now the Upstart project offers a fresh approach to the problem of booting Linux.

BY NICO DIETRICH, DIRK VON SUCHODOLETZ

The history of Linux includes many attempts to address the long Linux boot process. This comes as no surprise, as a long boot marathon will annoy all but the most patient of users. The legacy Unix System V boot design was once revolutionary, but has turned out to be a millstone around the necks of many distributions.

Although several tricks for speeding up the process have appeared through the years, most boot reforms have turned out to be unworkable in practice, and many of the turbo loaders employed by gurus are inaccessible to regular users. A new tool, Upstart, takes a fresh approach to speeding up Linux boot.

The high-flying Upstart project [1] introduces a generic init daemon that leverages many developments in modern Linux systems. Upstart, which has Unix roots, cleverly steers clear of unnecessary waits, runs start scripts simultaneously, and reduces the boot time to a minimum. The long-term plan is to replace generic background system services such as the at daemon, cron, and others with Upstart. Ubuntu 6.10 (Edgy Eft) demonstrates the first effects of this promising software.

It All Starts with Init

Most Unix-style systems share the init concept. The process calls the kernel and assigns the kernel process ID 1. This sequence is hard-coded into the kernel – in */usr/src/linux/init/main.c* on Linux. Init has the task of launching all other userspace processes and initializing the machine. The process and its helper scripts load kernel modules, check and mount the filesystem, set up the network, launch servers, and call the graphical login manager. Init has to launch the services in a meaningful order. For example, it doesn't make sense to set the system time by polling a time server on the network until the machine becomes available on the network. To do this, init has to first initialize the network hardware, and set up at least one network interface for external access.

The number of services and background agents has grown over the years, and this has made the init process clumsy. In contrast, desktop usage, which is typical for Ubuntu, demands a dynamic system configuration. Mobile devices also make life really difficult for Sys V Init. Roving devices require an adhoc approach to setting up network connections, as well as a means for dynamically configuring hardware.

To cope with this, Linux programmers have developed a number of tools: *acpid* and *apmd* for power management, the HAL Device Manager for dynamic mounting of drives, and the Resource Manager for dynamic device privilege assignments to the user working with the GUI desktop. Each of these sytems implements its own configuration logic, and admins have to be familiar with this logic to be able to run a required task at the right point in time.

Not all processes and services are tied to starting or stopping a machine. For example, there are some special services, like cron and the at daemon, that launch other processes at a certain time. They are not linked to the runlevel system in any way, although they have a similar underlying logic. This is another thing that Upstart looks to change [3].

Design Issues

Before the Ubuntu programmers decided to develop a new system, they first investigated contemporary alternatives to the familiar Sys V system [2]. None of the designs they looked at fulfilled their expectations or was available under an acceptable license.

When they started to think about the new design, they had to choose between a target or result-oriented option for the system launch. Target-oriented would mean defining the services that should be running at the end of the start sequence (KDM, SSH daemon, etc.).

In this case, it would be necessary to investigate each service and determine which other services it relied on. Based on these dependencies, the init system would need to derive a meaningful startup sequence. This is exactly the approach that Gentoo adopted with its *depend* system (see the "Gentoo" box); Suse also follows this approach with a modified version of the Sys V Init (see the "Suse Linux" box).

In the other corner of the ring, there were events. Instead of formulating dependencies, which a script would probably need to handle at system start time, an event-based system would not run a script until a specific set of preconditions was fulfilled. For example, it would not make sense to call an NFS client until the authoritative NFS became available. The system that Ubuntu opted for also understands more complex conditions, such as "network configuration completed," "Apache running," or (in the future) "USB stick plugged."

Event Horizon

Events are basically just simple strings. The Upstart developers divide events into three classes:

- Simple edge events, such as "the system is booting," or "the user has just pressed a button."
- Level events have an additional parameter, such as the network interface status. Services and tasks run either for any Level Event, or only when a parameter has reached a specific value.
- Temporal events occur after a specific interval, or at a specific point in time.

The developers kept to the open source codex of release early, release often. Thus, the code was released at a very early stage, and the self-confident developers presented a running system, in the guise of Edgy Eft, to demonstrate how far they have gotten. Their aim is to collect as much feedback as possible from developers working on other Linux distributions.

However, this also means that the specifications might change in the

Sys V Init

The early Unix versions used a simple shell script to configure the machine and launch services. The design behind the BSD family's */etc/rc*, for example, was convincingly simple, but it did have one major drawback. Integrating third-party software, or custom-built extensions, meant modifying the shell script. Unfortunately, modifying this code is quite dangerous – a single mistake could lead to an unbootable system.

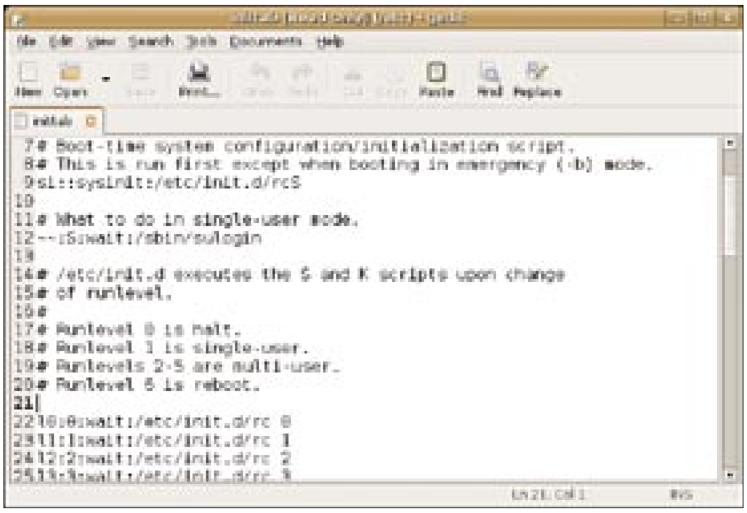
In many cases, it takes more than just a simple command to launch a service, with the details varying depending on the current environment. For example, the ISC DHCP server can be set up to listen on specific Ethernet interfaces, rather than on all ports. To remove the need for administrators to modify the start script to do this, daemons often come with configuration files that parse the script. This lets administrators update the start scripts without endangering the local configuration.

Sys V Init uses a far more flexible, but also more complex, approach than BSD, introducing runlevels that define specific machine states, based on the processes that run in them. A total of eight runlevels is possible, but not mandatory, for any system. Three runlevels have clearly defined tasks: Halt (0), Single User Mode, and Reboot (6).

The */etc/inittab* file specifies which runlevels exist and defines the runlevel the system enters on booting (Figure 1).

The Sys V design assumes that the system will use a small number of defined states, such as without network, with network, with X11, and so on. Administrators can change the runlevel by giving the *init Runlevel* command.

Another advantage of this approach is the separate scripts for each service or configuration task. For example, calling */etc/init.d/dhcpd restart* lets you relaunch the DHCP server without affecting any other services. The idea of using a collection of symlinks to determine the scope and order of the scripts used in each runlevel is also a good one.



```
# Boot-time system configuration/initialization script.
# This is run first except when booting in emergency (-b) mode.
si::sysinit:/etc/init.d/rcS

# What to do in single-user mode.
~~:S:wait:/sbin/sulogin

# /etc/init.d executes the S and K scripts upon change
# of runlevel.
#
# Runlevel 0 is halt.
# Runlevel 1 is single-user.
# Runlevels 2-5 are multi-user.
# Runlevel 6 is reboot.

l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
```

Figure 1: This typical inittab defines runlevels 0 through 6.

course of the next few months. The examples described here refer to version 0.3 from early December.

This version replaces the existing init process, however, don't assume that all the start scripts have been modified to use the event mechanism. Upstart does not support temporal events right now, and there are plans to use other programs, such as Udev and the ACPI or APM daemons, as event sources.

State of the Art

The current version of Ubuntu boots quite quickly, although you won't see much of what's going on if you use the default setup. The boot splash screen with its progress indicator, and the OS logo hide any useful information. Although normal users may not object to this, it takes some getting used to for admins. Even if you remove the colorful splash (by deleting the *splash* token from the kernel command line in grub), you won't see too many messages. If you want more, just remove the *quiet* entry.

On first inspection, the changes under the hood are also hidden. If you give the *man init* or *man telinit* command, you are informed that the runlevel system has a new engine. No */etc/inittab* file is another telltale clue. */etc/init.d* and the accompanying start scripts still exist at present as Ubuntu currently runs Upstart in compatibility mode. The mid-term plan is to allow */etc/event.d* to handle job definitions, which boil down to simple, non-executable files like the one shown in Listing 1. The example takes the easy way out, and simply calls the old start scripts for runlevel 2 (line 20).

As you can see from line 5, we want the script to run whenever the *runlevel-2* event occurs. It ends if the events *shutdown*, or *runlevel-3* through *runlevel-5* occur (lines 7 through 10). In the future, more complex semantics will support conditions with logical operators and pass in parameters to event scripts, if needed. These files play the same role as the entries in the legacy */etc/inittab*. This is why Edgy Eft has both *rc2* and the files shown at the top of Figure 2.

Upstart-Compliant Jobs

There are two ways of defining your own jobs. The simple method uses the *exec /path/program -O --optional parameter* approach. This works just like in the shell. Upstart actually uses a shell to handle quotes, *""* or *$*. If the job definition contains more than a simple command line, the shell script can reside between the *script* and *end script* tokens (Listing 1, lines 12 through 21).

There are two variations on this scripting theme, *start script* and *stop script*. The start script does what the service requires, like creating directories or check-

Gentoo

As one of the more recent additions to the Linux family, Gentoo solved the problem of organizing runlevel scripts in its own special way. To do so, it does not use simple bash scripts as runlevel scripts, but instead launches a separate interpreter: */sbin/runscript*. An example of a typical structure follows:

```
#!/sbin/runscript
opts="depend start stop restart"
depend() {
  # dependencies and conditions
}
start() {
  # commands for starting
services
  # including preparatory tasks
}
stop() {
  # commands for stopping
services
  # plus clean-up actions
}
restart() {
  # Restarting as service
}
```

The string that follows *opts* lists all the functions provided by the runlevel script. If you need to add your own functions, you just add them to the list, and script a function block of the same name to match. While the *start*, *stop*, and *restart* sections keep to the traditional design, more interesting things happen in *depend*. A service depends on other services or preparatory settings on the one hand; but on the other hand, it can provide specific functions that other services require:

- *need service*: Depends on the service.
- *use service*: Uses the service.
- *provide functionality*: Provides a specific functionality.
- *before service*: Should start before the other service.
- *after service*: Should start after the specified service.

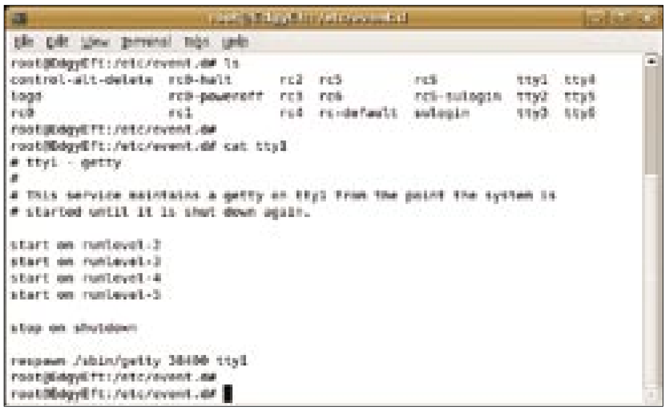
Gentoo also supports virtual services, such as *net*, as there are various kinds of network (Ethernet, Modem, WLAN). This also applies to mail servers (*mta*). The start script can even determine dependencies dynamically, as the */etc/init.d/syslog-ng* example shows:

```
case $(sed 's/#.*//'↩
 /etc/syslog-ng/syslog-ng.conf) in
  *source*tcp*|*source*udp*|↩
*destination*tcp*|*destination*udp*)
  need net ;;
esac
```

As long as the change does not conflict with existing dependencies, admins can alter the order in which services are started, using *before* or *after*.

Listing 1: Job Definitions

```
01 # /etc/event.d/rc2
02 # Runlevel 2 compatibility
   script for Upstart
03 # This job runs the old
   Sys V scripts.
04
05 start on runlevel-2
06
07 stop on shutdown
08 stop on runlevel-3
09 stop on runlevel-4
10 stop on runlevel-5
11
12 script
13     set $(runlevel --set 2
   || true)
14     if [ "$1" != "unknown"
   ]; then
15         PREVLEVEL=$1
16         RUNLEVEL=$2
17         export PREVLEVEL
   RUNLEVEL
18     fi
19
20     exec /etc/init.d/rc 2
21 end script
```



```
root@EdgyEft:/etc/event.d# ls
control-alt-delete  rc0-halt      rc2  rc5         rcS          tty1  tty4
logd                rc0-poweroff  rc3  rc6         rcS-sulogin  tty2  tty5
rc0                 rc1           rc4  rc-default  sulogin      tty3  tty6
root@EdgyEft:/etc/event.d#
root@EdgyEft:/etc/event.d# cat tty1
# tty1 - getty
#
# This service maintains a getty on tty1 from the point the system is
# started until it is shut down again.

start on runlevel-2
start on runlevel-3
start on runlevel-4
start on runlevel-5

stop on shutdown

respawn /sbin/getty 38400 tty1
root@EdgyEft:/etc/event.d#
root@EdgyEft:/etc/event.d# 
```

Figure 2: Edgy Eft stores files that define jobs for the typical events from the legacy inittab in /etc/event.d.

ing access privileges. The stop script cleans up after the service terminates.

Self-Executing

Using a simple server as an example, look at what creating your own Upstart scripts involves. The server doesn't have to do anything but keep on running. The following section is based on a two-liner in */usr/local/bin/simpleserver.sh*:

```
#/bin/sh
while true ; do sleep 1↩
 ; done
```

Let's call the event script for this service */etc/event.d/simple-server*. If we only want to support manual launching of the service, all we need is a single line in the event script to start the server:

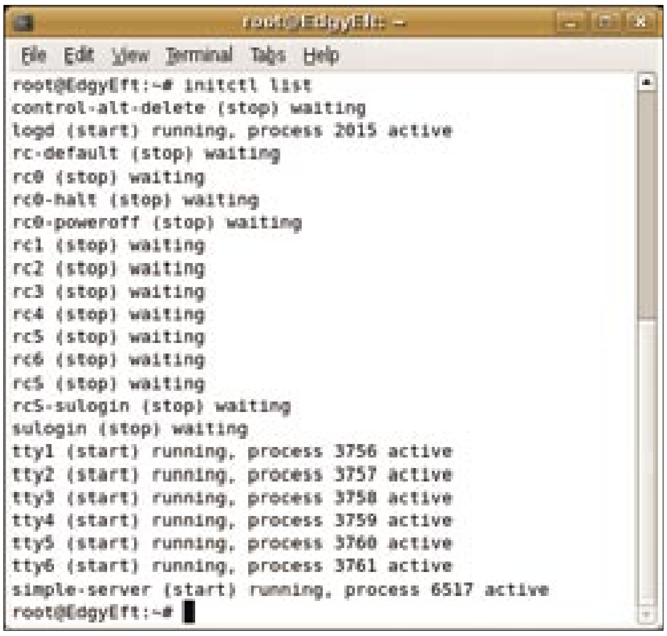
```
exec /usr/
local/bin/↩
simpleserver.↩
sh
```

To start and stop services, including the one just defined, we still rely on the *start* and *stop* commands, plus there is a new *status* call.

A simple *start simple-server* brings the service to life. To see whether the command worked, use *initctl list* or *status simple-server*. *stop simple-server* terminates the service (Listing 2).

If everything is working, most users do not want to see system messages. However, log information can be useful, especially if you just modified the system. If you prefer not to output messages on screen at boot time, you can check them later, of course.

Generally speaking, the Upstart script output is passed to the *logd* included with the package, and the daemon hands them on to */var/log/boot* (Listing 3). The *initctl list* call provides another source of debugging output (Figure 3).



```
root@EdgyEft:~# initctl list
control-alt-delete (stop) waiting
logd (start) running, process 2015 active
rc-default (stop) waiting
rc0 (stop) waiting
rc0-halt (stop) waiting
rc0-poweroff (stop) waiting
rc1 (stop) waiting
rc2 (stop) waiting
rc3 (stop) waiting
rc4 (stop) waiting
rc5 (stop) waiting
rc6 (stop) waiting
rcS (stop) waiting
rcS-sulogin (stop) waiting
sulogin (stop) waiting
tty1 (start) running, process 3756 active
tty2 (start) running, process 3757 active
tty3 (start) running, process 3758 active
tty4 (start) running, process 3759 active
tty5 (start) running, process 3760 active
tty6 (start) running, process 3761 active
simple-server (start) running, process 6517 active
root@EdgyEft:~# 
```

Figure 3: The initctl list command provides you with a system status overview.

Upstarting Debian

As Ubuntu is based on Debian, the chances of accelerating Debian, thanks

Listing 2: Service Control

```
root@EdgyEft:~# start simple-server
simple-server (start) running, process 6507 active
root@EdgyEft:~# stop simple-server
simple-server (stop) running, process 6507 killed
root@EdgyEft:~# status simple-server
simple-server (stop) waiting
root@EdgyEft:~# start simple-server
simple-server (start) running, process 6517 active
root@EdgyEft:~# status simple-server
simple-server (start) running, process 6517 active
```

Suse Linux

While the traditional Sys V Init folllows a strictly linear approach, more recent Suse Linux versions (10.0 or newer) support parallelization of boot script calls. Administrators can enable this feature in the */etc/sysconfig/boot* file by setting the *RUN_PARALLEL* variable to *yes*. This changes the legacy sequence defined by *S00script1* through *S99script25*. Instead, the *.depend.boot*, *.depend.start*, and *.depend.stop* dependencies are applied. If an administrator adds a simple script, say, *S12nbd-server*, to *rc3.d*, by creating a link in the traditional manner, the system will just ignore the change. The *insserv* command handles this task by evaluating the file header to ensure correct resolution of dependencies:

```
### BEGIN INIT INFO
# Provides: nbd-server
# Required-Start: $network
# Should-Start: $syslog
# Required-Stop:
# Default-Start:  3 5
# Default-Stop:   0 1 2 6
# Description:    ↩
Start Network Blockdevice Daemon
### END INIT INFO
```

This hides much of the complexity from the user, however, the approach does not provide much in the line of speed benefits. When we tested the Suse-style boot on the X41 mentioned earlier on – the machine admittedly has a fairly lame hard disk – the parallel boot took just over a minute, which is fairly close to the 70-second value for the legacy approach. You can actually see some evidence of parallel service launching; the screen output is mixed up.

Listing 3: Upstart Boot

```
[...]
Dec  3 18:44:59 rc2:
 * Starting deferred execution
 scheduler atd       [ ok ]
Dec  3 18:44:59 rc2:
 * Starting periodic command
 scheduler...          [ ok ]
Dec  3 18:44:59 rc2:
 * Enabling additional
 executable
  binary formats ...[ ok ]
Dec  3 18:44:59 rc2:
 * Checking battery state...
                      [ ok ]
Dec  3 18:44:59 rc2:
 * Running local boot scripts
 (/etc/rc.local)      [ ok ]
```

to the changes in Ubuntu, are pretty high. If you are prepared to take the risk, you can opt to either replace the existing Sys V Init completely, or to use Upstart in parallel.

The steps for implementing plan A (using Upstart to completely replace the legacy system) in Debian Unstable are fairly simple – the developers of the distribution have already completed the preparatory work by separating the *sysvinit-utils* from the *sysvinit* package. This means that you can easily replace *sysvinit* with Upstart, and just keep the old scripts.

There is an Upstart package in the Debian Experimental repository [2]. To use the package, add the following entry to your */etc/apt/sources.list*:

```
deb http://ftp.de.debian.org/⮐
debian/experimental main
```

After doing this, give the command to remove the legacy *sysvinit* package:

```
apt-get install upstart⮐
 upstart-compat-sysv
```

Because *sysvinit* is tagged as *required*, the package manager will wait until you type *Yes, do as I say!*. The next pitfall occurs when you update. *apt-get dist-upgrade* will remove the Upstart packages you just installed, and reinstate the *sysvinit* package.

If this is your intention – that is, if really do want to reinstate Sys V Init – you can simply give the *apt-get install sysvinit* command to achieve your goal.

Self-Administration

If you decide to build Upstart yourself, you should manually remove the *sysvinit* package. If you fail to do so, *make install* will overwrite the central binaries, and the package manager will either ignore the changes, or worse, decide that your system is corrupted. This said, it is easy to build and install Upstart from the source code [1]:

```
./configure --prefix=/usr ⮐
--exec-prefix=/⮐
  --sysconfdir=/etc
make
make install
```

After completing these steps, the system will ask you for your init scripts.

To get started, download the *example-jobs-2.tar.gz* tarball from the */download* directory [1] and unpack in */etc/event.d*.

Parallel Worlds

If you want to avoid an unsuccessful Upstart installation wrecking your working System V Init system, you can install Upstart side by side with Sys V. To do so, follow the same steps as for the Upstart-only install, but keep the *sysvinit* package, and make sure the new init ends up in */opt/upstart*:

```
./configure -prefix=/opt/⮐
upstart  --sysconfdir=/etc⮐
 --enable-compat
```

Listing 3a: Grub Configuration

```
01 # /boot/grub/menu.lst
02 [...]
03 title    Ubuntu, kernel
    2.6.17-10-generic
04 root     (hd0,0)
05 kernel   /boot/vmlinuz-
   2.6.17-10-generic root=/dev/
   hdb1 ro quiet splash init=/
   opt/
   upstart/sbin/init
06 initrd   /boot/initrd.img
   -2.6.17-10-generic
07 boot
08 [...]
```

In this scenario, you will need to modify the scripts you dropped into */etc/event.d*. To do this, just add the following line after the *script* line in *rc-default* and *rcS-sulogin*:

```
export PATH=/opt/upstart/⮐
sbin:$PATH
```

Because the Upstart directory is at the beginning of your search path, scripts will use the new *telinit* command.

The system will still boot Sys V Init by default, but on booting, you can tell the kernel to use an alternative to the legacy init. The following kernel command line parameters will do the trick: *init=/opt/upstart/sbin/init*.

For simple tests, you might prefer to enter the parameters at the bootloader prompt, but you could add a bootloader configuration menu entry if you prefer (Listing 3a, line 5).

Analysis

Bootchart [4] gives admins an excellent method for comparing the legacy and new boot processes. The tool logs the CPU load and hard disk I/O performance at boot time and later converts the results to a neat graph. To allow this to happen, you need to install the Bootchart package and add an entry to the kernel command line. *bootchartd* runs as an initial process and launches the *init* process proper.

Listing 3b shows you the entry for Grub. If you are running Upstart in side-by-side mode with the legacy Init, let Bootchart know by adding the following to the kernel command line:

```
bootchart_init=/opt/⮐
upstart/sbin/init
```

After doing so, Bootchart will log any interesting process data every 0.2 seconds and store the information in */var*

Listing 3b: Bootchart

```
# /boot/grub/menu.lst
[...]
kernel   /boot/vmlinuz
 -2.6.17-10-generic
  root=/dev/hdb1 ro quiet
  splash init=/sbin/bootchartd
[...]
```

advertisement

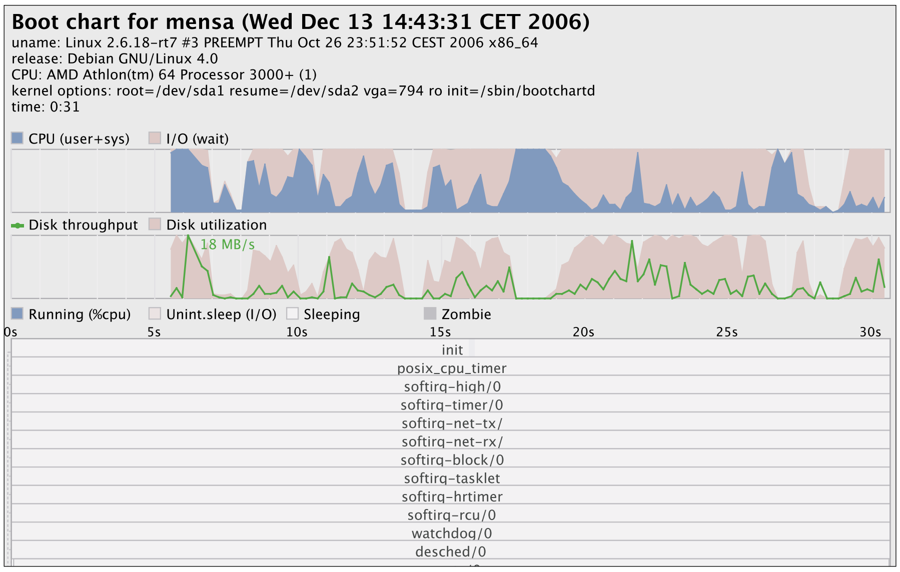


Figure 4a: This Bootchart analysis shows Debian GNU/Linux booting with the legacy init procedure.

/log/bootchart.tgz once the boot process has completed. The *bootchart -f png* generates a PNG graphic from the data, with *svg* and *eps* as your other options.

If you compare the Sys V boot graph in Figure 4a with the Upstart graph in 4b, the results are misleading. On our lab machine, Bootchart reports that the legacy Sys V Init takes a hefty 33 seconds, whereas Upstart is all done in 23 seconds flat.

Checking these results with a stopwatch revealed that the gain is actually a mere two seconds. Bootchart stops the clock as soon as KDM or another login manager is launched. The fact that Upstart launches jobs in parallel means that this step simply begins earlier, before all the other critical boot processes have completed.

It would be unfair to the Upstart project to ignore the promise of Upstart on the basis of the current test results. Remember that the init successor will be running in compatibility mode until more progress has been made.

You can expect major speed gains as soon as the individual start scripts have been adapted to support the new system. Thus, Ubuntu does not expect to see major boot speed gains until Edgy is replaced by Feisty Fawn.

Conclusion

The days when a computer booted in a couple of seconds are gone, and the kind of surgery that many distributions use to patch up boot performance are not likely to correct the problem. The future looks good for new designs like Upstart.

Even if Upstart does not offer the same kind of "switch on and go" feeling you get from a gaming console, it requires less patience than the legacy system. Of course, a fair level of administrative skill is required to migrate a working Linux machine without having to reinstall.

Our crystal ball reveals that Upstart is aiming to do more than simply rejuvenate the boot process: the developers envisage a central service daemon that will take over chores that are handled by a potpourri of tools right now. This includes executing specific events at specific times, in other words, replacing cron and at. In the meantime, Upstart still has to demonstrate its ability to organize the boot process in a meaningful way, and as of now, it's well on the way to doing just that. ■

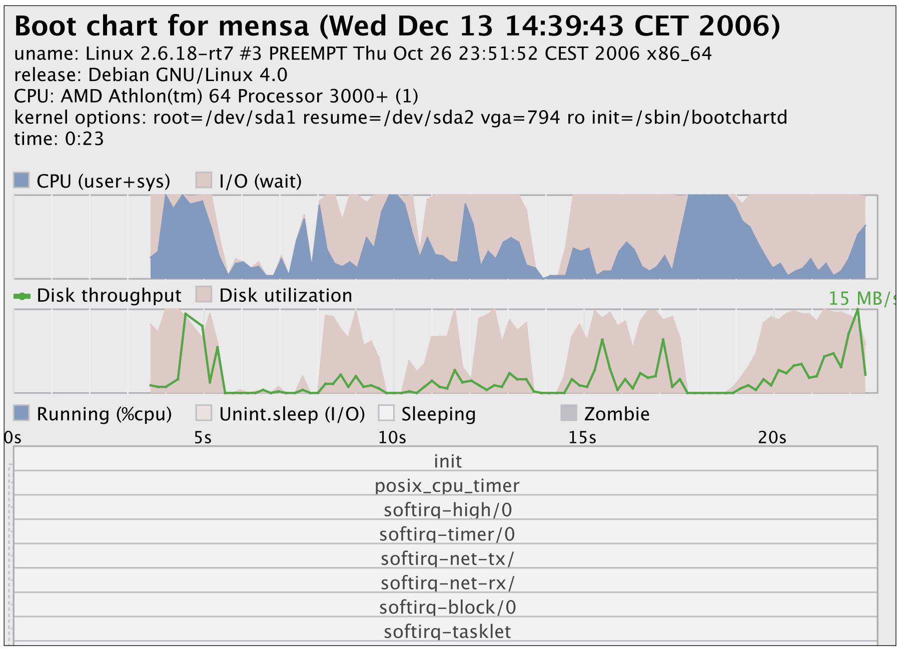


Figure 4b: Replaying Sys V Init with Upstart does not change the results greatly.

INFO

[1] Ubuntu Upstart: *http://upstart.ubuntu.com*

[2] Upstart in the Debian Experimental branch: *http://packages.debian.org/experimental/admin/upstart*

[3] Scott's blog entry on Upstart: *http://www.netsplit.com/blog/articles/2006/08/26/upstart-in-universe*

[4] "Boot Camp" by Charly Kühnast, *Linux Magazine,* March 2005, pg. 61

THE AUTHORS

Nico Dietrich studies computer science at the Technical University in Berlin, Germany, and is interested in free software, direct democracy, and developing countries. He will be looking to combine these fields after graduating.

Dirk von Suchodoletz works as an assistant at the Department of Communication Systems at the University of Freiburg, Germany. His development work focuses on diskless Linux projects, which explains his interest in system boot variants in general, and in accelerating the boot process in particular.