Parallelize Linux system services to improve boot speed

Level: Intermediate

James Hunt (jameshunt-at-uk.ibm.com)
Software Engineer, IBM
September 17, 2003

This article shows you how to improve the boot speed of your Linux system without compromising usability. Essentially, the technique involves understanding system services and their dependencies, and having them start up in parallel, rather than sequentially, when possible.

Although undoubtedly an excellent system, a common criticism of Linux -- voiced particularly by those from a Microsoft Windows background -- is that it takes a long time between pressing the "on" button and actually being able to use a Linux system. In essence, their argument goes, Linux takes a long time to boot.

Although simple to understand, the technique I present here for speeding the boot process requires careful implementation; my hope is that Linux distributions will adopt this technique and users will be spared the configuration task. But if you're feeling adventurous, read on.

Before you begin

If you intend to experiment with this technique, you must be familiar with Linux configuration scripts. Modifying system startup settings can be hazardous, and you might end up with a system that won't boot. If this happens, reboot into single-user mode (runlevel 1), undo the changes you made, and then reboot. As always, keep backups of all the files you change and have at least one system backup image in case the worst happens.

I would highly recommend that before you consider modifying a production system in the ways I suggest, use a test system that you can afford to trash. If you only have one machine, a very useful facility is *UML*, or User Mode Linux. UML is a kernel patch that allows the Linux kernel to be compiled into a binary that you can run as a normal program. This means that you can run a full Linux system as a process on your normal system. You can imagine it as running a Linux system *inside* your normal system. (See the Resources at the end of this article for a link to the UML download site and a *developerWorks* tutorial on UML.)

Using UML allows you to play with a test system that you can completely destroy *without* affecting your normal system.

Overview

The first part of this article covers the background of how a Linux system is started once the Linux kernel (the "core" of the Linux machine) has loaded. It then goes on to present a technique that can improve the speed at which your system boots.

If you are already familiar with runlevels and service startup scripts, you might want to jump to Limitations of the traditional services framework.

Linux boot sequence and runlevels

When a Linux machine boots up, it goes through a number of distinct phases. This article will not explain all the different phases, since we are only interested in the phases *after* the kernel has loaded.

To establish your machine's current runlevel, you can run the `/sbin/runlevel` command. (See

Contents:

Related content:

`man runlevel` for further details.)

Once the kernel has been loaded and has started to run, it calls the `/sbin/init` program. This program runs as root and sets the "runlevel" to that requested at initial boot time. (For more details on the `init` program, consult `man init`.)

What is a runlevel?
A *runlevel* is simply a number that Linux uses to distinguish different types of high-level configurations that the machine should be booted into. These runlevel numbers are on the whole "well-known," in the sense that most of them have very clearly defined meanings. On a Red Hat Linux system, the main runlevels are shown in Table 1.

Table 1. Red Hat Linux runlevels

Runlevel	Explanation
0	Halt the system
1	Single user mode (generally only used for administration purposes)
2	Multi-user mode with networking disabled
3	Multi-user mode with networking enabled
4	Unused runlevel
5	Multi-user mode with networking enabled, and X-Windows (graphical login)
6	Reboot

Slackware Linux 101: A look at what happens when you boot your Linux box

Introduction to User-Mode Linux

Subscribe to the developerWorks newsletter

developerWorks Toolbox subscription

Also in the Linux zone:

Tutorials

Tools and products

Code and components

Articles

How init initializes the system
`init` uses an ASCII configuration file (/etc/inittab) to tell it how to change the runlevel. Usually, this configuration file instructs `init` to run the script /etc/rc.d/rc, passing it the runlevel number.

From this article's perspective, the rc script is where things get interesting.

The rc.sysinit script
On Red Hat systems, before running the rc script, `init` runs the script /etc/rc.d/rc.sysinit, which performs required low-level setup tasks such as setting the system clock, checking the disks for errors, and subsequently mounting file systems.

System services
The rc script is responsible for starting all the services that the users require. As the name suggests, services are useful facilities that the system provides. There are potentially lots of services to start. Most Linux systems will start sshd (the SecureShell service), syslog (system logging facility), and lpd (printing service), but there could be many more. As an example, one of my Red Hat 9 system currently runs 29 services, but if I switched on all the services, this number would be closer to 50.

It is important to understand that an individual service may only start in particular runlevels. For example, there is little point in starting some form of graphical service in any runlevel apart from runlevel 5 (multi-user mode with graphics), since all the other runlevels are non-graphical. We will discuss this point further, below.

Where do services live?
Services are usually found in the /etc/rc.d/init.d/ directory.

Alternative services directory
On some Linux systems, services are sometimes located in the /etc/init.d/ directory instead.

If you browse around this directory, you'll find that quite a few (if not all) of the services are really shell scripts that call other programs to actually do the hard work.

How does the rc script know which scripts to run in each runlevel?
Going back to the point that we don't want certain services to start in certain runlevels, how do we tell this system to do this? The answer is that below the /etc/rc.d/ directory, alongside the init.d/ directory already discussed, is a set of directories, one for each runlevel. These directories are named rc<*runlevel*>.d/; so, for example, the directory for runlevel 5 is /etc/rc.d/rc5.d/. Each of these rc.d directories contains a set of symbolic links back to the actual service programs in the /etc/rc.d/init.d/ directory. In fact, as we will find out later, there are actually two symbolic links per service.

Service link names
The names of these symbolic links to the actual service programs are important, as they follow a strict naming convention that lets the rc script know how to handle them.

For easy identification, the name of each link is suffixed with the name of the service it is linked to.

The prefix is made up of two parts: a single uppercase character followed by a two-digit decimal number. The single uppercase character is either an "S" (meaning "start") or a "K" (meaning "kill," or stop). The two-digit number can range from 00 to 99.

Starting and stopping services

If we've decided to boot our Linux machine into graphical mode (runlevel 5), when init calls the rc script and passes it the runlevel number, the rc script will look in the directory /etc/rc.d/rc5.d/, and it will execute all the links it finds (in other words, it will run the program/script each link points to). It will run the links in two distinct phases; first it will execute all the links starting with a "K," passing these links the parameter "stop." This will have the effect of stopping all the services the links refer to.

Having stopped all the services it needs to, it will then execute all the links beginning with an "S," passing them the parameter "start," which will have the effect of starting all the services the links refer to. The rc script also passes the parameter "start" to each program.

Service link name regular expression

The names of the symbolic links to the services can be summarized by the `egrep` regular expression, `[SK][0-9]{2}[a-zA-Z]+`. (Read `man egrep` for further details on `egrep`).

The reason rc passes a parameter "start" or "stop" to each service program is to allow the same service program to be used to start and stop the service -- the service program knows whether the system is booting up or shutting down based on the value of the parameter passed to it.

There is one important aspect that I have not yet explained -- the numeric portion of the link names. The two-digit decimal number after the "S" or the "K" in the name of each link is used by the rc script to establish the *order* that the links (in other words, the services the links refer to) should be run. Links with a very low number (such as 00, 01, etc.) get run before links with high numbers (99 is the highest). We will come back to this important point later in the article.

Confused yet? An example should help explain. Listing 1 shows the links for runlevel 5. On booting to runlevel 5, the first link that will be executed will be K05saslauthd, since it starts with "K" and has the lowest two-digit decimal number of all the "K" links. The first startup link that will be run is S05kudzu, since it starts with "S" and has the lowest two-digit decimal number of all the "S" links. The last link to be run will be S99local.

Listing 1. Runlevel 5 links to service programs

```
# cd /etc/rc.d/rc5.d
# ls -al
total 8
drwxr-xr-x    2 root     root     4096 Jul 15 09:29 .
drwxr-xr-x   10 root     root     4096 Jun 21 08:52 ..
lrwxrwxrwx    1 root     root       19 Jan  1  2000 K05saslauthd ->
../init.d/saslauthd
lrwxrwxrwx    1 root     root       20 Feb  1  2003 K15postgresql ->
../init.d/postgresql
lrwxrwxrwx    1 root     root       13 Jan  1  2000 K20nfs -> ../init.d/nfs
lrwxrwxrwx    1 root     root       14 Jan  1  2000 K24irda -> ../init.d/irda
lrwxrwxrwx    1 root     root       17 Jan  1  2000 K35winbind -> ../init.d/winbind
lrwxrwxrwx    1 root     root       15 Jan  1  2000 K50snmpd -> ../init.d/snmpd
lrwxrwxrwx    1 root     root       19 Jan  1  2000 K50snmptrapd ->
../init.d/snmptrapd
lrwxrwxrwx    1 root     root       16 Jun 21 09:43 K50vsftpd -> ../init.d/vsftpd
lrwxrwxrwx    1 root     root       16 Jun 21 08:57 K73ypbind -> ../init.d/ypbind
lrwxrwxrwx    1 root     root       14 Jun 21 08:54 K74nscd -> ../init.d/nscd
lrwxrwxrwx    1 root     root       18 Feb  8 11:15 K92iptables -> ../init.d/iptables
lrwxrwxrwx    1 root     root       19 Feb  1  2003 K95firstboot ->
../init.d/firstboot
lrwxrwxrwx    1 root     root       15 Jan  1  2000 S05kudzu -> ../init.d/kudzu
lrwxrwxrwx    1 root     root       14 Jun 21 08:55 S09isdn -> ../init.d/isdn
lrwxrwxrwx    1 root     root       17 Jan  1  2000 S10network -> ../init.d/network
lrwxrwxrwx    1 root     root       16 Jan  1  2000 S12syslog -> ../init.d/syslog
lrwxrwxrwx    1 root     root       17 Jan  1  2000 S13portmap -> ../init.d/portmap
lrwxrwxrwx    1 root     root       17 Jan  1  2000 S14nfslock -> ../init.d/nfslock
lrwxrwxrwx    1 root     root       18 Jan  1  2000 S17keytable -> ../init.d/keytable
lrwxrwxrwx    1 root     root       16 Jan  1  2000 S20random -> ../init.d/random
lrwxrwxrwx    1 root     root       16 Jun 21 08:52 S24pcmcia -> ../init.d/pcmcia
lrwxrwxrwx    1 root     root       15 Jan  1  2000 S25netfs -> ../init.d/netfs
lrwxrwxrwx    1 root     root       14 Jan  1  2000 S26apmd -> ../init.d/apmd
lrwxrwxrwx    1 root     root       16 Jan  1  2000 S28autofs -> ../init.d/autofs
lrwxrwxrwx    1 root     root       14 Jan  1  2000 S55sshd -> ../init.d/sshd
lrwxrwxrwx    1 root     root       20 Jan  1  2000 S56rawdevices ->
```

```
../init.d/rawdevices
lrwxrwxrwx    1 root     root       16 Jan  1  2000 S56xinetd -> ../init.d/xinetd
lrwxrwxrwx    1 root     root       14 Feb  1  2003 S58ntpd -> ../init.d/ntpd
lrwxrwxrwx    1 root     root       13 Jun 21 10:42 S60afs -> ../init.d/afs
lrwxrwxrwx    1 root     root       13 Jan  1  2000 S60lpd -> ../init.d/lpd
lrwxrwxrwx    1 root     root       16 Feb  8 17:26 S78mysqld -> ../init.d/mysqld
lrwxrwxrwx    1 root     root       18 Jan  1  2000 S80sendmail -> ../init.d/sendmail
lrwxrwxrwx    1 root     root       13 Jan  1  2000 S85gpm -> ../init.d/gpm
lrwxrwxrwx    1 root     root       15 Mar 22 08:24 S85httpd -> ../init.d/httpd
lrwxrwxrwx    1 root     root       15 Jan  1  2000 S90crond -> ../init.d/crond
lrwxrwxrwx    1 root     root       13 Jan  1  2000 S90xfs -> ../init.d/xfs
lrwxrwxrwx    1 root     root       17 Jan  1  2000 S95anacron -> ../init.d/anacron
lrwxrwxrwx    1 root     root       13 Jan  1  2000 S95atd -> ../init.d/atd
lrwxrwxrwx    1 root     root       15 Jun 21 08:57 S97rhnsd -> ../init.d/rhnsd
lrwxrwxrwx    1 root     root       14 Jul 15 09:29 S98wine -> ../init.d/wine
lrwxrwxrwx    1 root     root       13 Feb  8 17:26 S99db2 -> ../init.d/db2
lrwxrwxrwx    1 root     root       11 Jun 21 08:52 S99local -> ../rc.local
#
```

This might seem like a fairly complicated system, but actually it offers great flexibility, because if you want to temporarily disable a service in a particular runlevel, just remove the appropriate link. However, manipulating links can become tiresome and error prone (especially if you're tired!), so there is a better way in the form of a command called `chkconfig`.

How to find out what services are enabled

To see how many services you have enabled, run this command:

```
/sbin/chkconfig --list
```

Listing 2 shows the output of this command. As you can see, there are eight columns per line.

The `chkconfig` command can also be used to switch any service off or on. See the manual page (`man chkconfig`) for further details.

chkconfig and xinetd

If you have a newer version of `chkconfig`, you will have a section at the end of the main output showing the configuration of xinetd (the Internet services daemon). This section has been removed from Listing 2 to simplify the explanation.

Listing 2. Output of chkconfig --list|sort

```
afs             0:off   1:off   2:off   3:on    4:off   5:on    6:off
anacron         0:off   1:off   2:on    3:on    4:on    5:on    6:off
apmd            0:off   1:off   2:on    3:on    4:on    5:on    6:off
atd             0:off   1:off   2:off   3:on    4:on    5:on    6:off
autofs          0:off   1:off   2:off   3:on    4:on    5:on    6:off
crond           0:off   1:off   2:on    3:on    4:on    5:on    6:off
db2             0:off   1:off   2:off   3:on    4:off   5:on    6:off
firstboot       0:off   1:off   2:off   3:off   4:off   5:off   6:off
gpm             0:off   1:off   2:on    3:on    4:on    5:on    6:off
httpd           0:off   1:off   2:off   3:off   4:off   5:on    6:off
iptables        0:off   1:off   2:off   3:off   4:off   5:off   6:off
irda            0:off   1:off   2:off   3:off   4:off   5:off   6:off
isdn            0:off   1:off   2:on    3:on    4:on    5:on    6:off
keytable        0:off   1:on    2:on    3:on    4:on    5:on    6:off
kudzu           0:off   1:off   2:off   3:on    4:on    5:on    6:off
lpd             0:off   1:off   2:on    3:on    4:on    5:on    6:off
mysqld          0:off   1:off   2:off   3:on    4:off   5:on    6:off
netfs           0:off   1:off   2:off   3:on    4:on    5:on    6:off
network         0:off   1:off   2:on    3:on    4:on    5:on    6:off
nfs             0:off   1:off   2:off   3:off   4:off   5:off   6:off
nfslock         0:off   1:off   2:off   3:on    4:on    5:on    6:off
nscd            0:off   1:off   2:off   3:off   4:off   5:off   6:off
ntpd            0:off   1:off   2:off   3:on    4:off   5:on    6:off
pcmcia          0:off   1:off   2:on    3:on    4:on    5:on    6:off
portmap         0:off   1:off   2:off   3:on    4:on    5:on    6:off
postgresql      0:off   1:off   2:off   3:off   4:off   5:off   6:off
random          0:off   1:off   2:on    3:on    4:on    5:on    6:off
```

```
rawdevices      0:off   1:off   2:off   3:on    4:on    5:on    6:off
rhnsd           0:off   1:off   2:off   3:on    4:on    5:on    6:off
saslauthd       0:off   1:off   2:off   3:off   4:off   5:off   6:off
sendmail        0:off   1:off   2:on    3:on    4:on    5:on    6:off
snmpd           0:off   1:off   2:off   3:off   4:off   5:off   6:off
snmptrapd       0:off   1:off   2:off   3:off   4:off   5:off   6:off
sshd            0:off   1:off   2:on    3:on    4:on    5:on    6:off
syslog          0:off   1:off   2:on    3:on    4:on    5:on    6:off
vsftpd          0:off   1:off   2:off   3:off   4:off   5:off   6:off
winbind         0:off   1:off   2:off   3:off   4:off   5:off   6:off
wine            0:off   1:off   2:on    3:on    4:on    5:on    6:off
xfs             0:off   1:off   2:on    3:on    4:on    5:on    6:off
xinetd          0:off   1:off   2:off   3:on    4:on    5:on    6:off
ypbind          0:off   1:off   2:off   3:off   4:off   5:off   6:off
```

The first column in Listing 2 is the name of the service, and the subsequent columns represent the runlevels and the status of the service in each runlevel. For example, the ntpd (Network time daemon) service is configured to be started only in runlevels 3 (multi-user mode without graphics) and 5 (multi-user mode with graphics), and the sshd service is switched on in runlevels 2, 3, 4, and 5.

Note that no services are started in runlevels 0 and 6. Looking back to Table 1, the reason for this is obvious. Runlevel 0 means halt or stop the system, so you would not want to *start* any services when the machine is "coming down." The same logic applies for runlevel 6.

Runlevel 1 -- "single-user mode" -- is a special runlevel that is generally only used when something goes wrong. Traditionally, the only application that runs in runlevel 1 is a shell to allow the superuser to repair damage to a system or to allow the superuser to modify the system in a safe environment. It is safe, since -- as the name "single-user mode" suggests -- only the superuser is able to access the system. Additionally, networking is disabled, so nobody can log in remotely. As Table 1 shows, the only service that runs for single-user mode is keytable, so that the superusers keyboard will work as expected.

Enabled services versus running services
Occasionally, services might fail to start for some reason, so to see which services are currently running, run this command:

```
/sbin/service --status-all
```

This command will output one or more lines per service, showing whether each service is running, and if so, it might show some service-specific output such as the PID (process id) the service is running under. The service command doesn't have a manual page, but if you run it with the `--help` option, you can get some help with its operation.

Limitations of the traditional services framework
Crucially, only when *all* the services that are configured to start have started can you log into your Linux system. Waiting for 50 services to start could take a number of *minutes*, which is time lost enjoying your Linux system.

I have found a way to speed up this process. Note that the technique does *not* involve stopping any services. However, shutting down unused services is highly prudent, not only because it increases boot up speed (as there is less to run before the machine can be logged into), but also because it reduces your exposure to security exploits, since many services run as the root user.

To recap, when a Linux system boots, it runs all the services that have been configured to start for the runlevel in a serial fashion -- *one after the other*. This can be a time-consuming operation.

Perhaps an obvious way to speed up the starting of services is to run all the services in parallel, so that they all start at the same time. Unfortunately, while this sounds highly appealing, it does not work. The reason is that there are *dependencies* between services. Linux does not make these dependencies 100 percent explicit, but they are there nonetheless. Remember the name format of the links to the service programs we discussed earlier? The two-digit numbers after the "S" or the "K" determine the order in which the links (and thus the services they refer to) are run. The numbers enforce a crude ordering, and hence *sometimes* also enforce dependencies between services.

Dependencies between services
Looking back to Listing 1, we can see that the network service (S10network) will run before the ntpd service (S58ntpd). This is as we would expect, since the ntpd service requires the network to be available so that it can contact a local time server. Unfortunately, this crude ordering does not tell us as much as we'd like and can be misleading. For example, in Listing 1 we see that the lpd service (S60lpd) would also run after the network service. Although this is correct for a Linux system that is connected to a network and uses network printers, it does not follow that the lpd service absolutely *must* run after the network service for a home system with an inkjet printer plugged in the back. Indeed, in this case, it might make more sense to start lpd

before the network.

Taking another example: the crond (cron daemon) service (S90crond in Listing 1) also runs after the network has started. However, unless you have a cron file that uses files on a remote machine, there is no reason that crond should not start before the network.

The general tendency is to "play it safe" and start all the important services first, and then run whatever is left, because the traditional method of starting services under Linux that I have just outlined is somewhat limited.

What is becoming apparent as we explore the system is that some services rely on others, whereas some services are "stand-alone" in that they are not dependent on any other service.

So, although we cannot start *all* the services in parallel, we could start all those services *that have no dependencies* in parallel. When these dependency-free services have started, we could then start all services whose dependencies have already been satisfied (in other words, those services with services they depend on having already been started). This process could be repeated until all the services had been started.

This sounds like a complicated problem, but luckily for us, a program has already been written that will solve it for us. This program is none other than `make`.

Normally associated with compiling software, `make` provides exactly the framework we need. All we need to do is tell `make` what the dependencies between services are; it does all the hard work of calculating cross-dependencies, and with its little-known `-j` flag, it will run as many "jobs" (or in our case, services) as it can *simultaneously*.

Working out dependencies between services

As I alluded to above, the traditional Linux system does not make explicit the dependencies between services, so unfortunately, it is now time to do some hard work and work out the dependencies ourselves. This will probably take a while, since you may not even know what each service is doing, let alone how it relates to other services. Unfortunately, if you don't go through this exercise, the technique will be of zero benefit to you. (As mentioned above, the hope is that if this technique is useful, the Linux distributions will adopt it and do the hard work for us.)

For now, lets take a simple example. We know that ntpd needs a network, hence it follows that the ntpd service is dependent on the network service. This is represented in `make` syntax as shown here:

```
ntpd : network
```

We can also say with certainty that the netfs service (that mounts all NFS directories we need) is dependent on a network. On my system (yours may be different), the autofs service (automatic mounting of network file systems) is also dependent on the network service, since I only ever automount remote file systems (you may mount local CD-ROMs or floppies, though). Our "dependency table" now looks like this:

```
ntpd : network
netfs : network
autofs : network
```

Understanding your services

If, when you run the command `/sbin/chkconfig --list`, you are confronted with services that you do not recognize, take the time to find out what they do. The easiest way to do this is by looking at the comments at the top of the script that controls the service. It may be that you can switch off the service if you do not need the facility. Even if you do need it, it pays to understand your system.

This doesn't look like much, but can you see what this means? It means that once the network service has started, we can then start the ntpd, netfs, and autofs services *in parallel*.

As a contrived example, imagine that all services take 10 seconds to run. With the traditional service startup method, starting the network, ntpd, netfs, and autofs services would take 40 seconds. Using this technique, it would only take 20 seconds -- a savings of 50 percent.

Why is this? Well, it takes 10 seconds to start the network service, but (because by the time the rc script is run, the machine is running in a fully multi-tasking way) the other three services can start *simultaneously*, so combined they only take 10 seconds to run *in total*.

In reality, most services probably will not take 10 seconds to start, but since each service is doing something completely different, the time to start them can vary greatly.

Sample implementation

The zip file I have provided in the Resources section contains a sample implementation of the technique outlined. It includes a modified rc script that calls the `make` command, along with sample GNU makefiles runlevel.mk, start5.mk, and stop5.mk. The runlevel.mk makefile is the controlling program, while start5.mk and stop5.mk encode the service dependencies when starting and stopping services for runlevel 5, respectively.

Note that the start and stop makefiles provided do not contain complete lists of dependencies between services. They are an example only. Note also that attempting to use these makefiles unmodified on your system will almost certainly not work, as you will probably have a different list of services from mine.

Conclusion (and a few additional considerations)
I have presented a technique for improving the speed that a Linux machine boots. The technique achieves this by allowing the latter part of the boot sequence to be run in parallel, rather than in the traditional serial fashion. The technique is aesthetically elegant and utilizes existing system tools.

The effectiveness of this technique depends on the number of services that need to be run as well as the time it takes for each service to run. The degree of parallelization possible is controlled largely by the dependencies between services. It may be that using this technique makes little improvement for some systems, while for others, it could have a dramatic impact on boot speed. This can be explained by the fact that each system has a different set of services enabled, and each of these services takes differing amounts of time to run. Once again, to use this technique, you need to establish the dependencies between the services you use *for your particular system*.

Additional considerations:

- Some service programs simply run a program in the background and then they themselves exit (in other words, the service program finishes, but the real work is still happening in the background). This points to the fact that the traditional system is deficient and that the writers of such services are attempting to shave off a few cycles while working within the confines of the existing framework. Adopting the technique described here would make the dependencies more explicit and would not require service writers to "cheat," so to speak. This technique allows for a more efficient framework to be built around the service programs.

- Using the technique outlined is not really appropriate when you wish to boot your system "interactively," since you will generally only do this when something is wrong; in this case, you probably want to run all the services serially to see which is causing a problem. However, it would be very easy to modify the system startup to allow the user to select at boot time whether they wished to boot with either "serial" (allowing interactive service startup) or "parallel" service startup.

- Adopting this technique might involve further thought, since if both traditional and new systems are provided to users, two sets of information on how services start up would need to be maintained in sync (the ordered rc.d/ link files and the runlevel make files). A better solution for Linux distributions might be to autogenerate the link files from the makefiles, since the makefiles encode more information about the services than the link files do.

- This system might not be suitable for a professional server for which, if a service fails, the administrator wants to see this failure by viewing the console as soon as it happens. However, for the average end-user system, the parallelization technique can dramatically improve boot-up speed while still allowing the user to see if any problems occurred.

- Interestingly, although the technique I have outlined is not "Linux-like" in the traditional sense, the Linux Standards Base (LSB) does not appear to specify the order in which the init.d scripts are run, so it may be that this technique could be adopted by Linux distribution vendors and still allow them to be LSB-compliant. This would be a boon for users, as mentioned earlier, since the distribution vendors could calculate all the package dependencies for us.

- It is possible that a more aggressive approach could be taken by modifying the "action field" in the /etc/inittab file to be "once" rather than "wait". This could allow the user to log in even *before* the services have finished executing. However, this is beyond the scope of this article. View `man inittab` for further details, and remember, UML is your friend.

Resources

- Download a zip file containing a sample implementation of the technique outlined this article.

- The official GNU Make homepage contains make downloads and links to make documentation.

- The LSB (Linux Standard Base) aims to define explicitly what constitutes a Linux system. The LSB common document covers system "System Initialization."

- The UML (User Mode Linux) homepage contains links to UML documentation and downloads, to allow you to run a "virtual" linux system on your real Linux system.

- For a thorough guide to getting started with UML, read the tutorial "Introduction to User-Mode Linux" (*developerWorks*,

January 2003).

- Find more resources for Linux developers in the *developerWorks* Linux zone.

About the author
James Hunt is a software engineer at IBM Hursley in the United Kingdom. He is not a racing driver and his car is definitely not of the racing variety. James is a self-confessed Linux fanatic and has spent many happy hours preaching on the subject to anyone within earshot. He works on the WebSphere MQ product on distributed platforms (AIX, HP-UX, Linux, OS/400, Solaris, and Windows). His background is in UNIX system administration and database administration and programming. His Linux interests include file systems and compilers, and he is the author of possibly the only Perl Coding Standards document in existence. James is one of those strange people who actually enjoys writing technical documentation. In his spare time he plays guitar, dabbles in Yoga, and sails dinghies, although he has not yet managed to parallelize these activities. You can reach James at jameshunt-at-uk.ibm.com.

Code e-mail it!

What do you think of this document?

Killer! (5) Good stuff (4) So-so; not bad (3) Needs work (2) Lame! (1)

Comments?

Submit feedback