The Linux BootPrompt–HowTo

Table of Contents

The Linux BootPrompt−HowTo....1
by Paul Gortmaker....1
1. Introduction....1
2. Overview of Boot Prompt Arguments....1
3. General Non−Device Specific Boot Args....1
4. Boot Arguments to Control PCI Bus Behaviour (`pci=')....1
5. Boot Arguments for Video Frame Buffer Drivers....2
6. Boot Arguments for SCSI Peripherals....2
7. Hard Disks....2
8. The Sound Drivers....2
9. CD−ROMs (Non−SCSI/ATAPI/IDE)....2
10. Serial and ISDN Drivers....2
11. Other Hardware Devices....2
12. Copying, Translations, Closing, etc....2
1. Introduction....3
1.1 Intended Audience and Applicability....3
1.2 Related Documentation....4
1.3 New Versions of this Document....4
2. Overview of Boot Prompt Arguments....4
2.1 LILO (LInux LOader)....5
2.2 LoadLin....5
2.3 The ``rdev'' utility....6
2.4 How the Kernel Sorts the Arguments....6
2.5 Setting Environment Variables....6
2.6 Passing Arguments to the `init' program....7
3. General Non−Device Specific Boot Args....7
3.1 Root Filesystem options....7
The `root=' Argument....7
The `rootflags=' Argument....8
The `rootfstype=' Argument....8
The `ro' Argument....8
The `rw' Argument....8
The `nfsroot=' Argument....8
The `ip=' or `nfsaddrs=' Argument....8
3.2 Options Relating to RAM Disk Management....8
The `ramdisk_start=' Argument....9
The `load_ramdisk=' Argument....9
The `prompt_ramdisk=' Argument....9
The `ramdisk_size=' Argument....9
The `ramdisk_blocksize=' Argument....10
The `ramdisk=' Argument (obsolete)....10
The `noinitrd' (initial RAM disk) Argument....10
3.3 Boot Arguments Related to Memory Handling....10
The `cachesize=' Argument....10
The `mem=' Argument....11
The `memfrac=' Argument....11
The `swap=' Argument....11
The `buff=' Argument....12

Table of Contents

The Linux BootPrompt−HowTo
3.4 Other Misc. Kernel Boot Arguments....12
The `acpi=' Argument....12
The `console=' Argument....12
The `debug' Argument....12
The `decnet=' Argument....13
The `devfs=' Argument....13
The `gpt' Argument....13
The `idle=' Argument....13
The `init=' Argument....13
The `isapnp=' Argument....13
The `isapnp_reserve_dma=' Argument....13
The `isapnp_reserve_io=' Argument....14
The `isapnp_reserve_irq=' Argument....14
The `isapnp_reserve_mem=' Argument....14
The `kbd−reset' Argument....14
The `lockd.udpport=' and `lockd.tcpport' Argument....14
The `maxcpus=' Argument....14
The `mca−pentium' Argument....14
The `md=' Argument....14
The `nmi_watchdog=' Argument....15
The `no387' Argument....15
The `no−hlt' Argument....15
The `no−scroll' Argument....15
The `noapic' Argument....15
The `noht' Argument....15
The `noisapnp' Argument....16
The `nomce' Argument....16
The `nosmp' Argument....16
The `noresume' Argument....16
The `notsc' Argument....16
The `nofxsr" Argument....16
The `panic=' Argument....16
The `pirq=' Argument....16
The `profile=' Argument....17
The `quiet' Argument....17
The `raid=' Argument....17
The `reboot=' Argument....17
The `reserve=' Argument....17
The `resume=' Argument....18
The `vga=' Argument....18
4. Boot Arguments to Control PCI Bus Behaviour (`pci=')....18
4.1 The `pci=assign−busses' Argument....18
4.2 The `pci=bios' and `pci=nobios' Arguments....18
4.3 The `pci=conf1' and `pci=conf2' Arguments....19
4.4 The `pci=irqmask=' Argument....19
4.5 The `pci=lastbus=' Argument....19
4.6 The `pci=noacpi' Argument....19

Table of Contents

The Linux BootPrompt−HowTo

4.7 The `pci=nopeer' Argument....19
4.8 The `pci=nosort' Argument....19
4.9 The `pci=off' Argument....19
4.10 The `pci=usepirqmask' Argument....19
4.11 The `pci=rom' Argument....20
5. Boot Arguments for Video Frame Buffer Drivers....20
5.1 The `video=map:...' Argument....20
5.2 The `video=scrollback:...' Argument....20
5.3 The `video=vc:...' Argument....21
6. Boot Arguments for SCSI Peripherals....21
6.1 Arguments for Upper and Mid−level Drivers....21
Maximum Probed LUNs (`max_scsi_luns=')....21
SCSI Logging (`scsi_logging=')....21
Parameters for the SCSI Tape Driver (`st=')....21
6.2 Arguments for SCSI Host Adapter Drivers....22
7. Hard Disks....23
7.1 IDE Disk/CD−ROM Driver Parameters....23
7.2 Old MFM/RLL/Standard ST−506 Disk Driver Options (`hd=')....24
7.3 XT Disk Driver Options (`xd=', `xd_geo=')....24
8. The Sound Drivers....25
8.1 Individual Sound Device Driver Arguments....25
ALSA ISA drivers....25
OSS drivers....26
ALSA PCI Drivers....27
9. CD−ROMs (Non−SCSI/ATAPI/IDE)....28
9.1 Old CD−ROM Driver Arguments....28
10. Serial and ISDN Drivers....28
10.1 The ISDN drivers....28
10.2 The Serial drivers....29
11. Other Hardware Devices....29
11.1 Ethernet Devices (`ether=', `netdev=')....29
11.2 The Floppy Disk Driver (`floppy=')....29
11.3 The Bus Mouse Driver (`bmouse=')....30
11.4 The MS Bus Mouse Driver (`msmouse=')....30
11.5 The Printer Driver (`lp=')....30
11.6 The Parallel port IP driver (`plip=')....30
12. Copying, Translations, Closing, etc....31
12.1 Copyright and Disclaimer....31
12.2 Closing....31

The Linux BootPrompt–HowTo

by Paul Gortmaker.

v1.4, Mar 21, 2003

This is the BootPrompt–Howto, which is a compilation of all the possible boot time arguments that can be passed to the Linux kernel at boot time. A discussion of how the kernel sorts boot time arguments, along with an overview of some of the popular software used to boot Linux kernels is also included.

1. Introduction

- 1.1 Intended Audience and Applicability
- 1.2 Related Documentation
- 1.3 New Versions of this Document

2. Overview of Boot Prompt Arguments

- 2.1 LILO (LInux LOader)
- 2.2 LoadLin
- 2.3 The ``rdev'' utility
- 2.4 How the Kernel Sorts the Arguments
- 2.5 Setting Environment Variables.
- 2.6 Passing Arguments to the `init' program

3. General Non–Device Specific Boot Args

- 3.1 Root Filesystem options
- 3.2 Options Relating to RAM Disk Management
- 3.3 Boot Arguments Related to Memory Handling
- 3.4 Other Misc. Kernel Boot Arguments

4. Boot Arguments to Control PCI Bus Behaviour (`pci=')

- 4.1 The `pci=assign–busses' Argument
- 4.2 The `pci=bios' and `pci=nobios' Arguments
- 4.3 The `pci=conf1' and `pci=conf2' Arguments
- 4.4 The `pci=irqmask=' Argument
- 4.5 The `pci=lastbus=' Argument
- 4.6 The `pci=noacpi' Argument
- 4.7 The `pci=nopeer' Argument
- 4.8 The `pci=nosort' Argument
- 4.9 The `pci=off' Argument
- 4.10 The `pci=usepirqmask' Argument
- 4.11 The `pci=rom' Argument

5. Boot Arguments for Video Frame Buffer Drivers

- 5.1 The `video=map:...' Argument
- 5.2 The `video=scrollback:...' Argument
- 5.3 The `video=vc:...' Argument

6. Boot Arguments for SCSI Peripherals.

- 6.1 Arguments for Upper and Mid–level Drivers
- 6.2 Arguments for SCSI Host Adapter Drivers

7. Hard Disks

- 7.1 IDE Disk/CD–ROM Driver Parameters
- 7.2 Old MFM/RLL/Standard ST–506 Disk Driver Options (`hd=')
- 7.3 XT Disk Driver Options (`xd=', `xd_geo=')

8. The Sound Drivers

- 8.1 Individual Sound Device Driver Arguments

9. CD–ROMs (Non–SCSI/ATAPI/IDE)

- 9.1 Old CD–ROM Driver Arguments

10. Serial and ISDN Drivers

- 10.1 The ISDN drivers
- 10.2 The Serial drivers

11. Other Hardware Devices

- 11.1 Ethernet Devices (`ether=', `netdev=')
- 11.2 The Floppy Disk Driver (`floppy=')
- 11.3 The Bus Mouse Driver (`bmouse=')
- 11.4 The MS Bus Mouse Driver (`msmouse=')
- 11.5 The Printer Driver (`lp=')
- 11.6 The Parallel port IP driver (`plip=')

12. Copying, Translations, Closing, etc.

- 12.1 Copyright and Disclaimer
- 12.2 Closing

---

1. Introduction

The kernel has the capability to accept information at boot in the form of a `command line', similar to an argument list you would give to a program. In general this is used to supply the kernel with information about hardware parameters that the kernel would not be able to determine on its own, or to avoid/override the values that the kernel would otherwise detect.

It is the job of the boot loader (e.g. LILO, loadlin or Grub) to take this information from the user and put it in a previously agreed upon place where the kernel can find it once it starts.

This present revision covers kernels up to and including v2.4.20. and v2.5.63

The BootPrompt−Howto is by:

Paul Gortmaker, p_gortmaker @ yahoo.com

This document is Copyright (c) 1995−2003 by Paul Gortmaker. Please see the Disclaimer and Copying information at the end of this document (copyright) for information about redistribution of this document and the usual `we are not responsible for what you manage to break...' type legal stuff.

1.1 Intended Audience and Applicability

Most Linux users should never have to even look at this document. Linux does an exceptionally good job at detecting most hardware and picking reasonable default settings for most parameters. The information in this document is aimed at users who might want to change some of the default settings to optimize the kernel to their particular machine, or to a user who has `rolled their own' kernel to support a not so common piece of hardware for which the automatic defaults are not optimal.

For the sake of this document it is best to break the boot arguments into two general categories; (a)ones handled by the kernel and (b)those being handled by a device driver. Examples would be `init=` which tells the kernel what the first program to run should be, versus `aha154x=` which tells a device driver for a SCSI card what hardware resources it should use are. This document concentrates on giving detailed information on those in (a) for reasons outlined below.

IMPORTANT NOTE: Driver related boot prompt arguments only apply to hardware drivers that are compiled directly into the kernel. They have *no effect* on drivers that are loaded as modules. Most Linux distributions come with a basic `bare−bones' kernel, and the drivers are small modules that are loaded after the kernel has initialized. If you are unsure if you are using modules then try `lsmod`, look at `man depmod` and `man modprobe` along with the contents of your `/etc/modules.conf`.

In light of this, device driver boot prompt arguments are only really used by a few people who are building their own kernels, and thus have the kernel source at hand. These people are usually going to check the source for the options and syntax required by that driver to get the most up to date info.

For example, if you were looking for what arguments could be passed to the AHA1542 SCSI driver, then you would go to the `linux/drivers/scsi` directory, and look in the file `aha1542.c` for `__setup(... , ...)`. The first thing in brackets is the argument you provide at boot, and the second thing is the name of the function that processes your argument. Usually near the top of this function or at the top of the source file you will find a description of the boot time arguments that the driver accepts.

1.2 Related Documentation

For a while now, the kernel source has come with the file `linux/Documentation/kernel−parameters.txt`. This file contains a brief listing of all the boot time arguments that you can provide, along with quick pointers to where in the source you can find where the arguments are parsed. The idea is that this file gives developers a quick and easy place to add in a brief description of any new arguments that they add while working on the source. As such, it will probably always be more up to date than this document. Actually, I'm considering discontinuing this document in light of the existence of `kernel−parameters.txt`. (Opinions?)

The `linux` directory is usually found in `/usr/src/` for most distributions. All references in this document to files that come with the kernel will have their pathname abbreviated to start with `linux` − you will have to add the `/usr/src/` or whatever is appropriate for your system. Some distributions may not install the full kernel source by default, and only put in the `linux/include` directory. If you can't find the file in question, then install the kernel source and/or make use of the `find` and `locate` commands. If you can't find the kernel source package in your distribution then the kernel source is available at:

Kernel Source Home

The next best thing to reading the kernel C source itself, will be any of the other documentation files that are distributed with the kernel itself. There are now quite a few of these, and most of them can be found in the directory `linux/Documentation` and subdirectories from there. Sometimes there will be `README.foo` files that can be found in the related driver directory (e.g. `linux/drivers/???/`, where examples of `???` could be `scsi`, `char`, or `net`). The general trend is to move these files into the Documentation directory, so if a file mentioned in this document is no longer there, chances are it has been moved.

If you have figured out what boot−args you intend to use, and now want to know how to get that information to the kernel, then look at the documentation that comes with the software that you use to boot the kernel (e.g. LILO or loadlin). A brief overview is given below, but it is no substitute for the documentation that comes with the booting software.

1.3 New Versions of this Document

New versions of this document can be retrieved via anonymous FTP from most Linux FTP sites in the directory `/pub/Linux/docs/HOWTO/`. Updates will be made as new information and/or drivers becomes available. If this copy that you are presently reading is more than six months old, then you should probably check to see if a newer copy exists. I would recommend viewing this via a WWW browser or in the Postscript/dvi format. Both of these contain cross−references that are lost in a simple plain text version.

If you want to get the official copy, here is URL.

BootPrompt−HOWTO

---

2. Overview of Boot Prompt Arguments

This section gives some examples of software that can be used to pass kernel boot−time arguments to the kernel itself. It also gives you an idea of how the arguments are processed, what limitations there are on the boot args, and how they filter down to each appropriate device that they are intended for.

It is *important* to note that spaces should *not* be used in a boot argument, but only between separate arguments. A list of values that are for a single argument are to be separated with a comma between the values, and again without any spaces. See the following examples below.

```
ether=9,0x300,0xd0000,0xd4000,eth0  root=/dev/hda1            *RIGHT*
ether = 9, 0x300, 0xd0000, 0xd4000, eth0  root = /dev/hda1    *WRONG*
```

Once the Linux kernel is up and running, one can view the command line arguments that were in place at boot by simply typing `cat /proc/cmdline` at a shell prompt.

2.1 LILO (LInux LOader)

The LILO program (LInux LOader) written by Werner Almesberger is the most commonly used. It has the ability to boot various kernels, and stores the configuration information in a plain text file. Most distributions ship with LILO as the default boot–loader. LILO can boot DOS, OS/2, Linux, FreeBSD, etc. without any difficulties, and is quite flexible.

A typical configuration will have LILO stop and print `LILO:` shortly after you turn on your computer. It will then wait for a few seconds for any optional input from the user, and failing that it will then boot the default system. Typical system labels that people use in the LILO configuration files are `linux` and `backup` and `msdos`. If you want to type in a boot argument, you type it in here, after typing in the system label that you want LILO to boot from, as shown in the example below.

```
LILO: linux root=/dev/hda1
```

LILO comes with excellent documentation, and for the purposes of boot args discussed here, the LILO `append=` command is of significant importance when one wants to add a boot time argument as a permanent addition to the LILO config file. You simply add something like `append = "foo=bar"` to the `/etc/lilo.conf` file. It can either be added at the top of the config file, making it apply to all sections, or to a single system section by adding it inside an `image=` section. Please see the LILO documentation for a more complete description.

2.2 LoadLin

The other commonly used Linux loader is `LoadLin' which is a DOS program that has the capability to launch a Linux kernel from the DOS prompt (with boot–args) assuming that certain resources are available. This is good for people that use DOS and want to launch into Linux from DOS.

It is also very useful if you have certain hardware which relies on the supplied DOS driver to put the hardware into a known state. A common example is `SoundBlaster Compatible' sound cards that require the DOS driver to set a few proprietary registers to put the card into a SB compatible mode. Booting DOS with the supplied driver, and then loading Linux from the DOS prompt with `LOADLIN.EXE` avoids the reset of the card that happens if one rebooted instead. Thus the card is left in a SB compatible mode and hence is useable under Linux.

There are also other programs that can be used to boot Linux. For a complete list, please look at the programs available on your local Linux ftp mirror, under `system/Linux-boot/`.

2.3 The ``rdev'' utility

There are a few of the kernel boot parameters that have their default values stored in various bytes in the kernel image itself. There is a utility called `rdev` that is installed on most systems that knows where these values are, and how to change them. It can also change things that have no kernel boot argument equivalent, such as the default video mode used.

The rdev utility is usually also aliased to swapdev, ramsize, vidmode and rootflags. These are the five things that rdev can change, those being the root device, the swap device, the RAM disk parameters, the default video mode, and the readonly/readwrite setting of root device.

More information on `rdev` can be found by typing `rdev -h` or by reading the supplied man page (`man rdev`).

2.4 How the Kernel Sorts the Arguments

Most of the boot args take the form of:

```
name[=value_1][,value_2]...[,value_11]
```

where `name' is a unique keyword that is used to identify what part of the kernel the associated values (if any) are to be given to. Multiple boot args are just a space separated list of the above format. Note the limit of 11 is real, as the present code only handles 11 comma separated parameters per keyword. (However, you can re–use the same keyword with up to an additional 11 parameters in unusually complicated situations, assuming the setup function supports it.) Also note that the kernel splits the list into a maximum of ten integer arguments, and a following string, so you can't really supply 11 integers unless you convert the 11th arg from a string to an int in the driver itself.

Most of the sorting goes on in `linux/init/main.c`. First, the kernel checks to see if the argument is any of the special arguments `root=', `ro', `rw', or `debug'. The meaning of these special arguments is described further on in the document.

Then it walks a list of setup functions (contained in the `bootsetups` array) to see if the specified argument string (such as `foo') has been associated with a setup function (`foo_setup()`) for a particular device or part of the kernel. If you passed the kernel the line `foo=3,4,5,6,bar` then the kernel would search the `bootsetups` array to see if `foo' was registered. If it was, then it would call the setup function associated with `foo' (`foo_setup()`) and hand it the integer arguments 3, 4, 5 and 6 as given on the kernel command line, and also hand it the string argument `bar`.

2.5 Setting Environment Variables.

Anything of the form `foo=bar' that is not accepted as a setup function as described above is then interpreted as an environment variable to be set. An example would be to use `TERM=vt100` or `BOOT_IMAGE=vmlinuz.bak` as a boot argument. These environment variables are typically tested for in the initialization scripts to enable or disable a wide range of things.

2.6 Passing Arguments to the `init' program

Any remaining arguments that were not picked up by the kernel and were not interpreted as environment variables are then passed onto process one, which is usually the `init` program. The most common argument that is passed to the `init` process is the word *single* which instructs `init` to boot the computer in single user mode, and not launch all the usual daemons. Check the manual page for the version of `init` installed on your system to see what arguments it accepts.

---

3. General Non−Device Specific Boot Args

These are the boot arguments that are not related to any specific device or peripheral. They are instead related to certain internal kernel parameters, such as memory handling, ramdisk handling, root file system handling and others.

3.1 Root Filesystem options

The following options all pertain to how the kernel selects and handles the root filesystem.

The `root=' Argument

This argument tells the kernel what device is to be used as the root filesystem while booting. The default of this setting is the value of the root device of the system that the kernel was built on. For example, if the kernel in question was built on a system that used `/dev/hda1' as the root partition, then the default root device would be `/dev/hda1'. To override this default value, and select the second floppy drive as the root device, one would use `root=/dev/fd1'.

Valid root devices are any of the following devices:

(1) /dev/hdaN to /dev/hddN, which is partition N on ST−506 compatible disk `a to d'.

(2) /dev/sdaN to /dev/sdeN, which is partition N on SCSI compatible disk `a to e'.

(3) /dev/xdaN to /dev/xdbN, which is partition N on XT compatible disk `a to b'.

(4) /dev/fdN, which is floppy disk drive number N. Having N=0 would be the DOS `A:' drive, and N=1 would be `B:'.

(5) /dev/nfs, which is not really a device, but rather a flag to tell the kernel to get the root fs via the network.

(6) /dev/ram, which is the RAM disk.

The more awkward and less portable numeric specification of the above possible disk devices in major/minor format is also accepted. (e.g. /dev/sda3 is major 8, minor 3, so you could use `root=0x803` as an alternative.)

This is one of the few kernel boot arguments that has its default stored in the kernel image, and which can thus be altered with the `rdev` utility.

The `rootflags=' Argument

This option allows you to give options pertaining to the mounting of the root filesystem just as you would to the `mount` program. An example could be giving the `noatime` option to an ext2 fs.

The `rootfstype=' Argument

This option allows you to give a comma separated list of fs types that will be tried for a match when trying to mount the root filesystem. This list will be used instead of the internal default which usually starts with ext2, minix and the like.

The `ro' Argument

When the kernel boots, it needs a root filesystem to read basic things off of. This is the root filesystem that is mounted at boot. However, if the root filesystem is mounted with write access, you can not reliably check the filesystem integrity with half–written files in progress. The `ro' option tells the kernel to mount the root filesystem as `readonly' so that any filesystem consistency check programs (fsck) can safely assume that there are no half–written files in progress while performing the check. No programs or processes can write to files on the filesystem in question until it is `remounted' as read/write capable.

This is one of the few kernel boot arguments that has its default stored in the kernel image, and which can thus be altered with the `rdev` utility.

The `rw' Argument

This is the exact opposite of the above, in that it tells the kernel to mount the root filesystem as read/write. The default is to mount the root filesystem as read only. Do not run any `fsck' type programs on a filesystem that is mounted read/write.

The same value stored in the image file mentioned above is also used for this parameter, accessible via `rdev`.

The `nfsroot=' Argument

This argument tells the kernel which machine, what directory and what NFS options to use for the root filesystem. Also note that the argument `root=/dev/nfs` is required. Detailed information on using an NFS root fs is in the file `linux/Documentation/nfsroot.txt`.

The `ip=' or `nfsaddrs=' Argument

If you are using NFS as a root filesystem, then there is no programs like `ifconfig` and `route` present until the root fs is mounted, and so the kernel has to configure the network interfaces directly. This boot argument sets up the various network interface addresses that are required to communicate over the network. If this argument is not given, then the kernel tries to use RARP and/or BOOTP to figure out these parameters.

3.2 Options Relating to RAM Disk Management

The following options all relate to how the kernel handles the RAM disk device, which is usually used for bootstrapping machines during the install phase, or for machines with modular drivers that need to be installed to access the root filesystem.

The `ramdisk_start=' Argument

To allow a kernel image to reside on a floppy disk along with a compressed ramdisk image, the `ramdisk_start=<offset>' command was added. The kernel can't be included into the compressed ramdisk filesystem image, because it needs to be stored starting at block zero so that the BIOS can load the bootsector and then the kernel can bootstrap itself to get going.

Note: If you are using an uncompressed ramdisk image, then the kernel can be a part of the filesystem image that is being loaded into the ramdisk, and the floppy can be booted with LILO, or the two can be separate as is done for the compressed images.

If you are using a two–disk boot/root setup (kernel on disk 1, ramdisk image on disk 2) then the ramdisk would start at block zero, and an offset of zero would be used. Since this is the default value, you would not need to actually use the command at all.

The `load_ramdisk=' Argument

This parameter tells the kernel whether it is to try to load a ramdisk image or not. Specifying `load_ramdisk=1' will tell the kernel to load a floppy into the ramdisk. The default value is zero, meaning that the kernel should not try to load a ramdisk.

Please see the file `linux/Documentation/ramdisk.txt` for a complete description of the new boot time arguments, and how to use them. A description of how this parameter can be set and stored in the kernel image via `rdev' is also described.

The `prompt_ramdisk=' Argument

This parameter tells the kernel whether or not to give you a prompt asking you to insert the floppy containing the ramdisk image. In a single floppy configuration the ramdisk image is on the same floppy as the kernel that just finished loading/booting and so a prompt is not needed. In this case one can use `prompt_ramdisk=0'. In a two floppy configuration, you will need the chance to switch disks, and thus `prompt_ramdisk=1' can be used. Since this is the default value, it doesn't really need to be specified. ((Historical note: Sneaky people used to use the `vga=ask' LILO option to temporarily pause the boot process and allow a chance to switch from boot to root floppy.)

Please see the file `linux/Documentation/ramdisk.txt` for a complete description of the new boot time arguments, and how to use them. A description of how this parameter can be set and stored in the kernel image via `rdev' is also described.

The `ramdisk_size=' Argument

While it is true that the ramdisk grows dynamically as required, there is an upper bound on its size so that it doesn't consume all available RAM and leave you in a mess. The default is 4096 (i.e. 4MB) which should be large enough for most needs. You can override the default to a bigger or smaller size with this boot argument.

Please see the file `linux/Documentation/ramdisk.txt` for a complete description of the new boot time arguments, and how to use them. A description of how this parameter can be set and stored in the kernel image via `rdev' is also described.

The `ramdisk_blocksize=' Argument

This can be tuned for better memory management behaviour. Quoting from the ramdisk driver `rd.c`:

It would be very desirable to have a soft−blocksize (that in the case of the ramdisk driver is also the hardblocksize ;) of PAGE_SIZE because doing that we'll achieve a far better MM footprint. Using a rd_blocksize of BLOCK_SIZE in the worst case we'll make PAGE_SIZE/BLOCK_SIZE buffer−pages unfreeable. With a rd_blocksize of PAGE_SIZE instead we are sure that only 1 page will be protected. Depending on the size of the ramdisk you may want to change the ramdisk blocksize to achieve a better or worse MM behaviour. The default is still BLOCK_SIZE (needed by rd_load_image that supposes the filesystem in the image uses a BLOCK_SIZE blocksize)

The `ramdisk=' Argument (obsolete)

(NOTE: This argument is obsolete, and should not be used except on kernels v1.3.47 and older. The commands that should be used for the ramdisk device are documented above. Newer kernels may accept this as an alias for `ramdisk_size`.)

This specifies the size in kB of the RAM disk device. For example, if one wished to have a root filesystem on a 1.44MB floppy loaded into the RAM disk device, they would use:

```
ramdisk=1440
```

This is one of the few kernel boot arguments that has its default stored in the kernel image, and which can thus be altered with the `rdev` utility.

The `noinitrd' (initial RAM disk) Argument

The v2.x and newer kernels have a feature where the root filesystem can be initially a RAM disk, and the kernel executes `/linuxrc` on that RAM image. This feature is typically used to allow loading of modules needed to mount the real root filesystem (e.g. load the SCSI driver modules stored in the RAM disk image, and then mount the real root filesystem on a SCSI disk.)

The actual `noinitrd' argument determines what happens to the initrd data after the kernel has booted. When specified, instead of converting it to a RAM disk, it is accessible via `/dev/initrd`, which can be read once before the RAM is released back to the system. For full details on using the initial RAM disk, please consult `linux/Documentation/initrd.txt`. In addition, the most recent versions of `LILO` and `LOADLIN` should have additional useful information.

3.3 Boot Arguments Related to Memory Handling

The following arguments alter how Linux detects or handles the physical and virtual memory of your system.

The `cachesize=' Argument

Override level 2 CPU cache size detection (in kB). Sometimes CPU hardware bugs make them report the cache size incorrectly. The kernel will attempt work arounds to fix known problems, but for some CPUs it is not possible to determine what the correct size should be. This option provides an override for these situations.

The `mem=' Argument

This argument has several purposes: The original purpose was to specify the amount of installed memory (or a value less than that if you wanted to limit the amount of memory available to linux).

The next (and hardly used) purpose is to specify `mem=nopentium` which tells the Linux kernel to not use the 4MB page table performance feature. If you want to use it for both purposes, use a separate `mem=` for each one.

The original BIOS call defined in the PC specification that returns the amount of installed memory was only designed to be able to report up to 64MB. (Yes, another lack of foresight, just like the 1024 cylinder disks... sigh.) Linux uses this BIOS call at boot to determine how much memory is installed. A newer specification (e820) allows the BIOS to get this right on most machines nowadays. If you have more than 64MB of RAM installed on an older machine, you can use this boot argument to tell Linux how much memory you have. Here is a quote from Linus on the usage of the `mem=` parameter.

``The kernel will accept any `mem=xx' parameter you give it, and if it turns out that you lied to it, it will crash horribly sooner or later. The parameter indicates the highest addressable RAM address, so `mem=0x1000000' means you have 16MB of memory, for example. For a 96MB machine this would be `mem=0x6000000'. If you tell Linux that it has more memory than it actually does have, bad things will happen: maybe not at once, but surely eventually."

Note that the argument does not have to be in hex, and the suffixes `k' and `M' (case insensitive) can be used to specify kilobytes and Megabytes, respectively. (A `k' will cause a 10 bit shift on your value, and a `M' will cause a 20 bit shift.) A typical example for a 128MB machine would be "`mem=128m`".

In some cases, the memory reported via e820 can also be wrong, and so the `mem=exactmap` was added. You use this in conjunction with specifying an exact memory map, such as:

```
mem=exactmap mem=640K@0 mem=1023M@1M
```

for a 1GB machine with the usual 384k of ISA memory mapped I/O space excluded from use.

The `memfrac=' Argument

Memory is broken down into zones; on i386 these zones correspond to `DMA' (for legacy ISA devices that can only address up to 16MB via DMA); `Normal' for memory from 16MB up to 1GB, and `HighMem' for memory beyond 1GB (assuming your kernel was built with high mem support enabled). The two (or three) integers supplied here determine how much memory in each zone should be kept free – with the size of the zone divided by the number supplied being used as the minimum (so smaller numbers mean keep more free in the zone). The defaults are currently `memfrac=32,128,128`.

The `swap=' Argument

This allows the user to tune some of the virtual memory (VM) parameters that are related to swapping to disk. It accepts the following eight parameters:

```
MAX_PAGE_AGE
PAGE_ADVANCE
PAGE_DECLINE
```

```
PAGE_INITIAL_AGE
AGE_CLUSTER_FRACT
AGE_CLUSTER_MIN
PAGEOUT_WEIGHT
BUFFEROUT_WEIGHT
```

Interested hackers are advised to have a read of `linux/mm/swap.c` and also make note of the goodies in `/proc/sys/vm`. Kernels come with some useful documentation on this in the `linux/Documentation/vm/` directory.

The `buff=' Argument

Similar to the `swap=' argument, this allows the user to tune some of the parameters related to buffer memory management. It accepts the following six parameters:

```
MAX_BUFF_AGE
BUFF_ADVANCE
BUFF_DECLINE
BUFF_INITIAL_AGE
BUFFEROUT_WEIGHT
BUFFERMEM_GRACE
```

Interested hackers are advised to have a read of `linux/mm/swap.c` and also make note of the goodies in `/proc/sys/vm`. Kernels come with some useful documentation on this in the `linux/Documentation/vm/` directory.

3.4 Other Misc. Kernel Boot Arguments

These various boot arguments let the user tune certain internal kernel parameters.

The `acpi=' Argument

Currently this only accepts `off' to disable the ACPI subsystem.

The `console=' Argument

Usually the console is the 1st virtual terminal, and so boot messages appear on your VGA screen. Sometimes it is nice to be able to use another device like a serial port (or even a printer!) to be the console when no video device is present. It is also useful to capture boot time messages if a problem stops progress before they can be logged to disk. An example would be to use `console=ttyS1,9600` for selecting the 2nd serial port at 9600 baud to be the console. More information can be found in `linux/Documentation/serial-console.txt`.

The `debug' Argument

The kernel communicates important (and not–so important) messages to the operator via the `printk()` function. If the message is considered important, the `printk()` function will put a copy on the present console as well as handing it off to the `klogd()` facility so that it gets logged to disk. The reason for printing important messages to the console as well as logging them to disk is because under unfortunate circumstances (e.g. a disk failure) the message won't make it to disk and will be lost.

The threshold for what is and what isn't considered important is set by the `console_loglevel` variable. The default is to log anything more important than `DEBUG` (level 7) to the console. (These levels are defined in the include file `kernel.h`) Specifying `debug` as a boot argument will set the console loglevel to 10, so that *all* kernel messages appear on the console.

The console loglevel can usually also be set at run time via an option to the `klogd()` program. Check the man page for the version installed on your system to see how to do this.

The `decnet=' Argument

If you are using DECnet, you can supply two comma separated integers here to give your area and node respectively.

The `devfs=' Argument

If you are using devfs, instead of the standard static devices in `/dev/` then you can supply the words `only` or `mount` with this argument. There are also additional debug arguments that are listed in the source.

The `gpt' Argument

If you are using EFI GUID Partition Table handling, you can use this to override problems associated with an invalid PMBR.

The `idle=' Argument

Setting this to `poll' causes the idle loop in the kernel to poll on the need reschedule flag instead of waiting for an interrupt to happen. This can result in an improvement in performance on SMP systems (albeit at the cost of an increase in power consumption).

The `init=' Argument

The kernel defaults to starting the `init' program at boot, which then takes care of setting up the computer for users via launching getty programs, running `rc' scripts and the like. The kernel first looks for `/sbin/init`, then `/etc/init` (depreciated), and as a last resort, it will try to use `/bin/sh` (possibly on `/etc/rc`). If for example, your init program got corrupted and thus stopped you from being able to boot, you could simply use the boot prompt `init=/bin/sh` which would drop you directly into a shell at boot, allowing you to replace the corrupted program.

The `isapnp=' Argument

This takes the form of: `isapnp=read_port,reset,skip_pci_scan,verbose`

The `isapnp_reserve_dma=' Argument

This takes the form of: `isapnp_reserve_dma=n1,n2,n3,...nN` where n1 ... nN are the DMA channel numbers to not use for PnP.

The `isapnp_reserve_io=' Argument

This takes the form of: `isapnp_reserve_irq=io1,size1,io2,size2,...ioN,sizeN` where ioX,sizeX are I/O start and length pairs of regions in I/O space that are not to be used by PnP.

The `isapnp_reserve_irq=' Argument

This takes the form of: `isapnp_reserve_irq=n1,n2,n3,...nN` where n1 ... nN are the interrupt numbers to not use for PnP.

The `isapnp_reserve_mem=' Argument

This takes the form of: `isapnp_reserve_mem=mem1,size1,mem2,size2,...memN,sizeN` where ioX,sizeX are I/O start and length pairs of regions in memory space that are not to be used by PnP.

The `kbd–reset' Argument

Normally on i386 based machines, the Linux kernel does not reset the keyboard controller at boot, since the BIOS is supposed to do this. But as usual, not all machines do what they should. Supplying this option may help if you are having problems with your keyboard behaviour. It simply forces a reset at initialization time. (Some have argued that this should be the default behaviour anyways).

The `lockd.udpport=' and `lockd.tcpport' Argument

These tell the kernel to use the given port numbers for NFS lockd operation (for either UDP or TCP operation).

The `maxcpus=' Argument

The number given with this argument limits the maximum number of CPUs activated in SMP mode. Using a value of 0 is equivalent to the `nosmp` option.

The `mca–pentium' Argument

The IBM model 95 Microchannel machines seem to lock up on the test that Linux usually does to detect the type of math chip coupling. Since all Pentium chips have a built in math processor, this test (and the lock up problem) can be avoided by using this boot option.

The `md=' Argument

If your root filesystem is on a Multiple Device then you can use this (assuming you compiled in boot support) to tell the kernel the multiple device layout. The format (from the file `linux/Documentation/md.txt`) is:

```
md=md_device_num,raid_level,chunk_size_factor,fault_level,dev0,dev1,...,devN
```

Where `md_device_num` is the number of the md device, i.e. 0 means md0, 1 means md1, etc. For `raid_level`, use –1 for linear mode and 0 for striped mode. Other modes are currently unsupported. The `chunk_size_factor` is for raid–0 and raid–1 only and sets the chunk size as PAGE_SIZE shifted left the

specified amount. The `fault_level` is only for raid–1 and sets the maximum fault number to the specified number. (Currently unsupported due to lack of boot support for raid1.) The `dev0–devN` are a comma separated list of the devices that make up the individual md device: e.g.
`/dev/hda1,/dev/hdc1,/dev/sda1`

See also `raid=`.

The `nmi_watchdog=' Argument

Supplying a non–zero integer will enable the non maskable interrupt watchdog (assuming IO APIC support is compiled in). This checks to see if the interrupt count is increasing (indicating normal system activity) and if it is not then it assumes that a processor is stuck and forces an error dump of diagnostic information.

The `no387' Argument

Some i387 coprocessor chips have bugs that show up when used in 32 bit protected mode. For example, some of the early ULSI–387 chips would cause solid lockups while performing floating point calculations, apparently due to a bug in the FRSAV/FRRESTOR instructions. Using the `no387' boot argument causes Linux to ignore the math coprocessor even if you have one. Of course you must then have your kernel compiled with math emulation support! This may also be useful if you have one of those *really* old 386 machines that could use an 80287 FPU, as Linux can't use an 80287.

The `no–hlt' Argument

The i386 (and successors thereof) family of CPUs have a `hlt' instruction which tells the CPU that nothing is going to happen until an external device (keyboard, modem, disk, etc.) calls upon the CPU to do a task. This allows the CPU to enter a `low–power' mode where it sits like a zombie until an external device wakes it up (usually via an interrupt). Some of the early i486DX–100 chips had a problem with the `hlt' instruction, in that they couldn't reliably return to operating mode after this instruction was used. Using the `no–hlt' instruction tells Linux to just run an infinite loop when there is nothing else to do, and to *not* halt your CPU when there is no activity. This allows people with these broken chips to use Linux, although they would be well advised to seek a replacement through a warranty where possible.

The `no–scroll' Argument

Using this argument at boot disables scrolling features that make it difficult to use Braille terminals.

The `noapic' Argument

Using this option tells a SMP kernel to not use some of the advanced features of the interrupt controller on multi processor machines. Use of this option may be required when a device (such as those using ne2k–pci or 3c59xi drivers) stops generating interrupts (i.e. `cat /proc/interrupts` shows the same interrupt count.) See `linux/Documentation/IO-APIC.txt` for more information.

The `noht' Argument

This will disable hyper–threading on intel processors that have this feature.

The `noisapnp' Argument

If ISA PnP is built into the kernel, this will disable it.

The `nomce' Argument

Some newer processors have the ability to self–monitor and detect inconsistencies that should not regularly happen. If an inconsistency is detected, a Machine Check Exception will take place and the system will be halted (rather than plundering forward and corrupting your data). You can use this argument to disable this feature, but be sure to check that your CPU is not overheating or otherwise faulty first.

The `nosmp' Argument

Use of this option will tell a SMP kernel on a SMP machine to operate single processor. Typically only used for debugging and determining if a particular problem is SMP related.

The `noresume' Argument

If software suspend is enabled, and a suspend to disk file has been specified, using this argument will give a normal boot and the suspend data will be ignored.

The `notsc' Argument

Use of this option will tell the kernel to not use the Time Stamp Counter for anything, even if the CPU has one.

The `nofxsr'' Argument

Use of this option will tell the kernel to not use any speed–up tricks involving the floating point unit, even if the processor supports them.

The `panic=' Argument

In the unlikely event of a kernel panic (i.e. an internal error that has been detected by the kernel, and which the kernel decides is serious enough to moan loudly and then halt everything), the default behaviour is to just sit there until someone comes along and notices the panic message on the screen and reboots the machine. However if a machine is running unattended in an isolated location it may be desirable for it to automatically reset itself so that the machine comes back on line. For example, using `panic=30` at boot would cause the kernel to try and reboot itself 30 seconds after the kernel panic happened. A value of zero gives the default behaviour, which is to wait forever.

Note that this timeout value can also be read and set via the `/proc/sys/kernel/panic` sysctl interface.

The `pirq=' Argument

Using this option tells a SMP kernel information on the PCI slot versus IRQ settings for SMP motherboards which are unknown (or known to be blacklisted). See `linux/Documentation/IO-APIC.txt` for more information.

The `profile=' Argument

Kernel developers can profile how and where the kernel is spending its CPU cycles in an effort to maximize efficiency and performance. This option lets you set the profile shift count at boot. Typically it is set to two. You need a tool such as `readprofile.c` that can make use of the `/proc/profile` output.

The `quiet' Argument

This is pretty much the opposite of the `debug' argument. When this is given, only important and system critical kernel messages are printed to the console. Normal messages about hardware detection at boot are suppressed.

The `raid=' Argument

Accepts `noautodetect` at the moment. See also `md=`.

The `reboot=' Argument

This option controls the type of reboot that Linux will do when it resets the computer (typically via `/sbin/init` handling a Control–Alt–Delete). The default as of v2.0 kernels is to do a `cold' reboot (i.e. full reset, BIOS does memory check, etc.) instead of a `warm' reboot (i.e. no full reset, no memory check). It was changed to be cold by default since that tends to work on cheap/broken hardware that fails to reboot when a warm reboot is requested. To get the old behaviour (i.e. warm reboots) use `reboot=w` or in fact any word that starts with `w` will work.

Other accepted options are `c', `b', `h', and `s', for cold, bios, hard, and SMP respectively. The `s' takes an optional digit to specify which CPU should handle the reboot. Options can be combined where it makes sense, i.e. `reboot=b,s2`

The `reserve=' Argument

This is used to *protect* I/O port regions from probes. The form of the command is:

```
reserve=iobase,extent[,iobase,extent]...
```

In some machines it may be necessary to prevent device drivers from checking for devices (auto–probing) in a specific region. This may be because of poorly designed hardware that causes the boot to *freeze* (such as some ethercards), hardware that is mistakenly identified, hardware whose state is changed by an earlier probe, or merely hardware you don't want the kernel to initialize.

The `reserve` boot–time argument addresses this problem by specifying an I/O port region that shouldn't be probed. That region is reserved in the kernel's port registration table as if a device has already been found in that region (with the name `reserved`). Note that this mechanism shouldn't be necessary on most machines. Only when there is a problem or special case would it be necessary to use this.

The I/O ports in the specified region are protected against device probes that do a `check_region()` prior to probing blindly into a region of I/O space. This was put in to be used when some driver was hanging on a NE2000, or misidentifying some other device as its own. A correct device driver shouldn't probe a reserved region, unless another boot argument explicitly specifies that it do so. This implies that `reserve` will most

often be used with some other boot argument. Hence if you specify a `reserve` region to protect a specific device, you must generally specify an explicit probe for that device. Most drivers ignore the port registration table if they are given an explicit address.

For example, the boot line

```
reserve=0x300,32  blah=0x300
```

keeps all device drivers except the driver for `blah' from probing `0x300-0x31f`.

As usual with boot–time specifiers there is an 11 parameter limit, thus you can only specify 5 reserved regions per `reserve` keyword. Multiple `reserve` specifiers will work if you have an unusually complicated request.

The `resume=' Argument

If you are using software suspend, then this will allow you to specify the file name of the suspend to disk data that you want the machine to resume from.

The `vga=' Argument

Note that this is not really a boot argument. It is an option that is interpreted by LILO and not by the kernel like all the other boot arguments are. However its use has become so common that it deserves a mention here. It can also be set via using `rdev -v` or equivalently `vidmode` on the vmlinuz file. This allows the setup code to use the video BIOS to change the default display mode before actually booting the Linux kernel. Typical modes are 80x50, 132x44 and so on. The best way to use this option is to start with `vga=ask` which will prompt you with a list of various modes that you can use with your video adapter before booting the kernel. Once you have the number from the above list that you want to use, you can later put it in place of the `ask'. For more information, please see the file `linux/Documentation/svga.txt` that comes with all recent kernel versions.

Note that newer kernels (v2.1 and up) have the setup code that changes the video mode as an option, listed as `Video mode selection support` so you need to enable this option if you want to use this feature.

4. Boot Arguments to Control PCI Bus Behaviour (`pci=')

The `pci=' argument (not avail. in v2.0 kernels) can be used to change the behaviour of PCI bus device probing and device behaviour. Firstly the file `linux/drivers/pci/pci.c` checks for architecture independent `pci=` options. The remaining allowed arguments are handled in `linux/arch/???/kernel/bios32.c` and are listed below for ???=i386.

4.1 The `pci=assign–busses' Argument

This tells the kernel to always assign all PCI bus numbers, overriding whatever the firmware may have done.

4.2 The `pci=bios' and `pci=nobios' Arguments

These are used to set/clear the flag indicating that the PCI probing is to take place via the PCI BIOS. The

default is to use the BIOS.

4.3 The `pci=conf1' and `pci=conf2' Arguments

If PCI direct mode is enabled, the use of these enables either configuration Type 1 or Type 2. These implicitly clear the PCI BIOS probe flag (i.e. `pci=nobios') too.

4.4 The `pci=irqmask=' Argument

This allows the user to supply an IRQ mask value, which is converted using strtol(). It will set a bit mask of IRQs allowed to be assigned automatically to PCI devices. You can make the kernel exclude IRQs of your ISA cards this way.

4.5 The `pci=lastbus=' Argument

This allows the user to supply a lastbus value, which is converted using strtol(). It will scan all buses till bus N. Can be useful if the kernel is unable to find your secondary buses and you want to tell it explicitly which ones they are.

4.6 The `pci=noacpi' Argument

This disables the use of ACPI routing information during the PCI configuration stages.

4.7 The `pci=nopeer' Argument

This disables the default peer bridge fixup, which according to the source does the following:

``In case there are peer host bridges, scan bus behind each of them. Although several sources claim that the host bridges should have header type 1 and be assigned a bus number as for PCI2PCI bridges, the reality doesn't pass this test and the bus number is usually set by BIOS to the first free value."

4.8 The `pci=nosort' Argument

Using this argument instructs the kernel to not sort the PCI devices during the probing phase.

4.9 The `pci=off' Argument

Using this option disables all PCI bus probing. Any device drivers that make use of PCI functions to find and initialize hardware will most likely fail to work.

4.10 The `pci=usepirqmask' Argument

This sets the USE_PIRQ_MASK flag during PCI init. The kernel will honour the possible IRQ mask stored in the BIOS PIR table. This is needed on some systems with broken BIOSes, notably some HP Pavilion N5400 and Omnibook XE3 notebooks. This will have no effect if ACPI IRQ routing is enabled.

4.11 The `pci=rom' Argument

This sets the ASSIGN_ROM flag during the probing phase. The kernel will assign address space to expansion ROMs. Use with caution as certain devices share address decoders between ROMs and other resources.

---

5. Boot Arguments for Video Frame Buffer Drivers

The `video=' argument (not avail. in v2.0 kernels) is used when the frame buffer device abstraction layer is built into the kernel. If that sounds complicated, well it isn't really too bad. It basically means that instead of having a different video program (the X11R6 server) for each brand of video card (e.g. XF86_S3, XF86_SVGA, ...), the kernel would have a built in driver available for each video card and export a single interface for the video program so that only one X11R6 server (XF86_FBDev) would be required. This is similar to how networking is now – the kernel has drivers available for each brand of network card and exports a single network interface so that just one version of a network program (like Netscape) will work for all systems, regardless of the underlying brand of network card.

The typical format of this argument is `video=name:option1,option2,...` where `name` is the name of a generic option or of a frame buffer driver. The `video=` option is passed from `linux/init/main.c` into `linux/drivers/video/fbmem.c` for further processing. Here it is checked for some generic options before trying to match to a known driver name. Once a driver name match is made, the comma separated option list is then passed into that particular driver for final processing. The list of valid driver names can be found by reading down the `fb_drivers` array in the file `fbmem.c` mentioned above.

Information on the options that each driver supports will eventually be found in `linux/Documentation/fb/` but currently (v2.2) only a few are described there. Unfortunately the number of video drivers and the number of options for each one is content for another document itself and hence too much to list here.

If there is no Documentation file for your card, you will have to get the option information directly from the driver. Go to `linux/drivers/video/` and look in the appropriate `???fb.c` file (the ??? will be based on the card name). In there, search for a function with `_setup` in its name and you should see what options the driver tries to match, such as `font` or `mode` or...

5.1 The `video=map:...' Argument

This option is used to set/override the console to frame buffer device mapping. A comma separated list of numbers sets the mapping, with the value of option N taken to be the frame buffer device number for console N.

5.2 The `video=scrollback:...' Argument

A number after the colon will set the size of memory allocated for the scrollback buffer. (Use Shift and Page Up or Page Down keys to scroll.) A suffix of `k' or `K' after the number will indicate that the number is to be interpreted as kilobytes instead of bytes.

5.3 The `video=vc:...' Argument

A number, or a range of numbers (e.g. `video=vc:2−5`) will specify the first, or the first and last frame buffer virtual console(s). The use of this option also has the effect of setting the frame buffer console to *not* be the default console.

6. Boot Arguments for SCSI Peripherals.

This section contains the descriptions of the boot args that are used for passing information about the installed SCSI host adapters, and SCSI devices.

6.1 Arguments for Upper and Mid−level Drivers

The upper level drivers handle all things SCSI, regardless of whether they be disk, tape, or CD−ROM. The mid level drivers handle things like disks, CD−ROMs and tapes without getting into low level host adapter device driver specifics.

Maximum Probed LUNs (`max_scsi_luns=')

Each SCSI device can have a number of `sub−devices' contained within itself. The most common example is any of the SCSI CD−ROMs that handle more than one disk at a time. Each CD is addressed as a `Logical Unit Number' (LUN) of that particular device. But most devices, such as hard disks, tape drives and such are only one device, and will be assigned to LUN zero.

The problem arises with single LUN devices with bad firmware. Some poorly designed SCSI devices (old and unfortunately new) can not handle being probed for LUNs not equal to zero. They will respond by locking up, and possibly taking the whole SCSI bus down with them.

The kernel has a configuration option that allows you to set the maximum number of probed LUNs. The default is to only probe LUN zero, to avoid the problem described above.

To specify the number of probed LUNs at boot, one enters `max_scsi_luns=n' as a boot arg, where n is a number between one and eight. To avoid problems as described above, one would use n=1 to avoid upsetting such broken devices

SCSI Logging (`scsi_logging=')

Supplying a non−zero value to this boot argument turns on logging of all SCSI events (error, scan, mlqueue, mlcomplete, llqueue, llcomplete, hlqueue, hlcomplete). Note that better control of which events are logged can be obtained via the `/proc/scsi/scsi` interface if you aren't interested in the events that take place at boot before the `/proc/` filesystem is accessible.

Parameters for the SCSI Tape Driver (`st=')

Some boot time configuration of the SCSI tape driver can be achieved by using the following:

```
st=buf_size[,write_threshold[,max_bufs]]
```

The first two numbers are specified in units of kB. The default `buf_size` is 32kB, and the maximum size that can be specified is a ridiculous 16384kB. The `write_threshold` is the value at which the buffer is committed to tape, with a default value of 30kB. The maximum number of buffers varies with the number of drives detected, and has a default of two. An example usage would be:

```
st=32,30,2
```

Full details can be found in the `README.st` file that is in the `scsi` directory of the kernel source tree.

6.2 Arguments for SCSI Host Adapter Drivers

These are arguments for low level SCSI host device drivers, and as such are typically only used by those that compile their own kernel with the SCSI driver built in. These people are advised to check the source for the latest list of options that can be supplied to their driver.

`aha152x=` Adaptec aha151x, aha152x, aic6260, aic6360, SB16–SCSI

`aha1542=` Adaptec aha1540, aha1542

`aic7xxx=` Adaptec aha274x, aha284x, aic7xxx

`advansys=` AdvanSys SCSI Host Adaptors

`in2000=` Always IN2000 Host Adaptor

`AM53C974=` AMD AM53C974 based hardware

`BusLogic=` ISA/PCI/EISA BusLogic SCSI Hosts

`eata=` EATA SCSI Cards

`tmc8xx=` Future Domain TMC–8xx, TMC–950

`fdomain=` Future Domain TMC–16xx, TMC–3260, AHA–2920

`ppa=` IOMEGA Parallel Port / ZIP drive

`ncr5380=` NCR5380 based controllers

`ncr53c400=` NCR53c400 based controllers

`ncr53c406a=` NCR53c406a based controllers

`pas16=` Pro Audio Spectrum

`st0x=` Seagate ST–0x

`t128=` Trantor T128

`u14-34f=` Ultrastor SCSI cards

`wd7000=` Western Digital WD7000 cards

7. Hard Disks

This section lists all the boot args associated with standard MFM/RLL, ST–506, XT, and IDE disk drive devices. Note that both the IDE and the generic ST–506 HD driver both accept the `hd=' option.

7.1 IDE Disk/CD–ROM Driver Parameters

The IDE driver accepts a number of parameters, which range from disk geometry specifications, to support for advanced or broken controller chips. The following is a summary of some of the more common boot arguments. For full details, you *really* should consult the file `ide.txt` in the `linux/Documentation` directory, from which this summary was extracted.

```
"hdx="  is recognized for all "x" from "a" to "h", such as "hdc".
"idex=" is recognized for all "x" from "0" to "3", such as "ide1".

"hdx=noprobe"          : drive may be present, but do not probe for it
"hdx=none"             : drive is NOT present, ignore cmos and do not probe
"hdx=nowerr"           : ignore the WRERR_STAT bit on this drive
"hdx=cdrom"            : drive is present, and is a cdrom drive
"hdx=cyl,head,sect"    : disk drive is present, with specified geometry
"hdx=autotune"         : driver will attempt to tune interface speed
                               to the fastest PIO mode supported,
                               if possible for this drive only.
                               Not fully supported by all chipset types,
                               and quite likely to cause trouble with
                               older/odd IDE drives.

"idex=noprobe"         : do not attempt to access/use this interface
"idex=base"            : probe for an interface at the addr specified,
                               where "base" is usually 0x1f0 or 0x170
                               and "ctl" is assumed to be "base"+0x206
"idex=base,ctl"        : specify both base and ctl
"idex=base,ctl,irq"    : specify base, ctl, and irq number
"idex=autotune"        : driver will attempt to tune interface speed
                               to the fastest PIO mode supported,
                               for all drives on this interface.
                               Not fully supported by all chipset types,
                               and quite likely to cause trouble with
                               older/odd IDE drives.
"idex=noautotune"      : driver will NOT attempt to tune interface speed
                               This is the default for most chipsets,
                               except the cmd640.
"idex=serialize"       : do not overlap operations on idex and ide(x^1)
```

The following are valid ONLY on ide0, and the defaults for the base,ctl ports must not be altered.

```
"ide0=dtc2278"         : probe/support DTC2278 interface
"ide0=ht6560b"         : probe/support HT6560B interface
"ide0=cmd640_vlb"      : *REQUIRED* for VLB cards with the CMD640 chip
                         (not for PCI -- automatically detected)
"ide0=qd6580"          : probe/support qd6580 interface
"ide0=ali14xx"         : probe/support ali14xx chipsets (ALI M1439/M1445)
```

```
"ide0=umc8672"          : probe/support umc8672 chipsets
```

During the install of some PCMCIA systems, you may be able to get detection of your CD−ROM by using:

```
"ide2=0x180,0x386"      : probe typical PCMCIA IDE interface location
```

Everything else is rejected with a "BAD OPTION" message. Also note that there is an implied `ide0=0x1f0 ide1=0x170` in the absence of any other ide boot args.

7.2 Old MFM/RLL/Standard ST−506 Disk Driver Options (`hd=')

The standard disk driver can accept geometry arguments for the disks similar to the IDE driver. Note however that it only expects three values (C/H/S) −− any more or any less and it will silently ignore you. Also, it only accepts `hd=' as an argument, i.e. `hda=', `hdb=' and so on are not valid here. The format is as follows:

```
        hd=cyls,heads,sects
```

If there are two disks installed, the above is repeated with the geometry parameters of the second disk.

7.3 XT Disk Driver Options (`xd=', `xd_geo=')

If you are unfortunate enough to be using one of these old 8 bit cards that move data at a whopping 125kB/s then here is the scoop. The probe code for these cards looks for an installed BIOS, and if none is present, the probe will not find your card. Or, if the signature string of your BIOS is not recognized then it will also not be found. In either case, you will then have to use a boot argument of the form:

```
        xd=type,irq,iobase,dma_chan
```

The `type` value specifies the particular manufacturer of the card, and are as follows: 0=generic; 1=DTC; 2,3,4=Western Digital, 5,6,7=Seagate; 8=OMTI. The only difference between multiple types from the same manufacturer is the BIOS string used for detection, which is not used if the type is specified.

The `xd_setup()` function does no checking on the values, and assumes that you entered all four values. Don't disappoint it. Here is an example usage for a WD1002 controller with the BIOS disabled/removed, using the `default' XT controller parameters:

```
        xd=2,5,0x320,3
```

If the disk geometry that the kernel prints out comes out all wrong to what you know the disk is set up as, you can override that as well, with:

```
        xd_geo=cyl_xda,head_xda,sec_xda
```

Add another comma and another three CHS values if you are silly enough to have two disks on the old hunk of scrap...

8. The Sound Drivers

Note that there was a rewrite of a lot of the sound core and related drivers. The older stuff is generally called `OSS' and the newer is called `ALSA'. The intention is to drop the OSS stuff eventually. To avoid name conflict, the ALSA stuff generally has `snd–' as a prefix to all the boot parameters.

Note that each driver has its own individual boot argument (very old kernels used a shared `sound=`). Also, generally no defaults are set at compile time (i.e. you *must* supply a boot argument for older non–PNP ISA cards to be detected.) Your best source of information for your card is the files in `linux/Documentation/sound/`.

8.1 Individual Sound Device Driver Arguments

ALSA ISA drivers

`snd–dummy=` Dummy soundcard

`snd–mpu401=` mpu401 UART

`snd–mtpav=` MOTU Midi Timepiece

`snd–serial=` Serial UART 16450/16550 MIDI

`snd–virmidi=` Dummy soundcard for virtual rawmidi devices

`snd–ad1816a=` ADI SoundPort AD1816A

`snd–ad1848=` Generic driver for AD1848/AD1847/CS4248

`snd–als100=` Avance Logic ALS100

`snd–azt2320=` Aztech Systems AZT2320 (and 2316)

`snd–cmi8330=` C–Media's CMI8330

`snd–cs4231=` Generic driver for CS4231 chips

`snd–cs4232=` Generic driver for CS4232 chips

`snd–cs4236=` Generic driver for CS4235/6/7/8/9 chips

`snd–dt019x=` Diamond Technologies DT–019x

`snd–es1688=` Generic ESS AudioDrive ESx688

`snd–es18xx=` Generic ESS AudioDrive ES18xx

`snd–gusclassic=` Gus classic

`snd-gusextreme=` Gus extreme

`snd-gusmax=` Gus Max

`snd-interwave=` Interwave

`snd-interwave-stb=` Interwave

`snd-opl3sa2=` Yamaha OPL3SA2

`snd-opti93x=` OPTi 82c93x based cards

`snd-opti92x-cs4231=` OPTi 82c92x/CS4231

`snd-opti92x-ad1848=` OPTi 82c92x/AD1848

`snd-es968=` ESS AudioDrive ES968

`snd-sb16=` SoundBlaster 16

`snd-sbawe=` SoundBlaster 16 AWE

`snd-sb8=` Old 8 bit SoundBlaster (1.0, 2.0, Pro)

`snd-sgalaxy=` Sound galaxy

`snd-wavefront=` Wavefront

OSS drivers

`ad1848=` AD1848

`adlib=` Adlib

`mad16=` MAD16

`pas2=` ProAudioSpectrum PAS16

`sb=` SoundBlaster

`uart401=` UART 401 (on card chip)

`uart6850=` UART 6850 (on card chip)

`opl3=` Yamaha OPL2/OPL3/OPL4 FM Synthesizer (on card chip)

`opl3sa=` Yamaha OPL3–SA FM Synthesizer (on card chip)

`opl3sa2=` Yamaha OPL3–SA2/SA3 FM Synthesizer (on card chip)

ALSA PCI Drivers

`snd-ali5451`= ALi PCI audio M5451

`snd-als4000`= Avance Logic ALS4000

`snd-cmipci`= C–Media CMI8338 and 8738

`snd-cs4281`= Cirrus Logic CS4281

`snd-cs46xx`= Cirrus Logic Sound Fusion CS46XX

`snd-emu10k1`= EMU10K1 (SB Live!)

`snd-ens1370`= Ensoniq ES1370 AudioPCI

`snd-ens1371`= Ensoniq ES1371 AudioPCI

`snd-es1938`= ESS Solo–1 (ES1938, ES1946, ES1969)

`snd-es1968`= ESS Maestro 1/2/2E

`snd-fm801`= ForteMedia FM801

`snd-intel8x0`= Intel ICH (i8x0) chipsets

`snd-maestro3`= ESS Maestro3/Allegro (ES1988)

`snd-korg1212`= Korg 1212 IO

`snd-rme32`= RME Digi32, Digi32/8 and Digi32 PRO

`snd-nm256`= NeoMagic 256AV and 256ZX

`snd-rme96`= RME Digi96, Digi96/8 and Digi96/8 PRO/PAD/PST

`snd-rme9652`= RME Digi9652 audio interface

`snd-hdsp`= RME Hammerfall DSP

`snd-sonicvibes`= S3 SonicVibes

`snd-trident`= Trident 4DWave DX/NX & SiS SI7018

`snd-via82xx`= VIA South Bridge VT82C686A/B/C, VT8233A/C, VT8235

`snd-ymfpci`= Yamaha DS1/DS1E

`snd-ice1712`= ICEnsemble ICE1712 (Envy24)

9. CD–ROMs (Non–SCSI/ATAPI/IDE)

This section lists all the possible boot args pertaining to these older CD–ROM devices on proprietary interface cards. Note that this does not include SCSI or IDE/ATAPI CD–ROMs. See the appropriate section(s) for those types of CD–ROMs.

Note that most of these CD–ROMs have documentation files that you *should* read, and they are all in one handy place: `linux/Documentation/cdrom`.

9.1 Old CD–ROM Driver Arguments

`aztcd=` Aztech Interface

`cdu31a=` CDU–31A and CDU–33A Sony Interface (Also Old PAS)

`sonycd535=` CDU–535 Sony Interface

`gscd=` GoldStar Interface

`isp16=` ISP16 Interface

`mcd=` Mitsumi Standard Interface

`mcdx=` Mitsumi XA/MultiSession Interface

`optcd=` Optics Storage Interface

`cm206=` Phillips CM206 Interface

`sjcd=` Sanyo Interface

`sbpcd=` SoundBlaster Pro Interface

---

10. Serial and ISDN Drivers

10.1 The ISDN drivers

Please see `linux/Documentation/isdn/` for the full details of all the options the following ISDN drivers accept.

`icn=` ICN ISDN driver

`pcbit=` PCBIT ISDN driver

`teles=` Teles ISDN driver

10.2 The Serial drivers

Please see `linux/Documentation/` and/or the `README` files in `linux/drivers/char` for the full details of all the options that the following support.

`digi=` DigiBoard Driver

`riscom8=` RISCom/8 Multiport Serial Driver

`baycom=` Baycom Serial/Parallel Radio Modem

---

11. Other Hardware Devices

Any other devices that didn't fit into any of the above categories got lumped together here.

11.1 Ethernet Devices (`ether=', `netdev=')

Different drivers make use of different parameters, but they all at least share having an IRQ, an I/O port base value, and a name. In its most generic form, it looks something like this:

```
ether=irq,iobase[,param_1[,param_2,...param_8]]],name
```

The first non–numeric argument is taken as the name. The `param_n` values (if applicable) usually have different meanings for each different card/driver. Typical `param_n` values are used to specify things like shared memory address, interface selection, DMA channel and the like.

The most common use of this parameter is to force probing for a second ethercard, as the default is to only probe for one (with 2.4 and older kernels). This can be accomplished with a simple:

```
ether=0,0,eth1
```

Note that the values of zero for the IRQ and I/O base in the above example tell the driver(s) to autoprobe.

IMPORTANT NOTE TO MODULE USERS: The above will *not* force a probe for a second card if you are using the driver(s) as run time loadable modules (instead of having them complied into the kernel). Most Linux distributions use a bare bones kernel combined with a large selection of modular drivers. The `ether=` only applies to drivers compiled directly into the kernel.

The Ethernet–HowTo has complete and extensive documentation on using multiple cards and on the card/driver specific implementation of the `param_n` values where used. Interested readers should refer to the section in that document on their particular card for more complete information. Ethernet–HowTo

11.2 The Floppy Disk Driver (`floppy=')

There are many floppy driver options, and they are all listed in `floppy.txt` in `linux/Documentation`. There are too many options in that file to list here. Instead, only those options that may be required to get a Linux install to proceed on less than normal hardware are reprinted here.

`floppy=0,daring` Tells the floppy driver that your floppy controller should be used with caution (disables all daring operations).

`floppy=thinkpad` Tells the floppy driver that you have a Thinkpad. Thinkpads use an inverted convention for the disk change line.

`floppy=nodma` Tells the floppy driver not to use DMA for data transfers. This is needed on HP Omnibooks, which don't have a workable DMA channel for the floppy driver. This option is also useful if you frequently get `Unable to allocate DMA memory' messages. Use of `nodma' is not recommended if you have a FDC without a FIFO (8272A or 82072). 82072A and later are OK). The FDC model is reported at boot. You also need at least a 486 to use nodma.

`floppy=nofifo` Disables the FIFO entirely. This is needed if you get `Bus master arbitration error' messages from your Ethernet card (or from other devices) while accessing the floppy.

`floppy=broken_dcl` Don't use the disk change line, but assume that the disk was changed whenever the device node is reopened. Needed on some boxes where the disk change line is broken or unsupported. This should be regarded as a stopgap measure, indeed it makes floppy operation less efficient due to unneeded cache flushings, and slightly more unreliable. Please verify your cable connection and jumper settings if you have any DCL problems. However, some older drives, and also some Laptops are known not to have a DCL.

`floppy=debug` Print (additional) debugging messages.

`floppy=messages` Print informational messages for some operations (disk change notifications, warnings about over and underruns, and about autodetection).

11.3 The Bus Mouse Driver (`bmouse=')

The busmouse driver only accepts one parameter, that being the hardware IRQ value to be used.

11.4 The MS Bus Mouse Driver (`msmouse=')

The MS mouse driver only accepts one parameter, that being the hardware IRQ value to be used.

11.5 The Printer Driver (`lp=')

With this boot argument you can tell the printer driver what ports to use and what ports *not* to use. The latter comes in handy if you don't want the printer driver to claim all available parallel ports, so that other drivers (e.g. PLIP, PPA) can use them instead.

The format of the argument is multiple i/o, IRQ pairs. For example, `lp=0x3bc,0,0x378,7` would use the port at `0x3bc` in IRQ–less (polling) mode, and use IRQ 7 for the port at `0x378`. The port at `0x278` (if any) would not be probed, since autoprobing only takes place in the absence of a `lp=` argument. To disable the printer driver entirely, one can use `lp=0`.

11.6 The Parallel port IP driver (`plip=')

Using `plip=timid` will tell the plip driver to avoid any ports that appear to be in use by other parallel port devices. Otherwise you can use `plip=parportN` where `N` is a non–zero integer indicating the parallel port

to use. (Using `N=0` will disable the plip driver.)

12. Copying, Translations, Closing, etc.

Hey, you made it to the end! (Phew...) Now just the legal stuff.

12.1 Copyright and Disclaimer

This document is Copyright (c) 1995−1999 by Paul Gortmaker. Copying and redistribution is allowed under the conditions as outlined in the Linux Documentation Project Copyright, available from where you obtained this document, OR as outlined in the GNU General Public License, version 2 (see linux/COPYING).

This document is *not* gospel. However, it is probably the most up to date info that you will be able to find. Nobody is responsible for what happens to your hardware but yourself. If your stuff goes up in smoke, or anything else bad happens, we take no responsibility. ie. THE AUTHOR IS NOT RESPONSIBLE FOR ANY DAMAGES INCURRED DUE TO ACTIONS TAKEN BASED ON THE INFORMATION INCLUDED IN THIS DOCUMENT.

A hint to people considering doing a translation. First, translate the SGML source (available via FTP from the HowTo main site) so that you can then generate other output formats. Be sure to keep a copy of the original English SGML source that you translated from! When an updated HowTo is released, get the new SGML source for that version, and then a simple `diff −u old.sgml new.sgml` will show you exactly what has changed so that you can easily incorporate those changes into your translated SMGL source without having to re−read or re−translate everything.

If you are intending to incorporate this document into a published work, please make contact (via e−mail) so that you can be supplied with the most up to date information available. In the past, out of date versions of the Linux HowTo documents have been published, which caused the developers undue grief from being plagued with questions that were already answered in the up to date versions.

12.2 Closing

If you have found any glaring typos, or outdated info in this document, please let me know. It is easy to overlook stuff, as the kernel (and the number of drivers) is huge compared to what it was when I started this.

Thanks,

Paul Gortmaker, `p_gortmaker @ yahoo.com`