NAME

bootparam − introduction to boot time parameters of the Linux kernel

DESCRIPTION

The Linux kernel accepts certain 'command-line options' or 'boot time parameters' at the moment it is started. 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.

When the kernel is booted directly by the BIOS, you have no opportunity to specify any parameters. So, in order to take advantage of this possibility you have to use a boot loader that is able to pass parameters, such as GRUB.

The argument list

The kernel command line is parsed into a list of strings (boot arguments) separated by spaces. Most of the boot arguments have the form:

```
name[=value_1][,value_2]...[,value_10]
```

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. Note the limit of 10 is real, as the present code handles only 10 comma separated parameters per keyword. (However, you can reuse the same keyword with up to an additional 10 parameters in unusually complicated situations, assuming the setup function supports it.)

Most of the sorting is coded in the kernel source file *init/main.c*. First, the kernel checks to see if the argument is any of the special arguments 'root=', 'nfsroot=', 'nfsaddrs=', 'ro', 'rw', 'debug' or 'init'. The meaning of these special arguments is described below.

Then it walks a list of setup functions 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 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 arguments 3, 4, 5, and 6 as given on the kernel command line.

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. A (useless?) example would be to use 'TERM=vt100' as a boot argument.

Any remaining arguments that were not picked up by the kernel and were not interpreted as environment variables are then passed onto PID 1, which is usually the **init**(1) program. The most common argument that is passed to the *init* process is the word 'single' which instructs it to boot the computer in single user mode, and not launch all the usual daemons. Check the manual page for the version of **init**(1) installed on your system to see what arguments it accepts.

General non-device-specific boot arguments

'init=...'

This sets the initial command to be executed by the kernel. If this is not set, or cannot be found, the kernel will try */sbin/init*, then */etc/init*, then */bin/init*, then */bin/sh* and panic if all of this fails.

'nfsaddrs=...'

This sets the NFS boot address to the given string. This boot address is used in case of a net boot.

'nfsroot=...'

This sets the NFS root name to the given string. If this string does not begin with '/' or ',' or a digit, then it is prefixed by '/tftpboot/'. This root name is used in case of a net boot.

'root=...'

This argument tells the kernel what device is to be used as the root filesystem while booting. The default of this setting is determined at compile time, and usually is the value of the root device of the system that the kernel was built on. To override this value, and select the second floppy drive as the root device, one would use 'root=/dev/fd1'.

The root device can be specified symbolically or numerically. A symbolic specification has the form */dev/XXYN*, where XX designates the device type (e.g., ’hd’ for ST-506 compatible hard disk, with Y in ’a’-’d’; ’sd’ for SCSI compatible disk, with Y in ’a’-’e’), Y the driver letter or number, and N the number (in decimal) of the partition on this device.

Note that this has nothing to do with the designation of these devices on your filesystem. The ’/dev/’ part is purely conventional.

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

’rootdelay=’
This parameter sets the delay (in seconds) to pause before attempting to mount the root filesystem.

’rootflags=...’
This parameter sets the mount option string for the root filesystem (see also **fstab**(5)).

’rootfstype=...’
The ’rootfstype’ option tells the kernel to mount the root filesystem as if it where of the type specified. This can be useful (for example) to mount an ext3 filesystem as ext2 and then remove the journal in the root filesystem, in fact reverting its format from ext3 to ext2 without the need to boot the box from alternate media.

’ro’ and **’rw’**
The ’ro’ option tells the kernel to mount the root filesystem as ’read-only’ so that filesystem consistency check programs (fsck) can do their work on a quiescent filesystem. No processes can write to files on the filesystem in question until it is ’remounted’ as read/write capable, for example, by ’mount −w −n −o remount /’. (See also **mount**(8).)

The ’rw’ option tells the kernel to mount the root filesystem read/write. This is the default.

’resume=...’
This tells the kernel the location of the suspend-to-disk data that you want the machine to resume from after hibernation. Usually, it is the same as your swap partition or file. Example:

```
resume=/dev/hda2
```

’reserve=...’
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 hardware that reacts badly to the probing, or hardware that would be mistakenly identified, or merely hardware you don’t want the kernel to initialize.

The reserve boot-time argument specifies an I/O port region that shouldn’t be probed. A device driver will not probe a reserved region, unless another boot argument explicitly specifies that it do so.

For example, the boot line

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

keeps all device drivers except the driver for ’blah’ from probing 0x300−0x31f.

’panic=N’
By default, the kernel will not reboot after a panic, but this option will cause a kernel reboot after N seconds (if N is greater than zero). This panic timeout can also be set by

```
echo N > /proc/sys/kernel/panic
```

'reboot=[warm|cold][,[bios|hard]]'

Since Linux 2.0.22, a reboot is by default a cold reboot. One asks for the old default with 'reboot=warm'. (A cold reboot may be required to reset certain hardware, but might destroy not yet written data in a disk cache. A warm reboot may be faster.) By default, a reboot is hard, by asking the keyboard controller to pulse the reset line low, but there is at least one type of motherboard where that doesn't work. The option 'reboot=bios' will instead jump through the BIOS.

'nosmp' and **'maxcpus=N'**

(Only when __SMP__ is defined.) A command-line option of 'nosmp' or 'maxcpus=0' will disable SMP activation entirely; an option 'maxcpus=N' limits the maximum number of CPUs activated in SMP mode to N.

Boot arguments for use by kernel developers

'debug'

Kernel messages are handed off to a daemon (e.g., **klogd**(8) or similar) so that they may be logged to disk. Messages with a priority above *console_loglevel* are also printed on the console. (For a discussion of log levels, see **syslog**(2).) By default, *console_loglevel* is set to log messages at levels higher than **KERN_DEBUG**. This boot argument will cause the kernel to also print messages logged at level **KERN_DEBUG**. The console loglevel can also be set on a booted system via the */proc/sys/kernel/printk* file (described in **syslog**(2)), the **syslog**(2) **SYSLOG_ACTION_CONSOLE_LEVEL** operation, or **dmesg**(8).

'profile=N'

It is possible to enable a kernel profiling function, if one wishes to find out where the kernel is spending its CPU cycles. Profiling is enabled by setting the variable *prof_shift* to a nonzero value. This is done either by specifying **CONFIG_PROFILE** at compile time, or by giving the 'profile=' option. Now the value that *prof_shift* gets will be N, when given, or **CONFIG_PROFILE_SHIFT**, when that is given, or 2, the default. The significance of this variable is that it gives the granularity of the profiling: each clock tick, if the system was executing kernel code, a counter is incremented:

```
profile[address >> prof_shift]++;
```

The raw profiling information can be read from */proc/profile*. Probably you'll want to use a tool such as readprofile.c to digest it. Writing to */proc/profile* will clear the counters.

Boot arguments for ramdisk use

(Only if the kernel was compiled with **CONFIG_BLK_DEV_RAM**.) In general it is a bad idea to use a ramdisk under Linux—the system will use available memory more efficiently itself. But while booting, it is often useful to load the floppy contents into a ramdisk. One might also have a system in which first some modules (for filesystem or hardware) must be loaded before the main disk can be accessed.

In Linux 1.3.48, ramdisk handling was changed drastically. Earlier, the memory was allocated statically, and there was a 'ramdisk=N' parameter to tell its size. (This could also be set in the kernel image at compile time.) These days ram disks use the buffer cache, and grow dynamically. For a lot of information on the current ramdisk setup, see the kernel source file *Documentation/blockdev/ramdisk.txt* (*Documentation/ramdisk.txt* in older kernels).

There are four parameters, two boolean and two integral.

'load_ramdisk=N'

If N=1, do load a ramdisk. If N=0, do not load a ramdisk. (This is the default.)

'prompt_ramdisk=N'

If N=1, do prompt for insertion of the floppy. (This is the default.) If N=0, do not prompt. (Thus, this parameter is never needed.)

'ramdisk_size=N' or (obsolete) **'ramdisk=N'**

Set the maximal size of the ramdisk(s) to N kB. The default is 4096 (4 MB).

'ramdisk_start=N'

Sets the starting block number (the offset on the floppy where the ramdisk starts) to N. This is needed in case the ramdisk follows a kernel image.

'noinitrd'

(Only if the kernel was compiled with **CONFIG_BLK_DEV_RAM** and **CONFIG_BLK_DEV_INITRD**.) These days it is possible to compile the kernel to use initrd. When this feature is enabled, the boot process will load the kernel and an initial ramdisk; then the kernel converts initrd into a "normal" ramdisk, which is mounted read-write as root device; then */linuxrc* is executed; afterward the "real" root filesystem is mounted, and the initrd filesystem is moved over to */initrd*; finally the usual boot sequence (e.g., invocation of */sbin/init*) is performed.

For a detailed description of the initrd feature, see the kernel source file *Documentation/admin−guide/initrd.rst* (or *Documentation/initrd.txt* before Linux 4.10).

The 'noinitrd' option tells the kernel that although it was compiled for operation with initrd, it should not go through the above steps, but leave the initrd data under */dev/initrd*. (This device can be used only once: the data is freed as soon as the last process that used it has closed */dev/initrd*.)

Boot arguments for SCSI devices

General notation for this section:

iobase -- the first I/O port that the SCSI host occupies. These are specified in hexadecimal notation, and usually lie in the range from 0x200 to 0x3ff.

irq -- the hardware interrupt that the card is configured to use. Valid values will be dependent on the card in question, but will usually be 5, 7, 9, 10, 11, 12, and 15. The other values are usually used for common peripherals like IDE hard disks, floppies, serial ports, and so on.

scsi-id -- the ID that the host adapter uses to identify itself on the SCSI bus. Only some host adapters allow you to change this value, as most have it permanently specified internally. The usual default value is 7, but the Seagate and Future Domain TMC-950 boards use 6.

parity -- whether the SCSI host adapter expects the attached devices to supply a parity value with all information exchanges. Specifying a one indicates parity checking is enabled, and a zero disables parity checking. Again, not all adapters will support selection of parity behavior as a boot argument.

'max_scsi_luns=...'

A SCSI device can have a number of 'subdevices' contained within itself. The most common example is one of the new 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.

Some poorly designed SCSI devices cannot handle being probed for LUNs not equal to zero. Therefore, if the compile-time flag **CONFIG_SCSI_MULTI_LUN** is not set, newer kernels will by default probe only LUN zero.

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 tape configuration

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 32k B, and the maximum size that can be specified is a ridiculous 16384 kB. The *write_threshold* is the value at which the buffer is committed to tape, with a default value of 30 kB. 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 file *Documentation/scsi/st.txt* (or *drivers/scsi/README.st* for older kernels) in the Linux kernel source.

Hard disks

IDE Disk/CD-ROM Driver Parameters

The IDE driver accepts a number of parameters, which range from disk geometry specifications, to support for broken controller chips. Drive-specific options are specified by using 'hdX=' with X in 'a'-'h'.

Non-drive-specific options are specified with the prefix 'hd='. Note that using a drive-specific prefix for a non-drive-specific option will still work, and the option will just be applied as expected.

Also note that 'hd=' can be used to refer to the next unspecified drive in the (a, ..., h) sequence. For the following discussions, the 'hd=' option will be cited for brevity. See the file *Documentation/ide/ide.txt* (or *Documentation/ide.txt* in older kernels, or *drivers/block/README.ide* in ancient kernels) in the Linux kernel source for more details.

The 'hd=cyls,heads,sects[,wpcom[,irq]]' options

These options are used to specify the physical geometry of the disk. Only the first three values are required. The cylinder/head/sectors values will be those used by fdisk. The write precompensation value is ignored for IDE disks. The IRQ value specified will be the IRQ used for the interface that the drive resides on, and is not really a drive-specific parameter.

The 'hd=serialize' option

The dual IDE interface CMD-640 chip is broken as designed such that when drives on the secondary interface are used at the same time as drives on the primary interface, it will corrupt your data. Using this option tells the driver to make sure that both interfaces are never used at the same time.

The 'hd=noprobe' option

Do not probe for this drive. For example,

```
hdb=noprobe hdb=1166,7,17
```

would disable the probe, but still specify the drive geometry so that it would be registered as a valid block device, and hence usable.

The 'hd=nowerr' option

Some drives apparently have the **WRERR_STAT** bit stuck on permanently. This enables a workaround for these broken devices.

The 'hd=cdrom' option

This tells the IDE driver that there is an ATAPI compatible CD-ROM attached in place of a normal IDE hard disk. In most cases the CD-ROM is identified automatically, but if it isn't then this may help.

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 expects only three values (C/H/S); any more or any less and it will silently ignore you. Also, it accepts only 'hd=' as an argument, that is, 'hda=' 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.

Ethernet devices

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_8]],name
```

The first nonnumeric 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 probe only for one. 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.

The Ethernet-HowTo has 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.

The floppy disk driver

There are many floppy driver options, and they are all listed in *Documentation/blockdev/floppy.txt* (or *Documentation/floppy.txt* in older kernels, or *drivers/block/README.fd* for ancient kernels) in the Linux kernel source. See that file for the details.

The sound driver

The sound driver can also accept boot arguments to override the compiled-in values. This is not recommended, as it is rather complex. It is described in the Linux kernel source file *Documentation/sound/oss/README.OSS* (*drivers/sound/Readme.linux* in older kernel versions). It accepts a boot argument of the form:

```
sound=device1[,device2[,device3...[,device10]]]
```

where each deviceN value is of the following format 0xTaaaId and the bytes are used as follows:

T – device type: 1=FM, 2=SB, 3=PAS, 4=GUS, 5=MPU401, 6=SB16, 7=SB16-MPU401

aaa – I/O address in hex.

I – interrupt line in hex (i.e., 10=a, 11=b, ...)

d – DMA channel.

As you can see, it gets pretty messy, and you are better off to compile in your own personal values as recommended. Using a boot argument of 'sound=0' will disable the sound driver entirely.

The line printer driver

'lp='

Syntax:

```
lp=0
lp=auto
lp=reset
lp=port[,port...]
```

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 port names. For example, lp=none,parport0 would use the first parallel port for lp1, and disable lp0. To disable the printer driver entirely, one can use lp=0.

SEE ALSO

klogd(8), **mount**(8)

For up-to-date information, see the kernel source file *Documentation/admin-guide/kernel-parameters.txt*.

COLOPHON

This page is part of release 5.02 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man−pages/.