

# How to Make Linux Faster

## Without Buying New Hardware

Kyle Wheeler

April 18, 2002

## 1 Introduction

A few years ago, most computers had a “turbo” button (and before that, they had a physical turbo *switch*) to switch between normal (say, 10MHz) and extra-fast (say, 12MHz). The reason was that while most of the time you wanted your programs to go faster, sometimes—especially playing games—12MHz was just a little *too* fast.<sup>1</sup> These days, programs are (usually) a little smarter than that, and there’s no such thing as a computer that’s too fast. You already have a very fast computer, but chances are, you aren’t taking advantage of how fast it really is.

## 2 The Disk

Most of the components of the computer are not all the same speed. Your CPU may be slower than your RAM, the main bus may be slower than either of those, your PCI bus may be slower than that, but still faster than your hard drive—all of which is faster than the Zip/floppy/CD-ROM drive. This creates bottlenecks in the computer—pieces of the computer that slow the rest of it down.

One of the main bottlenecks in a computer is disk access. The disk is the obstacle that slows down copying files (usually), and since most operating systems use the disk for “swap space” to augment their memory management, the disk can slow things down even more.

### 2.1 `hdparm`

Chances are your hard drive is a lot faster than it seems to be. Over the last twenty years or so, several advances in hard drive technology has made them faster and faster. However, not all hard drives support the same features. Thus, when the typical Linux distribution installs itself it plays things safe, and only talks to the hard drive in a way that is practically guaranteed to work, be stable, and unfortunately, be very slow. Fortunately, if your hard drive has been purchased in the last five to ten years, you can probably get a little more out of it than you’re currently getting with a tool called `hdparm`.

There are many options and features about your hard drive that `hdparm` can affect, and not all of them are particularly useful (from a speed perspective anyway). For example, you can put the drive to sleep, or you can set it to read-only, or whatever. The speed-related options, however, can be a little dangerous, especially if you are working with older hardware and a certain feature is not supported. It is recommended that you switch your machine to single-user mode (`telinit 1`) before fiddling with any of these flags.<sup>2</sup>

---

<sup>1</sup>The reason there is that the games used “timing-loops” to pace out the game. Essentially, they would do some simple routine a few thousand times in order to space out, for example, screen updates. At the slower speeds, a simple routine a few thousand times could be counted on to take a little while. The faster the computer got, the quicker the loop was processed, and the pacing of the game sped up along with the processor.

<sup>2</sup>That said, I’ve done it over the network with few processes running without ill effects, but some of these options warn that you can cause massive file-system corruption if you abuse them. You have been warned.

### 2.1.1 Speed Testing

Probably the first thing you want to do is test the speed of your drive in a quantitative way. There are two test options, one of which (-T) simply tests the speed of the Linux buffer cache. That option is generally a function of the speed of your CPU, main bus, RAM, and kernel version, but kind of interesting to know. The other option, -t is the raw disk I/O speed. You should run it 2-3 times to get meaningful results.

### 2.1.2 DMA

The 2.4.x Linux kernels turn DMA on by default, but on some types of hardware or on 2.2.x kernels it is an option. This is probably the biggest speed boost you will see—access speeds can jump up to 300

```
hdparm -d1 /dev/hdX && hdparm -t /dev/hdX
```

There are some other `hdparm` options that are related - the -x and -p options. These are almost never useful, and usually dangerous (this is a very quick and easy way to corrupt your filesystem). On modern kernels (2.2.x) these options are nearly always set to the correct and optimal value for the drive.

### 2.1.3 Multiple Sector I/O

Multiple Sector Mode (also known as IDE Block Mode) is a feature of most modern IDE hard drives, permitting the transfer of multiple sectors per I/O interrupt, rather than the usual one sector per interrupt. When this feature is enabled, it typically reduces OS overhead for disk access by 30-50%. On many systems, it also provides increased data throughput of anywhere from 5 to 50%. Some drives, however, run more slowly with this option turned on (like Western Digital Caviar drives). Do testing to find out which is the best setting. Generally, you should do `hdparm -i /dev/hdX`, look for the `MaxMultSect` record, and use that value initially, test, and back down from there (in powers of two). Try it like this:

```
hdparm -m16 /dev/hdX
```

### 2.1.4 Bandwidth

The most compatible and safe method for talking to hard drives is 16-bits at a time. All (E)IDE drives still have only a 16-bit connection over the ribbon cable, but the IDE chipsets in modern computers can usually talk to the computer (over the PCI or VLB bus) 32-bits at a time. I say usually, because there are a few IDE chipsets that get confused by 32-bits, and require a special “sync” sequence. This is still faster than 16-bit mode, but requires a little more overhead for the OS than pure 32-bit mode. For `hdparm`, there are three possible arguments here: 0 (16-bit mode), 1 (32-bit mode), and 3 (32-bit with sync). Sync mode works with all modern chipsets. Switching from 16-bit to 32-bit mode typically doubles your disk access speed. Use it like this:

```
hdparm -c3 /dev/hdX
```

### 2.1.5 Interrupts

You can allow Linux to unmask other interrupts during processing of a disk interrupt. This greatly improves Linux’s responsiveness and eliminates “serial port overrun” errors (where accessing the disk (e.g. one that is asleep and has to wake up) can delay the computer enough that the modem fills the serial port buffer, for example). However, some chipsets (specifically CMD-640B and RZ1000) are unreliable in this mode due to a hardware flaw (disabling “IDE prefetch” in the BIOS may help with those chipsets). Set this option like this:

```
hdparm -u1 /dev/hdX
```

### 2.1.6 Finally

Now that you’ve set all the options, test the speed again. See how much faster that is? To quote the O’Reilly website, “Now launch Netscape. And try not to fall out of your chair.”

## 2.2 Filesystem

Depending on what you're doing, the "format" or filesystem of your disk partition may be slowing you down. For example, if you used to dual-boot your machine into Windows and Linux and one of your partitions is all MS-DOS, that will slow down access to files on that partition. What filesystem to use depends on what you will use the disk for. EXT2 or EXT3 is generally stable and fast for desktop machines. Webservers or systems with many users may benefit from using something like SGI's XFS.

As a tip, there are a lot of "journaling" filesystems out there for Linux (EXT3 is one of them). Journaling filesystems are generally safer than standard filesystems in that the disk is *much* less likely to be in an inconsistent state if your machine loses power or crashes (you were probably doing kernel development, right?). Journaling filesystems are not inherently any faster or slower during normal use, but using one avoids the need to run `fsck` on your partitions after a crash—which can save you all kinds of time.

If you have a kernel that supports it (any kernel past 2.4.9, as long as you compiled in EXT3 support), you should probably convert your partitions to EXT3 format (unless you're running a high-availability, high-load server that may stress the EXT3 code in ways that hasn't been thought of before). You'll thank me the next time the power goes out and you think about buying a UPS again. Converting your partitions is very easy. First, run this command on them:

```
tune2fs -j /dev/hdX
```

Then edit `/etc/fstab` and change the partition's filetype. Then unmount or remount the partition (you can reboot if you like). It's that simple. And EXT3 is backwards-compatible with EXT2, so if you have an old rescue disk, it will be able to read the EXT3 partition as if it was an EXT2 partition.

Another thing that will increase your disk performance is telling the OS not to record file access time. Normally, every time you access a file (like on a webserver when a page is requested) the access time is incremented in the file's inode. You can add the `noatime` option to a partition's entry in `/etc/fstab` to disable this, which will speed up performance (because it requires fewer disk operations for each file access), particularly in busy servers.

## 2.3 Swap, Partition Position, and More

If you have multiple drives, move the partitions that are the most active (the `/usr` partition, for example) to the faster drive. Also, if you have multiple drives, put a swap partition on each drive. This speeds up swapping (because the kernel can interleave swap reads and writes and can then read from both swap partitions at the same time).

Also, leave empty space on your disk. If your disk is more than about 75

## 3 Memory

The other main bottleneck in your computer is RAM. RAM is very fast memory, and the CPU talks to it a lot for whatever it's currently doing. Ideally, the CPU keeps every program currently executing entirely in RAM. This makes switching between applications and doing any computation or memory access any of the running applications want to have done very fast and easy. However, if you don't have very much RAM, the OS will keep what it's working on *right now* in RAM, and will save what doesn't fit in memory to disk temporarily (read: swap space). The more things you do and the less RAM you have, the more things will need to be saved to disk while the CPU works on what's currently in RAM.

### 3.1 More Swap Space

If you enjoy running many many programs at once, but they aren't all always active (like, for example, if you want to keep several Mozilla windows open at once, even though you may only be using one at a time), then you may want to invest in more swap space. This may be a null-gain, because of the time it takes to take what was in swap and put it back in RAM and take what's in RAM and put it on disk when you change programs or windows, BUT, more swap space will allow more processes to run at once and will allow the OS to clear out more RAM for the current process.

Once upon a time, swap space was limited to being a maximum of 128MB. This is no longer the case (and hasn't for a few years), so load up with at least as much swap space as you have RAM.

## 3.2 Buy More Ram

Yes, the title of this paper includes "Without Buying...", but RAM is reasonably inexpensive. If you can afford it, get up to 512MB of RAM, and you will see marked speed improvements.

## 3.3 Choosing What to Run

The problem with RAM is that there's never enough of it to go around. One thing you can do to speed up your machine (and also to secure it) is to examine what exactly you are running, and perhaps use something else that is less RAM-hungry or don't use a certain program at all.

Start by disabling and removing default servers and similar items. For example, unless you're using NFS, you probably don't need portmap running and taking up RAM. If you don't need a web-server, don't run one.

In general, GUI programs, particularly GNOME and KDE and the like, are very memory hungry. You can cut down on that and speed up your computer by switching from them to a simpler, window-manager-only approach. For example, try blackbox, fluxbox, pwm, or WindowMaker as alternatives to GNOME and KDE—they provide many similar features, without nearly as great a demand on your computer's resources. Similarly, try out different terminal programs. Instead of gnome-terminal or kterminal (which both use large amounts of RAM and have many features you may not need), try out smaller and lighter terminal programs like `xterm`, `aterm`, and `rxvt`.

# 4 Programs

## 4.1 Kernel

### 4.1.1 Version

In general, newer kernels are frequently faster than older ones. The 2.4.x series of kernels contains many internal optimizations that make them faster than the 2.2.x series. Download the most recent version of the kernel, compile it and install it—quite likely, it will be faster than it used to be.

### 4.1.2 Modules

Ideally, you want your kernel to be tailored for your machine—support for exactly what hardware you have, and what features you want, and no others. This reduces kernel size, and in some cases increases kernel speed.

When compiling a kernel, many options have the ability to be built as modules, to be loaded during runtime, rather than being built into the kernel itself. This is something to consider when thinking of speed. If you want your computer to support something—a feature or a hardware device—but that device will not usually be attached or that feature will not always be used (such as the kernel pcap functions, or a USB driver), you should compile that as a module, to keep the kernel from thinking about a feature or device that may not be attached or used. However, if you have a device or feature that is always in use (like an ethernet card or video card), you should compile it into the kernel if you can (some people may have kernel size restrictions). This allows the compiler to optimize the kernel a little more, and doesn't require the kernel module auto-loader to go looking for the module.

### 4.1.3 Low-Latency and Preemptive Patches

There are some kernel patches out there for low-latency (<http://www.zip.com.au/~akpm/linux/schedlat.html>) or to make the kernel preemptable (<http://www.tech9.net/rml/linux/>). These can reduce the latency of your machine and make it respond faster and \*feel\* faster, even if it isn't technically processing things faster. Be warned that these patches are experimental.

## 4.2 Compiler

The compiler is able to improve execution speed of programs by "optimizing" the code. The compiler can compress multiple operations into single operations and can reformat flow control in programs (among other things) that it compiles in order to improve execution speed (this optimization does not affect the source code). In general, if you tell the compiler to optimize a program, the optimized version will execute more quickly and require less CPU-time. Most Linux programs are compiled with level two optimization. At further levels, the compiler may be unreliable. The next significant version of GCC to be released (3.1) will include significant improvements in its optimization code, and will be able to better optimize existing programs.