

# What on Earth is... THE HURD?

**Richard Drummond** answers your questions regarding the operating system kernel whose place Linux usurped.

## >> What is this free operating system called the Hurd?

The Hurd itself isn't actually an OS – it's an operating system kernel, just like Linux is a kernel not an operating system. And, like Linux, the Hurd is designed to be a kernel for the GNU Operating System.

## >> I don't get the distinction. What's a kernel and what's an OS?

Simply put, an operating system is the set of software which make a computer easy to use and easy to write software for. It controls and provides access to hardware resources, it furnishes the environment which allows you to run other programs such as application software, and it provides the interface with which the user interacts with the machine (such as the windowing systems favoured

by modern operating systems). The kernel is the central core of an operating system and provides the really essential services upon which the rest of the OS and other software depends – like process management, memory management, a file-system and so on.

## >> OK. So what's the GNU Operating System?

The GNU OS is the free reimplementation of the Unix operating system that was begun by Richard Stallman and the Free Software Foundation (FSF) in 1983.

The GNU OS and the Linux kernel are the two principal components in every Linux distribution. For example, the *bash* shell, the *gcc* C compiler, and Emacs are all part of the GNU OS. When you use Red Hat or Mandrake or any other current Linux distribution, you are using the GNU OS – whether you

realise it or not. So great is the role of GNU software in forming a usable Linux-based operating system that the FSF maintain that Linux distributions should properly be called GNU/Linux. However, that's a debate for another day.

## >> So the Hurd is meant to replace the Linux kernel?

You could say that, yes. But, actually, the Hurd predates the Linux kernel. It was always the intention of the FSF to create a kernel for their free Unix replacement, but for one reason and another very little progress was made early on. The Hurd project – as we know it today – wasn't begun until 1990 and a bootable Hurd kernel wasn't available until 1994.

The first public release of Linux was in 1991 – after the Hurd had been announced. Many people who started using the Linux kernel back then regarded it as stop-gap measure to get the GNU OS up and running until the Hurd was ready for release. Linux was considered a toy and thought not to be portable to





architectures other than the x86-based PC, a platform dismissed by hackers who grew up using Unix on 'real' computer hardware. Eventually, however, Linux matured and stole the limelight away from the Hurd. This is partly attributable to the ever-increasing popularity and power of the PC platform – but it is also due to the fact that developing the Hurd turned out to be a lot more difficult than anybody expected.

### »» So, why bother with the Hurd now that we have Linux?

Why not? The Hurd project was in full swing by time the Linux kernel was considered mature, and it didn't make sense to the Hurd developers to abandon all that work. Besides, the Hurd is fundamentally different than the Linux kernel and from any other mainstream operating system kernel. It's a good idea to have two projects trying to solve the same problem using radically different approaches. The most important reason for the existence of the Hurd project, however, is that the developers enjoy working on it. It's fun.

### »» What makes the Hurd so different from Linux?

It's a matter of architecture. The Linux kernel is a traditional monolithic design, whereas the Hurd is built on a micro-kernel architecture. This distinction is a technical one.

Typically, a kernel provides its services by executing software routines in a special mode of the CPU that it is running on. This kernel mode – or supervisor mode – allows the kernel to employ the special or privileged CPU instructions that it needs to perform such tasks as virtual memory management and process scheduling – which aren't available to software running in normal user mode.

The difference between a monolithic kernel and a micro-kernel is in how much of the kernel's services are provided by code running in kernel mode. A monolithic kernel such as Linux provides all of its services in kernel mode – including device drivers, network stacks (such as TCP/IP), and filesystems. A micro-kernel provides only the very

basic services such as process management, memory management and inter-process communication in kernel mode – everything else can be implemented in user mode.

### »» So which of these is actually better: a monolithic kernel or a micro-kernel?

It depends who you ask! Micro-kernels were a hot topic in academic circles during the 1980s but fell out of favour in the 1990s (although they are now gaining in popularity again). Linus Torvalds is a famous example of somebody who has always been a vocal opponent of the micro-kernel approach.

The principal advantages of micro-kernels are generally considered to be flexibility and ease of development. In a micro-kernel much of the kernel code executes in user-mode just like any other software. Thus it can be developed, tested and debugged just like other software. It also makes the kernel more modular – with fewer interdependencies between kernel modules. This also makes development easier, and means that modifying the kernel is easier when requirements change. Thus a micro-kernel can be more flexible and more scalable. In contrast, in a monolithic kernel – where everything runs in the special kernel mode of the processor – testing and debugging can be very difficult. And because the kernel is one monolithic slab of code, interdependencies between different parts of the kernel can make modifying it difficult.

Ironically, the Linux and the Hurd kernels can both be seen as evidence against such arguments. The Hurd has proved to be notoriously difficult to develop and debug, whereas as you'll know, development on the Linux kernel takes place at a lightning pace. Moreover, the Linux kernel is actually very modular. Still, Linux can be difficult to modify, as anybody who participates in the Linux kernel mailing list can attest. Very often changes to one module or sub-system in the unstable testing branch of the kernel tree can cause wholesale breakage in other kernel components.

### »» All very nice, but what difference is there to the end user between each architecture? Which one is faster?

The performance characteristics of micro-kernels versus monolithic kernels has always been a hotly debated issue. Traditionally, the argument has always been that monolithic kernels offer better raw performance, but that micro-kernels provide better interactive performance. You can see why if you consider the architecture of each.

All modern operating systems are multi-tasking – that is, they provide the illusion that more than one program can be run at one time, while in actual fact the processor is still only doing one thing at a time. This illusion is created by task switching or scheduling. Processor time is shared between all the tasks running on the machine, and each process in turn is allowed to run for a short period or time-slice before it is put to sleep (pre-empted) and another process scheduled and allowed to run.

How is this relevant? Well, typically, code running in kernel mode cannot be pre-empted. That means that when a program calls a kernel service that requires code to be executed in kernel mode then that program cannot be de-scheduled and another program run until that kernel service completes – no matter how long it takes. In a monolithic kernel this happens when any kernel service is used. Thus a monolithic kernel is bad news if you want tasks to get a bite at the processor at guaranteed, regular intervals – which is a requirement of real-time operating systems, and is essential to getting a good interactive response from an operating system. Micro-kernels don't have this problem since most kernel services run in user mode. On the flip side of the coin, if your interest is raw performance, then the monolithic kernel can usually offer better throughput of data for the very same reasons: code run in kernel mode can't be interrupted so it can be more efficient.

Because of the way they are designed, the determining feature of micro-kernel performance is how efficiently interprocess communication (IPC) operates. Early micro-kernels often gave poor





# WhatOnEarthTheHurd

« performance due to inefficient IPC. This is one reason why Torvalds and others are so dismissive of micro-kernels. Modern second-generation micro-kernels offer much better performance.

## » Enough theory. Tell me more about the Hurd.

The Hurd is implemented as a set of user mode services hosted over a micro-kernel. Currently, the Hurd is built on the GNU Mach micro-kernel.

## » Wait a minute. Mach? I've heard of that. Isn't that used in Mac OS X?

Yes. GNU Mach is derived from the Mach micro-kernel originally developed at Carnegie Mellon University during pioneering research into micro-kernel based operating systems in the 1980s. A version of Mach is also the basis for Mac OS X.

In fact, the Mach micro-kernel has popped up in various operating system projects over the years – and seemed at one point to be set to dominate the field of OS development. It was the basis for NextSTEP, Steve Jobs' first attempt at Unix for the masses when he got the boot from Apple; Apple themselves sponsored a port of Linux to the Power Mac based on Mach called MkLinux; and famously there's now Mac OS X (itself derived from NextSTEP). It's not only Apple and ex-Apple employees that have been interested in Mach, though. There was also a traditional BSD Unix implementation based on Mach, called Lites; and IBM were to use Mach as the basis of their port of OS/2 to the PowerPC architecture (which never appeared). Funnily enough, thanks to Mac OS X, Mach is now the most widely-deployed Unix kernel.

Various parties have contributed to Mach development over the years. The Open Software Foundation (the OSF) dabbled for a while producing OSF Mach. Utah University picked up where CMU left off, and produced Mach 4 – upon which GNU Mach 1.2 is based – and OSKit Mach – which is the basis for GNU Mach 2.0. Meanwhile, MacOS X is hosted on OSF Mach 3.0. Did you get all that?

## » If there's been all these attempts at building Unix and Unix-like operating systems on top of Mach, why bother with Hurd? What's new?

Glad you asked. All the previous efforts to create production operating system based on Mach have opted for what is known as a 'single server' design. The user space portion of the kernel – which provides the remainder of the kernel services that Mach itself doesn't provide and supplies the application interface to kernel – is implemented as a single program; hence the term 'single server'.

In fact, most of these single-server implementations have been ports of a traditional, monolithic kernel to run on top of Mach. NextSTEP, Lites and MacOS X all use BSD Unix implemented as a Mach server, while MkLinux obviously uses a Mach-based port of the Linux kernel.

The Hurd, on the other hand, is rather different. It opts for a radical 'multiple server' design, where kernel services – such as user authentication, network stacks, filesystems, and so on – are implemented as a series of separate programs which run in user space.

(As an aside, this is where the name 'Hurd' is derived from. It's a play on words, a long-favourite diversion of hackers. 'Hurd' stands for 'Hird of Unix-Replacing Daemons' and then 'Hird' stands for 'Hurd of Interfaces Representing Depth'. The Hurd developers seem quite proud of the fact this is first time a software project has been named by a pair of mutually recursive acronyms.)

A single-server design fails to leverage the full flexibility that a micro-kernel core can potentially offer and suffers from many of the drawbacks of a traditional monolithic kernel: that is, the kernel is effectively still implemented as a single, monolithic chunk of code. The Hurd, however, takes full advantage of its micro-kernel basis to offer unparalleled flexibility. A user can choose to run which services they need, modify the behaviour of any of the services, or replace services with different implementations.

## » How do programs communicate with the Hurd's kernel services?

The Hurd provides all the usual features of a Unix kernel, but these are implemented in a novel way that makes it much more flexible for the user and much easier to extend. A core tenet of the Unix philosophy is that everything is a file: directories are files, devices are files, sockets and pipes are files. The Hurd is the same on the surface, since it is designed to be Unix-like, but the underlying mechanism is much different.

In the Hurd, a program uses kernel services – not by entering kernel mode and running the appropriate kernel routine – but by using the Mach micro-kernel's IPC system. In this scheme a program communicates with another program by sending a message to its message port. It is exactly the same when using kernel services since kernel services are just regular programs. But how do you know which port to send messages to? Well, in the Hurd, a server is identified by a file path relative to the root filesystem (or, more properly, the inode pointed to by that path). Thus, in the Hurd, while everything might be a file, every file identifies a message port on some server.

Inodes may represent regular files or more abstract services. Special Unix files such as named pipes are supplied by the fifo server; sockets by the pflocal server and symlinks by the symbolic link server.

A Hurd server which provides an interface through the filesystem in this manner is known as a translator. A translator is a regular program whose job it is to go between a file path and the program accessing that path and perform some kind of 'translation'. The translator is just one of the concepts that makes the Hurd so flexible.

## » What are some examples of translators in the Hurd?

A filesystem server which implements a regular filesystem is just one example of a translator. In this case the translator provides access to a device file corresponding to the block device upon which the





filesystem is stored. You mount a filesystem in the Hurd by associating a translator and a device with a particular file path in a manner somewhat similar to mounting a filesystem on a traditional Unix.

Another example of a translator is the pfinet server, which implements the TCP/IP protocol (and is accessed by default via the path /servers/socket/2). Bringing up a network connection in the Hurd requires that you associate the pfinet server with a particular network device such as an Ethernet card.

### » I still don't get it. What makes this translator concept so powerful?

Translators are especially powerful because the user requires no special privileges to use a translator – they just need permission to access the file path that the translator is being attached to. Also, translators are just regular user-space programs. Users are free to create, modify and use translators as they see fit. Compare this to a traditional Unix, where only the superuser can mount filesystems and where adding support for a new filesystem involves modifying the kernel, recompiling and rebooting. What's more the Hurd provides a set of shared libraries to promote code re-use and to make writing new translators very easy.

### » What filesystems does the Hurd support?

The Hurd currently includes translators which understand the Linux *ext2*, BSD's *ufs* and the ISO9660 filesystems. Support for the FAT filesystem and for NFS is also available. In addition, the *ftps* server provides a transparent, filesystem-like interface to FTP sites.

### » I'm sold. The Hurd sounds intriguing. How do I try it out?

The focal point for development on the Hurd is the Hurd pages on the FSF website (point your browser at [www.gnu.org/software/hurd/](http://www.gnu.org/software/hurd/)). These pages contain a lot of useful information on the Hurd and

also links to where you can download the Hurd, get installation instructions and so on.

While you can build GNU Mach, the Hurd and rest of the GNU OS from source code, most people wishing to try out the Hurd are not that masochistic. The GNU site hosts a binary GNU/Hurd distribution provided as a series of (currently four) CD images. Download these, burn them to CD-R and install away. (See [ftp://ftp.gnu.org/iso/](http://ftp.gnu.org/iso/).) Alternatively, you can try Debian's distribution of GNU/Hurd. This is built from the same source tree as the various Debian Linux ports and includes all the same great Debian infrastructure, such as the package update system APT. A large number of pre-compiled binary packages are included with Debian GNU/Hurd.

The Debian GNU/Hurd port is currently in the unstable phase of development, and, as such, official installation media have not been released. However, if you visit [www.debian.org/ports/hurd/hurd-cd](http://www.debian.org/ports/hurd/hurd-cd) you can find a list of mirrors where you can download unofficial ISO images from which to install the Hurd.

### » What hardware platforms does the Hurd support?

The Hurd only works on an x86 PC at the moment. A port to the Alpha platform has also been started.

### » Will I have problems getting Hurd driver support for my hardware?

The Hurd lacks the maturity of the Linux kernel and the support of the big hardware vendors that Linux enjoys, so expect hardware support in the Hurd to be poorer. In particular, GNU Mach 1.2, the basis for current releases of the Hurd, make use of block and other drivers from the 2.0.x series Linux kernels – which have been superseded a long time ago. GNU Mach 2.0, the current development target for future Hurd releases, is built on OSKit Mach has a more modern and flexible driver architecture, so expect hardware support to improve.

### » What software can I run on the Hurd?

The Hurd is POSIX compatible, so if you have the source code, then most Unix software can be compiled and run on the Hurd. Most of the same software that will build and run on Linux will work on the Hurd, too. No binary compatibility is provided, so you cannot at the moment run Linux or BSD binaries on the Hurd – although such possibilities have been discussed.

In real terms, most of the free application software that makes up a modern Linux distro will work on the Hurd. Thus Xfree86 and the GNOME desktop will both run on the Hurd. Problematic software will be anything that use APIs that are specific to Linux – such as the /dev/fb or /dev/v4l interfaces – or those that make assumptions for Linux which are not valid on the Hurd. For example, most Unices including Linux assume that there is an upper limit on the length of file path allowed. This isn't the case with the Hurd, so not allocating files paths dynamically may cause a whole heap of problems.

### » When will the Hurd be finished?

How long is a piece of string? Seriously, a software project is only finished when nobody uses that software any more. The Hurd has a long way to go before it is ready for production use – a long way from that much sought-after stable '1.0' release. It's stable enough for everyday use, although somewhat lacking in features in some areas.

The Hurd project is looking for volunteers to contribute to all areas of Hurd development; so if you want to bring that '1.0' release nearer, lend a hand. If you think that Linux has become too mature and too dull, live on the cutting-edge of OS development and get the excitement back by choosing the Hurd. Visit [www.gnu.org/software/hurd](http://www.gnu.org/software/hurd) for more info. [LXF](#)

