PC emulation

Say goodbye to dual boot as Simon Goodwin tests ways to run PC software without losing Linux.

This major instalment of our series deals with PC emulation. That means running software designed for commercial systems like MS-DOS and Windows – or other environments developed for cheap commodity PC clones – without abandoning Linux.

Linux communicates well with other operating systems, but there are many misguided people who think all PCs run Windows. Sometimes they write useful programs, or generate files that cannot be decoded by open source applications. Or you may need to test files your Linux system generates with proprietary packages like *Lotus Notes*, *Autoroute*, *Internet Explorer* or *Excel*.

Then again, you may wish to run a program compiled for FreeBSD, a Unix-like system such as QNX, or some other niche system that revitalises generic PC hardware. Perhaps BeOS or OS/2 still tickle your fancy? Emulators let you do all this, while your customary Linux environment continues to run.

On the CD and DVD

You'll find Mtools, FreeDOS, OpenDOS, DOSemu, VMware Workstation and Express and Plex86 demos and docs, plus source for Bochs and PCemu, on your well stuffed coverdisc.

Simulate or Virtualise

There are two categories of PC emulation on Linux, depending on the host hardware you're using. If you are running a commodity x86 box made to run Windows, PC emulation is a matter of setting Linux on one side as neatly as possible so that other systems think they're running on the original hardware.

You might argue that it's easier to shut down Linux and boot up the other operating system. Certainly any PC emulation succeeds or fails in comparison with this approach – if it's easier to run the other system as an alternative to Linux, rather than alongside it, emulation is not worth the candle.

But there are good reasons for running Linux at the same time as these rivals for a PC's attention. For a start, shutting down one system and starting another is a tedious business; as computers and operating systems claim to get faster perversely it becomes more so, rather than less. Gone are the days when you could turn on a computer and start typing within a few seconds, or switch off without warning software first.

While you can speed up Linux boot times substantially, by recompiling your kernel to eliminate compatibility cruft irrelevant to your own setup, that's not an option Microsoft offer. It often takes minutes just to shut down one system, start another and reverse the process to get back where you started, even on a notionally fast, modern computer. Emulation is very attractive if it means you can skip the rebooting, giving instant access to two or more systems at once. It also eases data transfer between operating systems; sharing files, networks and clipboard data.

Compatibility

Another justification for emulation is compatibility. The Storm Linux distro on our second CD had a SCSI driver compiled into its kernel which locked up while probing my *Symbios* PCI controller. I removed the card to test Storm for my multi-Linux feature, till I got *VMware* and was able to boot Storm from the original CD, running in a virtual machine alongside my more robust distros. I still had access to SCSI devices because *VMware* was virtualising all the drives, using host drivers for raw data access and representing SCSI devices as IDE ones Storm could cope with.

In theory programs for the earliest PC should still run on a modern machine, but in practice 'PC compatibility' is hit and miss. You have a better chance of running an old program in the operating system and environment it was intended for and tested on, using an emulator, rather than trusting to the backward compatibility of a later system.

Metal bashers

Virtual hardware is not always a benefit – it's safer but slower and may hide essential aspects. One reason you might want to run software for DOS or Windows is to set up hardware that you want to run under Linux but which only comes with Microsoft 'plug and pray' configuration code. Unfortunately the way that the PC bus is virtualised means that programs which try to peek and poke the hardware are unlikely to reach through to the underlying metal.

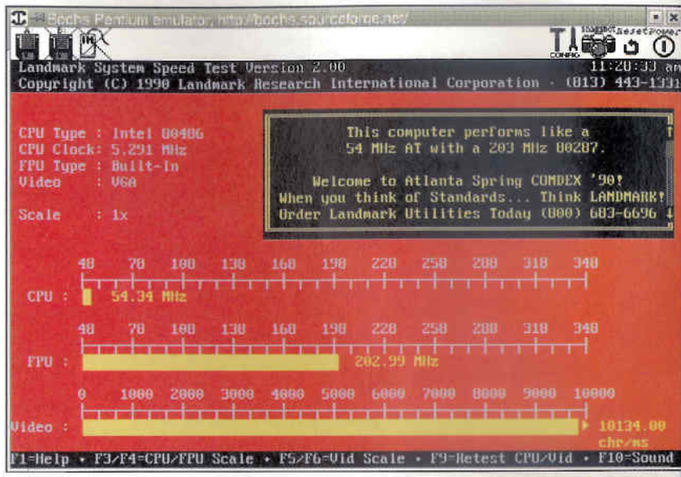
This is good news for system stability – wild pokes to ports can wreak havoc on an already-running system – but means that you'll rarely get away with running another system at the same time as such metal bashers. You'll probably need to boot into the required system, and nothing else, to run that code, but at least you should not need to do this on a regular basis.

Serial port emulation in Bochs and Plex86 has problems which will be hard to eliminate given the small size and tight timing requirements of serial transfers; serial input is especially hard for any system to emulate. The latest *VMware* supports USB peripherals, but warns that modems and other real-time streaming devices like USB speakers and web cameras are unlikely to work. Emulation overhead incites data overruns and loss of synchronisation.

Cross-emulation

We'll discuss systems that virtualise the PC hardware later, but first we should consider emulators that simulate the x86 as well as other PC chips. If you are using a PPC, 68K, Alpha, ARM, Playstation 2 or any other relatively modern CPU architecture, PC emulation is more of a challenge. It must mimic all the foibles of Intel's seventies x86 architecture, augmented by more than 20 years of 'compatible' extras, and IBM's original, arcane PC hardware. PC quirks and complications hamstring efficient software emulation.

The x86 processor has a prodigious throughput of instructions but many of those serve only to shuffle things internally, especially in legacy 16 bit code that uses segmented addressing and thrashes a small number of special-purpose CPU registers. The overhead of emulation depends more on the number of instructions that need to be processed than their complexity, so a clean wide host architecture with lots of registers is of only marginal benefit when you have to emulate a narrow and fiddly target system.



Bochs

Bochs is still in active development – indeed, the *Bochs* pages were sometimes the most active on SourceForge in 2001. *Bochs* can run Linux, Windows, and other systems intended for x86 processors and PC hardware, whatever the host CPU. I tested the CVS snapshot of *Bochs* from November 2001.

Bochs compiles easily and includes FPU emulation, fake serial or NE2000 networking, and optional audio. **bximage** creates floppy disk images or virtual hard drives of any size. '~/bochsrc' configures these for use. *Bochs* also supports direct media access from most systems.

Bochs binaries include a small Linux distribution to get users started, though it's a lot slower than the host. Developer Bryce Denney explains: "Bochs is emulating every instruction, and that's why it goes very slowly. It doesn't take advantage of the fact that it's running Linux in Linux." Of course the speed is much the same, or a bit less, running *Bochs* from Windows.

This is the key point to bear in mind when comparing *Bochs* and *PCemu* with virtualisers like *VMware*, *Plex86* or *Win4Lin*. Bryce on *VMware* on the PC: "they take advantage of running Intel instructions on an Intel, so they can zoom compared to *Bochs* emulating everything."

If you run a Mac or Amiga the situation is reversed when x86 users try to run your software, as they must emulate your 68K or Power PC. Before you struggle to run the x86 version of a program it's worth checking if there's a Mac version, as there is for many Microsoft programs, as *MacOnLinux* or *Basilisk II* may be a more efficient way to run the program.

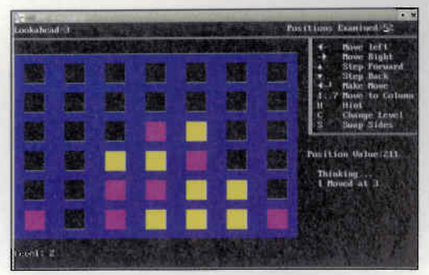
Almost all Linux programs come with portable source, usually in C, so you can recompile them for your favourite architecture and skip the emulation bottleneck. *Bochs* still suits old programs optimised with x86 assembler code to boost performance on older PCs.

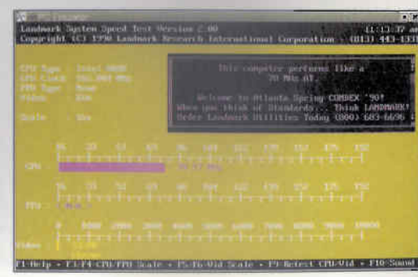
Bochs benchmarks depend on the configuration, but Landmark's SpeedCom classes our test setup as a 54 MHz 486 – draining some nine tenths of the CPU time of a 500 MHz K6.

PCemu

PCemu is a portable emulation of a PC-AT with an 8086 processor and mono text display. It was originally written as a Bristol University project on a Sun, tested with programs like *WordPerfect 5*, Microsoft *QBASIC* and *GWBASIC*, *BBC BASIC*, MS-DOS 5 and 6.2, *Borland Assembler* and *C++*.

Efficient compilation on Linux mandates tweaking the OPTIONS line in *PCemu's* Makefile. Use '-DLITTLE_ENDIAN' if you are running on an x86, ARM, Alpha or similar – keep it 'BIG_ENDIAN' on SUN, 68K or PPC, and remove -DALIGNED_ACCESS for a speed boost if your CPU allows misaligned memory accesses, reading words from odd addresses. CISC processors can do this, RISC ones typically can't. For a high density boot floppy change the default from '-DBOOT720' to '-DBOOT1_4'; you can override this later by editing





Below: **Speedcom rates PCemu as a humble 8088, albeit twenty times faster than IBM's on our AMD K6/2 Linux setup.**
Below left: **Block colour graphics are the limit for PCemu.**

'~/.pcemurc', the emulator runtime configuration file.

I also changed the Makefile XROOT assignment to '/usr/X11R6' from '/usr/local/X11R5' to suit current distros, and commented out parts of 'mfs.c': the includes for 'memory.h' and 'string.h' on lines 348 and 351, and a block 'added by DH' from line 1811 to 1823. *PCemu* then compiled and ran, despite warnings of confusion between pointers and integers.

After making a 'DriveA' boot disk image with **dd**, and converting the portable 'vga.bdf' font to '.pcf' format for SuSE, *PCemu* booted MS-DOS 3.3 and ran *GWBASIC* and *SPEEDCOM* – which scored it as a PC-AT clocked at between 34 and 105 MHz, depending on how much host CPU time the emulator gets.

Video was rather slow at 3,000 characters per second, but X does a lot more work than MDA, which updates a character for every byte poked to video RAM, and I can't read anywhere near that fast, anyhow. MS-DOS 5 and chunky versions of *Connect4* and *Invaders* worked OK, but *PCemu* does not support high resolution graphics so it's no good for most games.

File sharing

To gain access to Unix file system from MS-DOS you need a to move a couple of files from the emulator's programs directory to your DriveA image. 'vfat' and 'msdos' filesystems mean Linux can mount and use PC disks directly, but the *mtools* package remains a useful PC emulation accessory. I configured a boot disk image as drive G: with this line in '/etc/mtools.conf ':

```
drive g: path="/home/simon/pcemu10.1alpha/DriveA"
```

and copied the *PCemu* sharing files as follows:

```
mcopy programs/emufs.sys g:
mcopy programs/config.sys g:
```

This Mach extension gives access to compatibly-named files in the root directory of Linux in *PCemu's* drive C.

Virtualising Intel

PC emulation is easier if you already have an Intel or clone CPU. *DOSemu*, *Plex86* and *VMware* use x86 protected mode to create a virtual processor. Most instructions run directly, but exception handling and hardware abstraction mean it still takes a lot of effort to implement a virtual PC.

DOSemu

DOSemu ships in most Linux x86 distros. It is launched with the commands **dos**, **xdos** or **dosexe**. The latter correspond to **dos** with **-X** and **-L** options, opening an X window or a DEXE file. Once running, the **dosdebug** allows control and debugging.

DOSemu creates a virtual IBM PC AT in a console or X window. It emulates '286, '386 or '486 Intel chips in real mode (without the flat 32 bit addressing added for the '386) with paged memory expansion and support for BIOS access to drives, serial and printer ports.

The default **xdos** offers 640K main memory; 1MB of XMS expanded RAM; an 80386 CPU, 80387 FPU and 1993 PC-AT BIOS; VGA VESA 2 graphics and PS/2 mouse emulation.

DOSemu is configured by '/etc/dosemu.conf' with user access rights defined in '/etc/dosemu.users'. Switches **-2**, **-3** and **-4** set the CPU version; some MS-DOS programs require the 286 mode. Options **-A** to **-C** let you boot from a real or emulated drive. These can be disk image files, MS-DOS drives like '/dev/hdc', partitions like '/dev/hda9', or the Linux file system – seen as a network drive through *emufs*. Floppies can be disk images made with **dd** or real media via devices like '/dev/fd0'.

DEXE images are bootable hard files with a single MS-DOS application inside. **mkfatimage16** is used to create disk image files in MS-DOS format, copying in files specified on the command line:

```
mkfatimage16 -t 500 -p -l DOS -f dos.boot
```

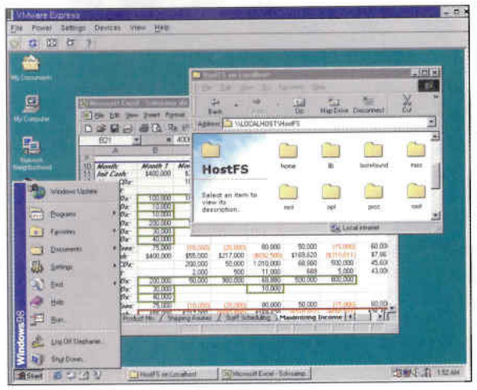
makes a 170M (500 track, AT-default 4 head 17 sector) disk image, internally labelled DOS, in the local file 'dos.boot'. Linux *mtools* can access this given a drive definition line in '/etc/mtools.conf ' like:

```
drive h: file="/path/dos.boot" partition=1 offset=128
```

The last parameters help *mtools* find the root of the image. Now you can read the directory with **mdir h:**, insert and get files out with **mcopy**, and commands like **mren** and **mdel** mimic their un-prefixed MS-DOS counterparts. Link from '/var/lib/dossemu/drives/d' to 'dos.boot' and that appears as drive D inside *DOSemu*. SuSE include a simple FreeDOS boot image as drive C, to get you started.

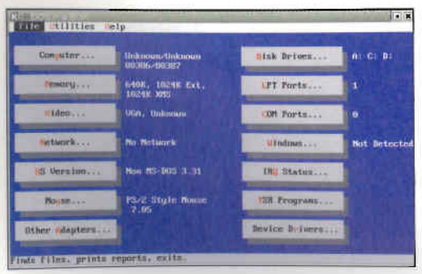
DOSemu has commands to integrate Unix and MS-DOS. You can run Linux commands from DOS by prefixing them with UNIX, so **UNIX dir** ~ lists home directory files, in the DOS window.

DOSemu suits the Linux console, where it can have raw access to the keyboard, bypassing a lot of GUI overhead, but



VMware express running Windows 98 on a Gnome desktop, with the RedHat root visible from Windows Explorer.

Below: **MSD takes a guess at our DOSemu configuration.**
Below right: **DOSemu resizes automatically to fit the EGA graphics of the PC port of the original SimCity.**



A mess of DOSes

The stranglehold of legacy software and hardware

Two software components underpin PC-compatibles from the original 1981 IBM PC launched to contemporary Windows 98 boxes. BIOS and DOS have no direct counterpart in Linux. This box explains their implications for 'PC compatibility'.

DOS stands for Disk Operating System, a general name for software that links applications and peripherals. The term pre-dates the IBM-compatible personal computer; like PC or Hoover, it's often hard to know if it's meant specifically or generically. This is especially important for emulators – PC-compatible DOSes include PC-DOS, DR-DOS, OpenDOS and FreeDOS as well as Microsoft's MS-DOS, and there are many incompatible ones. As some of those names suggest, you don't necessarily have to pay to run DOS on a PC clone, these days.

IBM's DOS/360 established the term in the sixties, but the first micro DOS was a component of Digital Research CP/M. Other pre-Microsoft DOSes included Apple's DOS3, Tandy's TRS-DOS, NEWDOS, DOS+, CDOS and MDOS. But CP/M influenced the PC most, and was what IBM originally wanted.

The resident part of CP/M is FDOS, comprising BDOS and BIOS. The Basic Input Output System (BIOS) sits between BDOS and hardware. It's the lowest-level software interface in CP/M, and a block of code is in every PC's ROM.

CP/M's BDOS (Basic Disk Operating System) dealt with applications and the rudimentary shell, or Console Command Processor, overlaid in remaining memory. Digital Research failed to agree terms with IBM, put off first by the giant firm's NDA which required them to promise not to say anything that they might later consider secret, then by a bid for unlimited rights for $250,000 rather than the customary $10-per-copy royalty.

IBM also visited Bill Gates at Microsoft, seeking his *MBASIC* interpreter, and got offered an alternative to CP/M. This became IBM PC-DOS, independently branded by Microsoft and sold as MS-DOS to IBM's rivals, and a global monopoly was born.

Quick and Dirty

In fact Microsoft had no DOS when the deal was struck, and hastily bought in a CP/M clone from Seattle Computer Products. This SCP-DOS, known internally as QDOS for Quick and Dirty Operating System, became MS-DOS 1. It mimics CP/M system calls, with the same 8.3 filenames – up to eight characters, a dot then up to three more – and similar internal structure.

PC DOSes use two chunks of system code, typically called IO.SYS and MSDOS.SYS, fetched by a boot block. These load CONFIG.SYS – a configuration text file detailing device drivers and further SYS extensions,the shell interpreter COMMAND.COM, and startup commands in AUTOEXEC.BAT.

The buggy PC-DOS 1 lacked support for subdirectories and hard drives. MS-DOS 2 added those and offered variants for other computers based on the 8086 processor but with hardware unlike IBMs – often superior designs from ACT, RML and Japanese companies.

MS-DOS 2.3 was the last to support these non-clones: from MS-DOS 3 onwards compatible systems had to have the same basic hardware as an IBM PC, and PC-compatibility moved from a software issue, dictated by MS-DOS, to a shifting hardware standard, led by Intel, Microsoft and IBM. This stifled innovation but laid the foundation for rival OSs on the same hardware, including IBM OS/2, FreeBSD, QNX, BeOS and Linux.

Digital Research did not give up. IBM briefly offered CP/M-86, their version of CP/M for x86 processors, as an option to PC purchasers, forestalling litigation, but PC-DOS predominated as it came with every IBM PC, and Microsoft sold MS-DOS to the clone makers. So Digital Research commissioned a UK team to make their own clone of Microsoft's clone of CP/M, and called that DR-DOS.

This worked well, despite Microsoft's attempts to block compatibility, and was bundled by Amstrad and other PC-clone manufacturers. DR-DOS failed to break the monopoly, but it is significant to Linux and emulator users following its sale to Caldera, who released it freely as OpenDOS.

A third option, FreeDOS, is genuinely Open Source. Source for DR-DOS has yet to surface, despite promises, but FreeDOS is distributed under the GNU GPL. It is not quite as compatible, but is under active development. So you don't need to buy Microsoft's DOS to use MS-DOS software, and our cover-mounted PC emulators are ready to run.

DOS versus Shell

There are tricky differences between MS-DOS and Linux conventions. The MS-DOS directory separator is a backslash \, ASCII 92, rather than the Unix /, code 47, which MS-DOS 1 claimed for parameter switches, akin to - in Unix shells.

Device names are single letters, A and B for floppies and C and later letters (up to a 'lastdrive' limit set in CONFIG.SYS) for other drives. A distinct path is kept for each, so **A:** and **C:** commands switch to the last directory selected on a drive, and full path specifications need a backslash after the drive prefix to anchor them at the root directory.

Wildcard expansion of * is crude compared with Linux globbing. An asterisk before the dot in a file or directory name expands to up to eight question marks that can stand for any character, or up to three after the dot. Names are padded to the right with spaces to fill the fixed 8.3 character format, and case-independent, as only upper-case is stored.

Thus **DIR *** only matches names with no explicit extension (or three spaces) and you need ***.*** to match any name in the current directory. **DIR PRE*** matches a prefix, but **DIR *FIX** matches *every* name without an extension; the initial asterisk expands to eight question marks and characters between the eighth and the dot are ignored!

The MS-DOS command line is primitive but the free *DOSEDIT* enables history and editing within commands, not just delete backwards. The superior FreeDOS shell offers completion. *BATMENU*, also on our coverdisc, aids menu selection for scripts.

Microsoft added utility commands, cloning third-party extensions, but the core remained the same in MS-DOS versions 4, 5 and 6. The MS-DOS version 7 beneath Windows 95 and 98 supports long file names by smearing them across several directory entries intended for short names. The capacity of each root directory is fixed at disk format time, so long names quickly fill the space.

Normal service will be resumed manyana...

Another DOS deviation is the proliferation of memory types. The original PC had 20 bit addressing, allowing for 640K main RAM and 384K for ROM and memory-mapped peripherals. That seemed loads when PCs shipped with 64K, but not for long. Lotus, Intel and Microsoft boiled up a bodge called *LIM*, *EMS*, *XMS* or Expanded memory. This pages sections of extra RAM through a small hole in the 1M space visible to legacy programs.

Twisting the knife, later CPUs supported real Extended (not Expanded) memory, at addresses above one megabyte, but this was invisible to programs running in the 'real mode' of MS-DOS and the original PC architecture. PCs need a non-DOS 'protected' mode to access Extended RAM, while Expanded memory is most plausible in 16 bit segmented 'real' mode.

Time to Thunk

Thus PC emulation reintroduces 'thunking', the practice of calling 16 bit code from 32 bit mode, which was designed out of Linux. That stopped Linux running on 286 systems, but meant that it could treat later chips as relatively orthogonal 32 bit parts, rather than souped-up seventies eight-bitters.

286 code is confined to 64K segments dating back to the original PC 8088 chip and the 1974-vintage 8080 which introduced a 64K address space. Some original PC ROM was auto-translated from 8080 code; in fact the limited registers and addressing modes of the 286 date back even further, to Intel's first 8 bit chip, the 8008 of 1972. Microsoft still struggle with this legacy.

Thunking involves remapping 32 bit registers into 16 bit segment and offset pairs, disabling much of the instruction set, and rethinking the interface between the CPU and memory. Linux and BeOS are 32 bit clean but MS-DOS and Windows versions up to 3.1.1 run mainly in 16 bit mode, unlike the Linux host, and context swaps add much overhead. The interface to BIOS uses 16 bit conventions, which is one reason Linux ignores it after *LILO*.

Even Windows 9x, old versions of NT, and OS/2 thunk furiously, though like Linux they insist on a '386 or later' processor at least capable of running a flat 32 bit memory model. A surprising amount of code still on sale uses the corner of the x86 dedicated to the PC-AT 286 subset which Intel tried but failed to kill almost 20 years ago.

Behaviour that Linux would trap with a segfault and optional core dump is still considered a normal way for Windows processes and libraries to communicate. This impacts on emulation hosts as well as virtual PC environments.

text-based programs often insist on the PC 80x25 character screen layout. Press Ctrl Alt Page Down to quit, rather than Ctrl Alt Del which might reboot Linux rather than just the emulator session! If you need to quit from a script, use **EXITEMU** from the *DOSemu* sample subdirectory.

DOSemu users must navigate through many dud configs, the bane of PCs in general. Windows 3.1 setup failed with 'runtime error R6001, null pointer assignment' – possibly related to FreeDOS – it worked with an MS-DOS 6.22 boot floppy.

Turbo Pascal worked fine, but *Norton Utilities* did not, complaining that disk information exceeded the program`s memory capacity. *RPED* ran but *QLAY* would not because it needed the x86 protected mode. *Reversi* aborted with 'i/o error 99'. *Cosmo* and *Psion Chess* just blanked the screen. Old DOS graphics games *Digger* and *Kiwilife* gave garbled displays. Text ones worked, though much faster than their programmers presumably planned. The PC port of *SimCity* ran, with mouse support, automatically enlarging the X window for its EGA screen.

Commercial emulators

Proprietary Linux emulators focus on Mac and Windows platforms. *Win4Lin* (LXF20) does this for Windows 9x; *VMware* is more general (LXF17); *Plex86* is the Free Software contender.

Suites like *Star Office* and *KOffice* give you as good a chance of decoding a Microsoft *Word* or *Excel* file as a randomly chosen Mac or Windows user. This is not to say that you and the world in general would not be better off using standard and stable file formats. Specialised applications present more of a problem, because they tend to do things and use files like nothing else. There are packages for Windows, MacOS and even MS-DOS with no equal on Unix. At times like this, you really need an emulator for one of the 'other' mainstream systems, such as MacOS, MS-DOS or Windows. If you're running Linux x86 you've almost certainly already got hardware compatible with the last two; Linux 68K and PPC systems have the right chips for Macintosh programs, though they need software assistance to emulate a real Mac (LXF18/19).

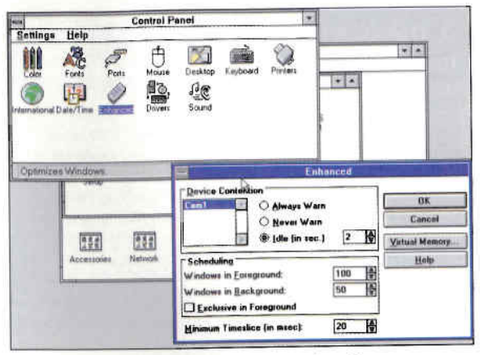
VMware can virtualise any listed host as well as these guests, and other x86 systems too, though without explicit support. NT4 emulation requires service packs 3, 4, 5 and 6A.

VMware 3 Workstation

I tested MS-DOS 6.22 and Windows 95 on the new VMware Workstation release 3. This now allows much bigger emulated hard drives and translucent filesystem emulation, where changes are written elsewhere, to be confirmed, discarded or undone later – potentially better than the real thing!

VMware Workstation 3 supports two hot-plugable USB devices, as well as USB mice and keyboards, emulating PS/2 protocols to suit even pre-USB guests. Streaming media devices may glitch but ones based on the usb-storage protocol, which allows for drive latency, work fine. These include memory card readers, real disk drives, and some cameras and PDAs.

Yes, Win4Lin gives you Windows on your Linux box.



Windows 3.11 prefs make a stab at multi-tasking.

USB printers and scanners should work reliably as such drivers and devices do not insist on a quick turnaround. Nor do TCP/IP connections; they are easily virtualised as they use data formats designed for portability, and offer no timing guarantees.

USB devices cannot be shared simultaneously by *VMware* hosts and guests, but hot-plugging provides a way round this limitation. Devices are available to only one system at a time, and *VMware* signals that they have been unplugged from one and plugged into another when you adjust the USB controller panel.

VMware Workstation 3 comes with a big PDF guide and lots of HTML support, but the setup illustrations presume a Window host, with Linux specifics irritatingly interspersed. Installation is a breeze, with a helpful GUI wizard. Custom code optimises device handling for each guest system, on both sides of the emulation.

Networking is very strong – you can use NAT to link out via virtual DHCP, sharing your host's IP, set up a 'bridged net' so your virtual machine looks like another computer on the host's network, or set up a private network between guest systems on a single machine. *Win4Lin* gives more direct access to host files, but *VMware* has more capabilities.

VMware Express is a cut-down package, targeted at much the same market as *Win4Lin*, with a keen $49 price tag. *Express* offers the same networking and Linux window or full screen display, but it is only designed to virtualise Windows 95 or 98.

Express lets you install Win9x without repartitioning, but won't boot from existing Windows partitions and does not support multiple emulations on one machine. It also lacks the 'instant restore' snapshot feature of *Workstation 3*, and the options for temporary or undoable disk writes. Both versions are on our cover disc; download a free 30 day keyfile to try before you buy.

WINE

There is another way to run Windows binaries – by replacing the Microsoft code with open-source routines that do the same job. This is the approach taken by *WINE*, which stands recursively for WINE Is Not an Emulator. Like *VMware*, it does not emulate the processor, so *WINE* requires an Intel-compatible host. But whereas *Win4Lin* and *VMware* aim to hide Linux from real Windows code, so they can co-exist, WINE sets out to provide Linux-native libraries that Windows applications can call as if they were running within Microsoft's OS.

This is an even bigger job than virtualising PC hardware, because Windows is a baroque, poorly-documented proprietary system. Microsoft conceal its inner workings, yet often emulation involves knowing how something works, as well as how it might be used. But at least the interface on the Linux side tends to be well documented, and there is no shortage of Windows programs for empirical testing of *WINE* components.

Corel support the development of *WINE* and it's been crucial to their release of *WordPerfect*, *PhotoPaint* and *Corel Draw* for Linux. Early versions of Borland's *Kylix* also relied on *WINE* for Linux compatibility, along with the Qt cross-platform GUI toolkit already used in *Delphi* for Windows. Qt is the mainstay of KDE on Linux – so the cross-platform traffic runs both ways.

The server software for IBM's *Small Business Suite* can run on Linux thanks to *WINE* (LXF13). On the client side, *Lotus Notes* is only available as a Windows binary but *WINE* can bring version *4* or *5* up on Linux. It helps that *Notes* developer Daniel Schwarz is a Linux devotee. The *WinISO* disc image utility is one of many free tools that *WINE* brings from Windows to Linux. The *MGT SAM* emulator for Windows runs nicely under *WINE*, while we wait for the SDL port for Linux.

The good news is that *WINE* runs many Windows programs quite legally, without making us buy any Microsoft system software. The bad news is that *WINE* only supports some Windows system-calls, with gaps, bugs and incompatibilities. There are many programs that *WINE* cannot handle. Given the difference in resources of Microsoft and the *WINE* developers, it's consistently several steps behind the latest Windows facilities.

Application installation is a particular problem with *WINE* while *InstallShield* is not supported. To get round this, install older Windows releases, with fewer dependencies, or resort to a dual boot system. After installing in native Windows you can reboot and run some applications alongside Linux with *WINE*, even if they won't install that way.

Cross-platform

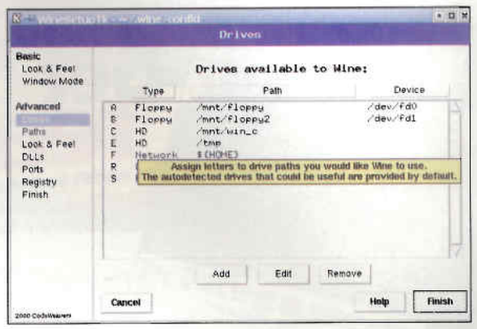
If you'd like to develop new software for both Linux and Windows another definition of 'compatibility' may appeal. *Willows*, formerly known as *TWIN*, is a GPLed implementation of the 32 bit Windows API for all variants of Unix, akin to *WINE* but less dynamic. Programs written for Windows can call corresponding *Willows* libraries; the re-compiled version should run the same way, but under Windows.

TWIN lacks Object Linking and Embedding and can't handle Microsoft programs like *Access* and the old Windows *File and Print Managers*, because those apparently rely on unpublished APIs. Third-party software is more likely to work, but you really need the source.

TWIN can run 16 bit (Windows 2 and 3) applications on non-Intel systems by CPU emulation, but that's slow and does not support Windows 9x, 2000 or NT4 which need 32 bit instructions that are not yet emulated properly. *TWINE* is a project which aims to merge the strengths of *WINE* and *TWIN*, under the more restrictive WINEL licence.

Another compatibility option is Qt, the interface toolkit which underlies KDE. This is free for non-commercial Linux development, but also sold under licence for Windows; so you can write and test your code, and only need to pay TrollTech licence fees when it's working and you need libraries to run the same code on commercial platforms.

Borland's *Kylix* is the most prominent cross-platform Qt application, pervasive as it's an application development system. Bristol and Mainsoft offer non-GPL variations on this theme. You might also consider the free SDL and *Allegro*.



Wine gives Windows applications full access to your drives and network, if you wish.

Prognosis

PC compatibility is a moving target. *Bochs*, *DOSemu* and *PCemu* cope well with MS-DOS programs, but these are obsolescent. *VMware* and *Win4Lin* are effective ways to run PC business programs from within Linux, but they are proprietary like most of the programs they support. *WINE* can run a lot of programs, but tends to be shaky on large packages.

VMware can run non-Microsoft operating systems, as can *MacOnLinux*, the nearest equivalent for Power PCs. *Plex86* offers the potential to virtualise the x86 for free, but lags far behind *VMware* in its hardware support, and has yet to live up to the potential of its original provocative name, *FreeMware*.

There are two areas where we wait at the brink of major progress in PC emulation – 3D graphics and dynamic compiling. PC emulators for non-Intel systems are slow because they must interpret every little fiddly x86 instruction each time it is encountered. But *ArmPhetamine*, *MX*, *Nestra*, *VEST*, *ZM-HT*, and recently *Basilisk* and *UAE* demonstrate that emulation can be much faster if alien code is dynamically translated, rather than interpreted. Work underway to add this feature to Bochs may make Windows emulation more viable on non-Intel architectures.

It's not yet practical to run 3D-intensive PC games or similar programs in Linux, because *VMware* and *Win4Lin* cannot virtualise PC graphics accelerators at the necessary low level, and *WINE* does not support the *DirectX* 3D application program interface. But work is underawy on *WineX*, which promises to bring Windows 3D capabilities to Linux. The project summary says: "TransGaming is developing improvements to the *WINE* project to achieve full Linux compatibility for the Microsoft *DirectX* APIs." We'll be taking a look at the *WineX*-based Mandrake/Transgaming pack soon!

PC emulation links

Follow these links for updates, support and related information

Bochs x86 simulation: http://bochs.sourceforge.net
DirectX on WINE: http://sourceforge.net/projects/winex
DOSemu home: http://www.dosemu.org
DRDOS/OpenDOS: http://planetmirror.com/pub/drdos
FreeDOS: http://www.ibiblio.org/pub/micro/pc-stuff/freedos
Lotus on WINE: http://www.winecentric.com/notes5.shtml
MTools: http://mtools.linux.lu
Plex86: http://www.plex86.org
Twin and Willows: http://www.willows.com
VMware virtual x86s: http://www.vmware.com
WINE headquarters: http://www.winehq.com