DOSEMU

The Linux MS-DOS Emulator Manual, Edition 1
for dosemu Version 0.49
May 1993

by Robert Sanders

This is edition 1 of the DOSEMU documentation; it describes version 0.49 of the Linux MS-DOS emulator. Any resemblance herein to persons or places, real or imaginary, is purely coincidental and most definitely *not* grounds for lawsuits or nastiness of any other kind directed at the author.

(Un)Published by Robert Sanders
38134 GA Tech Station
Atlanta, GA 30332

Printed on 20 May 1993

1 Introductory Stuff

Welcome to the DOSEMU manual! I hope you can find what you need in here.

Let me begin with a general explanation of what exactly dosemu *is*, before I get too bogged down trying to explain what dosemu *does*.

dosemu is a user-level program which uses certain special features of the Linux kernel and the 80386 processor to run MS-DOS in what we in the biz call a *DOS box*. The DOS box, a combination of hardware and software trickery, has these capabilities:

- the ability to *virtualize* all input/output and processor control instructions
- the ability to support the word size and addressing modes of the iAPX86 processor family's *real mode*, while still running within the full protected mode environment
- the ability to trap all DOS and BIOS system calls and emulate such calls as are necessary for proper operation and good performance
- the ability to simulate a hardware environment over which DOS programs are accustomed to having control.
- the ability to provide MS-DOS services through native Linux services; for example, dosemu can provide a virtual hard disk drive which is actually a Linux directory hierarchy.

The hardware component of the DOS box is the 80386's *virtual-8086 mode*, the real mode capability described above. The software component is dosemu.

These are the release dates of dosemu:

0.1, 0.2, 0.3, 0.4
: These were versions of the original dosemu, written by Matthias Lautner. I based my version—which I now consider the "official" version—on his dosemu 0.4.

0.47 January 27, 1993

0.47.7 February 5, 1993

0.48 February 16, 1993

0.48pl1 February 18, 1993
: These versions added console video, raw keyboard, XMS, timer tick, VGA attribute fix, simple printer support, BIOS serial ports, and lots of bug fixes.

0.49 May 20, 1993 The 0.49 release added EMS, a new disk redirector, revamped hard disk support, a more sophisticated printer driver, BIOS-driven VGA support, serial port emulation, keyboard TAMEing, and lots more bug fixes.

If you need me, you can reach me at these addresses[1]:

Internet: gt8134b@prism.gatech.edu

U.S. Mail:
Robert Sanders
38134 GA Tech Station
Atlanta, GA 30332

Enjoy!

---

[1] Even if you don't need me, you can...

1.1 Greetings

Hello, public,

This is dosemu version 0.49. Versions 0.41-0.46 were "internal" versions, and 0.48pl1 is the public release preceding this one. The RCS revision numbers printed in various places throughout the emulator mean very little to either of us, so don't worry about them. The version number (i.e. 0.49) should be printed on the invocation screen, in case you forget which version this is.

This is an ALPHA release; dosemu is far from stable. I do not expect dosemu to reach BETA stage for quite some time. However, this does not mean that you cannot use and enjoy dosemu free of all problems; it just means that you will be the exception if you do.

Dosemu 0.49 is based on dosemu0.4 from Matthias Lautner, but is quite a bit more useable. It is probably already obsolete by the time you read this, however, as I am continually at work making the code even uglier :-). Hopefully, with all this code uglification will come a stray feature or two.

1.2 Mach

I'm sure that you've all heard the expression "Don't reinvent the wheel." Well, it turns out that the kind folks at Carnegie Mellon University, purveyors of the Mach microkernelish operating system, have invented a very fine wheel of a DOS emulator. I did not discover this until well into my enhancing of dosemu, but it would not have made much difference: the Mach operating system is sufficiently different from a traditional UNIX implementation like Linux that a "port" would have been prohibitively difficult. The Mach DOS emulator takes full advantage of the thread capability of Mach, as well as some emulator-specific features added to the Mach kernel.

However, several of the subsystems of their DOS emulator are not only informative to read (which is a great boon in itself), but not *too* difficult to adapt for usage under Linux dosemu. As proof of this, dosemu as of this writing has absorbed two of the Mach emulator's feature subsystems. Below is a list, showing the Linux dosemu files on the left and the dosemu features contained therein:

`mfs.c`

`mfs.h`

`emufs.S` the drive redirector (also called the network redirector, as it is normally used to provide a remote disk drive over a network connection).

`bios_emm.c`

LIM EMS support. The original Mach DOS emulator file supplied version 3.2 of the specification; I have updated it to provide version 4.0.

`machsupport.h`

this is a header file supplying many of the common types and functions used in the Mach DOS emulator's files. Its contents are simply the union of the original Mach DOS emulator's files `base.h` and `bios.h`

An added advantage in my integrating code from the Mach DOS emulator is that, if I modify it carefully so that my Linux-specific changes do not affect the file when compiled under Mach, and my non-OS-specific changes work equally well under both Mach and Linux, then I have effectively benefitted both programs with almost no additional effort.

CMU placed Mach and the Mach DOS emulator under a very non-restrictive license, thanks to which we can legally recycle their code. I have kept the original copyright notices in the relevant files, and also reproduce it here:

1.3 Porting to 386BSD

Ha ha ha ha ha ha ha ha ha ha ha hooooooooooooooo ha ha ha ha ha ha ha ha ha ha ha ha ...

Oh, wow. That's a sweet one. (You can quote me on the above).

People do occasionally ask about just that: porting dosemu to 386BSD. I wouldn't mind helping someone port dosemu, but she had better be clever and committed. Just in case someone's game, here is a list of the Linux dependencies:

- Virtual Console control. mostly in `termio.c` and `video.c`.
- Unprocessed scancodes from a TTY. mostly in `termio.c`
- the method of accessing video card memory (`mmap` of `/dev/kmem`) assumes that the lower megabyte of kernel memory is identity-mapped.
- EMS assumes a particular way of sharing pages within a process's address space. There is probably an easy way of doing this under 386BSD.
- The emulator-code loader assumes the Linux shared library scheme.
- the `vm86()` system call and signal semantics. This is the biggie.

All in all, a better idea is to port the Mach DOS emulator. I heard from some people doing this a while back, but they've been quiet for a while. This could mean that they're busy, or dead.

1.4 Contribution

If you see some area of dosemu that is incomplete, buggy, or altogether nonexistent, and you feel halfway competent to do something about it, *please* consider helping me out by volunteering to work on that area. There's so much to do that I can't do it all by myself. Lately it seems like I can't do *any* of it by myself.

Seriously, I know most of you use Linux to get away from MS-DOS, but dosemu can help us get away from MS-DOS but still use those indispensible programs. If you have something to offer, not matter how large or small, please *please* **please** contribute!

I will maintain a list of people and their projects in this document. The current list is shown below.

- gt8134b@prism.gatech.edu — Howlin' Bob
  dosemu in general, the big picture, the whole enchilada, the buck stops here; integration of various charity work done by others; documentation. For the moment, I consider this guy the main dosemu developer/coordinator.
  Status: needs to be trashed[2].
- jmaclean@fox.nstn.ns.ca — James MacLean
  smart (non-BIOS) VGA graphics support; the save/restore routines which support BIOS-driven video
  Status: Fully functional for non-SVGA modes and integrated into dosemu. SVGA drivers for Trident and ET4000 under development.
- tridge@nimbus.anu.edu.au — Andrew Tridgell
  porting and enhancing the redirector from Mach's DOS emulator. Also some changes to the scrolling functions of the emulator proper, the anti-keyboard-polling mechanism tuned by the HogThreshold directive (See Section 3.3 [Run-time Configuration], page 12), and the int2f "give up time slice" discovery[3].
  Status: Fully functional and integrated into dosemu. Waged in a neverending war with the DOS "network redirector" kludge. Currently working with Stephen Tweedie to correct inconsistencies between MS-DOS and DR-DOS.

This table is here not only as a recognition of their work, and not only as a guide to who's doing what, and not only as filler...wait, maybe it *was* here only as filler.

1.5 Thanks

- Thanks to Matthias Lautner, the true and original author of dosemu. If he hadn't done it first, I wouldn't be doing it now.
- I can't thank Andrew Tridgell enough for his first class work on the redirector.
- ...or James MacLean for the (S)VGA screeen save/restore routines. Without these, graphics under dosemu would be practically unusable.

---

[2] dosemu, not the author—the author does not indulge.

[3] For those of us who wish to multitask under Linux while running dosemu, the latter two are a big win in CPU savings.

- Thanks to *all* those who sent in patches and suggestions; there were some *very* helpful people.
- Thanks to Krishna Balasubramanian for his excellent SYSV IPC implementation. (Not only because dosemu uses SYSV shared memory, but because I learned enough from reading his `shm.c` to write the `mmap()` modifications that EMS support needed).
- Thanks to Adam Goldberg for practical advice and sending me the XMS 3.0 spec.
- Thanks to Michael K. Johnson for early encouragement and advice.
- Thanks to Dennis Marer for sending me spec. sheets (which I still haven't gotten around to using! Hardware emulation WILL happen one day).
- Thanks to Darren Senn for loaning me his EGA/VGA reference book
- Thanks to Stephen Tweedie for ideas and feedback, and especially for helping with the disk redirector.
- And, of course, thanks to Linus Torvalds for creating the Linux miracle!

2 Installation

While *tsx-11.mit.edu* is the home of dosemu, you should be able to find dosemu on almost any Linux ftp site. These are the ones with which I am familiar:

tsx-11.mit.edu
: /pub/linux/ALPHA/dosemu

sunsite.unc.edu
: /pub/Linux/system/Emulators/dosemu

nic.funet.fi
: /pub/OS/Linux/BETA/dosemu

wuarchive.wustl.edu
: /mirrors/linux/??

The final release version of dosemu 0.49 will be called `dosemu0.49.tgz` and will be placed in the dosemu directory on *tsx-11.mit.edu*.

After you've gotten dosemu, but before you boot MS-DOS, there are some things you'll need to do.

- edit the Makefile to suit your system (see README, and below).
- edit '`/etc/dosemu/config`' to give it your disk setup (and whatever other parameters you feel like changing). See Section 3.3 [Run-time Configuration], page 12.
- Compile it. `make config; make dep; make clean; make install` does it best. If you have gcc earlier than 2.3.3, edit the Makefile to give the path to your shared library stubs.
- Make sure your termcap entries have these fields describing keyboard strings:

`kI`	Insert
`kD`	Delete
`kh`	Home
`kH`	End
`ku`	Up arrow
`kd`	Down arrow
`kr`	Right Arrow
`kl`	Left arrow
`kP`	Page Up
`kN`	Page Down
`k1` ... `k0`	Function keys F1 ... F10

  And these fields describing output control sequences:

`cl`	Clear screen
`cm`	Cursor motion
`mr`	Enter reverse video
`md`	Enter double bright
`so`	Enter standout mode

`se`	Leave standout mode
`me`	Turn off all appearance modes
`vi`	Hide cursor
`ve`	Return cursor to normal

Some of the output control sequences can safely go undefined, such as `vi` and `ve`.

I'm still trying to devise a way to map a variety of foreground and background colors, as well as high intensity, onto a very few termcap attributes. Some kind soul produced patches to dosemu 0.48pl1 and color xterm that allow the use of full color. I doubt I will integrate those patches, instead choosing to incorporate the `xdosemu` display mechanism, but I can't give an estimate about when that will happen. Either way, you're free to use the patches if they're updated for dosemu 0.49.

- I recommend that you use libc.so.4.2 or newer. [However, I recommend you NOT use libc.so.4.3, but use libc.so.4.3.2 or newer. libc.so.4.3 apparently has some reentrancy problems which can be exacerbated by signals, and dosemu LOVES signals...I am using 4.3.2 with no problems (yet)]
- If you plan to use raw console video (-c), make a kernel or LILO image spec. that will boot into an 80 column mode. Although it isn't overpolished yet, raw console video mode is nice. You get colors and the DOS character set. Highly recommended if you don't mind 80 column console screens. If you have a monochrome video card, change the define in Makefile. Actually, you can use dosemu with other than 80x25 screen sizes; See Section 5.3 [VGA], page 23.
- If you want to directly access or boot from a DOS hard disk partition, you may need to tell dosemu about your hard disk geometry by adding an entry to the configuration file. See Section 6.4 [Hard Disks], page 31, and Section 3.3 [Run-time Configuration], page 12, for more info.

  A safer route is to use the MFS redirector to access a Linux directory as a DOS network filesystem. See Section 6.3 [Disk Redirector], page 30, for more info.

  You can also use `diskimage` and `hdimage`, and a `ramdrive.sys` created ramdisk, which runs in XMS or EMS memory! See Section 6.5 [Disk Images], page 33, See Chapter 7 [Memory], page 37, for more info.

  NOTE: I have had problems when using an EMS-based ramdrive in conjunction with pkzip 2.04e. This is probably a bug in my code. So, if you want to use RAMDRIVE, I recommend you set it up in XMS memory (which is probably faster anyway).

Many thanks to Andrew Tridgell (`tridge@nimbus.anu.edu.au`) for porting the Mach DOS emulator's redirector, and for other tips/fixes as well. See Section 1.2 [Mach], page 2

2.1 General Info

Here is what you see if you type "dos -?":

```
usage: dos [-ACfckbVtsgxKm234] [-D flags] [-M SIZE] [-P FILE]
[-H|F #disks] > doserr
    -A boot from first defined floppy disk (A)
    -C boot from first defined hard disk (C)
    -f flip definitions for A: and B: floppies
```

```
-c use PC console video (!%)
-k use PC console keyboard  (!)
-D set debug-msg mask to flags (+-)(avkdRWspwgxhi01)
-M set memory size to SIZE kilobytes (!)
-P copy debugging output to FILE
-b map VGA BIOS into emulator RAM (%)
-H specify number of hard disks (1 or 2)
-F specify number of floppy disks (1-4)
-V use the BIOS VGA "emulation" (implies -c and -b)
-N No boot of DOS
-t deliver time interrupt 9
-s try new screen size code (COMPLETELY BROKEN)(#)
-x <SIZE> enable XMS with SIZE kilobytes
-e <SIZE> enable EMS with SIZE kilobytes (!)
-2,3,4 choose 286, 386 or 486 CPU
-K Do int9 within PollKeyboard (!#)

 (!) means BE CAREFUL! READ THE DOCS FIRST!
 (%) marks those options which require dos be run as root (i.e. suid)
 (#) marks options which do not fully work yet
```

The invocation I use under an 80x25 console (when not debugging) is this:

```
dos -Vcbkx 4096 -P /dev/tty8 -D0 > dbg
```

Under an xterm[1] :

```
dos -b -P /dev/tty8 > dbg
```

You **must** have any required disk image files in the current directory! This includes `diskimage` and `hdimage`. You can, of course, use symlinks instead.

2.2 Updating from 0.48pl1

This is a summary of the 0.48pl1 -> 0.49 changes *that affect configuration or setup.* See Chapter 11 [Changes], page 45, for a complete list of changes to dosemu.

- The format of hdimage has changed; you must add a 128-byte header to any hdimage you used in 0.48pl1. Use the *mkhdimage* command to create the appropriate header. You pass as its arguments the geometry of the hdimage file. For example, if you use the default hdmage size, this sequence of commands would work:

  ```
  mv hdimage hdimage.old
  mkhdimage -h 4 -s 17 -c 40 | cat - hdimage.old > hdimage
  ```

  If you use the default hdimage size, "make convhd" may or may not do this for you. You're probably better off doing it by hand.
- You can now use direct *partition* access instead of direct *disk* access. See Section 6.4 [Hard Disks], page 31, for information on this.

[1] I recommend that you resize the xterm to 25 lines first. The standard 24 is insufficient. Larger sizes will have no additional effect.

- The disk redirector has changed. You need to discard `linux.exe` and replace it with a new program, `emufs.sys`. `linux.exe` will no longer work, and might cause unpredictable results. See Section 6.3 [Disk Redirector], page 30 for more information on this.
- The `Makefile` configuration has changed. Allmost all configuration has been moved into the run-time configuration file '`/etc/dosemu/config`'. See Chapter 3 [Configuration], page 11, for more details on configuring dosemu.
- dosemu 0.49 requires 0.99pl7 kernels or newer with certain patches applied. See Chapter 3 [Configuration], page 11 for more details.
- A BIOS-driven VGA graphics mode has been added. See Section 5.3 [VGA], page 23, for more details on graphics under dosemu.
- the `exitemu.com` program has changed. Discard any old copies, as they will no longer work.
- See Section 2.1 [General Info], page 8, and See Chapter 11 [Changes], page 45, to see what the new dosemu options are. In particular, -e and -x have changed.

3 Configuration

This chapter tells about configuration of Linux dosemu.

3.1 Requirements

These are the requirements for running dosemu 0.49.

- dosemu 0.49 requires that you run one of these kernel configurations:
  - Linux version 0.99pl7 or 0.99pl7A with `ipcdelta` and `mmap.diff` patches applied[1].
  - Linux version 0.99pl8 with `ipcdelta`[2], `ALPHA-diff`, and `mmap.diff` patches applied.
  - Linux version 0.99pl9 or later with the `ipcdelta` and `mmap.diff` patches applied.
- gcc 2.2.2d7 or newer to compile dosemu
- libc 4.2 or newer, excepting libc 4.3 because of reentrancy problems. libc 4.3.2 and libc 4.3.3 are fine.
- a copy of MS-DOS versions 3.3–6.0. MS-DOS 6.0 compatibility was recently added to the disk redirector and has not been thoroughly tested. The author uses MS-DOS 5.0, so versions other than that may show imcompatibilities with some areas of dosemu.

3.2 Compile-time Configuration

At present, very little of dosemu is compile-time configured; that is, a very few of the dosemu configuration options require that you recompile dosemu for your changes to take effect. The basic sequence of events is this:

1. Edit the `Makefile` to reflect the configuration you desire
2. Create a new configuration file `config.h` by typing the command `make config`
3. re- make and install the emulator with the commands

```
make clean ; make dep ; make config ; make install
```

However, most of dosemu's configuration is effected through a file which dosemu parses at run-time, '`/etc/dosemu/config`'. See Section 3.3 [Run-time Configuration], page 12, for more information.

Here is a list of the variables you may set in '`Makefile`':

`KEYBOARD` what nationality your keyboard is

---

[1] If you will not use EMS emulation, then this patch is not necessary for any of the kernel versions

[2] The ipcdelta patches do not patch cleanly into kernels later than 0.99pl7. You'll need to manually edit two files, '`linux/kernel/sys.c`' and '`linux/kernel/fork.c`'. Hopefully an ipcgamma set of patches for a later kernel will be released soon.

3.3 Run-time Configuration

Unlike previous versions, which were configured primarily through compile-time preprocessor directives speified in the Makefile, dosemu version 0.49 receives most of its configuration information through a run-time-parsed plaintext configuration file, usually kept in '`/etc/dosemu/config`'.

The configuration file is made up of a series of *statements* separated by whitespace. A statement is composed of one or more *words*, which is a one or more characters bounded by whitespace. A statement has a property called *type*, which can be one of the following:

predicate a single word which specifies a boolean condition (on/off or true/false)

value a word which has one or more *arguments*, which appear after it. This usually sets some parameter of dosemu's operation.

list a word which begins a bracket-enclosed list of other statements. This is used to group related statements, like the many parameters for a single disk.

comment A comment is ignored. These are the comment formats:
Anything between the C-language comment strings /* and */
Anything from one of the characters `#` or `;` until the end of the line.
The C-like comment delimiters must be separated by whitespace.

dosemu's config file parser is not case-sensitive. That is, words are not differentiated by capitalization. However, certain *arguments* may be case-sensitive, like the filename given to the disk config and printer options; this is a function of how the arguments are used. It never hurts to keep things exact, so when in doubt ...

Here is a list of all the simple statements, their types, and any arguments. NOTE: none of these work yet; rely on the compile-time configuration information.

bootA Predicate
bootC Predicate
Specifies which disk to boot from; the default is C

cpu *model* Value
model can be 80286, 80386, or 80486. Alternately, you can simply specify the *x* in 80x86. Specifies the type of CPU EFLAGS emulation desired. See Section 8.2 [CPU type], page 39.

DebugFlags *flags* Value
sets the debugging flag set to *flags*. See Section 4.3 [Debugging Messages], page 18.

HogThreshold *usecs* Value
The minimum number of microseconds (millionths of a second) which must pass between int16 keyboard requests for a program to *not* be considered a keyboard hog. Defaults to 20000. You may wish to raise this number if the CPU usage of dosemu is too high; however, too high a number will result in poor program performance.

Keybint *flag* Boolean
Specifies whether the keyboard interrupt 9 will be delivered. Default off.

FastFloppy *flag* Boolean
If *flag* equals "on", dosemu will use a faster, but more unreliable, method of accessing real floppy disk drives. If this directive is specified, it may not be safe to remove a floppy disk from the drive until at least 5 seconds have passed from the last access. Turning this feature off will make floppy access safer at a great expense in speed. Default on.

MathCO *flag* Boolean

If the argument is a true synonym (yes, on, 1), then the int15h configuration information word will flag the existence of a math coprocessor.

Ports *list* List

Enables dosemu access to the ports specified within list. A port is expressed as a decimal or hexadecimal number; decimal is assumed unless the number is prefixed with the C-style `0x`. A range of ports may be specified by `range` *begin end*, where *begin* is the first port, and *end* is the last. This example enables access to I/O ports 0x42, 0x61, and 0x300–0x310:

```
ports { 0x42 0x61 range 0x300 0x310 }
```

Speaker *mode* Value

mode can be one of three values:

off no sound at all

on

native Allows access to I/O addresses 0x42 and 0x61. This may or may not be dangerous. It is definitely a bad idea for non-console access.

emulated Sends an ASCII BEL (7) to the tty when a DOS program generates sound. No attempt is made with the current emulation to match the BEL pattern to what native access would sound like. This may change.

RawKeyboard Predicate

If specified, dosemu will attempt to use RAW keyboard mode.

Timer *frequency* Value

Specifies the frequency of timer ticks in hertz. Default is 18.2. This is a debugging aid; dosemu will eventually deduce the requested rate from the virtual timer settings.

Timint *flag* Boolean

Specifies whether the timer interrupt 8 will be delivered. Default on.

NOTE: the current implementation of the timer interrupt is tied in with the periodic screen-refresh code. If you are not using console video, setting this value any higher than 18 is a bad idea. In fact, if your screen is refreshing too quickly, try this with a lower number.

See Section 6.6 [Disk Configuration], page 34, for more information about configuring dosemu's disks.

See Section 7.4 [Memory Configuration], page 38, for information about configuring dosemu's memory management.

See Section 6.2 [Printers], page 28, for more information about configuring dosemu's virtual printers.

See Section 6.1 [Serial Ports], page 27, for more information about configuring dosemu's virtual serial ports.

See Section 5.4 [Video Configuration], page 25, for more information about configuring dosemu's display.

For the words which configure one of a series of objects—which includes hard disks, floppy disks, and printers—the order in which you specify the separate devices is vital. The first disk list specified configures the first hard disk, the next configures the second hard disk; configuration works similarly for floppy disks and printers. So, put the configuration entries for each category of device in the order you wish them to appear to DOS.

This is a sample /etc/dosemu/config file:

```
# Linux dosemu 0.49 configuration file
# Robert Sanders, gt8134b@prism.gatech.edu, 5/16/93
#

# debug +Iv-dkmx
# "debug -a" turns all messages off, used when I'm not debugging.
debug -a

# NOTE: you will not be able to share the mouse with XFree86 because
# the mouse-sharing features of dosemu are broken!

serial { device /dev/cua0 }
serial { device /dev/cua1 }

# These two are set to the defaults.  keybint is still unstable,
# and timint is necessary for many programs.  These are mostly
# useful as debugging aids.

keybint off
timint on

# these are all the different video configurations I use for testing
# the last one is the one I use the most

# video { vga console graphics vbios_mmap }
# video { vga console graphics  vbios_file /etc/dosemu/vbios }
# video {  vga console }
video { vga console graphics chipset et4000 memsize 1024 vbios_mmap }

# if you have an ATI VGA Wonder card, uncomment the next line
# ports { 0x1ce 0x1cf }

RawKeyboard         # don't include this if you don't want it
mathco on           # on or off
cpu 80486           # or 80286 or 80386
bootC               # or bootA
xms 1000            # XMS size in K, or off
ems off             # EMS size in K, or off
dosmem 640
fastfloppy on       # set to off for safer but much slower floppies

ports { 0x388 0x389 }  # for SimEarth
speaker native         # also off or emulated
```

```
#******************* HARD DISKS *******************************
disk { image "hdimage" }
disk { partition "/dev/hda1" 1 }        # 1st partition on 1st disk

#******************* FLOPPY DISKS ******************************
floppy { device /dev/fd0 fiveinch }
floppy { device /dev/fd1 threeinch }

printer { options "%s" command "lpr" timeout 20 }
printer { options "-p %s" command "lpr" timeout 10 }    # pr format it
```

Use the supplied program `/etc/dosemu/mkpartition` to set up the appropriate other files in /etc/dosemu. See Section 6.4 [Hard Disks], page 31, for more details.

4 Running dosemu

4.1 Booting

To boot dosemu for the first time, all you need do is this:

1. Create a bootable disk under real MS-DOS and put these programs on it:
   - fdisk
   - format
   - sys
   - command.com

   Then, enter Linux and create a `diskimage` file from that boot. See Section 6.5.2 [Creating diskimage], page 34.
2. Boot dosemu from the `diskimage` file:

   ```
   dos -A > dosdbg
   ```

   The dosdbg file holds dosemu's diagnostic debugging messages. See Section 4.3 [Debugging Messages], page 18

Now you've booted. If you wish, you may go on to create a hdimage file.

1. Run fdisk. Make sure that the first disk has a primary partition. See *The DOS Reference Manual* for more information about using fdisk.
2. Exit dosemu and then reboot (as in step 2 above). See Section 4.2 [Exiting], page 18.
3. If the hdimage does not already have a DOS filesystem on it, create one with this line:

   ```
   format c:
   ```

   Make sure that it is bootable with these commands:

   ```
   sys c:
   c:
   ```
4. Exit dosemu. You should now be able to invoke dosemu with the `-C` option to boot from hard disk.

You may boot DOS under dosemu from any of these disk devices:

- A floppy image (`diskimage`). See Section 6.5.2 [Creating diskimage], page 34
- A hard drive image (`hdimage`). See Section 6.5.1 [Creating hdimage], page 33
- A real floppy drive. See Chapter 3 [Configuration], page 11
- A real direct-accessed DOS hard drive (both SPA and WDA). See Section 6.4 [Hard Disks], page 31.

4.2 Exiting

To kill the dos emulator, execute the EXITEMU.COM file found on the included hdimage, or send a `SIGTERM` or `SIGHUP` signal to it, or in RAW keyboard mode, type `ctrl-alt-pgdn`. If you need to simulate a ctrl-break in the emulator, you can send it a `SIGQUIT` from somewhere else. (Of course, in RAW keyboard mode, `ctrl-break` does what it's supposed to).

Send a `SIGKILL` to the emulator only as a last resort. This will not allow the emulator to restore the state of the tty, restore the state of the screen, or flush all open files, including the debugging file. Use `SIGTERM` or `SIGHUP` instead, if at all possible.

4.3 Debugging Messages

The emulator, being so early *alpha*, has a considerable amount of code dedicated to debugging itself. The most noticeable effect of all this code is the plethora of debugging messages printed to stdout. Some classes of messages are printed out, while others are suppressed. You, the intrepid user, can choose which classes of debugging messages are suppressed.

Dosemu takes an option `-D` *flags*, where *flags* is a string of letters which specify which options to print or suppress. Dosemu parses this string from left to right. You may also set the debugging flags in the run-time configuration file. See Section 3.3 [Run-time Configuration], page 12.

`+` turns the following options on; if the debug level is 0, sets it to 1. otherwise, leaves it the same. the initial state is `+` (level = 1).

`-` turns the following options off (sets debug level to 0)

`a` turns all the options on/off, depending on whether `+`/- is set

`0...9` sets the debug level. 0 is off (same effect as `-`, 1 is the least verbose level, 9 is the most verbose level)

`$` where $ is a letter from the valid class list (see below), turns that option off/on depending on the debug level.

Debugging message classes:

d	disk	v	video	R	disk Reads
k	keyboard	i	port I/O	W	disk Writes
s	serial	p	printer	h	hardware
w	warnings	g	general	x	XMS
m	mouse	I	IPC	E	EMS
c	configuration				

Any debugging classes following a "`+`" character, up to a "-" character, will be turned on (non-suppressed). Any after a "-" character, up to a "`+`" character, will be suppressed. The character "a" acts like a string of all possible debugging classes, so "`+`a" turns on all debugging messages, and "-a" turns off all debugging messages. The characters "0" and "1" are also special: "0" turns off all debugging messages, and "1" turns on all debugging messages.

There is an assumed "`+`" at the beginning of the FLAGS string. Some classes, such as error, can not be turned off. You must tolerate these; you could always redirect stdout to /dev/null if you REALLY don't want them.

Some examples:

`-D-a` turn off all messages (except errors)

`-D+a-v`

`-D1-v` turn on all debugging messages but video

`-D+kd` turn on keyboard and disk messages; other options are left at the default.

`-D-a+RW` turn on only disk READ and WRITE messages

Any option letter can occur in any place. Even pointless combinations, such as "-D01-a-1+0", will be parsed without error, so be careful. Some options are set by default, some are clear. This is subject to my whim, and will probably change between releases. You can ensure which are set by always explicitly specifying them.

5 Input/Output

This is the section dealing with the various ways dosemu gets its info to you.

5.1 Keyboard

When dosemu uses RAW keyboard on the Linux console, it causes the kernel to pass every key scancode directly to the emulator's stdin without performing any interpretation or translation. Therefore, it is the emulator's responsibility to translate key positions into ASCII codes.

The current emulator has perfect support for only the US keyboard. Some foreign keyboards which do not depend on dead keys/diacriticals will also work.

You may select the keyboard you wish to use in the same way you would select it for the kernel: edit the Makefile so that you have only one "KEYBOARD=" line uncommented. The default setup is a U.S. keyboard.

In RAW keyboard mode, VC switching is accomplished by using `LeftAlt-Fn`, where `RightAlt-Fn` gives you an alt-functionkey as in normal DOS. Otherwise, almost any key combination should act exactly as it would under native DOS. (write me if it doesn't)

XFree86 1.2 uses `ctrl-alt-Fn` to switch between consoles. Bother. If you want this, write me, and I'll think about it.

Some special keys under RAW keyboard mode:

`ctrl-scrlock`
: shows the first 0x32 int vectors on debug output

`alt-scrlock`
: show the vm86 registers

`rshift-scrlock`
: generate an int8 (timer tick)

`lshift-scrlock`
: generate an int9 (keyboard service)

`ctrl-break`
: ctrl-break as under DOS.

`ctrl-alt-pgup`
: reboot DOS. imperfectly done. don't trust this.

`ctrl-alt-pgdn`
: exit the emulator (same as using EXITEMU.COM)

`pause`
: pause/unpause dosemu. Extremely useful.

(most of these are useful for debugging purposes only)

I chose not to use `ctrl-alt-del` to reboot dosemu because it's a bad habit to get into. I accidentally rebooted my machine a couple of times, and decided that authenticity was somewhat less important than safety.

If you're using an unstable version of dosemu like one of the test releases, then you might wish to manually insert this patch into your linux kernel source.

file: `linux/kernel/chr_drv/keyboard.c` line: approximately line 103 in kernel version 0.99pl7a

```
          if (vc_kbd_flag(kbd,VC_RAW)) {
                  kbd_flags = 0;
+                 if (scancode == 0x54) clr_vc_kbd_flag(kbd,VC_RAW);
                  put_queue(scancode);
                  goto end_kbd_intr;
          }
+         else if (scancode == 0x54) set_vc_kbd_flag(kbd,VC_RAW);
          if (scancode == 0xe0) {
                  set_kbd_dead(KGD_E0);
                  goto end_kbd_intr;
```

This rather crudely redefines the `SysRq` key, generated by pressing `RightAlt-PrintScreen`, to toggle the `RAW` keyboard state. If you are using dosemu in `RAW` console keyboard mode, then this patch can be used to restore your keyboard's sanity after a nasty dosemu crash. I have to use this very infrequently when testing serious dosemu changes, and never with a "production" version of dosemu.

Once again, I recommend you NOT use the -k (raw console keyboard) option unless you really need it or are sure you can handle the risks, and that if you do, you apply the kernel patch listed above. If the emulator crashes (as it so rarely does :-), you might be left in suspended animation. You can now exit DOS with CTRL-ALT-PGDN , so you'll most likely not get stuck. However, a REALLY serious crash might leave the keyboard in RAW mode, and then, unless you applied my tiny patch, you really are stuck. Since I made the keyboard code safer (in 0.47.7), I've had this happen ONCE, and that was caused by what I suspect to be a kernel bug. So you're probably safe.

Also, the multinational support with RAW keyboard mode is very sketchy. I have the keyboard types selectable in the Makefile (like the kernel does), but I for some reason removed the deadkey/diacritical code when I first implemented RAW mode (my scancode parsing code is lifted wholesale from the kernel. Thanks, Linus! :-) I dunno why. Anyway, until I put it back in, you foreigners are probably disadvantaged. And the DOS "keyb" command crashes the emulator, so don't get any ideas. :-)

Could someone who **uses** diacriticals write me to confirm this? Do I need to do it, or can I put it at the bottom of my TODO list?

You'll need to use RAW keyboard mode to run WordPerfect, as it uses all sorts of godawful ctrl/alt-fkeys. I'm thinking of ways to support these sorts of combinations without RAW keyboard mode, but nothing really seems "transparent" enough yet.

5.2 Console Video

Hmmm. This is one of the first features I added to the DOS emulator, and I find it one of the most useful. Basically, when you enable console video by giving dosemu the "-c" flag, dosemu bypasses the whole tty universe and writes directly to video RAM.

The advantages:

- very fast
- you get color and PC graphics characters

The disadvantages:

- still doesn't support screen widths greater than 80 characters

- only works on the console (i.e. not over a serial line, not over a network)

With XFree86 1.2 you have the ability to switch virtual consoles even while running an X session. So, the best environment for X users is probably to boot Linux into an 80x25 screen size, run dosemu on one VC, and run XFree86 1.2 on another.

Just in case you're wondering, and because someone asked, if you want to start dosemu from a console, say 1, and run it on another console, say 7, you'd do it like this (add your own favorite options):

```
dos -ck > dosdebug </dev/tty7  2>/dev/tty7 &
```

For those of you who like text screen sizes other than 80x25, I apologize. I am working on support for these, but it hasn't seemed urgent, so it hasn't gotten done. The present way to coerce dosemu into using a screen size other than 80x25 is to use the BIOS-driven video emulation along with some ANSI.SYS or replacement driver. After booting dosemu, you'll have to run some program to set the screen size to the desired mode, as the video card initialization will put dosemu into 80x25 test mode. See Section 5.3 [VGA], page 23.

If you don't use console video, dosemu reverts to the old method of screen refresh, which is to translate the DOS virtual-screen memory into a normal stream of ASCII characters and terminal control sequences (read from your /etc/termcap). This method is much slower, doesn't support color, and doesn't support most of the PC graphics characters. However, you can use this in an xterm or over a serial line.

There is a modified DOS emulator (xdosemu) whose display routines are based upon color xterm. xdosemu runs in its own native X window. It supports color, enhanced keyboard (like my "-k" RAW keyboard mode), most PC graphics characters, and its output should be faster than my dosemu's termcap output to an xterm. The newest version (xdosemu 0.3) has many of dosemu 0.48pl1's features, so by all means give it a try.

You can get it on tsx-11.mit.edu as /pub/linux/ALPHA/dosemu/xdos03.tgz (it is a tar'ed and gzip'ed file).

In addition, I will one day incorporate some form of X support, whether it is taken from xdosemu or not.

5.3 VGA

Well, it isn't what I wanted, but it's something. Dosemu 0.49 can now run your video card's native BIOS within a dosemu session to provide VGA compatibility[1].

The simplest way of enabling this cheap emulation is to specify the `-V` option on the dosemu command line, or to specify the proper configuration words in the run-time configuration file. See Section 2.1 [General Info], page 8, for more information on command-line options. See Section 3.3 [Run-time Configuration], page 12, for more information about the configuration file.

These methods initialize the video card's BIOS by enabling the full range of video I/O ports and making a far call to the standard address of a video card's initialization routine (c000:0003). The video card should show its banner, then return to the DOS boot sequence.

[1] It should be possible to implement CGA/EGA/MDA compatibility also, but there are added difficulties. Contact me if you are interested.

You must enable graphics in this way if you wish to use another video driver such as ANSI.SYS.

To return to normal mode, run `vgaoff.com`. This simply turns off I/O permissions and revectors the int10h vector to the normal dosemu routine.

This is a list of programs I've tried with some or full success:

- tlivesa.com, a VESA extender for ET4000 cards
- dosshell in ALL the video modes, graphics and text.
- WordPerfect 5.0 page-preview with the driver set to IBM 640X480 VGA.
- Picture viewers:

  dvpeg
  jpg_menu JPEG & GIF viewers. Both of these detected and used the 15-bit HiColor capabilities of my ET4000 card.

  cview JPEG & GIF viewer. requires `tlivesa.com`; also utilized the 15- and 16- bit HiColor modes.

  cshow the standard CompuServe GIF viewer
- galactix. This is your standard arcade game. Run this with the `-t` command-line option and the speaker ports enabled.
- F19, a flight simulator
- SimEarth[2]
- SimCity
- Flight Simulator (with some keyboard hacks not fully implemented yet)
- Windows 3.0 in real mode (with a lot of hacks not fully implemented yet)
- fractint (using `tlivesa` for the 1024x768x256c mode)
- dmode and vmodeb; these are mode switchers for the ET4000. dmode can also be used to set certain operational parameters of the video card, such as default refresh rates for a given video mode.
- norton utilities in graphics mode; the text mode also ran better with VGA on.
- Scorched Earth, a video game

If you try it out, please send me what programs you got it to work with, what modes worked, and how well.

You may choose BIOS-driven graphics through the configuration file with a directive like `video { vga console graphics }`. See Section 5.4 [Video Configuration], page 25.

If a program crashes and leaves you screen in an indeterminate state, simply exit dosemu, boot a new one, and run `vgaon.com`. This will re-initialize your video card to its bootup state.

You can now switch virtual consoles when in BIOS VGA mode, with some restrictions. Super-VGA modes are only partially supported, and are not yet recommended for use.

---

[2] Give dosemu access to the ports 0x388 and 0x389 if you wish to run SimEarth. I don't know what these ports do or I would have dosemu emulate them.

5.4 Video Configuration

This section describes the run-time configuration options pertaining to dosemu's display. See Section 3.3 [Run-time Configuration], page 12, for an introduction to the configuration file.

The `video` word begins a list statement which specifies the combination of techniques dosemu is to use to provide a video display. The general format is this:

`video {` [*value arg*] ... [*predicate*] ... `}`

These are the video words:

VGA Predicate
EGA Predicate
CGA Predicate
MDA Predicate

Informs dosemu of the particular type of color display physically present in the machine. *EGA/CGA* are treated identically, while *VGA* must be enabled for graphics to work (this may change).

MDA is different from the *EGA/CGA* only in that it uses a different address range for display memory.

BUG: the *MDA* directive currently does not work. I hope to have this fixed fairly soon, but I need volunteers to help (hint hint).

Chipset *type* Value

Specify the chipset used in the SVGA card. Currently, only *Trident* and *ET4000* are supported. The word *Trident* refers to the T8900C; other models may or may not work.

Chunks *number* Value

Specify the granularity with which dosemu will examine text mode lines for change when updating non-console screens. A value of 1 causes dosemu to update the entire line if it has changed; a value of 4 causes dosemu to update only the quarter portions of the line that have changed. Too high a value may cause performance degradation because of the overhead of cursor positioning. Default is 1.

Console Predicate

Causes dosemu to activate the features only available on the console *if* dosemu is being run from the console. These features include direct screen updates and graphics support.

FullRestore Predicate
PartialRestore Predicate

FullRestore causes dosemu to save all the card's state and memory upon a console switch. *PartialRestore* only saves that part of the memory which may be altered by text-mode Linux. *PartialRestore* is sufficient unless one is running the X Window System concurrently with dosemu.

Graphics Predicate

Allows dosemu to use graphics modes. At present, this only works with *VGA* cards. Note that, unless the proper SVGA chipset is specified along with this predicate, dosemu may incorrectly restore SVGA text modes when switching Virtual Consoles and exiting. For example, dosemu works perfectly with my ET4000 when in the 100x40 mode, but does not properly restore the 132x25 screen.

MemSize *size* Value

Specifies the amount of memory present on the VGA card in units of kilobytes. Typical values are 128, 256, 512, and 1024 kilobytes.

VBIOS_copy Predicate
VBIOS_mmap Predicate
VBIOS_file *filename* Value

These three predicates specify the method dosemu will use to obtain access to your VGA BIOS ROM.

vbios_mmap causes dosemu to use the `mmap` system call to directly map the BIOS into the dosemu process's address space. This has the advantage of making the ROM write-protected, and is slightly more memory efficient.

vbios_copy causes dosemu to copy the entire ROM from '`/dev/mem`' into the dosemu's address space.

vbios_file causes dosemu to read the disk file *filename* into memory. This is useful for experimenting with alternate BIOS images. To create a file calle '`vbios`' with an image of your BIOS ROM in it, execute the following command as root:

```
/etc/dosemu/getrom > vbios
```

See Section 8.1 [BIOS], page 39, for an explanation of why these directives are necessary.

vbios_mmap and *vbios_copy* both require that the file '`/dev/mem`' exist and be a character special device that accesses Linux kernel memory. dosemu must be installed suid root to read this file.

NOTE: some video cards may require additional configuration information to function correctly. For example, the ATI Wonder VGA card requires this line:

```
ports { 0x1ce 0x1cf }
```

6 Peripherals

6.1 Serial Ports

The emulator provides some support for both BIOS-accessed and direct hardware-accessed serial ports. I will primarily discuss the latter, as the former is rarely used.

The serial ports may be used to provide access to the system's mouse, modem, or other serial device. While the emulation of the UART[1] is as faithful as necessary, there are some features which are not yet emulated:

- Overrun, framing, and parity errors. Overrun should never happen. The other errors are difficult to detect from the termios interface.
- The modem control lines (DTR, RTS)
- Loopback is only partially emulated
- the 16550's FIFOs[2]

You may use any Linux character device special file as a serial port, although certain features will only work correctly on tty devices. dosemu only supports the simultaneous use of two serial ports.

The current implementation comfortably supports my 1200 baud serial mouse; however, it chokes on continuous 9600 baud data. The problem is an obvious–though not simple–one, but I doubt I'll rush to fix it unless people have mouse-related problems. Don't expect too much from the serial emulation; it's intended for mouse usage, not ka9q.

Serial Port Configuration

This section describes the run-time configuration options pertaining to dosemu's virtual serial ports. See Section 3.3 [Run-time Configuration], page 12, for an introduction to the configuration file.

The `serial` word begins a list statement which specifies the operation of the virtual serial ports. The general format is this:

`serial {` [*value arg*] ... [*predicate*] ... `}`

These are the serial words:

Device *devname* Value

Causes dosemu to treat the character device special file *devname* as a virtual serial port. The defaults are '`/dev/cua0`' for the first serial port and '`/dev/cua1`' for the second serial port.

---

[1] Universal Asynchronous Receiver/Transmitter, the support chip through which all serial transactions are made

[2] First In, First Out: a 16-byte queue the 16550 UART uses to provide more efficient character transfer by reducing the number of serial interrupts which must be serviced

Base *port* Value

Sets the virtual UART's base port address to *port*. The defaults are 0x3f8 for the first serial port and 0x2f8 for the second serial port.

Interrupt *intnum* Value

Sets the virtual UART's interrupt vector number to *intnum*. The defaults are 0xc for the first serial port and 0xb for the second serial port.

Modem Predicate
Mouse Predicate

These options specify the method of resource-sharing dosemu is to employ.

The mouse directive does not work in the 0.49 release. You will not be able to share the mouse with another program such as X11.

Modem causes dosemu to open the virtual serial port's device at "bootup" and keep it open until dosemu is shutdown.

Mouse causes dosemu to treat the virtual serial port as a shared resource which always belongs to the program running on the foreground virtual console. dosemu will open the virtual serial port's device when it's console is made current and release it when its console is switched away from. This only applies to console video, of course.

BUG: the *Mouse* directive does not work in dosemu 0.49. I hope to have this fixed fairly soon.

6.2 Printers

dosemu can emulate up to three printers accessed through interrupt 0x17 BIOS calls. These printers may only be accessed through the BIOS; dosemu makes no attempt to emulate a parallel port[3] Each virtual printer may be configured separately to act in one of three ways:

1. Save all output to a disk file
2. Send all output directly to some device
3. Send output in groups to some external program, such as the Linux `lpd` daemon.

Method 1 is naive, but ensures that print jobs are not split unnaturally. All of the session's printer output for that printer will be sent to the specified disk file.

Method 2 is not recommended, but can be used to control a directly connected printer without intermediate print handlers.

Method 3 is the most useful and sophisticated, but requires some user effort to configure. Under Linux, you will most probably use the default setup for LPT1:, with perhaps some additional options. The default timeout of 8 seconds may be too long or short for you; adjust it to best fit your application.

What happens on timeout:

If using methods 1 or 2, then the file will simply be flushed and closed until further output happens. Upon further output, the same file will be appended to. When dosemu is exited, the file

[3] I do not believe such a feature is necessary. If you have need of parallel port emulation, let me know and I'll see whta I can do.

will remain in the state it was before exiting: device files will remain device files, and normal files will remain normal files which just happen to contain all the accumulated printer output.

If using method 3, the temporary file that dosemu has chosen will be flushed and closed. The printer command will be invoked with the option string processed to replace the first occurrence of "%s" with the filename, after doing which dosemu will delete the temporary file.

Printer Configuration

The `printer` word begins a list statements which adds a virtual printer device dosemu's configuration. This is the general format of such a statement:

`printer {` [*predicate*] ... [*value arg*] ... `}`

These are the words which may be used within the printer list:

Command *cmd* Value
: tells the virtual printer which Linux command to call to print a file. The default is `lpr`.

Options *opt* Value
: tells the virtual printer the option string to pass to the print command. The string "%s" will be replaced with the filename (this is necessary!). The default is "%s".

Timeout *secs* Value
: sets the printer timeout to *secs* seconds. This is the amount of time the virtual printer device must sit idle[4] before sending the accumulated output to the print command as a file.

File *name* Value
Device *name* Value
: Will cancel the "wrapup and send job to lpr" type of virtual printer handling for the relevant printer, and configure dosemu to simply save all characters in the file named *name*, or send characters to the device named *name*. The timeout parameter is still used to determine when to flush the printer buffer to the file or device.

A sample section of a configuration file:

```
; this is the printer section of /etc/dosemu/config
;
printer { command "lpr" options "-p %s" timeout 8 }
printer { file "dosemulpt2" timeout 10 }
printer { device "/dev/lp1" timeout 3 }
```

Here, lpt1 is configured to send printer jobs to `lpr` with the pr-format option after a timeout of 8 seconds (method 1); lpt2 saves all output to the file `dosemulpt2` with a buffer flush every 10 seconds (method 1), and lpt3 flushes all output to the `/devlp1` device every 3 seconds (method 2).

NOTE: Make sure that there exsists a directory called '`/usr/tmp`' with the world-execute, world-read, and sticky bits set. The temporary-file calls in lpt.c need this directory to store files in. The best way to create this if you don't have it is this:

[4] i.e. receive no characters

```
mkdir /usr/tmp
chmod 1777 /usr/tmp
```

6.3 Disk Redirector

While most DOS users will already have an MS-DOS partition containing the "operating system" (sic), and assorted utility, application, and game packages, it is desirable for dosemu to allow DOS to access native Linux filesystems transparently. That is, dosemu has the ability to offer any Linux directory as a disk for MS-DOS to access as it would any other. This not only facilites file transfer between the two operating systems, but it helps wasted disk space: DOS and Linux can share a single file system, so that no space goes unused on either side. The operation of making a Linux file system available to DOS is called *redirection*, thus the program to do that is called a *redirector*. Users of PC-NFS, Novell Netware, or CD-ROM drives and MSCDEX will recognize this class of software.

Dosemu versions earlier than 0.49 used a program called `LINUX.EXE` to provide drive redirection. However, that program had several fatal faults, which are listed here:

- `LINUX.EXE` was based on a program called `PHANTOM.PAS`, which was published in Andrew Schulmann's *Undocumented DOS*. As dosemu is intended to be a freely available program, inclusion of copyrighted code is ... discouraged.
- `LINUX.EXE` suffered from several bugs, the most annoying of which was the inability to execute `.exe` files from a redirected file system.
- Because of the possibly copyrighted nature of `LINUX.EXE`, your humble author was for a long time unable to find the source to it, and was thus unable to fix even the most trivial of its bugs.

In order to use the disk redirector, make sure that your boot disk contains the file `emufs.sys`. Also make sure that the `config.sys` file on your boot disk has a line like this one:

`device=emufs.sys` *directory* `[`*flag*`]`

for every redirected drive you wish to be configured. *directory* is the Linux directory you wish to appear as a DOS network drive. emufs.sys automatically assigns the next available drive letter to it. *flag* is the optional letter `R` to specify that you want the drive to be read-only.

Now, you ask, what if I want to access my DOS disk this way instead of my Linux disk? I mean, what good are Linux files going to do me? Well, here's the trick: you can mount a DOS partition on a Linux directory using the Linux Virtual Filesystem magic (no need to concern yourself with the technical details). Thanks to Werner Almesberger's excellent MS-DOS filesystem in the kernel, you can use the normal Linux mount command to do this. For example, let's say you have two dos partitions, /dev/hda1 and /dev/hdb3. Do this on the Linux side:

```
mount -t msdos /dev/hda1 /dos1
mount -t msdos /dev/hdb3 /dos2
```

This mounts the DOS partition /dev/hda1 over the Linux directory /dos1, and the DOS partition /dev/hdb3 over the Linux directory /dos2. Now you can use the MFS disk redirector to access the DOS disks by telling it to use /dos1 as the root directory of the first DOS drive, and /dev/hdb3 as the root of the other. And, just for kicks, you want to allow read-only access to your Linux kernel sources from dosemu. Put these lines into your config.sys and reboot dosemu:

```
device=c:\emufs.sys /dos1
device=c:\emufs.sys /dos2
device=c:\emufs.sys /usr/src/linux R
```

Assuming you only have C: drive already defined, this will give you /dos1 as read-write drive D:, /dos2 as read-write drive E:, and /usr/src/linux as a read-only drive F:.

NOTE: You will still need some "image"-type drive to boot from. This drive will either be your virtual floppy drive A: or a virtual hard drive C:. See Section 4.1 [Booting], page 17, for more information on the types of drives from which dosemu can boot.

Andrew Tridgell (`tridge@nimbus.anu.edu.au`) "ported" the Mach DOS emulator's redirection code to dosemu. I translated his modified `linux.asm` into a format acceptable by Bruce Evans' as86 assembler; this is the file `emufs.S`. Any mistakes in emufs.S are probably mine. See Section 1.2 [Mach], page 2.

To generate `emufs.sys` from the source code, you'll need as86 and ld86. These should come compiled with the SLS distribution of Linux, and can also be found in source form in the file `bin86.tar.Z`, available on *tsx-11.mit.edu* and other fine Linux (and perhaps Minix) archives. You also need as86/ld86 to build the Linux kernel, so if you can build the kernel, you know you already have them.

Andrew also made these extremely useful enhancements and bugfixes to the redirector:

- the original port to dosemu
- sane mapping of DOS attributes to Linux attributes
- sane handling of uid/gid when run suid root
- integration of the two DOS programs that Mach used—(`machfs.sys` and `mfsini.exe`)—into one program for Linux (`emufs.sys`).
- the ability to handle multiple redirected drives
- the command line syntax of `config.sys` described above,
- the read-only flag for redirected drives

6.4 Hard Disks

My current setup in emu.c has two hard disks: hdimage (C:)[5], and direct access to my DOS partition (D:). It has two floppies, a 1.2 MB floppy (A:), and a 1.44 MB floppy (B:). I currently Look in the Makefile to specify your particular floppy configuration. See Chapter 3 [Configuration], page 11, See Section 6.5 [Disk Images], page 33, for further information about floppy configuration.

The `hdimage` and `diskimage` files must exist at dos invocation time if dosemu references them. Only one of them need be bootable; and they can be of any size. In fact, with this installation, you only need to type `touch diskimage`" in the dosemu directory to make it work. You may then format the A: disk to make "diskimage" into an accessible disk. The disk/hd-image will grow to accomodate your usage of it (but will not shrink!)

You can give dosemu direct access to your hard drive, without using a redirector, so that MS-DOS uses normal int13h calls to read and write sectors. For this option, of course, you must have

[5] The drive names in parentheses refer to the dos emulator's view of the world, not your normal, real-DOS letters.

at least one partition on that hard disk containing a MS-DOS filesystem. This isn't a problem, as the utility of this feature lies in being able to use the same disks for dosemu that you use for native DOS.

WARNING WARNING WARNING WARNING WARNING WARNING
You may *not* give dosemu direct access to an MS-DOS partition that is also mounted as a Linux filesystem! Very Bad Things will ensue! Linux will be confused by a filesystem that changes without Linux asking it to change, and MS-DOS will probably be too stupid to notice—thereby corrupting whatever you doomed by trying such a silly thing.

It is for this reason I recommend using the Disk Redirector if at all possible. This *will* let dosemu share access to a Linux-mounted MS-DOS partition.
WARNING WARNING WARNING WARNING WARNING WARNING

If you have an entire hard drive dedicated to MS-DOS, look into Whole Disk Access; however, if your hard disk shares both DOS and Linux partitions, as mine does, Single Partition Access might be better suited to your system.

Better yet, scrap the whole direct access thing altogether and use the MFS redirector to access your DOS hard disk through Linux. See Section 6.3 [Disk Redirector], page 30, for an explanation.

6.4.1 Whole Disk Access

Whole Disk Access allows dosemu to directly read an MS-DOS filesystem from a *disk* block device (*not* a partition device!). This means that, if you have a DOS partition somewhere on your first hard drive, you could set up dosemu to find that partition on `/dev/hda`. This feature is deprecated and pretty much undocumented.

Whole Disk Access gives dosemu access to, well, the whole disk. This includes the Master Boot Record and every partition on the disk. You may boot from a disk configured with WDA, but I don't recommend it. In particular, LILO boots on my system if I try this; it will not, of course, boot Linux under dosemu.

dosemu is configured by default to attempt auto-detection of your hard disk's geometry by querying the Linux kernel. This may or may not work for non-IDE drives. If it does not work correctly, you will have to enter the geometry explicitly in the configuration file.

See Section 3.3 [Run-time Configuration], page 12, for more information on configuring your hard disk. Here's an example config line for my hard disk setup:

```
disk { wholedisk /dev/hda1 }
```

6.4.2 Single Partition Access

Single Partition Access (hereafter abbreviated to SPA, which may or may not be a registered trademark of Robert Sanders, Megaconglomerated) is very much like Whole Disk Access. dosemu used a block disk device to access the disk on a sector level and DOS sees the partition as a normal int13-accessible drive (i.e. non-network).

However, the difference lies in the fact that dosemu uses a different block disk device; Whole Disk Access needs the the Whole Disk devices like `/dev/hda`, while SPA access the Partition devices like `/dev/hda1`. The main difference between these two is that `/dev/hda` can access any part of the disk, while `/dev/hda1` can only access the area of the disk allocated to its specified partition.

Using SPA access is much safer than using WDA if you're worried about dosemu going ape all over your important data. I have my Linux and DOS partitions on the same drive, so this feature is a big win for me. Your mileage may vary.

Single Partition Access gives access to what DOS calls *primary* partitions only. You may not enable access to logical partition within an *extended* partition. If there is enough need for this feature I may add it. You may boot from an SPA partition.

To set up dosemu for SPA access, follow this procedure:

1. Run the included program mkpartition to read the partition data from your hard drive. mkpartition takes two arguments, both mandatory: the whole disk drive device, and the partition number within that drive. For example:

   ```
   mkpartition /dev/hda 1
   ```

   sets up the `/etc/dosemu/partition` configuration file for SP access to the first partition of the drive (`/dev/hda1`).
2. Add a line to the '`/etc/dosemu/config`' configuration file like this:

   ```
   disk { partition /dev/hda1 1 }
   ```

See Chapter 3 [Configuration], page 11, for more information on the files in /etc/dosemu.

6.5 Disk Images

Everyone seems to ask this: "How do I create diskimage/hdimage"? Well, it isn't really that complicated.

Image files of both types must reside in the directory from which you start dosemu, at least with the standard installation.

6.5.1 Creating hdimage

Follow these steps to create an image hard disk for Linux dosemu. This hard disk will act like a native DOS formatted hard disk drive.

1. First, you must create the actual `hdimage` file. dosemu has a special format for hard disk image files: each one has a 128 byte header which tells dosemu the disk's format. dosemu is supplied with a program to create an `hdimage` of any size.

   Invoke `mkhdimage` with the proper parameters to create the size image file you desire. Currently the `mkhdimage` program only accepts head, sector, and cylinder counts; all arguments are optional, and `mkhdimage`, and the default is to create a 1 megabyte drive. Make sure you redirect the standard output of `mkhdimage` into the `hdimage` file; otherwise you will see gibberish.

   This command generates a hard disk image file with a geometry corresponding to that of a real hard disk with 12 heads, 17 sectors per track, and 300 cylinders.

   ```
   mkhdimage -h 12 -s 17 -c 300 > hdimage
   ```

   At 512 bytes per sector, this equals a roughly 29 megabyte drive. This is probably too large for most needs; if you need this much space, consider using the disk redirector. See Section 6.3 [Disk Redirector], page 30).

2. Once the hdimage file is created, you must run DOS fdisk to create a partition. Then you exit the emulator, reenter it, and format the C: drive. You can make the disk bootable with the /s parameter to format, a shown in the example below:

```
format c: /s
fdisk /mbr
```

The second line is necessary because of the custom boot sector on the current distribution hdimage. I will probably fix this before the real release of 0.49.

6.5.2 Creating diskimage

There are two ways to create an image of a floppy (`diskimage`) file for the Linux DOS emulator.

1. To create an image file whose contents are a copy of an already existing, real floppy drive, simply copy the entire disk into a Linux file. For example, this command copies the disk from /dev/fd0 into the file `diskimage`:

```
dd if=/dev/fd0 of=diskimage bs=16k
```

Make sure that the configuration of dosemu properly understands what type of floppy the file was copied from. Once that is done, this diskimage is ready to use. See Chapter 3 [Configuration], page 11

If your original disk is bootable, so will `diskimage` be; this is an excellent way to create a boot disk for dosemu. (To boot from this disk, invoke dosemu with the `-A` switch, which tells dosemu to boot from the `A:` drive.) See Section 4.1 [Booting], page 17

2. To create a blank floppy image file which you will then format under dosemu, simply execute this command:

```
touch diskimage
```

This creates a 0-length file called `diskimage` in the current directory. To use this file, you must enter dosemu and format it for use under MS-DOS. This command will format it and make it bootable:

```
format a: /s
```

6.6 Disk Configuration

The `disk` and `floppy` words begin list statements which add a disk or floppy to dosemu's configuration. This is the general format of such a statement:

`disk` `{` [*value arg*] ... [*predicate*] ... `}`
`floppy` `{` [*value arg*] ... [*predicate*] ... `}`

Some words within these statements are only useful for hard disk configuration:

image *file* Value

Specifies that the disk is to be configured as an hdimage type file called *file*.

partition *partdev partnum* Value

Specifies that the disk is to be configured as a directly accessed partition, which Linux accesses as device special file *partdev*; the cardinal partition number *partnum*, which must be between 1 and 4, is the partition's number in the partition table.
See Section 6.4.2 [Single Partition Access], page 32

wholedisk *diskdev* Value

Specifies that the disk is to be configured as a diretly accessed disk, which Linux refers to as device special file *diskdev*.
See Section 6.4.1 [Whole Disk Access], page 32.

Some words are only useful for floppy disk configuration:

threeinch Predicate
fiveinch Predicate

Specifies whether the floppy drive should show up in the CMOS configuration as a five inch (1.2 MB) drive, or a three inch (1.44 MB) drive.

Device *dev* Value

Specifies the device *dev* as the floppy disk. *dev* may be either a block special device which points to a real floppy disk, or a floppy disk image. See Section 6.5.2 [Creating diskimage], page 34.

These words apply to both floppy and hard disks:

Sectors *count* Value

Configures the specified disk to have *count* sectors.

Heads *count* Value

Configures the specified disk to have *count* heads.

Cylinders *count* Value

Configures the specified disk to have *count* cylinders.

readonly Predicate

Sets the relevant disk to read-only mode. This still behaves strangely. See Chapter 10 [Bugs], page 43, for more information.

NOTE: it should be unnecessary to specify the geometry of a disk for real hard disks, partitions on such disks, real floppy drives, or new-style hdimage files. In other words, only floppy image files should need a geometry specification.

7 Memory

dosemu supports several kludgy methods that pseudo-allow MS-DOS to "break the 640K barrier," as the adverts like to say.

7.1 DOS memory

If you need extra memory, you can use the "-m" switch to set the memory size. This switch specifies, in Kilobytes, how much main memory the emulator has available. The default is 640, although the emulator allows up to 734. I believe that 734K is safe, although too much over that intrudes on the screen memory at b800:0000.

This option is incompatible with VGA video, so specifying VGA through either of the option methods will set the base memory size to 640K. If you need additional DOS memory, look into XMS and EMS. See Section 7.2 [XMS], page 37, see Section 7.3 [EMS], page 37, for more information on increasing the amount of free memory.

7.2 XMS

Adam Goldberg (`adamg@microware.com`) was kind enough to send me the XMS 3.0 specs so I have, to the best of my knowledge, implemented the entire XMS 3.0 specification. This includes:

- Extended Memory Block (EMB) Functions
- Upper Memory Block (UMB) Functions
- High Memory Area (HMA) Functions
- 32-bit Extended Memory Block Functions (new with XMS 3.0)

You can specify the maximum amount of memory the emulator can use in two ways: through the `-x` *size* command line option, and through the run-time configuration file (see Section 7.4 [Memory Configuration], page 38). *size* is given in kilobytes.

If you enable XMS, you can put the line

```
dos=high,umb
```

in your `config.sys` to allow MS-DOS itself to use the extra memory. Using XMS along with this line, I have well over 600K of memory left for DOS executables; with both SMARTDRV and RAMDRIVE loaded high I have 621K of memory left for DOS executables.

7.3 EMS

A partial EMS implementation is also present; I say "partial" because it implements version 3.2 of the Expanded Memory Specification, which is extremely old. The newer version, released in 1987, is EMS 4.0. I am currently working on extending the dosemu implementation to EMS 4.0.

The dosemu EMS implementation was based on the Mach DOS emulator's EMS implementation, which provided version 3.2 of the Lotus/Intel/Microsoft Expanded Memory Specification. [Ed: oh, the bletchery!] See Section 1.2 [Mach], page 2, for more info about Mach.

You can specify the maximum amount of memory the emulator can use in two ways: through the `-x` *size* command line option, and through the run-time configuration file (see Section 7.4 [Memory Configuration], page 38). *size* is given in kilobytes.

EMS requires a kernel patch, included with dosemu. It adds an `mmap()` operation for the device /proc/self/mem, which dosemu uses to manipulate regions of its address space.

7.4 Memory Configuration

These are the configuration words which pertain to dosemu's memory management. See Section 3.3 [Run-time Configuration], page 12, for an introduction to the configuration file.

DOSmem *size* Value

Sets the base DOS memory size to *size* kilobytes. Default size is 640k. The maximum is 734K.

EMS *size* Value

Activates Expanded Memory Specification 4.0 emulation with a memory pool size of *size* kilobytes.

XMS *size* Value

Activates Extended Memory Specification 3.0 emulation with a memory pool size of *size* kilobytes.

If you wish to explicitly disable XMS or EMS, you may specify its control word and the parameter *none*. The default for both is to be disabled.

8 Miscellaneous

These are some marginally useful or outright useless options that I haven't gotten around to removing from dosemu.

8.1 BIOS

The "-b" option maps the video BIOS ROM at c000:0000 into the emulator's address space.[1]

Do not use this option unless necessary, as the XMS Upper Memory Block support uses the address space normally occupied by the ROMs for Upper Memory Blocks; mapping in the BIOS reduces the amount of memory available for XMS. See Section 7.2 [XMS], page 37, for more information about Upper Memory Blocks.

NOTE: You may suffer ROM corruption problems if your computer uses shadow ram to speed BIOS access times. If this happens, turn off ROM shadowing in your system's CMOS setup[2].

Alternately, you may use the *vbios_copy* and *vbios_file* video directives. See Section 5.4 [Video Configuration], page 25, for more information on ways to prevent BIOS corruption.

8.2 CPU type

Many DOS programs act differently when run upon different Intel processors; for example, a program might use the 32-bit registers available on the 386, while on the 286 it makes do with 16-bit registers.

The canonical method of differentiating between the different processor types is explained in the *Intel 486 Programmer's Handbook*. Suffice it to say that the test relies upon various instruction peculiarities and the behavior of bits in the `eflags` register. While dosemu cannot emulate the former, the latter is easy to emulate using the techniques available through virtual-8086 mode.

These are the dosemu command-line switches to choose the processor mode:

`-2` act as an 80286

`-3` act as an 80386 (default)

`-4` act as an 80486

The run-time configuration file may also be used to select the processor to be emulated. See Section 3.3 [Run-time Configuration], page 12.

Because of a combination of circumstances, it is possible to make a 386 show up as a 386; it is possible to make a 486 show up as a 386. It is possible to make a 486 test as a 486, but I have not tested whether it is possible to make a 386 test as a 486 (although it should work, it is not recommended). Most programs are naive enough to believe that my 486 is a 286 when I select that option, but Turbo Debugger, for example, notices the 32-bit registers and is not fooled.

---

[1] This option is automatically activated by using BIOS-driven video.

[2] Linux can run into other problems if you have shadow ROM enabled, so it's always a good idea to turn it off

8.3 TTY copy

The "-P TTY" sends a copy of _dbg_ (the debugging messages) to a terminal. It can be left out if you wish (or have nothing to send it to!). This is no big deal, basically just a tee. The reason it's there at all is that it will one day provide me with an interactive debugging system. (One day, like everything else :-).

9 Programs That Work

Try to save the debugging messages for any bugs or problems that you find. They are more helpful to me than descriptions alone. See Section 4.3 [Debugging Messages], page 18.

DR-DOS 6.0 apparently works under the emulator. I have been told that, because it uses a variety of temporary files, your boot disk (whatever it may be) should be un-write-protected. If you have any problems, write me.

People have had some success using Stacker or SuperStor to access their compressed "partitions" under the emulator. I don't have either of these programs, haven't tried them, and most definitely won't guarantee that they'll work. However, you are free to experiment.

Here are some programs that I use/test under dosemu:

- Turbo Pascal 5.5
- PC Tools PCShell (and compress)
- QEdit
- Borland TASM/TLINK
- doskey (might not be impressive, but it's useful)
- 4DOS 4.0
- HelpPC 2.1
- a86 and d86 (a shareware assembler and debugger)
- WordPerfect 5.0
- Microsoft Word 5.5
- Norton Utilities v6.0[1]
- Checkit
- Turbo Debugger 3.0
- A very old version of Turbo C
- pkzip 2.04e
- MKS utilities
- Elvis 1.5
- SimEarth
- ColorView/386
- Telix & Kermit (at low baud rates)
- MouseSystems mouse and MicroSoft mouse driver (mscmouse.com)
- dBase III+

Here are some programs that are *reported* to work:

- DR-DOS 6.0
- SuperStor

---

[1] Go into configuration and turn off "Zooming Boxes" under the Video menu. This really speeds things up.

- Stacker
- uEmacs (don't even think about demacs!)
- Minitab
- the Big Mouth Voicemail card[2]
- Managing Your Money,
- Quicken 4.0
- FoxPro 2.0
- Microsoft Word 5.0
- Turbo C++ 1.0
- Alpha-4 2.1 (database)
- Microsoft Multiplan 4.2
- Lotus 123 v2.2

Also, see section Section 5.3 [VGA], page 23 for a listing of programs that work using graphics and extended text modes. Once the new VGA emulation is more thoroughly integrated into dosemu I will incorporate that list of programs here.

[2] Ed Carp, erc@apple.com posted some patches to get it to work

10 Bugs

I probably already know about the **lack of features** in dosemu, so please don't tell me "It won't recognize my mouse." However, if you do find some failure that you think ought to work, *please* do write me and tell me about it. Also, if you have some fairly simple feature that you really need, I can be flexible.

If you need to reach me, I can receive e-mail at this Internet address:

```
gt8134b@prism.gatech.edu
```

I'm sorry about the gobbledygook address, but my lovely educational institute doesn't see fit to give human names to non-computer science majors.

Also, I frequent the MSDOS mailing list; I answer questions posted there, and post dosemu updates and announcements there. To find out more about the mailing lists, send mail to `linux-activists-request@niksula.hut.fi` with a message body of `HELP`.

My implementation of EMS 4.0 (based on the Mach implemetnation of EMS 3.2) seems to have a few bugs; they have only manifested as errors when using PKZIP in conjunction with RAMDRIVE.SYS. If you have the choice, use XMS instead of EMS; not only is it (paradoxically) faster to use XMS, it's safer (easier implementation and it's been tested longer). Also, many EMS 4.0 functions aren't implemented, which could cause problems.

The VGA save/restore code for console switching while in SVGA modes only handles Trident and ET4000 chipsets, and the ET4000 portion of it is slightly buggy. In addition, the Trident section fails to correctly restore the screeon on at least the Trident 8800.

Non-console video still suffers from a bug that causes it to incorrectly update areas of the screen. This will hopefully be fixed soon.

As mentioned before, the *Mouse* and *MDA* configuration file directives—and their corresponding features—do not work.

I've fixed too many dosemu0.4 bugs to count, so I won't list them here. I've probably introduced as many, though, so watch out.

I occasionally hear of some program that used to work, but doesn't now. Please, mail me with the name of the program, what it does (and how it does it, if you have any idea), and the behavior before and now. I'll try to get it working again, but no promises.

If the emulator crashes on you, the first thing you should do is examine the dbg file for any line beginning with `ERROR:`; look at the last such line first. If you can't find any suspicious lines, try recreating the crash with full debugging-messages (i.e. specify `-Da` as a parameter to dosemu). You'll get a file full of trash, probably, but at least it's something.

Sending me reports like "XX doesn't work" really does me no good. What would be useful is a dosdbg file in which you have noted under what conditions and actions the bug occurred. For example comments like:

```
/* selected "Turbo Mode" here, screen turned blue */
```

at the proper points in the dosdbg file are necessary for me to make any sense of it.

11 Changes

This is a list of the changes from dosemu 0.48pl1 to 0.49. If you are currently running a version of dosemu 0.48pl1 or older, you need to make some changes to your configuration to make sure that 0.49 will work. See Section 2.2 [Updating from 0.48pl1], page 9, for more information on upgrading your current dosemu installation.

- Extended XMS 3.0 functions (the 32-bit functions). XMS is now almost done, including UMB functions. They aren't perfect, but they work OK. See Chapter 7 [Memory], page 37.
- fixed a nasty bug that sometimes occurred when a key was held and repeated
- added Bill Bogstad's auto-hd-geometry-detection patch. (I would probably have never thought of this, as dosemu always has my geometry correct, straight out of the box :-)
- early, buggy mouse support. Except that it really isn't even a mouse yet, just a keyboard mouse. And that doesn't even work. So don't use it.
- added CPU selection options, -2, -3, and -4. These make the flags register act like a 286, 386, or 486 (this isn't infallible; Turbo Debugger 3.0, for example, won't believe you have a 286 no matter what flags you try).
- switched to 2-process model (not really user-visible, except key repeat works MUCH better), and using SYSV IPC came with it. You'll need Krishna Balasubramanian's ipcdelta package (ipcbeta won't do it!) compiled into a 0.99pl7 or newer kernel.
- using IPC shared memory, I fully and properly implemented the High Memory Area.
- fixed the STI instruction emulation to work as advertised by Intel. However, this slows things down some, so I have disabled it by default. #define PROPER_STI in cpu.c to enable it. Until the DOS emulator emulates hardware by generating pseudo-interrupts, this option is just useless chaff.
- fixed the emulator's handling of the IF bit in general.
- MS-DOS 5.0 no longer exhibits the booting problems it did, so I have removed the boot in progress checks.
- fixed DOS STDIN redirection (i.e. debug < file). The problem here was that dosemu assumed the int21h get-character calls read from the keyboard.
- dosemu now (again) properly supports multiple screen pages, so things like Turbo Debugger 3.0 (and previous) now work
- revamped the Makefile. you need to do a "make config" after you edit the settings in the Makefile. In all honesty, the Makefile (unlike dosemu proper) is a bit too feature-laden, but I was having fun... See Chapter 3 [Configuration], page 11.
- you're all gonna love this one: Andrew Tridgell (tridge@nimbus.anu.edu.au) ported the Mach DOS emulator's "network" redirector drive code, so now we have a mostly bug-free alternative to LINUX.EXE (and no uncertainty about the copyright, either). I would have gotten around to it. Really. He's added multiple-disk and read-only support. See Section 6.3 [Disk Redirector], page 30
- changed the format of hdimage. It now has a header (currently 128 bytes) that allows dosemu to use hdimage files with different geometries without being recompiled. See Section 6.5 [Disk Images], page 33. See Section 2.2 [Updating from 0.48pl1], page 9.
- changed the direct-DOS-disk support to also use partitions instead of just disks. This means that you can tell it to use /dev/hda1 instead of /dev/hda. Aside from the added safety of

protecting your Linux partitions from a possibly insane dosemu running wild on the entire disk, this allows you to make your DOS partition accessible to a group dos, and make all dosemu users members of group dos, etc. UNIX permissions tricks like that. I run as root, so don't ask me :-) See Section 6.4 [Hard Disks], page 31.

- EMS 3.2 emulation with a kernel patch. I might have upgraded this to EMS 4.0 by the time you read this, or I might not have. The EMS 3.2 "engine" was taken from Mach's DOS emulator and bugfixed/modified a great deal for dosemu. See Chapter 7 [Memory], page 37.
- fixed a configuration problem (a math coprocessor was being flagged when there was none). By the way, there is a bug in kernel FPU-emu in at least 0.99pl7 and previous, so you'll have problems if you don't have a 387/486. The problem is fixed in kernel 0.99pl8, thanks to two people who independently found and reported the problem to me. Note that dosemu will not work with 0.99pl8 unless you apply a special patch (the sigdiff patch mentioned before; See Chapter 3 [Configuration], page 11, for more information about kernel versions and dosemu).
- more properly handles a monochrome video card with MDA_VIDEO
- Andrew Tridgell's int16() TAME-like anti-keyboard hogging pause. Untweaked, but very effective as is on my 486. 70% CPU -> 2.1% CPU for one program. Numbers will probably be different with the new high-resolution Linux timer (when usleep(100) actually means something).
- new printer handler can now send jobs to `lpr` (or whatever other command you like).
- the cheap VGA BIOS demo thing. See Section 5.3 [VGA], page 23, for details. Might also work for MDA/CGA/EGA, if the card's BIOS shows up at 0xc000:0000.
- the -e and -x (EMS and XMS) switches now take a numeric argument that specifies (in kilobytes) how much memory to allow for each. I will be changing this to use a unified memory pool...
- added the run-time configuration file parser
- fixed a problem with the int10h cursor emulation; this most often manifested in Borland/Turbo-compiled programs.
- Serial port emulation. This allows one to use modems and mice from dosemu.
- Fixes to the int10h code. Much better handling of int10h ah=9/Ah.
- a lot of little things here and there.

Concept Index

Program/File Index

Table of Contents