DANG The DOSEMU Alterer Novices Guide

Alistair MacDonald, alistair@slitesys.demon.co.uk For DOSEMU v0.63.1

This Document is the DOSEMU Alterer Novices Guide. It is known as the DANG.

Contents

1 Introduction

This document is the preliminary draft of a manual to help people understand the inner workings of dosemu. It is the goal of this document to create new dosemu hackers. This concept was inspired by the linux kernel hackers guide.

This Guide was concieved and originally written by "Corey Sweeney" <corey@>. It has been completely revised. It is now generated automatically directly from the source code. Special thanks to "James B. MacLean" <macleajb@ednet.ns.ca> for supplying the original information. (It was mostly ripped out of a mail message.) "Jochen Hein" has made many useful comments & suggestions.

At the end if this document is a section detailing how this guide is put together. This may help you when trying to locate the relevant pieces of code. If you add new code, it would be useful if the relevant markers are added where appropriate.

This file is a collective effort. If you don't like one of the explanations, or want to add anything, please send me something!

2 The Main group of Modules

These files are used to start DOSEMU as well as hold globally called functions and global vars.

2.1 dos.c Information

2.1.1 Functions in dos.c

These are the functions defined in dos.c.

dosemu Arguments are:

- argc - Count of argumnents.
- argc - Actual arguments.

Function created by entry point into libdosemu. Called to jump into the emulate function of DOSEMU.

2.2 emu.c Information

2.2.1 Functions in emu.c

These are the functions defined in emu.c.

jmp_emulate call the emulate function by way of the dll headers. Always make sure that this line is the first of emu.c and link emu.o as the first object file to the lib

SIG_int Requires the sig/sillyint.o driver loaded (using NEW modules package), or a kernel patch (implementing sig/int.c driver). The IRQ numbers to monitor are taken from config.sillyint, each bit corresponding to one IRQ. The higher 16 bit are defining the use of SIGIO

emulate Arguments are:

- argc - Argument count.
- argv - Arguments.

Emulate gets called from dos.c. It initializes DOSEMU to prepare it for running in vm86 mode. This involves catching signals, preparing memory, calling all the initialization functions for the I/O subsystems (video/serial/etc...), getting the boot sector instructions and calling vm86().

2.2.2 Remarks in emu.c

DOSEMU must not work within the 1 meg DOS limit, so start of code is loaded at a higher address, at some time this could conflict with other shared libs. If DOSEMU is compiled statically (without shared libs), and org instruction is used to provide the jump above 1 meg.

—–

At this time we have to use SIGALRM in addition to SIGIO I don't (yet) know why the SIGIO signal gets lost sometimes (once per minute or longer). But if it happens, we can retrigger this way over SIGALRM. Normally SIGIO happens before SIGALARM, so nothing hurts. (Hans)

2.3 include/emu.h Information

2.3.1 Functions in include/emu.h

These are the functions defined in include/emu.h.

NEWSETQSIG Arguments are:

- sig - the signal to have a handler installed to.
- fun - the signal handler function to install

All signals that wish to be handled properly in context with the execution of vm86() mode, and signals that wish to use non-reentrant functions should add themselves to the SIGNALS_THAT_QUEUE define and use SETQSIG(). To that end they will also need to be set up in an order such as SIGIO.

2.3.2 Remarks in include/emu.h

The 'vm86_struct' is used to pass all the necessary status/registers to DOSEMU when running in vm86 mode.

—–

We assume system call restarting... under linux 0.99pl8 and earlier, this was the default. SA_RESTART was defined in 0.99pl8 to explicitly request restarting (and thus does nothing). However, if this ever changes, I want to be safe

—–

DOSEMU keeps system wide configuration status in a structure called config.

—–

The var 'fatalerr' can be given a true value at any time to have DOSEMU exit on the next return from vm86 mode.

3 The DPMI group of Modules

DPMI is Lutz's Baby. It's a really important part of the Emulator as far as we are concerned, since it will allow us to run so many more programs and, most importantly, bcc. This is the one thing that the WINE developers want that we haven't been able to give them.

If you think you can help .... "Away you Go!" (Sorry to those non-UK folks ... Thats a reference to a UK kids sports programme from my youth ... anyway ... enough of this banter. You'll be wanting to know that this is all about DPMI ...)

3.1 dosext/dpmi/dpmi.c Information

3.1.1 Functions in dosext/dpmi/dpmi.c

These are the functions defined in dosext/dpmi/dpmi.c.

dpmi_control This function is similar to the vm86() syscall in the kernel and switches to dpmi code.

run_pm_int This routine is used for running protected mode hardware interrupts and software interrupts 0x1c, 0x23 and 0x24. run_pm_int() switches to the locked protected mode stack and calls the handler. If no handler is installed the real mode interrupt routine is called.

do_default_cpu_exception This is the default CPU exception handler. Exceptions 0, 1, 2, 3, 4, 5 and 7 are reflected to real mode. All other exceptions are terminating the client (and may be dosemu too :-)).

do_cpu_exception This routine switches to the locked protected mode stack, disables interrupts and calls the DPMI client exception handler. If no handler is installed the default handler is called.

dpmi_fault This is the brain of DPMI. All CPU exceptions are first reflected (from the signal handlers) to this code. Exception from nonpriveleged instructions INT XX, STI, CLI, HLT and from WINDOWS 3.1 are handled here. All here unhandled exceptions are reflected to do_cpu_exception()

3.1.2 Remarks in dosext/dpmi/dpmi.c

We are caching ldt here for speed reasons and for Windows 3.1. I would love to have an readonly ldt-alias (located in the first 16MByte for use with 16-Bit descriptors (WIN-LDT)). This is on my wish list for the kernel hackers (Linus mainly) :-))))))).

DPMI is designed such that the stack change needs a task switch. We are doing it via an SIGSEGV - instead of one task switch we have now four :-(. Arrgh this is the point where I should start to include DPMI stuff in the kernel, but then we could include the rest of dosemu too. Would Linus love this? I don't :-((((. Anyway I would love to see first a working DPMI port, maybe we will later (with version 0.9 or similar :-)) start with it to get a really fast dos emulator...............

Handling of the virtuell interruptflag is still not correct and there are many open questions since DPMI specifications are unclear in this point. An example: If IF=1 in protected mode and real mode code is called which is disabling interrupts via cli and returning to protected mode, is IF then still one or zero? I guess I have to think a lot about this and to write a small dpmi client running under a commercial dpmi server :-).

3.1.3 Items for Fixing in dosext/dpmi/dpmi.c

We shouldn't return to dosemu code if IF=0, but it helps - WHY?

---

we should not change registers for hardware interrupts

3.1.4 New Ideas for dosext/dpmi/dpmi.c

Simulate Local Descriptor Table for MS-Windows 3.1 must be read only, so if krnl386.exe/krnl286.exe try to write to this table, we will bomb into sigsegv() and and emulate direct ldt access

4 The Video group of Modules

All of the Video handling code is in the "video" subdirectory.

There is one file for each video card or chipset and the master file. To Add a new card, it needs a set of save & restore routines putting in a file here.

4.1 env/video/vc.c Information

4.2 env/video/video.c Information

4.2.1 Functions in env/video/video.c

These are the functions defined in env/video/video.c.

video_init Set pointer to correct structure of functions to initialize, close, etc... video routines.

4.2.2 Remarks in env/video/video.c

Here the sleeping lion will be awoken and eat much of CPU time !!!

The result of setting VM86_SCREEN_BITMAP (at state of Linux 1.1.56): Each vm86 call will set 32 pages of video mem RD-only (there may be 1000000 per second) Write access to RD-only page results in page-fault (mm/memory.c), which will set a bit in current->screen_bitmap and calls do_wp_page() which does __get_free_page(GFP_KERNEL) but frees it immediatly, because copy-on-write is not neccessary and sets RD/WR for the page. (this could happen 32000000 per second, if the CPU were fast enough) It would be better to get the DIRTY-bit directly from the page table, isn't it? A special syscall in emumodule could do this.

—–

reserve_video_memory()

This procedure is trying to eke out all the UMB blocks possible to maximize your memory under DOSEMU. If you know about dual monitor setups, you can contribute by putting in the correct graphics page address values.

4.3 env/video/X.c Information

4.3.1 Functions in env/video/X.c

These are the functions defined in env/video/X.c.

get_vga256_colors Allocates a colormap for 256 color modes and initializes it.

X_close Destroys the window, unloads font, pixmap and colormap.

X_setmode Resizes the window, also the graphical sizes/video modes. remember the dos videomodi

X_change_mouse_cursor This function seems to be called each screen_update :(It is called in base/mouse/mouse.c:mouse_cursor(int) a lot for show and hide.

X_redraw_screen Redraws the entire screen, also in graphics mode Used for expose events etc. returns: nothing

Arguments are:

- none

X_update_screen Updates, also in graphics mode Graphics in X has to be smarter and improved returns: 0 - nothing updated 2 - partly updated 1 - whole update

Arguments are:

- none

set_mouse_position places the mouse on the right position Not tested in X with graphics returns: nothing

Arguments are:

- x,y - coordinates

4.4 env/video/console.c Information

4.5 env/video/dualmon.c Information

4.5.1 Functions in env/video/dualmon.c

These are the functions defined in env/video/dualmon.c.

MDA_init Initializes the monochrome card. First detects which monochrome card is used, because the Hercules RamFont and the Hercules InColor need one more register to be initialized. If there is no monochrome card at all, we just think there is one and poke an peek in the void. After the detection the card is initialized. returns: nothing

Arguments are:

- none

4.5.2 Remarks in env/video/dualmon.c

After MDA_init() the VGA is configured, something in video.c or console.c "reprograms" the monochrome card again in such a way that I always have to run hgc.com before I can use any program that uses the monochrome card. I've spent a day trying to find it, but I can't figure out. Something is writing to one of the following ports: 0x3b4, 0x3b5, 0x3b8, 0x3b9, 0x3ba, 0x3bb, 0x3bf. The problem occurs at (at least) the following 2 systems:

- AMD 386DX40, Trident 9000/512Kb ISA, Hercules Graphics Card Plus - Intel 486DX2/66, Cirrus Logic 5426/1Mb VLB, Hercules clone

The problem doesn't occur when I start dosemu from a telnet connection or from a VT100 terminal. (Erik Mouw, jakmouw@et.tudelft.nl)

4.6 env/video/et4000.c Information

4.7 env/video/hgc.c Information

4.8 base/bios/int10.c Information

4.9 env/video/s3.c Information

4.10 env/video/terminal.c Information

4.11 env/video/trident.c Information

4.12 env/video/vga.c Information

5 The Keyboard group of Modules

All of the Keyboard handling code is in the "keyboard" subdirectory.

Latest addition is SLANG.

5.1 base/keyboard/Xkeyb.c Information

5.2 base/keyboard/keymaps.c Information

5.2.1 Remarks in base/keyboard/keymaps.c

The DEAD codes must refer to keys that don't exist on any language keyboard. I hope nobody has a smily face key :-) dead_key_table is a list of the dead keys supported. They must be placed on the correct key in the keymaps above. See key_map_es_latin1.

—–

dos850_dead_map consists of the triple, {deadkey, letter, result}. It should be correct for all the code page 850 users (Western Europe). If you uses a different code page, please create a map! Jon Tombs jon@gtex02.us.es

5.3 base/keyboard/slang-termio.c Information

6 The Misc group of Modules

These are the remaining important files, that do not really fit into another group. These should not be dismissed as unimportant - rather, they are often amongst the most important.

6.1 dosext/misc/emm.c Information

6.2 dosext/misc/xms.c Information

6.3 arch/linux/async/sigsegv.c Information

6.3.1 Functions in arch/linux/async/sigsegv.c

These are the functions defined in arch/linux/async/sigsegv.c.

dosemu_fault All CPU exceptions (except 13=general_protection from V86 mode, which is directly scaned by the kernel) are handled here.

6.4 include/int.h Information

6.5 include/ports.h Information

6.6 base/misc/dosio.c Information

6.7 base/misc/disks.c Information

6.7.1 Functions in base/misc/disks.c

These are the functions defined in base/misc/disks.c.

disk_init Test by opening all floppies/hardrives configured.

6.8 emi-i386/cpu.c Information

6.9 dev/misc/lpt.c Information

7 The Serial group of Modules

This is the code that works our serial emulation. This needs to be very fast if we are to convince DOS that we have a very fast serial port.

7.1 base/serial/ser_defs.h Information

7.1.1 Remarks in base/serial/ser_defs.h

Extensions to serial debugging.

SER_DEBUG_MAIN (0 or 1) - extra debug output on the most critical information.

SER_DEBUG_HEAVY (0 or 1) - super-heavy extra debug output, including all ports reads and writes, and every character received and transmitted!

SER_DEBUG_INTERRUPT (0 or 1) - additional debug output related to serial interrupt code, including flagging serial interrupts, or PIC-driven code.

SER_DEBUG_FOSSIL_RW (0 or 1) - heavy FOSSIL debug output, including all reads and writes.

SER_DEBUG_FOSSIL_STATUS (0 or 1) - super-heavy FOSSIL debug output, including all status checks.

You must recompile dosemu everytime one of these constants are modified. Just type 'make' in the dosemu dir and it will recompile the changes only.

—-

IMPORTANT INFO about com[] variable array structure used in serial.c

Most of the serial variables are stored in the com[] array. The com[] array is a structure in itself. Take a look at the 'serial_t' struct declaration in the serial.h module for more info about this. Only the most commonly referenced global variables are listed here:

config.num_ser Number of serial ports active. com[x].base_port The base port address of emulated serial port. com[x].real_comport The COM port number. com[x].interrupt The PIC interrupt level (based on IRQ number) com[x].mouse Flag mouse (to enable extended features) com[x].fd File descriptor for port device com[x].dev[] Filename of port port device com[x].dev_locked Flag whether device has been locked

The arbritary example variable 'x' in com[x] can have a minimum value of 0 and a maximum value of (config.numser - 1). There can be no gaps for the value 'x', even though gaps between actual COM ports are permitted. It is strongly noted that the 'x' does not equal the COM port number. This example code illustrates the fact, and how the com[] array works:

for (i = 0; i < config.numser; i++) s_printf("COM port number %d has a base address of %x", com[i].real_comport, com[i].base_port);

7.2 base/serial/ser_init.c Information

7.2.1 Maintainers

Mark Rejhon <markg@magmacom.com>
Mark Rejhon <ag115@freenet.carleton.ca>

7.2.2 Functions in base/serial/ser_init.c

These are the functions defined in base/serial/ser_init.c.

serial_init This is the master serial initialization function that is called upon startup of DOSEMU to initialize ALL the emulated UARTs for all configured serial ports. The UART is initialized via the initialize_uart function, which opens the serial ports and defines variables for the specific UART. If the port is a mouse, the port is only initialized when i

7.2.3 Items for Fixing in base/serial/ser_init.c

This needs more work before it is implemented into /etc/dosemu.conf as an 'rtscts' option.

7.3 base/serial/ser_ports.c Information

7.3.1 Functions in base/serial/ser_ports.c

These are the functions defined in base/serial/ser_ports.c.

do_serial_in The following function returns a value from an I/O port. The port is an I/O address such as 0x3F8 (the base port address of COM1). There are 8 I/O addresses for each serial port which ranges from the base port (ie 0x3F8) to the base port plus seven (ie 0x3FF). [num = abritary port number for serial line, address = I/O port address]

do_serial_out The following function writes a value to an I/O port. The port is an I/O address such as 0x3F8 (the base port address of COM1). [num = abritary port number for serial line, address = I/O port address, val = value to write to I/O port address]

7.3.2 Items for Fixing in base/serial/ser_ports.c

Should clearing UART cause THRE int if it's enabled?

—

Fix the calculation assumption

—

Is this safe to put this here?

—

Is this safe to put this here?

7.4 base/serial/ser_irq.c Information

7.4.1 Functions in base/serial/ser_irq.c

These are the functions defined in base/serial/ser_irq.c.

serial_int_engine This function is the serial interrupts scheduler. Its purpose is to update interrupt status and/or invoke a requested serial interrupt. If interrupts are not enabled, the Interrupt Identification Register is still updated and the function returns. See pic_serial_run() below it is executed right at the instant the interrupt is actually invoked. Since it is not possible to run the interrupt on the spot, it triggers the interrupt via the pic_request() function (which is in pic.c) and sets a flag that an interrupt is going to be occur soon. Please read pic_serial_run() for more information about interrupts. [num = port, int_requested = the requested serial interrupt]

pic_serial_run This function is called by the priority iunterrupt controller when a serial interrupt occurs. It executes the highest priority serial interrupt for that port. (Priority order is: RLSI, RDI, THRI, MSI) Because it is theoretically possible for things to change between the interrupt trigger and the actual interrupt, some checks must be repeated.

serial_run This is the main housekeeping function, which should be called about 20 to 100 times per second. The more frequent, the better, up to a certain point. However, it should be self-compensating if it executes 10 times or even 1000 times per second. Serial performance increases with frequency of execution of serial_run. Serial mouse performance becomes more smooth if the time between calls to serial_run are smaller.

7.4.2 Remarks in base/serial/ser_irq.c

Linux code hackers: How do I detect a break signal without having to rely on Linux signals? Can I peek a 'break state bit'? Also, how do I 'turn on' and 'turn off' the break state, via an ioctl() or tcsetattr(), rather than using POSIX tcsendbrk()?

7.4.3 Items for Fixing in base/serial/ser_irq.c

how do we cancel a PIC interrupt, when we have come this far?

—

Perhaps this can be modified to limit max chain length?

8 The Mouse group of Modules

All of the Mouse handling code is in the "mouse" subdirectory.

There are only 2 main files, mouse.c and mouseint.c.

8.1 base/mouse/mouse.c Information

8.1.1 Functions in base/mouse/mouse.c

These are the functions defined in base/mouse/mouse.c.

mouse_init Initialize internal mouse.

9 The Bios group of Modules

All of the Bios code is in the "bios" subdirectory.

DOSEMU requires certain code to be coded in assembler and also code to be located in the F000 segment. This is where all such code should be put.

9.1 base/bios/bios.S Information

10 The PIC group of Modules

All of the PIC handling code is in the "PIC" subdirectory.

10.1 dev/pic/pic.c Information

10.2 devpic/pic.h Information

11 And Finally ...

The Following items are used to delimit the text used to create this file. Whilst it is not necessary to know this, they are included because they may be useful for searching, as they are (at least at the moment) reasonably unique.

DANG_BEGIN_MODULE / DANG_END_MODULE This will bracket a description of the file (normally at the start).

DANG_BEGIN_FUNCTION / DANG_END_FUNCTION This brackets a description of functions (good this, isn't it!) Not every function needs to be described in this way - just the major ones.

DANG_BEGIN_REMARK / DANG_END_REMARK This brackets descriptions of obscure items, like data structures and architecture.

DANG_FIXTHIS This is a one line item, indicating a an area requiring a fix, or redesign.

DANG_BEGIN_NEWIDEA / DANG_END_NEWIDEA New Ideas Start Here! As Ideas are proposed, that get added with their description, so that future generations can laugh at or code the ideas ..... These bracket the idea description.

DANG_BEGIN_CHANGELOG / DANG_END_CHANGELOG Changelogs - very useful for bug fixing, and avvailable for use with DPR (or that's the theory)