

Re: Linux syscalls

Source: <http://coding.derkeiler.com/Archive/Assembler/alt.lang.asm/2004-06/0180.html>

From: Beth (*BethStone21_at_hotmail.NOSPICEDHAM.com*)

Date: 06/07/04

Date: Mon, 7 Jun 2004 14:46:37 +0100

Jim Carlock wrote:

> *Beth wrote:*

> <snip>

> *Although, for example, it MUST be true for opening a file in*

> *NTFS or file mapping, as NTFS was never supported by DOS and*

> *file mapping wasn't possible either...*

> </snip>

>

> *A device driver or a rewrite of DOS could permit this, no?*

Yes; But if a re-write of DOS or a significant device driver for specifically supporting a Windows NT filesystem then is this really "DOS inside Windows" or "Windows inside DOS"?

I was answering the specific point about Windows still containing DOS somewhere inside...and, as I stated, if it does, then it has been so "re-written" as to question whether it can be called "DOS" at all anymore?

> *After all NTFS is JUST a file system, like FAT and if the*

> *DOS interrupts are revectorred...*

Yes; Though NTFS is a little more due to its structure...for instance, there's the "journaling" and security facilities...journaling requires constantly writing information about what operations are about to be performed just before doing them, so that, basically, should the power be lost during that operation then, once the OS reboots, it can see the entry in the "journal" about an unfinished operation and quickly and sensibly deal with this...rather than doing that whole scan of the entire disk with "scandisk" as the FAT-based Windows does when it detects an "unclean shutdown"...to maintain the security facilities properly, this would also require additional checks and code to ensure that security is maintained...note that the security features depend on multi-user features (being "logged on" as a particular user to work out what level of "security" is applicable)...

In other words, as I was saying, you could re-write DOS to support this stuff but it would be such a re-write overall that you have to begin to question whether this is "DOS" at all anymore when it is contained inside Windows and re-written to use Windows device drivers...which was the point I was really trying to make there...not really that DOS doesn't have the capability for device drivers but that they are entirely different in nature...

Hence, if you re-write DOS to use the Windows device drivers rather than interrupts and ".SYS" files...then you re-write DOS to deal with NTFS, including caching, journalling, security, file mapping, etc....which implicitly imply re-writing DOS to deal with multi-user capabilities and changing its basic methods of operation somewhat completely...and then re-write DOS in so many different ways that, indeed, that was my point: It's now so "re-written" can it really be called DOS anymore, rather than "code brand new for Windows NT"? None of this was in pure DOS...it can be added and DOS substantially re-written but then you have to ask: "is this DOS anymore? Or is it now, basically, stuff specifically for Windows NT?" :)..

That was the original question and that was the particular context I was answering in...namely, Microsoft say they've eradicated DOS completely...and, though MS's word is hardly reliable (they've lied about this point before ;), things have now so moved on and changed since DOS that the amount of re-writes and modifications and features added on and so forth...

Well, basically, if you re-write every single line of a program then can it really be considered to actually be that original program anymore? And, being so radically different, then how much use would it actually really be to retain a few hundred lines of DOS code, anyway? Plus, DOS was 16-bit code so there would actually be quite some difficulty involved in really re-using these handful of lines under an otherwise 32-bit system against it not really being worth doing, anyway...

Windows 9x was basically "extended DOS", so to speak...slow re-writes of each part piece by piece...but NT was largely "New Technology"...the internals of NT have always been different and a whole lot more "respectable", to be honest...by now, though, DOS must have been eradicated – by re-write and replacement with the 32-bit Windows equivalents – in substantial measure, if it's there at all in any capacity...

Note, though, that I'm hardly defending Microsoft...taking decades to re-write their OS like this does not compare at all favourably to other OSes across many platforms in comparison...this could have happened in the mid '80s...once

the '386 32-bit protected mode was available, they could have set to work on an OS directly that immediately took the approach to "compatibility" with DOS that their current OSes do...by "emulation"...there is no real technical reason why this couldn't have happened then...it's all "other reasons" why it then took another two decades to reach that point...one reason being: "we own a monopoly here so what do we care about doing things quickly? Indeed, if we deliberately take longer then we can milk the technological improvements for all its worth"...for instance, rather than release an OS with all the features in it in one go, you "drip feed" out the technology over two decades, constantly making people pay over and over...you know, you can imagine working on a project where every time you change one or two subroutines, you alter the graphics to make it look like a major difference and then slap a new major version number on it and then sell it...change two more subroutines here and there...right, new version number! Sell it! Blah-blah-blah...

This couldn't really happen outside of a monopoly, though...because competitive companies all trying to undercut each other and "outwit" their rivals would be under constant pressure to make significant changes all the time to keep people from straying to their rivals...it's one of the many, many EVIL things about monopoly that really, really should NOT be tolerated by a single individual who claims to have any kind of trust / belief in the capitalist system...monopoly is where capitalism breaks down...it ceases to work in anyone's interest but the monopoly owner...all that "trickle down" stuff just doesn't work without competitive pressure...if it's a monopoly that no-one can break then the monopoly holder can then relax...no-one's going to break into their monopoly so there's no real need to drive ahead so fast with technological progress (just "tread water" in this regard)...no-one's going to break into their monopoly so why bother to make any efforts to reduce costs or prices (with a rival around, you've got to keep your prices competitive...note that the real reason why Internet Explorer and Media Player and others are free – and often built-in to further exploit the monopoly that rivals don't even get a look-in at all – is because Microsoft had to counter their rivals...rivals who – and this shows really how bad things have gotten in this regard – have to COMPLETELY SURRENDER ALL COMMERCIAL INTERESTS...I mean, you're not going to get very far commercially giving all your software away for free...but all of Microsoft's "rivals" are FORCED to do so because nothing short of "no price whatsoever" is even going to be looked at...after all, even when they are 100% free of charge and easily downloadable, Microsoft simply counter by doing the same themselves and "cheat" by pre-installing them into the OS itself...and considering the amount of money Microsoft has, you're never going to beat them on the "loss leader" front...a "loss leader", by the way, is a product that's deliberately sold

knowing it will make a loss but that entices customers into a supermarket or over to another software company or whatever (and then they try to make up for that "loss" with their other non-free products or with advertising or something similar)...practically no other company is so well-placed as Microsoft to win a "loss leader" war but, well, so desparate is the situation that you _can't_ even consider NOT using a "loss leader" (something 100% "free" to download) or you won't even be looked at...and, even then, Microsoft simply match your "loss leader" with one of their own (IE verus Netscape, Media Player versus Real Player, etc., etc.) and then just "cheat" by hard-wiring it into the OS itself...so who's going to look elsewhere when it's already pre-installed? Plus, there's those alleged "hidden API" which might mean that the OS is "rigging" things that it's not possible for any non-Microsoft product to perform as well on Windows as their own stuff because their stuff is basically "cheating" using "shortcut API" and so forth...

> *I haven't gotten into Assembly and the whole boot up
> procedure as much as I've wanted to... but I do know
> the BIOS reads the first sector of the disk drive during
> boot up to find a set of instructions there (and I don't
> know what instruction set it currently knows, but I think
> it konws the Intel Assembly instruction set, but as far as
> predefined Interrupts go, I don't think there are any. Am
> I correct in this statement ?*

Well, yeah, sure...the x86 CPU _ONLY_ knows the Intel x86 machine encodings (of course, it's really "machine code" in this instance – actual binary – but we could casually refer to this as "Intel assembly", I suppose, from the perspective that the two are basically equivalent but for how they are written out ;)...

As for the "predefined interrupts", then those are the BIOS interrupts...the BIOS comes off the ROM of the hardware itself and is a separate thing from any OS subsequently loaded in...although, in the case of DOS, it actually supplies very little beyond simple disk and memory management with that streamed teletype I/O stuff (you know, "stdin", "stdout", redirection, etc. ;), that usually itself simply calls through to the BIOS to do the actual "grunt work", anyway...

The BIOS stuff is available...which is quite logical, of course, when you think through that it's the BIOS – as you mention above – that has handed control to your bootsector...hence, it has to have initialised itself by that point or it wouldn't be there to load up your bootsector in the first place (and, really, the first sector of a disk is only "special" as the bootsector _because_ the BIOS loads it up in this way to

transfer control :)...

So, you're not entirely correct in your statement...the instructions *_are_* x86 but that stands to reason because the CPU is an x86 CPU (it doesn't understand *_anything_* but the x86 instruction encodings so it couldn't be any other way ;)...but there *_are_* predefined interrupts via the BIOS services...and these actually come off ROM chips in your hardware...

Not just the main BIOS stuff on the motherboard which handles the BIOS set-up...sits next to the back-up battery with a big "Award" or "AMI" or whatever sticker on top of it, usually...but also the video and hard drive BIOSes get copied off their ROM chips into the respective places...this is how "int 10h" knows how your ATI or nVidia or whatever card works in the BIOS interrupts...these are actually "in ROM" and get copied over to be vectored under the "int 10h" interrupts...if you could literally take the code from one BIOS interrupt and stick it onto another machine, then it's quite possible that it wouldn't work...

The BIOS is the original "device drivers" of the PC...each important hardware device having the respective BIOS routines coded in ROM and then these are vectored under the respective BIOS interrupts like "int 10h" for video, "int 13h" for disks...and then there is, of course, the main BIOS chip on the motherboard responsible for the POST memory test and searching for hard drives and setting up plug-and-play tables and, yeah, covers any hardware that doesn't itself supply BIOS routines (the keyboard and mouse, for instance)...

- > *So obviously the BIOS has a limited set of (Interrupts ?)*
- > *commands that it uses to read the disk. Would anyone be*
- > *able to tell me where to find such information about the*
- > *internal (Interrupts ?) commands of the BIOS ?*

They are under "int 13h" ...that's the BIOS's main "disk interrupt" vector...

INT 13 00-- -- DISK -- RESET DISK SYSTEM
INT 13 01-- -- DISK -- GET STATUS OF LAST OPERATION
INT 13 02-- -- DISK -- READ SECTOR(S) INTO MEMORY
INT 13 03-- -- DISK -- WRITE DISK SECTOR(S)
INT 13 04-- -- DISK -- VERIFY DISK SECTOR(S)
INT 13 05-- -- FLOPPY -- FORMAT TRACK
INT 13 05-- -- FIXED DISK -- FORMAT TRACK
INT 13 06-- -- FIXED DISK -- FORMAT TRACK AND SET BAD SECTOR FLAGS
(XT,PORT)
INT 13 07-- -- FIXED DISK -- FORMAT DRIVE STARTING AT GIVEN TRACK

(XT,PORT)
INT 13 08--- – DISK – GET DRIVE PARAMETERS
(PC,XT286,CONV,PS,ESDI,SCSI)

[...]

These are basic disk operations without respect for filesystems,
of course...just about reading and writing disk sectors and that
kind of thing...

More generally, the BIOS tends to categorise things as:

INT 10 – VIDEO
INT 11 – BIOS – GET EQUIPMENT LIST
INT 12 – BIOS – GET MEMORY SIZE
INT 13 – DISK
INT 14 – SERIAL
INT 15 – CASSETTE [now replaced as "BIOS MISCELLANEOUS" really
;)]
INT 16 – KEYBOARD
INT 17 – PRINTER
INT 18 – DISKLESS BOOT HOOK (START CASSETTE BASIC)
INT 19 – SYSTEM – BOOTSTRAP LOADER
INT 1A – TIME

Taking from int 10h to int 1Ah for its basic services (this
actually conflicts on int 10h with the CPU "coprocessor"
exception ('286+)...and on int 11h with the "alignment check"
exception ('486+)...but, well, _originally_ these slots were
chosen by the BIOS almost certainly to be well out of the range
of the CPU exceptions...but, well, as time went on, the numbers
of CPU exceptions went up and the two started overlapping...and
that, once set at these vectors, they couldn't really be easily
moved because of "backwards compatibility" ...an interrupt
routine, though, can check with the interrupt controller chip to
see whether an interrupt was triggered by an actual external
hardware IRQ or whether it was an internal "software
interrupt"...

Obviously, this stuff is "limited" with respect to a complete OS
with all of the typical modern services available...but, well,
it's quite a lot, really, considering it's just the most basic
services for booting up an OS...DOS really only puts things like
a filesystem, (very basic) memory management, a command prompt
and that kind of thing on top of the BIOS to turn it into a

useable system (and generally just "calls through" to the BIOS to actually do the "grunt work" when calling the DOS interrupts...that is, you make a DOS disk call and then DOS "translates" that (with respect to the filesystem) into BIOS calls and makes those BIOS calls to get most things actually done)...so, I know what you mean but "limited" should have its context specifically defined because in terms of "basic services", the BIOS actually provides a remarkably large set, all things considered..."limited" to what you'd expect from a modern OS but actually quite impressive from the perspective of what you'd typically get elsewhere and that this stuff tends to only represent "boot loading API" usually these days...

- > *After it reads the boot sector, the code for the internal*
- > *file system is loaded (Interrupts ?), and the limited set of*
- > *code for the hardware (monitor, disk drives) is loaded.*

No, not quite; The whole BIOS is loaded and ready to use before the bootsector gets control...all of the BIOS interrupts from int 10h to int 1Ah...and the video and hard drive stuff (copying the routines from ROM chips on those devices themselves to the standard BIOS interrupt vectors) happens BEFORE the bootsector gets control...basically, you know all that stuff you see with the copyright messages and the POST memory test and then it might say something about "plug and play" with a list of your detected disk drives appearing and so on and so forth? The whole "POST" ("power-on self test") thing? Well, the BIOS handles all that stuff at this point...in fact, some video card BIOSes deliberately "butt in" and present their own POST-like screens to "memory check" the video RAM and stuff (mine does this, anyway...and I've seen others that are similar, popping up a quick POST-like screen that contains a logo, copyright and that kind of thing :)...and you'll notice that these happen prior to the "POST" screen appearing...it basically immediately proceeds to initialise the BIOS when the power comes on and then "POST" is actually "post" much of that initialisation...to actually "test" that all is in order and that the memory is working and find out what hardware is plugged in and so forth...

There is nothing "after the bootsector", so to speak...the BIOS loads your bootsector and once it hands control to that bootsector, it hands over control...every bit of code executed thereafter is solely the bootsector code and, usually, the OS booting up procedure...

Your PC is fully "ready" for operations by the time the bootsector is loaded...indeed, that is what is going on in the bootsector process...this is where the BIOS is finally "ready" and then looks for an OS to hand over control of the system to...it's all done and dusted regards setting up the BIOS and the machine by the time the bootsector gets its hands on the

machine...

This is why bootsector code is as "raw" as a PC gets (well, an x86 chip could be stuck into a system that has no BIOS...but this isn't really a "PC" in the sense that a "PC" is not just the x86 chip but the whole system architecture surrounding the main x86 CPU)...

Bootsector code typically uses the BIOS disk interrupts to load more sectors off the disk – say, an OS kernel – and then transfer execution to that (where the kernel could proceed to start up an OS by, say, switching to protected mode :)...)

But, in fact, there is actually NO "official" way to go about it...the BIOS simply initialises itself and then hands over the CPU to the bootsector...what the bootsector says goes, so to speak...it doesn't even need to have anything to do with loading an OS really, in that you can just jump straight into some sort of program right there...you could just use the BIOS interrupts and direct hardware access to just, well, code a Space Invaders game or something straight from a floppy bootsector and that would work just fine...not bothering with any "filesystem" (just raw sectors on the floppy that the program is hard-wired to read for the game data :), not bothering with any OS but just doing it all yourself with BIOS or going straight to the hardware directly...

Basically, you can do anything from the bootsector that you feel like...it's just that most people kind of would like an OS of some kind to load...so, rather than a Space Invaders game on a floppy, Microsoft have put together a GUI OS that runs in protected mode (and uses "device drivers" to directly access the hardware, as is needed when you go "protected mode" because the BIOS is only designed for real-mode and stops working properly when in protected mode...)

In fact, one major thing to remember is to set up the "timer" interrupt to do something sensible or switch it off temporarily until you can set it up correctly or the second you switch to protected mode, the first "timer" interrupt goes off, it tries to run real-mode code as protected mode code – BANG! – an exception is sure to happen...so the CPU then tries to jump to the exception handler for this, which, oops, is still real-mode code and causes – BANG! – a "double fault exception" (or the "exception exception", so to speak, when an exception happens while already trying to handle an exception)...it'll try running that code (again, real mode code not designed for protected mode that'll just go AWOL ;) and – BANG! – another exception...and you've just managed the infamous "triple fault" which is totally unrecoverable (and most CPUs are designed to automatically reboot themselves on a "triple fault"...you can't recover

because this many exceptions in a row completely confuses the CPU beyond what it is designed to cope with...so, well, rebooting is about the only thing you can do...and, as I say, many CPUs are hard-wired to automatically reboot themselves when this happens ;)...there is arguably no more impressive an error to trigger than the infamous "triple fault" ...it basically means: you've so confused the machine up that it no longer knows its arse from its elbow and can't reasonably do anything at all except to reboot itself...now, that is one impressive error to generate but it's so incredibly easy to do when switching to protected mode without setting up your interrupts and tables properly beforehand...just switch the "PM" bit in the CR0 register without having done any kind of proper set-up for protected mode first...then sit back and watch the fireworks: BANG! BANG! BANG! "Triple fault" ;)...

But, no, by the time the BIOS hands over control to the bootsector, the BIOS has finished what it wants to do and the system is totally in the hands of the bootsector...this, in fact, is the most powerful and delicate part of the entire system, arguably...and that's why virus scanners try to make absolutely sure you have the proper bootsector...some BIOSes even have a built-in "write protect" / "alarm" settings on the bootsector that it can only be written to when you enable that option, otherwise it denies or issues up an alert if any program tries to take control of the bootsector...whatever program controls the bootsector effectively controls the entire machine because it solely decides what happens when the machine boots up...the BIOS simply blindly trusts the bootsector and hands over execution so long as the "0AA55h" signature in the last word of the bootsector is correct (and some BIOSes don't even bother with even that check)...

Yeah, a computer doesn't actually need an OS at all, you know...you could have a bunch of floppy disks where they all "autoboot" (have their own executable bootsectors :) and then not bother with the hard drive at all (well, you could still use the hard drive...the problem with this comes about from: "what filesystem?" for multiple programs to share the hard drive space without overwriting each other...but, well, you could make a "dedicated" machine where there is only one program and it uses up the entire hard drive for its own purposes ;)...of course, a somewhat limited system there...but possible, if you were ever insane enough to want to try it and have weird requirements that this would fit what you wanted to do...

You also, in fact, don't even need the BIOS either...what the BIOS is there for is as a kind of set of basic "device drivers"...IF you know what hardware you've got then you could even completely ignore this and have your code use the "in / out" I/O port commands and "mov" instructions to control the

entire system...

After all, the OS is `_SOFTWARE_`...so how do you think the OS manages it? And even the BIOS is `_SOFTWARE_` too...so, again, how do you think the BIOS manages to do what it does? It's just `_very low-level_` code (quite possibly ASM in the case of the BIOS because there are also very strict `_size_` requirements to keep to because there's only so much room in the real-mode memory for the BIOS routines to be housed in :) that handles the "nitty-gritty" stuff of dealing with things like the disk...conveniently, as things like video cards could actually be quite different to one another, the BIOS routines for the video are actually copied off the ROM on the video card itself...this is how it gets around "portability" problem...the video card carries its own BIOS routines on a ROM chip...so, clearly, the correct "BIOS driver software" actually moves `_with the hardware_` itself (stuck onto a ROM chip on the card itself :)...and, thus, you'll be guaranteed to always "have the right BIOS drivers", so to speak...

There is no "internal filesystem" or anything at all like that...an OS is just a piece of `_SOFTWARE_` (nothing special about it at all, except for its `_purpose_` in being an OS and being booted up to satisfy this role)...the bootsector uses the BIOS interrupts to load simple physical sectors off the disk (no "filesystem" but literally grabbing raw sectors of information from the disk :)...and then, once the OS is running, it merely "keeps to the filesystem rules"...yes, a "filesystem" is purely abstract, in that sense...just, really, a "set of rules" for storing data on the hard disk that the OS understands and keeps to when it is dealing with the disk...

For an OS like DOS, then it simply loads in its OS routines to memory...then simply changes the memory address in the interrupt table corresponding to "int 21h" (and the other few interrupts DOS takes up :) to point to code that reads the "function number" and jumps to the correct DOS function...`_JUST SOFTWARE_`...nothing special...nothing special at all...

BUT you might have got the wrong impression of this from Microsoft – unsurprisingly – not just in their "advanced technology" bullcrap hype marketing, as if only they could write the software needed...but also, FAT formatted floppies give you this impression because if you leave a DOS FAT-formatted floppy disk in the drive then the BIOS tries to load this and it says something like "Non-system disk; Switch disks and then press any key"...

So, logically, you might be thinking "hey, the BIOS knows its not a bootable disk with an OS on it...the BIOS must be doing stuff to check or something"...nope, not at all...it's just that

all Microsoft FAT formatted floppy disks include a bootsector that prints the "non-system disk" message itself...it isn't the BIOS that is saying this with a FAT formatted non-bootable disk but the disk `_IS_` bootable, in fact, but boots to a program that does nothing but print this message...

Yup, this bizarre method of doing things is standard Microsoft programming...`_every single_` floppy carries its own "non-system disk" message printing routine in the bootsector...and, also, just to show Microsoft Love to do this all over the place, `_every single_` PE executable has a "stub" DOS program at the front which prints out the "This program requires Windows" message...in actual fact, though, the BIOS has `_no idea_` that it's not a bootable disk and, in fact, they `_ARE_` bootable disks but they boot up a small program in the bootsector which, yeah, bizarrely prints "can't boot this disk!" on the screen...and DOS has `_no idea_` that a Windows PE file is a Windows PE file either...nope, every PE file carries a valid DOS program that says "can't run this program without Windows" message...

Yeah, Microsoft haven't quite got the idea of "code re-use" here exactly...they are "re-using" in the sense that the very same bootsector program is copied onto `_every single floppy disk_` to say "can't boot this disk!" and that the very same DOS program is stuck onto the front of every PE file to say "can't run without Windows!"...every single file, every single time...absolutely identical...duplicated thousands of times all over the system...total waste of space...

It all stems from a lack of foresight, basically...original DOS ran ".com" files which are raw binary files so they can have `_anything_` inside them and there's no header to say "yes, this is a valid executable" or anything...then they improved this with the ".exe" file format...then along came the NE and PE formats for Windows but, oops, older DOS was written before these formats were invented so it can't recognise them as it should and print some "this is a Windows program!" message...it'll just think they are ".com" files and try to run them, which is very likely to crash DOS completely...the "work around" was to stick an "MZ" DOS stub program at the start so that the older DOS would see a DOS ".exe" program and run the "This program requires Windows!" message program and quit with the rest of the Windows program "hidden" passed this...but Windows will recognise the NE PE formats and just ignore the quite pointless DOS "stub" program...

And if only `_all_` BIOSes correctly `_REJECTED_` all disks that don't have the "0AA55h" signature at the end and would automatically print the "can't boot!" message, if it can't find any bootable disks, then there'd also be no need for the bizarre bootable bootsector that boots up to say "can't boot!"...but

this was added because some BIOSes don't bother to look for the "0AA55h" signature and reject disks if it's not there, as they really SHOULD be doing, so the "cheap and cheerful" solution was to, well, make every single disk bootable but that it, ummm, just prints "Can't boot! Please swap disks!" then re-triggers the boot process to try again...

The original IBM PC from which all PCs stem actually would boot up a BASIC interpreter and had support for a cassette drive...stuff that was rather "standard" on all other home computers (or "desktops" as they are called these days ;)...but, in fact, it's all to do with IBM's many "mistakes" that things are really as they are...the PC "clones" couldn't copy IBM's BASIC because, unlike the rest of the PC system, this was not cloneable due to copyright (yeah, IBM had copyright on their BASIC software but hadn't really bothered with the PC design itself to stop "clones" springing up copying it ;)...the "clones" didn't mind not having the BASIC software because, well, they could just boot up an OS like DOS and use that instead...if you didn't have an OS? Ah, well...then the machine just wouldn't work...and, as the "clones" took over everything – something like the plot of Star Wars with "clone wars" and stuff ;) – then it was just "normal" that PCs would have hard drives with Oses installed on them...a certain Mr.Gates provided a crap not-really-an-OS-at-all-in-fact program called DOS, which he'd just sneakily bought off someone else for next to nothing, changed the logo to read "Microsoft" and added a few "features" – that, well, does anyone even remember what they were? – and then Mr.Gates did various naughty things that has lead us to this nightmare disaster of a situation where the software industry has been savagely struck down with VB programmers everywhere bred up to write absolutely crap bloatware without the slightest idea what the hell they are actually doing...

These really were all "accidental empires" from start to finish...IBM created the PC revolution by total accident and that's ultimately why they lost control of it all because, well, they really hadn't expected it to go anywhere but to, well, "spite" Apple by stealing some of their customers...Intel were lucky that IBM changed their mind at the last minute to stuff in an 8086 chip when they had actually designed it to use a Motorola and all those sales would have gone to Motorola instead...and Microsoft just snuck in at the right time in the right place with Sir Bill's mercilessly aggressive business tactics that he was working almost straight away on his "plans and ploys" like "bundling" pre-installed software...most of the people Bill conned and bluffed to get in his position, to be fair, would often incredibly stupid and naive in dealing with him (like the "trade" for Microsoft software for their Mac, if they gave Microsoft rights to use their "TrueType" ...without

that, Microsoft would likely have been in the crap because if you ever saw their pathetic "vector fonts" before they got TrueType from Apple, then they would likely have been blanked when their word-processing and spreadsheet software couldn't really manage "WYSIWYG" ("what you see is what you get") at all with those crap "vector fonts" – composed of simple straight lines only...no filled in areas or curves...just a set of lines and nothing else – that, like the original Windows which couldn't overlap windows, Microsoft's programming has always been crap and their technology behind or stolen...Apple should never have made that deal for TrueType because it made Windows _visually acceptable_ that people stopped noticing how badly programmed it was behind the facade...Microsoft seriously couldn't often program simple things that every other OS and software manufacturer could manage no trouble)...

> *Then it can go about displaying information via some
> internal commands (Interrupts ?) to the screen,*

Well, the "Starting MS-DOS..." message is probably written using the _BIOS_ teletype routine (AH = 0Eh, int 10h :)...until DOS itself is loaded in...note that DOS's print string routines basically just call through to these BIOS interrupts, anyway...but, yeah, after DOS is loaded, it might use its own routines – call itself – to get things done...though, ultimately, it's just calling the BIOS video interrupts, in the end...

> *reading
> the appropriate file to be loaded (in MSDOS, the first
> file was IO.SYS, then MSDOS.SYS, and then the
> last OS file was COMMAND.COM, and then device
> drivers and programs and applets were loaded from
> config.sys and autoexec.bat).*

Yes; Basically, roughly speaking, the bootsector will load that IO.SYS file and maybe MSDOS.SYS into memory (just "raw disk reads" of the disk sectors of the files into memory with the BIOS interrupt...the bootsector can be written to understand, perhaps, a little of the FAT filesystem to look for the "IO.SYS" file off the disk...or it can just have these files in fixed disk locations to load them up wherever they are :) and then just makes a "JMP" into that code it has loaded...

> *Windows 3.1 and Windows 9x continued along that
> precedent above but Win9x didn't need to have the
> Win.com typed out inside the autoexec.bat file.
> Win9x started loading win.com via the command.com
> file I think. Am I correct ?*

Don't know the exact details myself (never looked into it into any detail)...but, yeah, my understanding was that it was *_something_* like that...

> *When Win9x gets loaded, a lot more files are read.*

Oh, yeah, totally; With Windows, even the Library of Congress looks small in "file size" terms...boy, does it load "a lot more files"! That's a bit of an understatement...

> *I don't know about how NT loads itself. I think it loads the
> nldr file, then runs the NTDETECT.COM program finally.*

Basically, yes; From what I can gather, NT's bootsector loads NTLDR and then simply executes that to do the loading process...only the original bootsector is actually limited to 512 bytes...but, from that bootsector, you can load much bigger files that, indeed, can basically have the entire loading process / protected mode switch / Kernel and executive initialisation and then the little automatic "boot shell" command slotted at the end to load up the actual GUI shell and all that kind of thing put inside it...

> *"Beth" wrote:*

> *<snip>*

> *DOS never used device drivers either...*

> *</snip>*

>

> *I'm not sure what that means. My understanding of DOS*

> *using device drivers means the device driver is loaded*

> *by being placed inside of the config.sys or the autoexec.bat.*

> *Most device drivers in fact were loaded up inside of the*

> *config.sys, but that means everyone has to conform to the*

> *DOS disk reading rules and this is still true.*

I miswrote that (a "colloquial technical inaccuracy" ;)...DOS *_did_* have "device drivers" BUT what I meant was they are NOT the Windows-style of device driver at all...not compatible, not the same style, real-mode rather than protected mode, etc....retaining these under Windows XP would be, well, slightly useless...and that's all I really meant there...

> *Even Win2K does not support UDMA and the device*

> *drivers need to be loaded via a separate diskette.*

>

> *I think you must have a different definition of a device*

> *driver...*

Well, sort of...I generally don't have a different definition in the sense that, yeah, those things under DOS were "device drivers"...but, indeed, I generally don't think of them as being

quite the same thing as the device drivers in Windows (or Linux or OS/2 or whatever ;)...

- > *Windows XP has many more extra device drivers*
- > *included with it. In fact I think the device drivers, the help*
- > *files and the extra programs take up at least half of the*
- > *file space for the operating system when the OS is installed.*
- > *In fact XP backups about 500 MB of operating system*
- > *files (the DLLCACHE folder). In fact my DLLCACHE*
- > *folder currently 944,000 KB of backups. Win2K is very*
- > *much similar.*

Yeah; To be honest, trying to comprehend how Windows works might be the wrong approach...perhaps we should simply label it as "utter crap" and then look at how *_real_* OSes go about things instead...hehehe ;)...

Beth :)