

Re: devolopin a mew lang.....

Source: <http://linux.derkeiler.com/Newsgroups/comp.os.linux.misc/2006-06/msg01798.html>

- *From:* Aragorn <stryder@xxxxxxxxxxxxxxxxxxx>
 - *Date:* Thu, 29 Jun 2006 23:44:48 GMT
-

On Thursday 29 June 2006 19:08, prashant stood up and spoke the following words to the masses in /comp.os.linux.misc...:/

Tell me the way that we could make a executable programmin as basic as MS dos as like it can interract with machine directly with out the help of other prorams like other DOS.

Could it be that you are simply inquiring about a commandline shell – which is how DOS interacted with the user? If so, you have to understand how GNU/Linux works...

You see, *Linux* is only a kernel. It comprises of the actual kernel core and the drivers, which can be compiled in–line with the kernel core – in which case we say it is "monolithically compiled" – or which can be compiled as loadable (and unloadable) modules – in which case we say it is "modularly compiled".

Considering the wide variety of hardware on the market and the fact that the GNU/Linux operating system supports the largest amount of hardware architectures ranging from embedded devices over the Intel /IA32/ and /IA32–64/ platforms up to workstation–class RISC CPU's, mainframes and supercomputers – being a UNIX–style operating system, it's designed to be highly portable – most binary distributions ship their kernels in a highly modularly compiled fashion. This way, the kernel can detect what hardware is in the system and can load modules on demand or per superuser command – or as defined via **/etc/modprobe.conf** and **/etc/modprobe.preload.**

The kernel manages the hardware, which is a trait of monolithic kernels – not to be confused with "monolithically compiled kernels", but to be seen in contrast with a microkernel design or an exokernel design.

As I've explained in another post recently the differences are the following:

(a) A monolithic kernel manages process scheduling, memory management and hardware access.

Re: developin a mew lang.....

- (b) A microkernel manages process scheduling and memory management.
- (c) An exokernel manages process scheduling.

That which is present in (a) but not in (b) and (c), or present in (a) and (b) but not in (c) takes place in what we call userspace. To understand this concept, you have to take into account that all modern microprocessors feature four privilege levels – called /rings/ – ranging from "0" for the highest privileges to "3" for the lowest privileges. All modern operating systems compiled for such microprocessors only use two of those privilege levels, i.e. ring "0" and ring "3".

Ring "0" is what we call kernelspace. This is where the kernel lives. Ring "3" is userspace. This is where user-started programs live. If something in ring "3" crashes, it won't take down the whole system, as it runs in a shielded environment. As GNU/Linux is a UNIX-like operating system, it's designed to be a multi-tasking multi-user system, and thus it needs such protection.

I don't know if it's true, but I was told that DOS ran entirely in kernelspace. DOS was also not a real operating system since it only needed to load an executable from disk into memory, and then it handed over all access to the machine to this executable.

The latter is only possible when the CPU runs in /real/ /mode./ This is the /x86/ CPU's /i8086-compatibility/ mode, in which it can only use 16-bit instructions and access about 1 MB of physical memory – DOS itself could only access 640 KB.

In /real/ mode, the memory addresses – denoted by a 64 KB segment identifier offset and an in-segment offset address – actually addressed the real physical memory locations, and DOS – or the program loaded in memory by DOS – could directly access hardware, like the parallel port, the graphics adapter, the disk controller, etc. This is also what made DOS unstable, and why the DOS-based versions of Windows – i.e. Win95, Win98 and WinME – were so unstable as well.

If the above is what you're trying to accomplish, then you're attempting to reinvent the proverbial wheel, but at the same time you'd also be stepping back in time to where the wheel was actually invented. Modern CPU's don't work that way, as the need for multi-tasking and multi-user operating systems – basically, this is just a spill-over from the world of minicomputers and mainframes to what was once considered a gadget, i.e. "the PC" – require stability and security measures, which are offered via the design of modern 32-bit and 64-bit CPU's.

If on the other hand you only want to interact with the system the way you used to do in DOS – i.e. to use a commandline shell – then any UNIX and especially GNU/Linux must be paradise for you. GNU/Linux comes with at least two shells by default, and many distributions include a couple more. They are to UNIX what /COMMAND.COM/ was to DOS.

Re: developin a mew lang.....

Re: developin a mew lang.....

The most commonly used shell in GNU/Linux is `/bash/`, the GNU Bourne Again SHell. It's name is a pun to `/sh/`, the original UNIX Bourne Shell. `/bash/` however has far more features than the original Bourne Shell but is fully compatible with it. Yet even the original Bourne Shell already makes the DOS commandline interpreter look like a kindergarten project.

Next to `/bash/`, most GNU/Linux distributions also feature `/tsh/`, an Open Source/Free Software version of `/csh/`, the UNIX C SHell. It's also a highly functional command interpreter – it uses the C language – but is less flexable as a script interpreter than `/bash/`.

Yet another shell – which if not supplied by your GNU/Linux distribution can still be freely downloaded – is `/pdksh/`, the Public Domain Korn SHell. This is a Free Software version of the UNIX Korn SHell, which in turn was supposed to be an enhanced version of the older Bourne Shell but which was stripped of most of its enhancements by Sun Microsystems, very much to the dislike of its developer.

If you put the above three in a comparison table, then `/GNU/ /bash/` wins hands down in usability, flexibility and power. Not only is it a commandline interpreter, but it's also a(n interpreted) programming language in its own right, in which most of the shell scripts in the common GNU/Linux distribution have been drawn up.

GNU/Linux does allow tinkering with hardware settings and kernel settings from within userspace, though. For this purpose, it uses abstraction layers.

An example of such an abstraction layer is the `*/proc*` filesystem. The `*/proc*` directory contains a lot of files and other directories, all of which are not physically present on your disk but only exist in kernel memory. Naturally, it goes without saying that in a multi-user environment like GNU/Linux, one needs root privileges – i.e. superuser privileges – to write to kernespace via `*/proc.*`

Similar to `*/proc*` – and originating as a subdirectory of it at first – is `*/sys.*` This filesystem is also virtual – i.e. its contents are not on your hard disk – and it is used in conjunction with another abstraction layer, in the form of `/udev/`. Read on, it gets even more exiting – if you're into the technicals, that is... ;-p

Traditionally, everything in UNIX is considered to be a file. Files are files. Terminals are files. A hard disk is a file. A hard disk partition is a file, etc. UNIX operating systems group these special files in a directory called `*/dev.*`

Now, also traditionally, the contents of `*/dev*` did (and often still do) physically exist on the hard disk, but only as device special files – there are various kinds of those.

Re: developin a mew lang.....

Re: developin a mew lang.....

Now, what /udev/ does is make use of a RAM-based filesystem – in DOS-speak: a RAM-disk – for the contents of */dev,* and in conjunction with the virtual */sys* filesystem, /udev/ can thus populate the in-RAM */dev* filesystem with the required device special files on the fly – e.g. when a hotplug event occurs or at boot time – while the system doesn't have any device special files anymore that aren't needed, e.g. because the hardware for it isn't present in the system. /udev/ not only creates the device special file for newly found hardware, but also automatically loads a driver module for it if necessary.

Again, needless to say that whatever you want to change to the */proc,* */sys* or */dev* filesystems requires that you make your changes as the /root,/ i.e. the superuser – normal users don't have the required privileges to do so, and shouldn't have those.

This is the whole foundation of the UNIX security model, and history has proven the developers right at implementing things this way – you must never forget that UNIX operating systems are and were designed as multi-user operating systems for hardware that allowed concurrent use by multiple users and concurrent execution of multiple processes by those multiple users – unlike Windows or DOS for instance, which were designed as single-user platforms for standalone machines.

If however you still feel like you should want to interact with the hardware directly without making use of abstraction layers, there's only one other option...: become a programmer and submit kernel drivers to the kernel development team. But then again, I don't think this is what you had in mind.

You probably just want more control over your GNU/Linux box, but you just don't know how yet, right? Well, the control you seek is there. The only thing you need to do in order to harness that power is read up on a few things regarding the design of GNU/Linux.

Something very technical but of excellent quality and highly recommended reading – it's a very long read though – is the /RUTE,/ which you can find here...:

<http://www.chongluo.com/books/rute/>

The following link deals with the latest edition of the UNIX Filesystem Hierarchy Standard (FHS)... – it more or less makes a distinction between GNU/Linux and other UNIX systems:

<http://www.pathname.com/fhs/pub/fhs-2.3.html>

Various /HowTos/ regarding just about anything – including /bash/ and /bash/ scripting, but also kernel hacking [1] – can be found here...:

<http://www.tldp.org/>

Re: developin a mew lang.....

Re: developin a mew lang.....

.... and in regards to /bash/ scripting, I would also like to recommend the book written by /Chris/ /F.A./ /Johnson,/ who happens to be a regular poster in this and other GNU/Linux Usenet newsgroups. The following link leads to Chris's website...:

<http://cfaj.freeshell.org/>

[1] The word /hacking/ is most commonly used in an incorrect sense by the media – both in journalism and in movie storylines – and by the less savvy IT people as being "breaking into computer systems", with a distinction between "good hackers" – who only break into systems to expose security flaws – and the "bad hackers", who have criminal intentions.

However, those people are all *crackers,* not /hackers!/ A /hacker/ is nothing other than a programmer – a "geek", if you will. "Kernel hacking" therefore means nothing other than writing or modifying kernel code.

If you ever plan on doing the latter, then you must acquaint yourself with what Free Software really means. It's not so much about "free as in free beer" but about "free as in freedom". For this purpose, I would like to point you to the website of the Free Software Foundation, architects of the better part of what makes up for a GNU/Linux operating system minus the Linux kernel – that honor belongs to Linus B. Torvalds and friends – and of the GNU General Public License (GPL) under the terms and conditions of which the Linux kernel is being published.

<http://www.fsf.org>

<http://www.gnu.org>

Note: As Linux is only a kernel and most of the rest of the operating system software – including the software used to build and compile the Linux kernel – comes from the GNU project, many (but not all) of us call the operating system GNU/Linux. In general however, people usually just refer to the entire operating system as "Linux". I guess it's just a matter of opinion. ;-)

This was a somewhat long post – well... "somewhat" – but I felt that all of the above was required information in light of your somewhat vague question. You appear not to be a natively English speaker, and you left the estimation on your degree of knowledge regarding operating system design wide open to highly varying interpretations.

I hope to have answered your question adequately. ;-)

—

Re: developin a mew lang.....

Re: developin a mew lang.....

With kind regards,

Aragorn

(Registered GNU/Linux user #223157)

.