Operating System

Kernel and the boot process

Nick Urbanik <nicku(at)nicku.org>

Copyright Conditions: GNU FDL (see http://www.gnu.org/licenses/fdl.html)

A computing department
Operating System: Kernel and boot process

What is it?

What does it do?

How does it start up?
What is an operating system?

- Is it what you get when you install Linux, Windows XP or Windows 2000?
- Does it include such things as (g)notepad, g++ or Visual C++?
- How about `bash`, `cmd.exe` or `command.com`?
The os is the kernel

- The operating system is the kernel
- When the computer boots the operating system, it loads the kernel into memory.
In Linux, kernel can be loaded by LILO or grub

Kernel is in /boot

In RH 9, it is
- /boot/vmlinuz-2.4.20-20.9,
- or if you build your own, something like /boot/vmlinuz-2.4.22-ac6

It is a *monolithic kernel*
Kernel in Windows XP, 2000, Win NT

- In `%SystemRoot%\System32`
  - `%SystemRoot% = C:\winnt, or D:\winnt,...`
- Called `ntoskrnl.exe`
- Microsoft call it a *layered kernel* or *microkernel*.
- sometimes called the “Executive services” and the “NT executive”
- Bottom layer is the *hardware abstraction layer*
What does an OS do?

- Provides a “government” to share out the **hardware resources** fairly.
- Provides a way for the programmer to easily work with the hardware and software through a set of **system calls** — see slides §15–§18.
  - Sometimes also called **supervisor calls**.
Is there a User Friendly os?

- Some people have said that the Windows OSs are more user friendly than Linux.
- Can this be the case?
  - Are the system calls more user friendly?
    - (see slides §15–§18 for more about system calls)
  - Does Windows manage the hardware in a more user friendly way?
- No!
- The user interface is not an operating system issue. See your subject Human Computer Interfaces (HCI)
- Do you want a more user friendly interface for Linux?
  - Then write one! Contribute to the Gnome or KDE projects.
The Mac has a deserved reputation for a great user interface

**OS X** is the latest OS from Apple

Very beautiful, easy to use

But it is Unix, built on FreeBSD!
- The Unix that till now has mostly been used on servers;
- considered by some to be less user friendly than Linux

The User Interface is not part of the OS
Is Internet Explorer part of the Windows operating systems?

Please discuss this question with your neighbour.

See

What resources does OS manage?

- The OS manages resources such as:
  - Use of CPU
  - Memory
  - Files and disk access
  - Printing
  - Network access
  - I/O devices such as keyboard, mouse, display, USB devices, ...
An operating system can be *multiuser*
- In this case, resources must be allocated to the users fairly

“Proper” operating systems are *multitasking*
- Resources must be allocated fairly to the processes

Users, processes must be protected from each other.
Kernel mode and user mode

- **Kernel** means “central part”
- The kernel is the central part of OS
- It is a program running at all times
- Application programs run in “**user mode**”
  - Cannot access hardware directly
- Kernel runs in “**kernel mode**” (or “**supervisor mode**”)
  - Can access hardware, special CPU registers
How does user program access hardware?

- A program that writes to the disk accesses hardware
- How?
- Standard library call, e.g., `fprintf()`
- Library contains **system calls**
  - see slides §15–§18
- A system call passes the request to the kernel
- The kernel, (executing in kernel mode always) writes to the disk
- Returns telling user program that it was successful or not
This is the second important function of the operating system.

Provides a standard set of system calls, used by the libraries.

User programs usually use the system calls indirectly since libraries give higher level interface.
System Call

- Low level details:
  - CPU provides a *trap* instruction which puts the CPU into a privileged mode, i.e., kernel mode
  - On Intel ix86 architecture, the trap instruction is the `int 0x80` instruction
  - See `include/asm-i386/unistd.h` and `arch/i386/kernel/entry.S` in Linux source code. See also [http://en.tldp.org/LDP/khg/HyperNews/get/syscall/syscall86.html](http://en.tldp.org/LDP/khg/HyperNews/get/syscall/syscall86.html)
  - Sometimes called a *software interrupt*
  - put parameters into CPU registers before the call
  - save values of many registers on a stack
- High level: all this buried in external library interface
- **POSIX** specifies particular function calls that usually map directly to system calls — see `man` section 2
- Provide a higher level interface to system calls
- Less than 300 of them. Examples:

<table>
<thead>
<tr>
<th>Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pid = fork()</code></td>
<td>Create a child process identical to parent process</td>
</tr>
<tr>
<td><code>exit( status )</code></td>
<td>Terminate process and return status</td>
</tr>
<tr>
<td><code>fd = open( file, O_RDONLY )</code></td>
<td>Open a file for reading, writing or both</td>
</tr>
<tr>
<td><code>status = close( fd )</code></td>
<td>Close an open file</td>
</tr>
<tr>
<td><code>n = read( fd, buffer, nbytes )</code></td>
<td>Read data from file into a buffer</td>
</tr>
<tr>
<td><code>n = write( fd, buffer, nbytes )</code></td>
<td>Write data from buffer into a file</td>
</tr>
<tr>
<td><code>status = chdir( dirname )</code></td>
<td>Change working directory of process</td>
</tr>
</tbody>
</table>
Win32 API provides many thousands of calls
No one-one mapping to system calls
Not all make a system call
On some versions of Windows OSs, graphics calls are system calls, on others they are not
Win32 API documented on MSDN. Examples:

<table>
<thead>
<tr>
<th>POSIX</th>
<th>Win32</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fork</td>
<td>CreateProcess</td>
<td>create a new process</td>
</tr>
<tr>
<td>exit</td>
<td>ExitProcess</td>
<td>Terminate execution</td>
</tr>
<tr>
<td>open</td>
<td>CreateFile</td>
<td>Create a file or open existing file</td>
</tr>
<tr>
<td>close</td>
<td>CloseHandle</td>
<td>Close a file</td>
</tr>
<tr>
<td>read</td>
<td>ReadFile</td>
<td>Read data from a file</td>
</tr>
<tr>
<td>write</td>
<td>WriteFile</td>
<td>Write data to a file</td>
</tr>
</tbody>
</table>
A rough breakdown of the types of OS
What types of operating systems are there?

- There are **four main categories**; depends on *organisation* of the *kernel*

  - **Monolithic** operating systems
    - Linux is a monolithic OS
  
  - **Layered** operating systems
    - Windows NT/2000/XP/2003 is described as a layered architecture

  - **Microkernel with client server architecture**
    - The QNX real-time OS is truly a microkernel; the kernel is said to be only eight kilobytes in size!
    - Andrew Tanenbaum wrote the MINIX operating system as an example microkernel OS for students to study
    - The GNU Hurd OS has a microkernel architecture
    - Windows 2000 is described as having a hybrid layered-microkernel architecture, although Andrew Tanenbaum disagrees:
      

  - **Virtual machine** architecture
Monolithic Kernel

- A monolithic kernel has all *procedures* in the same *address space*.
  - This means that all the code can see the same global variables, same functions calls, and
  - there is only one set of addresses for all the kernel
- Purpose is *speed*:
  - to reduce overhead of communication between layers
Monolithic kernel — 2

Monolithic kernel includes:
- virtual memory
- I/O
- file handling
- scheduling
- device drivers,...
To avoid chaos, a monolithic kernel must be well structured.

Linux kernel uses *loadable modules*, which support hardware and various software features.

Such as RAID, Logical Volume Managers, various file systems, support for various networking protocols, firewalls and packet filtering,...
Monolithic kernel: loadable modules

- dynamically loadable modules to support hardware, device drivers

user mode

kernel mode

Hardware

client process 1

client process n
Monolithic kernel: Loadable Modules

- Loadable modules in Linux kernel support:
  - *Dynamic Linking*: modules can be loaded and linked with the kernel, or unloaded, while kernel is executing
  - *Stackable Modules*: Modules can provide support for each other, so many modules can be stacked on a lower level module.
  - Reduces replication of code
  - Hierarchical structure ensures that modules will remain loaded while required
  - View loaded modules by typing lsmod
Layered kernel

- Has different levels; example:
- Lowest level manages hardware
- Next level up manages, e.g., memory and disks
- Next level up manages I/O, . . . .
- Each layer may have its own address space
- Communication between layers requires overhead
- Advantage is different layers cannot interfere with each other.
<table>
<thead>
<tr>
<th>Level</th>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Primitive process management</td>
</tr>
<tr>
<td>1</td>
<td>Virtual memory</td>
</tr>
<tr>
<td>2</td>
<td>I/O and device management</td>
</tr>
<tr>
<td>3</td>
<td>Interprocess communication</td>
</tr>
<tr>
<td>4</td>
<td>File Systems</td>
</tr>
<tr>
<td>5</td>
<td>User Programs</td>
</tr>
</tbody>
</table>

**Layered Kernel — 2**

- Operating System: Kernel and boot process
- Role of OS
- System Calls
- Types of OS
- Monolithic Kernel
- Layered Kernel
- Layered kernel
- Layered Kernel — 2
- Microkernel
- Virtual machine
- Boot Process
- Runlevels
- References
Microkernel with Client-Server Arch.

- Microkernel architecture keeps the kernel as small as possible, for the sake of reliability and security
- As much is done in the user space as possible
- User space provides servers, such as memory server, file server, terminal server, process server
- Kernel directs requests from user programs to user servers
Microkernel Architecture — 2

- client process
- device drivers
- file server
- process server
- virtual memory

Hardware

Microkernel

user mode

kernel mode
Most of operating system is a set of user processes
the server processes do most of the work
The microkernel mostly just passes requests from client processes to server processes

Client obtains service by sending messages to server process

Microkernel

Client process
device drivers
file server
process server
virtual memory

Microkernel Architecture — 3
Microkernel Architecture — Examples

- Mach kernel used as core for many Unix OS
  - including the MAC OS X
- GNU Hurd OS, initiated by Richard Stallman for the GNU project
- The QNX distributed real-time Unix-like OS
  - kernel only 8 KB in size!
- It can be *debated* whether Windows NT/2000/XP/2003 operating systems are microkernels:
  
  “With all the security problems Windows has now, it is increasingly obvious to everyone that tiny microkernels, like that of MINIX, are a better base for operating systems than huge monolithic systems.”

  — Prof. Andrew Tanenbaum,

Windows 2000 is described as a hybrid between a layered architecture and microkernel architecture.

HAL provides an abstract machine—aim to make porting to other hardware architectures easier.

HAL + Microkernel ≈ normal microkernel

- **Windows 2000 Architecture**
  - Environment subsystem
  - Executive
  - Microkernel
  - Hardware Abstraction Layer (HAL)
  - Hardware

- **Windows 2000 Architecture**
  - Windows 2000 Architecture — 2
  - Windows 2000 Architecture — 3
  - Examples

- **Operating System**
  - Kernel and boot process
  - Role of OS
  - System Calls
  - Types of OS
  - Monolithic Kernel
  - Layered Kernel
  - Microkernel
  - Microkernel with Client-Server Arch.
  - Microkernel Architecture — 2
  - Microkernel Architecture — 3
  - Microkernel Architecture — Examples

- **Virtual machine**
- **Boot Process**
- **Runlevels**
- **References**
Environment subsystem aims to support DOS, Win32, OS/2 applications
- each environment subsystem uses a DLL (dynamic link library) to convert system calls to Windows 2000 calls
- The I/O manager contains file system and device drivers
- Microkernel, HAL and “many functions of the executive” execute in kernel mode.
  - Sacrifice advantage of microkernel of reduced code executing in kernel mode
  - to reduce communication overhead
Virtual machine

- Virtual hardware
- Many operating systems run independently on same computer
- IBM now selling mainframes running many instances of Linux to Telecom companies — see next slides
- VMWare allows something similar on PC: http://www.VMWare.com
- http://www.connectix.com/ used to sell Virtual PC and Virtual Server, but they have been bought out by Microsoft, who of course, have dropped Linux support: http://www.msfn.org/comments.php?id=5516&catid=1
- Java Virtual machine also provides virtual hardware that all programs can execute on.
IBM designed the CP/CMS virtual OS for their S/360 mainframe.
Later called VM/370 to run on their S/370 mainframes
Later called VM/ESA on the S/390 hardware
Now sold as zVM® running on zSeries mainframes
  Supports running many different OS, particularly Linux
  See http://www.vm.ibm.com/
See how MIT run Linux on VM/ESA on their S/390 mainframe:
http://mitvma.mit.edu/system/vm.html
Search the web for articles on Linux running on mainframes.
Virtual zSeries 900 machines

User apps

Linux 1 Linux 2 Linux n

trap here

zVM

zSeries 900 Hardware

OS executes
I/O instruction here:

Application
makes a system
call here:

Application
makes a system
call here:

Application
makes a system
call here:
A data centre may have many servers
- Each must be powerful enough to meet *peak demand*
- Most are not at peak demand most of the time
- ...so most are *underused*
- ...but must pay for electricity for cooling, and for powering all that *reserve capacity*
Many Virtual Machines, one Mainframe

- Can replace many individual servers with one mainframe running many instances of an OS such as Linux
  - The demand spread out among all the virtual machines, total *utilisation high* — demand shared
  - busy virtual machines get more CPU power to meet peak demand
  - Much lower power requirements
  - Much less air conditioning cost
  - Much less floor space required

- Virtual machines partitioned from each other, like the individual machines in data centre
With Kernels, “small is beautiful”

- The **reliable operation** of any computer depends on its operating system, i.e., it’s kernel.
- More complex software has **higher chance of bugs**, security problems, vulnerability to worms and viruses.
- Linus Torvalds imposes a strict discipline on kernel developers to carefully restrict code that will increase size of kernel.
- Linux does not suffer from “kernel bloat”
  - Compare the size of the Windows 2000 “microkernel:” several megabytes, cannot be booted from floppy
  - Linux: small enough to fit on one floppy together with many useful tools: [http://www.toms.net/rb/](http://www.toms.net/rb/)
- **Movies:**
  - Linus discusses Monolithic, Microkernel design, ETU, avi, avi2

Movies: [http://www.toms.net/rb/](http://www.toms.net/rb/)
The OS manages the hard disks.

How can the system read the hard disk to start the OS?
- The process of starting the computer ready for use
- How does a computer boot?
- Involves:
  - BIOS (“basic input output system”) finding the *boot loader*
  - The boot loader starting the kernel
  - For Linux:
    - The kernel starting `init`
  - `init` starting everything else
A **boot loader** is a piece of software that runs before any operating system, and is responsible for loading an operating system kernel, and transferring control to it.

Microsoft OS provides a boot loader that starts their OS from the first active primary partition.

We use the grub (Grand Unified Boot Loader) boot loader that can start any operating system from almost any hard disk, floppy or network.
The boot process for a PC

- The BIOS performs a power on self-test (POST)
- The BIOS initialises PCI (Peripheral Component Interconnect) devices
- The bootloader loads the first part of the kernel into system RAM
- The kernel identifies and initialises the hardware in the computer
- The kernel changes the CPU to protected mode
- Init starts and reads the file /etc/inittab
- The system executes the script /etc/rc.d/rc.sysinit
- The system executes scripts in /etc/rc.d/init.d to start services (daemons)
Before the bootloader: The BIOS

- The BIOS runs in *real* mode (like old 8086)
- BIOS tests hardware with basic Power On Self Test (POST)
- BIOS then initialises the hardware.
- Very important for the PCI devices, to ensure no conflicts with interrupts.
- See a list of PCI devices.
- BIOS settings determine order of boot devices; when finds one, loads first sector into RAM, starts executing that code.
<table>
<thead>
<tr>
<th>Operating System: Kernel and boot process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role of OS</td>
</tr>
<tr>
<td>System Calls</td>
</tr>
<tr>
<td>Types of OS</td>
</tr>
<tr>
<td>Monolithic Kernel</td>
</tr>
<tr>
<td>Layered Kernel</td>
</tr>
<tr>
<td>Microkernel</td>
</tr>
<tr>
<td>Virtual machine</td>
</tr>
<tr>
<td>Boot Process</td>
</tr>
<tr>
<td>Booting an Operating System</td>
</tr>
<tr>
<td>Booting a PC</td>
</tr>
<tr>
<td>Boot Loader</td>
</tr>
<tr>
<td>Boot Process</td>
</tr>
<tr>
<td>BIOS</td>
</tr>
<tr>
<td>VMWare Boot Screen</td>
</tr>
<tr>
<td>Boot Loaders: what they do</td>
</tr>
<tr>
<td>The kernel is loaded</td>
</tr>
<tr>
<td>Real and Protected mode init</td>
</tr>
<tr>
<td>Runlevels</td>
</tr>
<tr>
<td>References</td>
</tr>
</tbody>
</table>
Boot Loaders: what they do

- Syslinux is the simplest, grub has the most features, LILO in between

- Grub provides many interactive commands that allow:
  - Reading many different file systems
  - Interactively choosing what to boot
  - Many, many more things (do pinfo grub)
  - All before any operating system started!!

- Grub and LILO let you choose what OS to boot
The kernel is loaded

- Boot loader reads first part of the kernel into RAM, executes the code
- This initial kernel code loads the rest of the kernel into RAM
- The kernel checks the hardware again
- The kernel switches from *real* mode to *protected* mode
- **Real mode** exists for booting, and so that can run old DOS programs
- Uses only bottom 16 bits of registers
- Can only access the bottom 1 MB RAM
- BIOS only supports real mode
- **Protected mode** uses all 32 bits of address registers
- Allows access to all RAM
- Allows use of memory management unit
- Normal mode of operation for modern OSes on Intel platform.  
- Cannot call BIOS functions in protected mode
The kernel then starts the **first process**, process 1: `/sbin/init`

- `/sbin/init` reads the `/etc/inittab`
- Init starts reading the script `/etc/rc.d/rc.sysinit`
- `/etc/inittab` tells `init` to do this
- `init` then executes scripts in `/etc/rc.d/init.d` to start services
A standard Linux system has 7 modes called runlevels:

- **0**: halt (shut down the machine)
- **1**: single user mode
- **2**: multiuser with no network services
- **3**: full multiuser mode
- **4**: can be customised; default same as 3
- **5**: multiuser with graphical login
- **6**: reboot
directories for each runlevel

- If you look in /etc/rc.d, you see one directory for each runlevel, and a directory called init.d:
  
  ```
  $ ls /etc/rc.d
  init.d  rc0.d  rc2.d  rc4.d  rc6.d  rc.sysinit
  rc       rc1.d  rc3.d  rc5.d  rc.local
  ```

- init.d contains one script for each service. You execute these scripts with the service command, i.e.,

  ```
  $ sudo service autofs start
  ```
Runlevel directories

- Each of /etc/rc.d/rc[0-6].d contains *symbolic links* to scripts in /etc/rc.d/init.d
  - A symbolic link is a bit like a shortcut in Windows (but more fundamental)
  - We cover symbolic links in detail later
- If name of link begins with **K**, the script will stop (**kill**) the service
- If name of link begins with **S**, will start the service
- The *chkconfig* program creates these symbolic links

References

- Operating System: Kernel and boot process
- Role of OS
- System Calls
- Types of OS
- Monolithic Kernel
- Layered Kernel
- Microkernel
- Virtual machine
- Boot Process
- Runlevels
- Directories for each runlevel
- Example of service: *yum*
  - Turning yum off
  - Turning yum on
Example of service: `yum`

- In the laboratory, you set up the `yum` service to automatically install software updates.
- You used the `chkconfig` program to enable the service.
  - For a complete manual on `chkconfig`, type:
    `$ man chkconfig`
  - For a brief summary of options, type:
    `$ /sbin/chkconfig --help`
- Here we use the program `find` (covered in detail later) to see the links before and after.
Turning \texttt{yum} Service Off

\begin{itemize}
  \item $\texttt{sudo /sbin/chkconfig yum off}$
  \item $\texttt{/sbin/chkconfig yum --list}$
  \begin{verbatim}
  yum 0:off 1:off 2:off 3:off 4:off 5:off 6:off
  \end{verbatim}
  \item $\texttt{find /etc/rc.d -name ‘*yum’}$
  \begin{verbatim}
  /etc/rc.d/init.d/yum
  /etc/rc.d/rc0.d/K01yum
  /etc/rc.d/rc1.d/K01yum
  /etc/rc.d/rc2.d/K01yum
  /etc/rc.d/rc3.d/K01yum
  /etc/rc.d/rc4.d/K01yum
  /etc/rc.d/rc5.d/K01yum
  /etc/rc.d/rc6.d/K01yum
  \end{verbatim}
  \item After turning the service off, all the links start with ‘\texttt{K}’ in all runlevels: 0, 1, 2, 3, 4, 5 and 6.
\end{itemize}
$ sudo /sbin/chkconfig yum on
$ /sbin/chkconfig yum --list
yum 0:off 1:off 2:on 3:on 4:on 5:on 6:off
$ find /etc/rc.d -name ‘*yum’
/etc/rc.d/init.d/yum
/etc/rc.d/rc0.d/K01yum
/etc/rc.d/rc1.d/K01yum
/etc/rc.d/rc2.d/S50yum
/etc/rc.d/rc3.d/S50yum
/etc/rc.d/rc4.d/S50yum
/etc/rc.d/rc5.d/S50yum
/etc/rc.d/rc6.d/K01yum

Notice that after turning the service on, there are links that start with ‘S’ in runlevels 2, 3, 4 and 5.
References