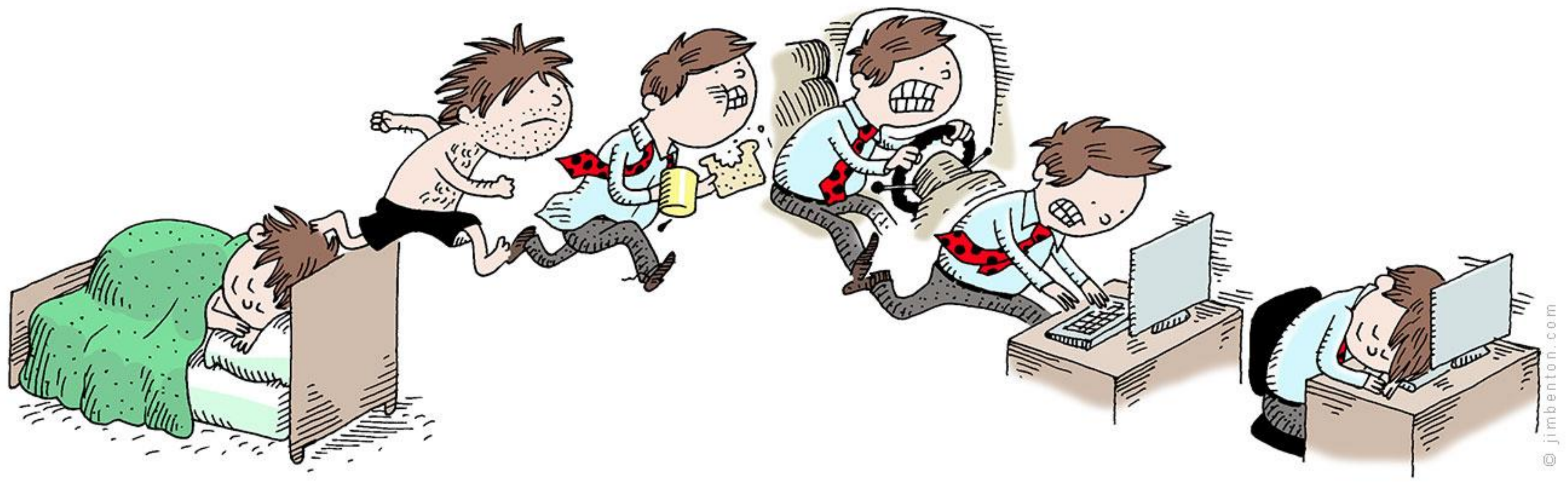
Good Afternoon All



Operating Systems in practice – Linux

SOUNDARARAJAN DHAKSHINAMOORTHY

SOUNDARARAJAN@OUTLOOK.COM

Today !

Windows 10
29th July 2015

Today in History
J.R.D Tata (birth anniversary)

More than ever before, we must be ready to think every problem afresh, to change and innovate…

Agenda

- Introduction
- Booting
- Kernel
- Hardware abstraction
- Process
- Virtual Memory
- Q & A

Booting

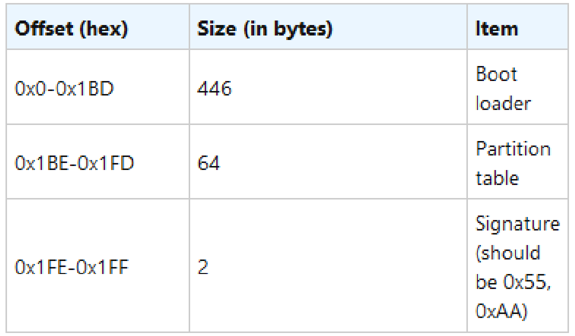
- When you power on the system, who initializes all the hardware ?
- What steps should be taken to initialize the hardware ?
- You remove the keyboard from your PC and who finds that ?
- How does the system date and time work, when I switch off the PC ?
- Where does operating system comes into picture ?
- how does the computer find and boot into an operating system ?

Booting

- On power on a “special program” will be loaded from a special ROM in motherboard. Remember it’s not in the hard disk.
- This program knows the details of RAM, Processor, USB ports, SMPS, Keyboard etc., and will run a test on all of them to see if booting can be continued.
- It will initiate then start the process of loading the operating system.

Booting….

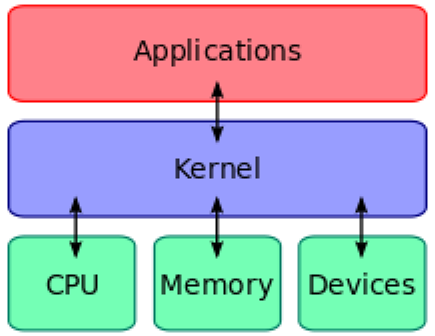
- Now the program finds the primary hard disk and runs the code in first 512 bytes of the harddisk a.k.a **MBR (Master Boot Record)**
- The MBR looks through all the partitions in the hard disk and finds the boot partition (a partition marked as boot) show `parted` command
- It then executes the first 512 bytes in the corresponding partition a.k.a boot sector. The 0th byte of that partition should be a valid instruction. This will load the operating system kernel ☺



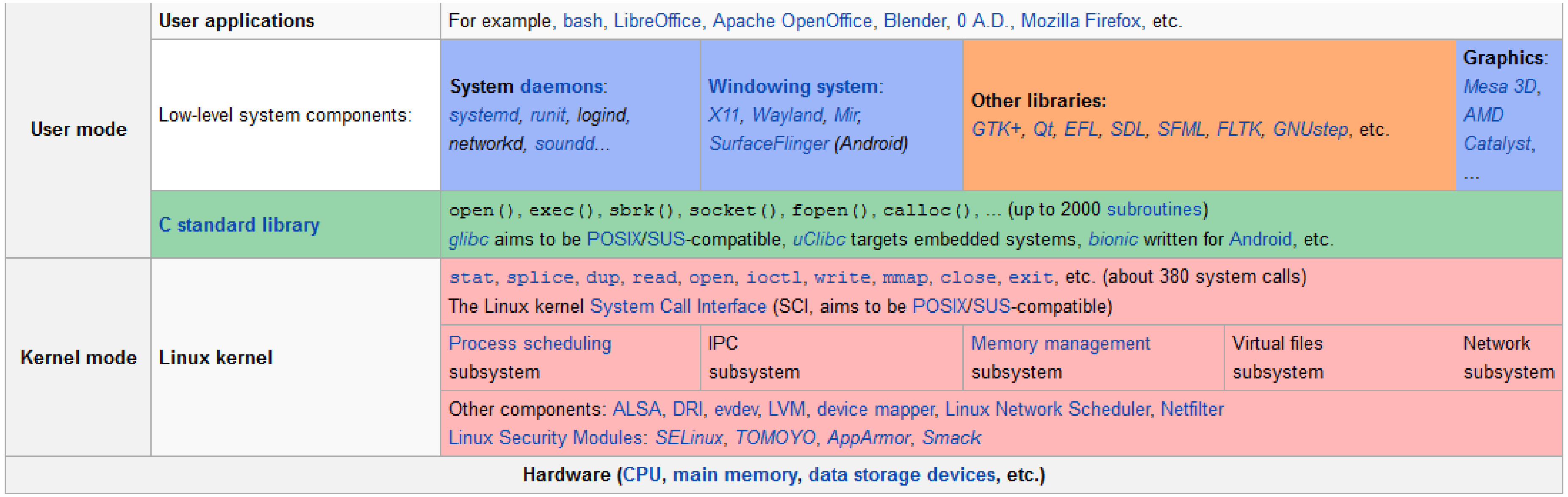
Offset (hex)	**Size (in bytes)**	**Item**
0x0-0x1BD	446	Boot loader
0x1BE-0x1FD	64	Partition table
0x1FE-0x1FF	2	Signature (should be 0x55, 0xAA)

Kernel

- The kernel is a computer program that
  - manages I/O (input/output) requests from software,
  - translates them into data processing instructions for the central processing unit and other electronic components of a computer.
  - Communicates with hardware.
- The kernel is usually loaded into the protected area of memory, which prevents it from being overwritten. [Discuss how this is supported in hardware]



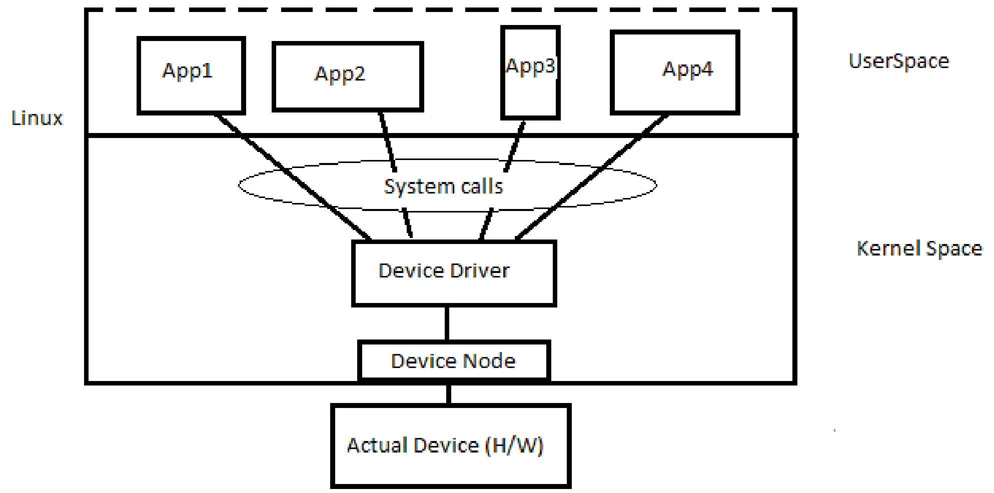
A practical view (linux kernel)



User mode	**User applications**	For example, bash, LibreOffice, Apache OpenOffice, Blender, 0 A.D., Mozilla Firefox, etc.				
	Low-level system components:	**System daemons:** *systemd, runit, logind, networkd, soundd...*	**Windowing system:** *X11, Wayland, Mir, SurfaceFlinger (Android)*	**Other libraries:** *GTK+, Qt, EFL, SDL, SFML, FLTK, GNUstep*, etc.		**Graphics:** *Mesa 3D, AMD Catalyst*, ...
	C standard library	`open()`, `exec()`, `sbrk()`, `socket()`, `fopen()`, `calloc()`, ... (up to 2000 subroutines) *glibc* aims to be POSIX/SUS-compatible, *uClibc* targets embedded systems, *bionic* written for Android, etc.				
Kernel mode	**Linux kernel**	`stat`, `splice`, `dup`, `read`, `open`, `ioctl`, `write`, `mmap`, `close`, `exit`, etc. (about 380 system calls) The Linux kernel System Call Interface (SCI, aims to be POSIX/SUS-compatible)				
		Process scheduling subsystem	IPC subsystem	Memory management subsystem	Virtual files subsystem	Network subsystem
		Other components: ALSA, DRI, evdev, LVM, device mapper, Linux Network Scheduler, Netfilter Linux Security Modules: *SELinux, TOMOYO, AppArmor, Smack*				
Hardware (CPU, main memory, data storage devices, etc.)						

Hardware abstraction

- Under Linux and UNIX each and every hardware device treated as a file. (show /dev)
- A device file allows to accesses hardware devices
  - Character files (1 byte at a time) (cat /proc/devices, ls –l)
  - Block files (512 bytes to 32 KB)



Hardware abstraction

- cat /proc/iomem
  - Why do all the address starts with ‘f’ ?
- All devices are IO memory mapped, meaning you can write to or read from any peripheral device using an address.
- The operating system loads the correct device driver and makes sure the data is properly interpreted and fed to the applications and vice versa.
- lsmod all devices
- Show dmesg on system initialization sequence.

A simple device driver

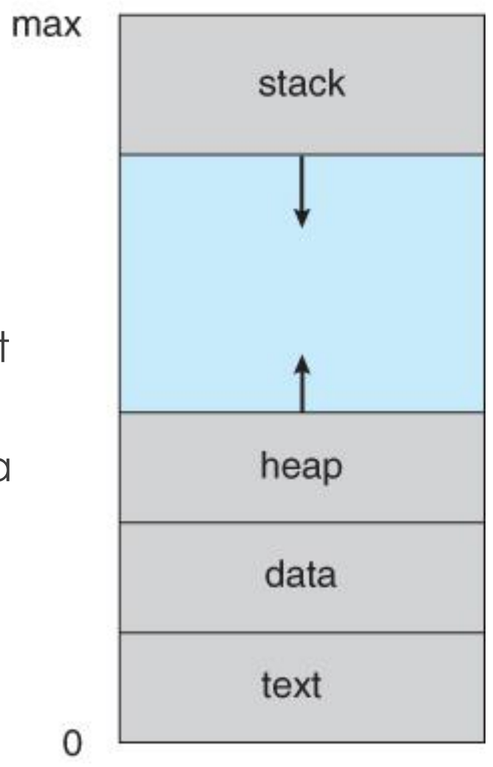
- /home/minion/kinetic/driver
  - The code (Hello.c)
  - Building the driver (Build.sh)
  - Registering a kernel mode driver module (Insmod hello.ko)
  - View whether it's loaded into kernel (dmesg (or) cat /var /log/syslog)
  - Interrupt handler , show that a key is pressed
  - Let them understand what a driver is capable of.
  - Where is the driver running ? (ps –A)

Processes

- A process is a special file (Agree ?)
- What is a difference between a text file and a exe file
- Write a small executable and run it.
- Objdump –s the executable.
- Objdump –s the text file.
- Binary vs content, valid instructions vs ascii data.

A detailed look into the process

- Objdump –h
  - The text section comprises the compiled program code,
  - The data section stores global and static variables, allocated and initialized prior to executing main.
  - The heap is used for dynamic memory allocation, and is managed via calls t new, delete, malloc, free, etc.
  - The stack is used for local variables. Note that the stack and the heap start a opposite ends of the process's free space and grow towards each other.
  - When processes are swapped out of memory and later restored, additional information must also be stored and restored. Key among them are the program counter and the value of all program registers.



A detailed look into process

- Ps –A (when the process is running), show that process when process finished. Foobar.c
- Top (show the processor usage of a process when it's running in full speed, full) fullcpu.c
- Top (show the processor usage when we insert sleep cycles, and change in delay, cpudelay.c)
- Htop and multiple instances of a process and how it works cpudelay.c multiple instances.
- Htop and multiple instances of a process > number of cores (cpudelay.c)

Memory management

- Why does memory management always speaks about virtual memory.
- What is the connection between physical memory and virtual memory
- Why is it called virtual ?
- Let's inspect a program to understand this (basic.c)
  - The address of a variable is the same all the time ??? Holy Cow
- Address of local variables and global variables ?? Nuts!!
  - (lessbasic.c) print address of local, global variables and main function

Explore the memory more

- Verylessbasic.c
  - Compile the program and run objdump
  - See the bss section, data section
- Break on a program, and explore the stack from /proc/<pid>/stack

- Add memory pressure and see what happens with htop (pressure.c)
  - Seg fault with a big allocation
- Also see the stress command
- Physical address
  - Use ps to find process id and explore /proc/<pid>/maps
  - They are not the same every time.

Virtual memory stats in general

- /proc/vmstat
  - Nr_page_table_tables
  - Nr_pagein
  - Nr_pageout
  - Pswpin
  - Pswpout
  - pgfault

That’s all about it

Q & A