The Cloud Operating System

Some slides have been adopted from IBM Linux Study Guide under IBM Academic Relationship program

AI and the Operating System

Setting the Context

A Must read:

https://towardsdatascience.com/ai-and-the-operating-system-4282edd3a930

Marriage between AI and OS

https://becominghuman.ai/the-concept-of-i-ai-and-the-operating-system-part-2-672752f1fa3

What makes an AI OS?

- An OS is a huge list of programmed system calls. Bring AI to interact with such system calls and you have an AI OS.
- An OS is somewhat a deterministic system, AI needs to be trained so AI can help personalize the OS
- An OS is designed for general purpose work suited for multiple personas and there is NO such thing called AGI or Artificial General Intelligence exist today

What makes a good cloud operating system?

- Fast boot up time
- Consumes less resources
- Provides containers or virtual machines
- Auto deployment and update
- Runs workload with excellent performance
- Good networking support
- Good storage support (block, network, endurance)
- Secure

Why Linux adoption is growing?

According to IDC, these 5 factors are fueling the growth:

- Cloud spending
- Large player adoption
  - https://azure.microsoft.com/en-us/services/azure-sphere/
  - https://clearlinux.org/
  - https://www.clearos.com/
  - https://www.redhat.com/en
- Internet of Things (IoT)
  - Gartner claims that there were 8.4 billion connected things in 2017 with expectations to rise to 20.4 billion in 2020. Nearly all of these “connected things” will be running a Linux kernel.
- Security concerns
- Cost sensitivity

Choices of cloud OS

- CoreOS - A lightweight Linux operating system designed for clustered deployments providing automation, security, and scalability for your most critical applications (In RedHat now)
- Project Atomic – Deploy and manage your docker containers http://www.projectatomic.io/ (also part of RedHat)
- OSv
  - Lockless single memory space collaborativehttp://osv.io/ memory management (http://osv.io/)
- CentOS https://www.centos.org/
- Ubuntu for cloud http://www.ubuntu.com/cloud

Docker Ecosystem

- Distributed runtime with REST API for Linux containers (LXC)
- Linux containers virtualize the host kernel
- Thinner virtualization than hypervisors
- Completely integrated with Linux
- Docker container inherits the performance of host kernel

https://resources.sei.cmu.edu/asset_files/Presentation/2017_017_001_497378.pdf

https://docker-curriculum.com/

Serverless Computing

Serverless computing is a cloud-**computing** execution model in which the cloud provider runs the server, and dynamically manages the allocation of machine resources.

The difference between traditional cloud computing and serverless is that you, the customer who requires the computing, doesn't pay for underutilized resources.

– Instead of spinning up a server in AWS for example, you're just spinning up some code execution time.

– The serverless computing service takes your functions as input, performs logic, returns your output, and then shuts down. You are only billed for the resources used during the execution of those functions.

Major vendors initiative:

- https://cloud.google.com/functions/
- https://cloud.ibm.com/functions/
- https://azure.microsoft.com/en-us/services/functions/
- https://cloud.google.com/knative/
- https://aws.amazon.com/serverless/

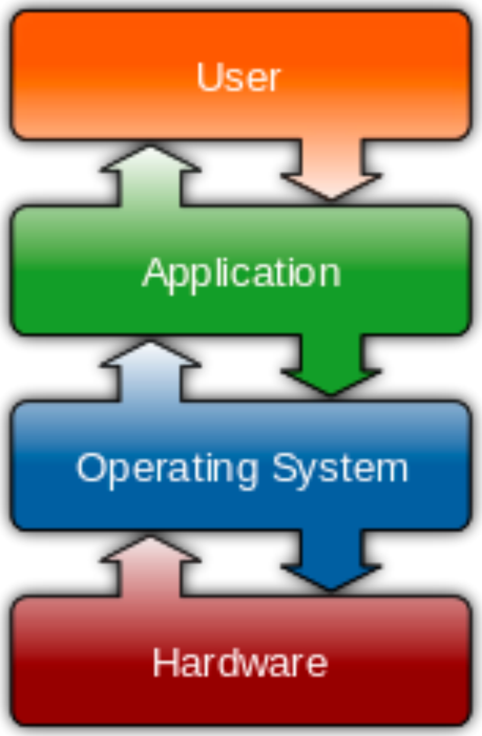
Top Trends in Virtualization

- Multi-cloud deployment and management of workload
- Elastic AI infrastructure (GPU virtualization)
- Virtual Remote attached GPUs FPGAs etc..
- Improving Data Center efficiency using AI

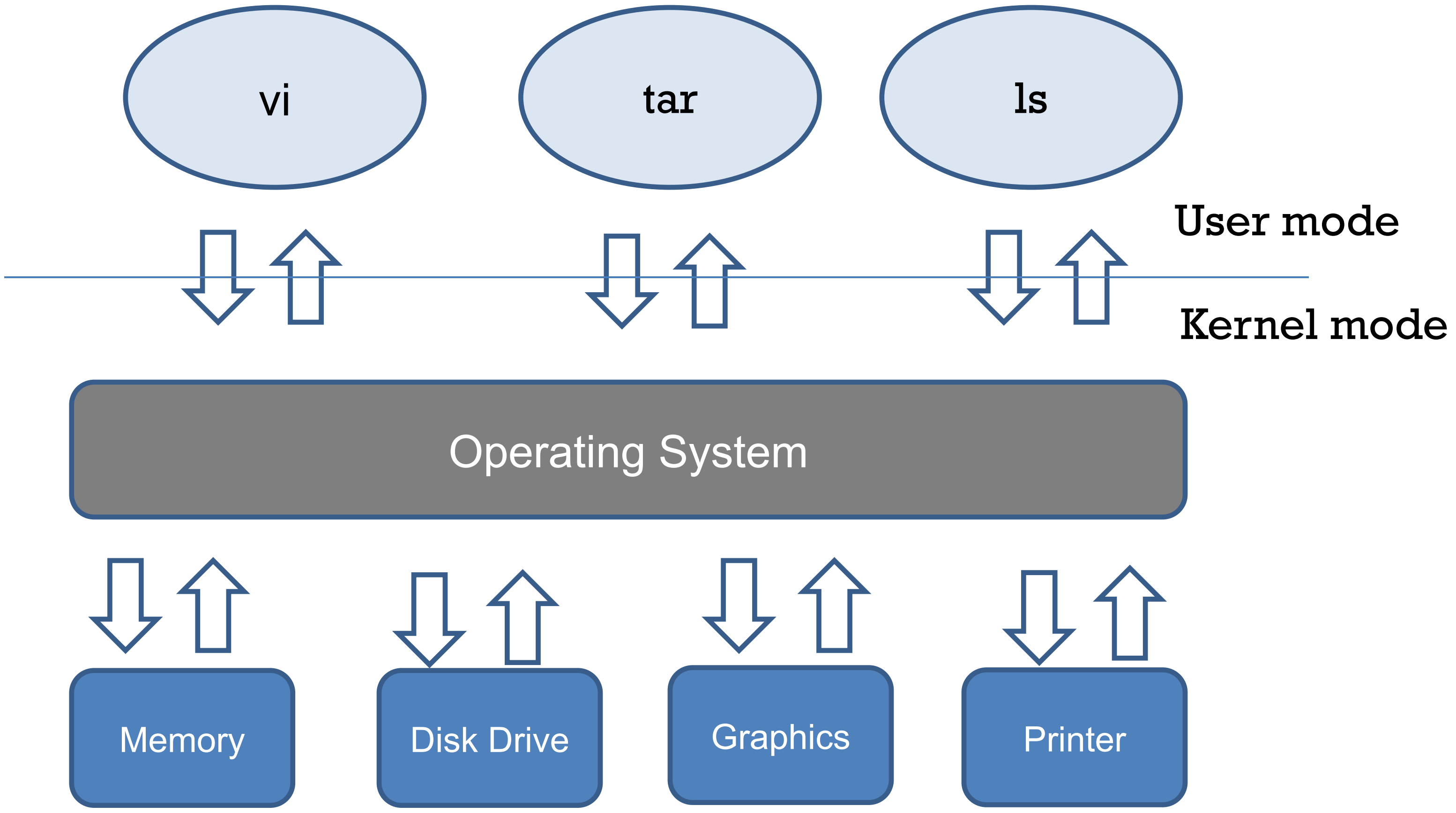
- https://bitfusion.io/
- https://docs.google.com/a/google.com/viewer?url=www.google.com/about/datacenters/efficiency/internal/assets/machine-learning-applicationsfor-datacenter-optimization-finalv2.pdf

The operating system

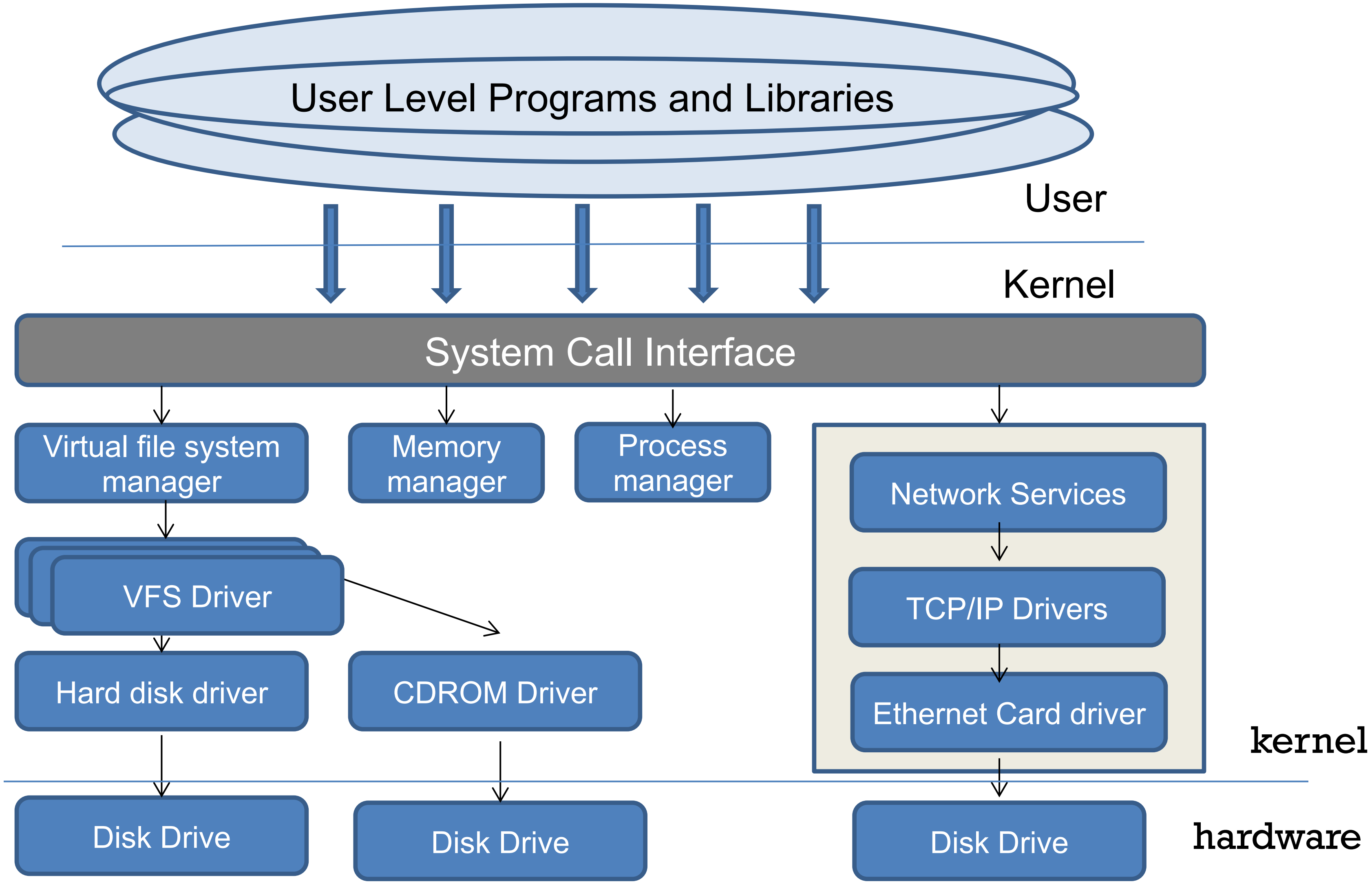
- Abstracts the real hardware of the system and presents the system's users and its applications with a virtual machine.
- Handles I/O to and from attached hardware devices
- Manages file I/O
- Supports file systems
- Manages all other programs running on the computer
  - Determines which applications should run
  - In what order
  - Determines the time slice(CPU time is divided into "slices") of a processes
- Manages sharing of the internal memory among multiple applications



Detailed version of OS



Linux Kernel Subsystem



Kernel Subsystem

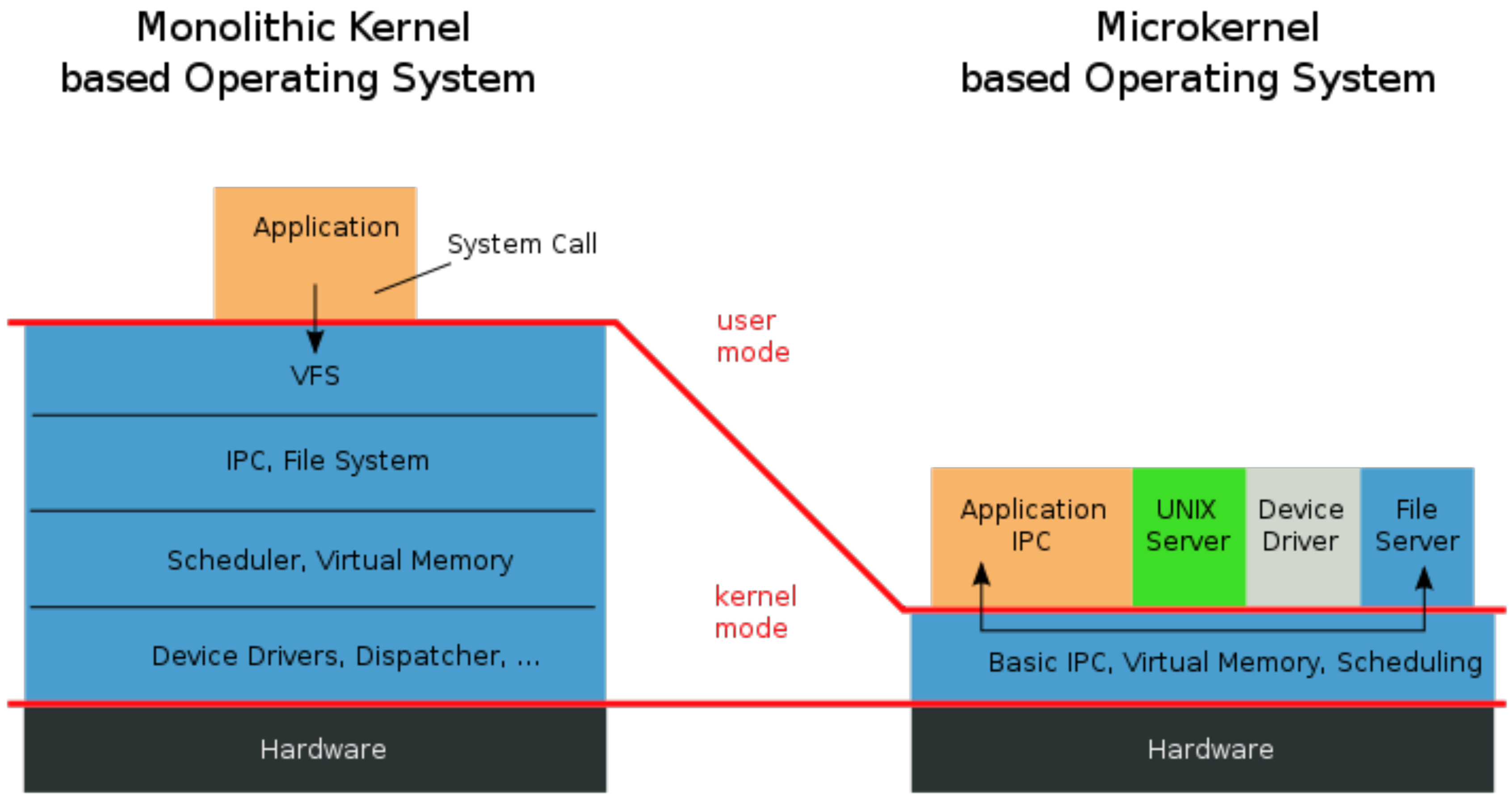
- Normal programs execute in user mode
- Kernel programs execute in kernel mode (privileged mode)
- Kernel takes requests from applications running in user mode and perform requested operations on the hardware.
- Linux kernel consists of several important parts: process management, memory management, hardware device drivers, file system drivers, network management and various other bits and pieces.

- Optional Reading:
- http://tldp.org/HOWTO/KernelAnalysis-HOWTO-3.html

Kernel Types

- Monolithic kernel
  - All OS related code are stuffed into a single module
  - Available as a single file
  - Functions fast
- Micro kernel
  - OS components are isolated and run in their own address space
  - Device drivers, programs and system services run outside kernel memory space. Only a few functions such as process scheduling, and interprocess communication are included in the microkernel
  - Supports modularity

Kernel Types



Kernel mode (privileged mode)

- Kernel mode also referred as system mode
- Designed for restriction/protection from unauthorized user application program
- When the CPU is in kernel mode, it is assumed to be executing "trusted" software and thus it can execute any instructions and reference any memory address.

- Thus all user mode software must request use of the kernel by means of system calls in order to perform privileged instructions, such as process creations or input/output operations

- Ex: device drivers

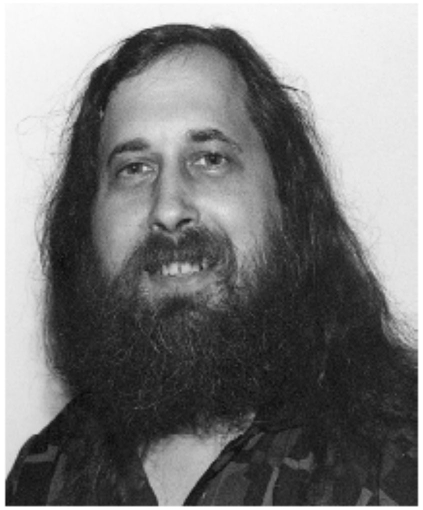
User mode (un-privileged mode)

- User mode is normal mode for operating programs
  - Web browsers, calculator
- Code running in user mode must delegate to system APIs to access hardware or memories
- Most of the code running on your computer will run in user mode

Question: why did we come up with two different modes?

Short history of Linux

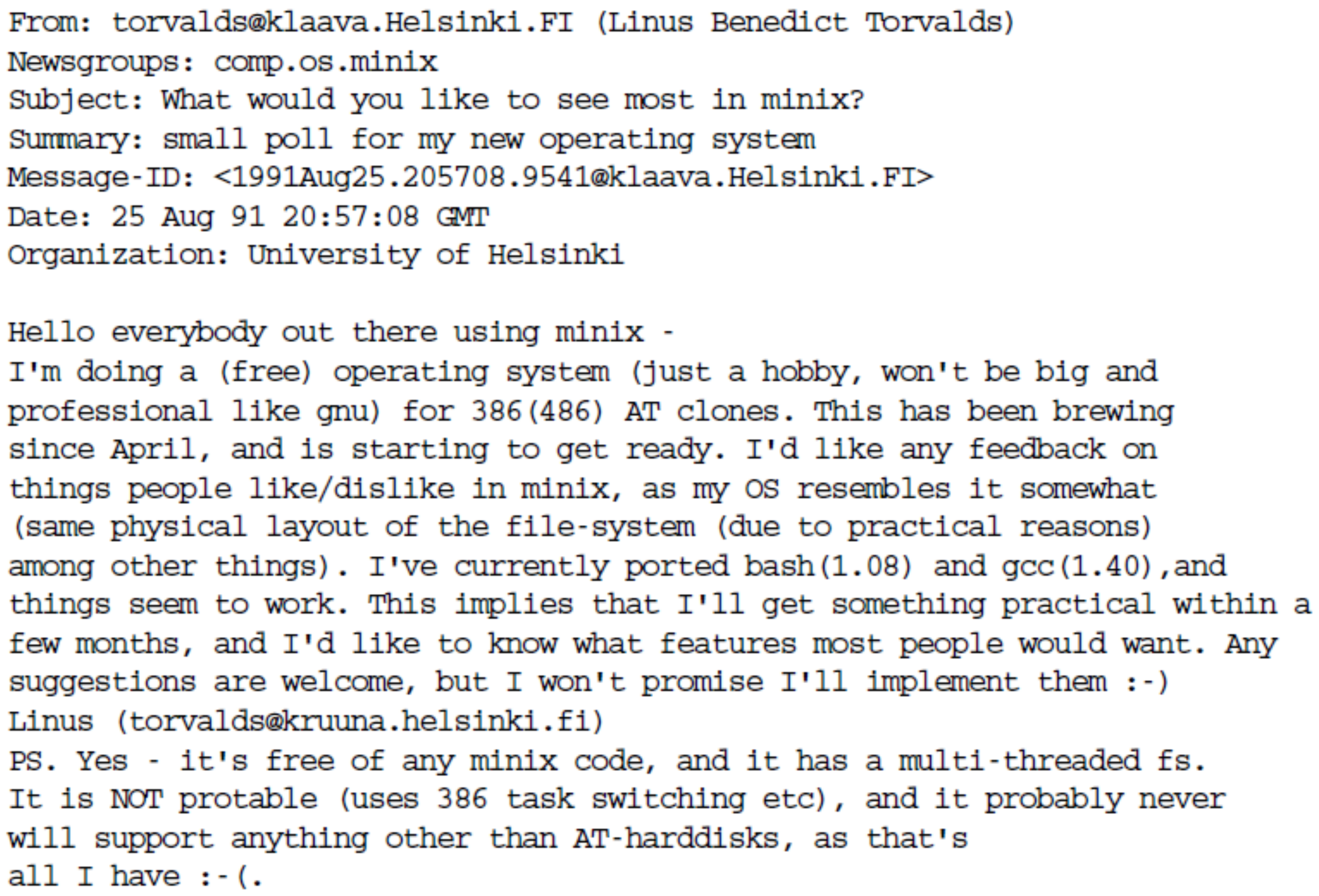
- 1984: Richard Stallman starts GNU project
  - GNU's not UNIX
  - http://www.gnu.org
- Purpose: Free UNIX
  - "Free as in Free Speech, not Free Beer"
- First step: re-implementation of UNIX Utilities
  - C compiler, C library
  - emacs
  - bash
- To fund the GNU project, the Free Software Foundation is founded
  - http://www.fsf.org



Short history of Linux (2)

- 1991: Linus Torvalds writes first version of Linux kernel.
  - Initially, a research project about the 386 protected mode
  - Linus' UNIX -> Linux
  - Combined with the GNU and other tools to form a complete UNIX system
- 1992: First distributions emerge.
  - Linux kernel
  - GNU and other tools
  - Installation procedure
- The rest is history.

Short history of Linux (3)



```
From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Message-ID: <1991Aug25.205708.9541@klaava.Helsinki.FI>
Date: 25 Aug 91 20:57:08 GMT
Organization: University of Helsinki

Hello everybody out there using minix -
I'm doing a (free) operating system (just a hobby, won't be big and
professional like gnu) for 386(486) AT clones. This has been brewing
since April, and is starting to get ready. I'd like any feedback on
things people like/dislike in minix, as my OS resembles it somewhat
(same physical layout of the file-system (due to practical reasons)
among other things). I've currently ported bash(1.08) and gcc(1.40),and
things seem to work. This implies that I'll get something practical within a
few months, and I'd like to know what features most people would want. Any
suggestions are welcome, but I won't promise I'll implement them :-)
Linus (torvalds@kruuna.helsinki.fi)
PS. Yes - it's free of any minix code, and it has a multi-threaded fs.
It is NOT protable (uses 386 task switching etc), and it probably never
will support anything other than AT-harddisks, as that's
all I have :-(.
```

Linux source in browser

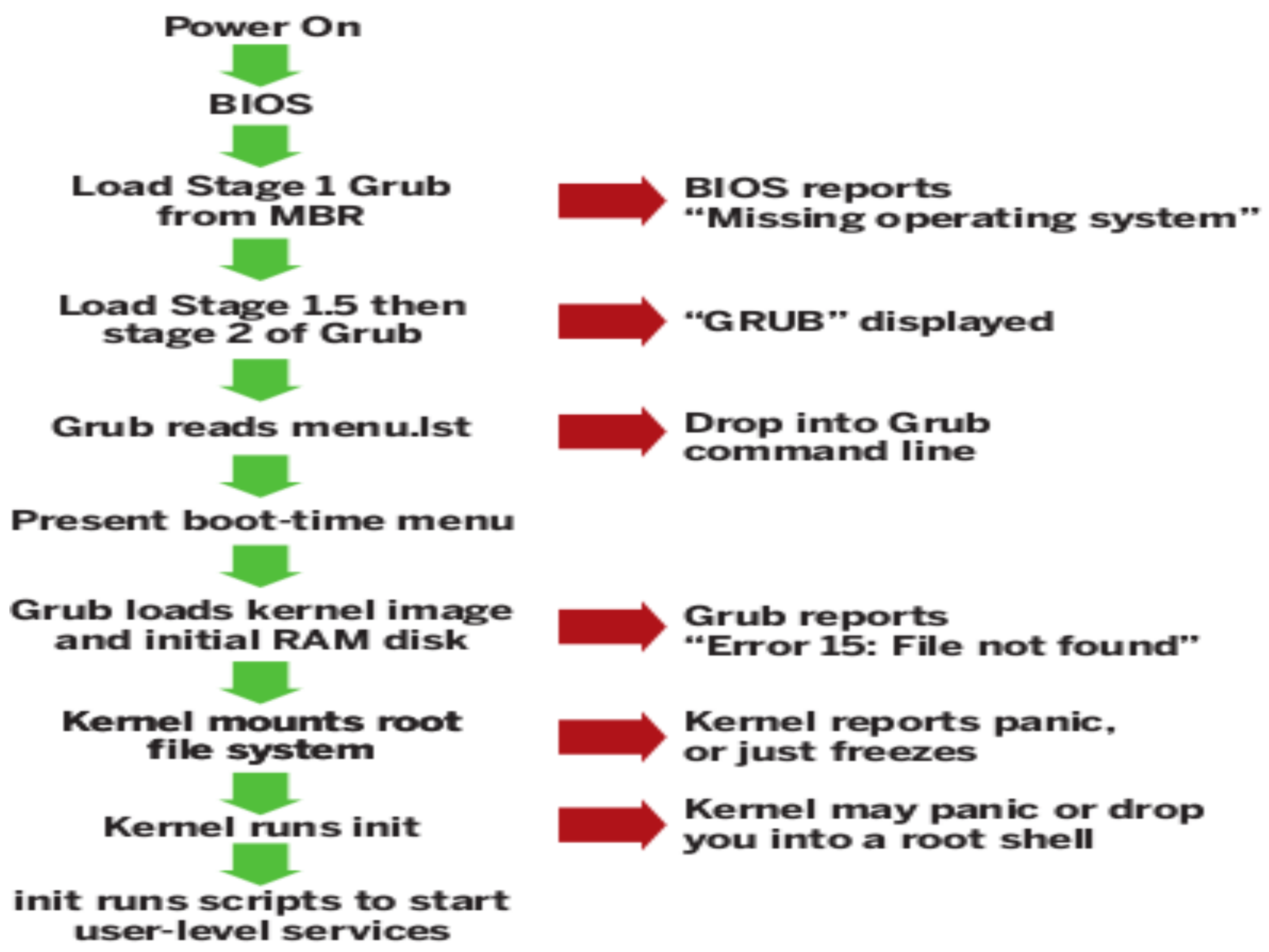
Linux Cross Reference
Free Electrons
Embedded Linux Experts
• Source Navigation • Identifier Search • Freetext Search •
Version: 2.0.40 2.2.26 2.4.37 3.0 3.1 3.2 3.3 3.4 3.5 3.6 3.7 3.8 3.9 3.10 3.11 3.12 3.13 3.14 3.15 3.16

http://lxr.free-electrons.com/

Linux Boot Process

1. CPU Initialization
2. Execute single instruction and jump to BIOS (ROM)
3. BIOS finds boot device and fetches MBR(on the hard disk/CDROM/or any registered devices)
4. MBR points to LILO(Linux Loader/boot manager, similar to windows boot.ini)
5. LILO loads compressed Linux kernel
6. Compressed kernel is decompressed and loaded
7. Root file system mounted
8. Init(mother of all processes) process started
9. Rest is all forking of new processes (kernel remains uncompressed in protected mode)

Linux Boot Process


Power On
BIOS
Load Stage 1 Grub from MBR
BIOS reports "Missing operating system"
Load Stage 1.5 then stage 2 of Grub
"GRUB" displayed
Grub reads menu.lst
Drop into Grub command line
Present boot-time menu
Grub loads kernel image and initial RAM disk
Grub reports "Error 15: File not found"
Kernel mounts root file system
Kernel reports panic, or just freezes
Kernel runs init
Kernel may panic or drop you into a root shell
init runs scripts to start user-level services

Linux Process

- A program is an executable file.
- A process is a program that is being executed.
- Each process has its own environment:

process environment

Program name
Internal data
Open Files
Current Directory
additional parameters

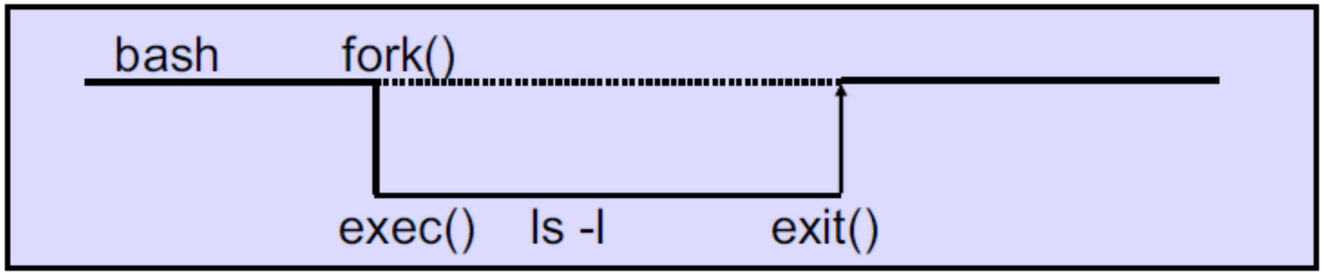
User and Group ID
Process ID (PID)
Parent PID (PPID)
Program variables

- To see the PID of your current shell process, type:
  $ echo $$

Starting and Stopping of a process

- All processes are started by other processes
  - Parent/Child relationship

$ ls –l



- A process can be terminated for two reasons:
  - The process terminates itself when done.
  - The process is terminated by a signal from another process.

Process Summary

- A program is an executable stored somewhere(perhaps on hard disk) in a computer. This program owns no resources and can not do anything until it gets executed.
- Once executed, a process is created which does own resources.
- Inside of this running process, threads can be used to do extra work for the process.
- Example:
  - A database interface application can use one thread to create a connection to the database, another thread to display the data in the UI and third thread to play music in the background(for fun)

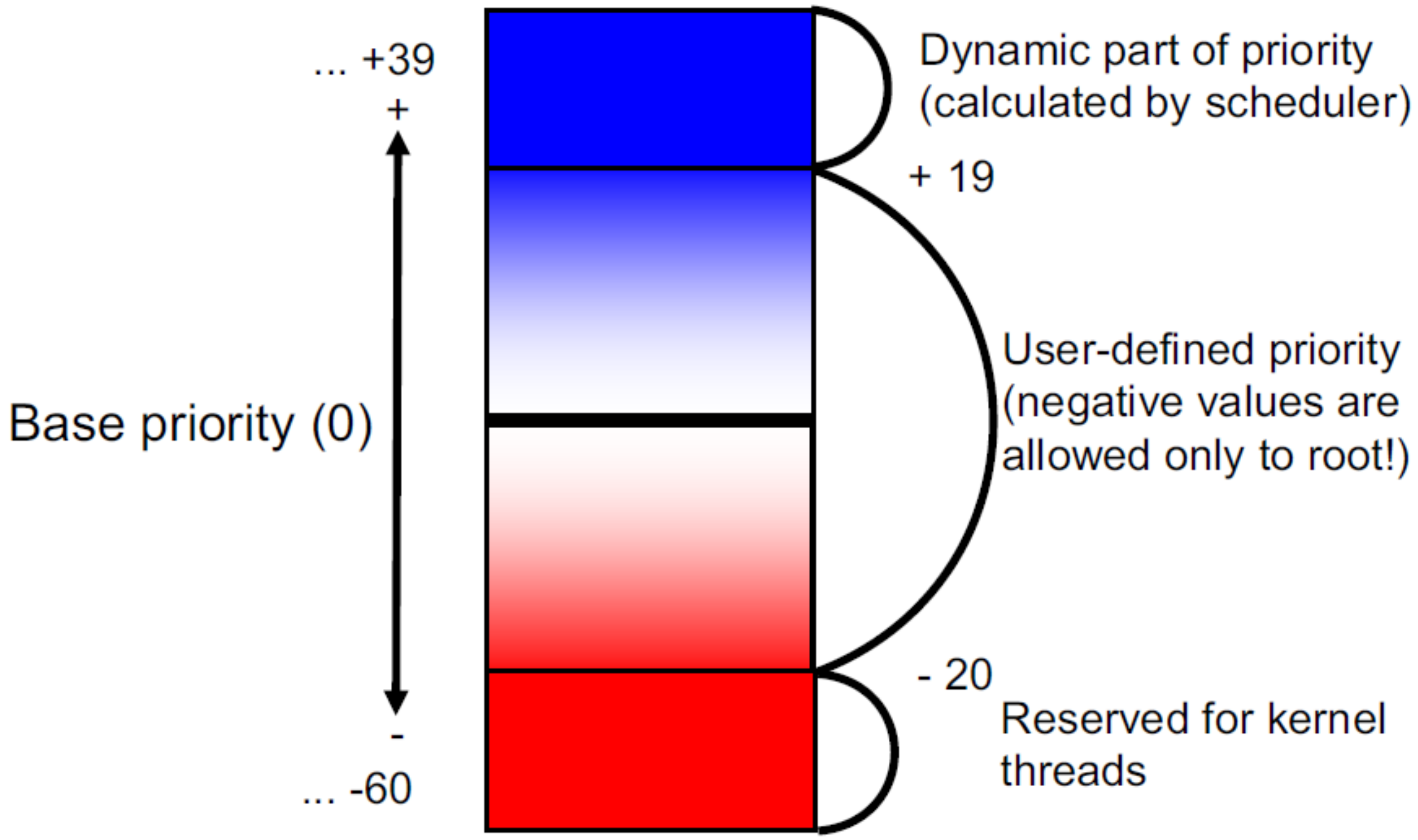
The Life cycle of a Process

- Linux process is created using fork() and terminated either using exit() or by receiving a signal. Implementation is in kernel/fork.c and kernel/exit.c respectively.
- Processes communicate with each other (IPC) and with the kernel
- Linux supports IPCs such as signals and pipes
- Processes are protected from one another – kernel gives them separate address space.
- New processes (specially child threads) are created by cloning a currently running process using clone()

- Optional readings:
- http://linux.die.net/man/2/clone
- http://oreilly.com/catalog/linuxkernel/chapter/ch10.html

Managing process priorities

- Processes are scheduled according to priority.


... +39
+
Base priority (0)
-
... -60
Dynamic part of priority (calculated by scheduler)
+ 19
User-defined priority (negative values are allowed only to root!)
- 20
Reserved for kernel threads

Time slice

Processes on a Linux system are scheduled according to priority:

- When the CPU is free to run a process, it looks through the process table for a process with the lowest *priority number*.
- This process then gets a timeslice on the CPU.
- The priority number of a process is continuously changed. Basically three factors influence this:
  1. After a process has had a certain amount of CPU time, its priority number is increased, meaning that next time the CPU becomes available, the process is less likely to be first in the list.
  2. After a process has been idle (not using CPU time) for a while (either because it is waiting for something to happen, or because other processes are keeping the CPU busy), the priority number is decreased.
  3. The priority number can never become lower than the *nice value* that was set for that process. This scheme results in a usage pattern where every process with the same nice value gets an equal amount of CPU time. Processes with a higher nice value get less CPU time than processes with a low nice value.

Linux Memory Management

- Makes system appear to have more RAM than it really has
- Shares RAM between competing processes as they need it
- Hard drive space used to extend physical memory
- Virtual memory provides:
  - Large address spaces (swapper, pagefile etc..)
  - Protection (process has its own virtual address space)
  - Memory mapping (image and data files are mapped into virtual address space)
  - Fair physical memory allocation
  - Shared virtual memory
  - Copy on write
  - A nice book on Memory Management in Linux(Optional Reading)

http://www.ecsl.cs.sunysb.edu/elibrary/linux/mm/mm.pdf

Copy on write

- Optimization strategy used by OS
- Every process uses the single shared copy of the same data until it is modified, at that time true private copy is created
- True benefit is that if a caller never modifies the data no private copy ever need to be created
- Linux OS uses demand paging that uses copy-on-write technique
  - A process acquires its page table from its parent process (during a folrk() with the entries mark as read only
  - If the process tries to write to that memory space and page is copy-on-write page then it is copied and marked as read-write

Paging vs. Swapping

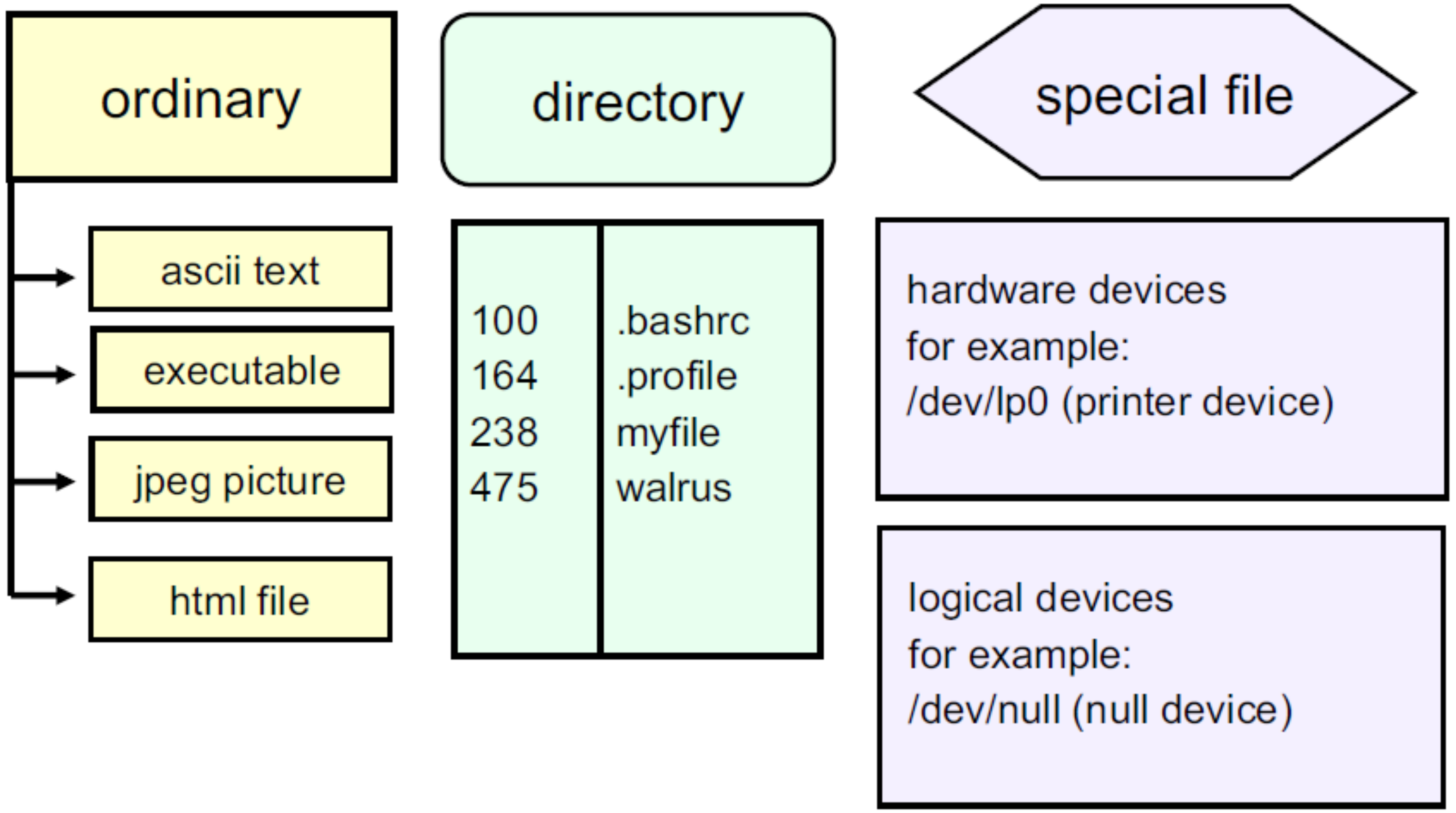
- Linux uses a technique called demand paging
- Memory pages of a process are brought into RAM only when a process attempts to use them
- Kernel alters the process's page table marking the virtual areas as existing but not in memory

- Linux uses a Least Recently Used(LRU) technique to swap a page
- More frequently accessed pages are younger
- Less frequently accessed pages are older
- Old pages are good candidates for swapping

File systems

- A file is a:
  - Collection of data
  - A stream of characters or byte stream
- No structure is imposed on a file by the operating system
- Linux supports many file system
- Ext2 is Linux's default file system and is widely used
- Implemented using a VFS(Virtual File system) layer
- Kernel sends a file system calls to VFS
- VFS redirects to FS
- Designated file system uses its own code and buffer cache functions to request I/O on physical devices

File Types


ordinary
ascii text
executable
jpeg picture
html file
directory
100 .bashrc
164 .profile
238 myfile
475 walrus
special file
hardware devices
for example:
/dev/lp0 (printer device)
logical devices
for example:
/dev/null (null device)

Displaying Binary files

- With the **od** command:

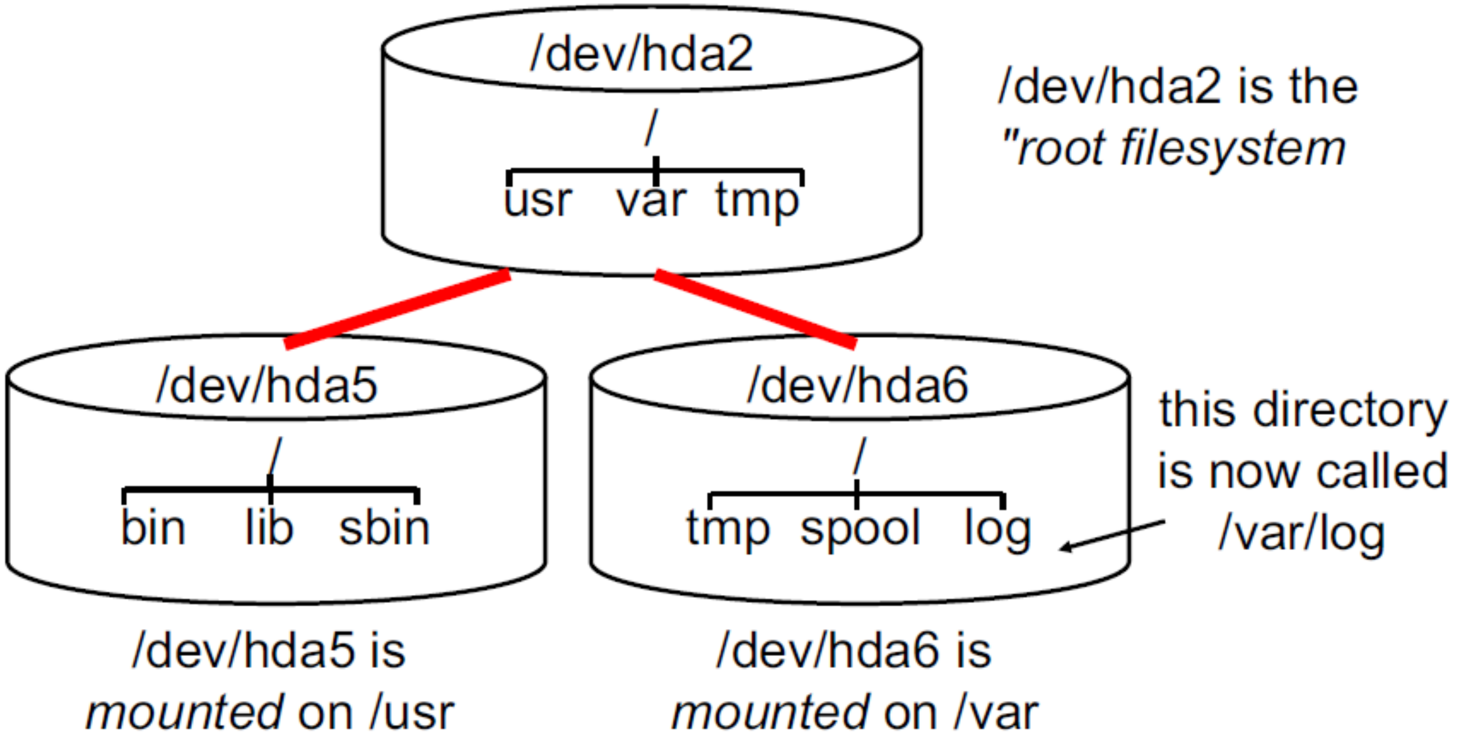
```
$ od /usr/bin/passwd
0000000 042577 043114 000401 000001 000000 000000 000000 000000
0000020 000002 000003 000001 000000 107300 004004 000064 000000
0000040 051430 000000 000000 000000 000064 000040 000006 000050
$
```

- With the **strings** command:

```
$ strings /usr/bin/passwd
/lib/ld.so.1
__gmon_start__
__deregister_frame_info
__register_frame_info
...
$
```

Virtual File systems

- Linux does not use drive letters (A:, C:, D:) to identify drives/partitions, but creates a virtual, unified filesystem.
- Different drivers/partitions are *mounted* on a *mountpoint.*



Interrupts

- A signal sent to the operating system from hardware
  - Hardware device (interrupt routines)
  - Software (system calls)
- Used to allow the hardware to communicate with OS
- Interrupt handling details are hardware and architecture specific
- Code for handling interrupts is located in:
  - /usr/src/linux/arch/i386/kernel/irq.c
  - /usr/src/linux/include/asm/irq.h
  - /usr/src/linux/include/linux/irq.h

IPC Mechanisms

- Classic IPC mechanisms
  - Signals (i.e.. SIGTERM)
  - Pipes
    - Basic I/O between 2 processes
    - Cane be **Named** or **Unnamed**
- Sockets
- System V IPC mechanism
  - Shared memory
  - Semaphores
    - Locking & Waiting
  - Message Queues
- http://tldp.org/LDP/lpg/node7.html

IPC: Signals

- Signals are most common IPC
- 0-31 standard signals and 32-64 real-time configurable
- Used to send asynchronous events to one or more processes
- Signals are like software interrupt and unidirectional
- Signals are used to exit processes, kill programs etc.
- Signal numbers are defined in the /usr/include/bits/signum.h file, and the source file is /usr/src/linux/kernel/signal.c.
- Signals are:
  - Delivered in system mode
  - Received in user mode
- Received signals can be:
  - Accepted
  - Ignored
  - Blocked
  - Handled (at later time)

Kill Signals

- Several signals can be sent to a process:
  - Using keyboard interrupts (if foreground process)
  - Using the **kill** command
  - Synopsis: **kill *-signal PID***
  - Using the **killall** command to kill all named apps
  - Synopsis: **killall *-signal application***
- Most important signals:

Signal	**Keyboard**	**Meaning**	**Default action**
01		hangup	end process
02	Ctrl-C	interrupt	end process
03	Ctrl-\	quit	end process and core dump
09		kill	end process - cannot be redefined - handled by kernel
15		terminate	end process

IPC: Unnamed vs. Named Pipes

- Unnamed pipes are used since early Unix days (1973)
- Facilitates bidirectional communication between two processes
- Processes must reside on same machine
- Processes must have same ancestor for pipes to work
- Unnamed pipes are temporary
- Command: pipe

- Named pipes are also called as FIFO
- Have a file name
- Can be used by any process as long as they have access
- Entities in a file system (not temporary object)
- Command: mkfifo
- Uses same data structure and operations as used by unnamed pipes

System V IPC

AT&T introduced these 3 forms of IPC in System V:

- Shared Memory
  - Allows one process to allocate memory that other processes can attach and detach
  - Process communicate directly through this shared memory(read/write)
  - Fastest IPC (no intermediation)
  - Information is directly mapped from a memory segment to address space of a calling process
- Message queues
  - Can be considered as an internal linked list within kernel's address space
  - Kernel keeps a linked list of messages, labeled by unique identifiers
  - Messages are added to the end of the queue
  - Messages can be removed from front, middle or anywhere like voice mail
- Semaphores
  - Implemented as a set
  - guarding access to a resource
  - Used for coordinating activities of processes
  - Binary semaphores (Just one key 0/1)
  - Counted semaphores (more than one key >=0)
  - http://tldp.org/LDP/lpg/node47.html#SECTION00743100000000000000

Sockets

- Sockets are extension of pipes
- Commonly used for client/server relationships
- Can be used as locally between 2 apps or remotely between apps on different machines
- TCP/IP apps communicate through sockets
- An application connects to a port number using socket (http, ftp, telnet)
- Port defines what type of server program will be used
  - http uses port 80
  - ftp uses 21