



Høyskolen  
Kristiania

# Digital technology

TK1104-1 22H

Lecturer: Toktam Ramezanifarkhani

Toktam.Ramezanifarkhani@kristiania.no

Toktamr@ifi.uio.no

# Operating System

## What are the different types?

**Mac OS** is a series of graphical user interface-based operating systems developed by Apple Inc. for their Macintosh



**Linux** is a Unix-like computer operating system assembled under the model of free and open source software development and distribution.



**Microsoft Windows** is a series of graphical interface operating systems developed, marketed, and sold by Microsoft.



**iOS** (previously **iPhone OS**) is a mobile operating system developed and distributed by Apple Inc. Originally unveiled in 2007 for the iPhone, it has been extended to support other Apple devices such as the iPod Touch



**Android** is a Linux-based operating system designed primarily for touchscreen mobile devices such as smartphones and tablet computers. Initially developed by Android, Inc.



**BSD/OS** had a reputation for reliability in server roles; the renowned Unix programmer and author W. Richard Stevens used it for his own personal web server for this reason.

# Today

- Focus on getting an overview
- Learn the **concepts** used in literature and documentation
- Exercises
  - Theory and concepts, as well as some history
  - Learn command-line interface on your own OS, how to maneuver and how to do simple tasks in command-line
  - Other things you need; opening archive files and installing software

# Booting

- Main memory (DRAM, SRAM) is volatile
  - Turn off the power, lose the contents
  - Turn on the power, memory is empty
- We need the OS to be loaded and running in memory to load and run programs
- If you turn your computer on, how do we get the OS loaded into memory and running?
  - We need a process – booting
  - The boot process (or portions of it) are stored in ROM (non-volatile memory)

# The Boot & Initialization Process

- CPU initializes itself (registers, control signals)
- BIOS performs power on self test
- Disk controllers tested
- BIOS determines where the OS is stored
- Boot loader runs to load OS (if multiple OSs available, user might be able to select)
- OS kernel loaded and initialized
  - Kernel often stored on hard disk (possibly also available on optical disk, flash drive or over network)
- OS runs initialization scripts
  - Establish runlevel (Linux levels from 0 to 6, windows has safe mode, safe mode with network, user mode, etc)
- OS running, computer ready for user

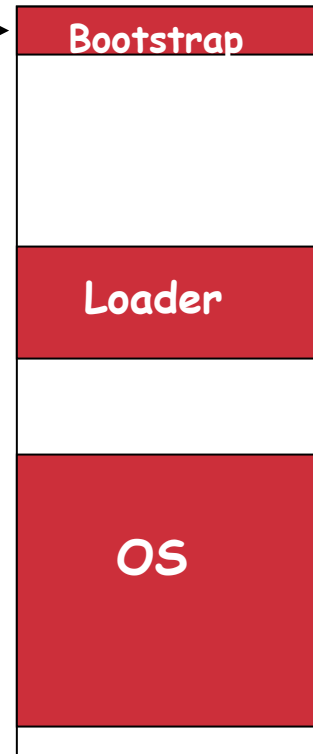
# Start-up (bootstrapping)

## Memory (ROM)

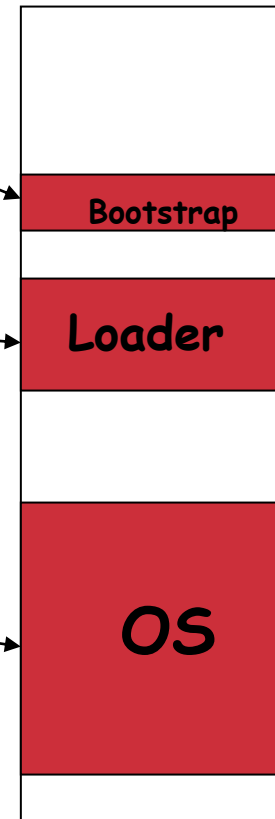
BIOS: performs  
power on self test

POST  
(BIOS/  
UEFI)

### Disk



### Memory (RAM)



- 1 Get bootstrap
2. Get Boot-loader
3. Get OS
4. Initialize hardware
5. Create user interface
6. Start fixed programs  
(initialization scripts)

# Boot Procedure

- You press the power button on your laptop/desktop.
  - The CPU starts up, needs some instructions to work on
  - the main memory is empty at this stage,
- CPU defers to load instructions from the firmware chip on the motherboard and begins executing instructions.
- The firmware code, BIOS, does a **Power On Self Test (POST)**,
  - initializes the remaining hardware,
  - Detects and checks the connected peripherals (mouse, keyboard)
    - You might remember it as a 'beep' that desktops used to make after POST is successful.
- Finally, the firmware code cycles through all storage devices and looks for a **boot-loader** (usually located in first sector of a disk).
- If the boot-loader is found, then the firmware hands over control of the computer to it.

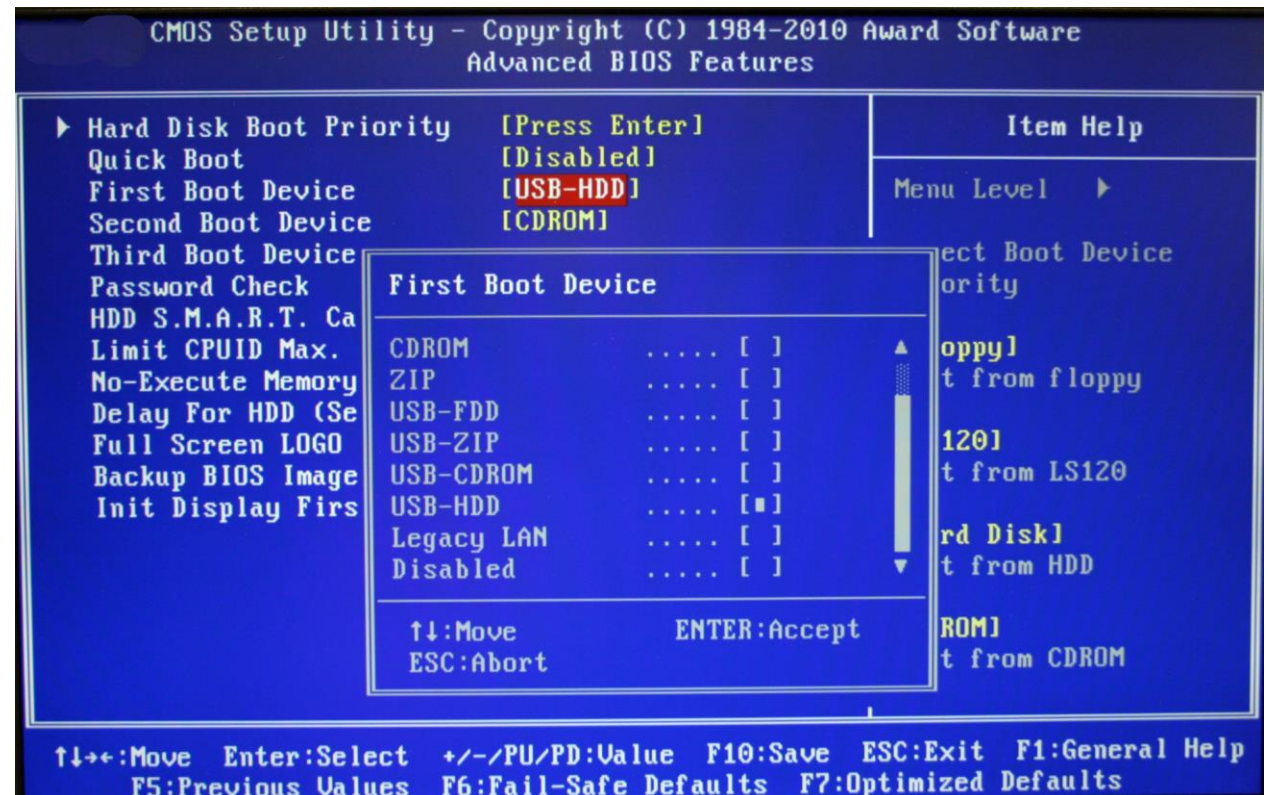


# Boot Procedure

- the boot-loader is loaded, its job is to load the rest of the operating system (GRUB in unix-like os)
- Boot-loader is usually available in the first sector of a disk, which is 512 bytes.
- The boot-loader then loads the kernel into memory.
  - Unix-like operating systems then run the init process (the master process, from which other processes are forked/executed) and finally initialize the run-levels.
  - In Windows, wininit.exe is loaded along with some other processes like services.exe for service control, lsass.exe for local security and authority (similar to run-levels) and lsm.exe for local session management.
- Then some other drivers are initialized, the **Graphical User Interface (GUI)** is loaded and you are presented with the login screen.

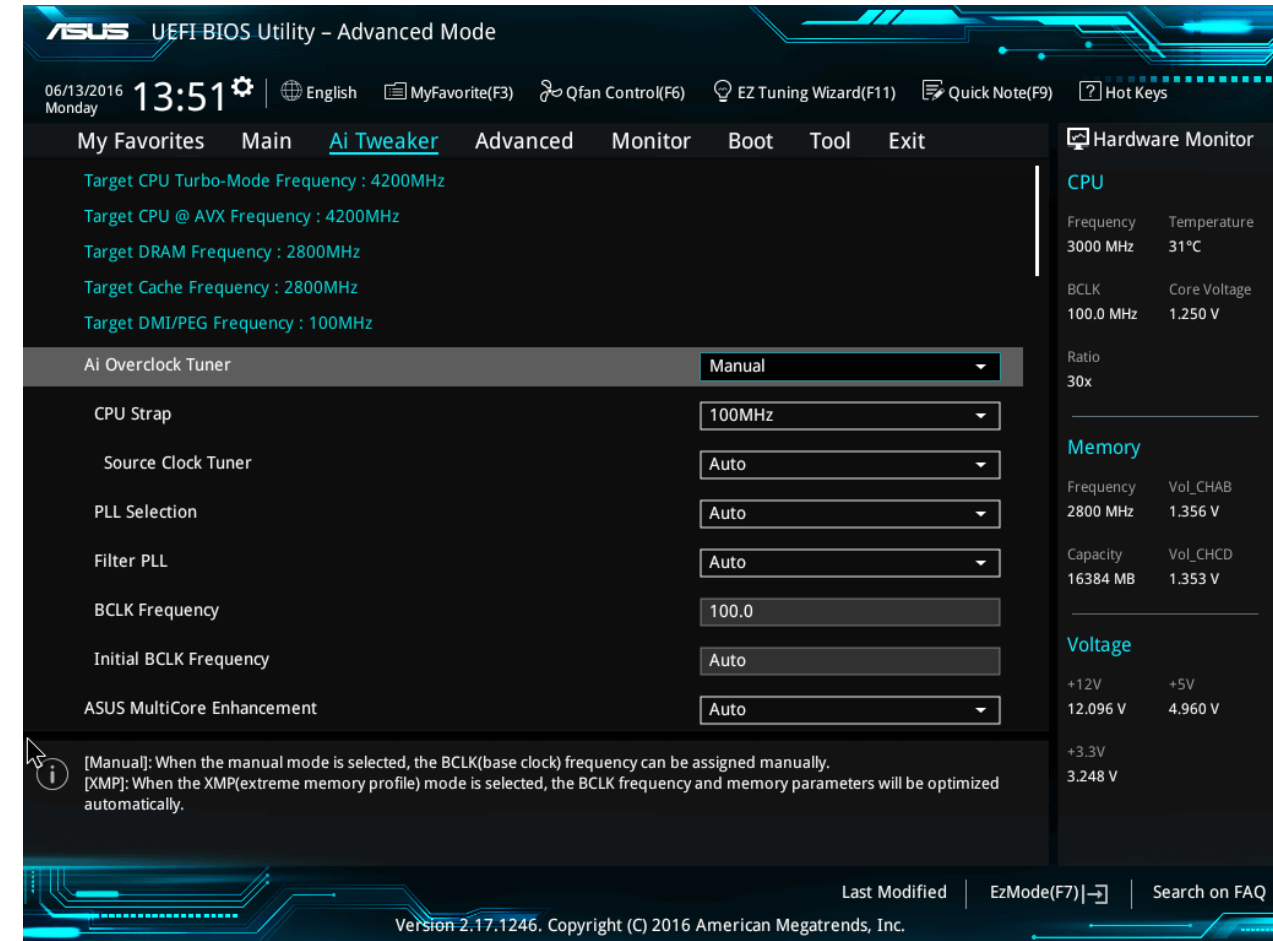
# BIOS

- BIOS stands for Basic Input/Output System, the **firmware** we talked about in the boot procedure.
- It is stored on an EPROM (Erasable Programmable Read-Only Memory),
  - allowing the manufacturer to push out updates easily.
- It provides many helper functions that allow reading boot sectors of attached storage and printing things on screen.
  - You can access BIOS during the initial phases of the boot procedure by pressing del, F2 or F10.



# UEFI (Unified Extensible Firmware Interface)

- UEFI does the same job as a BIOS, but with one basic difference:
- it stores all data about initialization and startup in an .efi file, instead of storing it on the firmware.
- This .efi file is stored on a special partition called EFI System Partition (ESP) on the hard disk (not ROM). This ESP partition also contains the bootloader



# UEFI was designed to overcome many limitations of the old BIOS

- UEFI provides faster boot time.
- UEFI has discrete driver support, while BIOS has drive support stored in its ROM, so updating BIOS firmware is a bit difficult.
- UEFI offers security like "Secure Boot", which prevents the computer from booting from unauthorized/unsigned applications. This helps in preventing rootkits, but also hampers dual-booting, as it treats other OS as unsigned applications. Currently, only Windows and Ubuntu are signed OS.
- UEFI runs in 32bit or 64bit mode, whereas BIOS runs in 16bit mode. So UEFI is able to provide a GUI (navigation with mouse) as opposed to BIOS which allows navigation only using the keyboard.

Standard runlevels		
ID	Name	Description
0	Off	Turns off the device.
1	<u>Single user mode</u>	Does not configure <u>network interfaces</u> or start <u>daemons</u> . <sup>[a]</sup>
6	Reboot	Reboots the device.

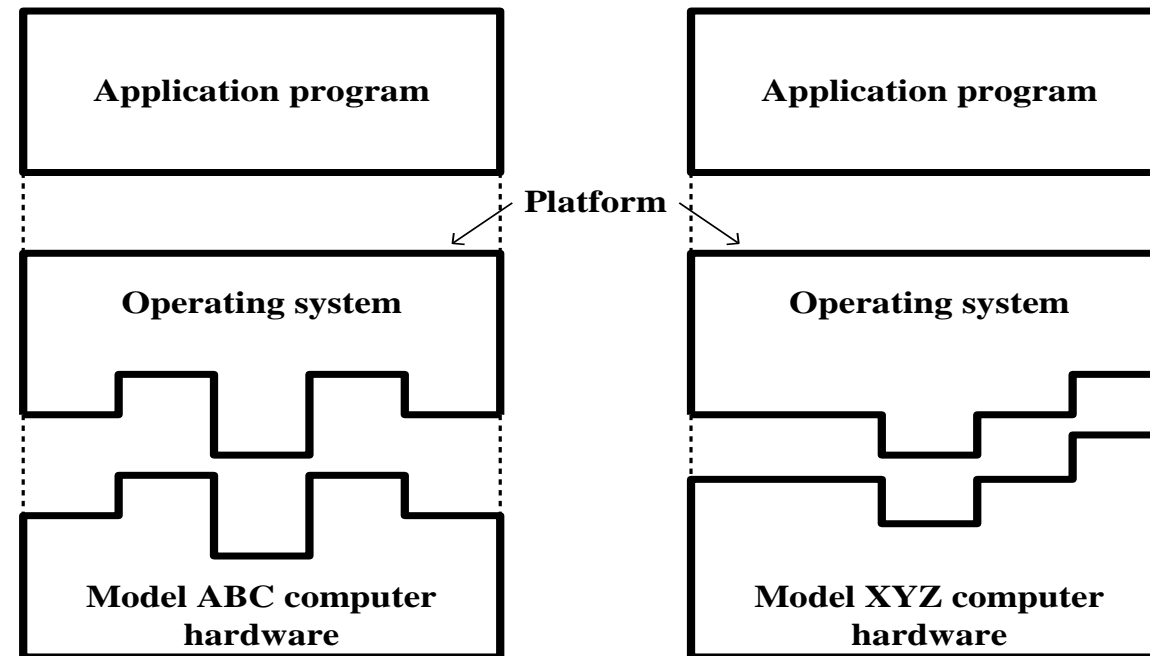
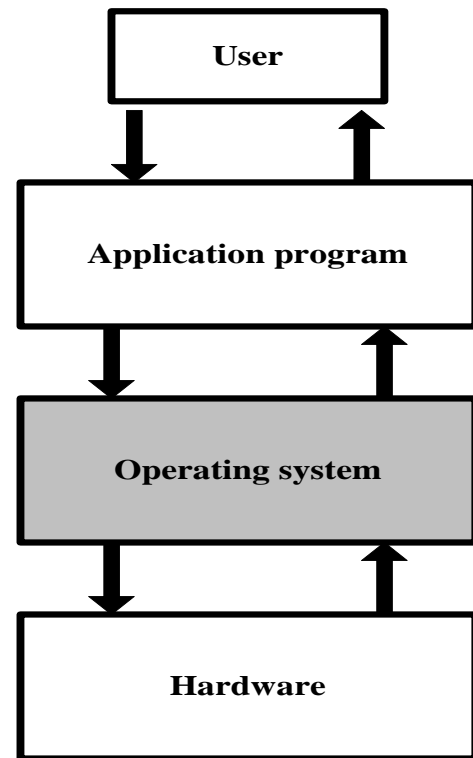
<u>Linux Standard Base</u> (LSB)		
ID	Name	Description
0	Off	Turns off the device.
1	Single-user mode	Mode for administrative tasks. <sup>[2][b]</sup>
2	Multi-user mode	Does not configure network interfaces and does not export networks services. <sup>[c]</sup>
3	Multi-user mode with networking	Starts the system normally. <sup>[1]</sup>
4	Not used/user-definable	For special purposes.
5	Full mode	Same as runlevel 3 + <u>display manager</u> .
6	Reboot	Reboots the device.

# What is an Operating System? (1)

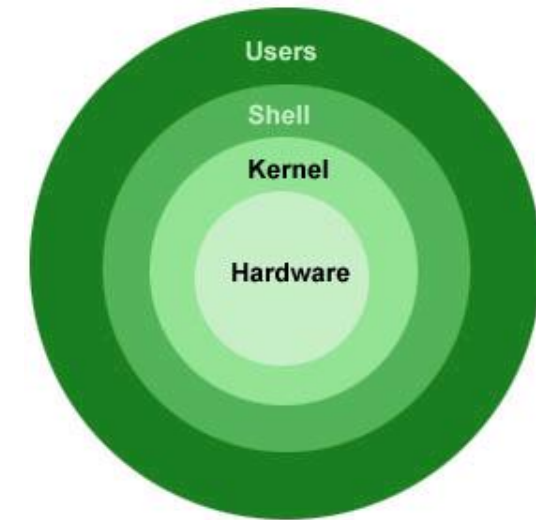
- The OS is a program
  - Permit easy access, control hardware, communicate between user and software, user and hardware, software and hardware, maintain the computer system (hardware and software)
  - Users can access the OS via **GUI** and command line
- Kernel
  - Always resident in memory
  - Responsible for primary OS tasks (process management, memory management, resource management, protection, security)
- Device drivers
  - Program interfaces between OS and peripheral devices
- Shell
  - User interface
- Utility programs
  - Add on programs to further manage and fine-tune your system (anti-viral programs, disk utilities, screen savers, etc)

# What is an operating system? (2)

**The operating system is software that lies between the user / programmer and the hardware**



**A computer platform is a system that consists of a hardware device and an operating system that an application, program or process runs upon.**



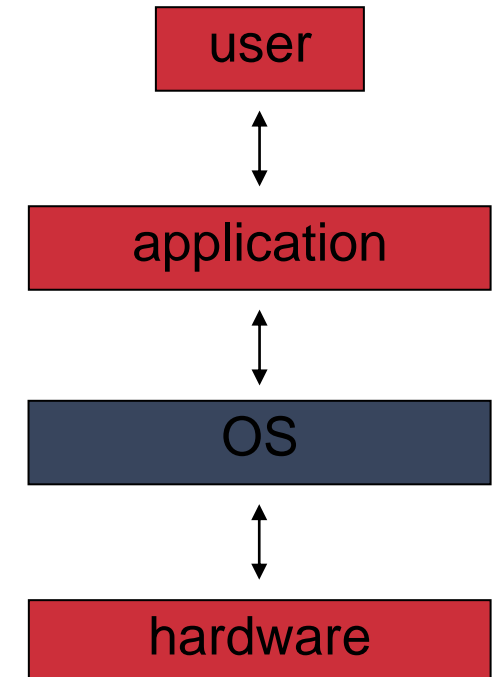


# What is an operating system (OS)? (3)

- “An operating system (OS) is a collection of programs that acts as an intermediary between the hardware and its user(s), providing a high-level interface to low level hardware resources, such as the CPU, memory, and I/O devices. The operating system provides various facilities and services that make the use of the hardware convenient, efficient, and safe”

Lazowska, E. D.: Contemporary Issues in Operating Systems , in: Encyclopedia of Computer Science, Ralston, A., Reilly, E. D. (Editors), IEEE Press, 1993, pp.980

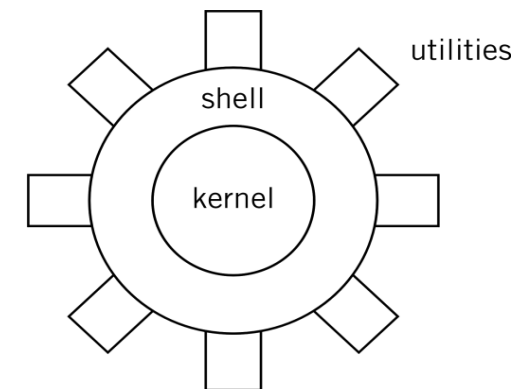
- It is an **extended/abstract** machine (top-down view))
  - Hides the "dirty" details in HW
  - Offers the user and the programmer a **virtual machine** that is easier to program / use
- It is a **resource manager** (bottom-up view))
  - Each program gets **time** on the resource
  - Each program has **space** on the resources (CPU, Memory,...).





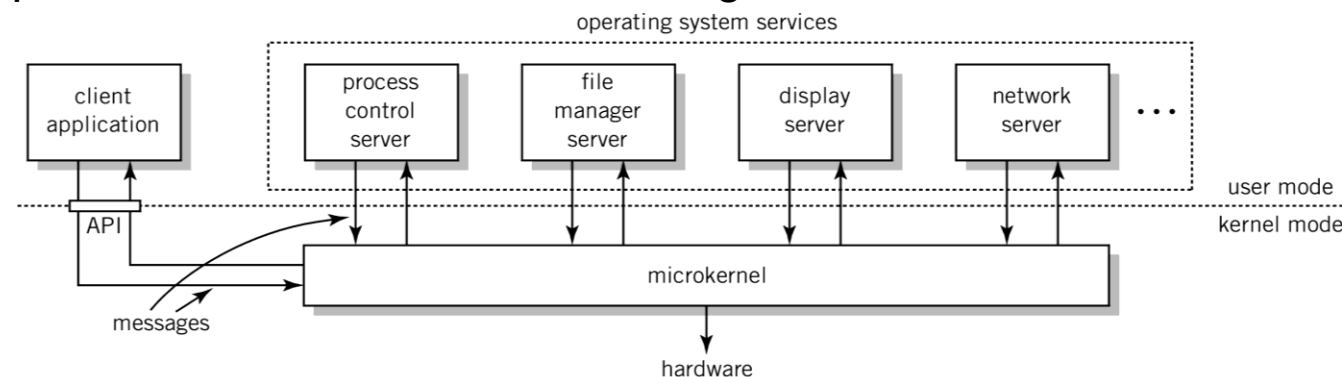
# OS Organization

- Several possible approaches, no standard.
- **Monolithic kernel** (“the big mess”):
  - Written as a collection of functions linked together into one object.
  - Usually effective (no boundaries crossed in the core)
  - Large, complex, crashes easily
  - UNIX, Linux, Windows NT, OSX (in practice, but MACH in the beginning))



A monolithic kernel is **an operating system architecture where the entire operating system is working in kernel space.**

- **Microkernel**
  - Core with minimal functionality (manage interrupt, memory, processor)
  - Other services are implemented as server processes in user mode according to a client-server model.
  - Lots of message exchange (inefficient)
  - small, modular, expandable, portable,...
  - MACH, L4, Chorus, AmigaOS, Minix, K42

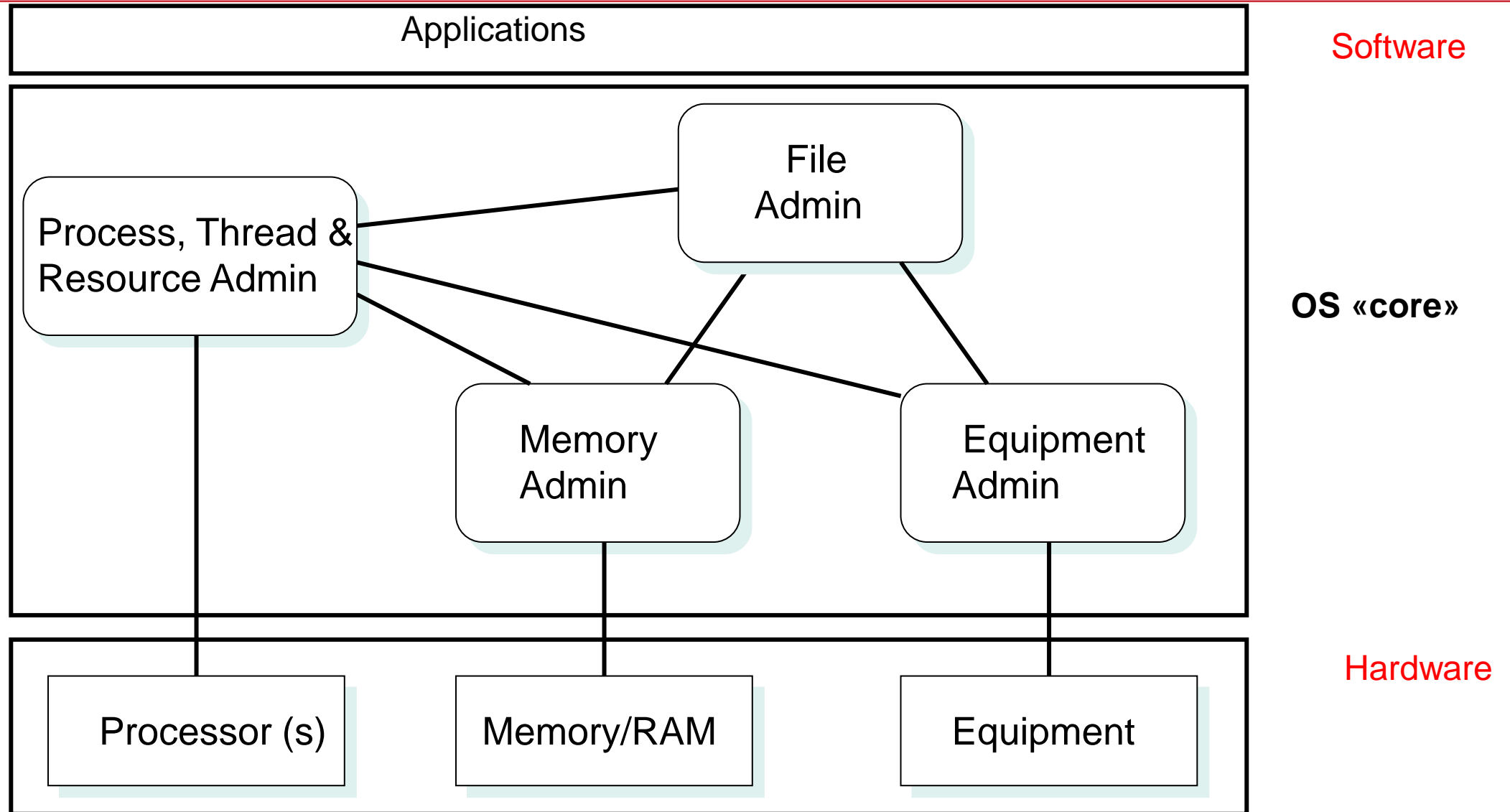


Hybrid Kernel and Exokernel

# Features of operating systems

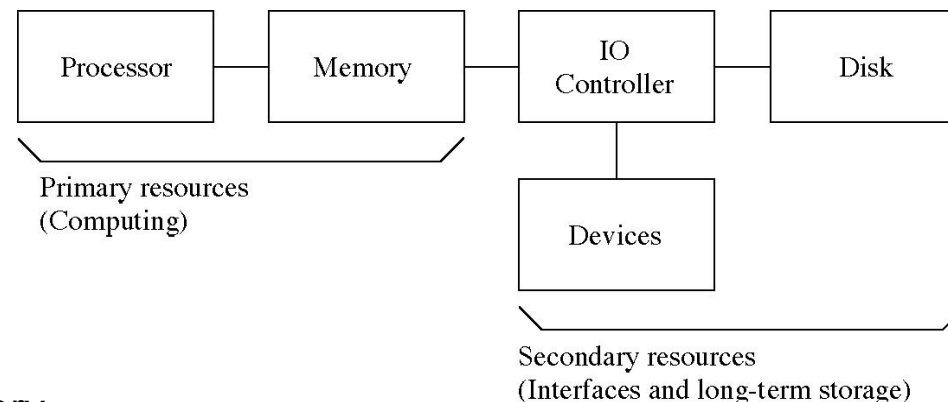
- User interface (GUI, shell!)
- Application run
  - Provides API (Application Programming Interface)
  - In contrast to a user interface, which connects a computer to a person, an application programming interface connects computers or pieces of software to each other.
  - Offers SPI (System Programming Interface)
  - **Service provider interface (SPI)** is an API intended to be implemented or extended by a third party. A service is a well-known set of interfaces and (usually abstract) classes.
- Resource management
  - Processes, memory, external storage, I/O devices
- Hardware management
  - Drivers!
- Network management
  - Safety
- Most often closely linked to the file system
- Not all computer applications need an OS

# Analysis of OS «core»



# Operating system - main functions

- Starting up the machine
  - Internal treatment
  - GUI, "auditing", security, error handling
- Process management or processing
  - Start-up, removal, scheduling, response time
- Memory processing
  - Paging, segmentation, virtual memory, shared memory
- Disk processing
  - File system, File management
  - Device / Equipment processing
  - Drivers, interrupt processing, spools
- Network processing
- Utility programs
  - File manager, backup, defragmentation .....



# Process Management and Scheduling Memory Management

Single and multiprocess Management

# Forms of Process Management

- Can the OS run more than one program at any time?
  - not simultaneously but concurrently
- One process at a time
  - single tasking
  - batch processing
- Concurrent processing
  - multiprogramming
  - multitasking
  - multithreading
  - multiprocessing

<https://slidetodoc.com/operating-systems-lecture-3-multiprogramming-multithreading-multiprocessing-and/>

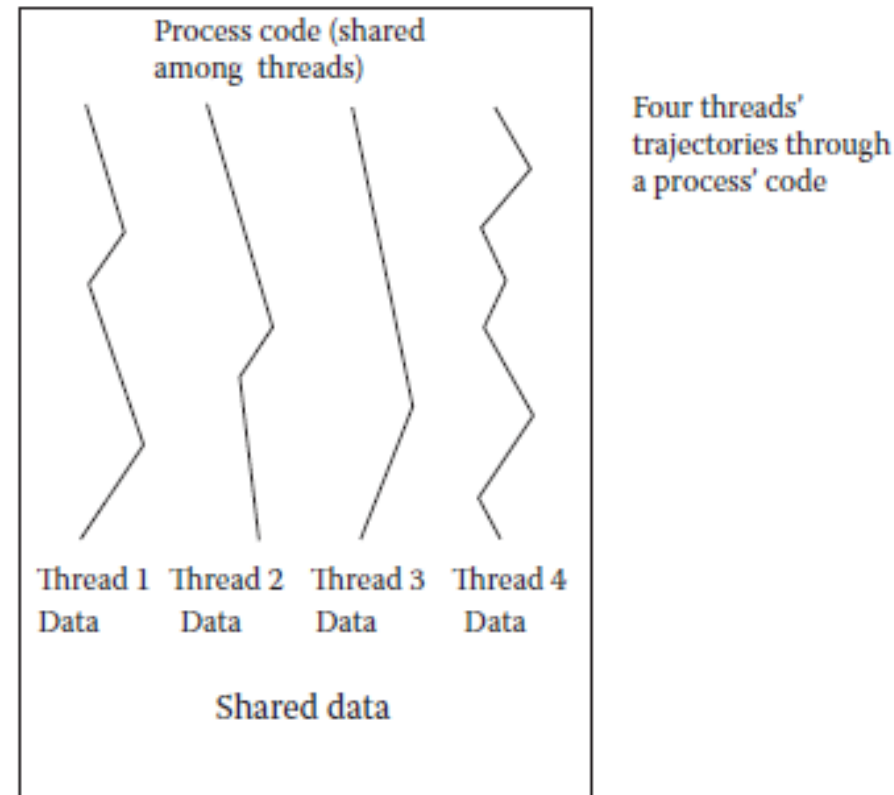
# Concepts

- **Process**
  - A program in progress with associated resources.
- **Resources.**
  - Everything a program needs to run.
  - CPU, RAM, I/O,...

# Threads & Multithreading

- Threads – multiple instances of the same process sharing the same code
  - But with separate data
- For instance, you might have 3 Firefox windows open, these are threads of the same process
- Threads make their way through the same code along different paths
  - See figure to the right

- Multithreading is multitasking across both processes and threads
  - Switching between threads is simpler (less time consuming)





# Process Management and Scheduling

- Process – a running program
  - Processes have a status (running, ready, waiting, stopped)
  - And data (stored in memory, cache, registers)
- Process management is how the OS handles the tasks of
  - starting processes, managing running processes, performing interprocess communication, terminating processes
- Another aspect is process scheduling – selecting the next process to run
- Scheduling algorithms include
  - round robin, priority, first come first serve, shortest job first, longest job first

CPU as a Resource

# Single Tasking and Batch Processing

- Single Tasking
  - User starts the process
  - Process runs to completion
  - If I/O is needed, CPU waits
  - User is not allowed to run more than 1 program at a time
  - Most early operating systems were single tasking
  - Most PC operating systems were single tasking until mid 90s

## Process's life cycle

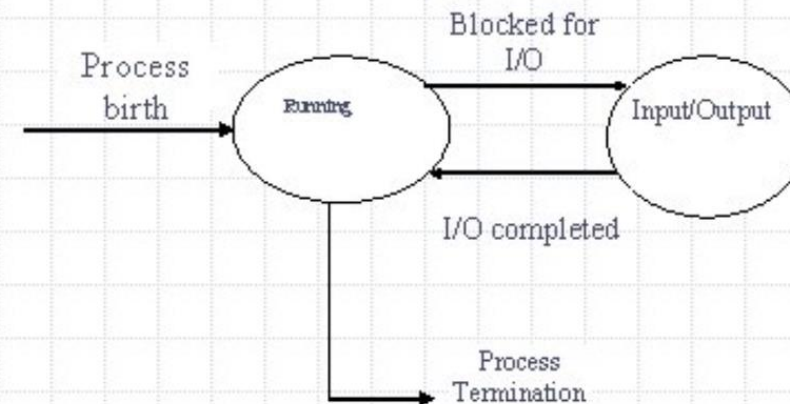


Figure 1: The life cycle of processes in single-programming environments

# Single Tasking and Batch Processing

- Single Tasking

- User starts the process
- Process runs to completion
- If I/O is needed, CPU waits
- User is not allowed to run more than 1 program at a time
- Most early operating systems were single tasking
- Most PC operating systems were single tasking until mid 90s

- Batch Processing

- For multiple user systems
  - users submit processes at any time
  - off-line system receives requests
- OS schedules processes
- Very similar to single tasking (one program at a time)
- No interactivity because the process may not be executed until the user is gone
  - input supplied with the program (e.g., punch cards, magnetic tape)
  - output sent to off-line source (tape, printer)

# Operating system - multitasking

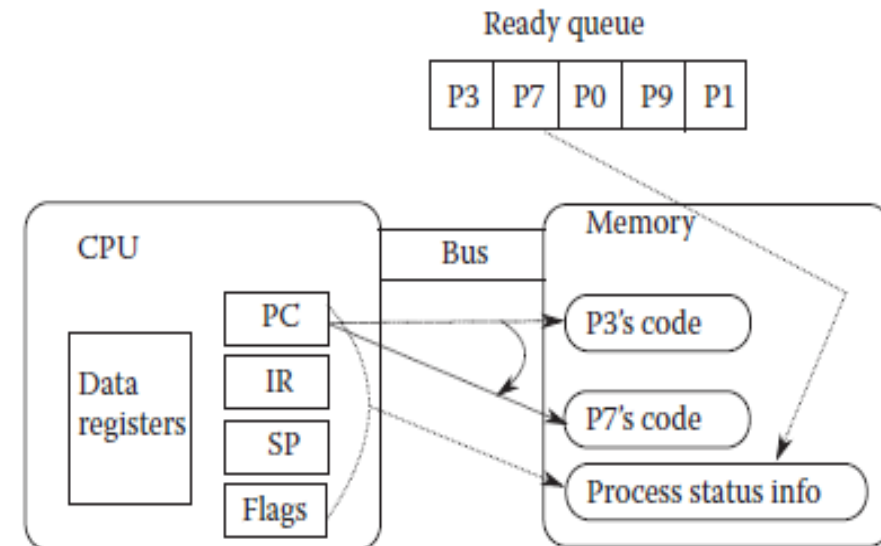
- The computer can run several processes "**simultaneously**"
- The CPU can only process one instruction from one process at a time (on each processor core)
- The operating system must therefore keep track of which process is to use the CPU
- CPU switching between processes is called **context switching**
- **Process status:**
  - Ready: Activated and waiting in line, in memory, but not currently executing by the CPU
  - Running: Running now
  - Blocked: Waiting to be activated,
    - Waiting queue – waiting to be loaded into memory
    - I/O queue – waiting for I/O to complete
- We have a queue for processes with the same status



# A Context Switch

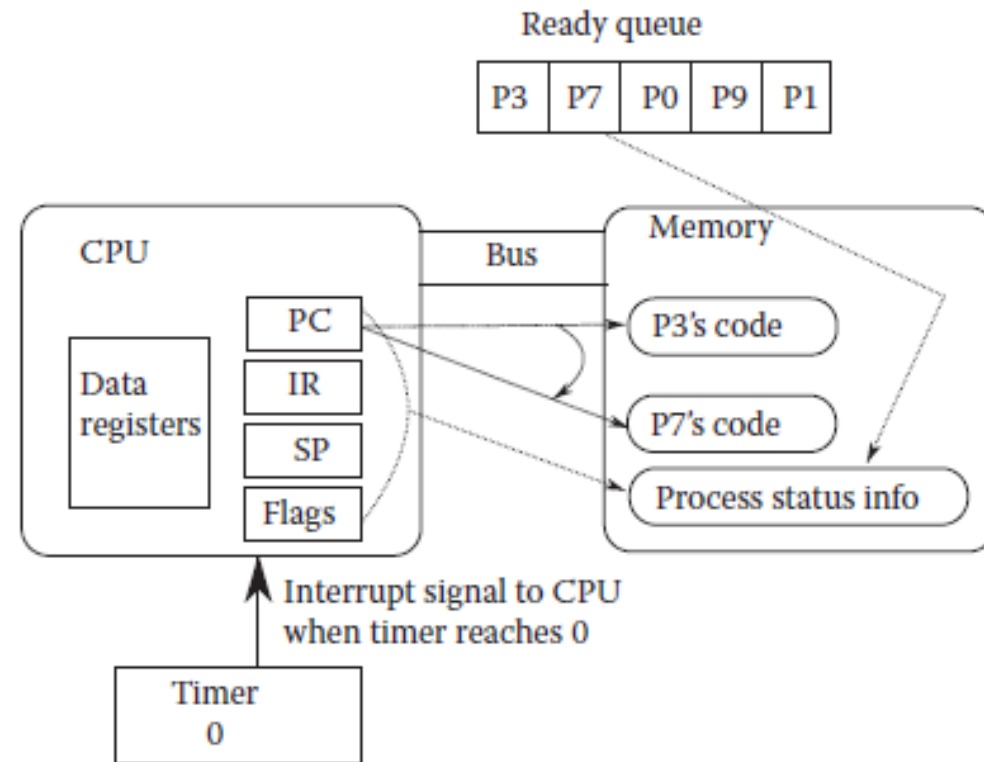
- Concurrent processing needs a mechanism for the CPU to switch from one process to another
  - New process needs to be loaded into memory
  - Old process' status (PC, IR, stack pointer, status flags, etc) saved to memory
  - New process status (register values) restored (from memory)
  - During the switch, the processor is idle

- Ready queue – stores those processes available for switching
  - These are processes loaded into memory which have either started execution and been paused or can start execution
- Below, we see a switch between process P3 and P7



# Multitasking

- Computer appears to be executing two or more processes simultaneously
  - it is switching quickly between processes



**Competitive** multitasking was originally called **time sharing** in the 1960s

# Multiprocessing

- Many computers today have multiple processors
  - Or **multiple cores** on one chip (a core is basically a processor which shares pins and a cache with other cores on the same chip)
- Multiprocessing is multitasking spread across multiple processors
- Most OSs are capable of multiprocessing but do not necessarily share the cores effectively
  - For instance, if you have 4 cores, you would not achieve a 4 times speedup over a computer with a single core processor

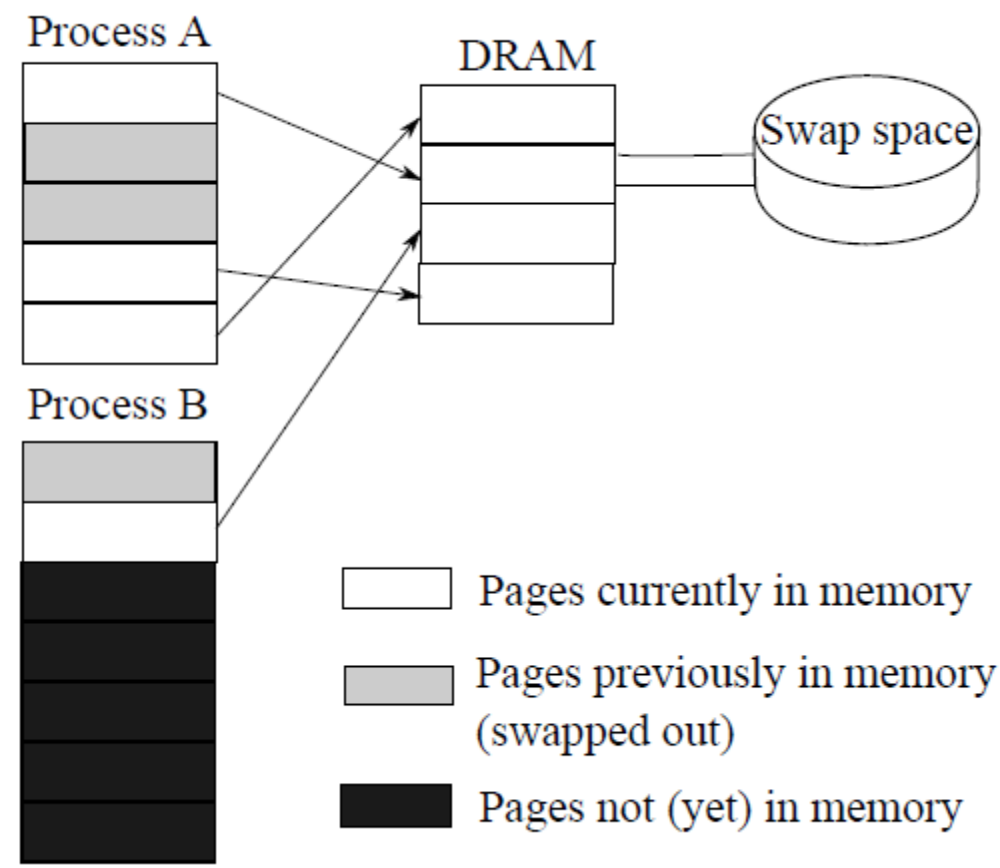
# Memory Management

- We may not be able to fit all of our running program(s) in memory
  - We use swap space as “backing storage”
  - Swap space is placed on the hard disk
- We break our programs into fixed sized units called pages
  - The OS moves pages for us between swap space and memory as needed (on demand)
- This is known as virtual memory
- Program broken into fixed sized pages
  - Memory broken into fixed sized frames
    - when process begins running, first pages are moved to available frames
    - OS maintains page table
  - CPU generates a memory address:
    - page #, position on page
  - Use page table to translate this to physical address
    - frame #, position on frame
  - If page not in memory, page fault occurs

Memory as a Resource



# Virtual Memory: Paging and Page Tables



Process A page table

Page	Frame	Valid
0	1	T
1	-	F
2	-	F
3	3	T
4	0	T

Process B page table

Page	Frame	Valid
0	-	F
1	2	T
2	-	F
3	-	F
4	-	F
5	-	F
6	-	F

Valid means page in memory

# Page Faults and Swapping

- A page fault causes an interrupt
- OS uses a replacement strategy to find frame to free
  - if frame is of a page that has been modified, the page has to be saved back to swap space
  - once a frame is free, new page is loaded from swap space into memory
  - page table updated
- Movement of pages from swap space to memory (and memory to swap space) is swapping
  - swapping slows down the process because disk access time is much slower than memory access time

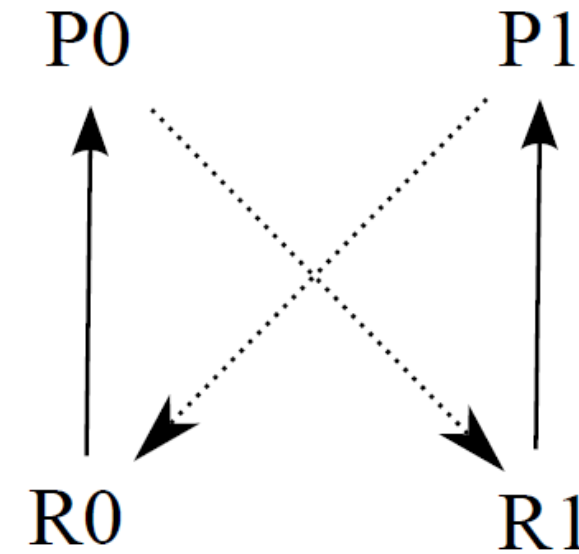
# Resource Management

This is data corruption

- The OS is also responsible for maintaining all other system resources
  - disk files, tape drives, flash drives
  - network access
  - printer access
- Most resources require *mutually exclusive* access
  - no more than 1 process can access a device at a time, others must wait until device is free
- Multitasking processes P0 & P1
  - P0 opens access to file F0
    - P0 reads datum X
    - P0 adds 10 to X
  - CPU switches to P1
  - P1 opens access to file F0
    - P1 reads datum X
    - P1 subtracts 300 from X
    - P1 writes new value to F0
  - CPU switches to P0
  - P0's value of X is now incorrect

# Deadlock - Resource Management

- In this scenario, P0 is using R0, P1 is using R1, P0 wants to access R1 and P1 wants to access R0
  - Deadlock



Solid lines indicate granted resources  
Dotted lines indicate requested resources

- The result is that neither P0 nor P1 can continue
- We would have to kill one of the processes to let the other continue (restart the killed process later)
- OSs will often ignore deadlock and require that the user discover it and kill one of the processes off

# Programmer perspective: The core ("KERNEL")

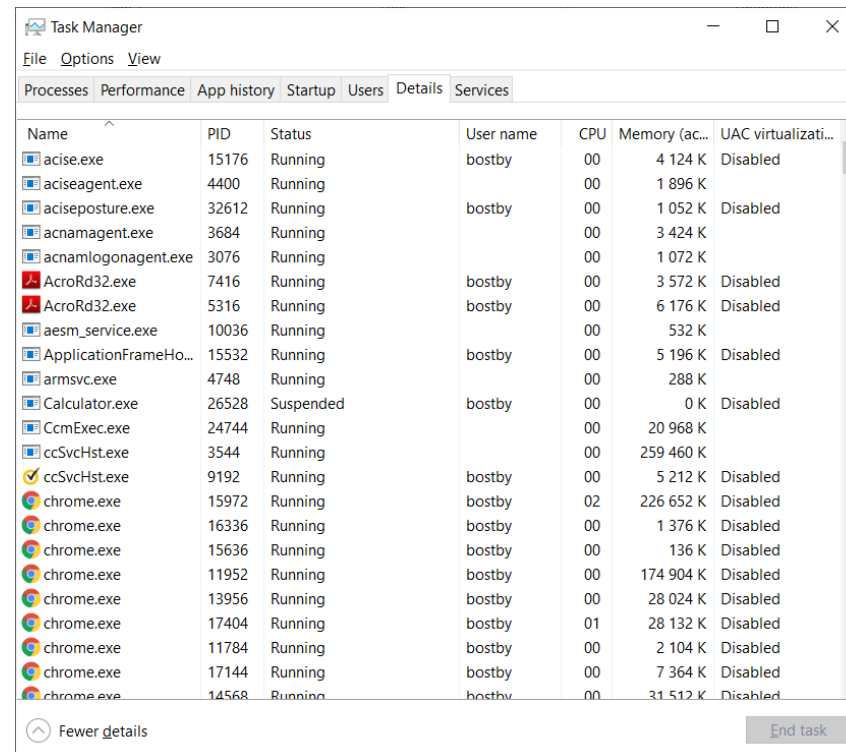
- **The kernel starts up and configures hardware**
- **The core manages processes, threads and resources**

# User mode vs core mode

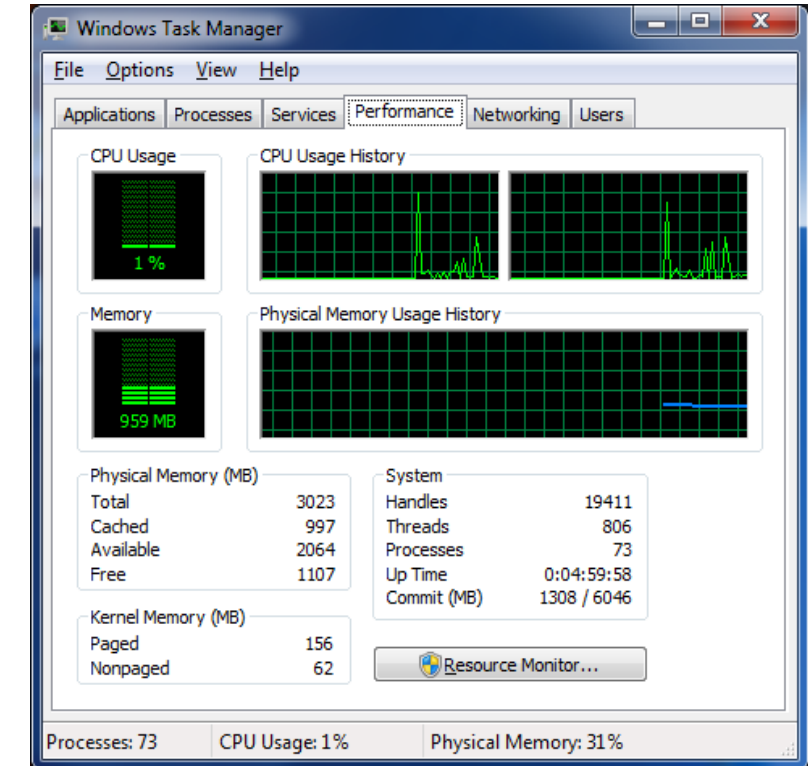
- For security and protection, most CPUs provide the ability to run **instructions** in either **application** or **kernel** mode
- **Common applications** and many OS services run in "**user mode**" (Intel / AMD "ring 3").
  - Unable to access HW, equipment drivers directly, must use an API.
  - Only access the memory allocated by the OS.
  - **Limited instruction set**
- **OS** runs in **core mode** ("call 0")
  - Access to the entire memory
  - All instructions can be run

# Processes and threads: Task manager

- Windows - Task Manager
- Ctrl-Shift-Esc
- Lets you inspect processes and their use of resources

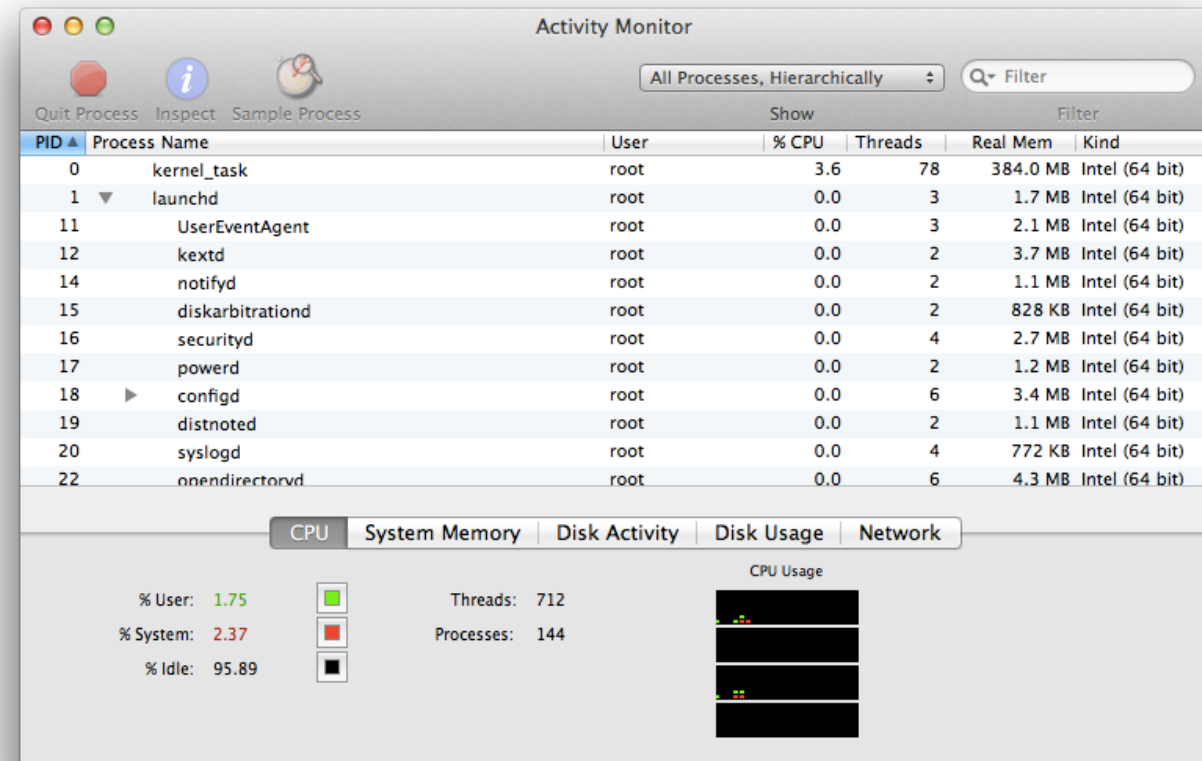


Name	PID	Status	User name	CPU	Memory (ac...	UAC virtualizati...
acise.exe	15176	Running	bostby	00	4 124 K	Disabled
aciseagent.exe	4400	Running	bostby	00	1 896 K	
aciseagent.exe	32612	Running	bostby	00	1 052 K	Disabled
acnamagent.exe	3684	Running	bostby	00	3 424 K	
acnamlogonagent.exe	3076	Running	bostby	00	1 072 K	
AcroRd32.exe	7416	Running	bostby	00	3 572 K	Disabled
AcroRd32.exe	5316	Running	bostby	00	6 176 K	Disabled
aesm_service.exe	10036	Running	bostby	00	532 K	
ApplicationFrameHo...	15532	Running	bostby	00	5 196 K	Disabled
armsvc.exe	4748	Running	bostby	00	288 K	
Calculator.exe	26528	Suspended	bostby	00	0 K	Disabled
CcmExec.exe	24744	Running	bostby	00	20 968 K	
ccSvcHst.exe	3544	Running	bostby	00	259 460 K	
ccSvcHst.exe	9192	Running	bostby	00	5 212 K	Disabled
chrome.exe	15972	Running	bostby	02	226 652 K	Disabled
chrome.exe	16336	Running	bostby	00	1 376 K	Disabled
chrome.exe	15636	Running	bostby	00	136 K	Disabled
chrome.exe	11952	Running	bostby	00	174 904 K	Disabled
chrome.exe	13956	Running	bostby	00	28 024 K	Disabled
chrome.exe	17404	Running	bostby	01	28 132 K	Disabled
chrome.exe	11784	Running	bostby	00	2 104 K	Disabled
chrome.exe	17144	Running	bostby	00	7 364 K	Disabled
chrome.exe	14568	Running	bostby	00	31 512 K	Disabled



# OSX – Activity Monitor

- Displays (sorted) overviews of processes, threads, resources.
- Can also use the bash tools...
- Bash is a **Unix shell and command language**





# Linux: ps, top

- Linux /OSX – ps, top og /proc

– **ps** -- process status

– **top** -- display sorted information about processes

– /proc –«part of the "file system»

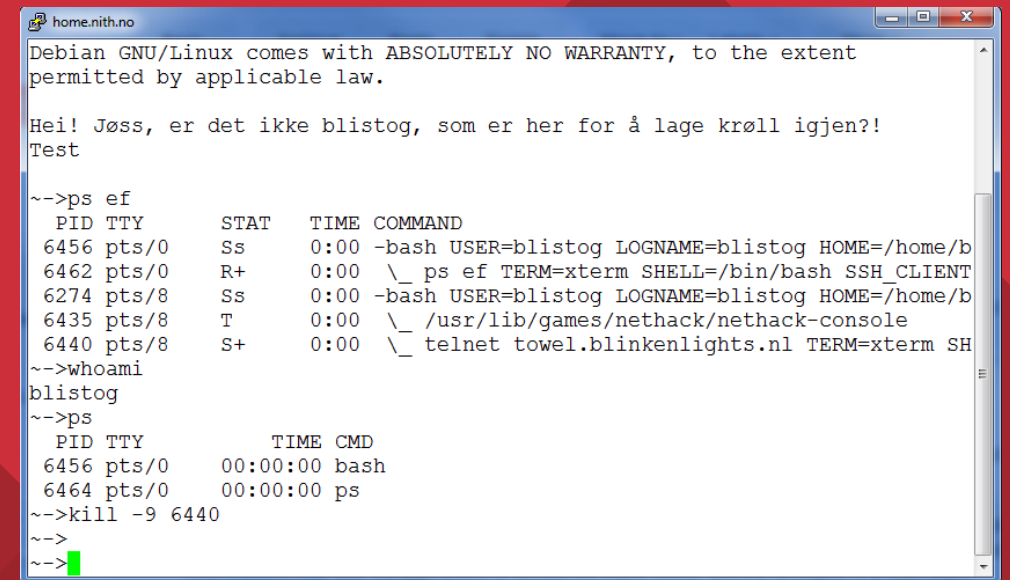
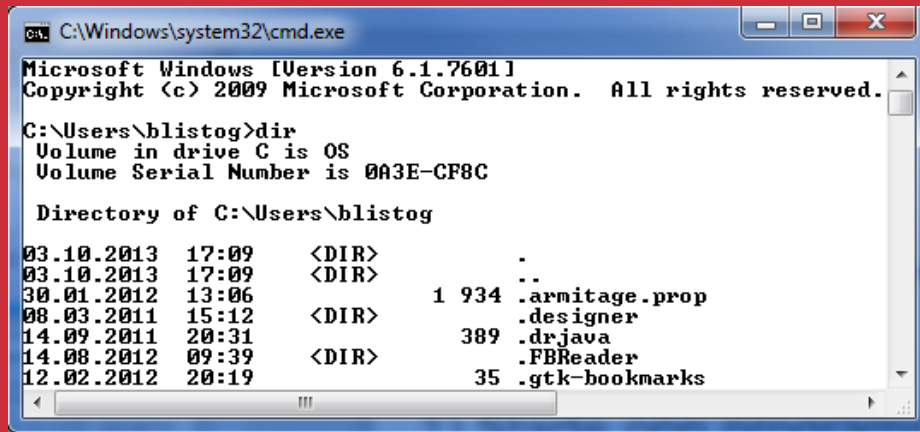
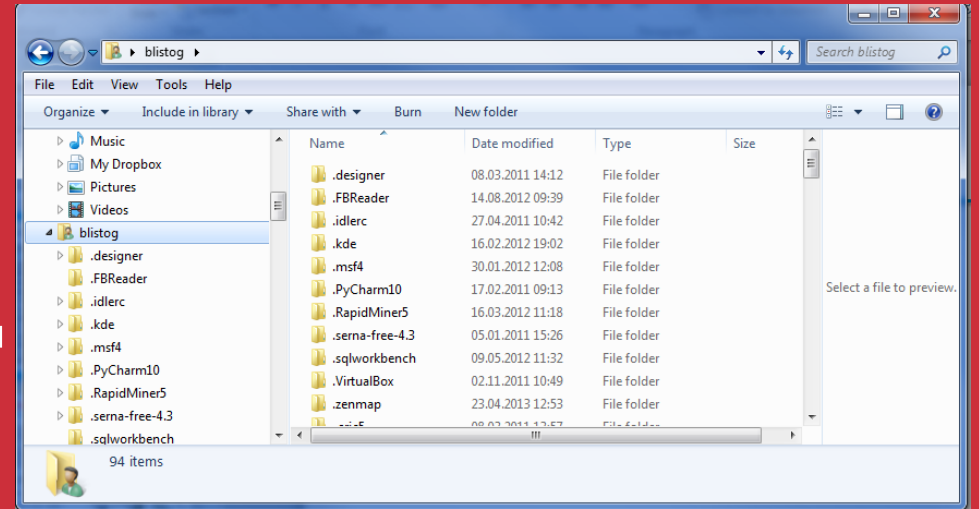
```
top - 17:14:56 up 138 days,  2:02,  3 users,  load average: 0.00, 0.00, 0.00
Tasks:  72 total,   2 running,  70 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,   0.0%sy,   0.0%ni, 96.7%id,   3.3%wa,   0.0%hi,   0.0%si,   0.0%st
Mem:   2059632k total, 1587720k used,   471912k free,   342932k buffers
Swap:  2096472k total,    72k used,  2096400k free,   855740k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	15	0	10348	632	536	S	0.0	0.0	0:01.58	init
2	root	RT	-5	0	0	0	S	0.0	0.0	0:00.00	migration/0
3	root	34	19	0	0	0	S	0.0	0.0	0:00.04	ksoftirqd/0
4	root	10	-5	0	0	0	S	0.0	0.0	3:27.07	events/0
5	root	10	-5	0	0	0	S	0.0	0.0	0:00.00	khelper
22	root	11	-5	0	0	0	S	0.0	0.0	0:00.00	kthread
26	root	10	-5	0	0	0	S	0.0	0.0	0:00.26	kblockd/0
27	root	20	-5	0	0	0	S	0.0	0.0	0:00.00	kacpid
186	root	20	-5	0	0	0	S	0.0	0.0	0:00.00	cqueue/0
189	root	20	-5	0	0	0	S	0.0	0.0	0:00.00	khubd
191	root	10	-5	0	0	0	S	0.0	0.0	0:00.00	kseriod
259	root	10	-5	0	0	0	S	0.0	0.0	0:14.39	kswapd0
260	root	20	-5	0	0	0	S	0.0	0.0	0:00.00	aio/0
466	root	11	-5	0	0	0	S	0.0	0.0	0:00.00	kpsmouse
496	root	10	-5	0	0	0	S	0.0	0.0	0:30.63	mpt_poll_0
497	root	10	-5	0	0	0	S	0.0	0.0	0:00.00	scsi_eh_0

User perspective:

# THE SHELL

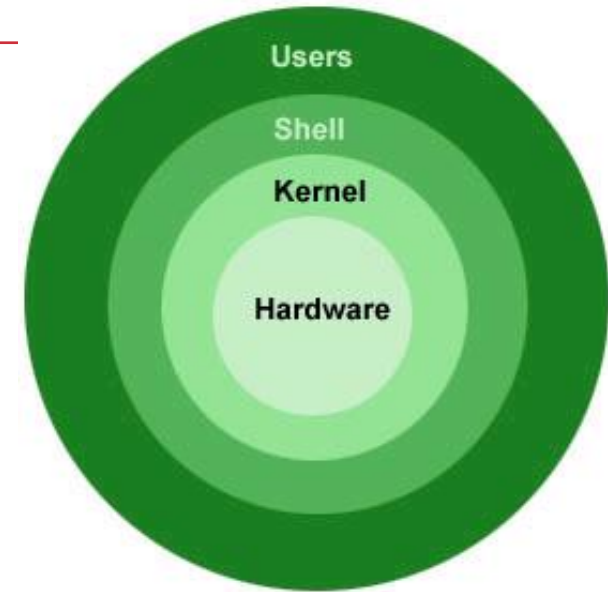
- Most important in normal use:
  - 1) Start up programs ("processes")
  - 2) Maneuver around the file system



- GUI
  - Point and click, drag, double click
  - Touch screen devices use **gesture-based motions (swiping, tapping, pinching, reverse pinching)**
- Command line
  - Enter commands from a command line prompt
  - Commands executed by an interpreter
    - breaks instructions into component parts
    - converts instructions to machine code and executes them
    - maintains a “session”
  - Commands may look cryptic and be hard to learn but offer more power and flexibility
    - Linux command: `find ~ -name 'core*' -exec rm {} \;`

# User perspective: The shell

- The user can order the OS via the shell
  - Start up programs
  - Move files
- Windows
  - Explorer (GUI)
  - Cmd.exe is the command line for Microsoft Windows operating system, with command-based features.
  - **Powershell is a task-based command-line** interface, specifically designed for system admins and is based on the **Windows PowerShell** is still present on the latest versions of Windows 11 and 10, but the latest versions of PowerShell are called **PowerShell** or **Microsoft PowerShell**.
- Linux
  - Example: Nautilus (Gnome)
  - Nautilus, is the official file manager for the GNOME desktop
  - Bash . Net Framework. Bash is a command-line and scripting language for most Unix/Linux-based operating systems.
- OS X
  - Finder (GUI)
  - bash



```
PowerShell
PowerShell 7.1.5
Copyright (c) Microsoft Corporation.

https://aka.ms/powershell
Type 'help' to get help.

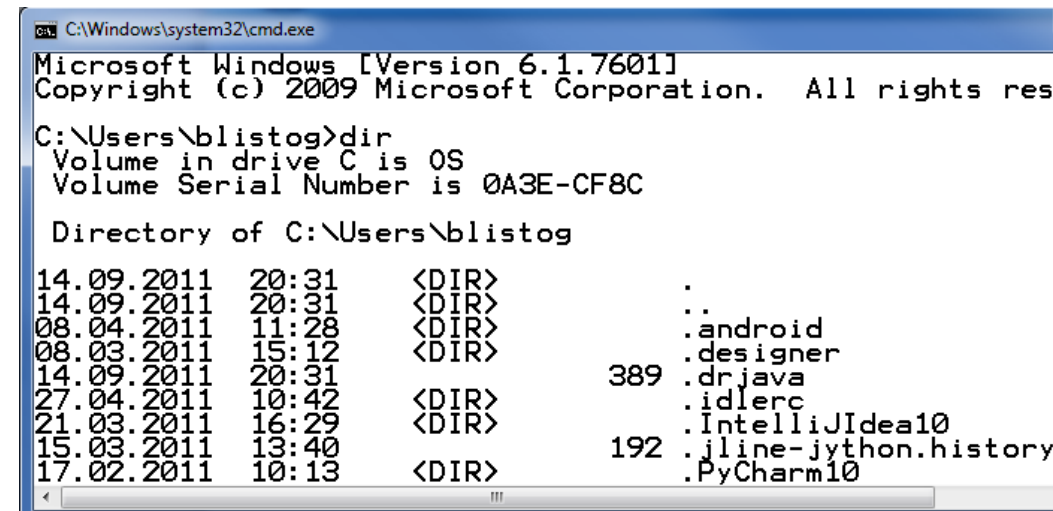
PS C:\Users\Kitten> $PSVersionTable

Name                           Value
----                           -
PSVersion                      7.1.5
PSEdition                      Core
GitCommitId                    7.1.5
OS                              Microsoft Windows 10.0.22000
Platform                      Win32NT
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1
WSManStackVersion              3.0

PS C:\Users\Kitten> function SayHello {
>> Write-Host "Hello, World!"
>> }
PS C:\Users\Kitten> SayHello
Hello, World!
PS C:\Users\Kitten> |
```

# "DOS" (cmd.exe) – UNIX/OSX (bash)

- Similarities.
  - Text and **command** based.
  - **Hierarchical file structure.**
  - **File access** (read, write, ...)
  - Standard I / O with **piping** (>, >>, <, |)



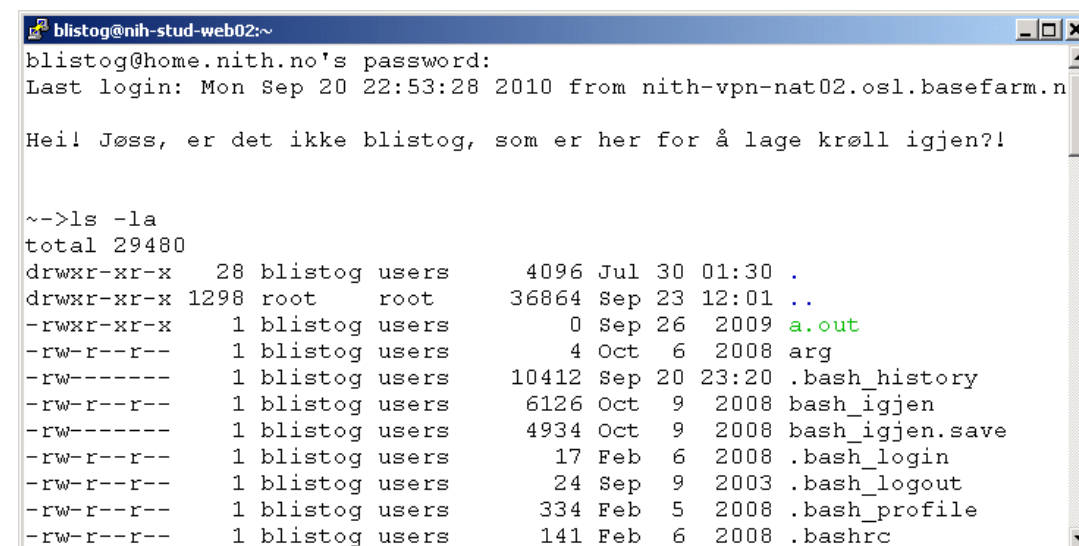
```

C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\blistog>dir
Volume in drive C is OS
Volume Serial Number is 0A3E-CF8C

Directory of C:\Users\blistog

14.09.2011  20:31    <DIR>          .
14.09.2011  20:31    <DIR>          ..
08.04.2011  11:28    <DIR>          .android
08.03.2011  15:12    <DIR>          .designer
14.09.2011  20:31    <DIR>          389 .drjava
27.04.2011  10:42    <DIR>          .idlerc
21.03.2011  16:29    <DIR>          .IntelliJ IDEA10
15.03.2011  13:40    <DIR>          192 .jline-jython.history
17.02.2011  10:13    <DIR>          .PyCharm10
  
```



```

blistog@nih-stud-web02:~
blistog@home.nith.no's password:
Last login: Mon Sep 20 22:53:28 2010 from nith-vpn-nat02.osl.basefarm.n

Heil Jøss, er det ikke blistog, som er her for å lage krøll igjen?!

~~>ls -la
total 29480
drwxr-xr-x  28 blistog users    4096 Jul 30 01:30 .
drwxr-xr-x 1298 root   root    36864 Sep 23 12:01 ..
-rwxr-xr-x   1 blistog users      0 Sep 26  2009 a.out
-rw-r--r--   1 blistog users      4 Oct  6  2008 arg
-rw-----   1 blistog users  10412 Sep 20 23:20 .bash_history
-rw-r--r--   1 blistog users   6126 Oct  9  2008 bash_igjen
-rw-----   1 blistog users   4934 Oct  9  2008 bash_igjen.save
-rw-r--r--   1 blistog users    17 Feb  6  2008 .bash_login
-rw-r--r--   1 blistog users    24 Sep  9  2003 .bash_logout
-rw-r--r--   1 blistog users   334 Feb  5  2008 .bash_profile
-rw-r--r--   1 blistog users   141 Feb  6  2008 .bashrc
  
```

# Commands CMD vs BASH

- display list of files
- display contents of file
- copy file
- rename file
- delete file
- delete directory
- find string
- create directory
- change working directory
- get help
- print file
- Change permissions

## DOS

dir  
type  
copy  
ren  
del  
rd  
find  
md  
cd  
help  
print  
attrib

## UNIX

ls  
cat  
cp  
mv  
rm  
rmdir  
grep  
mkdir  
cd  
man  
lpr  
chmod

# Commands and flags

- How a command behavior can be modified by setting different flags.
  - placed directly or after a flag marker (/, -, --)

```
dir /?
```

```
ls -la
```

```
ps aux
```

# Commands and redirection

- Usually the output is to the monitor (console), but in addition you have some standard redirection operators.



- Writes output into a file.
- Ex:  
`dir > katalog.txt, or`  
`ls -la > directorylist`



- Writes output at **the end** of a file.



- Takes the **contents** of a file and uses it as arguments to the command.



- Takes output from one command and delivers it as input to the next command.
- Ex: `dir | sort /R | more`  
`ps aux | grep blistog | wc`
- «**pipe**»



# Signals

- Under both Windows and Linux / OSX, you can in the CLI shell (command line interface) give signals to the OS with a few keyboard shortcuts.
  - `Ctrl-C` Abortion running process.
  - `Ctrl-D` End of file (EOF)
  - `Ctrl-Z` Put running process to sleep
  - Etc.

# Why use command shell?

- GUI is excellent for "point, drag, click" tasks.
  - Bad for e.g. automation due to **poor semantics**
  - Command shells usually offer a complete programming language (called batch / bash).
- Servers
  - Most web server runs Apache on Linux, and is not set up with GUI as it would be poor resource use.
  - Microsoft has also started to focus more on shell than GUI for servers .. (Powershell)

# File management

# Operating system files

- Data stored in a memory disappears when the power is turned off
- Data can also be stored permanently
  - Disk, floppy disk, CD, tape, .....
  - Such a repository consists of folders and files
  - Hierarchical structure
- Name, access rights, creation date, .....

# Abstraction

- To abstract means to remove (insignificant) details.
- For example: A **file** will (most often) consist of metadata (data about data)
  - A name
  - when created, last modified, o.l.
  - how big (KiB)
  - which sectors of the disk data is physically located on
  - ... and the content itself (data / instructions)
- The files are organized in directories (directories / folders))
  - another abstraction, which is also a **type of file** that contains metadata and addresses of other files.

# File management as an OS Tasks

- File management
  - Users dictate file system operations through windows explorer or command line instructions (DOS, Linux)
    - creating directories, moving files, copying files, deleting files, etc
  - Every file operation is a *request*
    - the OS must ensure that the user has access to the given resource to perform the operation
- Protection
  - Resources need protection from misuse
  - Each resource has permissions associated with it (access, read, write, execute)
  - User accounts/authentication are used to establish protection
- Security extends permissions across a network
  - A user can then remotely control the computer

# File management

- The file manager in the OS must:
  - Store information securely on equipment (storage device: hard disk,...).
  - Map the connection between block storage on the equipment and the file abstraction (logical "image")
  - Allocate / deallocate storage devices
  - Manage data sharing
  - Protect against errors, crashes and malicious users
  - Simultaneous control

# Drivers

- Drivers are software that can translate back and forth between OS requirements and the instruction set of the **controller** on the equipment.
- The main cause of crashes is poorly written drivers
  - Microsoft claims "at least" 70%



# Single and multiprocess Management

# Forms of Process Management

- Can the OS run more than one program at any time?
  - not simultaneously but concurrently
- One process at a time
  - single tasking
  - batch processing
- Concurrent processing
  - multiprogramming
  - multitasking
  - multithreading
  - multiprocessing

<https://slidetodoc.com/operating-systems-lecture-3-multiprogramming-multithreading-multiprocessing-and/>

# Single Tasking and Batch Processing

- Single Tasking

- User starts the process
- Process runs to completion
- If I/O is needed, CPU waits
- User is not allowed to run more than 1 program at a time
- Most early operating systems were single tasking
- Most PC operating systems were single tasking until mid 90s

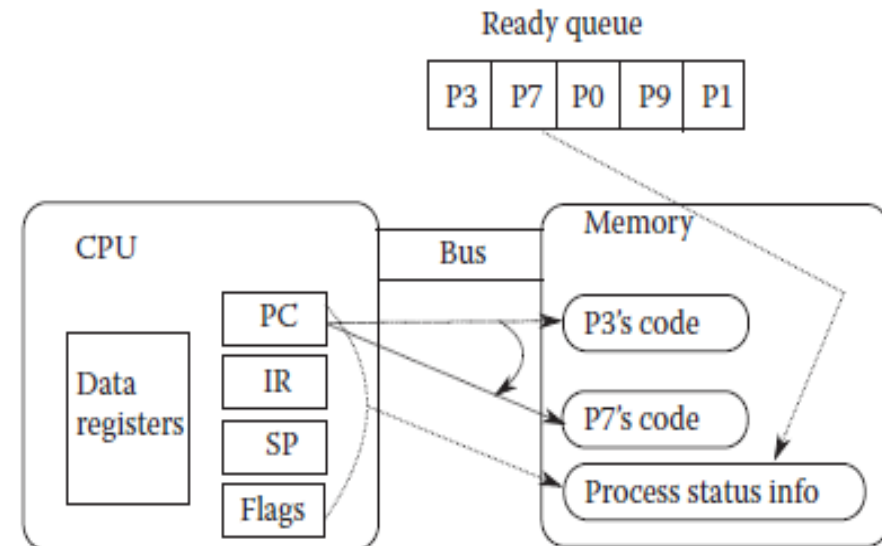
- Batch Processing

- For multiple user systems
  - users submit processes at any time
  - off-line system receives requests
- OS schedules processes
- Very similar to single tasking (one program at a time)
- No interactivity because the process may not be executed until the user is gone
  - input supplied with the program (e.g., punch cards, magnetic tape)
  - output sent to off-line source (tape, printer)

# A Context Switch

- Concurrent processing needs a mechanism for the CPU to switch from one process to another
  - New process needs to be loaded into memory
  - Old process' status (PC, IR, stack pointer, status flags, etc) saved to memory
  - New process status (register values) restored (from memory)
  - During the switch, the processor is idle

- Ready queue – stores those processes available for switching
  - These are processes loaded into memory which have either started execution and been paused or can start execution
- Below, we see a switch between process P3 and P7

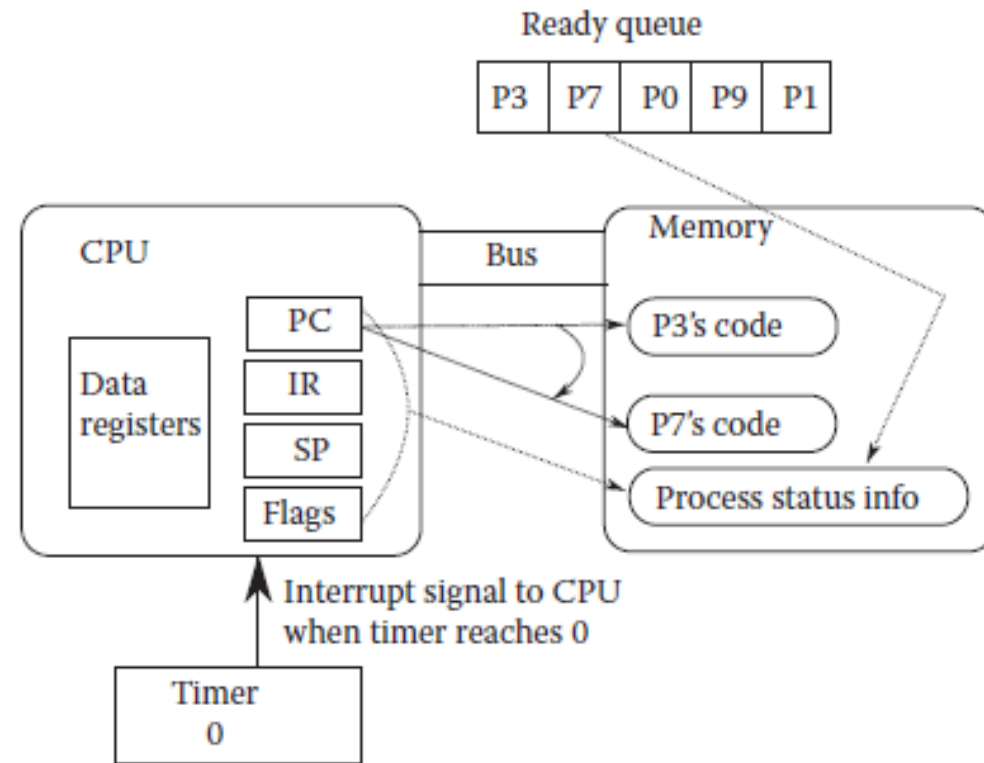


# Multiprogramming & Multitasking

- Multiprogramming (or cooperative multitasking) is like batch processing in that one process is executed at a time except
  - When a process requires I/O, it is moved to an I/O queue
  - Context switch to next waiting process
  - When process finishes with I/O, switch back to it
  - More efficient than batch processing or single tasking because the CPU does not remain idle during time consuming I/O, only delay is during the context switches
- For multitasking, add a timer and before a process starts, set the timer to some value (e.g., 10,000)
  - After each machine cycle, decrement the timer
  - When timer reaches 0, force a context switch to the next process in the ready queue
  - User will not notice the time it takes to cycle through the processes

# Multitasking

- More appropriately called competitive (or **pre-emptive**) multitasking
- Computer appears to be executing two or more processes simultaneously
  - it is switching quickly between processes
- Most OSs do both cooperative and competitive multitasking today

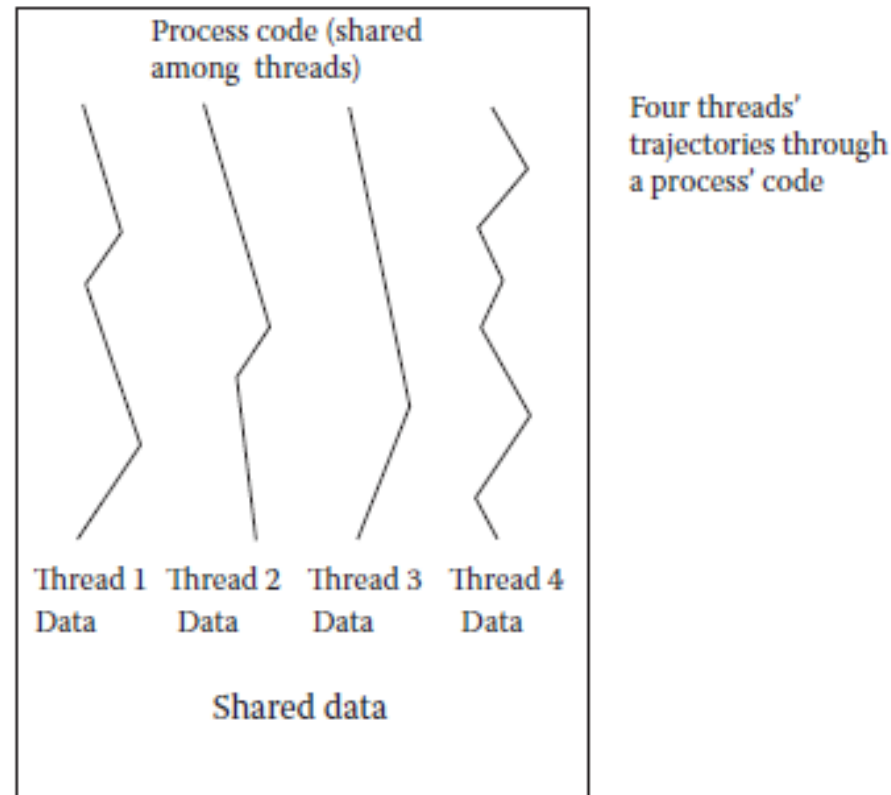


**Competitive** multitasking was originally called **time sharing** in the 1960s

# Threads & Multithreading

- Threads – multiple instances of the same process sharing the same code
  - But with separate data
- For instance, you might have 3 Firefox windows open, these are threads of the same process
- Threads make their way through the same code along different paths
  - See figure to the right

- Multithreading is multitasking across both processes and threads
  - Switching between threads is simpler (less time consuming)



# Multiprocessing

- Many computers today have multiple processors
  - Or **multiple cores** on one chip (a core is basically a processor which shares pins and a cache with other cores on the same chip)
- Multiprocessing is multitasking spread across multiple processors
- Most OSs are capable of multiprocessing but do not necessarily share the cores effectively
  - For instance, if you have 4 cores, you would not achieve a 4 times speedup over a computer with a single core processor





# Interrupts

- The CPU's fetch-execute cycle runs continuously unless interrupted
  - Interruptions can come from hardware or the running program
    - control+alt+delete
    - mouse moved
    - printer out of paper
    - program has run-time error
  - An interrupt interrupts the CPU at the end of fetch-execute cycle
  - Upon interrupt, CPU determines what device (or user or software) raised interrupt
    - select the proper interrupt handler (piece of OS code)
    - execute interrupt handler to handle the interrupt
    - resume the interrupted process

# Operating system - memory

- Programs that are running are in memory
- The operating system is also a program!
- Regular RAM is enhanced in several ways
- Cache is a very fast memory (SRAM)
- Used both between CPU and RAM and for some devices
- If the RAM is too small, some of the contents may be temporarily swapped.

- The OS uses a roadmap to determine which thread should be allowed to run on the processor
- Context switching takes time
- There are two roadmap principles
  - Non-preemptive
    - the CPU is allocated to the process till it terminates or switches to waiting state.
  - Preemptive
    - **the CPU is allocated to the processes for the limited time.**
    - A clock-controlled interruption periodically ensures that the thread that is running is set to ready
- The power method
  - NT, OSX and Linux have **preemptive schedules** based on priority queues and the Round Robin algorithm

# Different types of executable files

- **Different OSes use different formats for executable program files.**
  - binary executable file formats which, once loaded by a suitable executable loader, can be directly executed by the CPU rather than being interpreted by software.
- Windows: Portable Executable (PE)
  - The **Portable Executable (PE)** format is a file format for executables, object code, DLLs and others used in 32-bit and 64-bit versions of Windows operating systems.
- Linux: ELF
  - In computing, the **Executable and Linkable Format**<sup>[2]</sup> (**ELF**, formerly named **Extensible Linking Format**), is a common standard file format for executable files, object code, shared libraries.
- OSX: Mach-O
- They also use different system calls and standard libraries

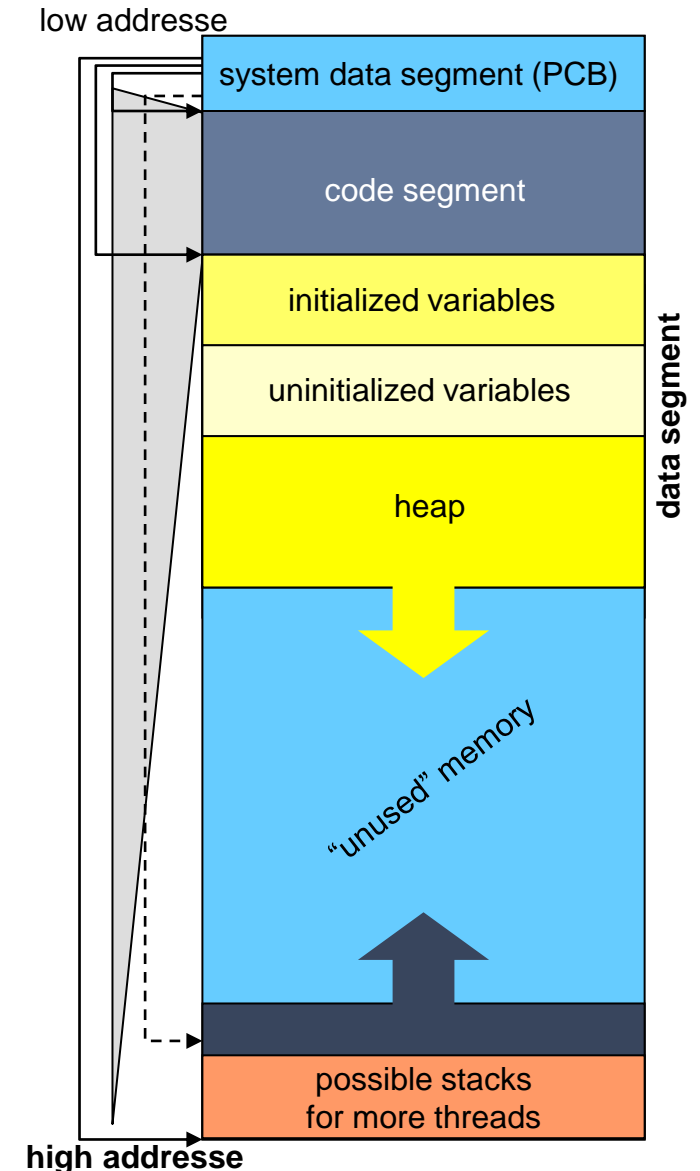
# The process (thread)'s Memory

On the Intel architecture, a task partitions its assigned memory

..

- **TEXT (code) segment**
  - Read from program file for example by exec
  - usually read-only
  - Can be shared by several threads
- **DATA segment**
  - initialized global variables (0 / NULL)
  - uninitialized global variables
  - heap
  - dynamic memory for example, allocated with malloc
  - growing upwards
- **STACK segment**
  - Variables in a function
  - Stored registry states (calling function EIP)
  - Grows downwards
- **system data segment(PCB)**
  - segment pointers
  - pid
  - program and stack pointers.
    - ...
  - Multiple stacks for the threads

8048314	<add>:
8048314:	push %ebp
8048315:	mov %esp,%ebp
8048317:	mov 0xc(%ebp),%eax
804831a:	add 0x8(%ebp),%eax
804831d:	pop %ebp
804831e:	ret
804831f	<main>:
804831f:	push %ebp
8048320:	mov %esp,%ebp
8048322:	sub \$0x18,%esp
8048325:	and \$0xffffffff0,%esp
8048328:	mov \$0x0,%eax
804832d:	sub %eax,%esp
804832f:	movl \$0x0,0xffffffffc(%ebp)
8048336:	movl \$0x2,0x4(%esp,1)
804833e:	movl \$0x4, (%esp,1)
8048345:	call 8048314 <add>
804834a:	mov %eax,0xffffffffc(%ebp)
804834d:	leave
804834e:	ret
804834f:	nop
...	
...	
...	

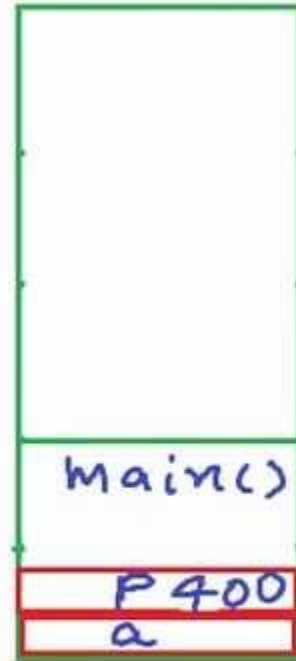


```

#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a; // goes on stack
    int *p;
    p = (int*)malloc(sizeof(int));
    *p = 10;
    free(p);
    p = (int*)malloc(20*sizeof(int));
}

```

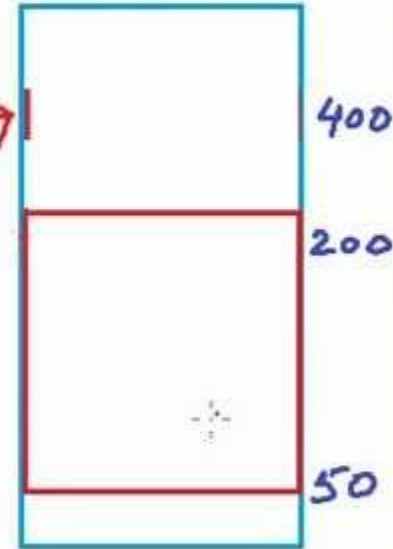
Stack



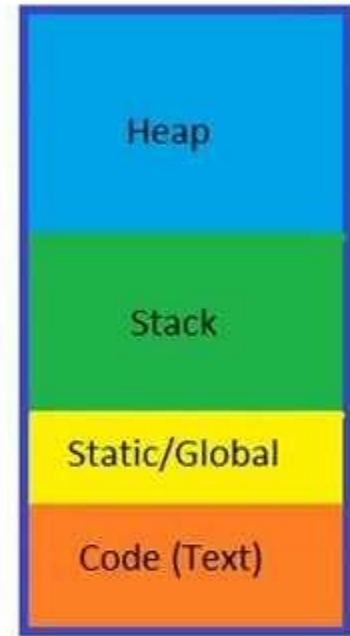
Global



Heap



Application's memory



Free Store

- Hardware has controlled what the OS offers / does;
- New hardware has often been added based on OS needs.

# HISTORY

(Not necessary, but exciting and useful to have been through)



# Operating system - early development

- Until the 1960s, there was no need for an operating system. The computer was used by one person or one job at a time
- When **batch running**, the machine had to have a queue organizer
- When **time-sharing** (multitasking), the machine had to keep track of several processes at once
- The use of mass storage required an organized file system
- Virtual memory required better control of the file system
- Constantly new external equipment required simplified I / O control
- Users were computer people

# Operating system - further development

- Complicated start-up procedure
- Multiple interactive users on the same machine
- Need for general service programs for users
- Networking machines
- Protect machines and users from machines and users
- PCs underwent the same development as larger machines already had
- Increasing supply of applications on machines and online
- The weight of users without a data background

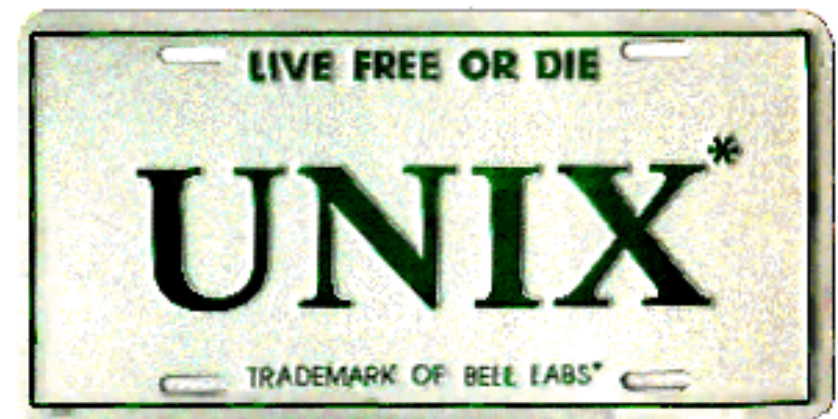
# Specialized operating systems

- Parallel systems.
  - More than one CPU in the same machine
  - Can run multiple jobs simultaneously.
- Distributed systems.
  - Several machines work together
  - Sharing jobs and resources.
- Real-time systems.
  - Time-critical response time, controlled by interrupt.
- Network
- Mobile phones and the like

- UNIX
  - Written in C (1971), text-based, portable, multi-user
  - Linux as a modern (PC) version
  - Great degree of freedom both advantage and disadvantage.
- MS-DOS
  - Written in machine code (1980) by Microsoft for IBM
  - Text-based, single-user
  - Later built Windows 3.0-11 GUI achieve DOS
  - Small degree of freedom

- DOS taken a lot from UNIX and may therefore seem to be relatively similar.
  - CP / M: Gary Kildall, 1973 for 8080 and 8 "disk (DR-DOS)
  - 86-DOS: Tim Paterson, 1980 for 8086.
    - **FAT file system**
  - MS-DOS: Microsoft for IBM, 1981, based on 86-DOS that they bought for \$ 75,000.
- The main difference does not apply so much in the commands but in freedom, structure and portability

- Differences.
  - DOS was single-user, UNIX was multi-user
  - DOS only supports single-processing, UNIX supports multi-processing
  - DOS had command interpreters (command.com -> cmd.exe), UNIX uses shell (bash, bash, csh, tcsh, ...)
  - DOS / NT runs on Intel compatible processors, UNIX is much more portable
  - DOS / NT used bat and cmd files, UNIX uses shell scripts



# UNIX history

- Started as Multics at MIT 1965.
  - First time-sharing OS.
- Further developed at Bell 1965-1969.
  - Ken Thompson, Dennis Ritchie, ,.....
  - Transferred from PDP-7 to PDP-11 1970
  - Unics □ Unix
  - Written in C
  - Formed basis for portability.
- Linux - Linus Torvalds 1991



**Simplified  
time map!**

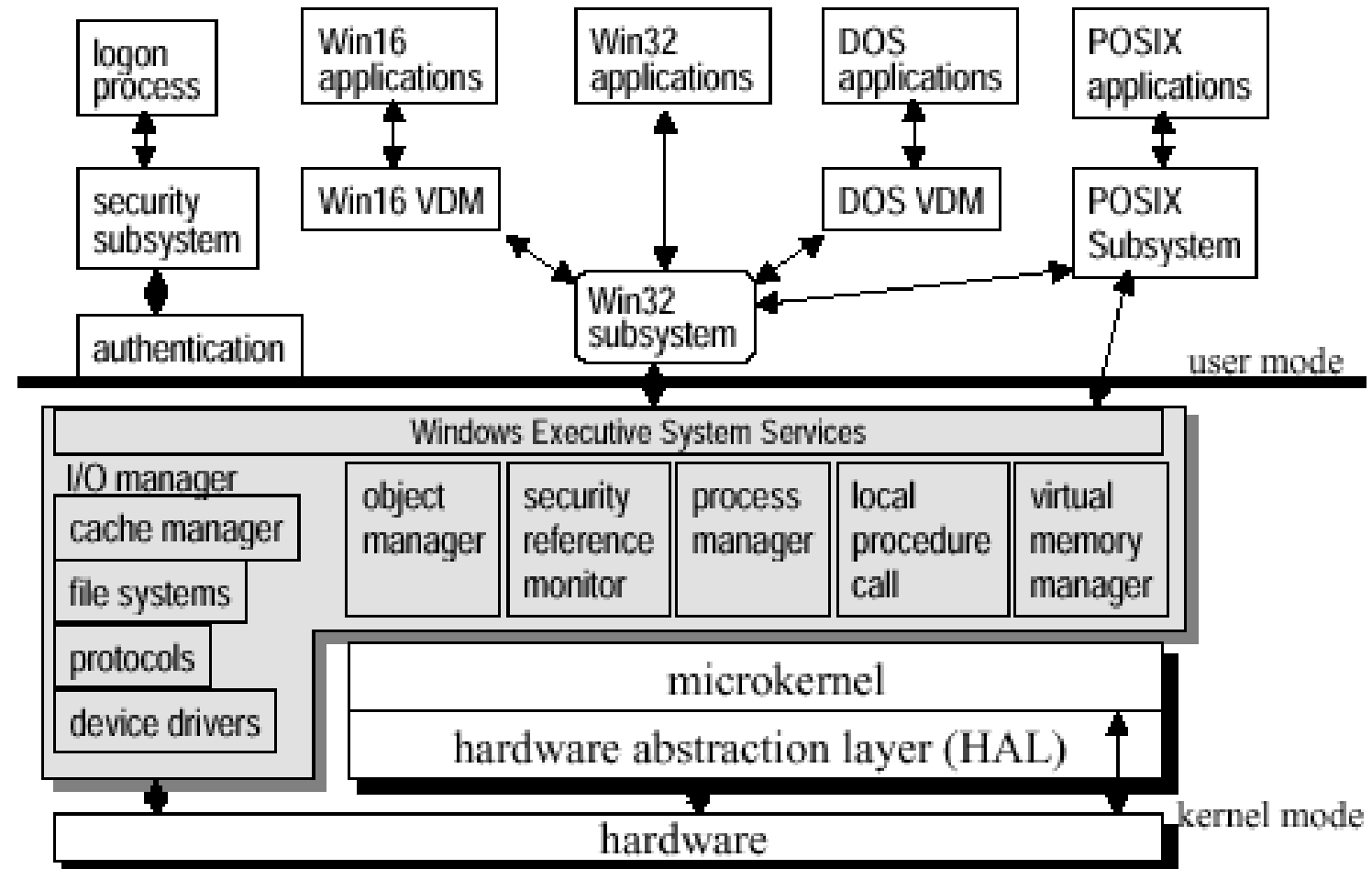
# MULTI-TASKING

- All modern OSs offer multitasking
- Multiple processes run "simultaneously",
- "Simultaneously" means that they are allocated time on the CPU to run their thread (s)
- or run on separate processor cores
- Sync issues
- OS provides scheduling
- OS provides memory isolation and collaboration between processes / threads



- Windows 1.0, 2.0, 3.0 and 3.1 are graphical interfaces for DOS
- Windows 95/98 includes DOS as a service
- Windows NT was a more portable version of Windows
- New core design from scratch. Not based on DOS
- Modular, 32 bit system for many simultaneous environments
- Robust mechanisms for critical processes
- Runs all Windows applications with the same interface
- In Windows XP, Windows 98 and Windows 2000 met
- Vista adds new security model and interface
- Windows 7 and Server 2008 are still the same kernel (NT)
- Windows 8 and 10 are the next step, but involve no / small changes in the kernel (Windows Vista is internally «Windows 6.0», Windows 10 is internally «Windows 6.3»...)

- System structure



- Registry
  - Centralized database for system configuration information
  - Continuation of DOS and Windows INI files
  - Used by various Win devices
  - Recognizer: Recognizes hardware at startup
  - Setup: System configuration tool
  - Kernel: CPU configuration and system load balancing
  - Drivers: Hardware initialization information

- Registry
  - Hierarchical structure
    - HKEY\_LOCAL\_MACHINE
    - HKEY\_CLASSES\_ROOT
    - HKEY\_CURRENT\_USER
    - HKEY\_USERS
    - HKEY\_CURRENT\_CONFIG
    - HKEY\_DYN\_DATA
  - Updates are not validated against other parts of the registry
  - Backup and restore are sensitive

---

#### REGEDIT4

```
[HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\explorer\Advanced\Folder\StartMenuScrollPrograms]
"Type"="checkbox"
"Text"="Multi-Column Start Menu"
"HKeyRoot"=dword:80000001
"RegPath"="Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced"
"ValueName"="StartMenuScrollPrograms"
"CheckedValue"=dword:00000000
"UncheckedValue"=dword:00000001
"DefaultValue"=dword:00000000
"HelpID"="update.hlp"
```

- NT File System (**NTFS**)
  - Great improvement over DOS 'FAT
  - Better features and performance
  - Data recovery, security, large files, compression
  - Metadata stored in Master File Table (MFT)
  - Data about data: How, when, by whom and formatting
  - NTFS is not just byte streams
  - MFT contains information about the data
  - File references are more than just addresses
  - 48 bit file number (pointer), 16 bit sequence number (multidisc)
  - Logging of transactions
  - NTFS is not further developed and is waiting for an heir

# OSX (prehistory)

- Apple also started with a "clean" DOS (Disk Operating System) 1976-1984.
  - Non-multitask, 1-level file system ("without directories").
- MacIntosh introduced in 1984
  - **Finds** as GUI «shell»
  - New file system after each : **H**ierarchical **F**ile **S**ystem
  - Toolbox for GUI app development

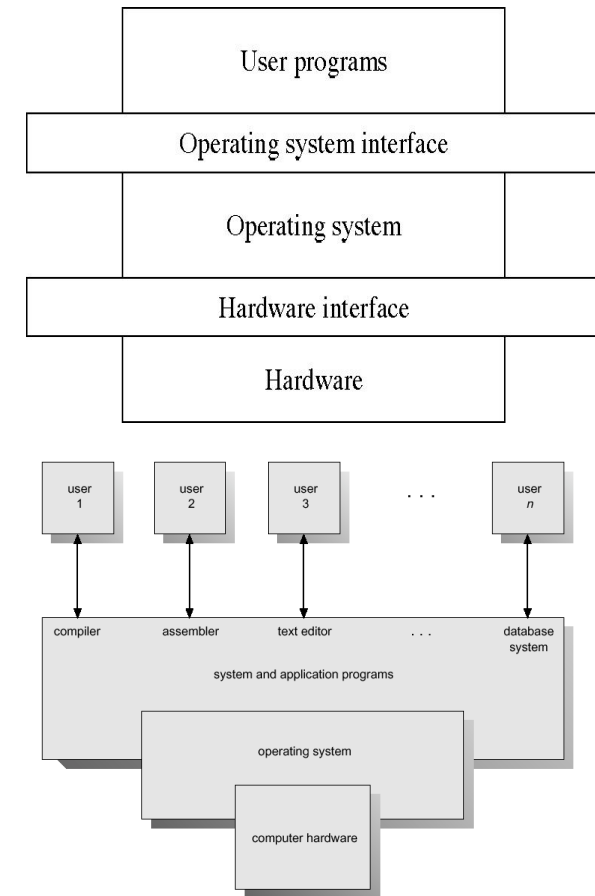
- Further development of NeXTSTEP.
  - Based on Mach micro core, FreeBSD (Unix) and I / O Kit driver package
  - The kernel is called Darwin / XNU (X is Not Unix)
  - Lots of Open Source thanks to the FreeBSD background

# Summary



# Operating system - general requirements

- The operating system must therefore organize the available resources and manage them in the best possible way for both machines and users
- Users will concentrate on applications without thinking about the machines' software and hardware ingenuity
- Users will preferably be introduced to an environment that is independent of existing hardware
- The training threshold should preferably be as low as possible



# What should we know?

- Define **process, thread and resource**.
- Explain how OS **abstracts** Hardware and thus offers a simpler programming and **user interface**.
- Be able to explain the OS's role as **resource** administrator: Process / thread-adm, Memory-adm, File-adm, Equipment-adm.
- Know what problems need to be solved to get **multitasking**
- Know the **memory layout** and **running pattern** of a regular program on an Intel-type processor.
- Explain the difference between **User** and **Kernel mode** on CPU

# EXERCISE

- Advanced operating system operations
  - TK1100\_F05\_Ø05\_OS\_Advanced.pdf
  - Archive files; ZIP, TAR.GZ, encrypted archives.
  - Install software (.MSI / .EXE on Windows, .DMG on Mac, TAR.GZ on Linux)
  - Use CMD (shell) in operating system
  - Set aside several hours for these exercises, and USE the tutors, for most people this is more advanced PC use than you are used to - and practical tasks for the rest of the semester are based on this!
  - Practical assignments come to the exam
- TK1102\_F05\_OS\_Øvingsoppgaver.pdf

# For optional self-study

For those who want to learn some topics more in depth to better understand, here are some extra topics related to today's teaching, it must be expected some personal work to understand these topics.

There will be no questions on the exam from these, and this is therefore not considered to be part of the syllabus.

# PC system organization

- Basic **IBM PC** from 1982
  - Mac used to have their own path, but had to change when they switched to Intel.

- Some PC systems still boot into 16 bit mode.

- max 1 MiB RAM
- limited instruction set and memory model
- only **BIOS** routines / interrupt.
- Goes to 32/64 bit only when the OS is loaded.



## Intel (x86) and AMD

Constructs and produces CPU, chipset etc.

Focus: **Desktop**, **server**, mobile, embedded.

**C**omplex **I**nstruction **S**et **C**omputing

## MIPS and ARM (apple)

Constructs and licenses CPU cores etc. to various manufacturers

Focus: **Embedded**, **mobile**

**R**educed **I**nstruction **S**et **C**omputing

Now x86 and ARM processors both have features of RISC and CISC

- More and more systems are now UEFI; but backward compatibility is taken care of.

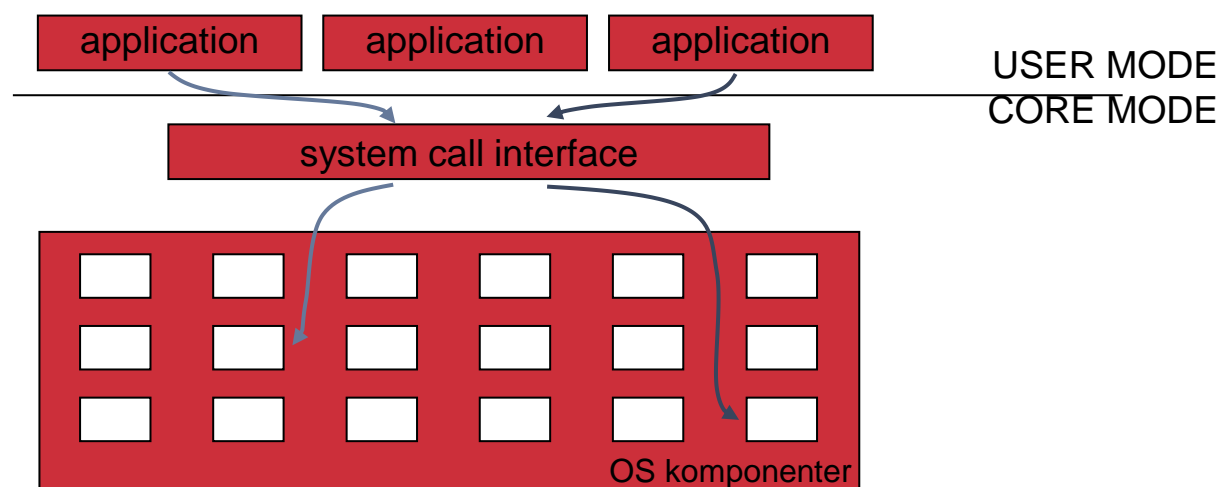
What is the boot process?

- To abstract means to remove (insignificant) details.
- For example: A **file** will (most often) consist of metadata (data about data)
  - A name
  - when created, last modified, o.l.
  - how big (KiB)
  - which sectors of the disk data is physically located on
  - ... and the content itself (data / instructions)
- The files are organized in directories (directories / folders))
  - another abstraction, which is also a **type of file** that contains metadata and addresses of other files.
- Physically / specifically, it is the only magnetized areas in addressed sectors on a disk / CD / memory stick
- To the user, this abstraction appears as symbols h @ n can click on.
- For the programmer, OS offers system calls that allow h @ n to create, open, close, position itself within, read from / write to

# System call

## Eksempel fra Linux:

- The **interface** between the OS and the users is defined by a **number of system calls**.
- Performing a system call is similar to a regular method call, but the system call runs kernel code:

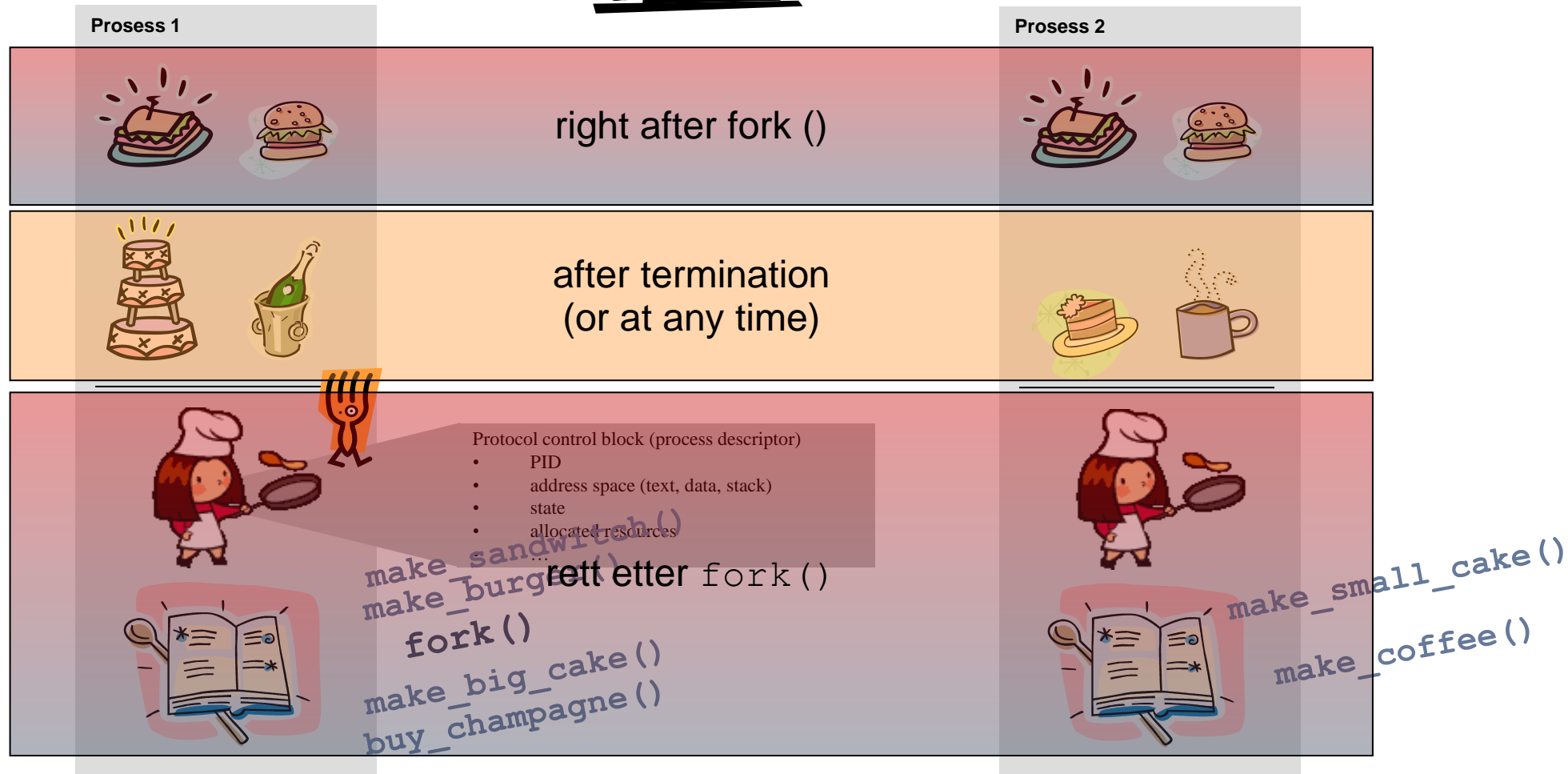


```

sys_acct(const char *name)
sys_acct(const char * filename)
sys_capget(cap_user_header_t header, cap_user_data_t dataptr)
sys_capset(cap_user_header_t header, const cap_user_data_t data)
sys_exit(int error_code)
sys_wait4(pid_t pid,unsigned int * stat_addr, int options, struct rusage * ru)
sys_waitpid(pid_t pid,unsigned int * stat_addr, int options)
sys_futex(void *uaddr, int op, int val, struct timespec *utime)
sys_sysinfo(struct sysinfo *info)
sys_getitimer(int which, struct itimerval *value)
sys_setitimer(int which, struct itimerval *value,
sys_sync(void); /* it's really int */
sys_syslog(int type, char * buf, int len)
sys_nice(int increment)
sys_sched_setscheduler(pid_t pid, int policy,
sys_sched_setparam(pid_t pid, struct sched_param *param)
sys_sched_getscheduler(pid_t pid)
sys_sched_getparam(pid_t pid, struct sched_param *param)
sys_sched_setaffinity(pid_t pid, unsigned int len,
sys_sched_getaffinity(pid_t pid, unsigned int len,
sys_sched_yield(void)
sys_sched_get_priority_max(int policy)
sys_sched_get_priority_min(int policy)
sys_sched_rr_get_interval(pid_t pid, struct timespec *interval)
sys_ni_syscall(void)
sys_setpriority(int which, int who, int niceval)
sys_getpriority(int which, int who)
sys_reboot(int magic1, int magic2, unsigned int cmd, void * arg)
sys_setregid(gid_t rgid, gid_t egid)
sys_setgid(gid_t gid)
sys_setreuid(uid_t ruid, uid_t euid)
sys_setuid(uid_t uid)
sys_setresuid(uid_t ruid, uid_t euid, uid_t suid)
sys_getresuid(uid_t *ruid, uid_t *euid, uid_t *suid)
sys_setresgid(gid_t rgid, gid_t egid, gid_t sgid)
sys_getresgid(gid_t *rgid, gid_t *egid, gid_t *sgid)
sys_setsuid(uid_t uid)
sys_setfsuid(gid_t gid)
sys_times(struct tms * tbuf)
sys_setpgid(pid_t pid, pid_t pgid)
sys_getpgid(pid_t pid)
sys_getpgrp(void)
sys_getsid(pid_t pid)
sys_setsid(void)
sys_getgroups(int gidsetsize, gid_t *grouplist)
sys_setgroups(int gidsetsize, gid_t *grouplist)
sys_newuname(struct new_utsname * name)
sys_sethostname(char *name, int len)
sys_gethostname(char *name, int len)
sys_setdomainname(char *name, int len)
sys_getrlimit(unsigned int resource, struct rlimit *rlim)
sys_old_getrlimit(unsigned int resource, struct rlimit *rlim)
sys_setrlimit(unsigned int resource, struct rlimit *rlim)
sys_getrusage(int who, struct rusage *ru)
sys_umask(int mask)

```

# Process created (Unix) - fork ()





# Virtual memory

- = **paging and swapping**
- The program is run instruction by instruction
  - Not all instructions need to be present in memory, only those to be run
- Abstract the memory usage!
  - The program itself thinks it has all the memory (**all the addresses**) available.
  - The program is compiled with internal (**logical**) memory addresses...
  - The program is divided into pages that are only loaded into memory if the address the page contains is referenced..
    - Reference to a page that is not loaded in memory triggers a **PAGE FAULT**.
  - CPU has a **MMU (Memory Management Unit)** that **translates** between program internal (**logical**) addresses and actual **physical** addresses in RAM (**page table**)
- If the memory becomes too full, memory pages can be displayed on disk (swapping).

# Compilation

```
#include <stdio.h>
```

```
int main() {
    printf("Hello World");
    return 0;
}
```

i C

```
MOV AH, 09
```

```
MOV DX, 109
```

```
INT 21
```

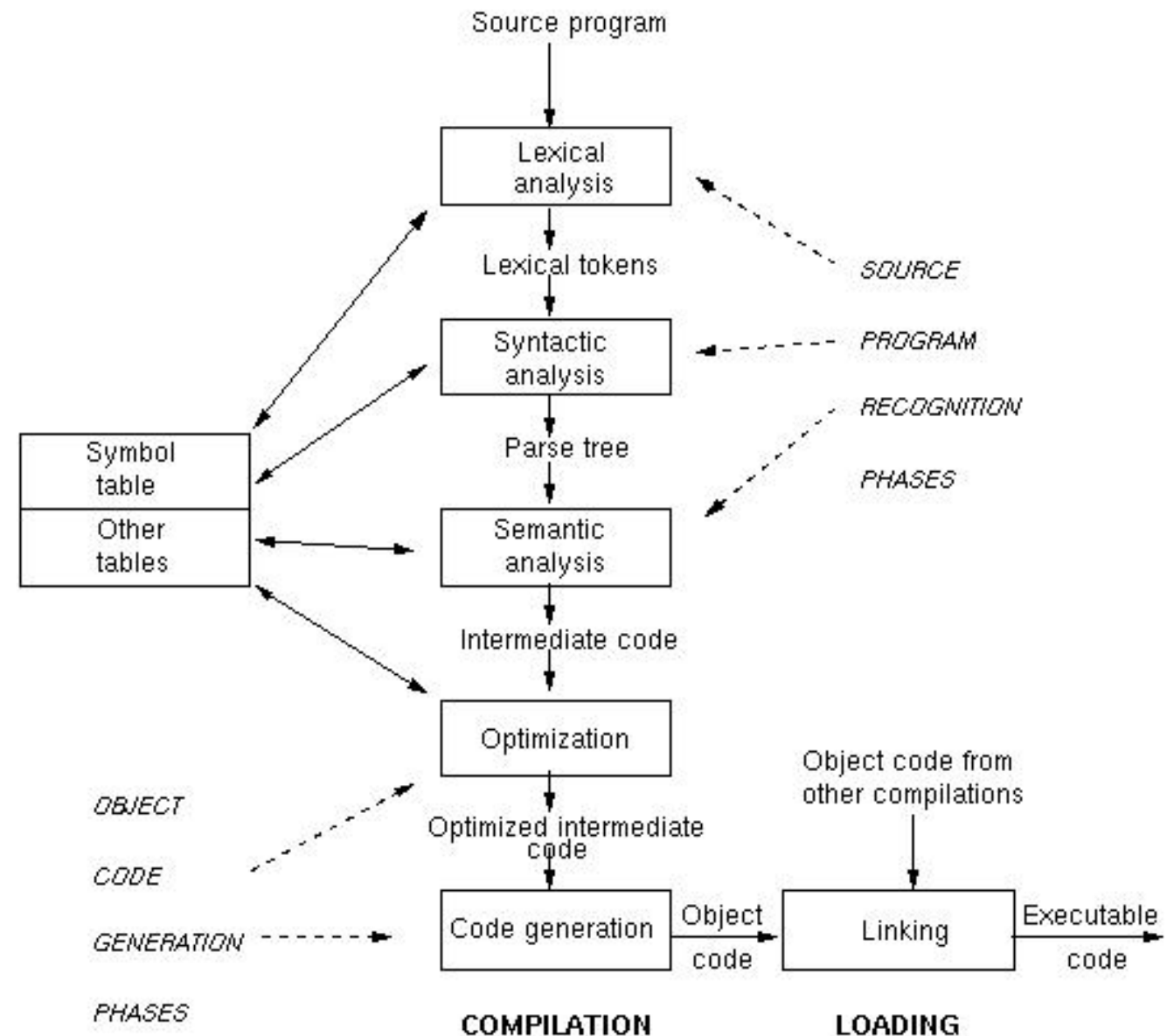
```
INT 20
```

```
DB 'Hello World!','$'
```

Assembly

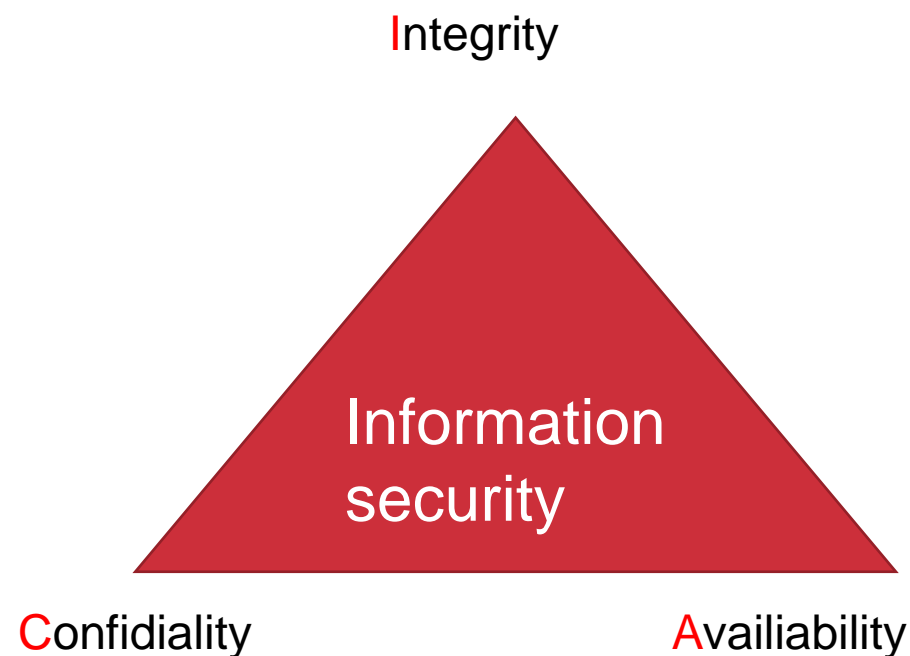
```
B4 09 BA 09 01 CD 21 CD
20 48 65 6C 6C 6F 20 57
6F 72 6C 64 21 24
```

```
..!!.. Hello W
orld!$. ....4.T.
```



# LEVELS for **safety** and measures

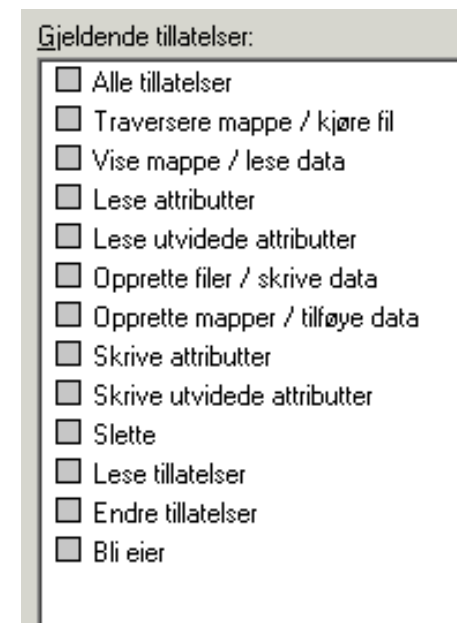
- Data
  - Access Control Lists, encryption (of file systems), ...
- Application
  - Patching, Certificates, Anti-virus,...
- Host machine
  - **OS**, authentication and authorization, useradm, group policies.....
- LAN (Internal nett)
  - IP-segmentering, IPSec, IDS, ...
- Boundary (Perimeter)
  - IDS, Firewall, VPN, NAT, ...
- Physical security.
  - Guard, locks and surveillance.
- Policy, procedures, awareness.
  - User training



# Security and file system

- Both Linux / OSX and Windows use different types of **ACL (Access Control List)**.
- Linux file systems share access rights(**e**xecute, **r**ead, **w**rite) based on the grouping
  - **u**ser
  - **g**roup
  - **a**ll
  - Directories (d) are just a special type of user
  - Only the user root has all rights / is administrator
- Windows (NTFS) has a more fine-grained and complicated system.  
Regular users are often set up as administrators as well

```
-rw-r--r-- 1 blistog users 256 Sep 15 2008 TUBE.COM
-rw-r--r-- 1 blistog users 148480 Sep 11 2008 UTF-8AA.jpg
drwxr-xr-x 3 blistog users 4096 Nov 18 2008 xhtml
```



# Operating system - I / O control

- All equipment connected to the computer is perceived as a resource by the operating system
  - Screen, keyboard, mouse, CD, speakers, .....
- The operating system must know how the computer will communicate with all the equipment (**drivers**)
  - Plug and play
- The operating system must present the equipment in a simple and straightforward manner, also for programming.
  - **UNIX / Linux**: "Pretend" everything is files that can be read / written to. (loadable modules)
  - **Windows**: "Pretend" everything is objects. (complicated object space, many levels of drivers)

# Interrupt (og Exception) Handling

- The IA-32 architecture has an IDT with pointers to 256 different interrupts and exceptions
- 32 (0 - 31) predefined and reserved
- 224 (32 - 255) user / OS defined
- Each interrupt has a pointer in the Interrupt table (IDT) and a unique index value that provides handling as follows
  1. process running, there comes an interrupt
  2. maintain driving condition, transfer control and find interrupt handling routine
  3. Execute interrupt handling
  4. retrieve aborted process
  5. continue execution

