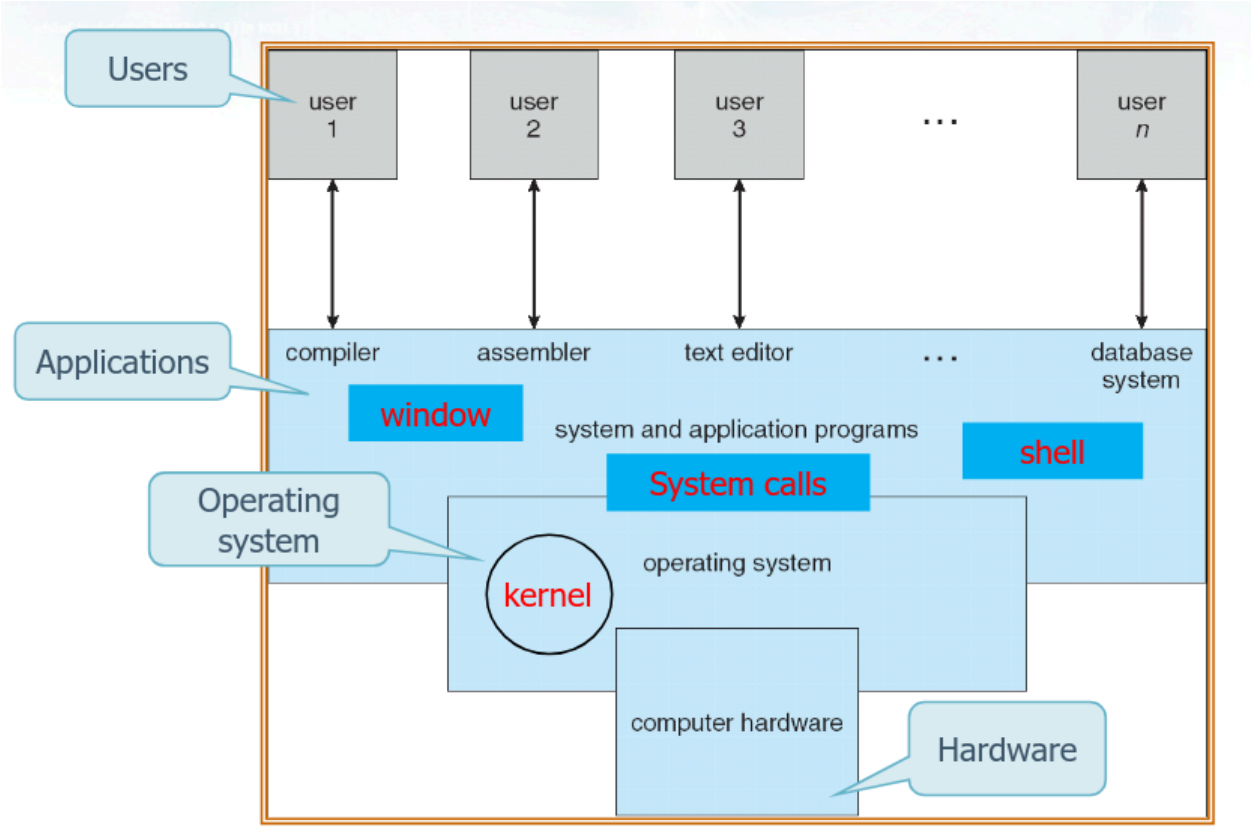
Components

An elaboration system is composed of the four following components:

1. Hardware:
   - Provides basic computing resources (CPU, memory, I/O devices)
2. OS:
   - controls and coordinates the use of the hardware among the various application programs for the various users
3. System and application programs
   - User services
4. Users:
   - People, machine, other computers



Its divided in two parts, kernal space and user space. The kernal space is made up of OS, kernel, drivers, hardware. User space is made up of users, applications and system programs.
These two spaces are connected by system calls.

Operating System

It's a software interface between user or application and hardware.
The goal is to execute command and programs, make system friendly to users, and use and share hardware efficiently.
It can be considered a VM that manages and allocates available resources or a "program" that controls the execution of user programs and operations of I/O devices.

Two views

It can be viewed top-down:

- Its an abstract manager of resources and information:
  - it abstracts information and resources
  - it allows to manage them in a simplifyed way
- Its also a software interface between system users and the hardware

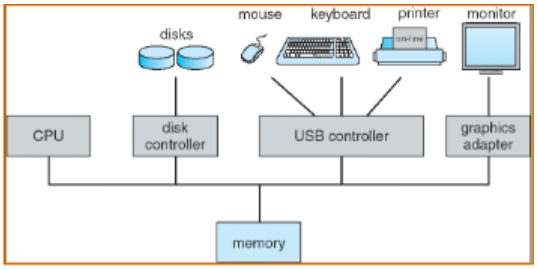
From a bottom-up view:

- A program that controls:
  - Devices and operations on devices
  - The execution of users programs
- It can be considered a set of modules, each of which provides services to the user.

Modules and Services

- **Command interpreter**
- **Process management**
- Main Memory Management
- Secondary Memory Management
- Management of I/O devices
- **File, and file system management**
- Implementation of protection mechanisms
- Network management, and distributed systems

(bold ones are anylized in the course)



Command interpreter

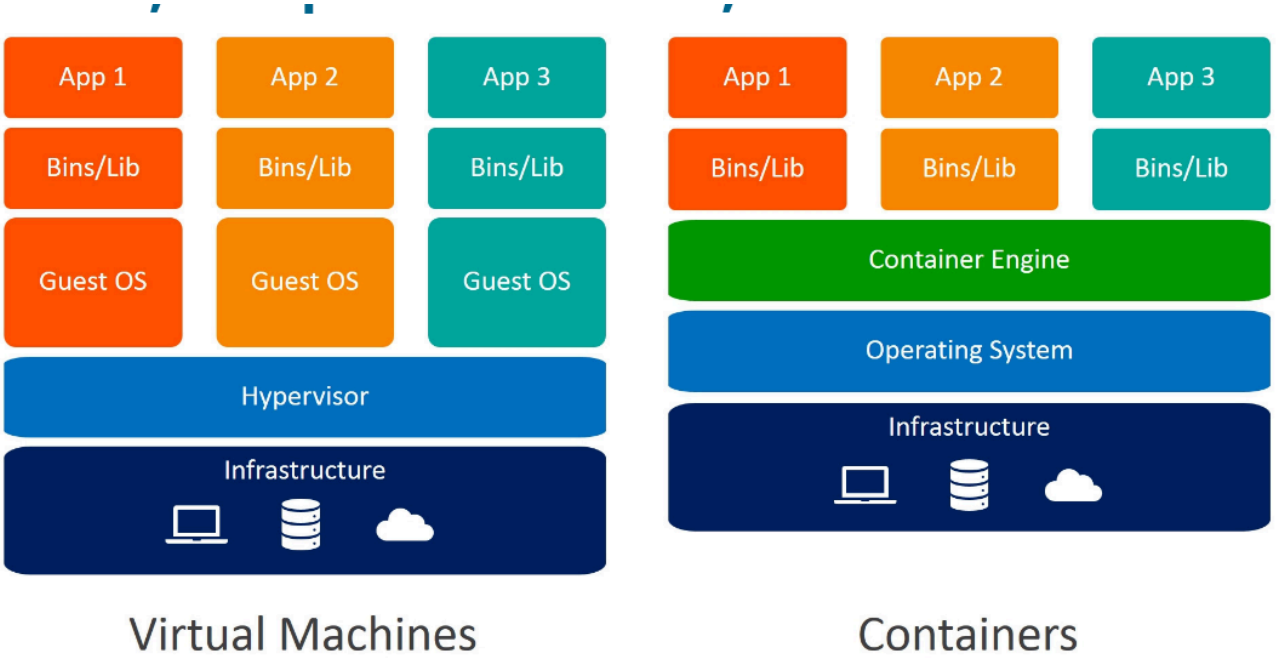
- The user and OS communicate through an textual or graphical interface
- The user performs its tasks through a command interpreter (shell)
- The OS allows the user to
  - Manage processes
  - Manage main and secondary memory
  - Establish protection policies
  - Manage the network and external connections

Process management

- A process (active unit) is a program (passive unit) in execution
- To run it requires resources
  - CPU, memory, devices, etc.
- The OS offers support for
  - Creating, suspending and deleting processes
  - Establishing communication mechanisms and synchronization among processes

Main memory management

- The data and instructions of a program must be in a region of main memory to allows a process to be executed



- Logically, main memory is a vector of elements (words)
- The OS must
  - Manage the use of memory (which regions are used and which are free)
  - Decide which processes to allocate in memory, and which can be eallocated
  - Optimize CPU access to memory

I/O devices management

- I/O devices cannot be managed directly by the users (complexity, driver, sharing, etc.)
- The OS must
  - Hide the details of a device to users by providing a uniform interface to the user
  - Providing read, write, control operations on devices

File, and file system management

- Data on secondary memory are organized into one or more file systems, which contain directories and files
- The OS must
  - Create, read, write, remove files and directories
  - Establish appropriate access protection mechanisms for data privacy and sharing
  - Optimize R/W operations

Implementation of protection mechanisms

- Protection indicates access control for users and processes to system resources
- The OS must
  - Define the access rights associated to users and resources
  - Distinguish between authorized and unauthorized use
  - Keep track of which users is using system resources

Network and distributed systems management

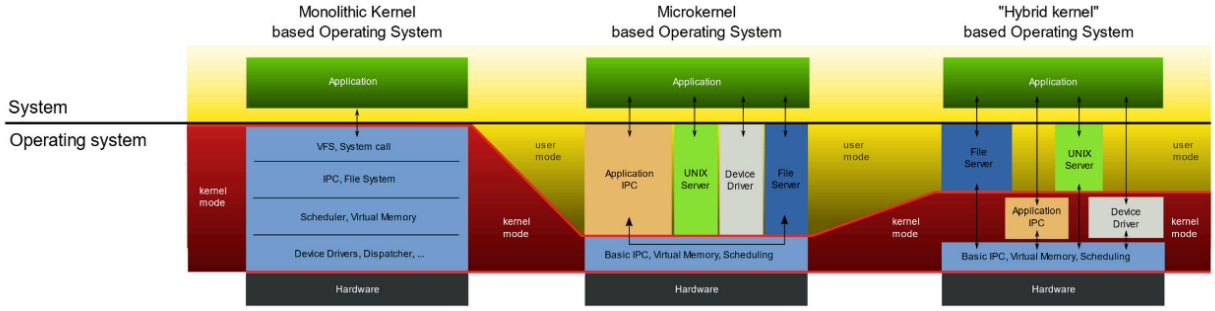
- A network is a collection of processors that do not share memory and clock
- The nodes of the network are interconnected by communication paths
- The OS must
  - Grant access to system resources
  - Increase the performance and reliability of the computing system, and the amount of data that can be processed

Terminology and basic concepts of OS

- Kernel, bootstrap, kernel protection, system call
- Login, shell
- Filesystem, filename, pathname, working directory, home directory, root directory
- Program (sequential and concurrent), process, thread
- Pipe
- Deadlock, livelock, starvation, polling (busy waiting)

Kernel

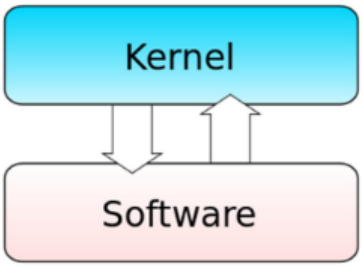
- Is the core of an OS
- It manages all system resources
  - In particular, it manages memory and processors
  - A program (or better module) always in execution
  - All other programs are system programs or applications
- There are different types of kernel
  - Microkernel that provides only basic functionalities
  - Monolithic kernel that provides functionalities through the device drivers (most common)
  - Modular kernel that is subdivided into levels
  - Esokernel, the systme has almost direct interact with the hardware, almost removing the OS
    This immage is a very good summary



Monolithic kernels

Think of this as a giant Swiss Army knife where every tool is part of the same handle.
The services are realized through a set of system calls made by separate modules but whose integration is very tight

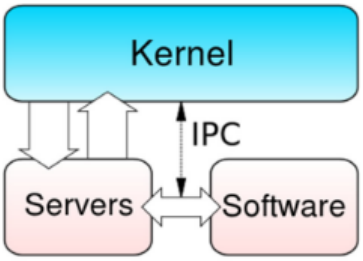
- Structure: All OS services (File System, VFS, Device Drivers, Memory Management) run in Kernel Mode.
- Drawbacks
  - A problem on a module can block the entire system
  - Adding a new hardware device involves adding its module to the kernel and recompile all the kernel (in modern kernel modules can be loaded at runtime)
- Advantages
  - Since everything is in the same memory space, components talk to each other directly via simple function calls. There is no overhead of switching "modes."
- Common solution
  - Unix, Linux, Open VMS, XTS-400



Microkernel

This follows the principle of "Least Privilege." The kernel is stripped down to the bare essentials.
It defines very small and simple software modules on top of the hardware that implement minimal services. It separates core service implementations from operating system operational structures.

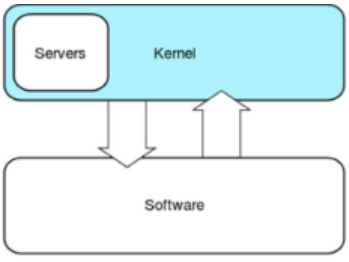
- Structure: Only the most critical tasks (Basic IPC, Virtual Memory, Scheduling) stay in Kernel Mode. Everything else (File Servers, Device Drivers) is moved to User Mode as separate processes.
- Drawbacks
  - To communicate, components must use **IPC (Inter-Process Communication)**, which requires expensive "context switches" between User Mode and Kernel Mode.
- Advantages
  - Very stable: If the File Server crashes, you can just restart it without the system crashing. It's also easier to port to new hardware
- Not common solution
  - March, L4, AmigaOS, Minix



Hybrid aproach

tries to get the best of both worlds.
Intermediate approach between the previous two. Microkernel with additional and "nonessential" kernel-level code, which can be executed very quickly. It's a compromise adopted by many developers.

- Structure: It looks like a microkernel because it is modular, but it runs some non-essential services (like drivers or the network stack) in Kernel Mode to reduce the IPC overhead.
- Drawbacks
  - Performance (slightly worse) but comparable with monolithic kernels
- Advantages
  - It integrates advantages of monolithic and microkernels
- Common solution
  - Windows , Netware, Mac OS, Dragonfly BSD



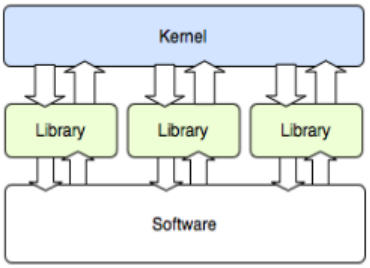
Esokernel

The Kernel's Only Job: To safely multiplex and protect resources. It ensures that App A doesn't overwrite App B's memory, but it doesn't tell App A how to manage its own memory.

Known as "vertical operating systems", it has Radically different approach to operating system design, with an almost direct access to hardware.
Extremely small and compact, as their functionality is arbitrarily limited to resource protection and multiplexing.

- Drawbacks
  - They involve more work in application development (libraries decrease this effort)
- Advantages
  - Less hardware abstraction
- Examples

- Nemesis, ExOS

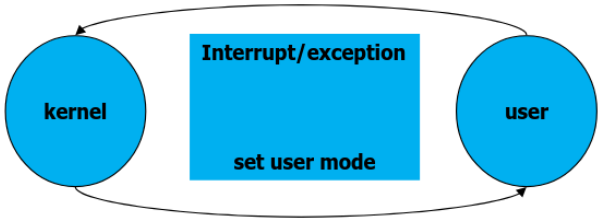


Bootstrap

Bootstrap or booting program.
Initialization program. Executes at power-on performing a proper check and initialization of the computer hardware, then it loads the kernel into main memory.
The bootstrap program is usually stored in ROM and EEPROM (firmware).

Kernel protection

A mode bit is added to computer hardware to indicate the current mode: kernel (0) or user (1). When an interrupt, exception, or fault occurs, hardware switches to kernel mode. Privileged instructions, that can be issued only in kernel mode.



Changing the content of a system register can only be done in kernel mode.

- Dual mode ensures that a user program cannot gain control of the computer in kernel mode
- Memory protection does not allow a user to write in kernel memory, e.g., store a new address in the interrupt vector.
  - Loading the memory protection registers is a privileged instruction
- Timer commonly used to implement time sharing
  - Loading the timer is a privileged instruction

System call

It is the interface with the services provided by the OS, i.e., it is the entry point of the OS. It's often implemented in assembler.
Often accessible with high level Application
Program Interface (API)

- Win32/64 API (for Windows)
- POSIX API (for UNIX, Linux, MAC OS X)
- Java API (for Java Virtual Machine)

How many system calls exist in an OS?

- UNIX 4.4 BSD: about 110
- Linux: between 240 and 260
- UNIX FreeBSD: about 320

The difference between system calls and functions are:

- Both provide services to users
- Any system call is typically coupled with one or more functions with the same name of the system call, and defined with a high-level programming language (e.g., C)
- Functions can be substituted and modified, system calls are kept stable over the years
- System calls provide basic functionalities (and they have a non-complex prototype), functions inside libraries provide more elaborate functionalities

Most common calls:

- Process management
  - `fork`, `wait`, `exec`, `exit`, `kill`
- File management
  - `open`, `close`, `read`, `write`, `lseek`, `stat`
- Directory management
  - `mkdir`, `rmdir`, `link`, `unlink`, `mount`, `umount`, `chdir`, `chmod`

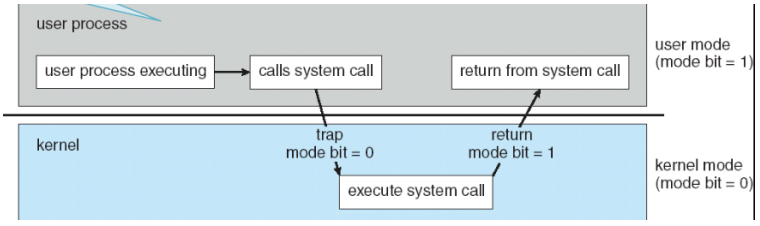
Example:
difference between windows and linux

```
int read (int fd, void *buffer, size_t nbytes);
```

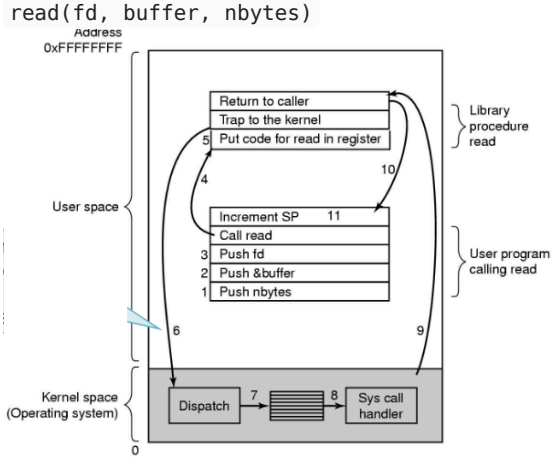
```
BOOL ReadFile (
     HANDLE fileHandle,
     LPVOID dataBuffer,
     DWORD numberOfByteToRead,
     LPDWORD numberOfByteRead,
     LPOVERLAPPED overlappedDataStructure
);
```

Example:
What happens when we make a system call
A system call causes an exception, and CPU switches to kernel mode (mode bit = 0)

- The exception, a software interrupts (or trap), activates the corresponding service routine
  - It verifies that the parameters are correct and legal, executes the request, and returns control to the instruction following the system call



Example:
`read(fd, buffer, nbytes)`



Example:
System calls versus library functions

- `printf` function uses system call `write`
- The allocation function `malloc` plausibly call the system call `sbrk`
- Data/time management
  - Only one system call `time`
  - The system call `time` provides the number of seconds since `01.01.1970`
  - Date and time are provided by different functions that produce different result formats

Login

To login you must provide

- Username
- Password
  - Passwords were usually stored in /etc/passwd

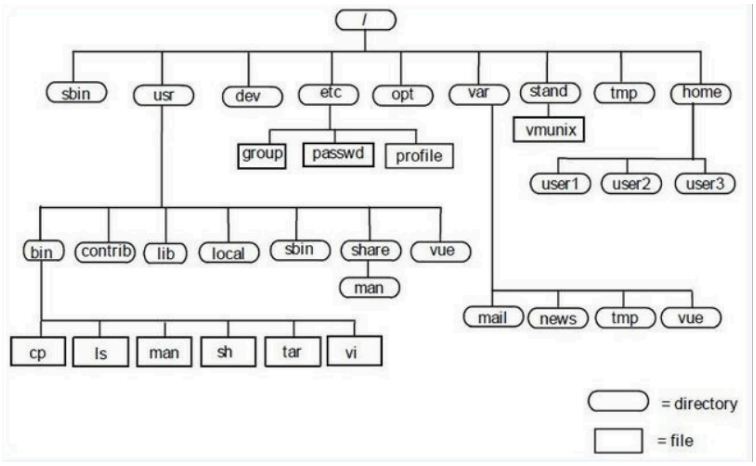
Shell

Its the command line interpreter, and its not directly part of the OS. It is a set of user space commands included in the OS distribution.
It Reads user commands and executes them. The commands are typed on the terminal or read from a "script file"
There are several shells:

- Bourne shell (sh)
- Bourne againg shell (bash)
- tcsh, ksh, zsh, etc.

File system

Hierarchical structure of directories and files.



Filename

There are few composition rules and length limitations (i.e., typical 255 bytes).

- In UNIX the only characters that cannot be used for a filename are:
  - slash `/`
  - character `null`

Pathname

A sequence of names separated by slashes ’/’

- `.` indicates the current directory
- `..` indicates the parent directory

Home directory

- Directory that is accessed after login operation
- It contains files and directories of the user that performed the login
- Identified by tilde ~ in UNIX-like systems
  - The home directory of user foo is usually `/home/foo`, which corresponds to `~` for that user
- Origin point to interpret absolute pathnames

Working directory

Origin point for interpreting relative pathnames

- Initially equal to the Home directory (i.e., ~)
- It can be changed by following the structure of the file system
- Owned by each process
  - i.e., each process has its own Working directory
- The reference to the Working directory is implicit, if any pathname is specified
  - directory/file1 is equal to ./directory/file1

Program

executable file that resides on disk:

- Passive entity
- Specifies a set of operations to execute a defined task

Sequential program

- Its operations are performed in sequence
- A new instruction starts at the end of the previous one (fetch - decode - execute)

Concurrent or parallel program

- Several statements can be executed in parallel
- An operation can be performed without waiting for the completion of the previous one

Atomic

An operation is atomic if it cannot be interrupted.
Why not always guarantee atomicity?
Atomicity is expensive to guarantee and slows down the entire computer
Example:

```
x++;
```

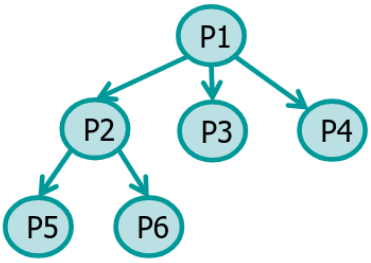
which is translated to:

```
add BYTE PTR [0x20], 1
```

If, in the meanwhile, another processor executes the same operation, one of the two increments can be lost. Atomicity guarantees it wont be lost.

Process

A running program (which includes a program counter, registers, variables, etc.). Is active entity. On UNIX systems, each process is characterized by a unique integer (positive) identifier
Process tree:

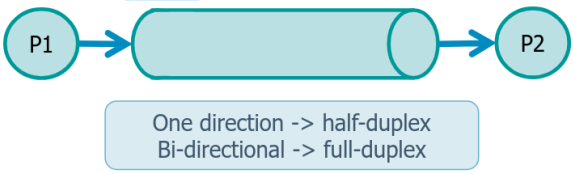


Threads

They are also called light process.
A process uses a set of resources. it can have one or more control streams running, each of these streames is a thread. Each thread, belong to a process, and shares its resources, but it has its own identifier, and "life".

Pipe

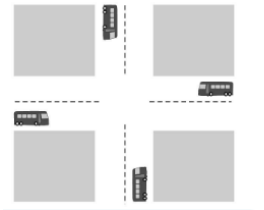
A pipe allows a communication data flow to be established between two processes
Typically, the channel is half-duplex (mono-directional)
● Communication in one direction from P1 to P2 or from P2 to P1



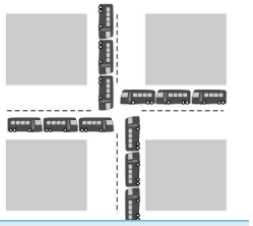
One direction -> half-duplex
Bi-directional -> full-duplex

Deadlock

A deadlock is a situation in which entities (processes) sharing the same resource wait indefinitely an event, which is caused by other entities, resulting that one or more entities are blocked forever.



Potential deadlock



Deadlock

Livelock

Called also active deadlock.
Situation similar to the deadlock in which the entities are not actually blocked but do not make any progress because they are too busy responding to each other to resume work.

Examples:

- Two people meeting in a corridor and trying to pass, repeatedly move from one side to the other of the corridor
- Two units perform polling (busy waiting) to check the status of the other and do not show progress (mutual livelock), but they are not in deadlock because each is doing the poll operation

Starvation

Access to a resource needed for its progress is repeatedly refused to an entity.
Starvation does not imply deadlock because while an entity may starve other can progress
Instead, deadlock imply starvation. No entity can progress, consequently all are in starvation.

THE LAST PPT COMPARING Unix MacOs and Windows has been skipped. Highly uinnteresting and unimportant, long live the penguin.