Overview

Introduction

Computer systems have two major components:

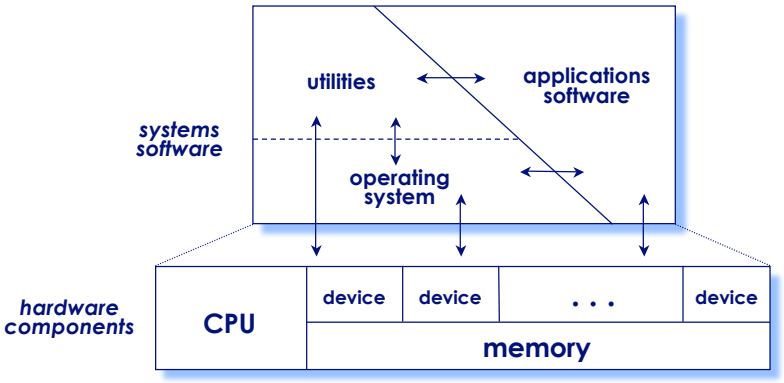
- *hardware*—electronic, mechanical, optical devices.
- *software*—programs.

Without support software, a computer is of little use. With its *software*, however, a computer can store, manipulate, and retrieve information, and can engage in many other activities.

Software can be grouped into the following categories:

- *systems software* (operating system & utilities)
- *applications software* (user programs)

Contemporary Hardware/Software Structure



*What is an Operating System

It is an extended machine

- Hides the messy details which must be performed
- Presents user with a "virtual machine", easier to use

It is a resource manager

- Each program gets **time** with the resource
- Each program gets **space** on the resource

What is *an* operating system?

An operating system (OS) is a *resource manager*.

It manages the resources of a computer, including processor(s), main memory, and I/O devices.

An operating system provides orderly and controlled allocation and use (i.e., *sharing*) of the resources by the users (jobs) that compete for them.

One major function of an operating system is to "hide" the complexity of the underlying hardware and give the *user* a better view (an abstraction) of the computer.

Why we study operating systems?

Likely will NOT actually write an OS, but…

- one of the largest and most complicated software system
- draws on lots of areas:
  - *software engineering, computer architecture, data structures, networks, algorithms.*
- if certain things (in an OS) need to be changed, better understand it first!
- can apply techniques used in an OS to other areas:
  - *interesting, complex data structures*
  - *conflict resolution*
  - *concurrency*
  - *resource management*

In the beginning...

The earliest computers, developed in the 1940s, were programmed in *machine language* and they used front panel switches for input. In fact, the programmer was also the operator interacting with the computer directly from the system console (control panel).

Problems:

- programmers needed to sign-up in advance to use the computer one at a time.
- executing a single program (often called a *job*) required substantial time to setup the computer.

*History of Operating Systems

First generation 1945 - 1955

- vacuum tubes, plug boards

Second generation 1955 - 1965

- transistors, batch systems

Third generation 1965 – 1980

- ICs and multiprogramming

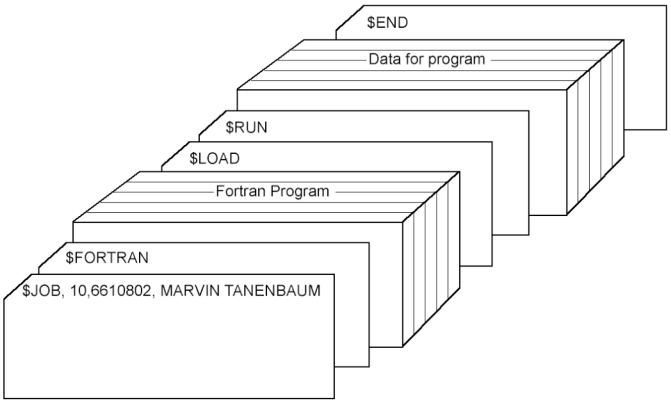
Fourth generation 1980 – present

- personal computers

Next generation ??

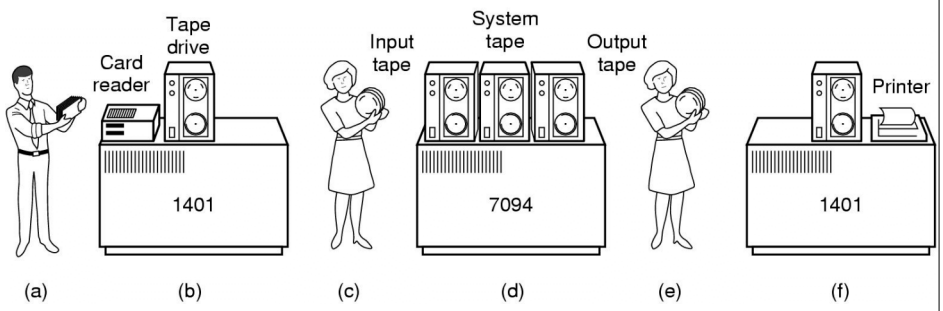
- personal digital assistants (PDA), information appliances

*History of Operating Systems



Structure of a typical FMS job – 2nd generation

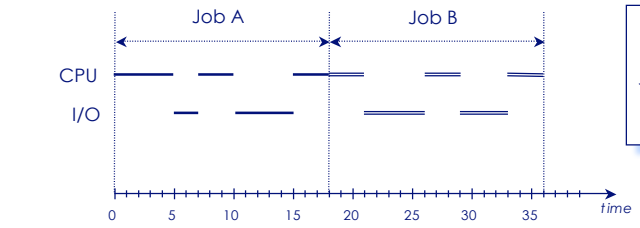
*History of Operating Systems



Early batch system

- bring cards to 1401
- read cards to tape
- put tape on 7094 which does computing
- put tape on 1401 which prints output

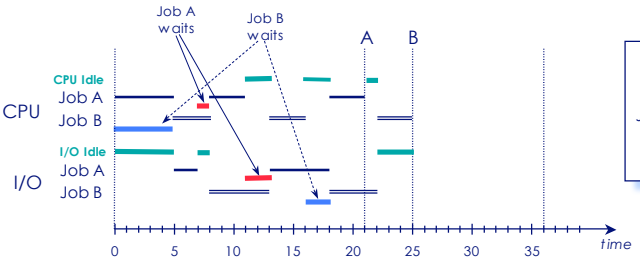
Job interleaving



uniprogramming

Jobs A and B end at time 36.

CPU is idle 47%
I/O is idle 53%

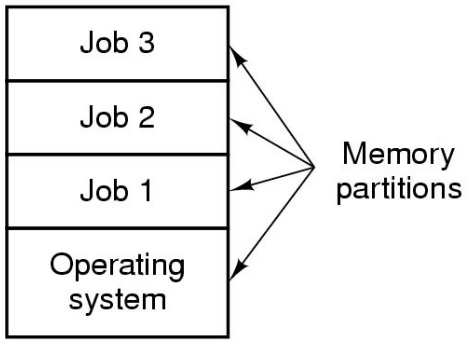


"pure" multiprogramming

Jobs A and B end at time 25.

CPU is idle 20%
I/O is idle 36%

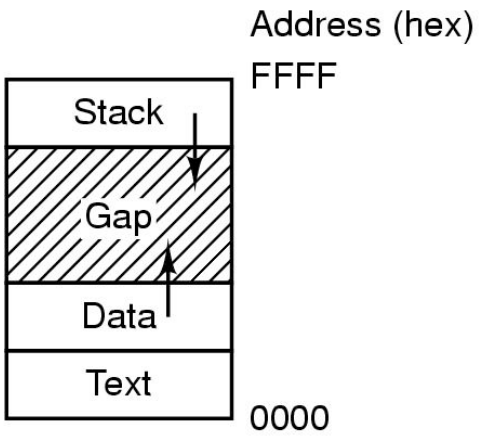
*History of Operating Systems (4)



Multiprogramming system

- three jobs in memory – 3rd generation

*Memory Layout



Processes have three segments: text, data, stack

Other systems

Personal computing
- single-user, dedicated.

Parallel processing
- multiprocessors (share a common bus, clock, and memory).
- tightly-coupled; multiprocessing.

Distributed processing
- multicomputers (do not share memory and clock); loosely-coupled.

Real-time
- deadline (time critical) requirements.
- soft real-time; hard real-time.

Interrupts and ...

The operating system gets the control of the CPU (which may be busy waiting for an event or be in a busy loop) when either an *external* or an *internal* event (or an exception) occurs.

- **external events**:
  - Character typed at console.
  - Completion of an I/O operation (controller is ready to do more work).
  - Timer: to make sure operating system eventually gets control.

An *interrupt* is the notification of an (external) event that occurs in a way that is *asynchronous* with the current activity of the processor. Exact occurrence time of an interrupt is not known and it is not predictable.

... Traps

- **internal events**:
  - System call.
  - Error item (e.g., illegal instruction, addressing violation).
  - Page fault.

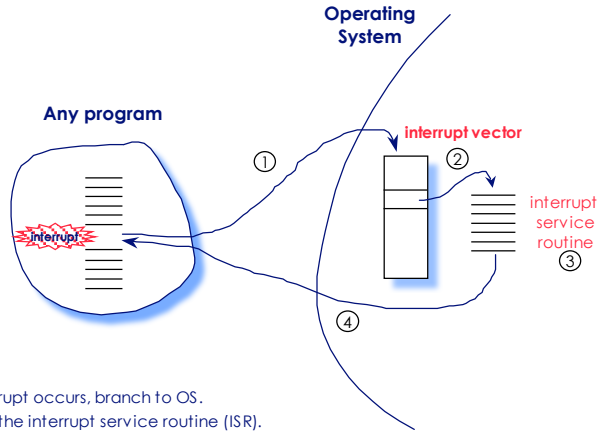
A *trap* is the notification of an (internal) event that occurs while a program is executing, therefore is *synchronous* with the current activity of the processor.

Traps are immediate and are usually predictable since they occur while executing (or as a result of) a machine instruction.

More on interrupts

- Systems that generate interrupts have different priorities for various interrupts; i.e., when two interrupts occur simultaneously, one is serviced "before" the other.
- When a new "higher priority" interrupt occurs while lesser interrupt is being serviced, the current handler is "suspended" until the new interrupt is processed. This is called the "*nesting of interrupts*."
- When interruption of an interrupt handler is undesirable, other interrupts can be "*masked*" (inhibited) temporarily.

Interrupt handling by "picture"



1. An interrupt occurs, branch to OS.
2. Locate the interrupt service routine (ISR).
3. Execute the ISR
4. Return to interrupted program

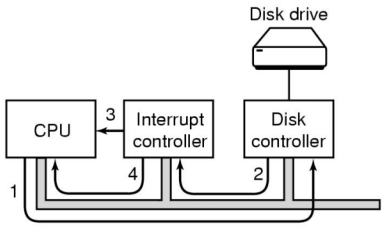
Interrupt handling by "words"

When the CPU receives an interrupt, it is *forced* to a different context (kernel's) and the following occur:

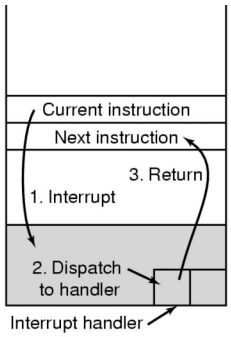
- the current state of the CPU (PSW) is saved in some specific location.
- the interrupt information is stored in another specified location.
- the CPU resumes execution at some other specific location—the interrupt service routine.
- after servicing the interrupt, the execution resumes at the saved point of the interrupted program.

Although the details of the above differ from one machine to another, the basic idea remains the same: *the CPU suspends its (current) execution and services the interrupt.*

*Computer Hardware Review (6)



(a)



(b)

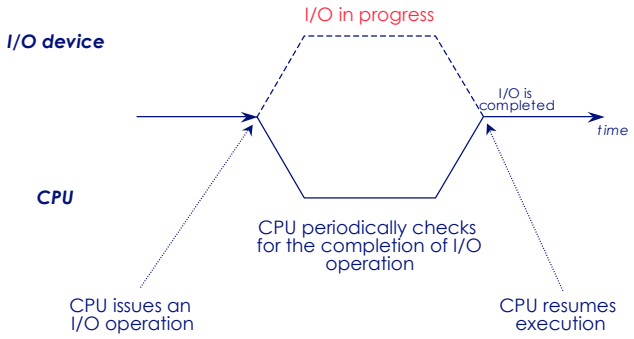
(a) Steps in starting an I/O device and getting interrupt

(b) How the CPU is interrupted

I/O techniques

Programmed I/O

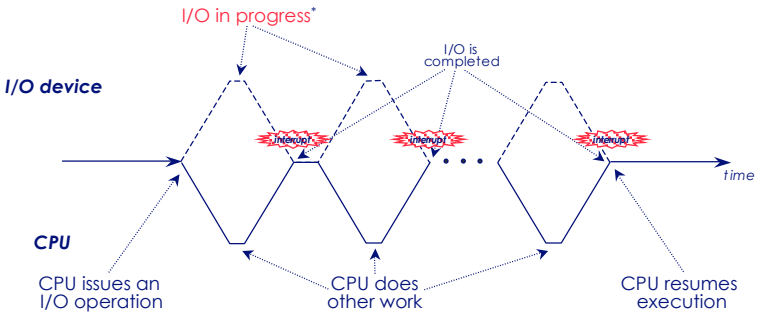
The CPU transfers the data from (or to) the device buffers. After issuing an I/O operation the CPU continuously checks (polls) for its completion.



I/O techniques *cont.*

Interrupt-driven I/O *(slow speed, character device)*

The CPU issues an I/O operation and goes on to some other work. The device notifies (interrupts) the CPU as each byte or word arrives. Again, the CPU handles the data transfer.

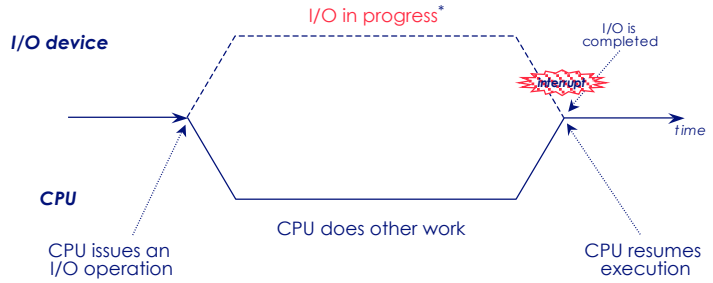


*One unit of data (a byte or a word) is transferred.

I/O techniques *cont.*

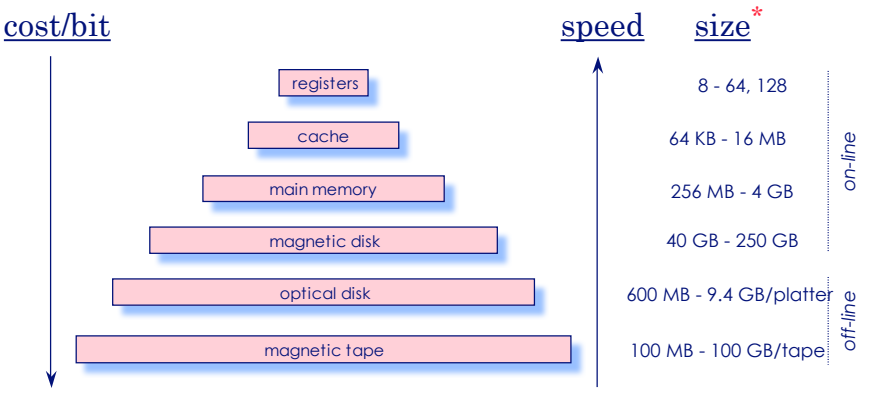
Direct Memory Access (DMA) *(high speed, block device)*

CPU issues an I/O operation specifying the device, the memory location of the data, and the block size. The CPU is now free to do work for others. The DMA device interrupts the CPU upon the completion of the requested operation.



*A block of data is transferred.

Storage structure and hierarchy



This hierarchy also measures relative capacity of the devices. However, the capacity difference at the lower levels (e.g., between a magnetic disk and a magnetic tape) is narrowing rapidly.

* These values are for current single processor systems or single disk and tape drives.

Architectural support

Modes of operation
- supervisor (protected, kernel) mode: *all* (basic and privileged) instructions available.
- user mode: a *subset* (basic only) of instructions.

I/O protection
- all I/O operations are privileged.

Memory protection
- base/limit registers (in early systems).
- memory management unit, MMU (in modern systems).

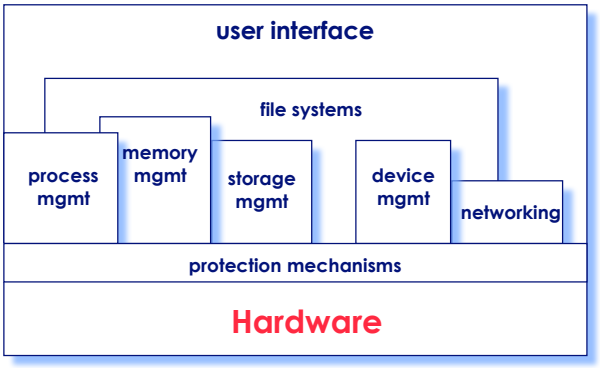
CPU control
- timer (alarm clock); time-quantum.
- context switch.

Operating system components

An operating system generally consists of the following components:

- Process management
- (Disk) storage management
- Memory management
- I/O (device) management
- File systems
- Networking
- Protection
- User Interface

OS architecture



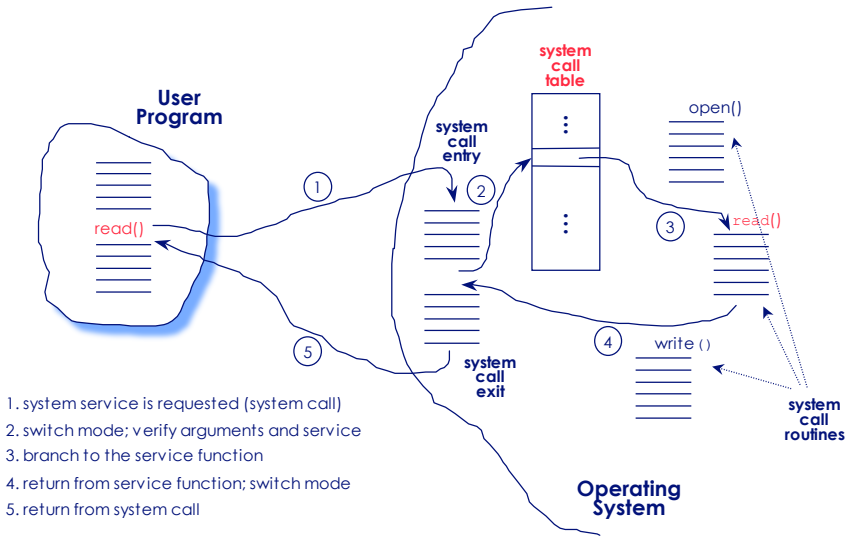
Accessing OS services

The mechanism used to provide access to OS services (i.e., enter the operating system and perform a "privileged operation") is commonly known as a *system call*.

The (only) difference between a "procedure call" and a "system call" is that a system call changes the execution mode of the CPU (to *supervisor mode*) whereas a procedure call does not.

System call interface: A set of functions that are called by (user) programs to perform specific tasks.

System call mechanism



1. system service is requested (system call)
2. switch mode; verify arguments and service
3. branch to the service function
4. return from service function; switch mode
5. return from system call

System call groups

- Process control
  - `fork()`, `exec()`, `wait()`, `abort()`
- File manipulation
  - `chmod()`, `link()`, `stat()`, `creat()`
- Device manipulation
  - `open()`, `close()`, `ioctl()`, `select()`
- Information maintenance
  - `time()`, `acct()`, `gettimeofday()`
- Communications
  - `socket()`, `accept()`, `send()`, `recv()`

Utilities: user commands

- File manipulation
  - *cp, mv, cat, tar, sort, compress, gzip*
- File modification
  - *vi, emacs, od*
- Status information
  - *date, df, who, users*
- Programming language support
  - *gcc, tcl, perl, yacc, lex, rcs*
- Program loading, execution and debugging
  - *ld, gdb*
- Communications
  - *telnet, ftp, netscape, mail, ssh, scp*

Applications software

- Typesetting and word processing
  - Latex, Troff, FrameMaker, MS Word
- Database management
  - Oracle, Sybase
- Spreadsheets
  - Lotus 1-2-3, MS Excel
- Graphics
  - XV, CorelDraw, MS PowerPoint
- Games
  - Tetris, Chess, Xsokoban
- Internetworking
  - Netscape, Lynx, Arena, WEB authoring tools

Bootstrapping

- The process of initializing the computer and loading the operating system is known as *bootstrapping*.

  This usually occurs when the computer is powered-up or reset.

- The initial loading is done by a small program that usually resides in non-volatile memory (e.g., EPROM).

  This in turn loads the OS from an external device.

- Once loaded, how does the operating system know what to do next?

  It waits for some event to occur: e.g., the user typing a command on the keyboard.

OS kernel

During "normal" operations of a computer system, some portions of the operating system remain in main memory to provide services for critical operations, such as dispatching, interrupt handling, or managing (critical) resources.

These portions of the OS are collectively called the *kernel*.

$$\text{Kernel} = \text{OS} - \underset{\textit{comes and goes}}{\text{transient components}}$$

remains

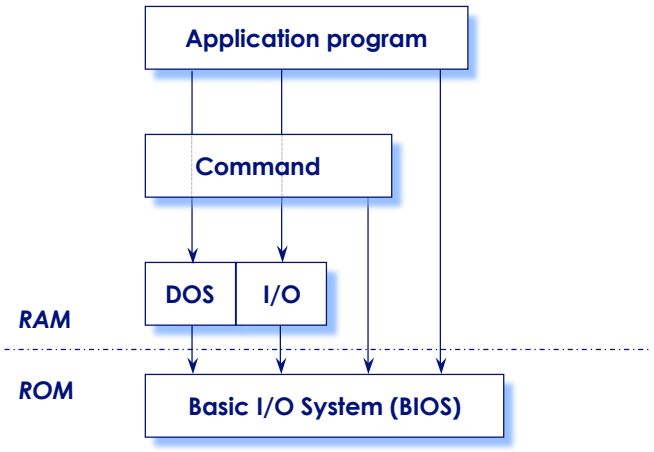
System structure

An operating system is usually large and complex. Therefore, it should be engineered carefully.

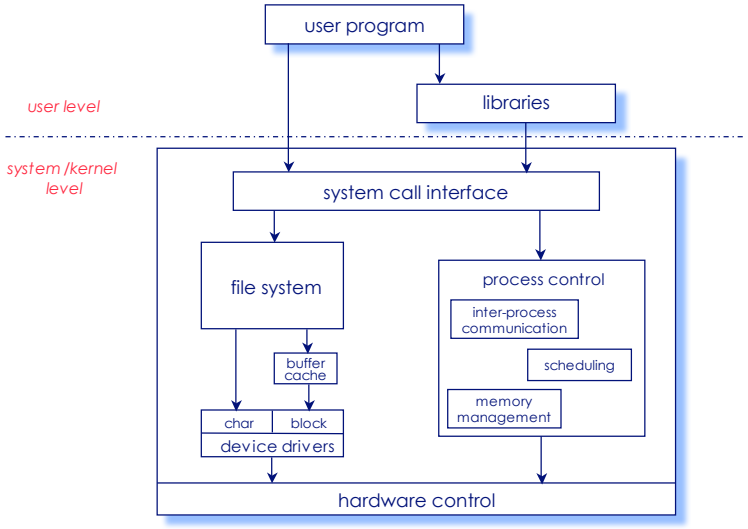
Possible ways to structure an operating system:

- Simple, single-user
  - *MSDOS,* MacOS, Windows
- Monolithic, multi-user
  - *UNIX*, Multics, OS/360
- Layered
  - *T.H.E. operating system*
- Virtual machine
  - *IBM VM/370*
- Client/Server (microkernel)
  - *Chorus/MiX*

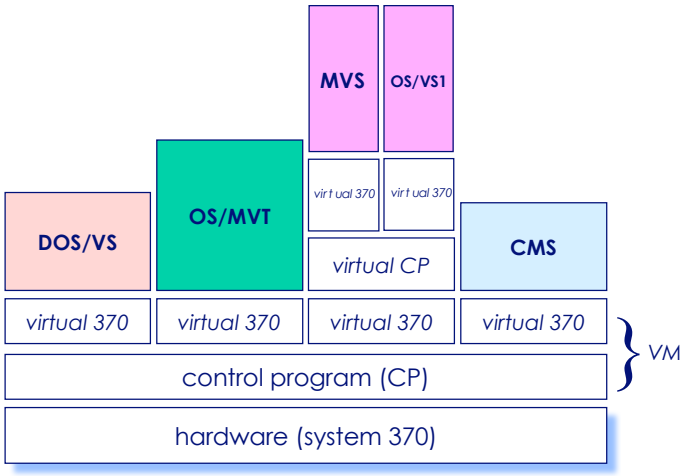
Structure of MSDOS



UNIX system/kernel structure



Structure of IBM VM/370



Structure of Chorus/MiX