Operating systems

Salvatore Andaloro

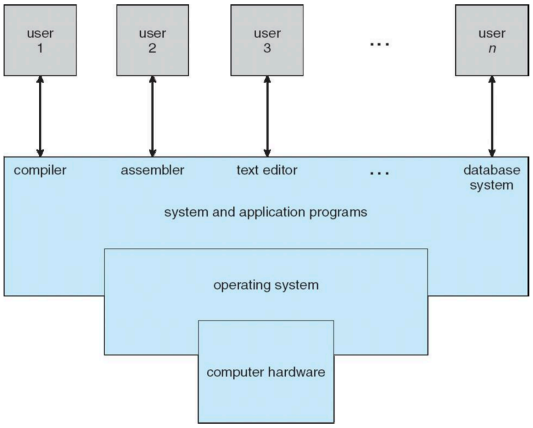
Contents

1 Introduction 2
1.1 Computer startup 2
1.2 General computer architecture 2
1.2.1 Interrupts 2
1.2.2 Storage 3
1.3 Modern system architectures 3
1.3.1 Difference between multiprocessor and multi-core 4
1.3.2 Clustered systems 4
1.3.3 Multi-programmed systems 5
1.3.4 Process management 5
1.3.5 Memory management 5
1.3.6 Storage management 5
1.3.7 I/O management 6
1.4 OS protection 6
1.5 Computing environments 6
1.6 Services provided by operating systems 7
1.7 System calls 7
1.7.1 Parameter passing 8
1.8 OS design 8

1 Introduction

An operating system is a program that acts as an intermediary between the user and the hardware. The main goals of an operating system are to:

- execute programs that solve problems
- make the computer easy to use
- use the hardware in an efficient manner



To do so it coordinates access to the hardware among the different applications and users.

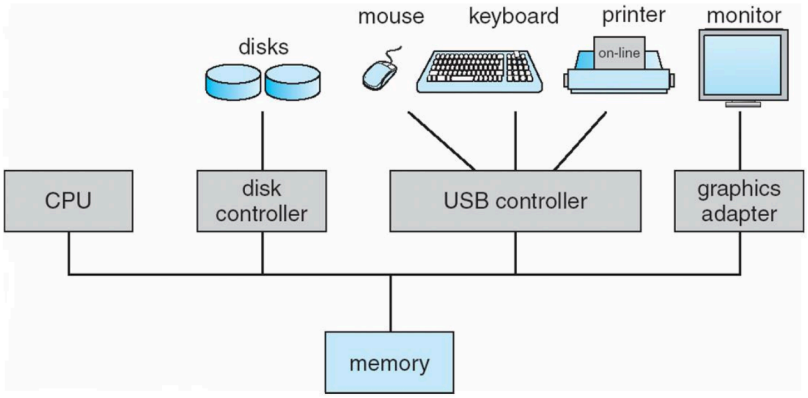
The program that runs all the time is the **kernel**, everything else is either a system program (ships with the operating system) or an application program.

1.1 Computer startup

The first program that runs on startup is the **bootstrap program**. This is stored in ROM or EEPROM and is usually called **firmware**. It initializes registers, memories and device controllers and loads the kernel into the main memory. The kernel starts **daemons**, i.e. different programs from kernels but that must be started at startup. For example in UNIX there is *systemd*, that in turn starts other daemons.

1.2 General computer architecture

The general model of a computer architecture is the following: there are one or more CPUs and device controllers that are connected through a common bus with a shared memory.



1.2.1 Interrupts

In interrupt driven operating systems, device controllers communicate with the CPU using **interrupts**. While waiting for the CPU device controllers store information in local buffers, that are then read/written by the CPU and the stored data is transferred from/to the main memory.

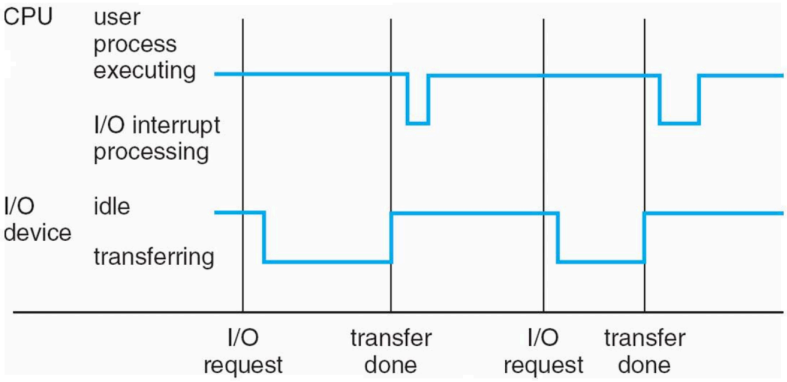


Figure 3: Interrupt timing diagram

When an interrupt happens, the OS saves the current program counter (PC), jumps to the routine that handles the interrupt and then resumes normal execution by jumping back to where the PC was pointing to. The position to where to jump to when an interrupt is received is stored in the **interrupt vector or table**, which is a map where each interrupt is linked to an instruction memory address.

A **trap** or **exception** is a software-generated interrupt caused by an error or a user request.

Depending on the importance of the interrupt, some interrupts must be handled immediately, while others can wait. The first ones are called **non-maskable**, while the latter are **maskable**.

While transferring data, the CPU receives an interrupt when every chunk of data has been successfully received. This can be very very wasteful, therefore a feature called DMA (Direct Memory Access) has been introduced. In this technique the CPU receives an interrupt only after all the data has been transferred successfully.

1.2.2 Storage

Storage systems are categorized by speed, cost and volatility. Each storage systems has therefore its advantages and disadvantages, therefore there is no “best” storage device.

The main memory of a computer, which can be accessed directly by the CPU is DRAM (dynamic random access memory, based on charged capacitors) or SRAM (static random access memory, based on inverters), which is usually volatile. Secondary storage has much bigger capacity and is non-volatile (ex. hard disks, solid-state drives).

Each storage system has a device controller and a device driver. The driver provides an uniform interface between the controller and the kernel.

1.2.2.1 Caching

Caching is a very common technique for speeding up access to commonly used data. Information is copied from slower to faster storage temporarily and then every time that information is needed the OS will first check if it is present in cache. Due to cost constraints, cache is very often smaller than the storage being cached, therefore cache management must be properly optimized.

1.3 Modern system architectures

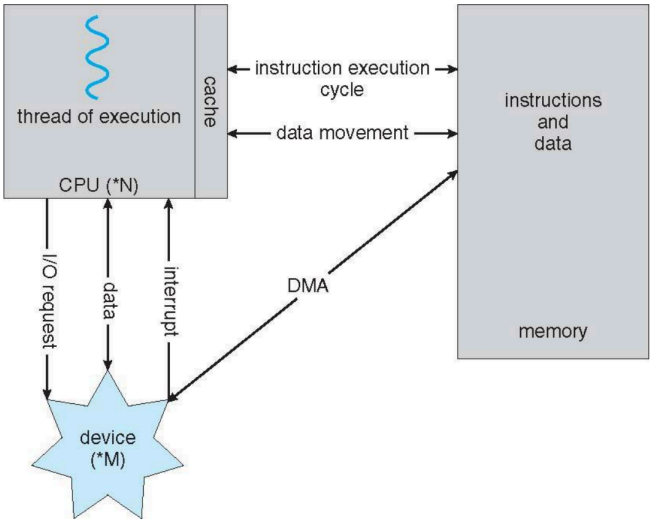


Figure 4: A Von-Neumann architecture

Currently most systems are multiprocessor and/or multi-core. In these systems, we can allocate tasks to processors in different ways:

- asymmetric processing: each processor/core is assigned a specific task
- symmetric processing: each processor/core performs all tasks

1.3.1 Difference between multiprocessor and multi-core

Multiprocessor systems have multiple processors with a single CPU and share the same system bus and sometimes the clock. Multi-core systems have a single processor that contains multiple CPUs.

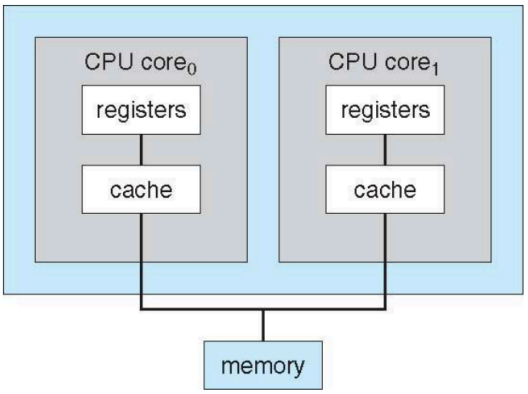


Figure 5: A multi-core processor

Multi-core systems are more widespread because they usually consume less power than multiprocessor systems and because on-chip buses are faster.

1.3.2 Clustered systems

Clustered systems are systems composed of multiple machines that usually share the same storage via a storage-area network (SAN). These systems provide a high-availability service that can survive failures of single machines.

- Symmetric clustering: all machines can run tasks and they monitor each other. If a machine fails the other can take over.

- Asymmetric clustering: each machine is assigned to a specific set of tasks. If a machine fails another machine that was turned on and in "hot-standby mode" takes over.

1.3.3 Multi-programmed systems

The OS can run multiple tasks on the same CPU by using a technique called multiprogramming (batch system): the OS organizes jobs so that the CPU has always one ready to execute. When a job has to wait (for example for I/O) the OS switches to another job. This is called job scheduling. Timesharing (multitasking) is an extension of this technique where the OS switches so frequently among different tasks that the user doesn't notice and can interact with all applications at the same time. This is needed for "window" based systems, where the user can see multiple things are the same time.

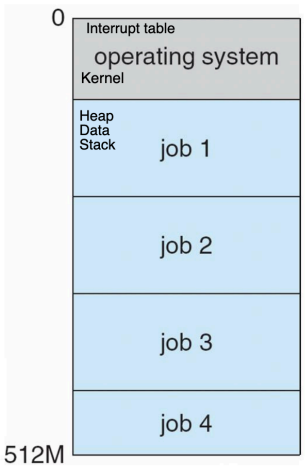


Figure 6: Memory layout for multiprogramming systems

The OS and users share the same hardware, devices and software resources. To protect the system and avoid that different jobs can write in some areas of the memory a privilege system is established. In dual-mode systems jobs can be run in user mode or kernel mode. Some instructions are allowed only for kernel mode systems. Intel processors have for modes of operation, where 0 is fully privileged and 3 is fully restricted.

1.3.4 Process management

A process is a program in execution. The *program* is a passive entity, while the *process* is an active entity. The life of the process is generally managed by the operating system. Single-threaded processes have a **program counter** specifying the location of the next instruction to execute. Instructions are executed sequentially, until the end fo the program is reached. Multi-threaded process has one program counter per thread. If a system has more cores, each core has its own program counter.

1.3.5 Memory management

To execute a program, the instructions must be in memory. Memory management is handled by the operating system and has the following goals:

- Keeping track of which parts of memory are currently being used and by whom
- Deciding which processes (or parts thereof) and data to move into and out of memory
- Allocating and deallocating memory space as needed

1.3.6 Storage management

The OS provides a logical view of the storage and abstracts the physical properties in **files**. Files are organized in directories and there usually is an access control system. The OS deals with:

- Free-space management
- Storage allocation

- Disk scheduling

The memory is therefore organized in a hierarchy, where each level offers different access speeds. While transferring data from a level to another, the OS must ensure that the data stays consistent. Moreover, multiprocessor environment must provide cache coherency in hardware such that all CPUs have the most recent value in their cache.

1.3.7 I/O management

The OS hides the peculiarities of hardware devices from the user using I/O subsystems. These subsystems are responsible for the device-driver interfaces and memory management of I/O including buffering (storing data temporarily while it is being transferred), caching (storing parts of data in faster storage for performance), spooling (the overlapping of output of one job with input of other jobs).

1.4 OS protection

OS must provide mechanisms to defend the system against external attacks. An attack is anything posing a threat to:

- Confidentiality
- Availability
- Integrity

For example OS distinguish among users, where each has a specific set of privileges. Privilege escalation is an attack where a user can gain privileges of a more privileged user.

1.5 Computing environments

There exist many computing environments, such as:

- Stand-alone general purpose machines
  - Network computers (thin clients)
  - Mobile computers
- Real-time embedded systems: operating system that runs processes with very important time constraints
- Cloud computing
  - Client-server computing
  - Peer-to-peer computing
- Distributed computing: many systems connected together over a network
- Virtualization: guest OS emulates another OS or hardware and runs software on it. The program that manages this is called VMM (Virtual machine manager).

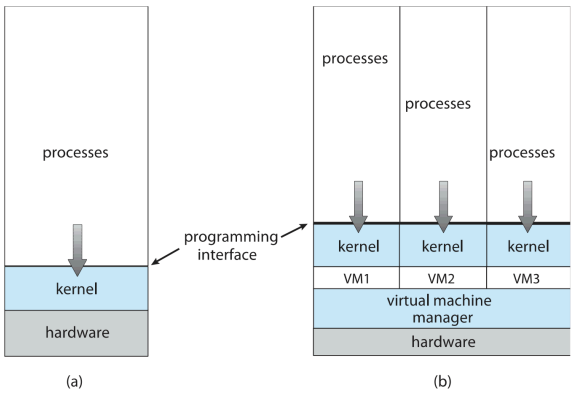


Figure 7: Virtualization

1.6 Services provided by operating systems

Operating systems provide the following services:

- User interface: can be command-line (CLI), Graphics User Interface (GUI), Batch
- Program execution - The system must be able to load a program into memory and to run that program
- I/O operations
- File-system manipulation
- Communication between processes
- Error detection: errors may occur in CPU and memory hardware, in I/O devices, in user program
- Resource allocation: when multiple users or multiple jobs running concurrently, resources must be allocated to each of them
- Accounting: to keep track of which users use how much and what kinds of computer resources
- Protection and security

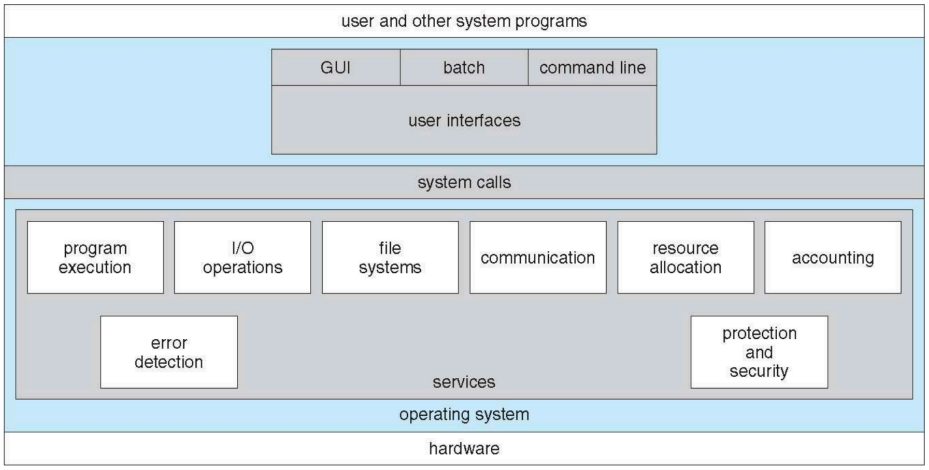


Figure 8: Services provided by an operating system

1.7 System calls

System calls are an interface provided by the operating system to interact with it. They are mostly accessed by using a high-level API provided by a language such as C, C++ etc. In this way developers can use a single API that works on all operating systems and leave the actual system call to the underlying library written for that specific platform. The high-level API can also check for errors before calling the system call.

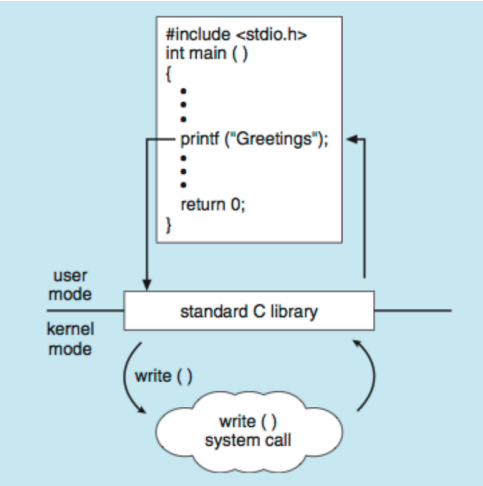


Figure 9: C program invoking printf() library call, which calls write() system call

1.7.1 Parameter passing

A system call usually requires some parameters, for ex. the *open_file* system call needs to know the name of the file. Parameters can be passed using predefined specific registers. Often there are not enough registers for all required parameters, so parameters can be also stored in memory in a table and just the address of the table is put into the register.

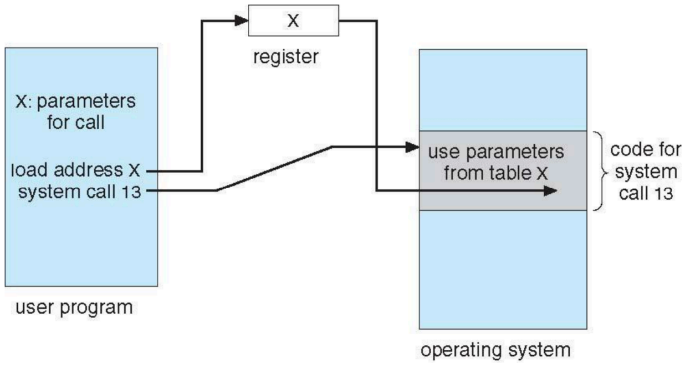


Figure 10: Parameter passing

Examples of system calls:

- Process management: create, terminate, load, execute, get process attributes, set process attributes, wait time, wait event, signal event, dump memory on error, single step executing for debugging, locks for managing shared data
- File management: create, open, delete, read, write
- Device management: request device, release device, read, write, get device attributes, set device attributes
- Information maintenance: get time or date, set time or date, get system data, set system data
- Communications: send/receive messages, open/close connection, gain access to shared memory
- Protection: control access to resources, get and set permissions, allow/deny user access

1.8 OS design

When designing an operating system, the following aspects need to be taken into consideration:

- User goals: friendly, reliable, safe, fast
- System goals: easy to design, modular, error-free, flexible, efficient