Tema 1
Estructuras de Sistemas Operativos

1. Recordatorio de TOC y FS.
2. Componentes de un SO.
3. Estructuras/Arquitecturas de los SO's.
4. SO's de propósito específico.

Sistemas Operativos
Dpto. Lenguajes y Sistemas Informáticos, 2025

Objetivos

- Conocer el **soporte hardware** (HW) para el SO.
- Conocer los requisitos funcionales de un SO (**Servicios del SO**).
- Conocer diferentes formas de diseñar un SO (**Estructura/Arquitectura del SO**) y los beneficios de cada una.
- Distinguir diferentes tipos de **SO's de propósito específico**.

Bibliografía

- [Sta2005] W. Stallings. “Sistemas Operativos. Aspectos Internos y Principios de Diseño” (5/e). Prentice Hall, 2005.
- [Car2021] Jesús Carretero. “Sistemas Operativos. Una Visión Aplicada. Volumen I” (3/e). 2021.
- [Car2015] Jesús Carretero. “Problemas de Sistemas Operativos”. 2015.
- [Sta2018] W. Stallings, Operating Systems. Internals and design principles (9th ed.). Pearson Education, 2018.

Tema 1
Estructuras de Sistemas Operativos

1 **Recordatorio de TOC y FS**

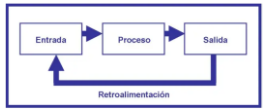
Componentes de un SO

Estructuras/Arquitecturas de los SO’s

SO’s de propósito específico

Sistemas Operativos
Dpto. Lenguajes y Sistemas Informáticos, 2025

Sistema informático (Computer System)



Usuario

Superusuario

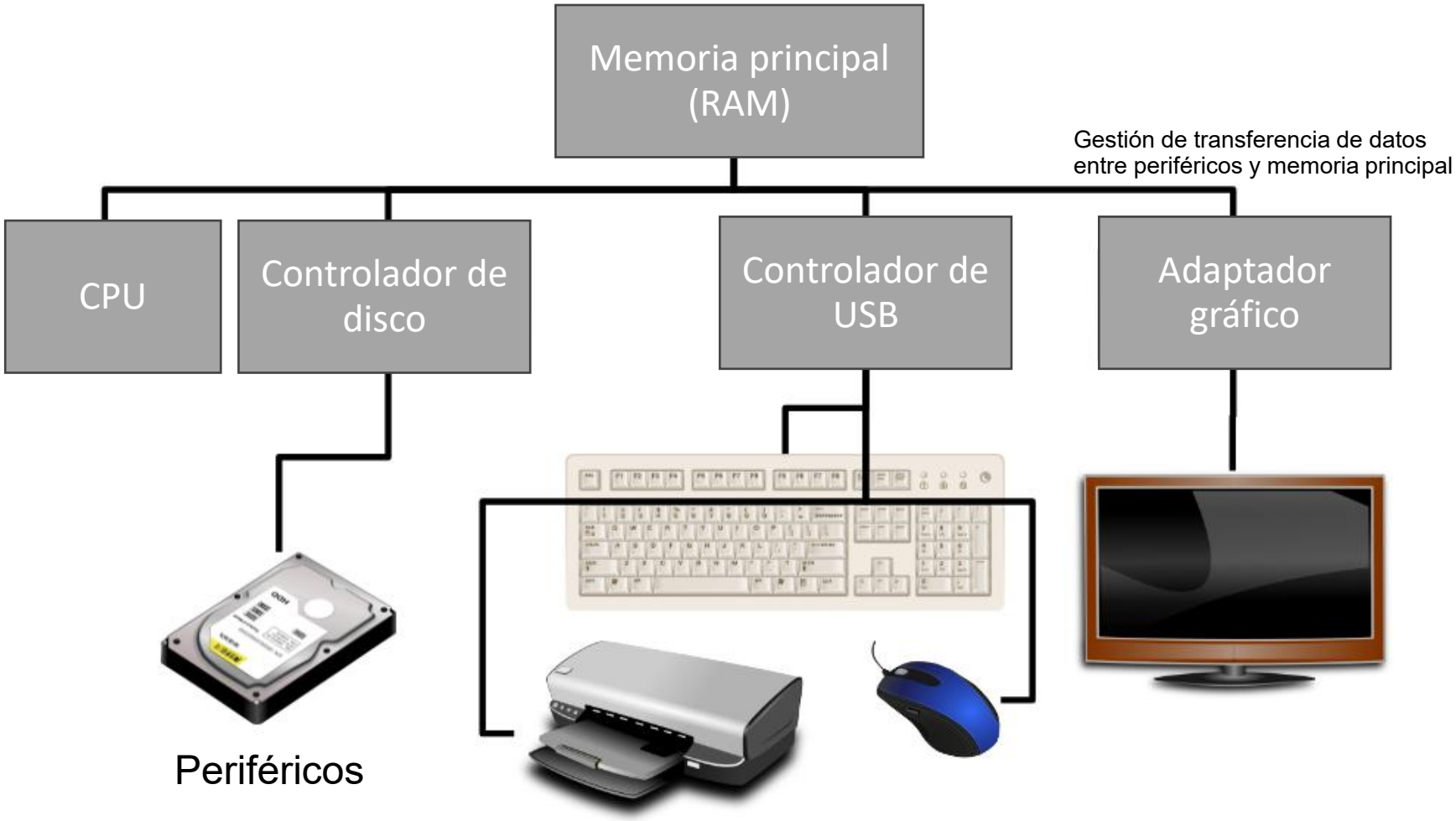
Aplicaciones y programas del sistema

Sistema operativo (SO)

Hardware (HW)

Procesar
Almacenar
Gestionar
(información)

Sistema informático. Áreas funcionales



HW. Principales componentes

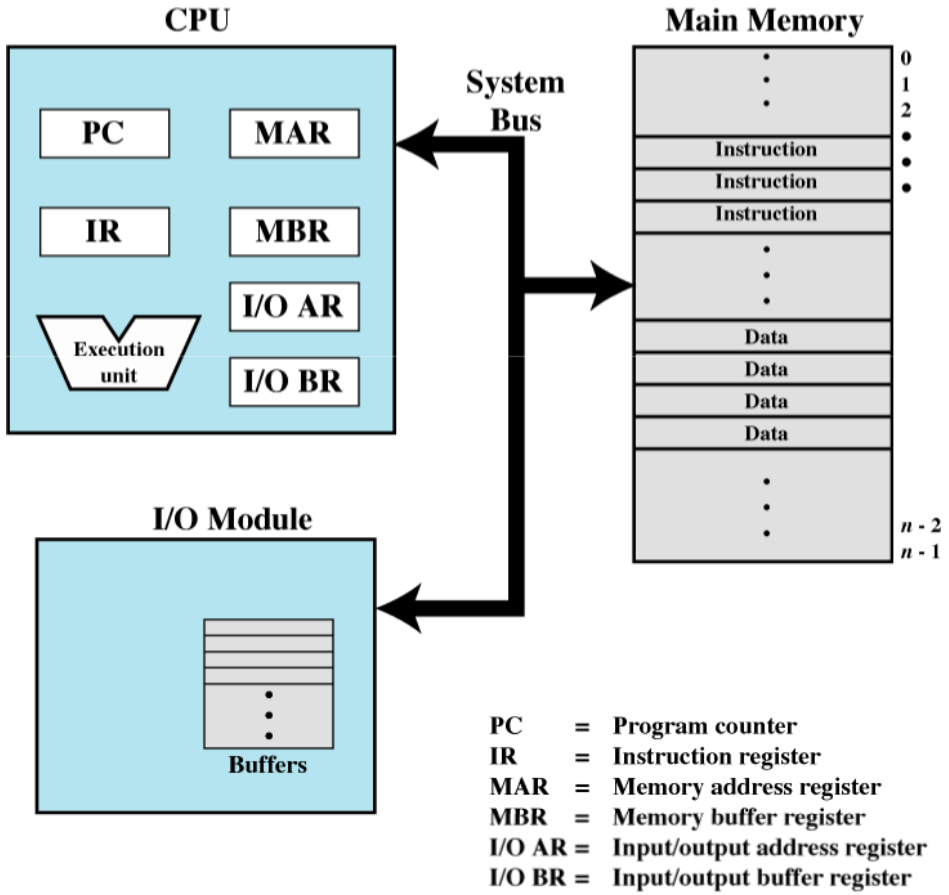
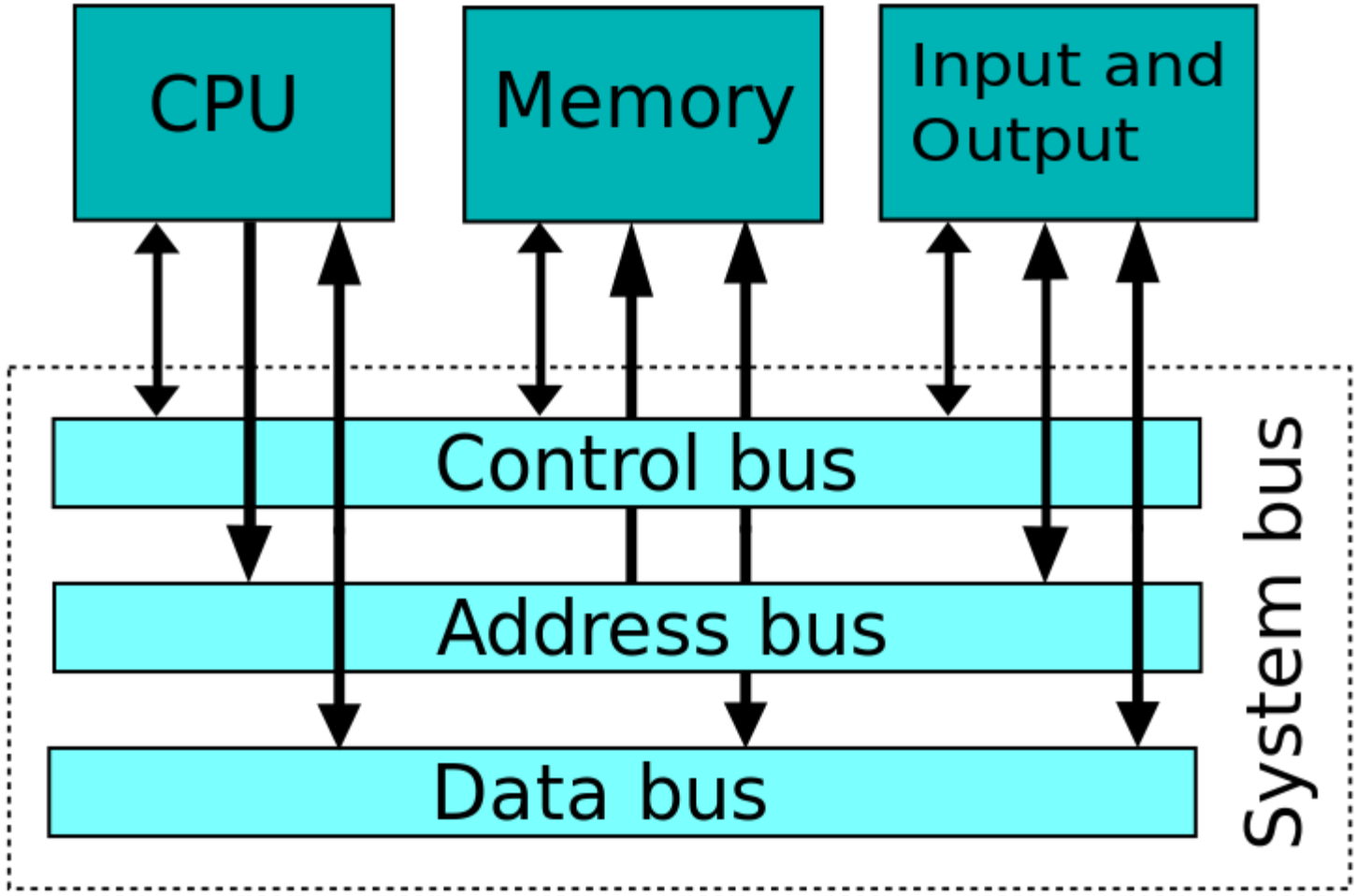


Figure 1.1 Computer Components: Top-Level View

Direcciones de memoria usando lenguaje C++

HW. Principales componentes



Bus de control. Bus bidireccional**.** Transporta señales de control y temporización que la CPU usa para enviar comandos. Sincroniza los otros buses.
Bus de dirección. Bus unidireccional usado para especificar la ubicación de los datos.
Bus de datos. Bus bidireccional usado para transportar los datos. desde la memoria a la CPU (lectura) o desde la CPU a la memoria (escritura).

HW. CPU

Registros internos. Registros internos relacionados con memoria y entrada/salida

- Registro de dirección de memoria, **MAR**, que contiene la dirección de la siguiente R/W.
- Registro de *buffer* de memoria, **MBR**, contiene los datos que van a escribirse en memoria o recibe los datos que se leerán de memoria.
- I/O address register, **I/O AR**, registro de dirección de E/S.
- I/O buffer register, **I/O BR**, registro de *buffer* de E/S.

HW. CPU

Registros de propósito general

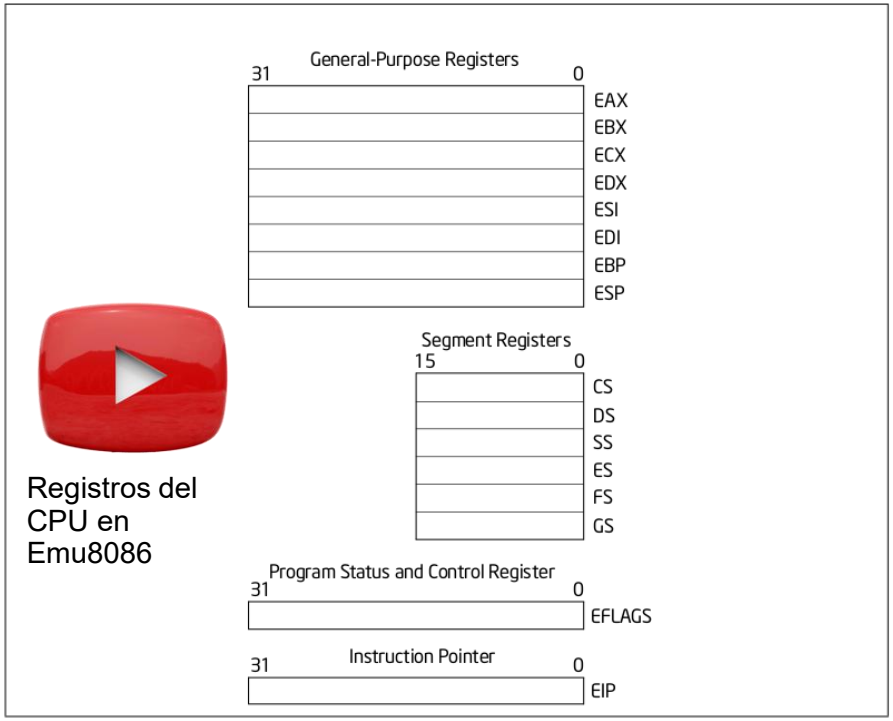
- Pueden ser accedidos por los programas y podemos clasificarlos en registros de datos y registros de direcciones.
- Entre los registros de direcciones podemos encontrar como importantes:
  - **Registro índice** (direccionamiento que consiste en sumar un índice a un valor de base para obtener una dirección efectiva),
  - **Puntero a segmento** (usa un registro para mantener la dirección base que es la posición de inicio del segmento), y
  - **Puntero a la pila** (en direccionamiento de pila para el usuario hay un registro que apunta la cima de la pila).

HW. CPU

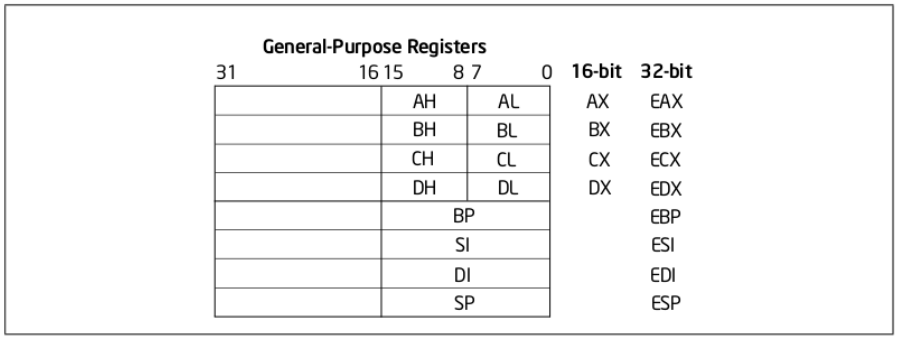
Otros Registros

- Program Counter (**PC**). Contiene la dirección de la siguiente ejecución a ir a buscar (*fetch*).
- Instruction Register (**IR**). Contiene la última instrucción que se fue a buscar (*fetched*).
- Program Status Word (**PSW**). Contiene códigos de condición, información para interrupciones y modo de ejecución del procesador.
- **Ej.**: **IA-32** ofrece 16 reg. básicos: 8 reg. propósito general, 6 reg. de segmento, 1 registro de estado y control y 1 registro PC.

HW. IA-32: General-purpose Regs.



Registros del CPU en Emu8086



From Intel® 64 and IA-32 Manuals

EAX – Acumulador. Es el registro principal para operaciones aritméticas y lógicas. También se utiliza para almacenar el valor de retorno de las funciones.

EBX – Puntero a datos del segmento DS

ECX – Contador para operaciones sobre strings y loops

EDX – Puntero de I/O

Se utiliza para operaciones de entrada/salida (I/O) y junto con el registro EAX en operaciones aritméticas complejas.

ESI – Puntero a datos situados en el segmento apuntado por el registro DS; puntero al origen para operaciones sobre strings

EDI – Puntero a datos (o destino) situados en el segmento apuntado por el registro ES; puntero al destino para operaciones sobre strings

Puntero a la ubicación de destino en operaciones de cadenas o transferencias de memoria.

EBP – Puntero a datos de la pila (*stack*) situados en el segmento SS.

ESP – Puntero de pila (*Stack Pointer*) situada en el segmento SS

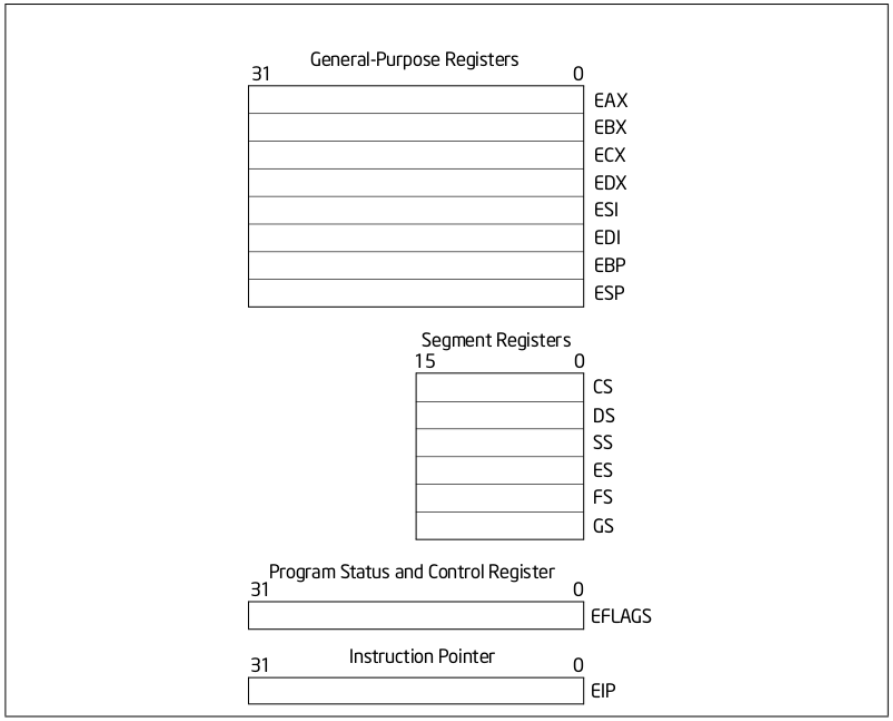
CS – Contiene el selector del segmento de código

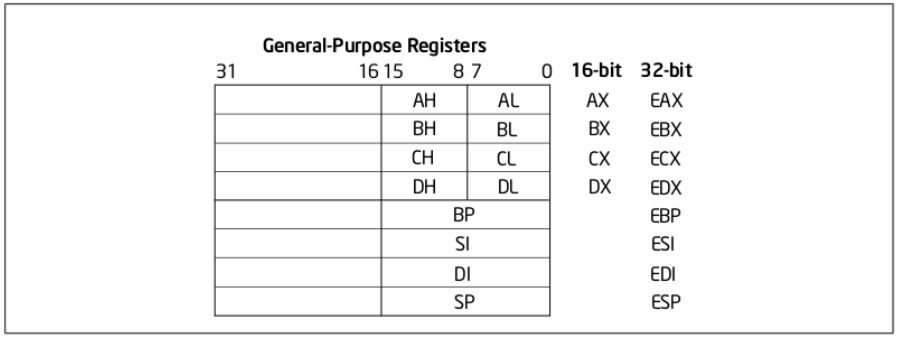
SS – Contiene el selector de segmento para el segmento de pila

DS, ES, FS, GS – Contienen los selectores para segmentos de datos

EIP – *offset* dentro del segmento de código de la siguiente instrucción a ejecutar

HW. IA-32: General-purpose Regs.





From Intel® 64 and IA-32 Manuals

EAX — Accumulator

- Es el **acumulador**: se usa para almacenar operandos y resultados de operaciones aritméticas/lógicas.
- Muchas instrucciones lo usan por defecto.

```
MOV EAX, 5
ADD EAX, 3    ; EAX = 8
```

EBX — Base register

- Apunta a datos dentro del **segmento de datos (DS)**.
- Se usa como **registro base** para direccionamiento de memoria.

```
MOV EBX, [DS:1000h] ; EBX carga el dato en la dirección DS:1000h
```

ECX — Counter

- Es el **contador** para bucles (`LOOP`) y operaciones con cadenas.
- Muchas instrucciones repetitivas usan **ECX** automáticamente.

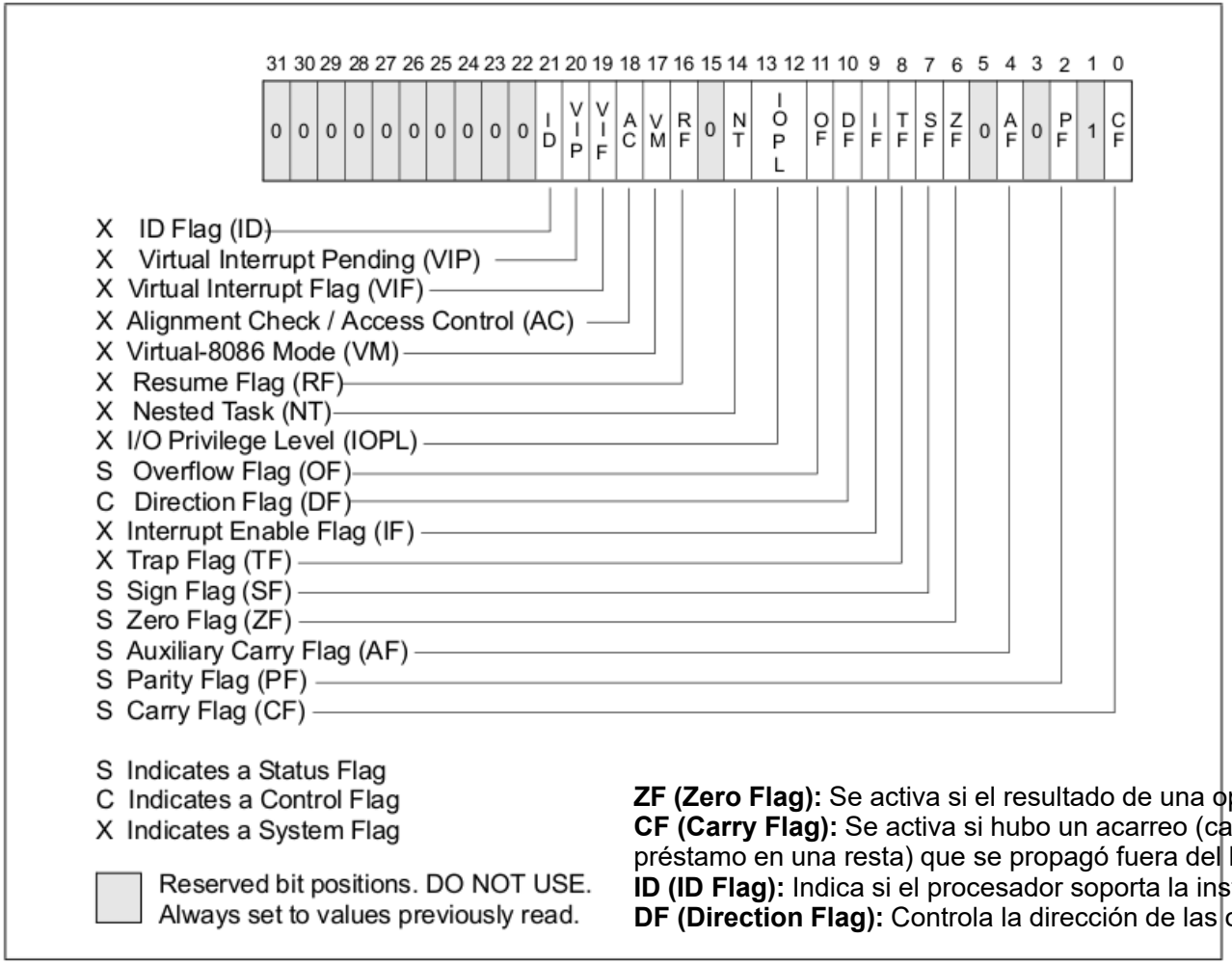
```
MOV ECX, 5
LOOP etiqueta    ; Repite 5 veces el bloque en "etiqueta"
```

EDX — Data / I/O register

- Usado como **extensión de EAX** en multiplicaciones y divisiones grandes.
- También sirve como **puntero para operaciones de Entrada/Salida (I/O)**.

```
MOV EAX, 10
MOV EBX, 3
DIV EBX   ; Cociente en EAX, residuo en EDX
```

HW. IA-32: EFLAGS register

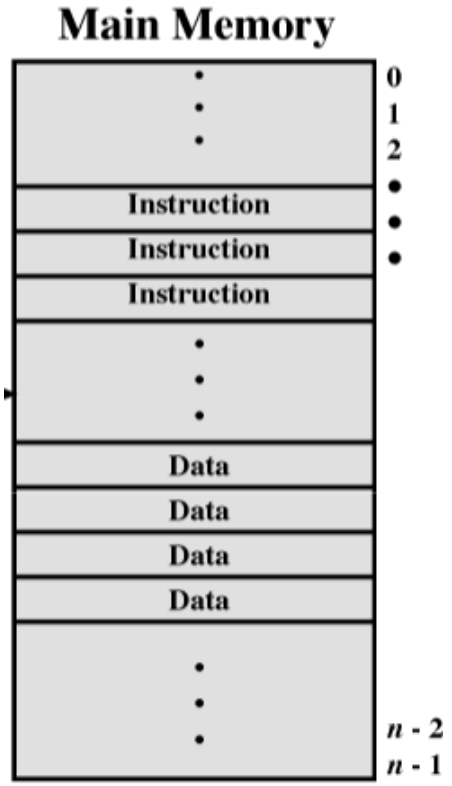


ZF (Zero Flag): Se activa si el resultado de una operación es cero.
CF (Carry Flag): Se activa si hubo un acarreo (carry) en una suma (o un préstamo en una resta) que se propagó fuera del bit más significativo.
ID (ID Flag): Indica si el procesador soporta la instrucción CPUID
DF (Direction Flag): Controla la dirección de las operaciones de cadena.

From Intel® 64 and IA-32 Manuals

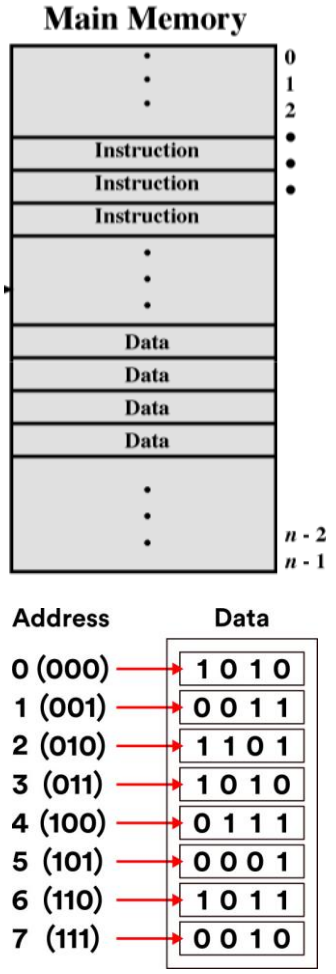
HW. Memoria principal (RAM)

- Modelo abstracto. Tabla o array lineal compuesto por un número de elementos (**celdas (o palabras) de memoria**) con un tamaño.
- Las palabras son direccionables con números naturales desde el 0. Cada número es la **dirección de memoria** (*memory address*) de la correspondiente celda de memoria.



HW. Memoria principal (RAM)

- **Espacio de direcciones** (*address space*). Conjunto de números que representa las direcciones de una memoria.
- **Ejemplo 1: Memoria pequeña de 1 KB (1024 bytes).** Supongamos que tenemos 1 KB de memoria RAM. Como cada byte tiene su dirección, necesitamos 10 bits (porque 2^10=1024).
  - 0000000000 (0) → Primera celda
  - 0000000001 (1) → Segunda celda
  - ...
  - 1111111111 (1023) → Última celda
- Rango 0 a 1023 es el espacio de direcciones de la memoria.



HW. Memoria principal (RAM)

Operaciones:

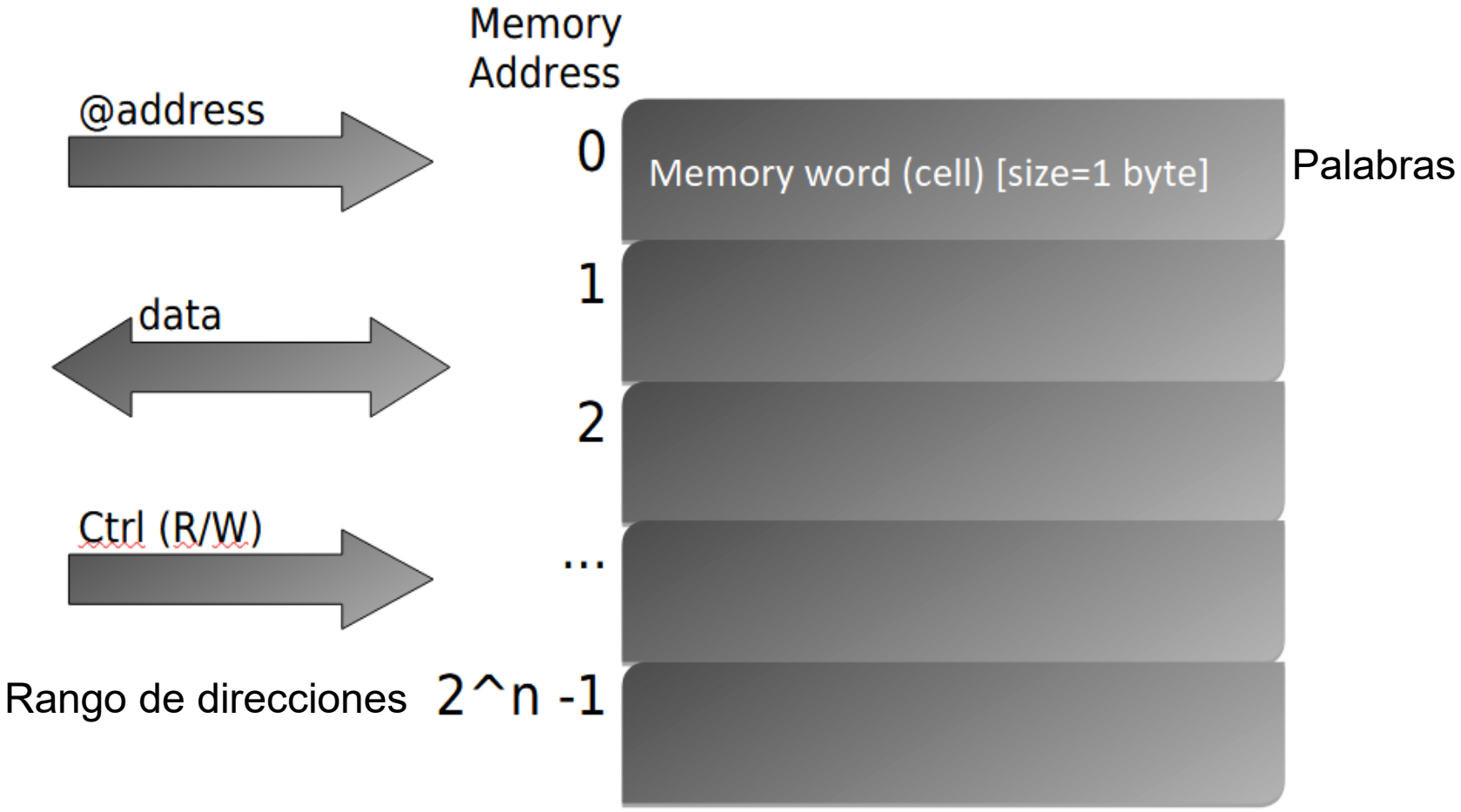
- Lectura (**R**). La memoria recibe una dirección y devuelve el byte contenido en la celda identificada por dicha dirección.
  - byte read(address i);
- Escritura (**W**). La memoria recibe una dirección y un byte y sobreescribe el byte contenido en la celda indicada por la dirección de memoria.
  - void write(address i, byte b);

HW. Memoria principal (RAM)

Direcciones y tamaño:

- Las direcciones se codifican en base 2 (obviamente, al igual que el contenido). Este hecho influye directamente en el **tamaño**.
- Si utilizamos n bits para codificar las direcciones, el espacio de direcciones tiene una cardinalidad de 2^n direcciones, [0…,2^n - 1].
- Luego el tamaño de la memoria es 2^n multiplicado por el tamaño de la palabra.
- Normalmente el ancho del bus de direcciones determina el número de bits que codifican el espacio de direcciones.
  - CPU con 32 bits de bus de direcciones → 2^32=4,294,967,296=4 GB máximo direccionable.
  - La RAM instalada físicamente en el sistema no puede superar la capacidad máxima direccionable del procesador.

HW. Memoria principal (RAM)



En un procesador de **32 bits**, una palabra = 4 bytes (32 bits).

En un procesador de **64 bits**, una palabra = 8 bytes (64 bits).

HW. Módulo de E/S

Funciones de un módulo E/S:

- Control y temporización. Por ejemplo, controlar la transferencia de datos al procesador.
- Comunicación con el procesador y con los dispositivos.
- Almacenamiento temporal de datos (*buffers*). La velocidad de transferencia entre los dispositivos y el módulo E/S es mucho menor que entre procesador/memoria, procesador/módulo o memoria/módulo.
- Detección de errores. Por ejemplo: bit de paridad o códigos CRC (*cyclic redundancy check*) en sectores de disco.

HW. Módulo de E/S

- La CPU coloca en las líneas de dirección qué dispositivo quiere usar.
- Indica en las líneas de control si desea leer o escribir.
- El módulo de E/S usa sus registros internos (data y control) para manejar la operación.
- La lógica de interfaz se comunica con el periférico real.
- Si es necesario, el dispositivo avisa a la CPU mediante interrupciones cuando la operación termina.

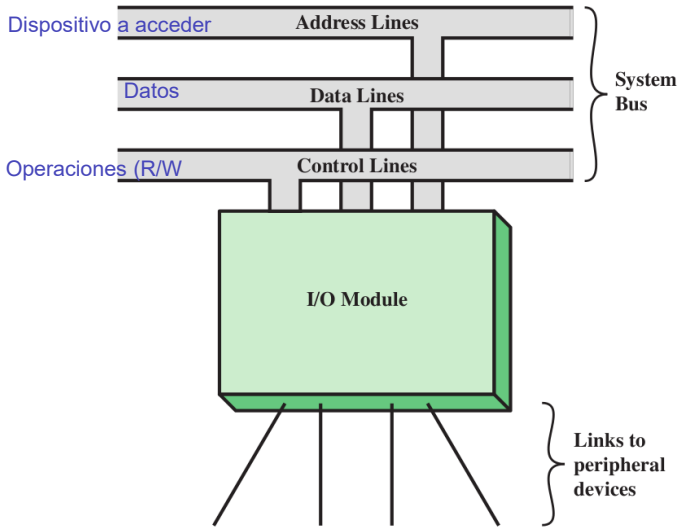


Figure 7.1 Generic Model of an I/O Module

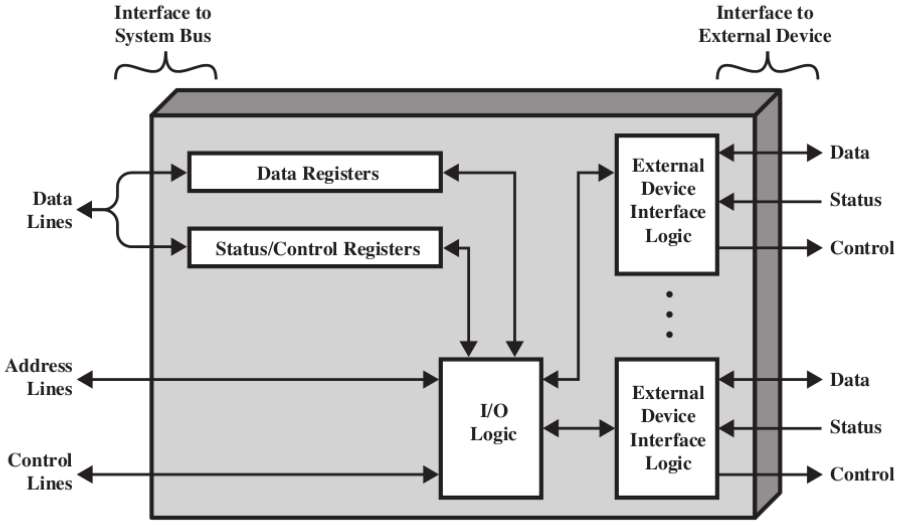
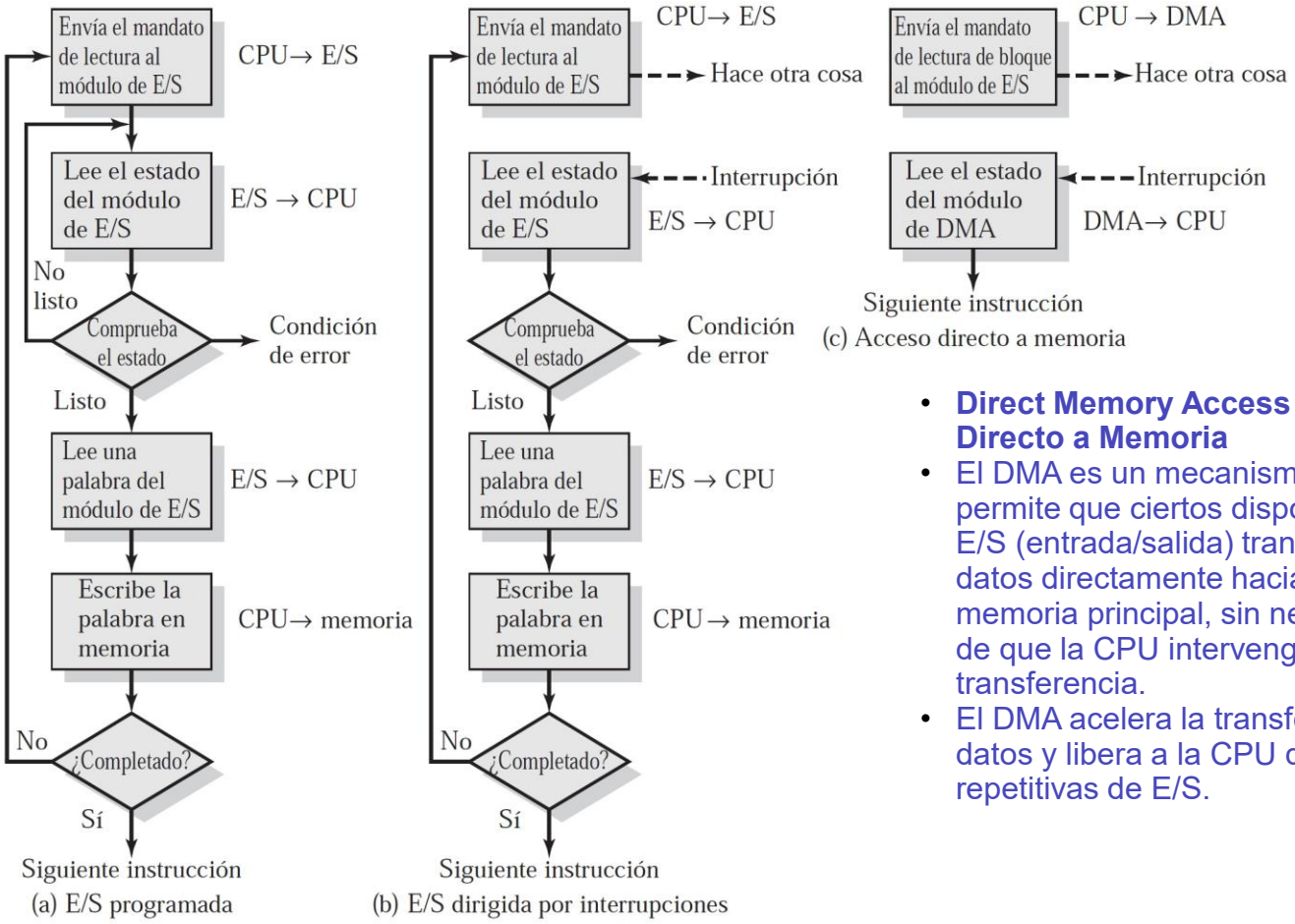


Figure 7.3 Block Diagram of an I/O Module From [Sta2005]

HW. Principales técnicas de E/S



- **Direct Memory Access / Acceso Directo a Memoria**
- El DMA es un mecanismo que permite que ciertos dispositivos de E/S (entrada/salida) transfieran datos directamente hacia/desde la memoria principal, sin necesidad de que la CPU intervenga en cada transferencia.
- El DMA acelera la transferencia de datos y libera a la CPU de tareas repetitivas de E/S.

Figura 1.19. Tres técnicas para leer un bloque de datos. De [Sta2005]

HW. Ciclo de ejecución de instrucción con interrupciones

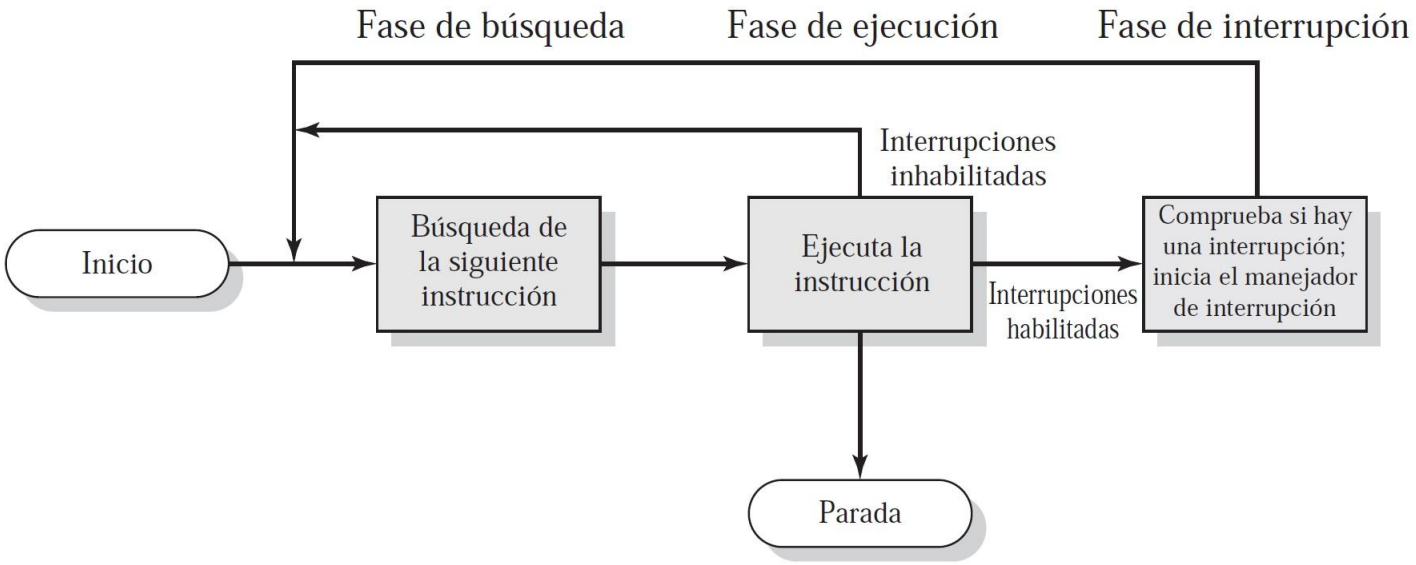


Figura 1.7. Ciclo de instrucción con interrupciones. De [Sta2005]

HW. Tratamiento de interrupciones

Program Status Word
Program Counter

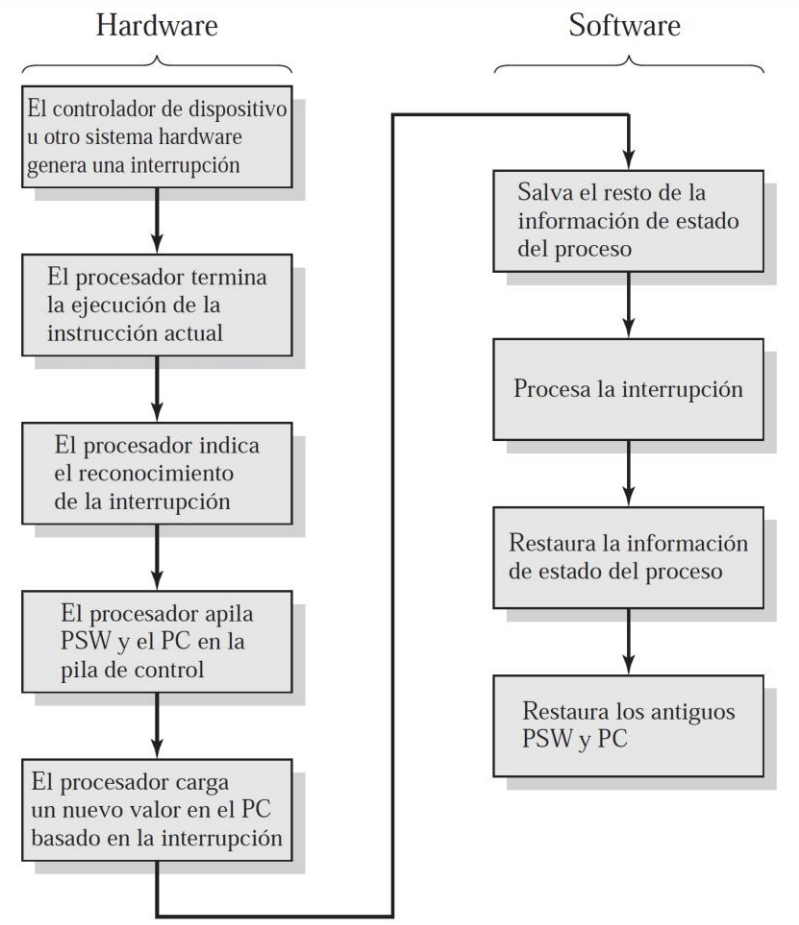


Figura 1.10. Procesamiento simple de interrupciones.

De [Sta2005]

HW. Modo dual de ejecución de CPU

User/kernel mode:

- En respuesta a una **interrupción** solicitada desde un módulo de E/S (controlador), el HW apila PSW y PC actuales, activa modo kernel, y carga el PC con el contenido del ***vector*** correspondiente al dispositivo controlado por el controlador que solicita la interrupción.
- En respuesta a una condición de **excepción**, como resultado de la ejecución de una instrucción, el HW apila PSW y PC actuales, activa modo kernel, y carga PC con el contenido del **vector** correspondiente a la excepción (intento de recuperación).

HW. Modo dual de ejecución de CPU

User/kernel mode (cont.):

- En respuesta a una **llamada al sistema** (interrupción software) solicitada explícitamente por el programa en ejecución (ej. **syscall** en IA-32), el HW apila PSW y PC actuales, activa modo kernel, y carga el PC con el contenido del ***vector*** correspondiente al gestor general de llamadas.

Una llamada al sistema es un mecanismo que permite que un programa de usuario solicite un servicio al sistema operativo (SO). En otras palabras, es la puerta de entrada controlada entre el espacio de usuario y el espacio del kernel.

1. **Gestión de procesos**
   - `fork()` → crear un proceso.
   - `exec()` → ejecutar un programa.
   - `exit()` → terminar un proceso.
2. **Gestión de archivos**
   - `open()` → abrir un archivo.
   - `read()` → leer datos de un archivo.
   - `write()` → escribir en un archivo.
   - `close()` → cerrar archivo.
3. **Comunicación**
   - `socket()` → crear un socket de red.
   - `send()` / `recv()` → enviar y recibir mensajes.
4. **Gestión de memoria**
   - `mmap()` → mapear memoria.
   - `brk()` / `sbrk()` → cambiar el tamaño del heap.

Tema 1
Estructuras de Sistemas Operativos

Recordatorio de TOC y FS

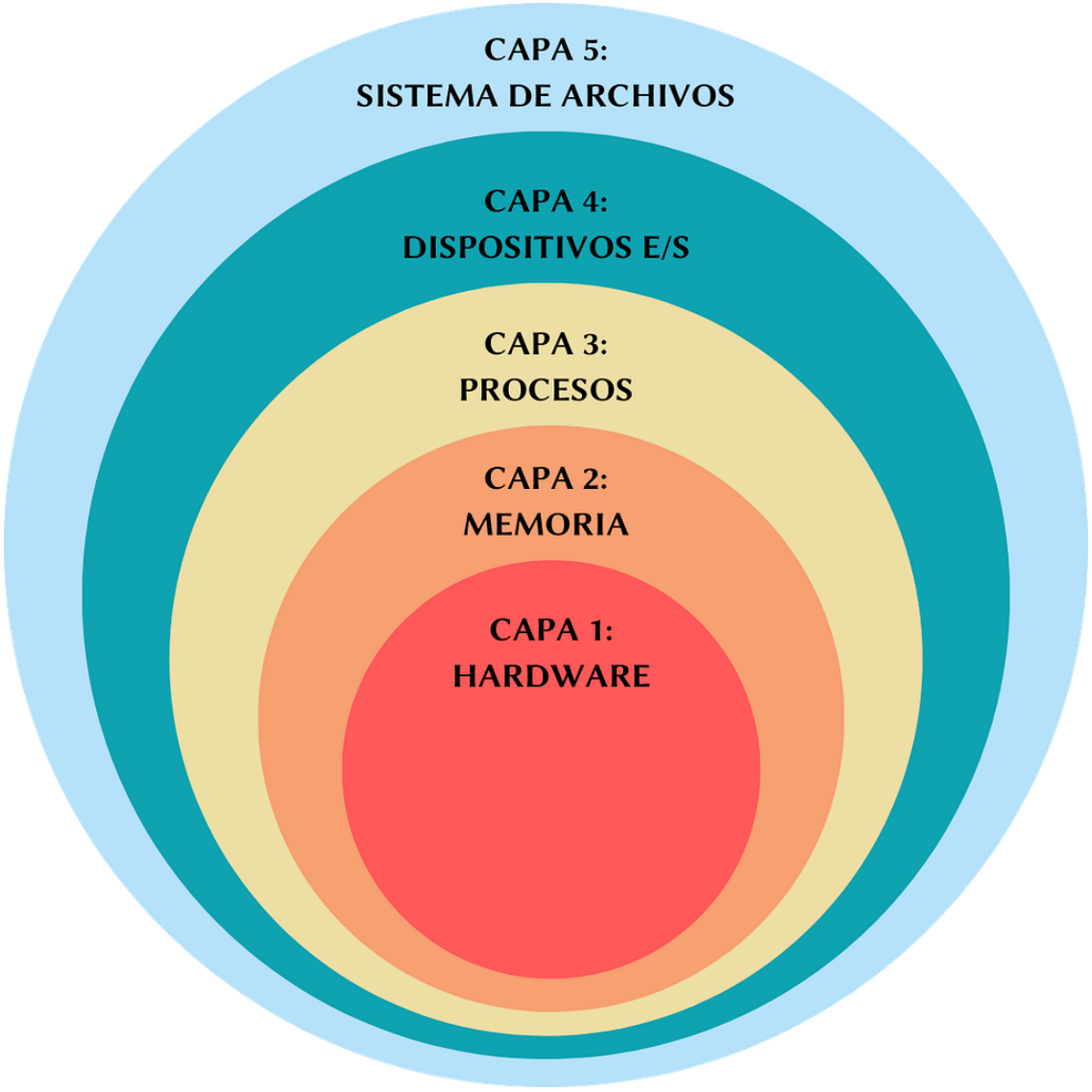
2 **Componentes de un SO**

Estructuras/Arquitecturas de los SO’s

SO’s de propósito específico

Sistemas Operativos
Dpto. Lenguajes y Sistemas Informáticos, 2025

Componentes de un SO



Programa que tiene encomendado una serie de funciones cuyo objetivo es simplificar el manejo y utilización del ordenador haciéndolo seguro y eficiente (Carretero, 2020).

Funcionalidad para procesos

Proceso. Concepto que permite monitorizar y controlar sistemáticamente la ejecución de varios programas en el procesador. Un proceso debe tener asociados dos elementos:

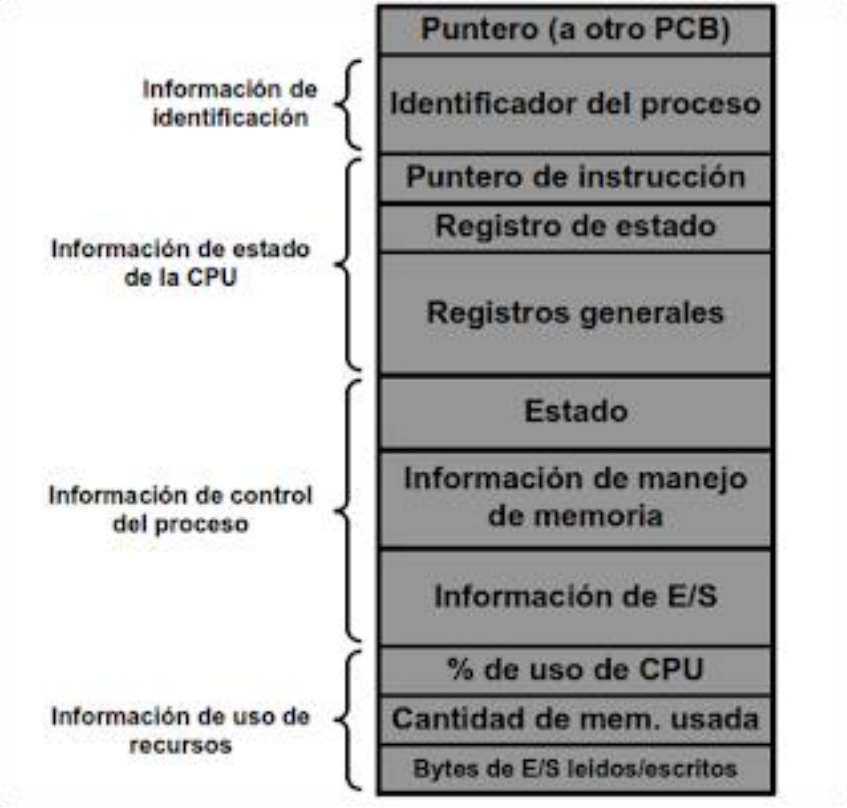
- El programa a ejecutar y,
- La información para supervisión y control de la ejecución del programa: **PCB**, del inglés *Process Control Block.*
- Consideraciones: *Multiprogramming, Timesharing, CPU scheduling, swapping, virtual memory*.

Funcionalidad para procesos

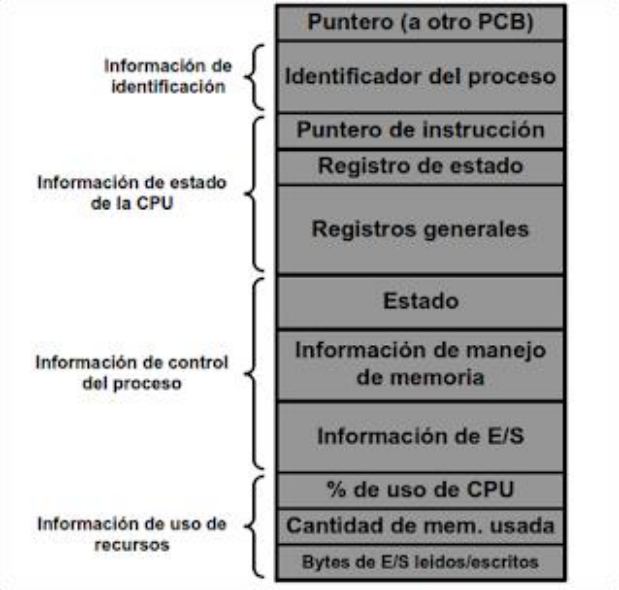
PCB, del inglés *Process Control Block. El DNI del proceso.*

- Datos de control
- Estado
- Recursos

Tabla de procesos es una estructura cuyas entradas son bloques de control de procesos.



Funcionalidad para procesos



1. **Identificación del proceso**
   - PID = 1056 (Process ID)
   - PPID = 1000 (ID del proceso padre, por ejemplo, la terminal que lo lanzó).
2. **Estado del proceso**
   - Estado actual: **Ejecutando** (Running)
   - Posibles valores: Nuevo, Listo, Bloqueado, Terminado.
3. **Contador de programa (PC)**
   - Dirección de la próxima instrucción a ejecutar.
   - Ejemplo: `0x7FFE1234`.
4. **Registros de la CPU**
   - AX = 5, BX = 10, ... (valores intermedios de la ejecución).
   - Guardados cuando el proceso es interrumpido o se hace un cambio de contexto.
5. **Información de memoria**
   - Tabla de páginas o segmentos asignados.
   - Dirección base = `0x00400000`.
   - Límite = 2 MB asignados.
6. **Información de planificación (scheduling)**
   - Prioridad = 5.
   - Tiempo de CPU usado = 120 ms.
   - Tiempo de espera = 30 ms.
7. **Información de E/S (entrada/salida)**
   - Archivos abiertos: `doc1.txt`, `config.ini`.
   - Dispositivos asignados: teclado y pantalla.
8. **Información contable**
   - Usuario propietario = `juan`.
   - Tiempo total de CPU = 5 s.
   - Uso de memoria = 50 MB.

Funcionalidad para procesos

- **Creación** del PCB asociado a un programa que va a ejecutarse. **Eliminación** del PCB una vez finalizada la ejecución.
- **Bloqueo (*sleep*)** y **desbloqueo (*wakeup*)** de los procesos dependiendo de los eventos por los que debe esperar un programa para poder continuar su ejecución.
- Proporcionar mecanismos que posibiliten la **comunicación** y la **sincronización** entre procesos.

Funcionalidad para procesos

- Mecanismos de **comunicación**
  - Memoria compartida. Dos o más proceso pueden acceder a una misma región de memoria.
  - Pipes. Canal unidireccional de comunicación entre proceso padre-hijo.
  - Colas de mensaje. Estructuras donde los procesos envía/reciben mensajes.
  - Señales. Mecanismo para notificar a un proceso de un evento.

Funcionalidad para procesos

- Mecanismos para la **sincronización** entre procesos
  - Semáforos. Variables especiales que controlan el acceso a recursos limitados.
  - Mutex (exclusión mutua). Semáforo binario. Permiten que un solo proceso entre en la sección crítica.
  - Monitores. Garantizan que solo un proceso/hilo ejecute un método del monitor a la vez.
  - Barreras. Sincronizan grupos de procesos, todos deben llegar a un punto antes de que alguno continúe.
  - Bloqueos por condición. Un proceso se suspende hasta que cierta condición lógica se cumpla.

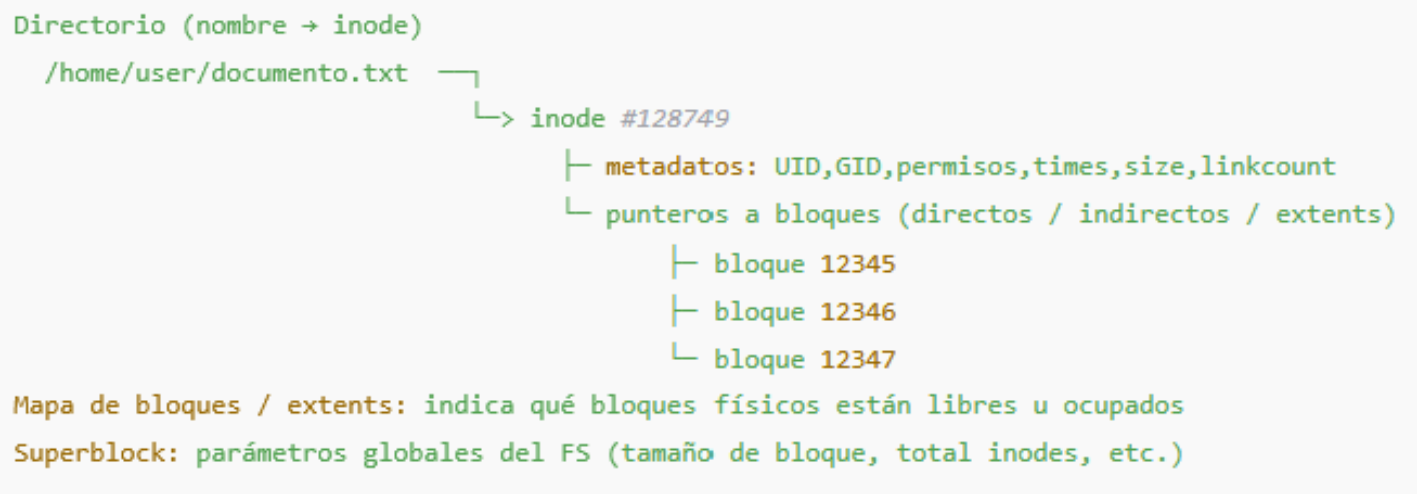
Funcionalidad para memoria

- **Protección** de la región de memoria principal ocupada por el kernel y protección de las regiones de memoria principal ocupadas por los programas.
- **Compartición** de parte de las regiones ocupadas por los programas para permitir comunicación entre estos.
- Gestión automática de la **asignación/liberación** de la memoria disponible para un programa en cualquier nivel de la **jerarquía de memoria** (registros CPU, caché, RAM, almacenamiento secundario) y de forma transparente al programador.
- Mantener información sobre la **memoria asignada** a los procesos, núcleo, etc. y la **memoria libre** en cualquier nivel de la jerarquía.
- Implementar algoritmos para decidir cuanta memoria asignar a cada proceso y cuando debe ser liberada toda la memoria asociada a un proceso manteniéndolo en almacenamiento secundario.
  - Si la RAM no alcanza, se saca un proceso completo a disco (swap out) y luego se trae de vuelta (swap in) cuando se necesite.

Funcionalidad para archivos y directorios

- Un **archivo** es una colección de información identificada por nombre que reside en almacenamiento secundario.
- La funcionalidad relacionada con los archivos está asociada al sistema de archivos, el cual permite:
  - Crear, eliminar, copiar, renombrar,... archivos y directorios.
  - Mantiene los **atributos de archivo** en una estructura de datos (**inode** en UNIX/Linux).
    - Nombre, tipo, tamaño, fecha-creación, fecha-modificación, fecha-último-acceso, propietarios, ubicación-en-disco, otros.
  - Abrir y cerrar sesiones de trabajo con archivos, leer, escribir, variar el puntero de lectura/escritura.

Funcionalidad para archivos y directorios



Funcionalidad para archivos y directorios

- Gestionar la asignación y liberación de espacio en disco para mantener la información de los archivos y directorios.
  - Contigua. Cada archivo se almacena en bloques consecutivos del disco.
  - Encadenada. Cada bloque apunta al siguiente. Bloques dispersos.
  - Indexada. Usa un bloque especial con punteros a todos los bloques del archivo.
- Mantener la ubicación en disco de los datos asociados a archivos y directorios, así como los **metadatos del Sistema de Archivos**.

Funcionalidad para dispositivos de E/S

- Controlar el funcionamiento de los dispositivos de E/S.
- Proteger su utilización de forma directa por parte de los programas de aplicación: acceso mediante API de llamadas al sistema (controlado por el kernel).
- Proporcionar una **interfaz independiente** de las particularidades de los dispositivos al resto del SO y a los programadores: interfaz estándar para todos los dispositivos.
- Manejador de dispositivo (***Device Driver***). Módulo software que gestiona un determinado dispositivo y se encarga de implementar las operaciones del interfaz, implementar el manejo de interrupciones, etc.
  - Aplicaciones – SO – Driver - HW

Funcionalidad para recursos del SO

Planificación y gestión de recursos (CPU, memoria, E/S).

- **Equitatividad**. El acceso a los recursos debe estar garantizado para todos los procesos.
- **Respuesta diferencial**. El SO debe planificar la asignación de los recursos teniendo en cuenta características diferenciales de los clientes, procesos u otras entidades (p.e. IORBs, Bloque de Petición de Entrada/Salida) es una estructura de datos usada por el sistema operativo para gestionar operaciones de E/S).
- **Eficiencia**. El SO debe maximizar la productividad, minimizar el tiempo de respuesta y, en sistemas de tiempo compartido, permitir el trabajo simultáneo de tantos usuarios como sea posible.

Funcionalidad para recursos del SO

Protección y seguridad.

- **Protección**. Cualquier mecanismo para controlar el acceso de procesos o usuarios a recursos del SO (ej. control de acceso a archivos en el *filesystem*).
- **Seguridad**. Defensa del sistema frente a ataques internos o externos (ej. denegación de servicio, *worms*, virus, robo de identidad).
- Los sistemas distinguen usuarios para determinar quién puede hacer que. UID y GID se asocian con procesos y archivos para determinar control de acceso.

Componentes generales de un SO

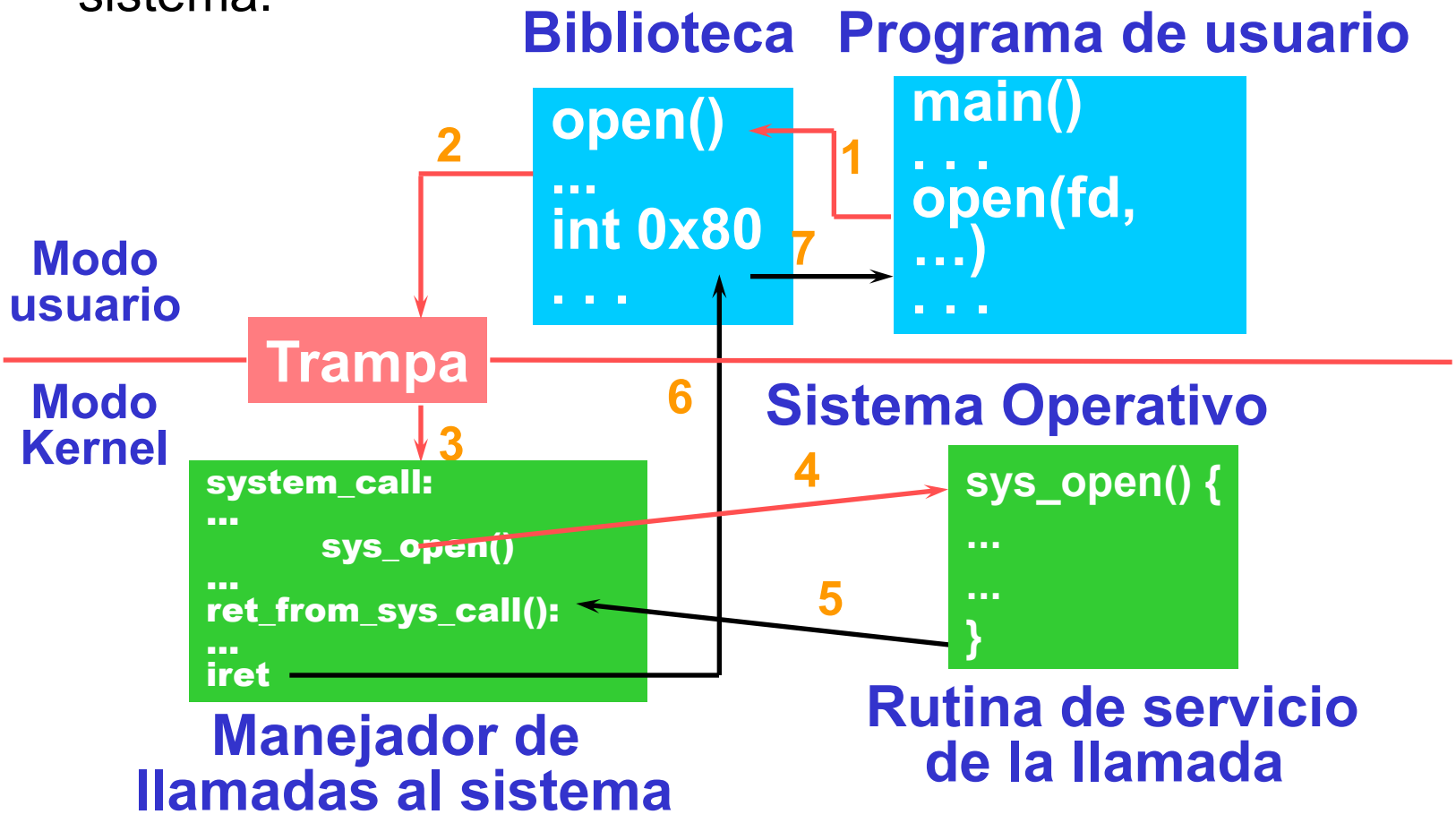
- La funcionalidad descrita previamente se suele agrupar en los siguientes componentes fundamentales.
  - Gestor de procesos.
  - Gestor de memoria.
  - Gestor de archivos.
  - Gestor de E/S.
  - Gestor de comunicaciones.

Interfaces entre usuario y SO

- *Command Line Interface* (CLI). A veces implementado en el kernel y la mayoría mediante *shells*.
- Los *shells* permiten ejecutar órdenes *built-in* (implementadas en el propio programa *shell*) u órdenes (programas) que residen en el sistema de archivos(FS).
- *Graphic User Interface* (GUI). Interfaz que utiliza la metáfora de escritorio (Desktop metaphor). Usa teclado/ratón.
- *System calls* (Llamadas al sistema). Interfaz de programación a los servicios proporcionados por el SO. Los programas lo utilizan mediante una API en lugar de acceso directo a la *system call*.

System call

- Vistazo a los componentes implicados en una llamada al sistema.



System call. Implementación

- Se asocia un número con cada llamada al sistema proporcionada por el SO.
- El interfaz de llamadas invoca la llamada requerida por el programador y devuelve el estado de finalización y los valores de retorno.
- Formas para pasar parámetros:
  - En registros de la CPU.
  - Parámetros en memoria paso la dirección del bloque en un registro.
  - Parámetros en una pila.

System call. Implementación

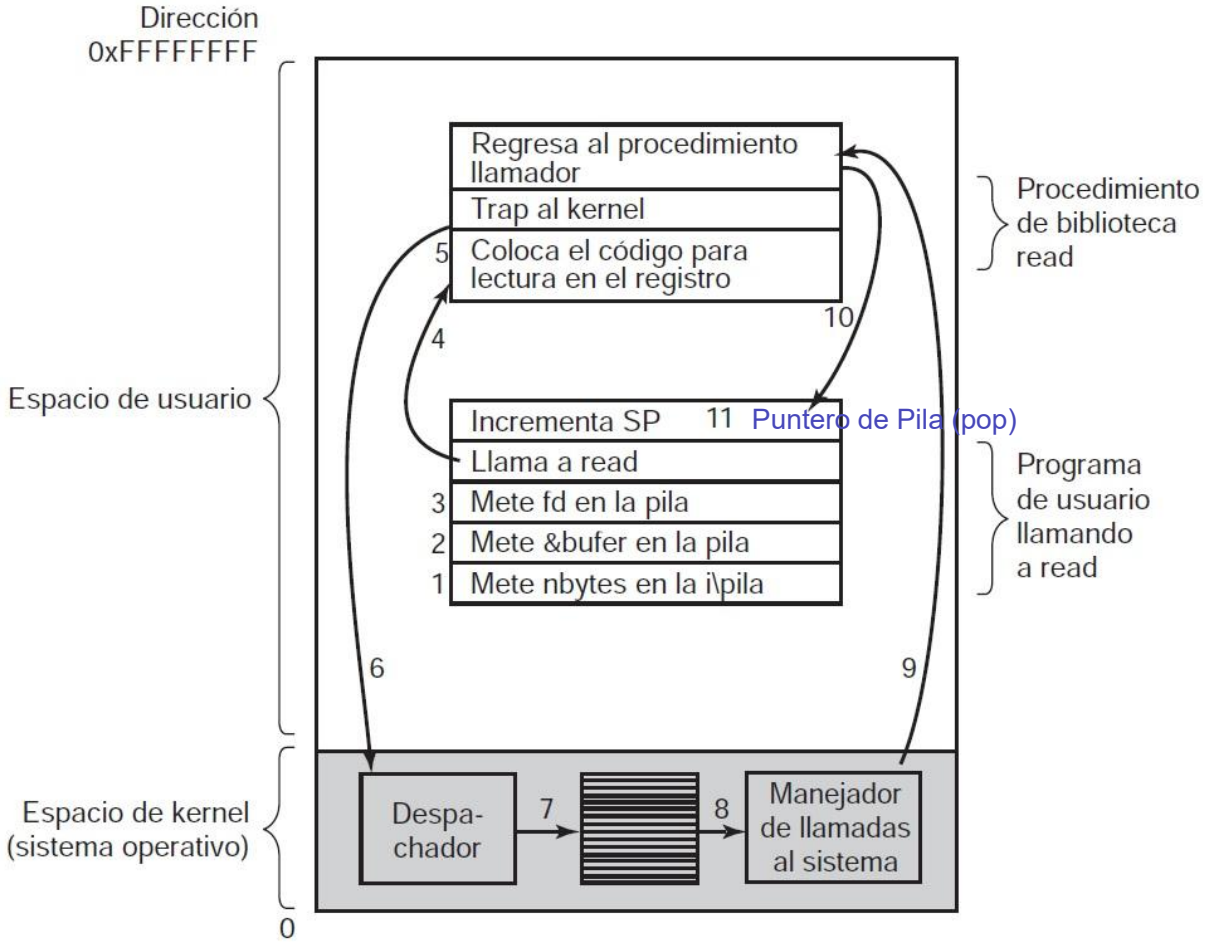


Figura 1-17. Los 11 pasos para realizar la llamada al sistema `read(fd, bufer, nbytes)`. De [Tanenbaum 2009]

System call. Implementación

- ¿Cuales son los identificadores de llamada en Linux x86_64?

https://github.com/torvalds/linux/blob/16f73eb02d7e1765ccab3d2018e0bd98eb93d973/arch/x86/entry/syscalls/syscall_64.tbl

```
#
# 64-bit system call numbers and entry vectors
#
# The format is:
# <number> <abi> <name> <entry point>
#
# The abi is "common", "64" or "x32" for this file.
#
0    common     read                sys_read
1    common     write               sys_write
...
60   common     exit                sys_exit
...
332  common     statx               sys_statx
```

Lectura de archivo o dispositivo

Escribir datos en un archivo o salida estándar

Termina un proceso

Devuelve información detallada sobre un archivo (metadatos)

System call. Implementación

- Paso de parámetros a syscall siguiendo *x86-64 calling conventions*.

https://github.com/torvalds/linux/blob/16f73eb02d7e1765ccab3d2018e0bd98eb93d973/arch/x86/entry/entry_64.S

```
...
 * Registers on entry:
 * rax  system call number
 * rcx  return address
 * r11  saved rflags (note: r11 is callee-clobbered register in C ABI)
 * rdi  arg0
 * rsi  arg1
 * rdx  arg2
 * r10  arg3 (needs to be moved to rcx to conform to C ABI)
 * r8   arg4
 * r9   arg5
...
```

System call. Implementación

- Programas en C. La biblioteca C (libc) implementa la API principal en UNIX, e incluye la biblioteca estándar de C y la interfaz de llamadas al sistema.

```c
/* helloWorld.c */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
  printf("Hello World...!\n");
  return EXIT_SUCCESS;
}
```

Función de biblioteca libc -> llama internamente a la syscall write

`printf` → **Biblioteca de C** (`libc`) → llama a → `write` → Kernel.

```c
/* helloWorldWrapper.c */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

int main(int argc, char *argv[])
{
  write(STDOUT_FILENO,"Hellow World...!\n\n",strlen("Hellow World...!\n\n") * sizeof(char));
  return EXIT_SUCCESS;
}
```

Usa directamente la syscall declarada en unistd.h `write` → **Syscall** → Kernel → pantalla.

System call. Implementación

```
# helloSyscall.S
# Programa en ensamblador (x86-64, Linux)
# Uso de llamadas al sistema (syscalls) puras

# Compilación y ejecución:
# $ gcc -c helloSyscall.S
# $ ld -o helloSyscall helloSyscall.o
# $ ./helloSyscall

    .data                           # Sección de datos inicializados
msg:
    .ascii "Hello, world!\n"        # Cadena a imprimir
len = . - msg                       # Longitud del mensaje

    .text                           # Sección de código
    .global _start                  # Punto de entrada

_start:
    # --- write(fd=1, buf=msg, count=len) ---
    movq $1, %rax        # syscall: 1 = sys_write
    movq $1, %rdi        # arg1: file descriptor (stdout)
    movq $msg, %rsi      # arg2: dirección del buffer
    movq $len, %rdx      # arg3: número de bytes
    syscall              # invoca la llamada al sistema

    # --- exit(status=0) ---
    movq $60, %rax       # syscall: 60 = sys_exit
    xorq %rdi, %rdi      # arg1: código de salida (0)
    syscall              # termina el proceso
```

Ensamblador sintaxis ATRT

syscalls puras en **x86-64 Linux**

`%rax` : número de syscall.

`%rdi` : primer argumento.

`%rsi` : segundo argumento.

`%rdx` : tercer argumento.

`%r10` , `%r8` , `%r9` : argumentos siguientes.

El valor de retorno se recibe también en `%rax` .

System call. Implementación

```
; helloSyscall.asm — Linux x86-64, syscalls puras (NASM / Intel syntax)
; Compilar y ejecutar:
;   $ nasm -f elf64 -o helloSyscall.o helloSyscall.asm
;   $ ld -o helloSyscall helloSyscall.o
;   $ ./helloSyscall

section .data                           ; ---- Datos inicializados
    msg     db  "hello, world!", 0x0A   ; cadena + '\n'
    len     equ $ - msg                 ; longitud de la cadena

section .text                           ; ---- Código
    global _start

_start:
    ; write(fd=1, buf=msg, count=len)
    mov     rax, 1                      ; nº de syscall: 1 = sys_write
    mov     rdi, 1                      ; arg1: fd = 1 (STDOUT)
    mov     rsi, msg                    ; arg2: puntero al buffer
    mov     rdx, len                    ; arg3: nº de bytes a escribir
    syscall                             ; entra a kernel y escribe

    ; exit(status=0)
    mov     rax, 60                     ; nº de syscall: 60 = sys_exit
    xor     rdi, 0                      ; arg1: código de salida = 0
    syscall                             ; termina el proceso
```

Ensamblador sintaxis NASM (Intel syntax)

`RAX`: número de syscall (y valor de retorno).

`RDI`, `RSI`, `RDX`, `R10`, `R8`, `R9`: argumentos 1..6 de la syscall.

Tema 1
Estructuras de Sistemas Operativos

Recordatorio de TOC y FS

Componentes de un SO

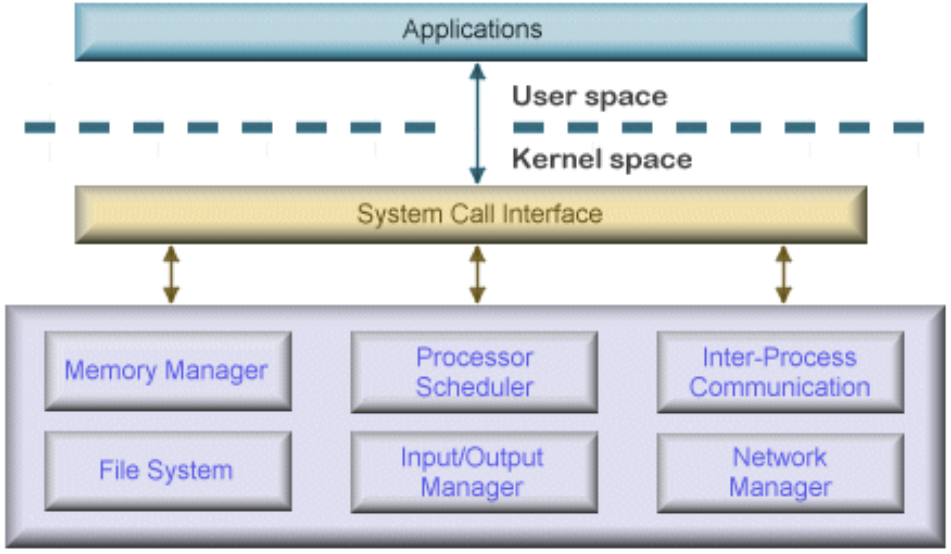
3

Estructuras/Arquitecturas de los SO's

SO's de propósito específico

Arquitectura Monolítica

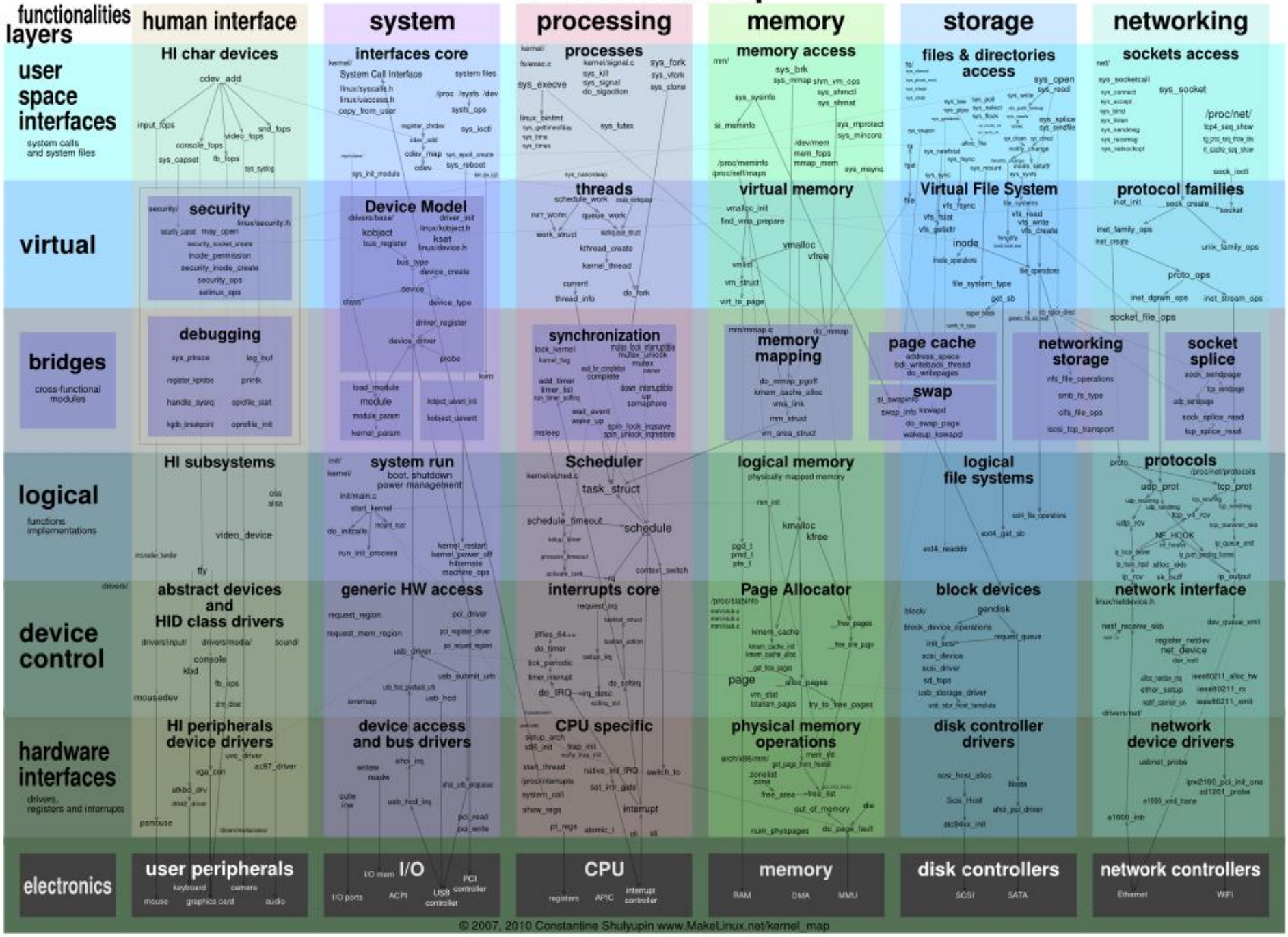
- Es la arquitectura comúnmente utilizada.
- El SO es un único programa que se ejecuta en el modo más privilegiado del procesador (**ring 0** en x86).
- Las dependencias entre los distintos módulos son complejas salvo para algunos elementos bien establecidos (¡NO oculta información!).
- El modelo de obtención de servicios es la llamada a procedimiento.



Arquitectura monolítica: Problemas

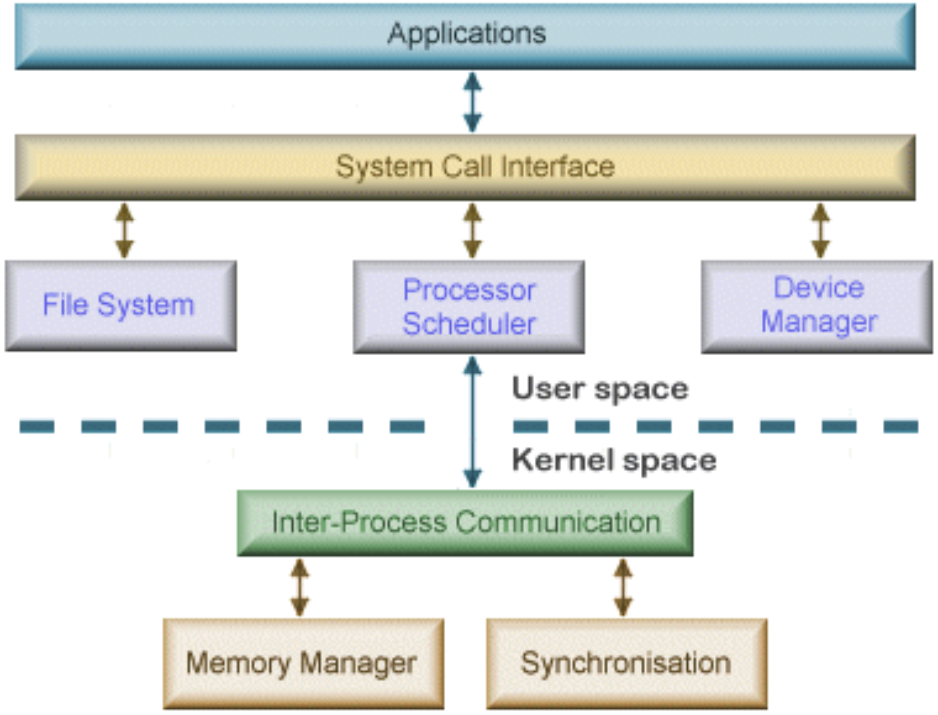
- La fuerte dependencia entre módulos provoca dificultades a la hora de comprender el código y de realizar modificaciones.
- Al ser un solo programa cargado en memoria el fallo de un módulo puede provocar la caída del sistema.
- UNIX clásico (años 70s-80s), MS-DOS, Linux (aunque con soporte para módulos dinámicos, sigue siendo monolítico en esencia), FreeBSD, OpenBSD, NetBSD.

Linux kernel map



Arquitectura microkernel

- Una pequeña parte de la funcionalidad del SO se implementa como kernel y el resto como procesos de usuario.
- El microkernel soporta memoria virtual de bajo nivel, creación de procesos (hebras) y, comunicación y sincronización.



Arquitectura microkernel

- El modelo de obtención de servicios es por **paso de mensajes** entre procesos.
- La aplicación solicita un servicio al SO, `send(Server,&m)`, y espera la resolución del servicio, `receive(Server,&r)`.
- El microkernel obtiene la solicitud de servicio y entrega el mensaje al servidor `Server`.
- El servidor `Server` obtiene el mensaje m, genera el resultado y envía el resultado al kernel para que este lo devuelva a la aplicación, `r`.

Arquitectura microkernel: Ventajas y desventajas

- **Fiabilidad**. Un error en un módulo que se implemente como proceso de usuario solo provoca una caída parcial del sistema que puede recuperarse levantando el proceso de servicio.
- **Extensibilidad**. Podemos incluir nuevos servicios como procesos de usuario.
- **Peor rendimiento** que la arquitectura monolítica porque para obtener un servicio se realizan más cambios de modo y de espacio de direcciones.

Tema 1
Estructuras de Sistemas Operativos

Recordatorio de TOC y FS
Componentes de un SO
Estructuras/Arquitecturas de los SO's
SO's de propósito específico

Sistemas Operativos
Dpto. Lenguajes y Sistemas Informáticos, 2025

SO de Tiempo Real (RTOS)

- Los RTOS se utilizan para aplicaciones especializadas, normalmente sistemas de control.
- Un RTOS debe garantizar la corrección no solo del resultado lógico de la computación sino del tiempo empleado en producir los resultados.
- **Problema**: Planificar las actividades (procesos) con el fin de satisfacer todos los requisitos de tiempo: **planificabilidad**.

SO de Tiempo Real (RTOS)

- En un RTOS algunos procesos se clasifican como procesos de tiempo real (RT task or process).
- Los procesos RT tienen como objetivo procesar eventos que se producen, regularmente o no, en el sistema de control.
- Los eventos ocurren en “tiempo real” por lo que los procesos RT tienen un tiempo límite que especifica cuando debe comenzar a ejecutarse el proceso o cuando debe finalizar su computación.

SO de Tiempo Real (RTOS): Características

- **Determinismo y Reactividad**. Grado en el que el sistema puede resolver todos los procesos RT cumpliendo todos los plazos de tiempo.
- El determinismo tiene que ver con la velocidad de respuesta del RTOS frente a interrupciones: tiempo empleado en reconocer una interrupción. ¿siempre responde en el tiempo límite? Comportamiento predecible
  - Un sistema de control de airbag en un auto debe desplegarse dentro de 30 ms después del impacto (se sabe en cuánto tiempo responderá).
- La reactividad tiene que ver con el tiempo que tarda el RTOS en la RSI (rutina de servicio de interupción). ¿Responde rápido?
  - Un robot industrial que detecta un obstáculo debe detenerse inmediatamente (en microsegundos) para no chocar.
- Ambas son los factores determinantes del **tiempo de respuesta**.

SO de Tiempo Real (RTOS): Características

- **Control de la prioridad de los procesos RT**. El usuario debe poder controlar la prioridad del proceso RT.
- **Fiabilidad (tolerancia a fallos)**. La degradación en las prestaciones de un RTOS puede ser muy peligrosa. Por tanto, la **estabilidad** del sistema es crítica.
- Un RTOS es estable cuando, ante la imposibilidad de cumplir los plazos de tiempo de todos los procesos RT, el sistema cumple los plazos de los procesos más críticos.

SO para Sistemas Empotrados

- Un SO para sistemas empotrados (*embedded operating system, EOS*) esta especializado para utilizar computadores incluidos en sistemas más grandes.
- Un sistema empotrado (*embedded system*) es un computador que forma parte de una máquina, por ejemplo:
  - Un coche, ABS (Sistema antibloqueo de frenos), EPS (Dirección asistida eléctrica),...
  - Televisión digital, cámara digital,...
  - Sistemas de navegación GPS.

SO para Sistemas Empotrados

- Un EOS tiene una funcionalidad más limitada que un SO “normal” que viene determinada por el sistema empotrado.
- La principal característica que debe proporcionar es la **robustez**, en cuanto a la ejecución de procesos, ya que debe lidiar con restricciones de memoria y potencia de cómputo.
- Normalmente se les conoce como RTOS, porque realmente casi todos tienen requisitos de tiempo real.

SO para Sistemas Empotrados

RTOS	Tipo	Uso principal	Ejemplos famosos	Compatibilidad tiempo real
FreeRTOS	Open Source	IoT, microcontroladores	ESP32, STM32	🟩 Blando
VxWorks	Comercial	Aeroespacial, militar, telecom	Mars Rover	🟥 Duro
QNX	Comercial	Automoción, sistemas críticos	Coches, trenes	🟥 Duro
RTEMS	Open Source	Aeroespacial, satélites	NASA	🟥 Duro
Integrity	Comercial	Aviación, automoción, médico	Boeing 787	🟥 Duro
ThreadX	Comercial	IoT, embebidos, nube	Azure IoT	🟩 Blando
Zephyr	Open Source	IoT, wearables	Intel, Nordic	🟩 Blando
µC/OS-III	Comercial	Automoción, médico, industrial	Sistemas embebidos	🟥 Duro
NuttX	Open Source	Drones, UAVs	PX4 Autopilot	🟩 Blando/Medio
eCos	Open Source	Red, dispositivos embebidos	Routers, sensores	🟩 Blando
RIOT OS	Open Source	IoT, bajo consumo	Sensores IoT	🟩 Blando