Tema 1 – Estructuras de SO's

Estructura del SO

Arquitecturas monolíticas, microkernel y máquinas virtuales

Sistemas operativos de propósito específico

Sistemas Operativos
Alejandro J. León Salas, 2020
Lenguajes y Sistemas Informáticos

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 (**arquitecturas de SO**) y los beneficios de cada una.
- Distinguir diferentes tipos de **SO's de propósito específico**.

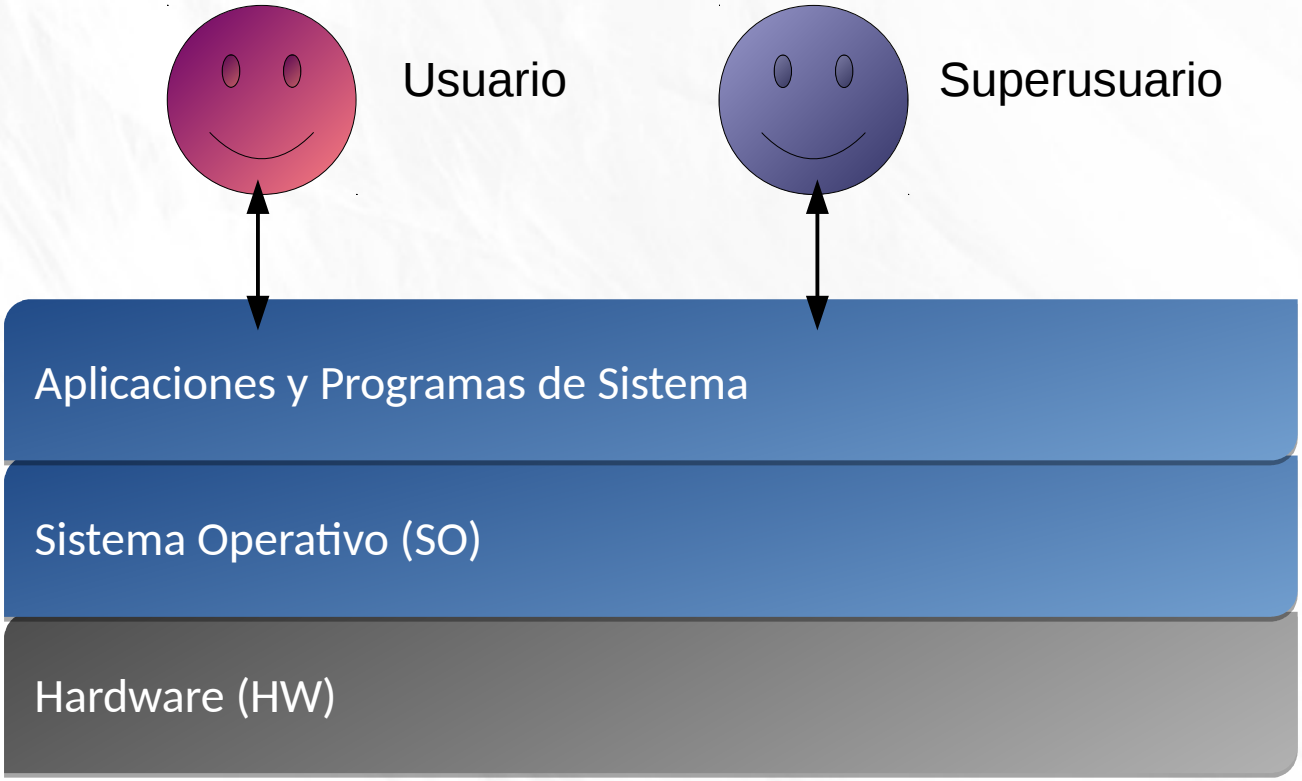
Bibliografía

- [Sta2005] W. Stallings. Sistemas Operativos. Aspectos Internos y Principios de Diseño (5/e). Prentice Hall, 2005.
- [Sta2006] W. Stallings, Organización y Arquitectura de Computadoras (7/e), Pearson Educación, 2006.
- [Car2007] J. Carretero et al., Sistemas Operativos (2ª Edición), McGraw-Hill, 2007.

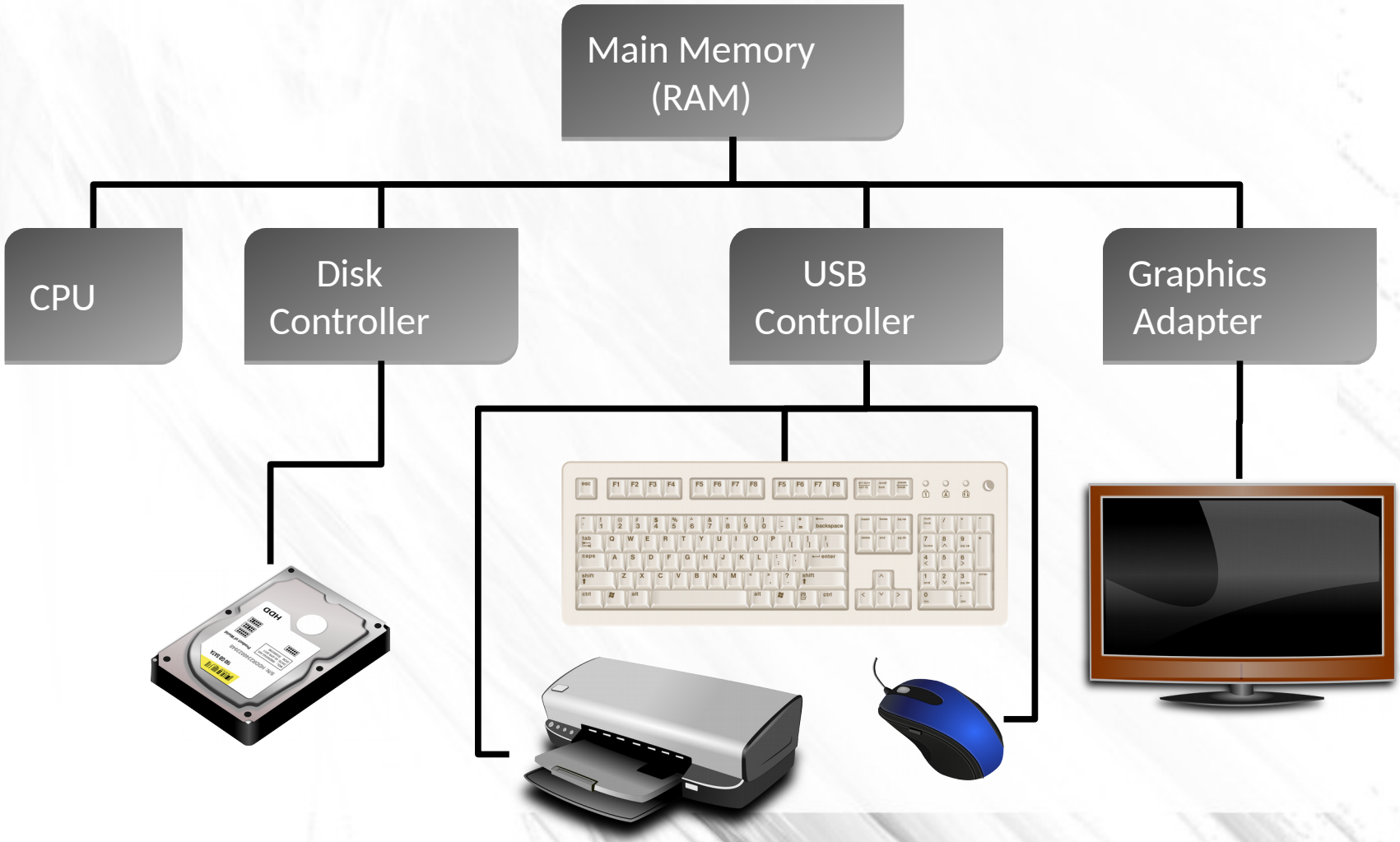
Contenidos

- Soporte hardware (HW) al sistema operativo.
- Estructura del sistema operativo.
- Arquitecturas monolíticas, microkernel y máquinas virtuales.
- Sistemas operativos de propósito específico.

Sistema informático (Computer system)



Sistema informático. Áreas funcionales



HW. Principales componentes

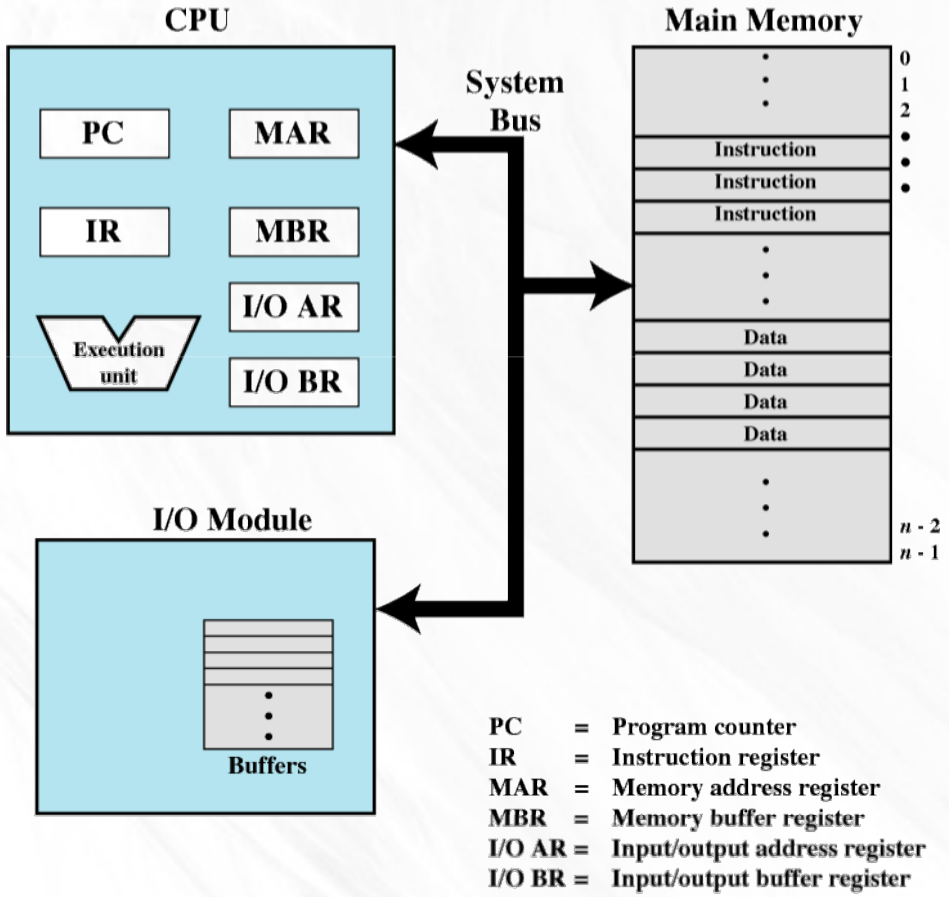
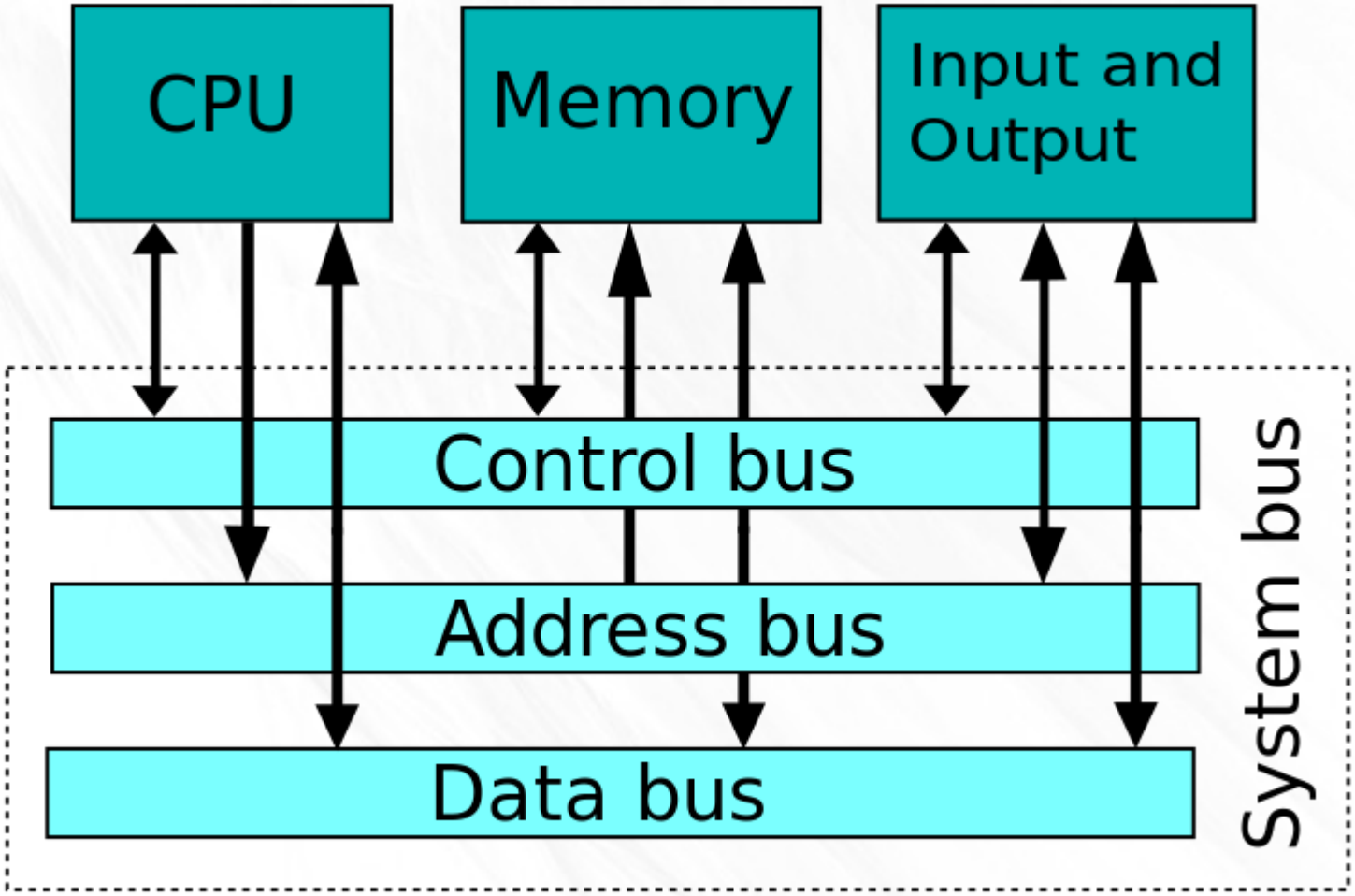


Figure 1.1 Computer Components: Top-Level View

From [Sta2005]

HW. Principales componentes



HW. CPU

Registros internos

- Registro de dirección de memoria, **MAR**, que contine 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

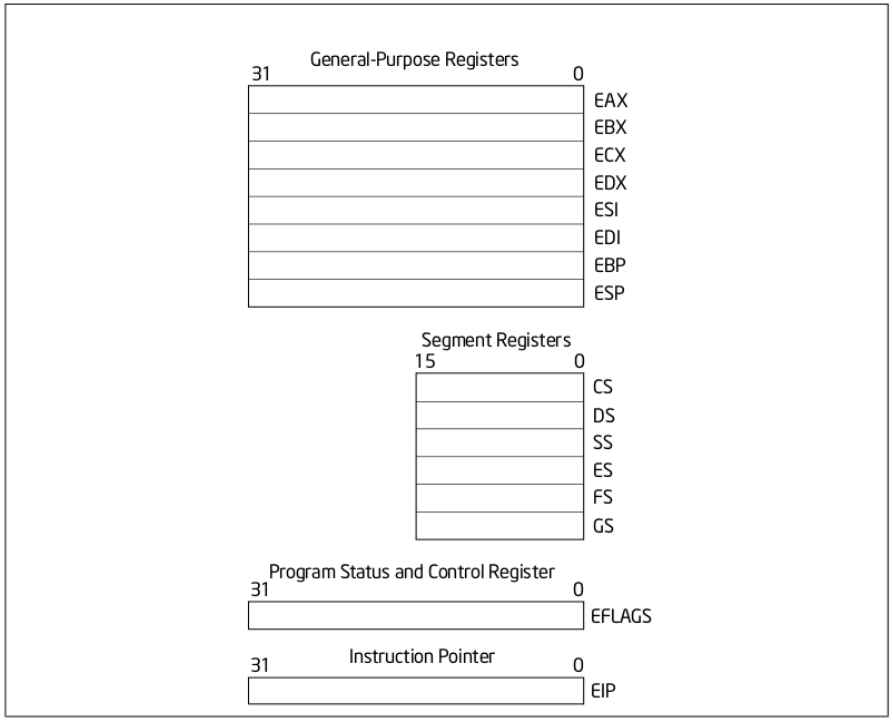
- Pueden ser accedidos por los programas y podemos clasificarlos en registros de datos y registros de direcciones.
- Entre los registros de direcciones podemos encontrar como importantes: registros índice, puntero a segmento y puntero a la pila.

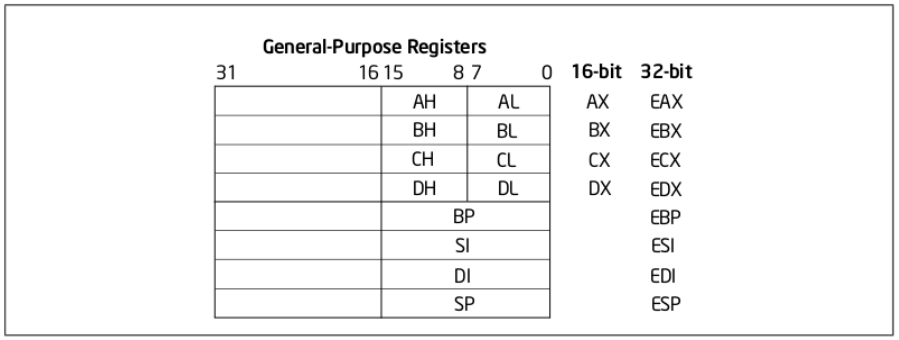
HW. CPU

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 fué a buscar (*fetched*).
- Program Status Word (**PSW**). Contine códigos de condición, información para interrupciones y modo de ejecución del procesador.
- **Ej.**: **IA-32** ofrece 16 reg. básicos: 8 reg. proposito general, 6 reg. de segmento, 1 registro de estado y control y 1 reg. CP.

HW. IA-32: General-purpose Regs.

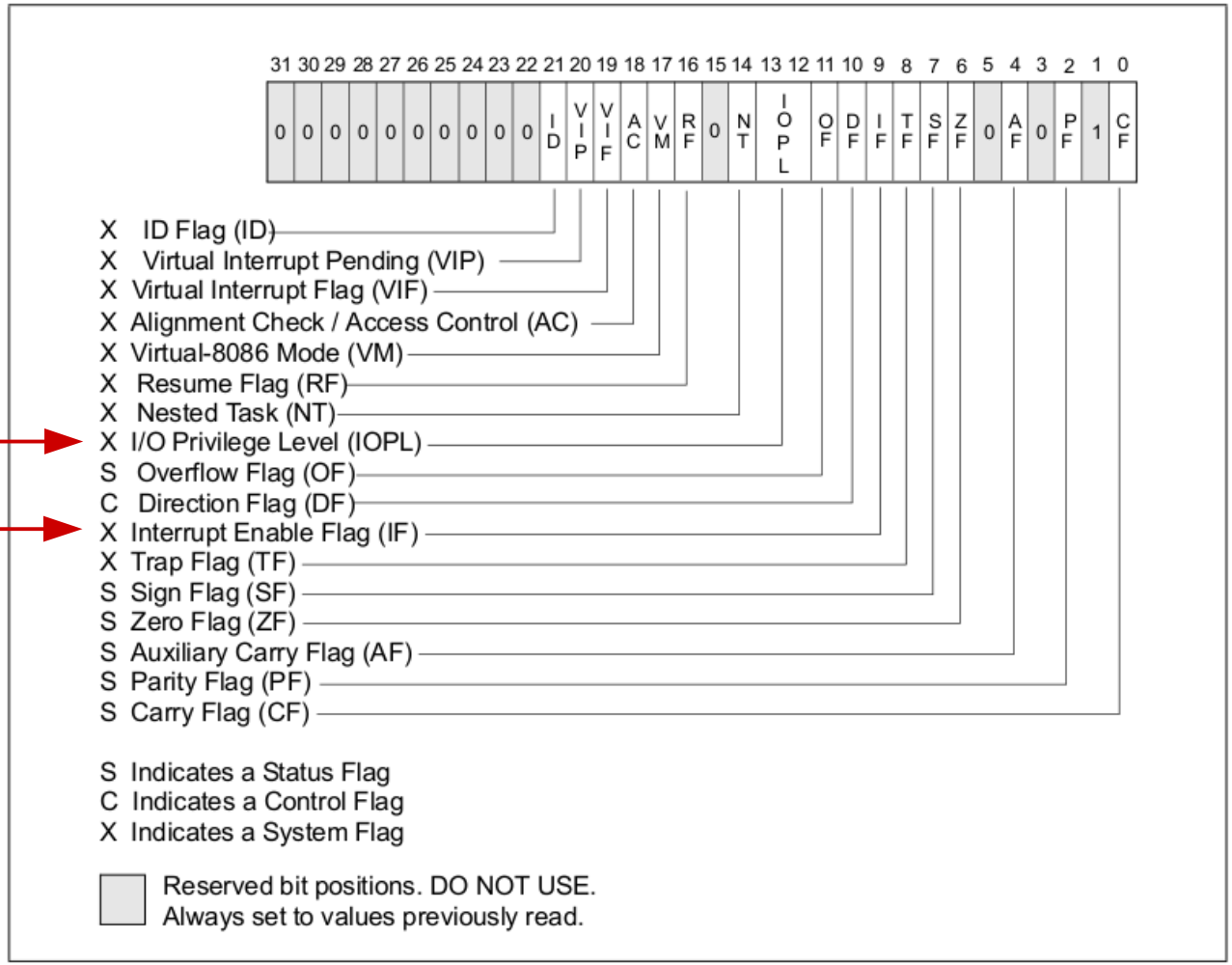




From Intel® 64 and IA-32 Manuals

- EAX – Acumulador
- EBX – Puntero a datos del segmento DS
- ECX – Contador para operaciones sobre strings y loops
- EDX – Puntero de I/O
- 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
- 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: EFLAGS register



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.
- **Espacio de direcciones** (*address space*). Conjunto de números que representa las direcciones de una 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.
- Escritura (**W**). La memoria recibe una dirección y un byte y sobreescribe el byte contenido en la celda indicada por la dir.

byte read(address i);

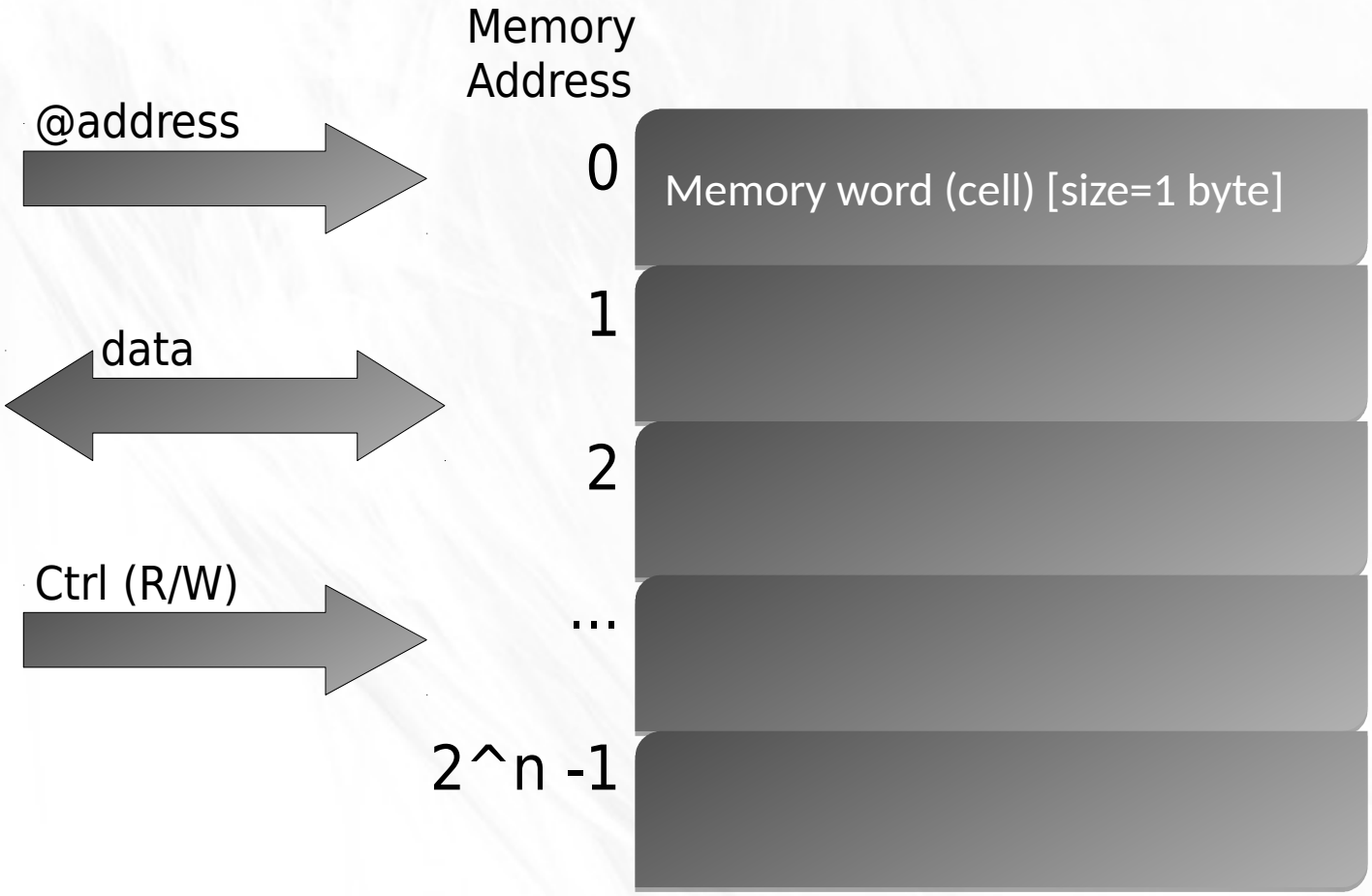
void write(address i, byte b);

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

HW. Memoria principal (RAM)



HW. Memoria principal (RAM)

Ejemplo espacio de direcciones IA-32

- En IA-32 los programas no acceden directamente a la memoria física sino indirectamente usando los **modelos de memoria**.
- ***Flat memory model***. Es un espacio lineal con direcciones consecutivas en el rango [0,2^32-1] (***linear address space***). Permite direccionar con granularidad de un byte.
- ***Segmented memory model***.
- ***Real-address mode memory model***.

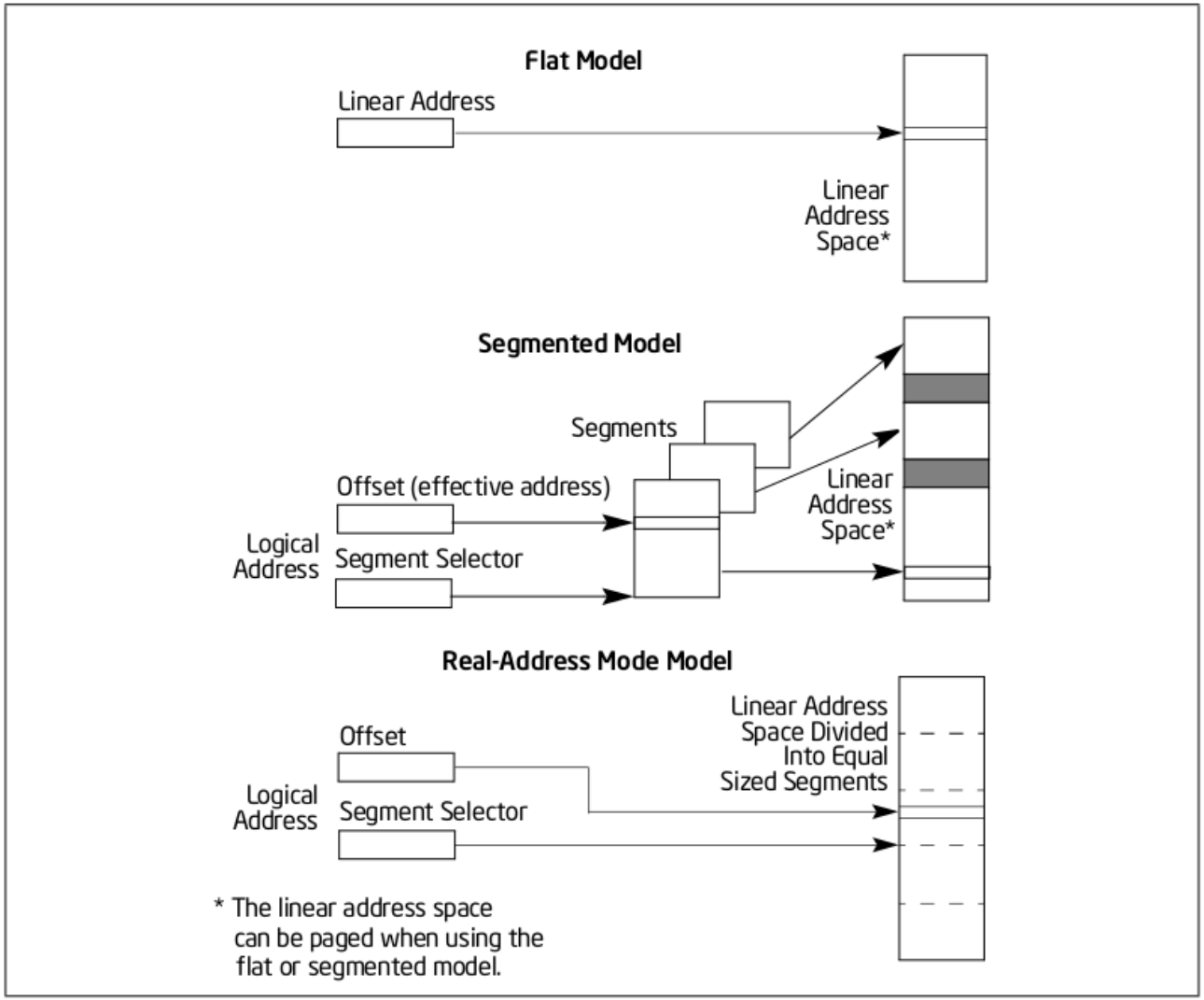
HW. Memoria principal (RAM)

- ***Segmented memory model***. Los programas ven el espacio de memoria como un grupo de espacios independientes llamados segmentos.
- Para acceder a un byte dentro de un segmento el programa utiliza una dirección bidimensional (dirección lógica): (selector de segmento,offset)
- IA-32 permite identificar 16383 (aprox. 2^14) segmentos de diferentes tipos y tamaños con un tamaño máximo por segmento de 2^32 bytes

HW. Memoria principal (RAM)

- ***Real-address mode memory model***. Proporcionado por compatibilidad hacia atrás con el procesador 8086.
- Implementación de segmentación con limitaciones en el tamaño de los segmentos, 64KB, y en el espacio de memoria final accesible, 2^20 bytes.

HW. Memoria principal (RAM)



From Intel® 64 and IA-32 Manuals

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 en sectores de disco.

HW. Módulo de E/S

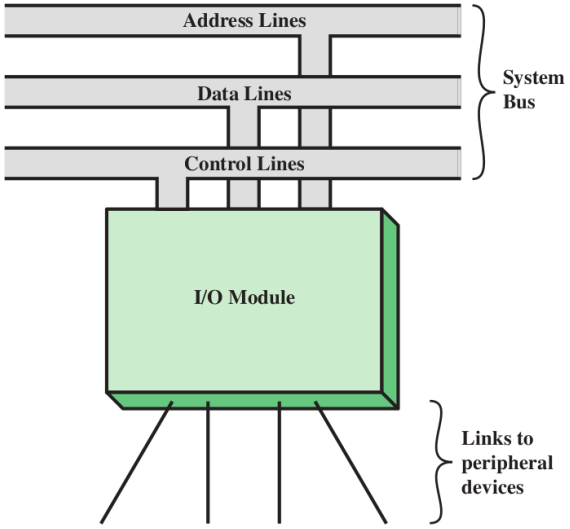


Figure 7.1 Generic Model of an I/O Module

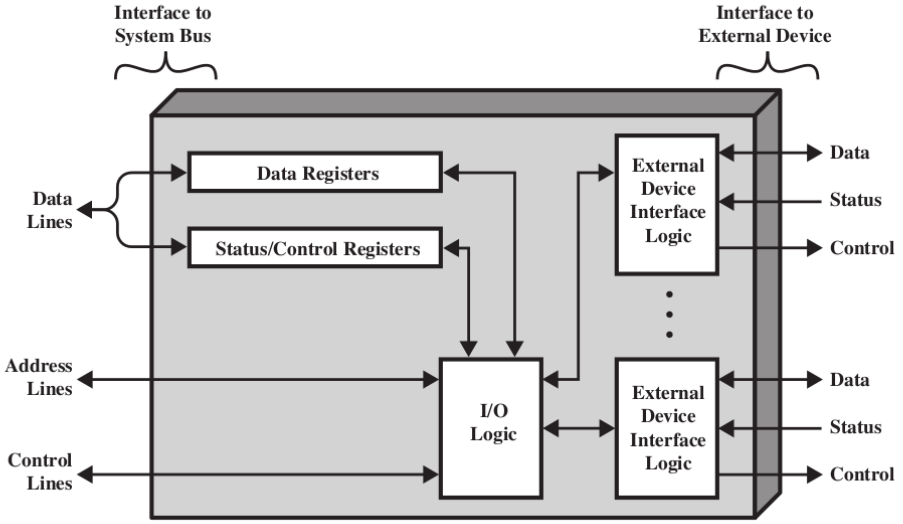


Figure 7.3 Block Diagram of an I/O Module

From [Stal05b]

HW. Principales técnicas de E/S

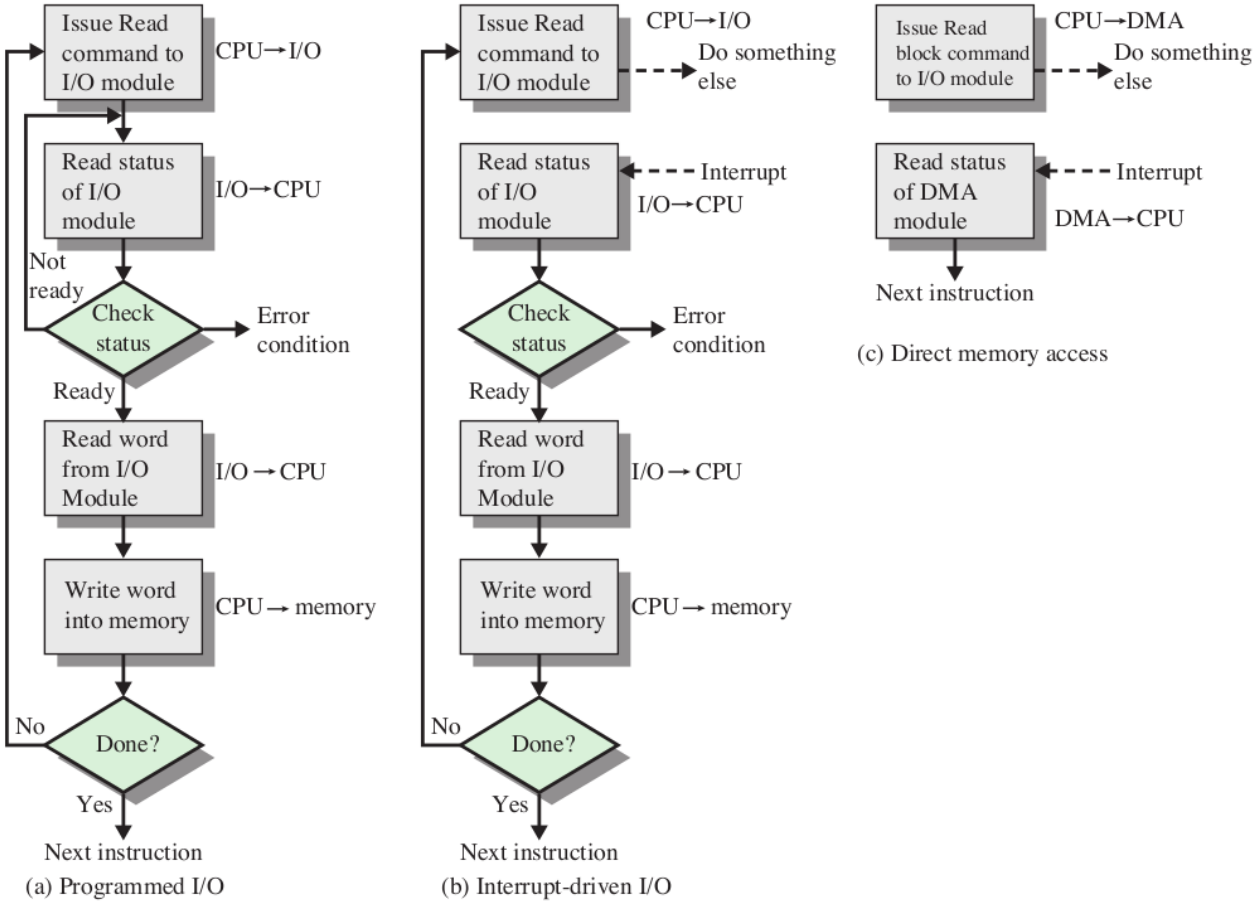


Figure 7.4 Three Techniques for Input of a Block of Data

From [Stal05b]

HW. Ciclo de ejecución de instrucción con interrupciones

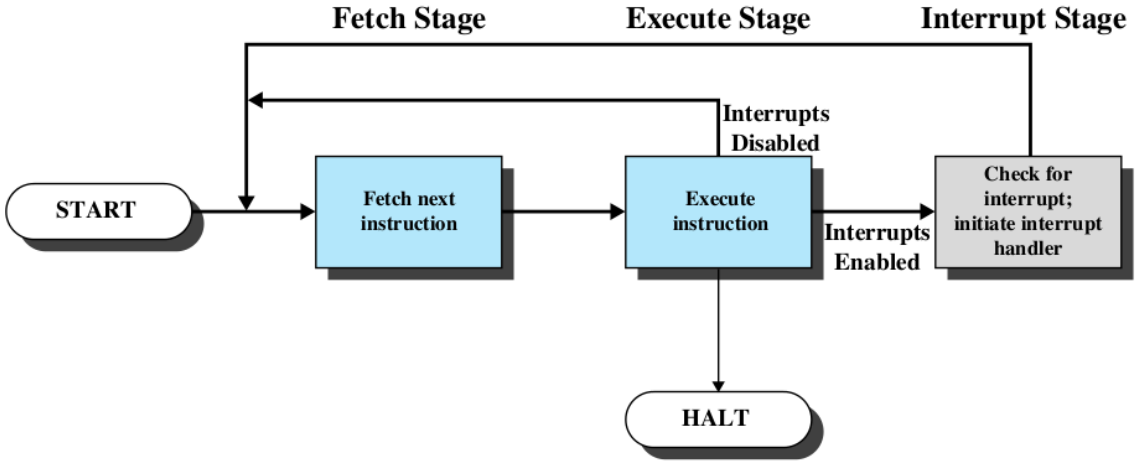


Figure 1.7 Instruction Cycle with Interrupts

From [Stal05a]

HW. Tratamiento de interrupciones

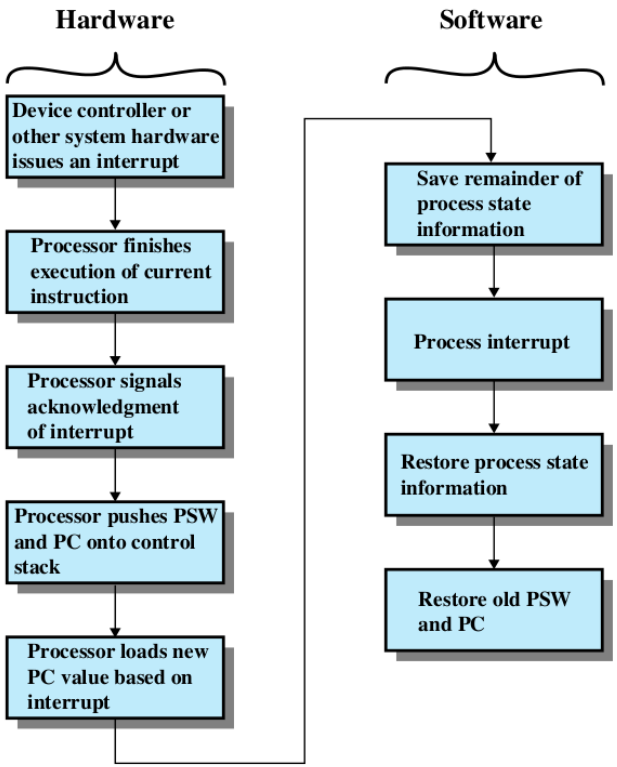


Figure 1.10 Simple Interrupt Processing

From [Stal05a]

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.

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 (pej. **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.

Contenidos

- Soporte hardware (HW) al sistema operativo.
- Estructura del sistema operativo.
- Arquitecturas monolíticas, microkernel y máquinas virtuales.
- Sistemas operativos de propósito específico.

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

- **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 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** 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 manteniendolo en almacenamiento secundario.

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).
  - Abrir y cerrar sesiones de trabajo con archivos, leer, escribir, variar el puntero de lectura/escritura.

Funcionalidad para archivos y directorios

- Gestionar la asignación y liberación de espacio en disco para mantener la información de los archivos y directorios.
- 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.
- 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.

Funcionalidad para recursos del SO

Planificación y gestión de recursos.

- **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).
- **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, viruses, 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

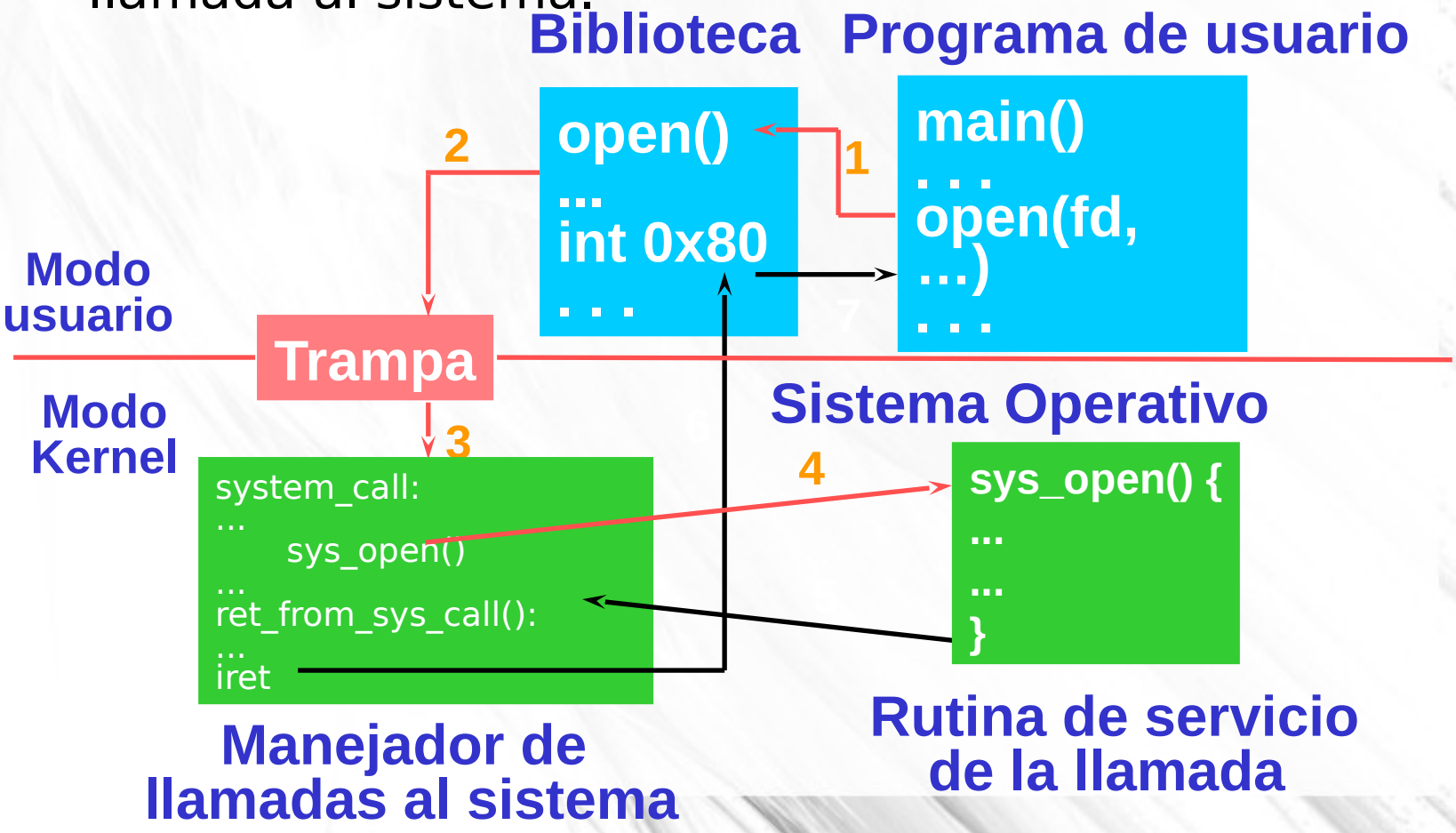
- La funcionalidad descrita previamente se suele agrupar en los siguientes componentes fundamentales.
  - Gestor de procesos.
  - Gestor de memoria.
  - Gestor de archivos.
  - Gestor de E/S.
  - Gestor de comunicaciones.

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) o órdenes (programas) que residen en el 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.
- Paso de parámetros:
  - En registros de la CPU.
  - Parámetros en memoria paso la dirección del bloque en un registro.
  - Parámetros en una pila.

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
```

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.

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

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

/* 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;
}
```

System call. Implementación

- Programa helloSyscall.S

```
# helloWorld assembly program for the Linux x86_64 architecture.
# Compile like this:
# $ gcc -c helloSyscall.S
# $ ld -o helloSyscall helloSyscall.o


.data                       # Initialized data section

msg:
    .ascii "Hello, world!\n"
    len = . - msg

.text                       # text section
    .global _start

_start:
    movq  $1, %rax
    movq  $1, %rdi
    movq  $msg, %rsi
    movq  $len, %rdx
    syscall                 # SYSCALL invokes an OS system-call handler at privilege level 0. $1 => write

    movq  $60, %rax
    xorq  %rdi, %rdi
    syscall                 # SYSCALL invokes an OS system-call handler at privilege level 0. $60 => exit
```

System call. Implementación

- Programa helloSyscall.asm

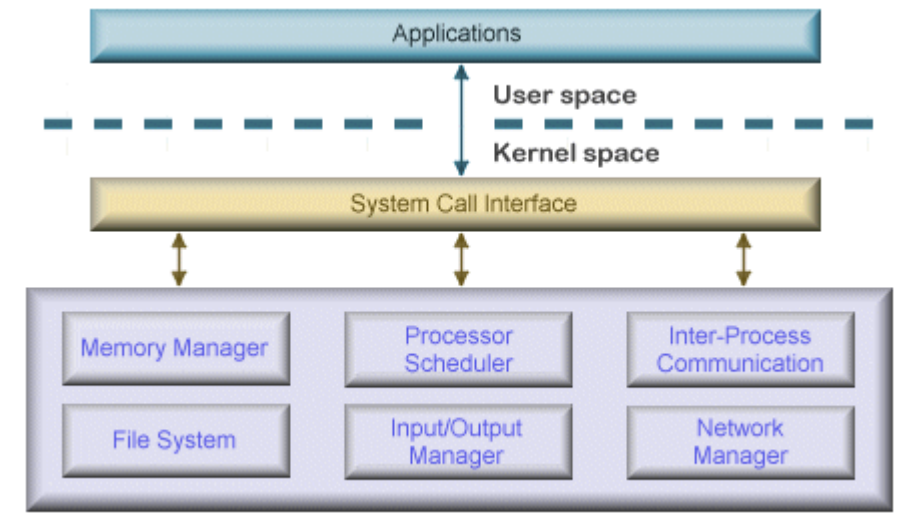
```
;; helloWorld assembly program for the Linux x86_64 architecture.
;; Compile like this:
;; $ nasm -f elf64 -o helloSyscall.o helloSyscall.asm
;; $ ld -o helloSyscall helloSyscall.o

section .data                   ; Initialized data section
    NEW_LINE db 0xa                 ; New line character (\n)
    msg db      "hello, world!"

section .text                   ; text section
    global _start
_start:
    mov     rax, 1
    mov     rdi, 1
    mov     rsi, msg
    mov     rdx, 14
    syscall                 ; SYSCALL invokes an OS system-call handler at privilege level 0. 1 => write
    mov    rax, 60
    mov    rdi, 0
    syscall                 ; SYSCALL invokes an OS system-call handler at privilege level 0. 60 => exit
```

Contenidos

- Soporte hardware (HW) al sistema operativo.
- Estructura del sistema operativo.
- Arquitecturas monolíticas, microkernel y máquinas virtuales.
- Sistemas operativos de propósito específico.

Arquitectura Monolítica

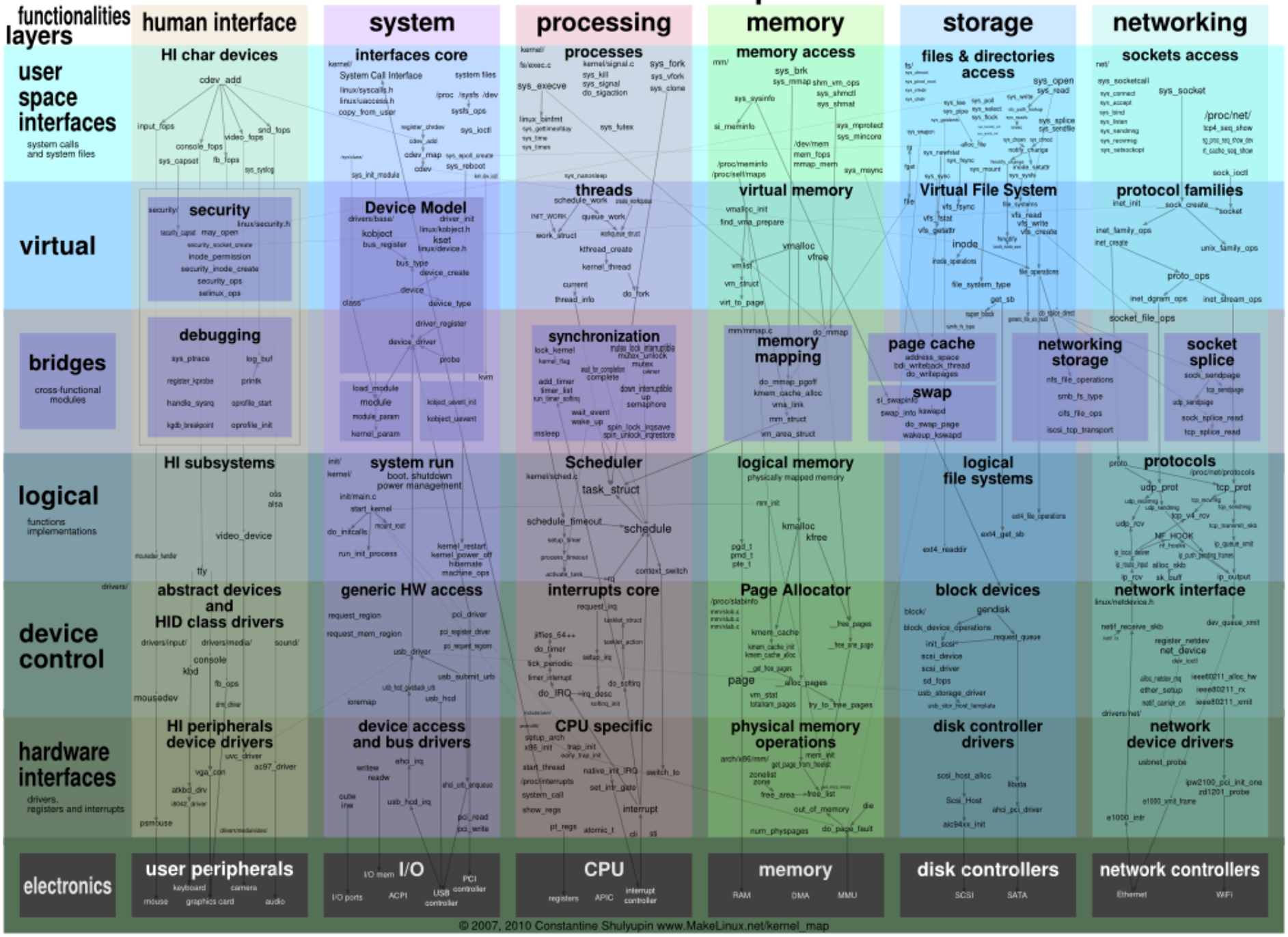
- El SO es un único programa que se ejecuta en el modo más privilegiado del procesador (**ring 0** en x86).
- Las dependencias entre los distintos módulos son complejas salvo para algunos elementos bien establecidos (¡NO oculta información!).
- El modelo de obtención de servicios es la llamada a procedimiento.



Arquitectura monolítica: Problemas

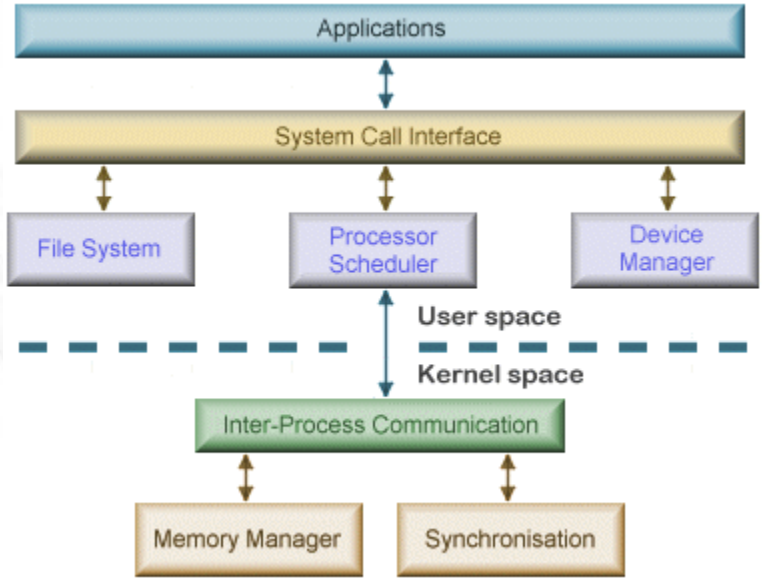
- La fuerte dependencia entre módulos provoca dificultades a la hora de comprender el código y de realizar modificaciones.
- Al ser un solo programa cargado en memoria el fallo de un módulo puede provocar la caida del sistema.

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 caida 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.

Concepto de virtualización

- El término virtualización se usó en 1960 para referirse a una máquina virtual (VM, *Virtual Machine*), entendida como un duplicado aislado de un computador real.
- Una VM es capaz de usar los recursos HW (CPU, memoria, almacenamiento y E/S) virtualizados (abstracción de los recursos reales o físicos) por el VMM.
- Un VMM (***Virtual Machine Monitor***) o hipervisor es el software que proporciona la abstracción de la máquina real a las VMs.

Concepto de virtualización

- Popek y Goldberg (1974) establecen requisitos para que un HW soporte virtualización: equivalencia, control de recursos y eficiencia.
- **Equivalencia**. Un programa ejecutándose sobre un hipervisor debe comportarse de forma idéntica a si se ejecutáse sobre la máquina física real equivalente.
- **Control de recursos**. El hipervisor es el único software que controla los recursos virtualizados.
- **Eficiencia**. La mayor proporción de instrucciones deben ejecutarse sin la intervención del hipervisor.

Tipos de virtualización

- **Hipervisor y Máquinas virtuales**. Capa software (hipervisor) que proporciona recursos virtuales (CPU, memoria, almacenamiento y dispositivos) a máquinas virtuales (SO,bibliotecas y aplicaciones).
- ***OS-level virtualization***. El kernel permite multiples instancias aisladas a nivel de usuario (*containers*). Un programa no verá todos los recursos del SO sino solamente los asignados al *container*.

Hipervisor y máquinas virtuales

- La virtualización actualmente se entiende como una capa de abstracción software situada entre el software a ejecutar y el hardware real.
- **El hipervisor crea un entorno de trabajo, la VM, para el software a ejecutar.**
- El hipervisor puede ser software, firmware o hardware y el ordenador que el que se ejecuta se denomina *host machine*, y cada máquina virtual se denomina *guest machine*.
- Más recientemente, el desarrollo y gestión de hipervisores y VMs se denomina *platform virtualization* o *server virtualization*.

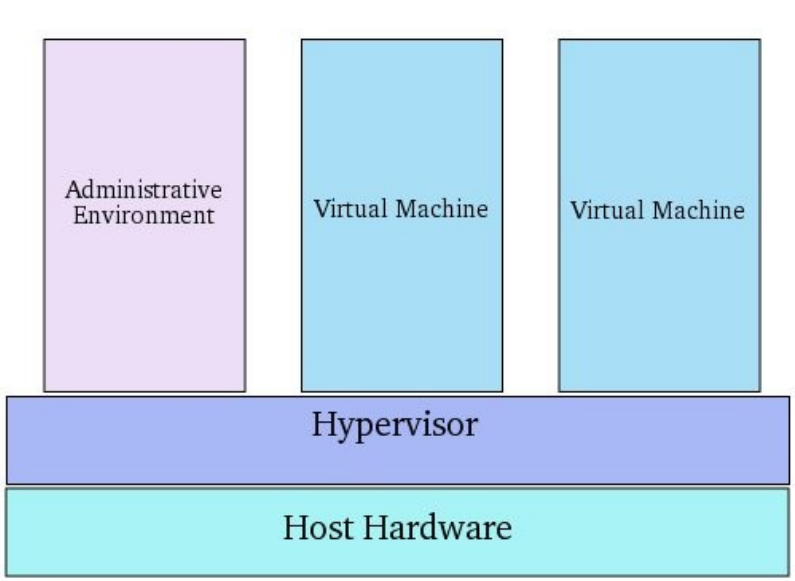
Hipervisor y máquinas virtuales

- La virtualización permite que un único ordenador ejecute distintos SOs o múltiples ejecuciones del mismo SO.
- La máquina virtual dispone de una serie de recursos virtuales proporcionados por el hipervisor y sobre ella se ejecuta el SO (*guest OS*), las bibliotecas y las aplicaciones.
- El ratio de consolidación indica el número de VM que proporciona un determinado hipervisor.
- Ejemplo: ratio 7:1 indica 7 VMs en una máquina anfitriona (*host machine*).

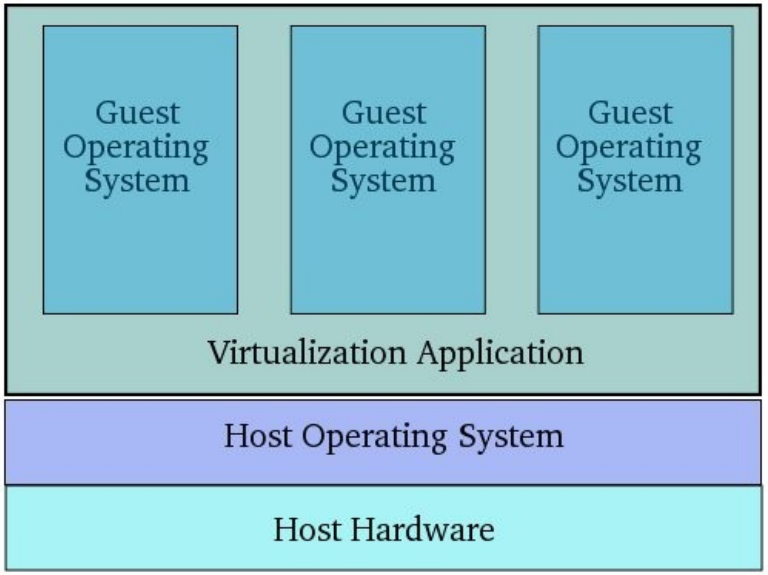
Tipos de hipervisor desde Popek y Goldberg

- **Tipo 1, *native* o *bare-metal***. Se ejecutan directamente en el host HW para proporcionar VM a los SO invitados. Ej. Xen y Vmware ESX/ESXi.
- **Tipo 2, hosted**. Se ejecutan sobre un SO convencional como el resto de programas y el SO invitado se ejecuta sobre la abstracción proporcionada por el hipervisor. Ej. VMware Workstation y VirtualBox .

Tipos de hipervisor desde Popek y Goldberg



Hipervisor Tipo 1 (*native* o *bare-metal)*



Hipervisor Tipo 2 (hosted*)*

El porqué de virtualizar

- Pemitir ejecutar diferentes SOs en el mismo HW fue el primer objetivo.
- Consolidación de servidores. Un HW servidor se virtualiza y tienes distintos SO actuando como *servers* en distintas VM.
- Desarrollo de *kernels*. VM es más fácil de controlar y observar que un HW real.
- *VM relocation*. VM con aplicaciones se traslada completamente a otro HW real.
- Evaluación de distintos SOs.

Un poco sobre implementación

- El hipervisor traduce las peticiones sobre los recursos virtuales que proporciona a peticiones sobre los recursos reales del hardwre.
- Esta traducción provoca degradación en el rendimiento.
- Una VM se representa mediante archivos:
  - Archivo de configuración: CPUs, memoria, dispositivos E/S accesibles, etc.
  - Archivo de almacenamiento: discos virtuales y su correspondiente soporte mediante archivos reales.

Funciones del hipervisor

- Gestión de la ejecución de MVs:
  - Planificación de MVs para ejecución en CPU
  - Gestión de la memoria virtual para aislar VMs.
  - Cambio de contexto y emulación de *timer* e interrupciones.
- Emulación de dispositivos y control de acceso.
- Ejecución de instrucciones privilegiadas.
- Gestión de Máquinas Virtuales.
- Administración del hipervisor.

Comparando hipervisores tipo 1 y tipo 2

- Los hipervisores tipo 1 obtienen mejor rendimiento que los tipo 2 ya que:
  - Disponen de todos los recursos para las VM.
  - Los múltiples niveles de abstracción entre el SO invitado y el HW real en el tipo 2 no permiten alto rendimiento de la máquina virtual.
- Los hipervisores tipo 2 permiten realizar virtualización sin tener que dedicar toda la máquina a dicho fin. Ejemplo, prueba de *kernels* o puesta a punto de servidores.

HW-assisted virtualization

- Primer HW para asistir a la virtualización incluido en IBM System/370 de 1972 como soporte para VM/370 (1er SO que proporciona VM).
- Arquitectura x86 no soportaba *HW-assisted virtualization* por lo que se desarrollaron soluciones software. El problema es que los SO que se ejecutan sobre las máquinas virtuales necesitan ejecutarse en ring 0… Soluciones:
  - Paravirtualización
  - *Full virtualization*.

Paravirtualización

- Técnica en la que el hipervisor proporciona una API (*hypercalls* en Xen) y el SO que se ejecuta en una VM utiliza dicha API.
- Implicaciones de esta técnica:
  - El código del *guest OS* debe estar disponible.
  - Reemplazar instrucciones que necesitan ring 0 por *hypercalls*: ej. cli por vm_handle_cli.
  - Recompilar el *guest kernel* y usarlo.

Full virtualization

- Esta técnica fue implementada en la primera generación de VMM (*hypervisor*) en x86.
- La idea es traducción binaria para atrapar y virtualizar la ejecución de instrucciones sensibles (ring 0), ej. cli.
- Las instrucciones sensibles se detectan (estática o dinámicamente) y se reemplazan con trampas al VMM.
- Esta técnica puede provocar sobrecarga en el rendimiento con respecto a una VM que se ejecute en HW que soporte virtualización, IBM System/370.
- Solución final: *HW-assisted virtualization*

HW-assisted virtualization: Ordenadores PC

- Intel (VT-x) and AMD (AMD-V) presentan HW que soporta virtualización (CPU flag vmx en Intel)
- ¿Cómo se entiende actualmente la virtualización? Cada elemento HW (ISA, I/O, interrupciones, memoria...) debe poder ser proporcionado por el hipervisor mediante una VM.
- El hipervisor (Super/Hiper OS) tipo 1 debe proporcinar:
- Almacenamiento independiente de SO para proporcionar recursos a las VM (también se les conoce como *virtual environments, VE*)
- *Switching* de VEs para asignar recursos gestionados por el hipervisor.

OS-level virtualization: containers

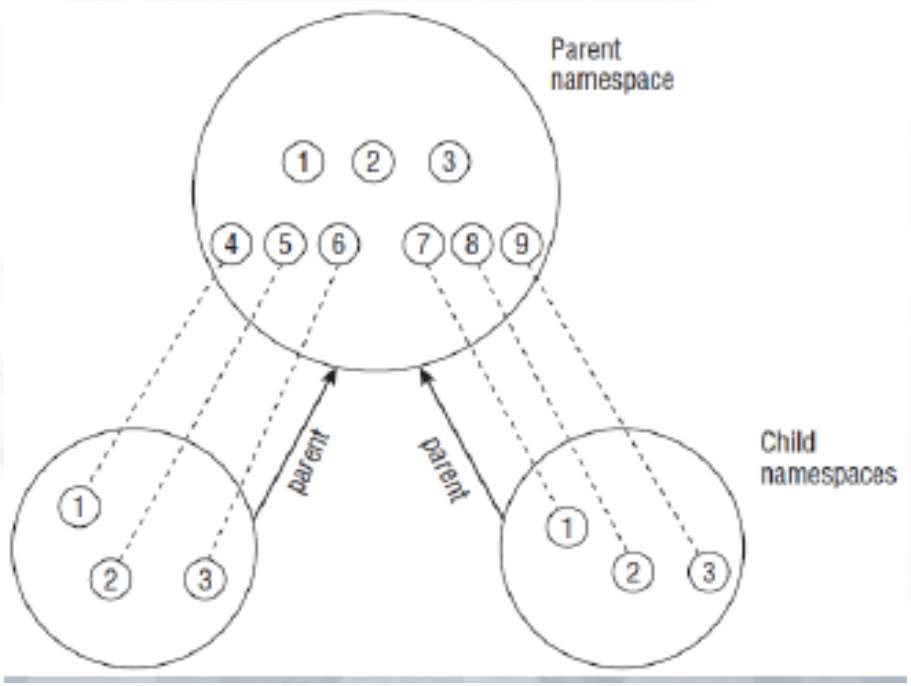
- Los containers proporcionan un entorno aislado para la ejecución de programas sobre el mismo SO.
- Linux *kernel control group* (`cgroup`). Permite la agrupación de procesos y gestión de recursos del sistema.
- `cgroup` permite que distintas jerarquías de procesos coesistan en el mismo SO.
- A cada jerarquía se le asignan un conjunto de recursos del sistema:
  - Memoria máxima.
  - Prioridad para CPU
  - Prioridad para dispositivos de E/S.

OS-level virtualization: Linux *Namespaces*

- Un espacio de nombres permite hacer visibles ciertos recursos del kernel de forma única dentro de dicho espacio.
- Algunos espacios de nombres incluidos actualmente:
  - System V IPC (Inter-process communication) y POSIX message queues.
  - Mounts. Sistema de archivos montado
  - Pid. Espacio de nombres para identificadores de procesos.
  - Userid. Permite limitar los recursos por usuario.

OS-level virtualization: Linux *Namespaces*

- Ejemplo de tres espacios de nombres y el pid del proceso varía dependiendo del espacio de nombres utilizado.



OS-level virtualization: UML y KVM

- UML. El SO anfitrión es un kernel modificado que contiene extensiones para controlar múltiples máquinas virtuales cada una con su SO invitado.
- *Kernel-based Virtual Machine* (KVM). Es una implementación *full virtualization* para Linux sobre HW x86 que contenga las extensiones correspondientes: Intel VT o AMD-V.

Contenidos

- Soporte hardware (HW) al sistema operativo.
- Estructura del sistema operativo.
- Arquitecturas monolíticas, microkernel y máquinas virtuales.
- Sistemas operativos de propósito específico.

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: tº empleado en reconocer una interrupción.
- La reactividad tiene que ver con el tiempo que tarda el RTOS en la RSI.
- 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, como por ejemplo:
  - Un coche, ABS, EPS,...
  - Televisión digital, cámara digital,...
  - Sistemas de navegación GPS.

SO para Sistemas Empotrados

- Un EOSs 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.

Computación de altas prestaciones

- Las tareas en High-performance Computing (**HPC**) se caracterizan por estar compuestas de un grupo de procesos fuertemente acoplados para resolver la tarea, por lo que necesitan una conexión de alta velocidad entre nodos de cómputo.
- Las tareas en High-throughput Computing (**HTC**) se caracterizan por estar compuestas de trabajos secuenciales que se pueden planificar individualmente en diferentes nodos de cómputo.

Arquitectura multiprocesador

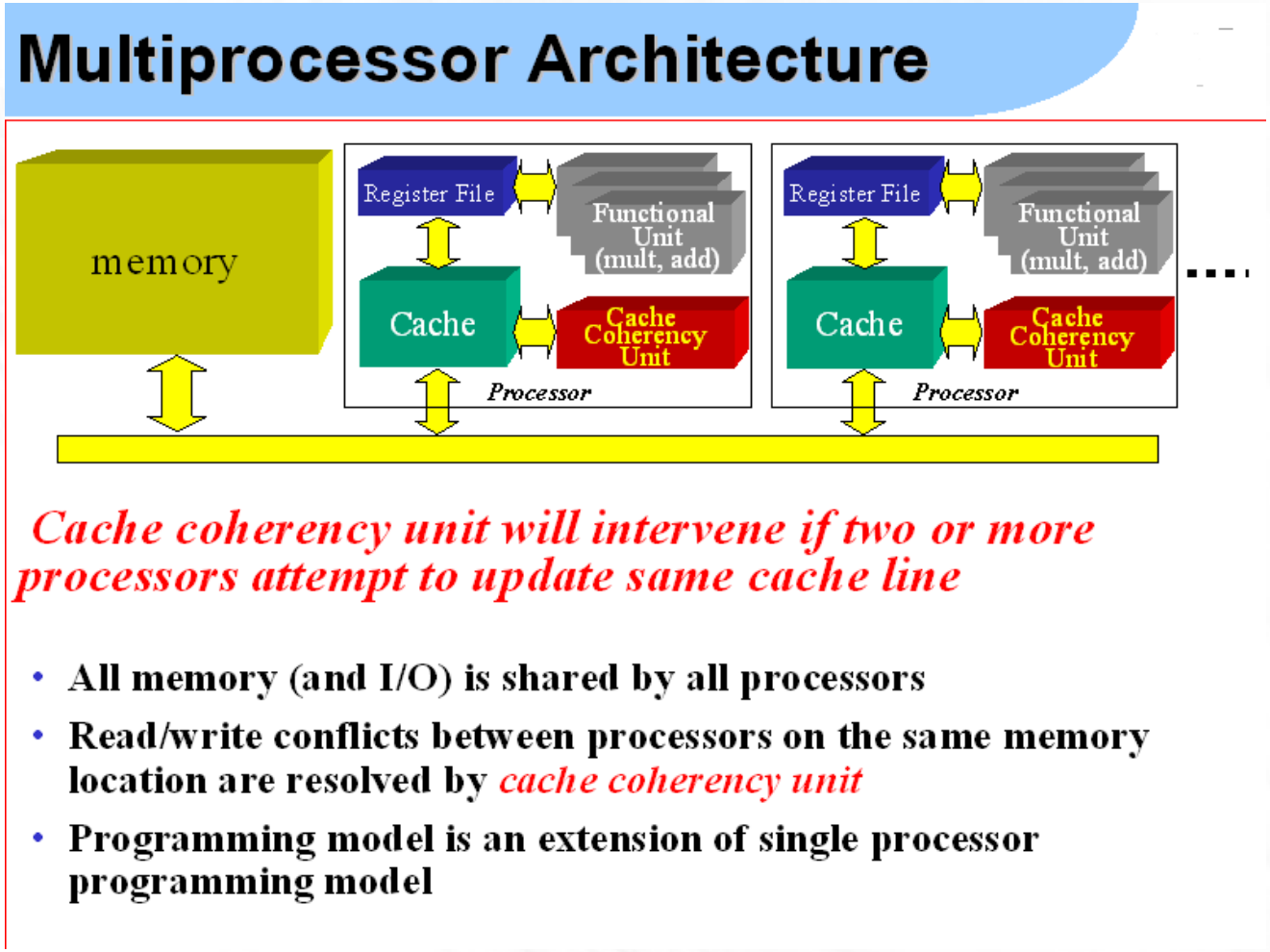
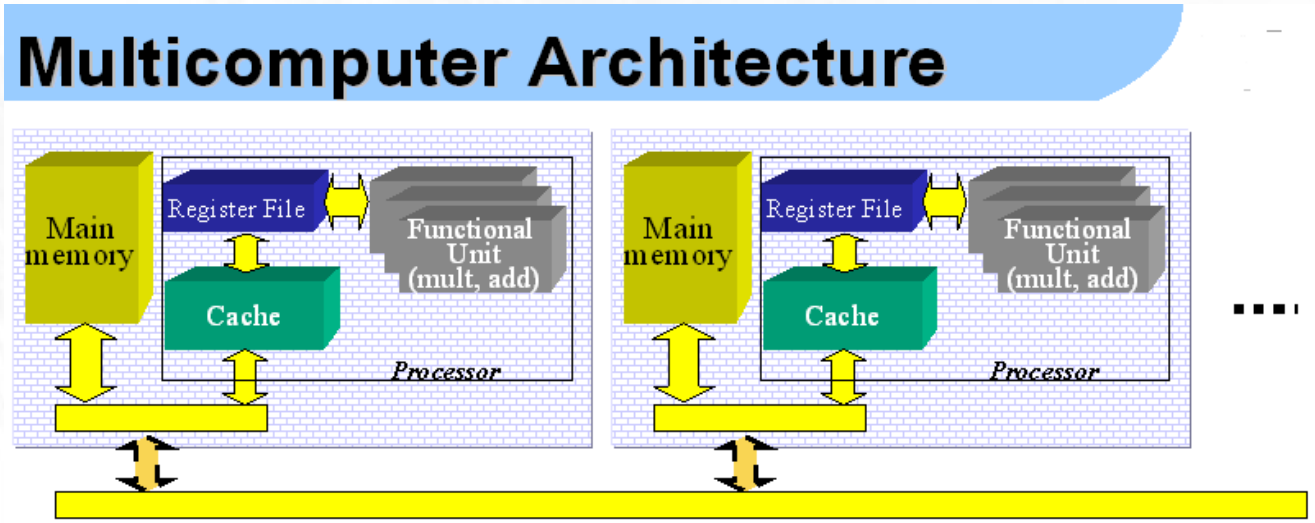


Imagen cortesía de http://sc.tamu.edu/

Arquitectura multicomputador



- All memory and I/O path are independent
- Data movement across the interconnect is "slow"
- Programming model is based on *message passing*
  - Processors explicitly engage in communication by sending and receiving data

Imagen cortesía de http://sc.tamu.edu/

Sistemas Operativos para HPC

- Se plantean dos alternativas de cara la procesamiento paralelo (multiprocesamiento):
  - Procesamiento Simétrico (SMP) - cada procesador ejecuta una copia idéntica del SO - buen rendimiento
  - Asimétrico (ASMP) - Un procesador **maestro** ejecuta el SO, los procesadores **esclavos** ejecutan procesos de usuario. Peor escalabilidad.

Sistemas Operativos para HPC

- Soportan aplicaciones paralelas que desean obtener aumento de velocidad de tareas computacionalmente complejas.
- Necesitan primitivas básicas para dividir una tarea en múltiples actividades paralelas.
- Proporciona una comunicación y sincronización eficiente entre esas actividades.
- Requieren mecanismos de Tolerancia a fallos.