

Apuntes de Arquitectura de Sistemas

Daniel Monjas Miguélez

March 19, 2022

Contents

1 Tema 1: Soporte Hardware	3
1.1 Motivación	3
1.2 Clasificación	5
1.2.1 Clasificación "práctica" de arquitecturas paralelas	5
1.2.2 Clasificación Arquitectura de Computadores	7
1.2.3 Componentes	11
1.2.4 Procesador	12
1.2.5 Memoria	17
1.2.6 Interacción	27
2 Tema 2: Introducción a los sistemas operativos	40
2.1 Abstracciones	40
2.2 Llamadas al sistema	48
3 Tema 3: Historia de los Sistemas Operativos	51
3.1 Definición	51
3.2 Historia	51
3.2.1 Primera generación (1945-1955): Tubos de vacío y paneles	51
3.2.2 Segunda generación (1955-1965): Transistores y sistemas por lotes	51
3.2.3 Tercera generación (1965-1980): Circuitos integrados y multiprogramación	52
3.2.4 Cuarta generación (1980-hoy): Ordenador personal (era μ)	52
3.3 Estructura	53
3.4 Ejemplos	61
3.5 Comparativa	66
3.5.1 Coste estructural de una llamada al sistema: caso monolítico	66
3.5.2 Coste estructural de una llamada al sistema: caso micronúcleo	67
3.5.3 Coste estructural: multiservidor	68

1 Tema 1: Soporte Hardware

1.1 Motivación

matriz

```
int matriz[10000][10000];
```

inicialización a cero: **bien**

```
for (unsigned i = 0; i < 10000; ++i)
    for (unsigned j = 0; j < 10000; ++j)
        matriz[i][j] = 0;
```

inicialización a cero: **mal**

```
for (unsigned i = 0; i < 10000; ++i)
    for (unsigned j = 0; j < 10000; ++j)
        matriz[j][i] = 0;
```

Para una matriz de tamaño 10000×10000 :

	matriz[i][j]		matriz[j][i]	
Mac OS X 1.25GHz PPC G4 512MB RAM	user	0.45	user	17.413
	system	1.15	system	2.037
	real	1.84	real	20.097
Linux 2 x 3GHz P4 Xeon 4 GB RAM	user	0.42	user	12.25
	system	0.73	system	0.733
	real	1.2	real	12.99
Sun OS 2 x 1GHz UltraSparc 8 GB RAM	user	1.645	user	45.495
	system	0.89	system	0.885
	real	2.87	real	47.725
Windows XP 2 x 3GHz P4 Xeon 4 GB RAM	user	0.843	user	9.937
	system	0.25	system	0.250
	real	1.125	real	10.344
Linux (casa) 2GHz AMD Athlon64 1 GB RAM	user	0.224	user	13.317
	system	0.524	system	0.660
	real	0.799	real	14.554

La causa de este fenómeno es la forma en que C gestiona la memoria, pues C/C++ almacenan las matrices por filas. Ejemplo: `int m[4][4]`

m[0][0]	m[0][1]	m[0][2]	m[0][3]
m[1][0]	m[1][1]	m[1][2]	m[1][3]
m[2][0]	m[2][1]	m[2][2]	m[2][3]
m[3][0]	m[3][1]	m[3][2]	m[3][3]

Memoria Virtual

Es una utilidad que permite a los programas direccionar la memoria desde un punto de vista lógico, sin importar la cantidad de memoria principal física disponible. Se concibió como método para tener múltiples trabajos de usuario residiendo en memoria principal de forma concurrente, de forma que no exista un intervalo de tiempo de espera entre la ejecución de procesos sucesivos, es decir, mientras un proceso se escribe en almacenamiento secundario y se lee el sucesor. Se introdujeron los sistemas de paginación, que permiten que los procesos se compriman en un número determinado de bloques de tamaño fijo, denominados páginas. Un programa referencia a una palabra por medio de una

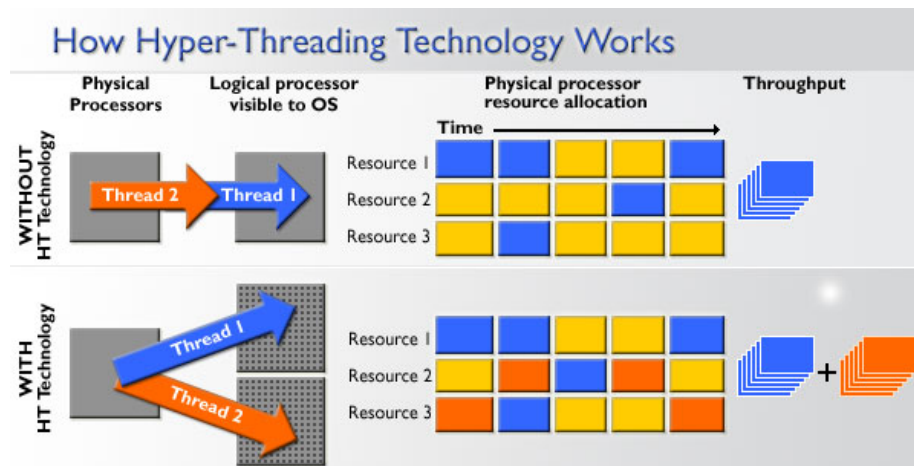
dirección virtual, que consiste en un número de página y un desplazamiento dentro de la página.

Todas las páginas de un proceso se mantienen en disco. Cuando un proceso está en ejecución, algunas de sus páginas se encuentran en memoria principal, y si se referencia a una página que no está en memoria principal el hardware de gestión de memoria lo detecta y permite que la página que falta se cargue (carga bajo demanda).

1.2 Clasificación

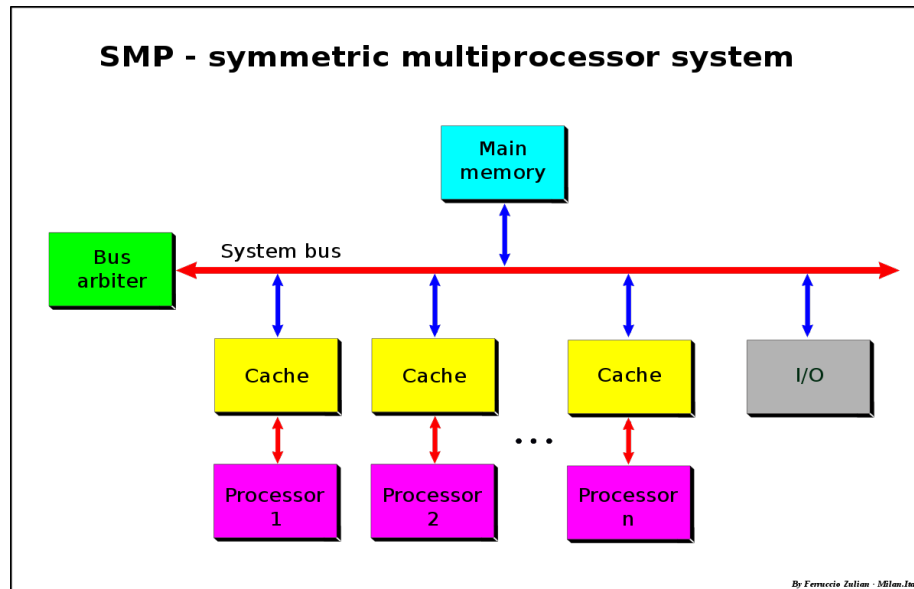
1.2.1 Clasificación "práctica" de arquitecturas paralelas

- Multiprocesadores de memoria compartida:
 - SMT(Simultaneous Multithreading/Hyperthreading): permite a una única CPU ejecutar varios flujos de control. Esto requiere tener múltiples copias de algunos componentes hardware de la CPU, como contadores de programa y registros de archivo, mientras otras partes siguen siendo únicas como las unidades que realizan aritmética con punto flotante. Cuando un procesador tiene Hyper-Threading puede tener de 2 a 64 hebras (puede tener más, veanse procesadores de servidor), dependiendo del número de núcleos físicos del mismo.

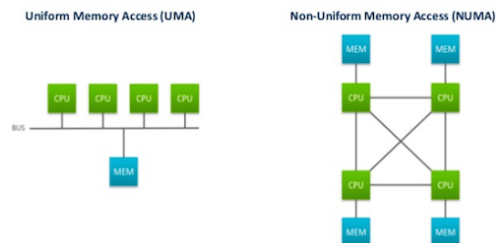


- SMP(Symmetric Multi-Processing): el núcleo puede ejecutar en cualquier procesador, y normalmente cada procesador realiza su propia planificación del conjunto disponible de procesos e hilos. El núcleo puede construirse como múltiples procesos o múltiples hilos, permitiéndose la ejecución de partes del núcleo en paralelo. El enfoque SMP complica el sistema operativo, ya que debe asegurar que dos procesadores no seleccionan un mismo proceso y que no se pierde ningún proceso

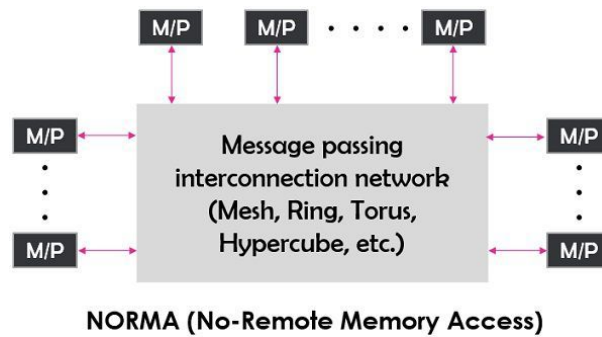
de la cola. Se deben emplear técnicas para resolver y sincronizar el uso de los recursos. Suelen tener entre 2 y 256 procesadores.



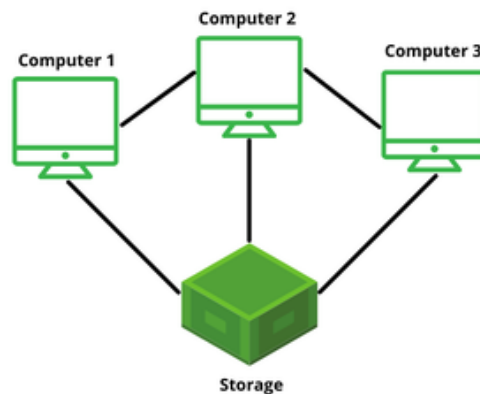
- UMA/ccNUMA (Uniform Memory Access/Cache Coherent UMA): Se define como la situación en la cual el acceso a cualquier RAM desde CPU toma siempre la misma cantidad de tiempo. Suele tener entre 2 y 4096 procesadores.
- Multiprocesadores masivamente paralelos:
 - NUMA/ccNUMA (Non Uniform Memory Access/ Cache Coherent NUMA): A diferencia de UMA, algunas partes de la memoria pueden tomar más tiempo de acceso que otras, creando una penalización en el rendimiento. Esta penalización se puede minimizar por medio de la administración de recursos.



- Paso de mensajes/NoRMA (No Remote Memory Access): en las arquitecturas NoRMA, el espacio de direcciones global no es único y la memoria no es globalmente accesible desde todos los procesadores. El acceso a módulos de memoria remotos es solo posible indirectamente a través del paso de mensajes por medio de la red de interconexión a otros procesadores, lo que en respuesta recibirá los datos buscados en un mensaje de respuesta.



- Cluster → +10M procesadores. Al igual que los sistemas multiprocesadores, los sistemas clústeres juntan múltiples CPUs para conseguir un trabajo computacional. La diferencia respecto a los sistemas multiprocesadores es que los clústeres se componen de dos o más sistemas individuales unidos juntos, a los que se denominan nodos.
 - GPU_s



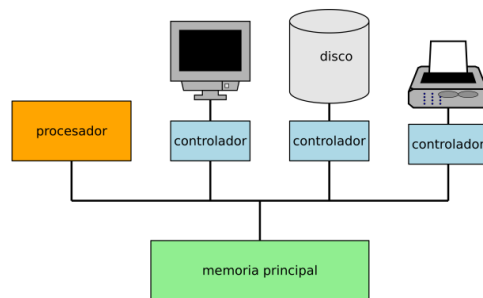
1.2.2 Clasificación Arquitectura de Computadores

- Sistemas monoprocesador

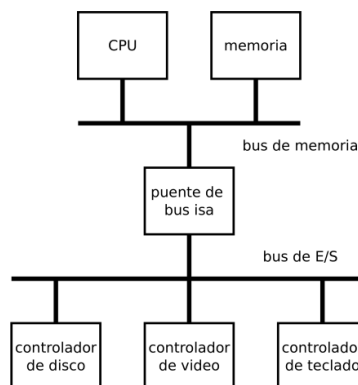
- Bus único
- Buses separados/especializados
- Sistemas multiprocesador
 - Multiproceso simétrico (SMP)
 - Multihebra simultánea (SMT)
 - Multinúcleo (SMP)
- Sistemas distribuidos

Sistema monoprocesador: Es el modelo más simple, pues conecta todo en un bus común.

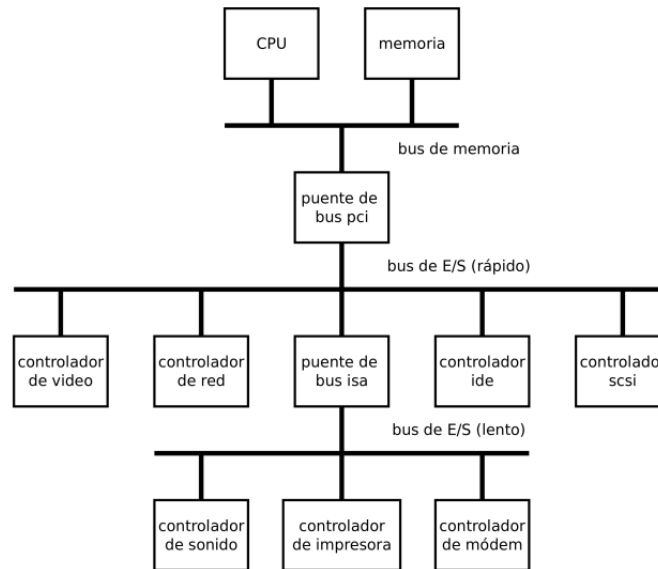
- Ventaja → precio.
- Inconveniente → infrautilización de componentes por la diferencia de velocidad.



Una posible solución para la diferencia de velocidad es aislar los componentes por velocidad y conectarlos por medio de un puente

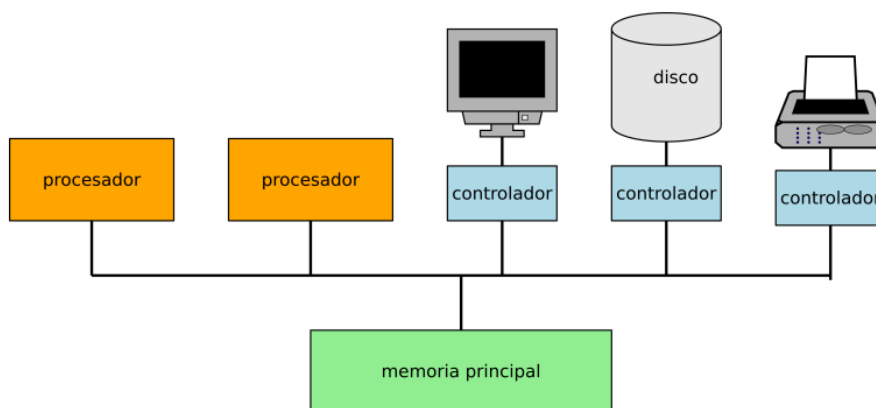


Otra posible solución incluso mejor es separar el bus de E/S en dos buses en función de los dispositivos E/S más rápidos y más lentos, y conectar ambos buses por medio de un puente isa.

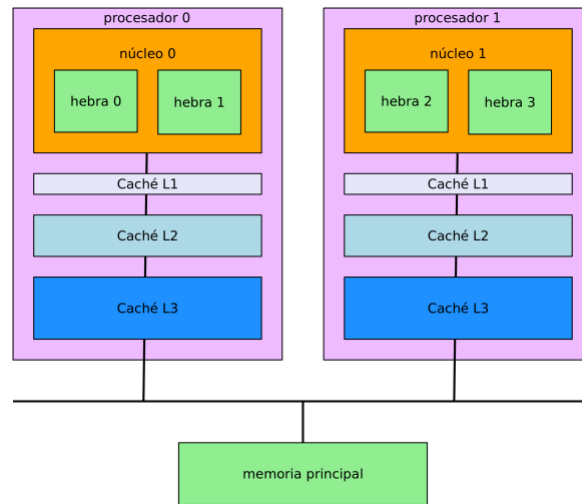


Sistema multiprocesador: multiproceso simétrico. Lo más simple es conectar todos los elementos a un bus común.

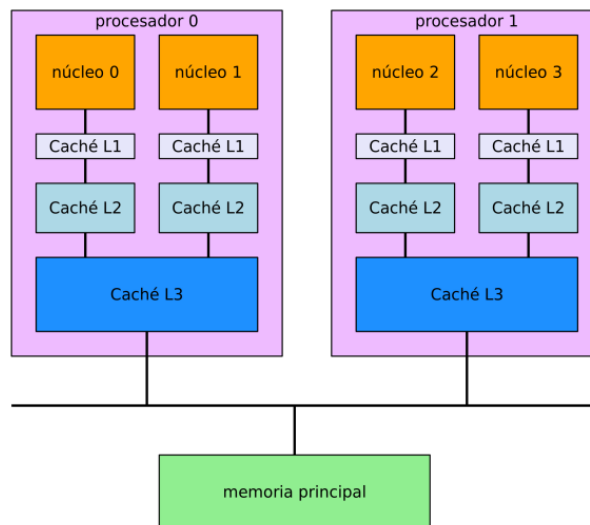
- Ventaja → precio.
- Inconveniente → se agrava la infrautilización de componentes.



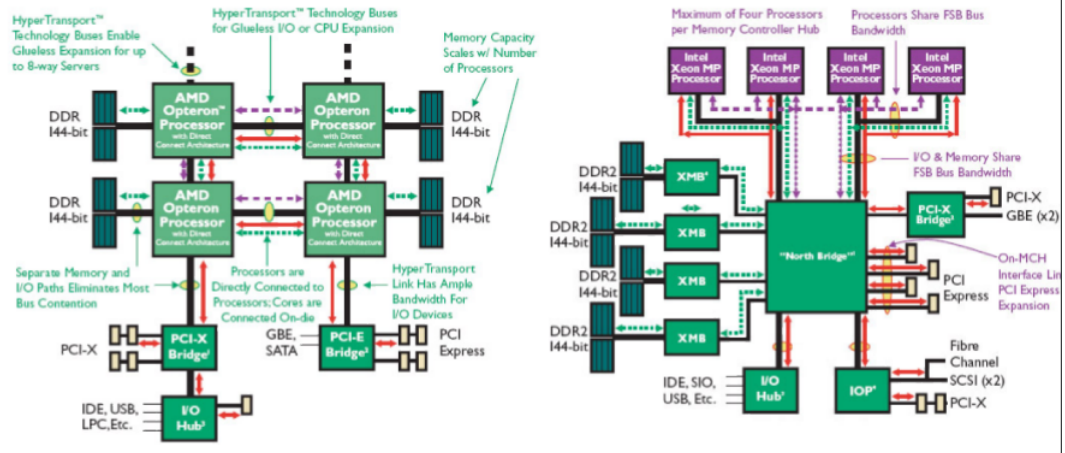
Sistemas multiprocesador: multihebra simultánea



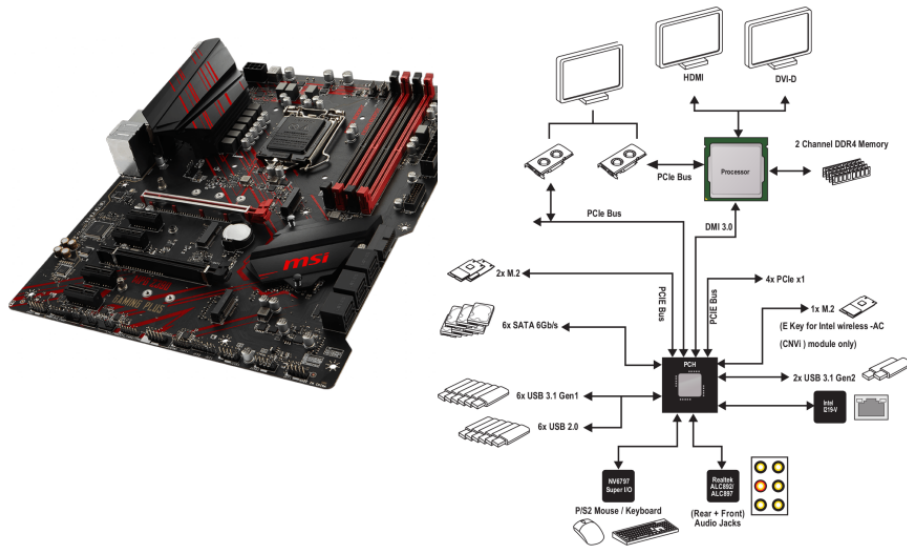
Sistemas multiprocesador: multiproceso simétrico



Sistemas multiprocesador actuales



Arquitecturas de un sistema actual



1.2.3 Componentes

Componentes básicos

- Procesadores
- Jerarquía de memoria.

Surge el problema de que cuanto menor es el tiempo de acceso mayor es el coste por bit, y que cuanto mayor es la capacidad menor la velocidad de acceso. Para lidiar con este dilema surge la jerarquía de memoria. En la jerarquía de memoria según se desciende disminuye el coste por bit, aumenta la capacidad, aumenta el tiempo de acceso y se reduce la frecuencia de acceso a ese nivel de la jerarquía por parte del procesador.

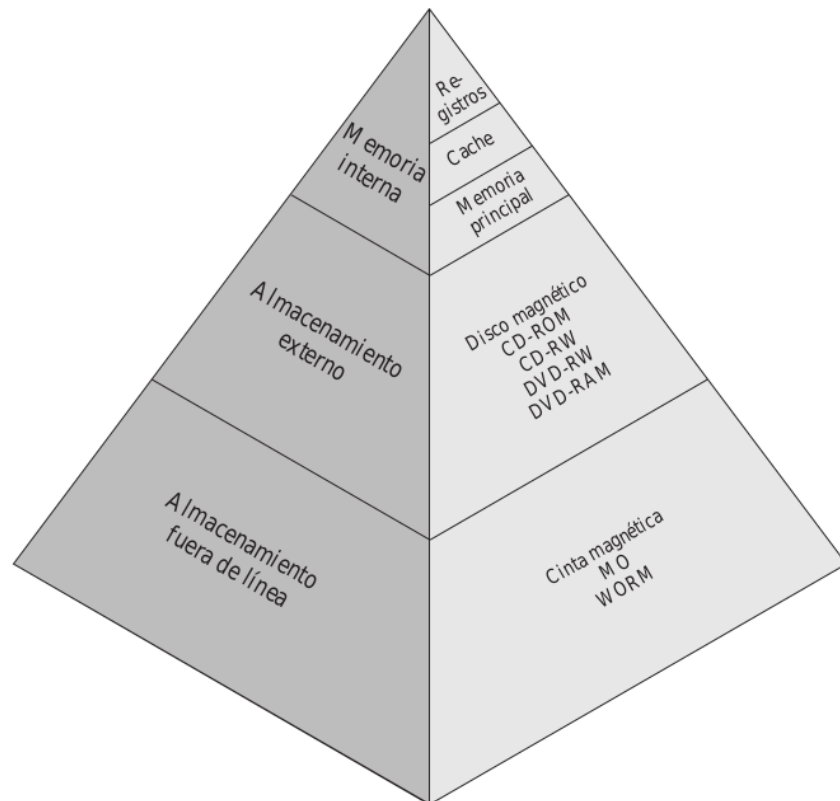


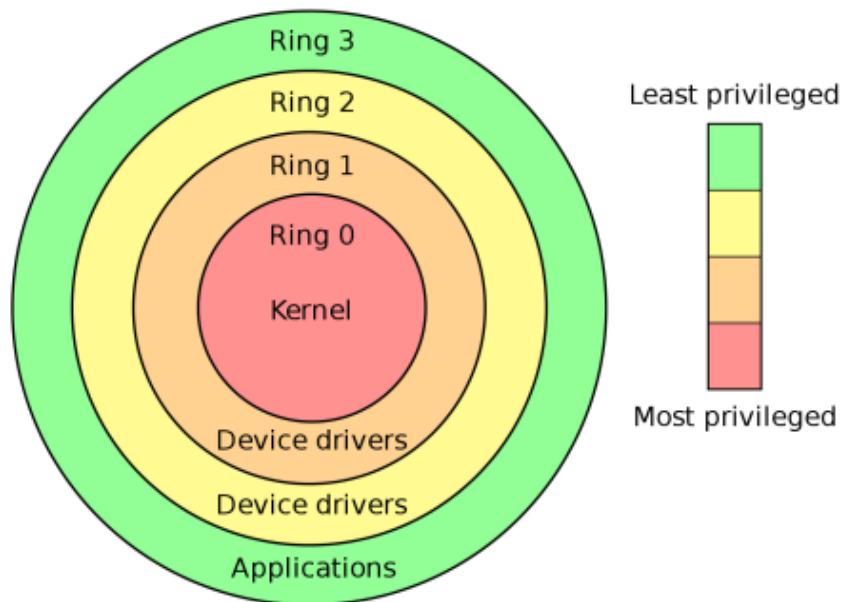
Figura 1.14. La **jerarquía de memoria**.

- Buses de interconexión: AGP, Hypertransport, IDE, IEEE 1394, ISA, M.2, PCI, PCIe, SATA, SCSI, USB, ...
- Entrada/Salida: controladores, canales de DMA, procesadores de E/S,...
- Periféricos: altavoz, disco, impresora, micrófono, monitor, ratón, teclado, ...

1.2.4 Procesador

Interfaz del procesador

- Conjunto de instrucciones
 - transferencia:
 - `in, mov, out, ...`
 - modificación:
 - `add, and, div, mul, or, sub, ...`
 - control:
 - `cli, sti, ...`
- Chuleta instrucciones 8086
- Registros generales y especiales
- Al menos dos modos de ejecución con diferentes privilegios:
 - privilegiado: acceso completo
 - no privilegiado: acceso limitado \Rightarrow excepción



Registros del procesador: familia x86-64

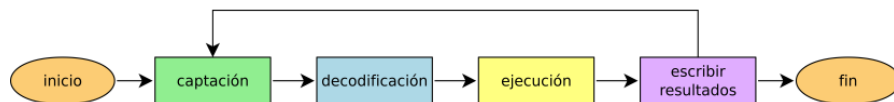
En el modo núcleo (modo privilegiado, modo kernel, ...) se pueden ejecutar instrucciones privilegiadas y se puede acceder a áreas de memoria protegidas. Sólo mente el núcleo o kernel del sistema operativo puede ejecutar instrucciones en modo privilegiado. Algunos ejemplos de instrucciones privilegiadas son:

- Acceso a los dispositivos de E/S: consultar el estado de los dispositivos de E/S, llevar a cabo DMA (Direct Memory Access), atrapar interrupciones.
- Manipular la unidad de gestión de memoria (MMU): manipular las tablas de segmento y páginas, cargar y vaciar el búfer de traducción anticipada (TLB)
- Configurar varios modos de funcionamiento: nivel de prioridad de interrupciones, alterar el vector de interrupción.
- Utilizar la instrucción *halt* para activar el modo de ahorro de energía. Esta instrucción detiene abruptamente la CPU hasta que la siguiente interrupción externa ha sido tratada. También es común que se ejecute cuando no hay trabajo inmediato que ejecutar, de forma que se pone al procesador en un estado ocioso.

El procesador comprueba el nivel de privilegio en la ejecución de cada instrucción. Los posibles cambios de privilegio son:

- Usuario \Rightarrow Núcleo: ganar privilegios
 - Al arrancar.
 - Llamada al sistema.
 - Interrupción hardware.
 - Excepción.
- Núcleo \Rightarrow Usuario: perder privilegios
 - El sistema operativo prepara el entorno necesario para que la aplicación comience su ejecución.
 - El sistema operativo termina alguna de sus actividades y devuelve el control a la aplicación.

Ciclo de instrucción: se denomina ciclo de instrucción al procesamiento requerido por una única instrucción. El ciclo de instrucción básico es:



- El procesador capta una instrucción desde memoria.
- La instrucción debe ser decodificada para averiguar su tipo.
- Conocido el tipo puede ser necesario la captación de nuevos operandos.
- Se ejecuta la instrucción.

- Se almacenan los resultados de la ejecución.
- El proceso se repite instrucción a instrucción hasta que el programa termina.

Tendencias en el diseño de procesadores

- CISC (Complex Instruction Set Computing) \Rightarrow RISC (Reduced Instruction Set Computing) \Rightarrow VLIW (Very Long Instruction Word).

CISC: Es un tipo de diseño de microprocesador. Este contiene un conjunto de instrucciones muy grande que van desde instrucciones muy simples a instrucciones muy especializadas. Se introducen instrucciones que en un diseño RISC se requieren de varias instrucciones, reduciendo así el número de instrucciones de los programas, pero al ser instrucciones más complejas acaban requiriendo más ciclos de reloj para su ejecución.

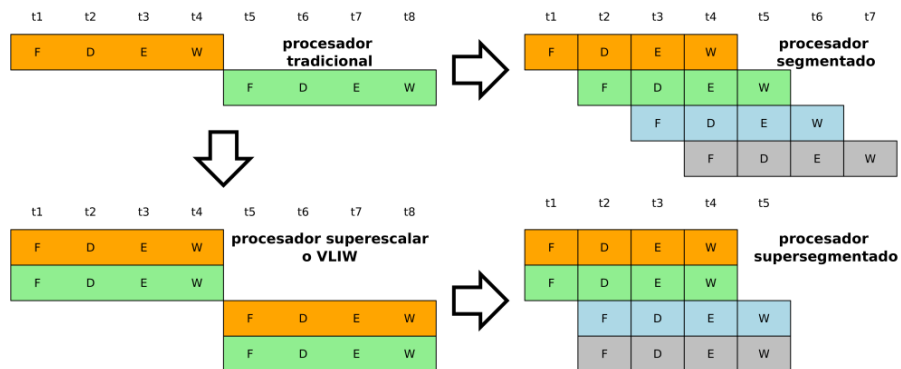
RISC: Como su nombre dice se trata de un diseño de microprocesador. Este diseño contiene instrucciones muy simples y con la combinación de ellas se puede obtener cualquier programa. Si bien su ejecución es rápida, pues son instrucciones muy simples, su tamaño en memoria puede llegar a ser muy grande, pues la ejecución de un programa simple puede requerir de muchas instrucciones RISC.

VLIW: se trata de un procesador segmentado que puede terminar más de una operación por ciclo en el que el compilador es el principal responsable de agrupar operaciones que pueden procesarse en paralelo para definir instrucciones que, de esta forma, se codifican a través de las denominadas palabras de instrucción larga (LIW) o muy larga (VLIW)

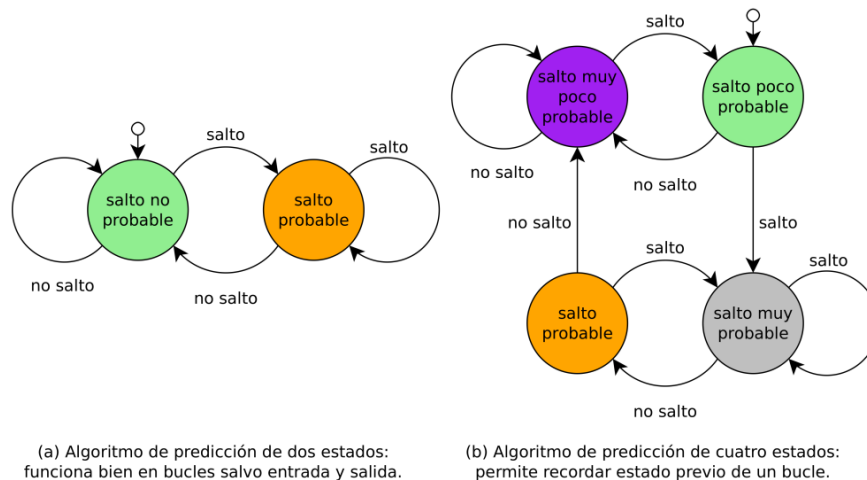
- Ejecución concurrente sobre procesadores: intento de explotación del paralelismo entre instrucciones (ILP). ILP es una medida de cuántas operaciones pueden ejecutarse simultáneamente sin afectar al resultado.

Técnicas de explotación de ILP

- **Segmentación de cauce:** la ejecución de múltiples instrucciones puede solaparse total o parcialmente.
- **Ejecución superescalar:** múltiples unidades de ejecución se utilizan para ejecutar múltiples instrucciones en paralelo. Un procesador superescalar es un procesador segmentado que puede finalizar más de una instrucción por ciclo y que posee recursos hardware para extraer el paralelismo entre instrucciones. Para aprovechar al máximo el procesamiento de instrucciones en paralelo que proporcionan las distintas etapas, el procesador incluye una serie de elementos como ventanas de instrucciones o estaciones de buffers, buffers de renombramiento, buffers de reordenamiento, etc.



- **Computación con instrucciones explícitamente paralelas (EPIC):** uso del compilador en lugar de complejos circuitos para identificar y explotar el ILP. La intención era permitir un escalado simple del rendimiento sin disparar las frecuencias del reloj. Tiene su base en VLIW.
- **Ejecución fuera de orden:** ejecución de instrucciones en cualquier orden que no viole las dependencias entre instrucciones. El orden de ejecución depende de la disponibilidad de los datos de entrada y las unidades de ejecución, no del orden original del programa.
- **Renombrado de registros:** técnica para evitar la innecesaria serialización de instrucciones por la reutilización de registros. Puede ser aplicado por el propio compilador al asignar los registros de la arquitectura, pero también puede implementarse en hardware. Esto es lo usual en procesadores superescalares, donde se incluyen estructuras de buffers con una serie de campos específicos (por ejemplo, buffers de renombramiento).
- **Ejecución especulativa:** permitir la ejecución de instrucciones completas, o partes, antes de conocer con seguridad si su ejecución debe tener lugar. De esta forma se previene el retraso que habría, de tener que hacer el procesamiento después de saber que es necesario. Si resulta que el procesamiento no era necesario, los cambios realizados se revierten y los resultados se ignoran.
- **Predicción de salto:** se utiliza para evitar quedar parado antes de que se resuelvan las dependencias de control. Se utiliza en conjunción con la ejecución especulativa. Se basa en determinar la alternativa más probable, y continuar el procesamiento, tras la instrucción de salto, con la secuencia de instrucciones que corresponde a dicha opción más probable. Cuando la condición de salto se evalúa, se comprueba si la predicción que se había hecho era correcta o no. Si no lo era habrá que retomar el procesamiento a partir de la primera instrucción de la alternativa que no se tomó, es decir, de la menos probable.



- **Multihebra simultánea (SMT):** técnica que permite la ejecución de múltiples hebras de ejecución para aprovechar mejor las unidades funcionales de los procesadores superescalares.

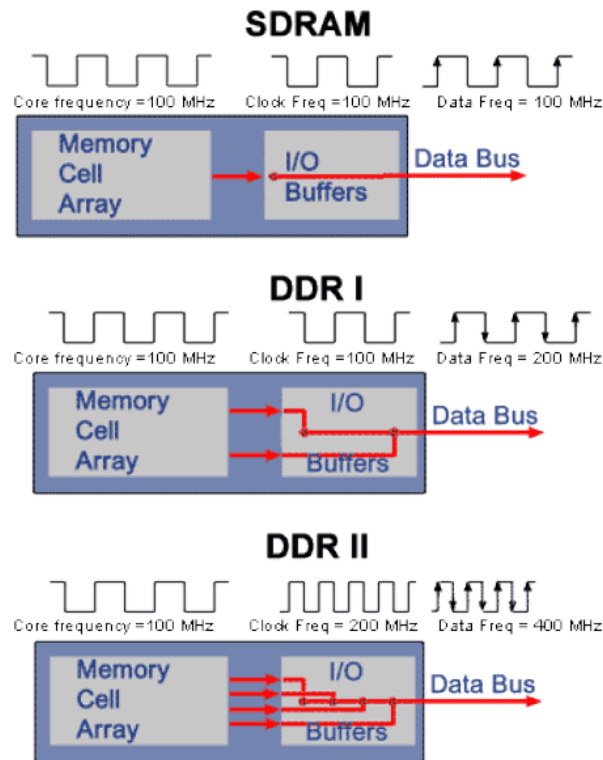
1.2.5 Memoria

Jerarquía de memoria: Se requiere de jerarquía de memoria entre otras cosas por la gran diferencia de velocidad entre procesador y memoria. La jerarquía de memoria puede aliviar este problema gracias a **los principios de localidad** y a la **regla 90/10**:

- Localidad **espacial**: la información a la que se accede suelen estar próxima a la que ha sido accedida con anterioridad.
- Localidad **temporal**: la información a la que se accede una vez suele volver a ser utilizada.
- Regla 90/10: el 10% del código realiza el 90% del trabajo.

Tipos de memoria RAM:

- **SRAM (Static random-access memory):** Es un tipo de memoria que utiliza circuitería flip-flop para almacenar cada bit. La diferencia con una DRAM es que la segunda debe de refrescarse periódicamente. La SRAM es más rápida y más cara que la DRAM. Se utiliza típicamente para la caché y los registros internos de la CPU mientras que la DRAM se utiliza para la memoria principal.
- **DDR SDRAM I (Double Data Rate Synchronous RAM):** Entre otras cosas las memorias DDR como su nombre indica tienen dos ciclos de reloj, es decir se hace un envío de información cuando el reloj sube y otro cuando



el reloj baja. Con esto hace que una memoria DDR con una frecuencia de reloj x duplique el ancho de banda de una SDR SDRAM con igual frecuencia de reloj. Este tipo de memoria se ha visto superada por las versiones 2, 3, 4 y 5 de la misma. Ninguno de sus sucesores tienen compatibilidad ni con su predecesor ni con su sucesor, lo que quiere decir que una RAM DDR1 SDRAM no es compatible con DDR2, DDR3,

- DDRAM II

Cantidad de memoria en registros

- Tipos de registros:
- RISC:
 - 32 de propósito general (32 ó 64 bits)
 - 32 de punto flotante (64 bits IEEE 754)
 - multimedia (64,...,256 bits)
- CISC:
 - IA32: 8 de propósito general, 8 de punto flotante, 8 multimedia.

- IA64: 128 de propósito general, 128 de punto flotante.
- Algunos procesadores tienen varios conjuntos de estos registros (ventanas de registros)

Análisis de una jerarquía de dos niveles

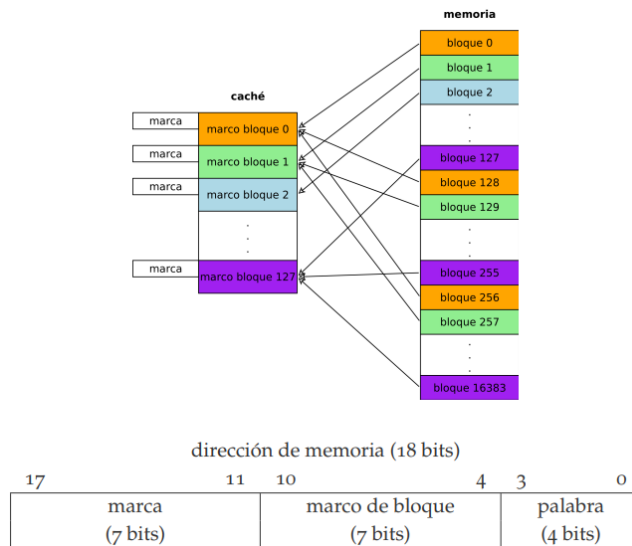
$$\bar{T}_{acceso} = (1 - T_{fallos}) \times T_{cache} + T_{fallos} \times (T_{cache} + T_{ram})$$

Parámetros de diseño de memorias caché

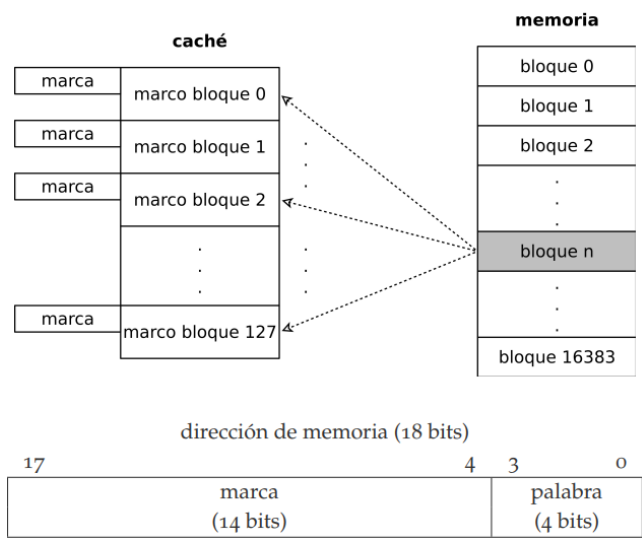
- Tamaño:
 - L1:8,...,256KB
 - L2:1,...,16MB
 - L3:4,...,128MB
- Tamaño de bloque: 32,...,128B
- Tiempo de acceso: 1,...,10ns
- Política de búsqueda:
 - bajo demanda
 - anticipativas
- Política de colocación:
 - correspondencia directa: el bloque B_j de memoria principal se puede ubicar sólo en el marco de bloque que cumple la siguiente relación $i = j \bmod m$, donde m es el número total de líneas que tiene la caché.
 - asociativa por conjuntos: En la correspondencia asociativa por conjuntos las líneas de memoria caché se agrupan en $v = 2^d$ conjunto con k líneas/conjunto o vías. Se cumple que el número total de marcos de bloque que tiene la caché $m = v * k$. Un bloque B_j de memoria principal se puede ubicar sólo en el conjunto C_i de memoria caché que cumple la siguiente relación $i = j \bmod v$.
 - completamente asociativa: la caché se organiza en un único conjunto de caché con varias líneas de caché. Un bloque de memoria puede ocupar cualquiera de las líneas de caché. La organización de la caché se puede enmarcar como una matriz de filas ($1 * m$).
- Política de reemplazo:
 - LRU
 - FIFO
 - aleatoria

- Política de actualización:
 - escritura directa
 - post-escritura
- Otras características importantes:
 - caché de víctimas
 - inclusiva/exclusiva
 - unificada/separada

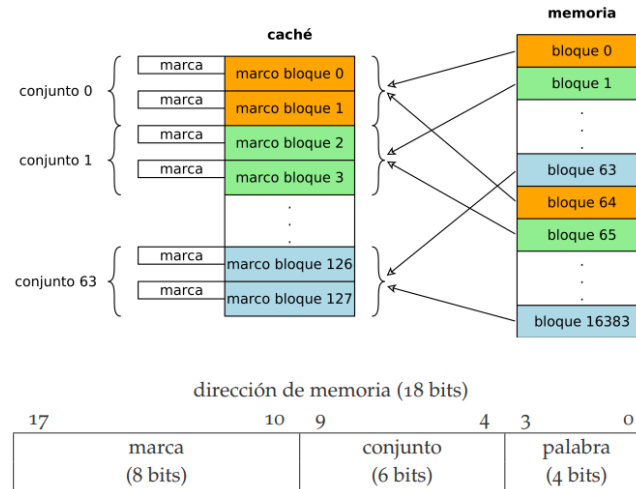
Políticas de colocación: correspondencia directa



Políticas de colocación: correspondencia totalmente asociativa



Políticas de colocación: correspondencia asociativa por conjuntos
(2 M/C)



Protección de memoria.

El sistema operativo debe proteger a los programas entre sí y a él mismo de programas maliciosos o erróneos. Este problema se soluciona utilizando los llamados espacios de direcciones (AS, Address Space). El espacio de direcciones de un computador corresponde al rango de direcciones disponible para un programa de computador.

El procesador dispone de recursos para establecer límites al espacio de direcciones de memoria accesibles por los programas: memoria virtual.

Tabla 8.1. Características de la paginación y la segmentación.

Paginación sencilla	Paginación con memoria virtual	Segmentación sencilla	Segmentación con memoria virtual
Memoria principal particionada en fragmentos pequeños de un tamaño fijo llamados marcos	Memoria principal particionada en fragmentos pequeños de un tamaño fijo llamados marcos	Memoria principal no particionada	Memoria principal no particionada
Programa dividido en páginas por el compilador o el sistema de gestión de la memoria	Programa dividido en páginas por el compilador o el sistema de gestión de la memoria	Los segmentos de programa se especifican por el programador al compilador (por ejemplo, la decisión se toma por parte el programador)	Los segmentos de programa se especifican por el programador al compilador (por ejemplo, la decisión se toma por parte el programador)
Fragmentación interna dentro de los marcos	Fragmentación interna dentro de los marcos	Sin fragmentación interna	Sin fragmentación interna
Sin fragmentación externa	Sin fragmentación externa	Fragmentación externa	Fragmentación externa
El sistema operativo debe mantener una tabla de páginas por cada proceso mostrando en el marco que se encuentra cada página ocupada	El sistema operativo debe mantener una lista de páginas por cada proceso mostrando en el marco que se encuentra cada página ocupada	El sistema operativo debe mantener una tabla de segmentos por cada proceso mostrando la dirección de carga y la longitud de cada segmento	El sistema operativo debe mantener una lista de huecos en la memoria principal
El sistema operativo debe mantener una lista de marcos libres	El sistema operativo debe mantener una lista de marcos libres	El sistema operativo debe mantener una lista de huecos en la memoria principal	El sistema operativo debe mantener una lista de huecos en la memoria principal
El procesador utiliza el número de página, desplazamiento para calcular direcciones absolutas	El procesador utiliza el número de página, desplazamiento para calcular direcciones absolutas	El procesador utiliza el número de segmento, desplazamiento para calcular direcciones absolutas	El procesador utiliza el número de segmento, desplazamiento para calcular direcciones absolutas
Todas las páginas del proceso deben encontrarse en la memoria principal para que el proceso se pueda ejecutar, salvo que se utilicen solapamientos (<i>overlays</i>)	No se necesita mantener todas las páginas del proceso en los marcos de la memoria principal para que el proceso se pueda ejecutar. Las páginas se pueden leer bajo demanda	Todos los segmentos del proceso deben encontrarse en la memoria principal para que el proceso se pueda ejecutar, salvo que se utilicen solapamientos (<i>overlays</i>)	No se necesitan mantener todos los segmentos del proceso en la memoria principal para que el proceso se ejecute. Los segmentos se pueden leer bajo demanda
	La lectura de una página a memoria principal puede requerir la escritura de una página a disco		La lectura de un segmento a memoria principal puede requerir la escritura de uno o más segmentos a disco

Hay varias implementaciones:

- Segmentaciones: tamaño variable y arbitrario.
 - registro base: principio del espacio de direcciones.
 - registro límite: tamaño del espacio de direcciones.
- Paginación: tamaño variable en múltiplos de página.
 - tablas de páginas: lista de páginas de un proceso.

La mayoría de los SO asocian un espacio de direcciones diferente para cada proceso (salvo SASOS, Single Address Space Operating System):

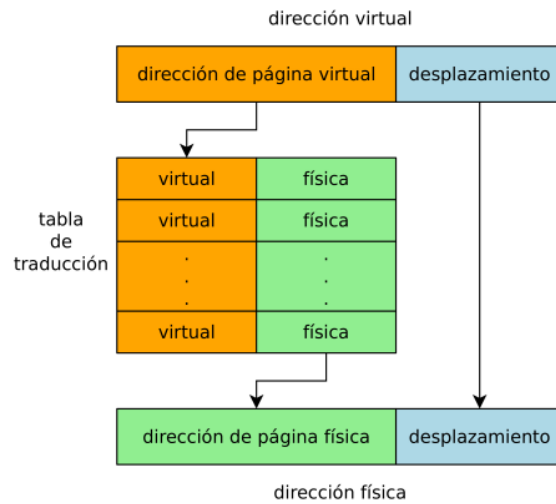
- Ventaja \Rightarrow protección automática.
- Inconveniente \Rightarrow dificulta la compartición

Las partes de un espacio de direcciones pueden colocarse en cualquier parte de la memoria física. Algunos espacios de direcciones deben ocupar lugares específicos de la memoria física, ej: dispositivos de E/S.

El procesador debe tener una unidad de gestión de memoria (MMU, Memory Management Unit) extremadamente eficiente porque la traducción entre direcciones virtual-física puede realizarse más de una vez por instrucción. Se requiere una traducción por cada acceso a memoria.

Traducción de dirección virtual a dirección física

1. La ejecución de una instrucción puede requerir varias traducciones, por ejemplo:
 - Captación de la instrucción.
 - Captación del dato.
 - Escritura del resultado.
2. Cada traducción puede requerir varios accesos a memoria:
 - Segmentación + paginación.
 - Tablas de página multinivel.
3. Se necesita una tabla de traducción por cada espacio de direcciones.
4. Dependerá del tamaño de las direcciones virtuales y físicas, pero habrán al menos tantas entradas como páginas tenga el proceso.



Fucnionamiento de la segmentación (x86)

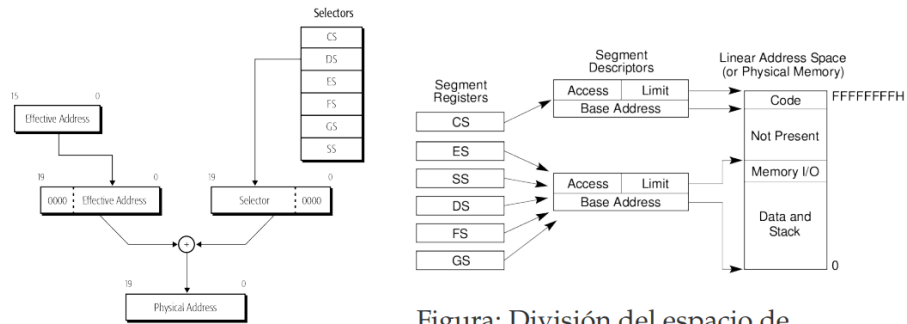


Figura: Cálculo de la dirección física.

Figura: División del espacio de direcciones de un proceso.

Segmentación + paginación (x86_64)

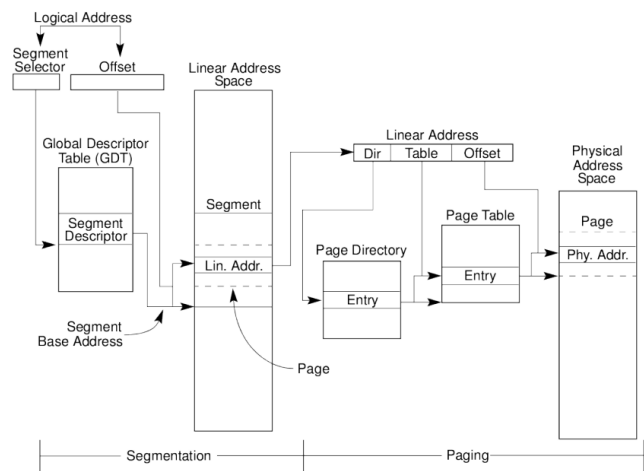
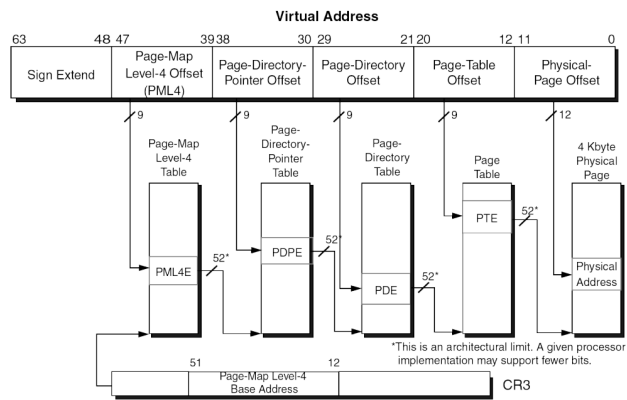


Tabla de páginas multinivel (x86_64)



Estructura habitual tabla de páginas

Dirección virtual

Número de página	Desplazamiento
------------------	----------------

Entrada de la tabla de páginas

P	M	Otros bits de control	Número de marco
---	---	-----------------------	-----------------

(a) Únicamente paginación

Dirección virtual

Segmento de página	Desplazamiento
--------------------	----------------

Entrada de la tabla de segmentos

P	M	Otros bits de control	Longitud	Comienzo de segmento
---	---	-----------------------	----------	----------------------

(b) Únicamente segmentación

Dirección virtual

Segmento de página	Número de página	Desplazamiento
--------------------	------------------	----------------

Entrada de la tabla de segmentos

Bits de control	Longitud	Comienzo de segmento
-----------------	----------	----------------------

Entrada de la tabla de páginas

P	M	Otros bits de control	Número de marco
---	---	-----------------------	-----------------

P = bit de presente
M = bit de modificado

(c) Combinación de segmentación y paginación

Figura 8.2. Formatos típicos de gestión de memoria.

1.2.6 Interacción

Buffer de traducción anticipada (TLB)

Traducir direcciones por software es muy lento, mientras que mediante hardware puede hacerse más rápido y además podemos añadir una caché de traducciones (TLB). En principio, toda referencia a la memoria virtual puede causar dos accesos a memoria física, uno para buscar la entrada en la tabla de página apropiada y otra para buscar los datos solicitados. De esa forma, el esquema de memoria virtual básico causaría el efecto de duplicar el tiempo de acceso a la memoria. Para solventar este problema, la mayoría de esquemas de la memoria virtual utilizan una *cache* especial de alta velocidad para las entradas de la tabla de páginas, habitualmente denominada buffer de traducción anticipada. TLB:

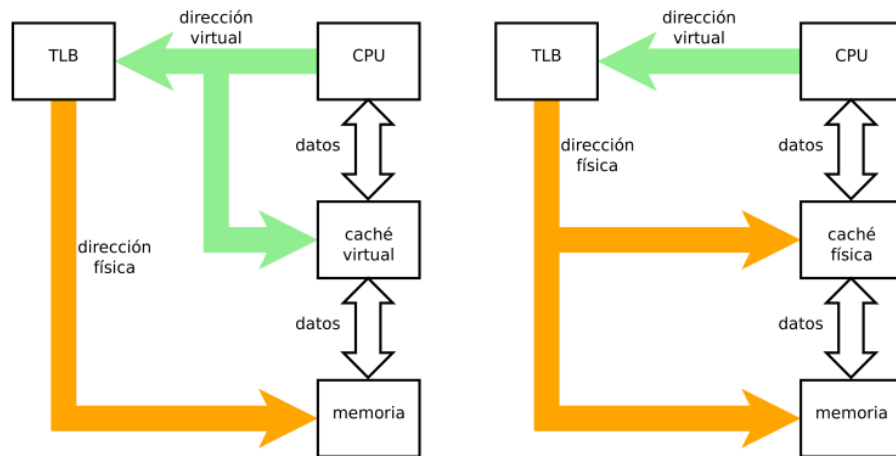
- Contenido: pares de direcciones virtual/física.
- Implementado mediante memorias asociativas.
- Tamaño típico: de 32 a 128 entradas.

Hay dos tipos fundamentales:

- Etiquetadas: añaden una marca del espacio de direcciones al que pertenece.
- No etiquetadas: no poseen la capacidad anterior, la mayoría.

Funcionamiento:

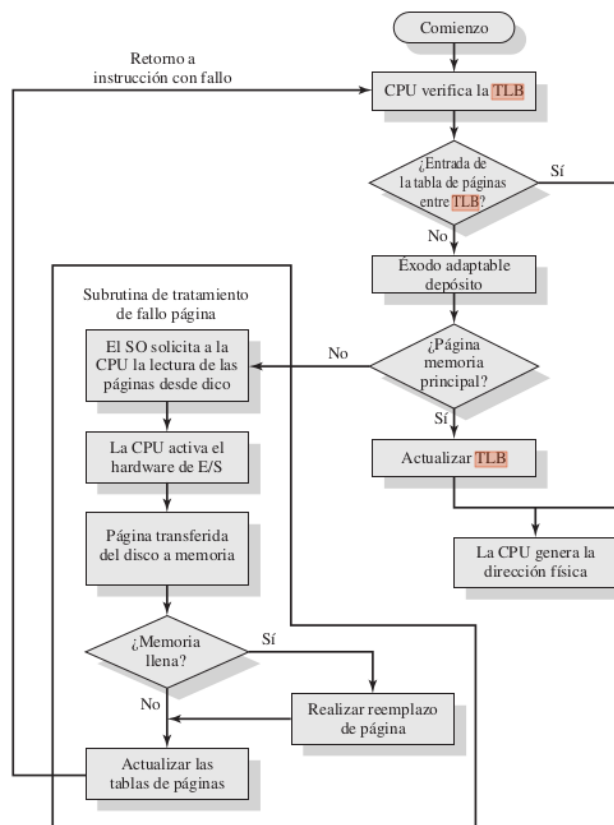
- Si se encuentra la dirección en el TLB se devuelve su traducción.
- Si no se encuentra es necesario calcular la traducción.
 - Hardware: se devuelve la traducción correcta (x86).
 - Software: más lento pero más flexible (PowerPC).
- Es fundamental conocer la interacción TLB/caché.



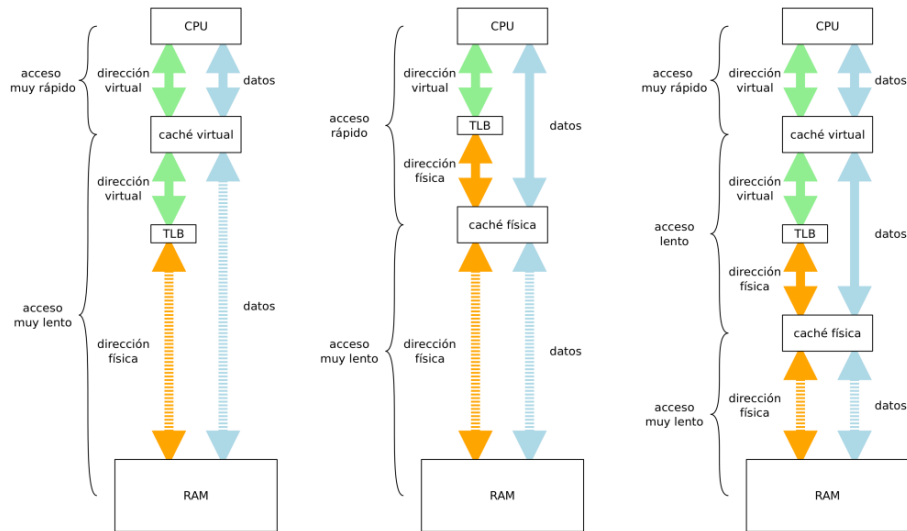
El proceso que sigue generalmente es el siguiente:

1. Dada una dirección virtual, el procesador primero examina la TLB.
2. Si la entrada de la tabla de páginas solicitada está presente (acierto en la TLB), entonces se recupera el número de marco y se construye la dirección real.
3. Si la entrada de la tabla de páginas no se encuentra (fallo en la TLB), el procesador utiliza el número de página para indexar la tabla de páginas del proceso y examinar la correspondiente entrada de la tabla de páginas.

- (a) Si el bit de presente está puesto a 1, entonces la página se encuentra en memoria principal, y el procesador puede recuperar el número de marco desde la entrada de la tabla de páginas para construir la dirección real. El procesador también autorizará la TLB para incluir esta nueva entrada de tabla de páginas.
- (b) Si el bit presente no está puesto a 1, entonces la página solicitada no se encuentra en memoria principal y se produce un fallo de acceso memoria, llamado **fallo de página**.



Esquemas de direccionamiento de caché



Control de Entrada/Salida

El sistema operativo inicia una operación E/S:

- Instrucciones especiales: **in/out** (x86)
- E/S en memoria: **mov** (x86)
 - Acceso al hardware como direcciones de memoria.
 - Requiere de una MMU capaz de traducir el bus de E/S.

Por otro lado es SO puede averiguar cuando una orden E/S finaliza de las siguientes formas:

- Sondeo: leer el puerto de estado hasta encontrar el valor adecuado. La CPU continuamente rueba todos y cada uno de los dispositivos conectados a él para detectar si algún dispositivo necesita atención de la CPU. Algoritmo de sondeo:

1. Cuando un dispositivo tiene algún comando para ser ejecutado por la CPU, comprueba continuamente el bit de ocupado de la CPU hasta que se borra (0).
2. Cuando se borra el bit de ocupado, el dispositivo establece el bit de escritura en su registro de comando y escribe un byte en el registro de salida de datos.
3. Ahora el dispositivo establece (1) el bit de comando listo.
4. Cuando la CPU verifica el bit de comando listo y lo encuentra establecido (1), establece (1) su bit de ocupado.

5. Luego, la CPU lee el registro de comando del dispositivo y ejecuta el comando del dispositivo.
 6. Después de la ejecución del comando, la CPU borra (0) el bit de comando listo, el bit de error del dispositivo para indicar la ejecución exitosa del comando del dispositivo y además borra (0) su bit de ocupado para indicar que la CPU está libre para ejecutar el comando de algún otro dispositivo.
- Interrupción: existe una línea física mediante la cual se avisa del fin de la operación y se cede el control al SO. Procesamiento de interrupciones:
 1. El dispositivo genera una señal de interrupción hacia el procesador.
 2. El procesador termina la ejecución de la instrucción actual antes de responder a la interrupción.
 3. El procesador comprueba si hay petición de interrupción pendiente, determina que hay una y manda una señal de reconocimiento al dispositivo que produjo la interrupción. Este reconocimiento permite que el dispositivo elimine su señal de interrupción.
 4. En ese momento, el procesador necesita prepararse para transferir el control a la rutina de interrupción. Para comenzar, necesita salvar la información requerida para reanudar el programa actual en el momento de la interrupción. La información mínima requerida es la palabra de estado del programa (PSW) y la posición de la siguiente instrucción que se va a ejecutar, que está contenida en el contador de programa. Esta información se puede apilar en la pila de control de sistema.
 5. A continuación, el procesador carga del contador del programa con la posición del punto de entrada de la rutina de interrupción que responderá a esta interrupción. Dependiendo de la arquitectura de computador y del diseño del sistema operativo, puede haber un único programa, uno por cada tipo de interrupción o uno por cada dispositivo y tipo de interrupción. Si hay más de una rutina de manejo de interrupción, el procesador debe determinar cuál invocar. Esta información puede estar incluida en la señal de interrupción original o el procesador puede tener que realizar una petición al dispositivo que generó la interrupción para obtener una respuesta que contiene la información requerida.
 6. En este momento, el contador del programa y la PSW vinculados con el programa interrumpido se han almacenado en la pila del sistema. Sin embargo, hay otra información que se considera parte del estado del programa en ejecución. En concreto, se necesita salvar el contenido de los registros del procesador, puesto que estos registros los podría utilizar el manejador de interrupciones. Por tanto, se deben salvar todos estos valores, así como cualquier otra información de

estado. Generalmente, el manejador de interrupción comenzará salvando el contenido de todos los registros en pila.

7. El manejador de interrupción puede en este momento comenzar a procesar la interrupción. Esto incluirá un examen de la información de estado relacionada con la operación de E/S o con otro evento distinto que haya causado la interrupción. Asimismo, puede implicar el envío de mandatos adicionales o reconocimientos al dispositivo de E/S.
8. Cuando se completa el procesamiento de la interrupción, se recuperan los valores de los registros salvados en la pila y se restituyen los registros.
9. La última acción consiste en restituir de la pila los valores de la PSW y del contador del programa. Como resultado, la siguiente instrucción que se va a ejecutar corresponderá al programa previamente interrumpido.

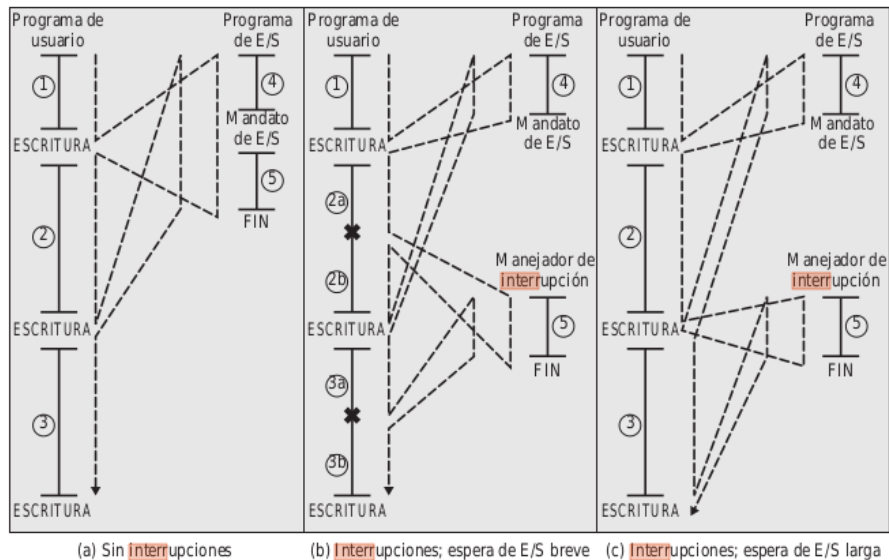


Figura 1.5. Flujo de programa del control sin interrupciones y con ellas.

Eventos: excepciones e interrupciones

Eventos o "situaciones especiales":

- Excepciones (síncronas): llamadas al sistema.
- Interrupciones (asíncronas): fin de operación de DMA.

Hay otras diferencias como el origen, la predictibilidad o la reproducibilidad. El manejo de excepciones e interrupciones debe producirse a tiempo y es específico para cada tipo de procesador.

<p>Excepciones síncronas e internas al procesador:</p> <ul style="list-style-type: none"> ⊙ origen: la instrucción actual. ⊙ excepciones erróneas: <ul style="list-style-type: none"> ◦ puntero inválido. ◦ división entre 0. ⊙ normalmente el programa es abortado. ⊙ excepciones no erróneas: <ul style="list-style-type: none"> ◦ fallo de página. ◦ punto de ruptura. ⊙ SO gestiona de forma transparente al usuario. 	<p>Interrupciones asíncronas y externas al procesador:</p> <ul style="list-style-type: none"> ⊙ origen: dispositivos de E/S, temporizador,... ⊙ no están relacionadas con el flujo de instrucciones. ⊙ la mayoría de las interrupciones está causadas por la finalización de operaciones de E/S. ⊙ algunas interrupciones están causadas por fallos en dispositivos (atasco de papel).
---	---

Interrupciones software (int/syscall - swi)

Se tratan de un mecanismo para ceder el control al SO elevando el nivel de privilegio. Son síncronas, predecibles y reproducibles, luego aunque se llamen interrupciones son excepciones.

Un mismo programa aislado siempre provoca las mismas:

- Instrucción desconocida.
- Instrucción errónea (división entre 0).
- Modo de direccionamiento erróneo.
- Violación del espacio de direcciones.
- Llamada al sistema.
- Fallo de página (solo políticas de paginación locales).

Deben atenderse, sino se daría un comportamiento erróneo. Producen una sobrecarga en espacio y tiempo fuera del programa que la solicita.

Interrupciones

Se trata de un mecanismo para solicitar la atención de la CPU. Son asíncronas, no predecibles y no reproducibles. Un periférico puede solicitar una interrupción independientemente del estado de la CPU o el proceso actual.

Ejemplos:

- Eventos externos: sensores
- Final de una operación DMA.
- Final de una operación E/S.

Vector de interrupciones

Se trata del vector que contiene las direcciones de los manejadores de interrupción para los distintos dispositivos conectados y las distintas interrupciones.

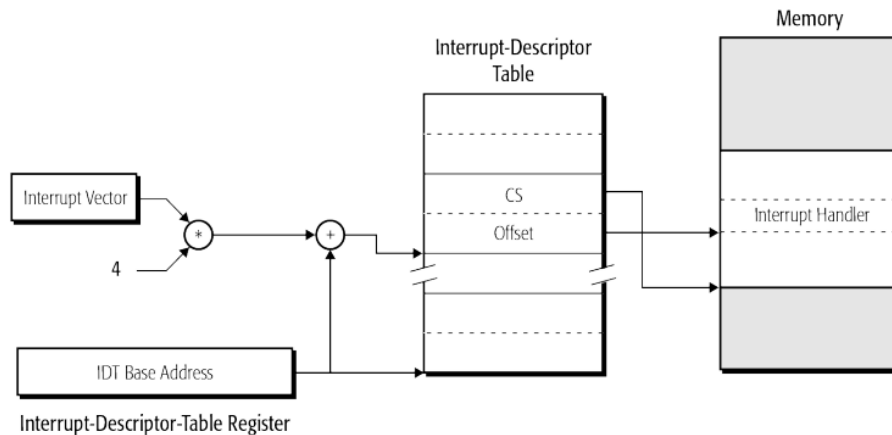


Figura: Vector de interrupciones (procesadores 80x86).

Manejo de excepciones e interrupciones

Los siguientes eventos podrían lanzar una excepción o una interrupción:

- Señales de periféricos:
 - Final de una lectura de disco
 - Atasco de papel
- Cambio del dominio de protección (segmentación/paginación).
- Errores de programación (acceso a dirección inválida).
- Desbordamiento de memoria (desbordamiento de pila).
- Paginación bajo demanda.
- Fallos hardware (paridad de memoria).

El manejo del evento se hace durante la ejecución del proceso actualmente en ejecución.

Efectos de la temporización

El manejo de cada evento ralentiza la ejecución del programa interrumpido. El resultado de los programas no se pone en peligro, salvo las aplicaciones de tiempo real que sí pueden fallar. Esta es una de las razones por las que los programadores no deben confiar en las condiciones de temporización (ej: sincronización).

Control de interrupciones:

Las interrupciones pueden llegar en cualquier momento:

- Usuario modificando datos compartidos.
- SO realizando una operación no interrumpible.

El sistema operativo y el hardware permiten sincronizar actividades concurrentes mediante operaciones atómicas: secuencias de instrucciones que no pueden ser interrumpidas.

Soluciones:

- Deshabilitar las interrupciones. En una máquina monoprocesador, los procesos concurrentes no pueden solaparse, sólo pueden entrelazarse. Es más, un proceso continuará ejecutando hasta que invoque un servicio del sistema operativo o hasta que sea interrumpido. Por tanto, para garantizar la exclusión mutua, basta con impedir que un proceso sea interrumpido. Esta técnica puede proporcionarse en forma de primitivas definidas por el núcleo del sistema para deshabilitar y habilitar las interrupciones. Esta solución no funcionará sobre una arquitectura multiprocesador. Cuando el sistema de cómputo incluye más de un procesador, es posible (y típico) que se estén ejecutando al tiempo más de un proceso. En este caso, deshabilitar interrupciones no garantiza exclusión mutua.
- Instrucciones atómicas basadas en leer/modificar/almacenar: En una configuración multiprocesador, varios procesadores comparten acceso a una memoria principal común. No hay mecanismo de interrupción entre procesadores en el que pueda basarse la exclusión mutua.

A un nivel hardware, el acceso a una posición de memoria excluye cualquier otro acceso a la misma posición. Los diseñadores de hardware han propuesto varias instrucciones que llevan a cabo dos acciones atómicamente, como leer y escribir o leer y comprobar, sobre una única posición de memoria con un único ciclo de búsqueda de instrucción. Durante la ejecución de la instrucción, el acceso a la posición de memoria se le bloquea a toda otra instrucción que referencia esa posición. Estas acciones se realizan (típicamente) en un solo ciclo de instrucción.

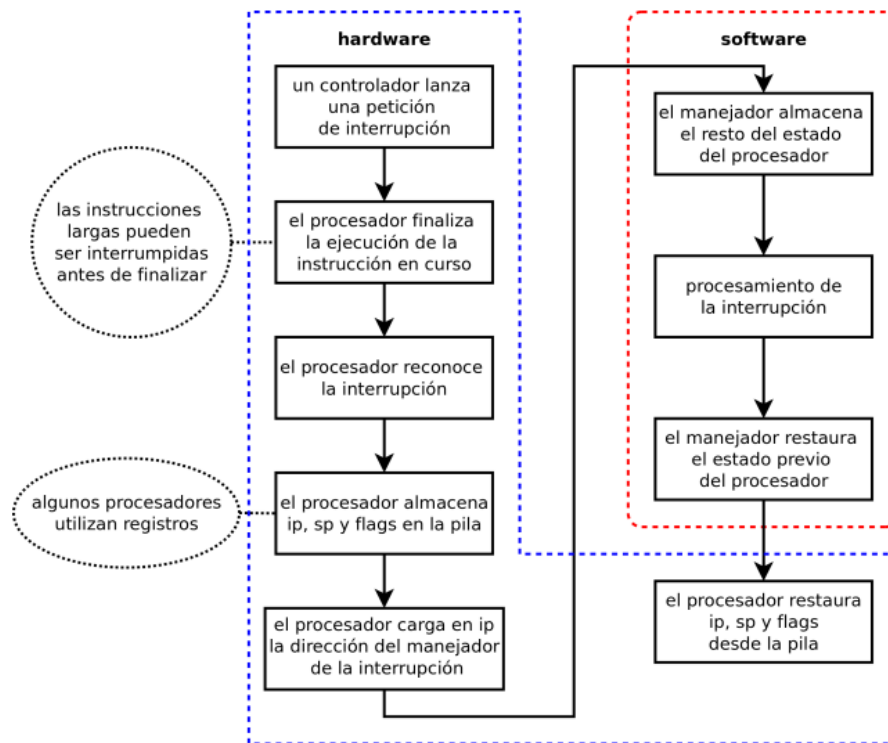
- **tas**: *test and set* (comprueba y establece)
- **cmpxchg**: comparar e intercambiar.
- **ll/sc**: carga enlazada y almacenamiento condicional.

La mayoría de los ordenadores permiten que los controladores de E/S interrumpan la actividad del procesador. La CPU transfiere el control a un manejador de interrupción que normalmente es parte del SO (excepción: controladores en espacio de usuario).

El procesador puede prevenir las interrupciones:

- Enmascarándolas
- Deshabilitándolas
- Estableciendo un nivel mínimo de prioridad.

Procesamiento de interrupciones



Ventajas del uso de interrupciones

- Los eventos son atendidos más rápidamente
- No se consume tiempo del procesador para descubrir el final de un evento.

- Los programas pueden ejecutarse más rápidamente.
- Se mejora el aprovechamiento del procesador.

Con sólo el uso de interrupciones el procesador sigue siendo el encargado de realizar las transferencias de información. La solución ideal es utilizar DMA para evitarlo.

Múltiples interrupciones

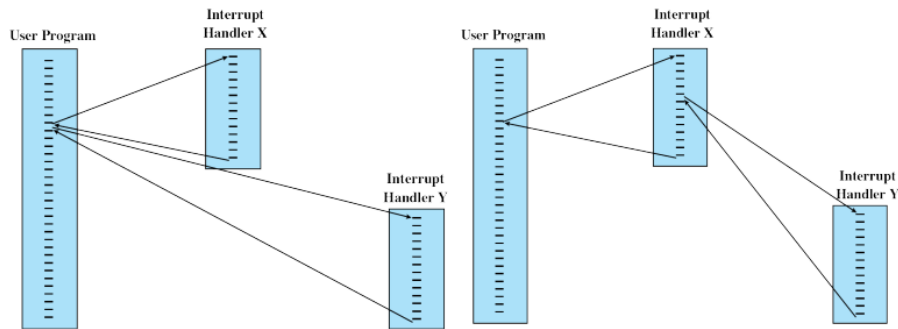


Figura: Procesamiento secuencial (interrupciones deshabilitadas).

Figura: Procesamiento anidado (interrupciones con prioridades).

Técnicas de Entrada/Salida

- **E/S programada (sondeo).** En E/S programada, el módulo de E/S realiza la acción solicitada y fija los bits correspondiente en el registro de estado de E/S, pero no realiza ninguna acción para avisar al procesador. En concreto, no interrumpe al procesador. Por tanto, después de que se invoca la instrucción de E/S, el procesador debe tomar un papel activo para determinar cuándo se completa la instrucción de E/S. Por eso el procesador comprueba periódicamente el estado del módulo de E/S hasta que se encuentre que se ha completado la operación.

Tiene la desventaja de que es un proceso que consume un tiempo apreciable que mantiene al procesador ocupado innecesariamente.

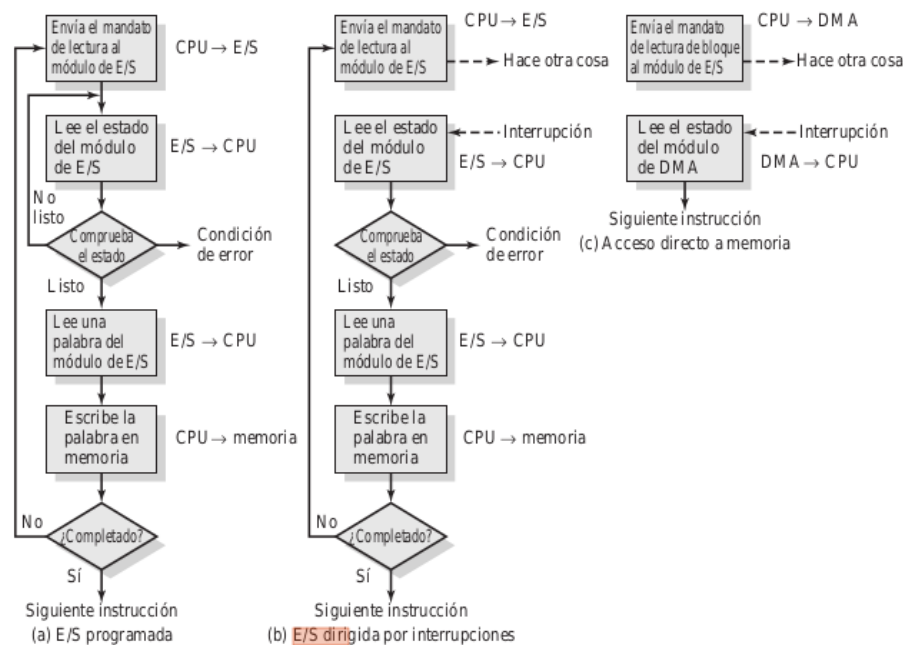
- **E/S dirigida mediante interrupción.** Es una alternativa a la E/S programada. En ella el procesador genera un mandato de E/S para un módulo y, acto seguido, continúa realizando algún otro trabajo útil. El módulo de E/S interrumpirá más tarde al procesador para solicitar su servicio cuando esté listo para intercambiar datos con el mismo. El procesador ejecutará la transferencia de datos, como antes, y después reanudará el procesamiento previo.

El mayor problema de esta alternativa es que el procesador tiene que hacer la transferencia de datos el mismo, cosa que se arreglará con DMA.

- **Acceso directo a memoria (DMA).** La función DMA puede ser llevada a cabo por un módulo separado conectado en el bus del sistema o puede estar incluida en un módulo de E/S. La técnica funciona como sigue, cuando el procesador desea leer o escribir un bloque de datos, genera un mandato de DMA, enviando la siguiente información:

- Si se trata de una lectura o una escritura.
- La dirección del dispositivo de E/S involucrado.
- La posición inicial de memoria en la que se desea leer los datos o donde se quieren escribir.
- El número de palabras que se pretende leer o escribir.

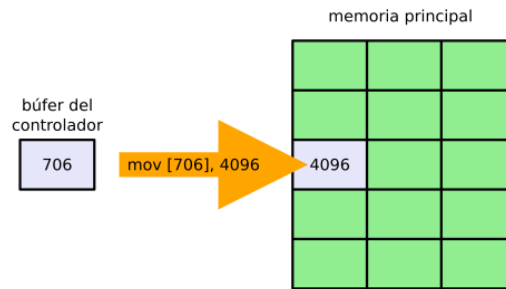
A continuación, el procesador continúa con otro trabajo. Ha delegado esta operación de E/S al módulo de DMA, que se ocupará de la misma. El módulo de DMA transferirá el bloque completo de datos, palabra a palabra, hacia la memoria o desde ella sin pasar a través del procesador. Por tanto, el procesador solo se involucra al principio y al final.



Direccionamiento físico de la memoria principal

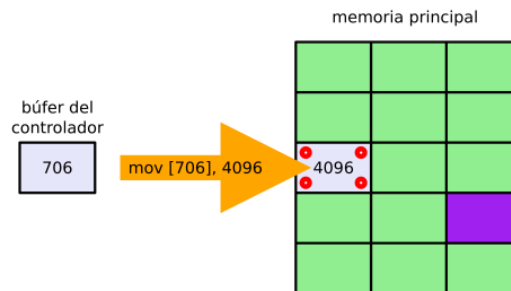
El direccionamiento virtual es algo que sólo existe en el interior del procesador. Para acceder a memoria los controladores de dispositivos y DMA sólo emplean direcciones físicas. Por ejemplo:

Un retraso en la E/S podría provocar que el marco 4096 fuese asignado a otro proceso a través de los mecanismos de segmentación o paginación. La solución



a este problema es fijar (**pinning**) el marco de página durante el tiempo que dure el proceso de transferencia. Fijar (**pinning**) y liberar (**unpinning**) es un mecanismo de protección.

- Usos: soporte de DMA y procesos de tiempo real.
- Problema: el abuso para acaparar recursos.



Temporizador

Los niveles de privilegio son un mecanismo de protección útil para el SO. Los eventos pueden transferir el control al sistema operativo. Si un proceso no genera eventos nunca cede el control del procesador. A este problema hay varias soluciones:

- La más habitual: generar una interrupción de forma periódica mediante el temporizador, ej: cada 10ms.
- La más conveniente: reprogramar el temporizador tras cada evento para que sólo genere interrupciones cuando sea necesario.

2 Tema 2: Introducción a los sistemas operativos

2.1 Abstracciones

Se denomina espacio de direcciones a la memoria utilizable por un proceso. Se trata además de una unidad de protección de memoria, pues el hardware de direccionamiento en memoria asegura que un proceso se pueda ejecutar únicamente dentro de su propio espacio de direcciones.

Habitualmente este está formado por 3 componentes:

- **Código:** .text
- **Datos:** .data, .bss y heap
- **Pila:** Stack
 - Variables locales
 - Marcos de procedimiento

El tamaño de cada zona es:

- Código: tamaño fijo
- Datos: pueden crecer hacia arriba
- Pila: pueden crecer hacia abajo

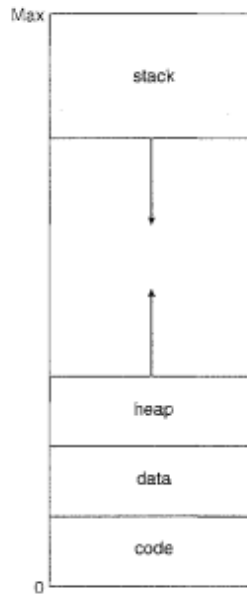


Figure 9.2 Virtual address space.

Procesos y/o hebras:

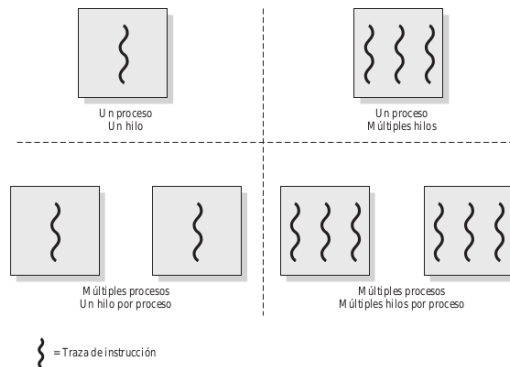
Se trata de una unidad de trabajo. Incluye el contexto del procesador (que contiene el contador del programa y el puntero de pila) y su propia área de datos para una pila (para posibilitar el salto a subrutinas). Un hilo se ejecuta secuencialmente y se puede interrumpir de forma que el procesador pueda dar paso a otro hilo. A parte de la distribución dada hay otras distintas como pueden ser

- (Porción de) un programa en ejecución.
- Instancia de un programa ejecutándose.
- Hebra/hilo/fibra: mínima unidad de ejecución
- Proceso/tarea: unidad de posesión/protección de recursos.

Comunicación entre procesos

Hay varios 3 tipos de comunicación entre procesos:

- Ficheros: tuberías, sockets.
 - Tubería: es un *buffer* circular que permite que dos procsos se comuniquen siguiendo el modelo productor consumidor. Por tanto, se



trata de una cola de tipo FIFO, en la que se escribe un proceso y se lee otro.

- Socket: permite la comunicación entre un proceso cliente y un proceso servidor y puede ser orientado a conexión o no orientado a conexión. Un socket se puede considerar como un punto final en una comunicación. Un socket cliente en un computador utiliza una dirección para llamar a un socket de servidor en otro computador. Una vez que han entrado en comunicación los sockets, los dos computadores pueden intercambiar datos.

- Paso de mensajes. Los procesos se comunican por medio de mensajes, que son un conjunto de bytes con un tipo asociado. Asociada a cada proceso existe una cola de mensajes, que funciona como buzón.
- Memoria compartida. Es la forma más rápida de comunicación entre procesos. Se trata de un bloque de memoria virtual compartido por múltiples procesos. Los procesos leen y escriben en la memoria compartida utilizan las mismas instrucciones de máquina que se utilizan para leer y escribir en otras partes de su espacio de memoria virtual.

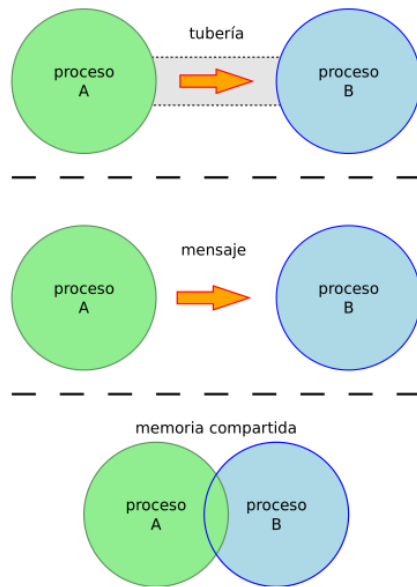
En algunos libros no se considera la tubería como un tipo de comunicación en sí mismo, ya que quería dentro de alguno de los otros dos.

Concurrencia y paralelismo:

Varios procesos pueden ejecutarse,

- en paralelo en un multiprocesador.
- de forma concurrente en un uniprocador.

No se debe confundir concurrencia con paralelismo, pues aunque la concurrencia da una impresión de paralelismo, realmente lo único que hace es un empleo muy eficiente del tiempo de ejecución del procesador, dando lugar a la sensación de ejecución de varios procesos simultáneamente, pero realmente en ningún momento el procesador tiene en ejecución dos procesos distintos en el mismo instante.



Los procesos multihebra pueden ser ejecutados de forma paralela o concurrente en función de

- el número de procesadores
- el modelo de hebras: usuario o núcleo.

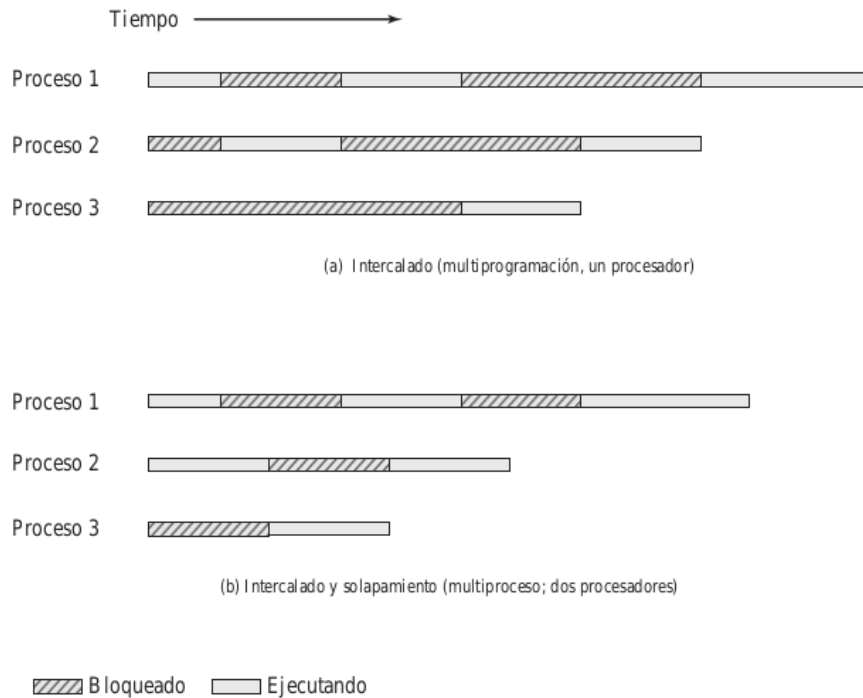
Podrán aparecer "condiciones de carrera" si no manejamos cuidadosamente la concurrencia, lo que implica la necesidad de métodos de sincronización de procesos y hebras.

Condiciones de carrera: sucede cuando múltiples procesos o hilos leen y escriben datos de forma que el resultado final depende del orden de ejecución de las instrucciones en los múltiples procesos.

Gestión de memoria.

La memoria RAM es limitada. De aquí surge el problema de que los procesos no saben qué posición ocuparán en la RAM. La posición de los procesos en RAM es responsabilidad del compilador y del sistema operativo. La solución a este problema es la utilización de código relocizable. Este código tiene la particularidad de que en vez de trabajar con direcciones físicas trabaja con direcciones relativas al código, es decir, en lugar de hacer un salto a la dirección 0xffff, se hace un salto a la dirección en la que está la instrucción actual-5. De esta forma ya no es necesario saber exactamente donde está cada cosa, sino que se buscará donde está la ejecución instrucción actual-5, cuya dirección puede variar entre ejecuciones (o incluso dentro de la misma ejecución).

Otro problema que surge es que la necesidad de memoria de todos los procesos activos pueden ser mayores que la RAM. La solución es el uso de memoria



virtual, la cual da al proceso la impresión de tener más memoria disponible de la que realmente tiene, y esta diferencia entre la memoria real y la virtual se suple con la carga de páginas de memoria bajo demanda. De esta forma no se requiere que todo el proceso esté cargado en memoria, sino unas ciertas páginas concretas, y se cargan las que se necesitan según la demanda del programa.

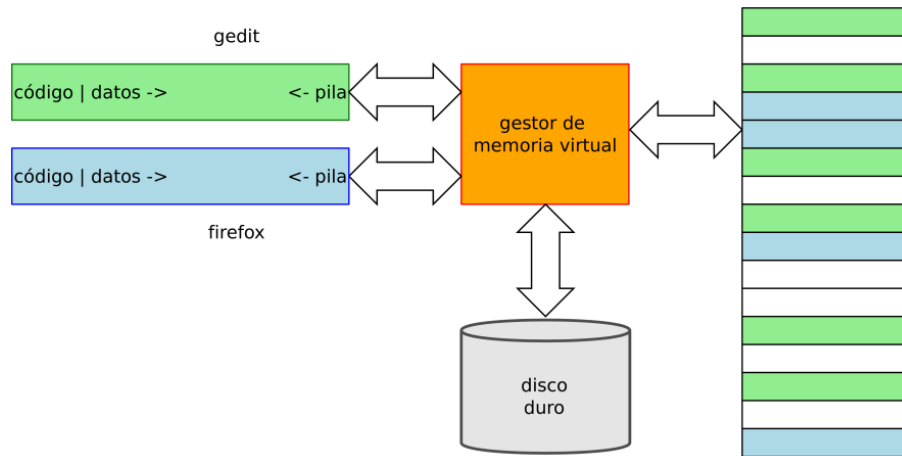
Memoria virtual: Permite al programador direccionar memoria de forma razonable en cuanto a

- Cantidad: los procesos creen disponer de toda la RAM.
- Seguridad: los espacios de direcciones son independientes. Una misma dirección virtual en dos espacios de direcciones distintos puede ser mapeada en diferentes posiciones en la RAM.
- El mapeado de porciones de memoria virtual en memoria física se hace de forma automática, liberando así al programador de esta tarea.

El funcionamiento eficiente de la memoria virtual requiere soporte hardware.

Con la memoria virtual las aplicaciones creen tener un espacio de direcciones plano. La memoria física se divide en porciones. Las regiones no necesitan mapearse de forma continua, a excepción de la E/S mapeada en memoria (contro-

ladores). (Para más información ver la explicación de memoria virtual del tema 1).



Planificación/gestión de recursos

- **Equidad:** normalmente, se desea que todos los procesos que compiten por un determinado recurso, se les conceda un acceso equitativo a dicho recurso. Esto es especialmente cierto para trabajos de la misma categoría, es decir, trabajos con demandas similares.
- **Tiempo de respuesta:** el sistema operativo puede necesitar discriminar entre diferentes clases de trabajos con diferentes requisitos de servicio. El sistema operativo debe tomar las decisiones de asignación y planificación con el objetivo de satisfacer el conjunto total de requisitos. Además, debe tomar las decisiones de forma dinámica.
- **Eficiencia:** el sistema operativo debe intentar maximizar la proximidad, minimizar el tiempo de respuesta, y, en caso de sistemas de tiempo compartido, acomodar tantos usuarios como sea posible. Estos criterios entran en conflicto: encontrar un compromiso adecuado en una situación particular es un problema objeto de la investigación sobre sistemas operativos.

Diferencia entre políticas y mecanismos:

- Planificación \Leftrightarrow apropiación/exclusión
- Paginación \Leftrightarrow reemplazo.
- Interacción \Leftrightarrow comunicación.

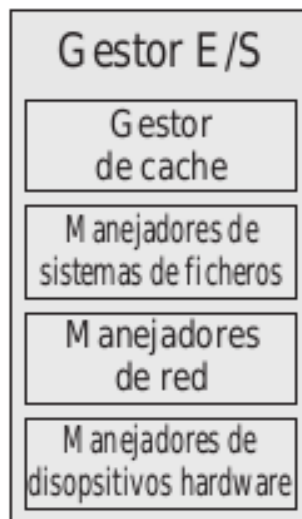
Gestión de E/S:

Clasificación de dispositivos de E/S:

- Dispositivos de caracteres: aquellos que envían o reciben un flujo de caracteres, sin sujetarse a una estructura de bloques. No se pueden utilizar direcciones ni tienen una operación de búsqueda. Ejemplos: puerto de serie, módem, ratón, ...
- Dispositivos de bloques: aquellos que almacenan la información en bloques de tamaño fijo, cada uno con su propia dirección. Ejemplos: discos duros, tarjetas de red, ...

Papel de la gestión de dispositivos \Rightarrow interfaz:

- Proporcionar una interfaz genérica, como en UNIX. En UNIX cada dispositivo de E/S está asociado con un fichero especial, que lo gestiona el sistema de ficheros y se lee y escribe de la misma manera que los ficheros de datos de usuario. Esto proporciona una interfaz bien definida y uniforme para los usuarios y los procesos. Para leer o escribir de un dispositivo, se realizan peticiones de lectura o escritura en el fichero especial asociado con el dispositivo.
- Proporcionar una interfaz específica para cada tipo de dispositivo, como en Windows. Windows dispone de un gestor de E/S, que es responsable de todo el sistema de E/S del sistema operativo y proporciona una interfaz uniforme a la que todos los tipos de manejadores pueden llamar.



Componentes de un controlador de dispositivo (software):

- Código de inicialización

- Llamada al sistema para responder a las peticiones de usuario
- Manejador de interrupción para responder a las peticiones del controlador de dispositivo (hardware)

Ficheros

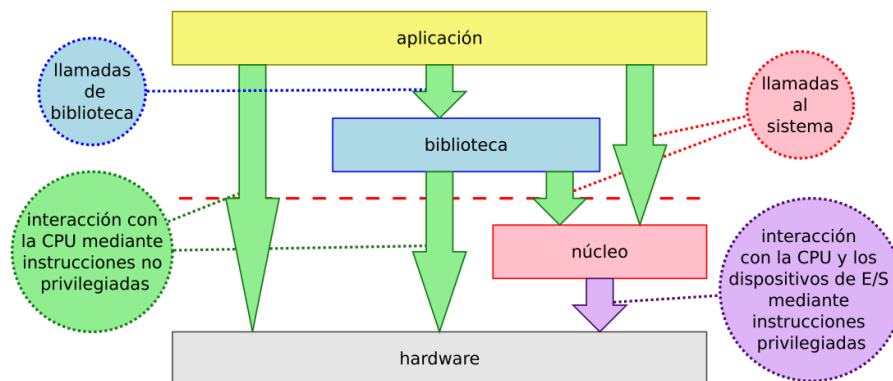
Implementación del almacenamiento persistente o a largo plazo. Las unidades almacenadas persistentemente son objetos como:

- Ficheros
- Directorios

Tipos de ficheros:

- Tradicionales: gestionados a través de llamadas al sistema.
- Mapeados en memoria: se gestionan igual pero se almacenan en la memoria principal.

Interacción de los componentes del sistema



Interfaz de programación de aplicaciones (API)

Existen dos interfaces de programación para acceder a los servicios proporcionados por el núcleo del SO:

- Llamadas al sistema: interfaz directa con el núcleo.
- API: interfaz indirecta escrita en algún lenguaje de alto nivel (funciones de biblioteca).

Las 3 APIs más conocidas son:

- Win32/Win64
- POSIX: Portable Operating System Interface (UNIX)

- Java

Paso de parámetros:

A menudo es necesaria más información que la identificación de la llamada al sistema. El tipo y cantidad de información varía entre llamadas.

Métodos de paso de parámetros:

- Registros: método rápido pero de poca capacidad
- Memoria: la aplicación agrupa los parámetros en un área de memoria y pasa la localización de dicha área mediante un registro.
- Pila: los parámetros son apilados por la aplicación y desapilados por la llamada al sistema.

El empleo de memoria tiene la ventaja de no limitar el número ni el tamaño de los parámetros, pero es más lenta que los registros (velocidad+copia). La estrategia más flexible para el paso de parámetros es la pila.

2.2 Llamadas al sistema

- Gestión de procesos

llamada	descripción
<code>pid = fork()</code>	crea un proceso hijo y devuelve su identificador
<code>waitpid(pid, estado, opciones)</code>	espera la finalización de un proceso hijo
<code>estado = execve(fichero, argumentos, entorno)</code>	reemplaza el núcleo de un proceso
<code>exit(estado)</code>	finaliza un proceso y devuelve un valor de estado

- **Gestión de ficheros:**

llamada	descripción
<code>descriptor = open(nombre, modo)</code>	abre un fichero y devuelve su descriptor
<code>estado = close(descriptor)</code>	cierra un fichero
<code>cantidad = read(descriptor, bufer, bytes)</code>	lee datos de un fichero
<code>cantidad = write(descriptor, bufer, bytes)</code>	escribe datos en un fichero
<code>posicion = lseek(descriptor, desplazamiento, desde)</code>	mueve el puntero del fichero
<code>estado = stat(nombre, &búfer)</code>	obtener información de estado

- **Gestión de directorios**

llamada	descripción
<code>estado = mkdir(nombre, modo)</code>	crea un nuevo directorio
<code>estado = rmdir(nombre)</code>	borra un directorio vacío
<code>estado = link(nombre1, nombre2)</code>	crea un enlace
<code>estado = unlink(nombre)</code>	borra una entrada de directorio
<code>estado = mount(origen, destino, tipo, opciones)</code>	monta un sistema de ficheros
<code>estado = umount(destino)</code>	desmonta un sistema de ficheros

- Otras llamadas al sistema

llamada	descripción
estado = chdir (nombre)	cambia el directorio de trabajo
estado = chmod (nombre, modo)	cambia los bits de protección
estado = kill (pid, señal)	envía una señal a un proceso
segundos = time (&segundos)	nº de segundos desde 1/1/1970

- Comparativa POSIX/Win32

POSIX	Win32/Win64	descripción
chdir	SetCurrentDirectory	cambia el directorio actual
chmod		cambia los permisos
close	CloseHandle	cierra un fichero
execve	CreateProcess	cambia la imagen de un proceso
exit	ExitProcess	finaliza la ejecución de un proceso
fork	CreateProcess	crea un nuevo proceso
kill		envía una señal
link		crea un enlace
lseek	SetFilePointer	mueve el puntero de fichero
mount		monta un sistema de ficheros
mkdir	CreateDirectory	crea un directorio
open	OpenFile	crea/abre un fichero
read	ReadFile	lee de un fichero
rmdir	RemoveDirectory	borra un directorio
stat	GetFileAttributesEx	obtener atributos de un fichero
time	GetLocalTime	obtiene la hora
unlink	DeleteFile	borra un fichero
unmount		desmonta un sistema de ficheros
waitpid	WaitForSingleObject	espera la finalización de un proceso
write	WriteFile	escribe en un fichero

3 Tema 3: Historia de los Sistemas Operativos

3.1 Definición

Sistema operativo: programa o conjunto de programas encargado de gestionar los recursos de la máquina y proveer servicios al resto de programas. Suele ejecutarse en modo privilegiado. Controla la ejecución de aplicaciones y programas y actúa como interfaz entre las aplicaciones y el hardware del computador.

Suelen venir junto con multitud de otros programas, que sin forma parte del sistema operativo, resultan de suma utilidad: intérprete de órdenes, editor de texto, gestor de ficheros, navegador, ...

3.2 Historia

3.2.1 Primera generación (1945-1955): Tubos de vacío y paneles

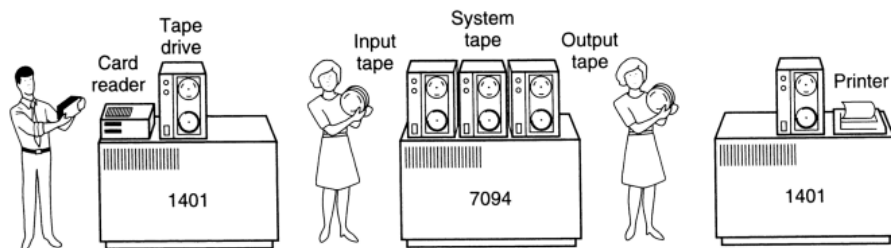
En esta primera generación los sistemas operativos servían como máquinas de cálculo. Se trataban de dispositivos mecánicos que utilizaban tubos de vacío y paneles.

Para programas estos se utilizaban cables, interruptores y tarjetas perforadas. Un gran problema de estos era que el diseño, construcción, operación, programación y mantenimiento requería de genios como Aiken, von Neuman o Mauchley.

3.2.2 Segunda generación (1955-1965): Transistores y sistemas por lotes

En esta segunda generación los sistemas operativos se utilizaban para el cálculo científico y la ingeniería. La invención del transistor permitió que se redujera el tamaño y precio de estos y los popularizó dando lugar a mainframes como el IBM 1401 o el IBM 7094.

Para programas estos se utilizaba lenguaje ensamblador y lenguajes de alto nivel (FORTRAN) sobre tarjetas perforadas. Por otro lado, se pasó del procesamiento secuencial al procesamiento por lotes. Ejemplos: FMS y IBSYS.



3.2.3 Tercera generación (1965-1980): Circuitos integrados y multiprogramación

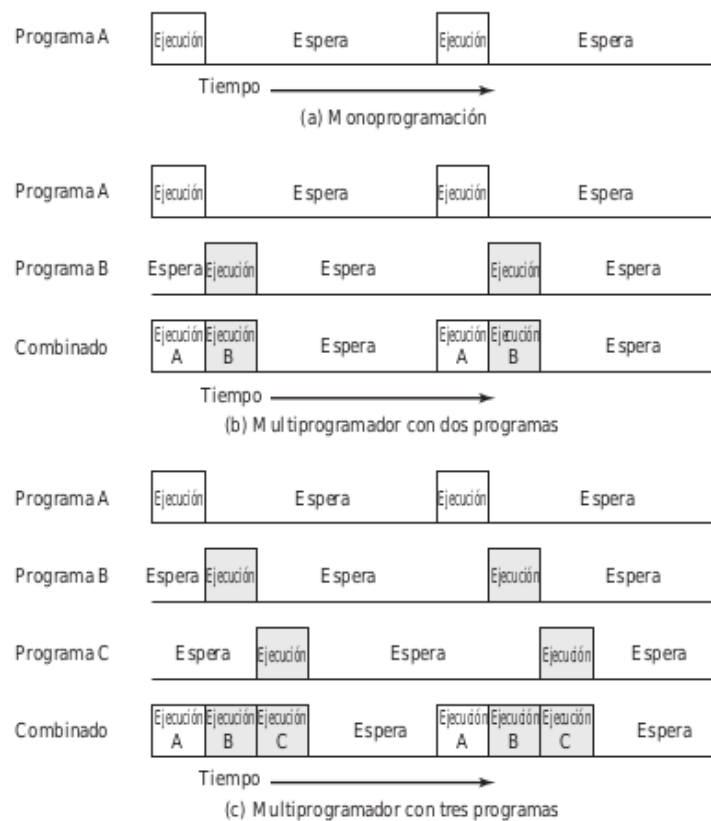
En esta tercera generación a parte de seguir utilizandose los computadores para el cálculo científico y la ingeniería se utilizan para el procesamiento de caracteres.

Se comienzan a utilizar circuitos integrados que son más baratos y se convierten en más populares (IBM 360, GE-645, DEC PDP-1).

Como logro más destacables tenemos:

- Multiprogramación.
- Spooling.
- Tiempo compartido.

Ejemplos: OS/360, CTSS, MULTICS, UNIX.



3.2.4 Cuarta generación (1980-hoy): Ordenador personal (era μ)

La empresa (V)LSI de circuitos integrados proporciona circuitos más baratos y por consiguiente se convirtieron en más populares, dando lugar a computadores

como el IBM PC.

Surgen arquitecturas de procesadores como los 8080, Z80, 80x86, Alpha, Ultraspac, ARM.

Como logros destacables tenemos:

- GUI (Graphic User Interface).
- SO de red. Sistema operativo especializado para dispositivos de red como pueden ser un router o un switch.
- SMP (Symmetric Multi-Processing).
- SO distribuidos.

Ejemplos: UINX, CP/M, MS-DOS, Linux, MacOS, Windows.

3.3 Estructura

Clasificación:

- Estructura simple:
 - Monolíticos. Son sistemas operativos en los cual el núcleo proporciona la mayoría de las funcionalidades propias del sistemas operativo, incluyendo la planificación, los sistemas de ficheros, las redes y otras funciones.
 - Capas. Las funciones se organizan jerárquicamente y sólo hay interacción entre las capas adyacentes. Con el enfoque por capas, la mayor parte o todas las capas ejecutan en modo núcleo.
 - Modulares
- Estructura cliente/servidor:
 - Micronúcleo. Asigna unas pocas funciones esenciales al núcleo, incluyendo los espacios de almacenamiento, comunicación entre procesos y planificación básica.
 - Exonúcleo.
- Máquina virtual.
- Híbridos

Tendencias:

- Núcleos Extensibles
- Multiservidores sobre un micronúcleo.

- Núcleos híbridos.

Monolítico:

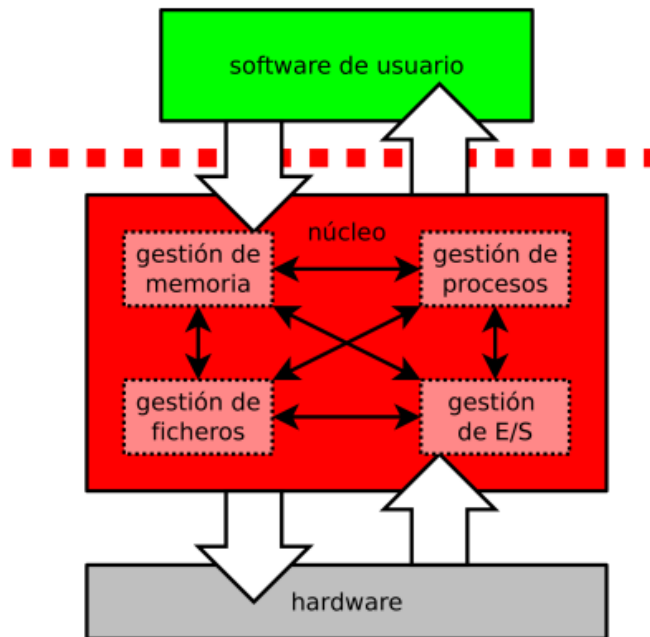
El Sistema operativo completo se ejecuta en modo protegido. Como consecuencia hay una protección nula entre los componentes del mismo, pues cualquier fallo puede afectar a cualquier parte del sistema operativo.

Ventajas:

- Es muy eficiente gracias a la economía de cambios de contexto. Esto se debe a que gracias a que todo el SO se ejecuta en modo protegido no requiere de cambios de contexto mientras se ejecutan llamadas al sistema.

Desventajas:

- La falta de protección implica una menor fiabilidad.
- Mal manejo de la complejidad. Es más sencillo escribir 1000 programas de 1000 líneas que uno de 1000000



Capas/Niveles:

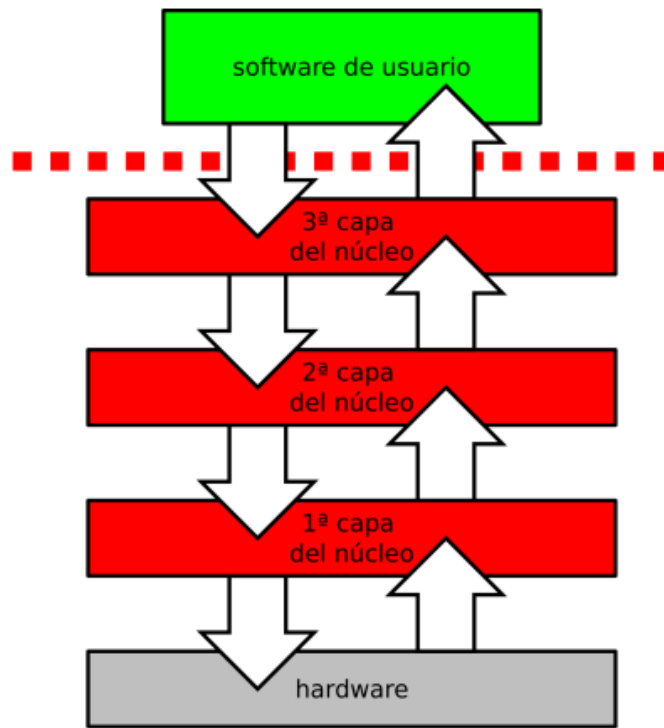
El Sistema Operativo completo se ejecuta en modo protegido. Hay una escasa división entre los componentes.

Ventajas:

- Es muy eficiente gracias a la economía de cambios de contexto.
- La complejidad es menor que la de los sistemas operativos monolíticos.

Desventajas:

- La falta de protección implica una menor fiabilidad.
- Menos flexible que un sistema operativo monolítico.
- Es difícil subdividir adecuadamente las capas.



Modular:

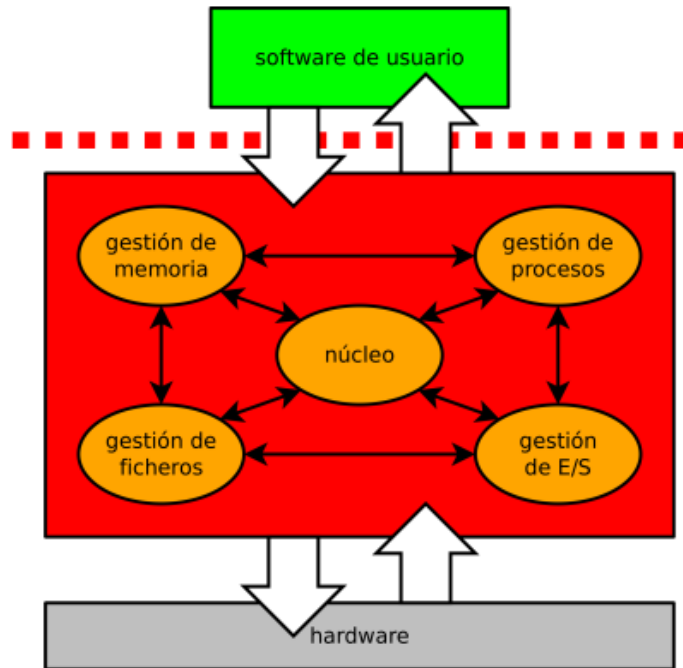
El Sistema Operativo se ejecuta en modo protegido. Una vez más esto implica una escasa protección entre los componentes.

Ventajas:

- De nuevo la economías de cambios de contexto lo hace eficiente.
- Tiene una menos complejidad.

Desventajas:

- La falta de protección implica una menor fiabilidad.
- Es menos flexible que un sistema operativo monolítico.
- Es difícil elegir que colocar en el núcleo y que en los módulos.



Micronúcleo/Microkernel

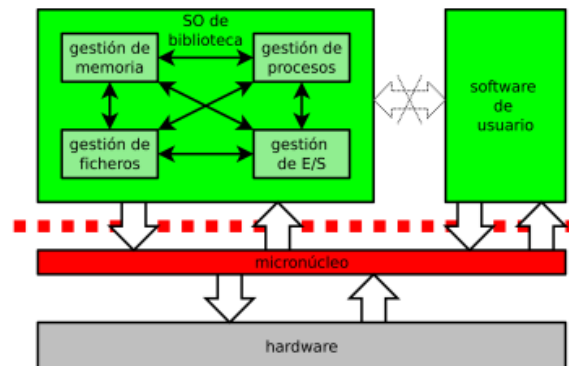
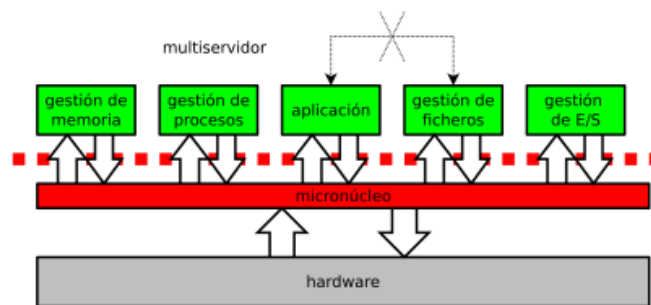
Una mínima parte del Sistema Operativo se ejecuta en modo protegido.

Ventajas:

- La perfecta protección entre componente nos da una mayor fiabilidad
- Manejo de la complejidad.
- Facilidad de programación

Desventajas:

- Debido a la sobrecarga en las comunicaciones hay una menor eficiencia.



Exonúcleo:

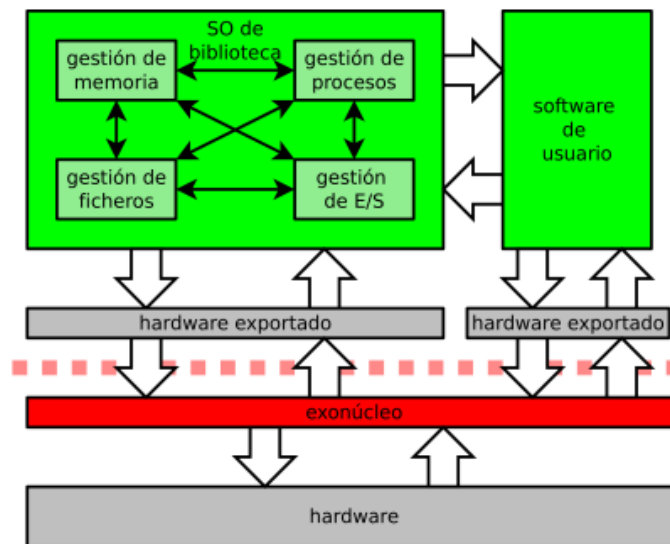
Apenas existe el sistema operativo, sólo un gestor de recursos. En este tipo de sistema operativo dejamos que el software acceda directamente al hardware.

Ventajas:

- Perfecta protección entre componentes, lo cual da lugar a una alta fiabilidad.
- El acceso directo al hardware nos da lugar a una máxima eficiencia.

Desventajas:

- Pobre reutilización del código.



Máquina virtual:

Se trata de copias virtuales de la máquina rela:

- SW: Bochs, Qemu, VMWare, Xen.
- HW: VMWare, Xen

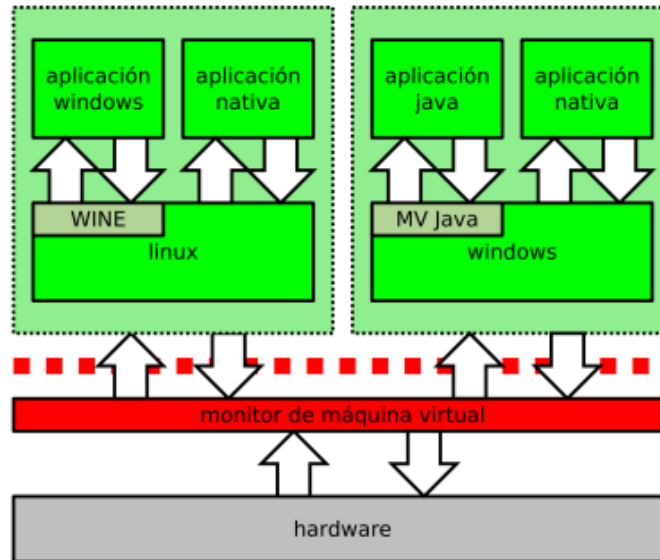
Ventajas:

- Perfecta protección entre componentes, lo que garantiza una alta fiabilidad.
- Mejor aprovechamiento del hardware.
- Máxima reutilización del código.

Desventajas:

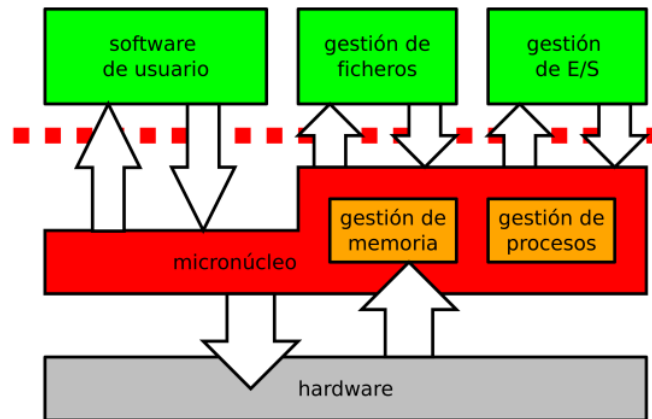
- La simulación del hardware real puede ser costosa, dando lugar a una baja eficiencia.

Esta opción es un éxito gracias a mejoras en el HW.



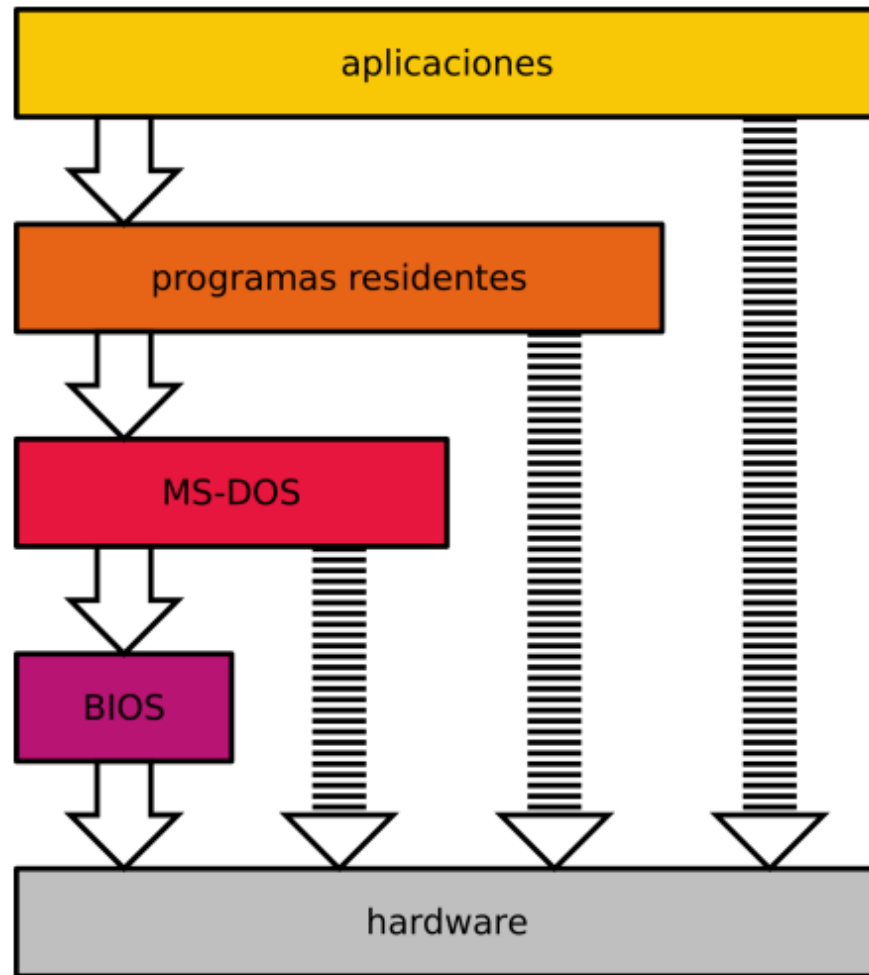
Híbrida

- La mezcla más frecuente es la combinación de microneúcleo y monolítico.
- Ventaja: ganamos velocidad respecto al microneúcleo.
- Desventaja: perdemos protección entre componentes.

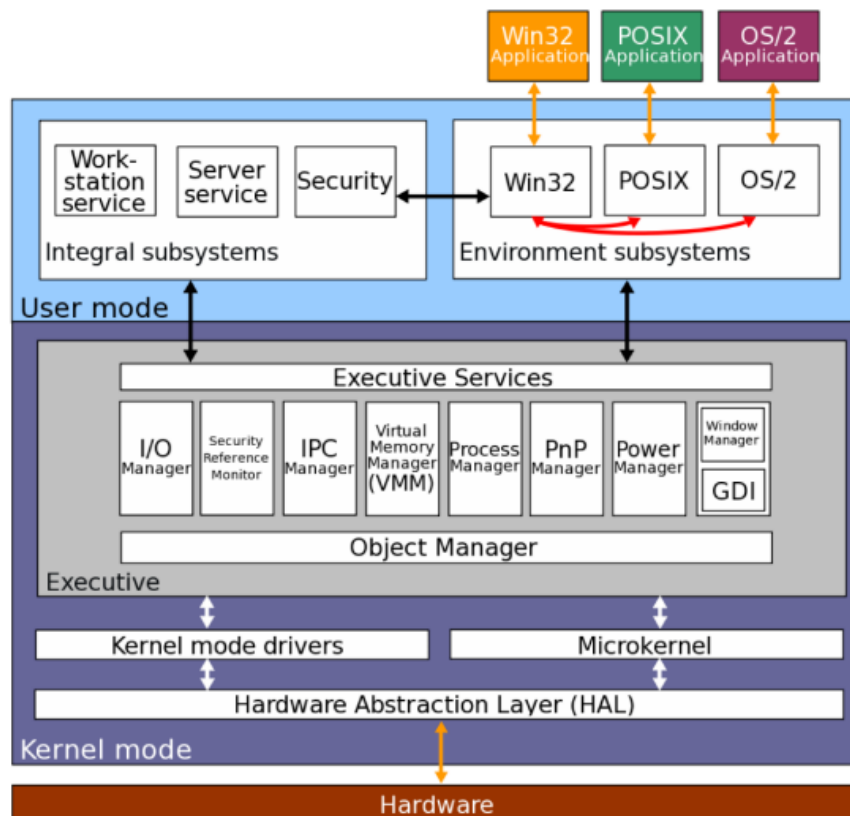


3.4 Ejemplos

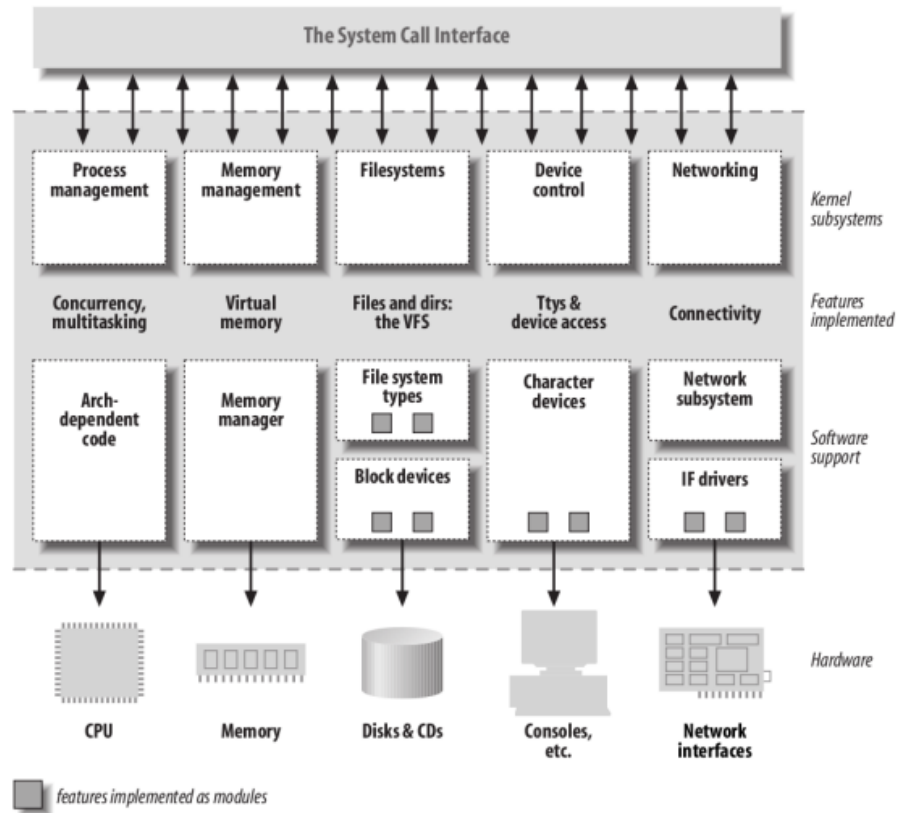
MS-DOS



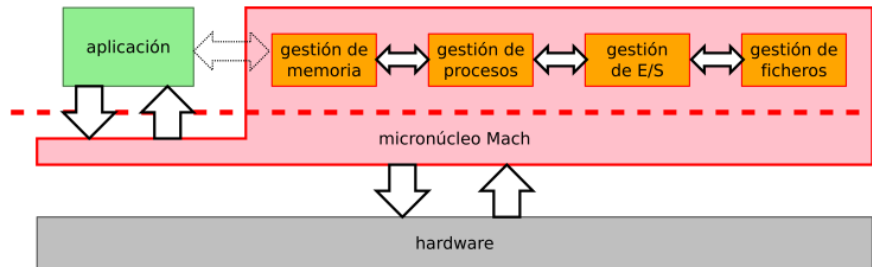
Windows 2000



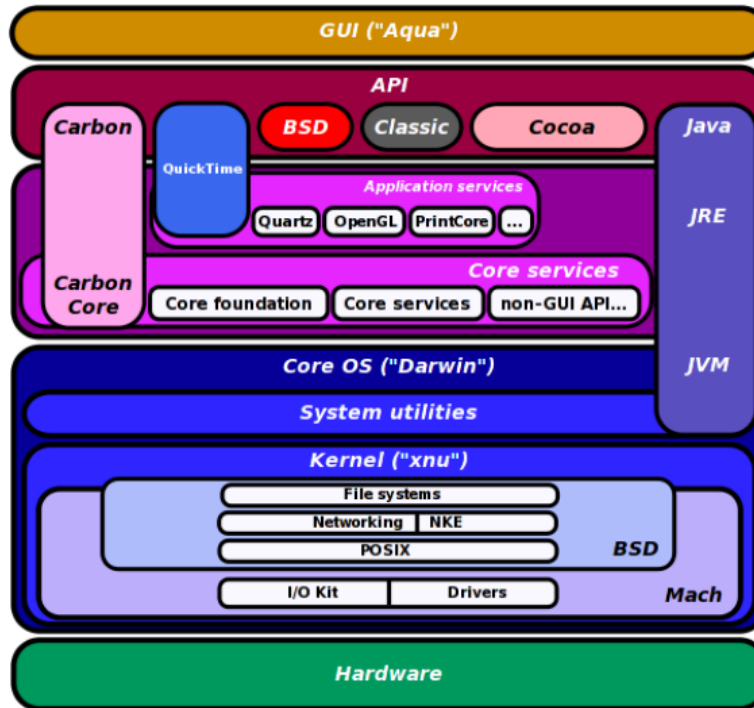
Linux



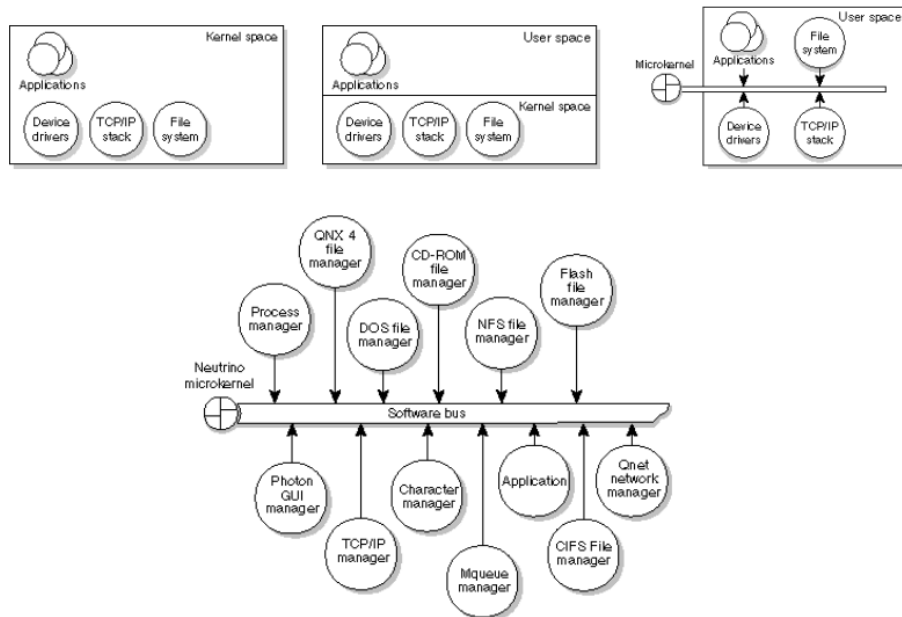
Mach



MacOS X



QNX



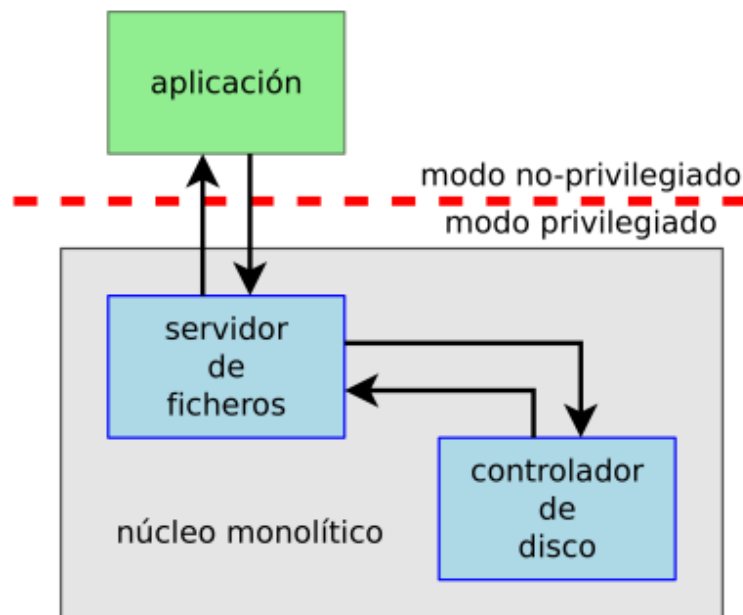
3.5 Comparativa

3.5.1 Coste estructural de una llamada al sistema: caso monolítico

1 llamada al sistema:

1. Entrada al núcleo.
2. Cambio al espacio de direcciones del núcleo.
3. Salida del núcleo.

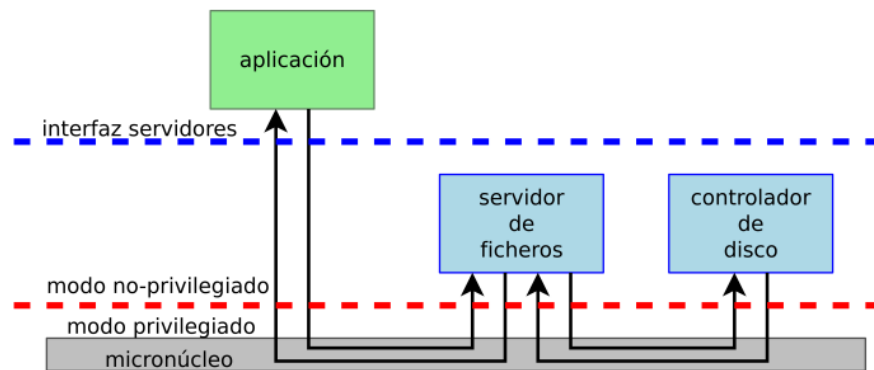
1 llamada a procedimiento: llamada y retorno en el interior del espacio de direcciones del núcleo y pudiendo compartir información.



3.5.2 Coste estructural de una llamada al sistema: caso micronúcleo

4 llamadas al sistema:

1. Entrada al micronúcleo.
2. Cambio al espacio de direcciones del micronúcleo.
3. Transferencia del mensaje.
4. Recuperar el espacio de direcciones original.
5. Salida del micronúcleo.



3.5.3 Coste estructural: multiservidor

