

Apuntes de Arquitectura de Sistemas

Daniel Monjas Miguélez

April 5, 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)	53
3.3 Estructura	54
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 microcódigo	67
3.5.3 Coste estructural: multiservidor	68
4 Tema 4: Procesos	68
4.1 Definición	68
4.2 Control	70
4.3 Ejecución del Sistema Operativo	77
4.3.1 Núcleo independiente	77
4.3.2 Ejecución dentro de los procesos de usuario	77
4.3.3 Sistemas operativos basados en procesos	78
5 Tema 5: Hebras	84
5.1 Definición	84
5.2 Comparativo Procesos/Hebras	85
5.3 Ejecución multihebra	85

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 concebió 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.

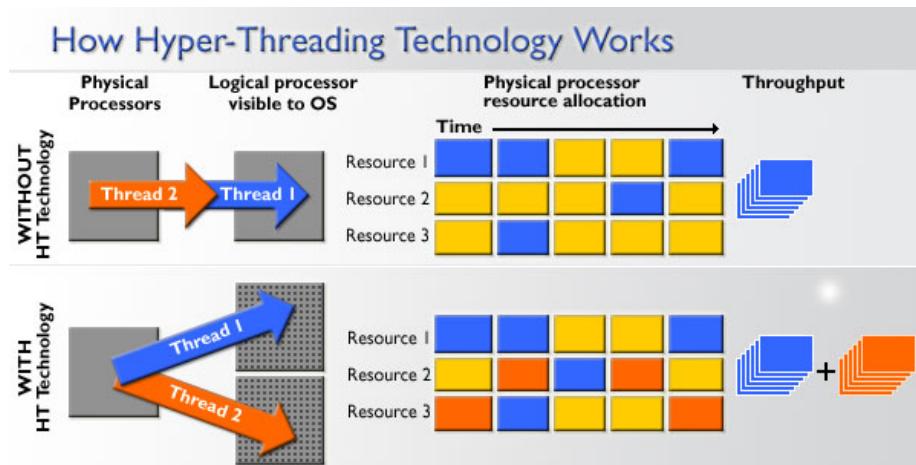
Todas las páginas de un proceso se mantienen en disco. Cuando un proceso está en ejecución, algunas de sus páginas se encuentran en memoria principal, y si se referencia a una página que no está en memoria principal el hardware de gestión de memoria lo detecta y permite que la página que falta se cargue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

1.2 Clasificación

1.2.1 Clasificación "práctica" de arquitecturas paralelas

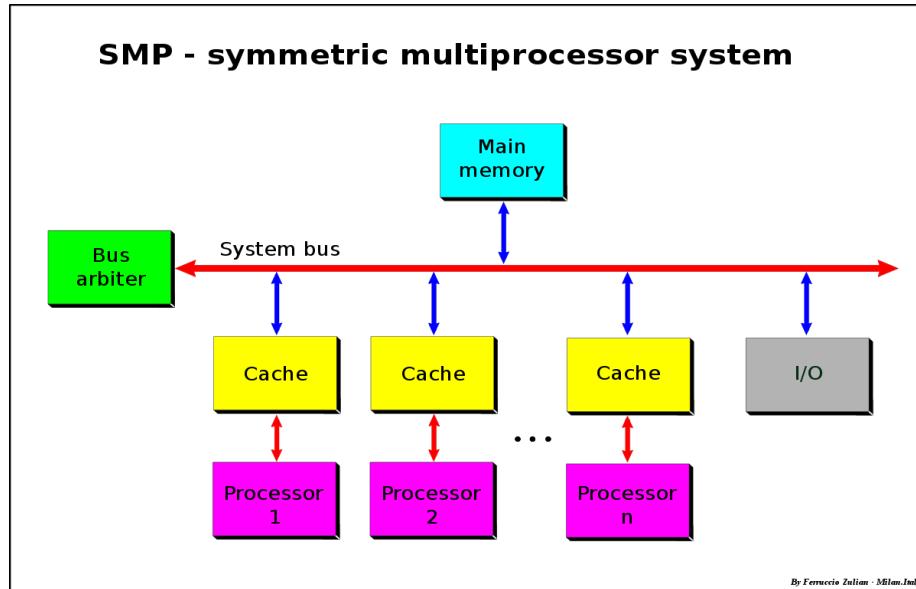
- Multiprocesadores de memoria compartida:

- SMT(Simultaneos 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.

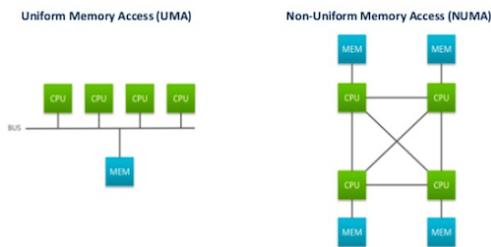


- SMP(Symmetric Multi-Processing): el núcleo puede ejecutar en cualquier procesador, y normalmente cada procesador realiza su propia planificación del conjunto disponible de procesos e hilos. El núcleo puede construirse como múltiples procesos o múltiples hilos, permitiéndose la ejecución de partes del 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

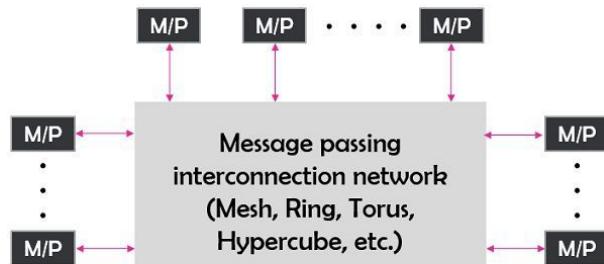
de la cola. Se deben emplear técnicas para resolver y sincronizar el uso de los recursos. Suelen tener entre 2 y 256 procesadores.



- 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.

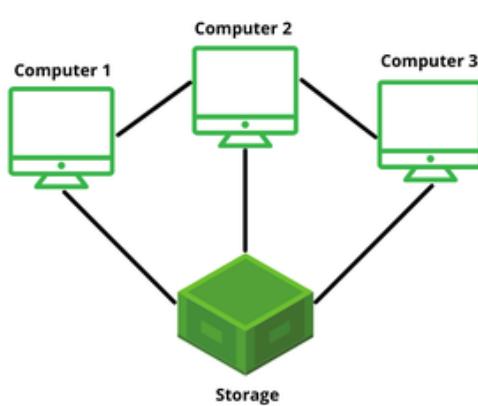


- Paso de mensajes/NoRMA (No Remote Memory Access): en las arquitecturas NoRMA, el espacio de direcciones global no es único y la memoria no es globalmente accesible desde todos los procesadores. El acceso a modulos de memoria remotos es solo posible indirectamente a través del paso de mensajes por medio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



NORMA (No-Remote Memory Access)

- 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 en que los clústeres se componen de dos o más sistemas individuales unidos juntos, a los que se denominan nodos.
 - GPUs



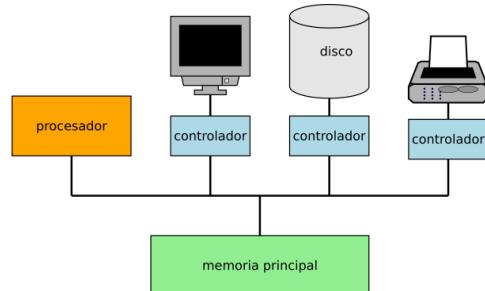
1.2.2 Clasificación Arquitectura de Computadores

- Sistemas monoprocesador

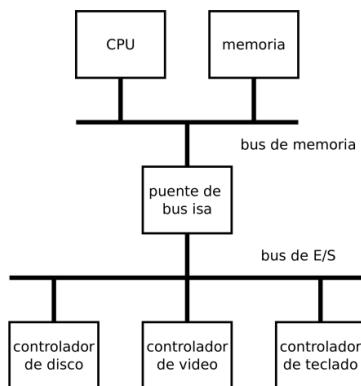
- Bus único
- Buses separados/especializados
- Sistemas multiprocesador
 - Multiproceso simétrico (SMP)
 - Multihebra simultánea (SMT)
 - Multinúcleo (SMP)
- Sistemas distribuidos

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

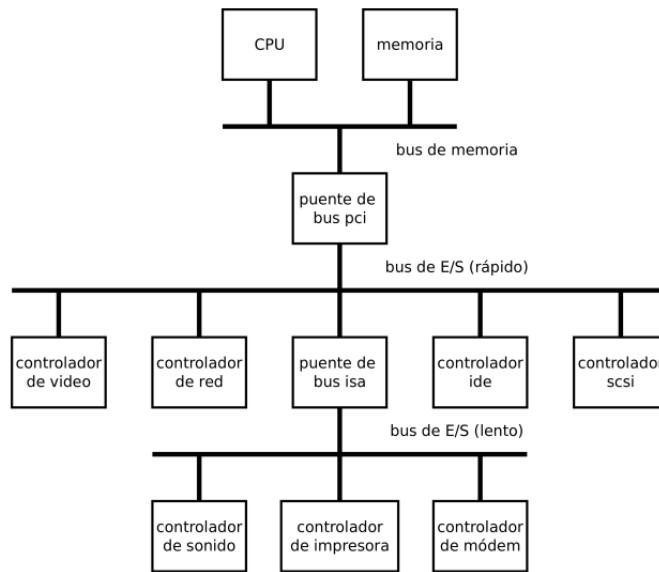
- Ventaja → precio.
- Inconveniente → infrautilización de componentes por la diferencia de velocidad.



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

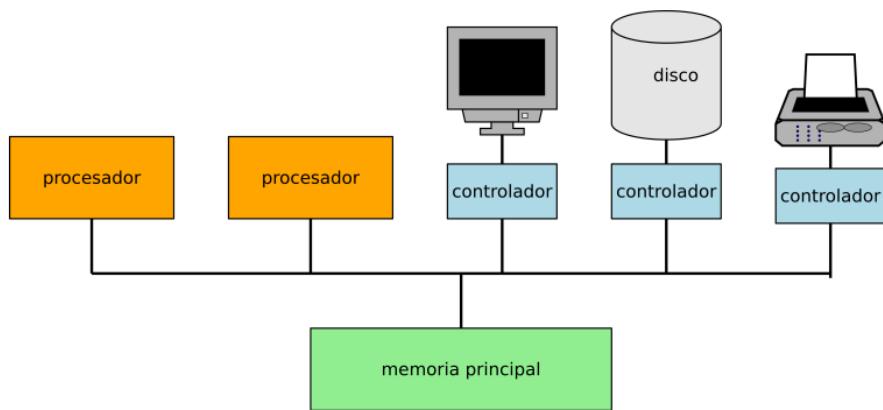


Otra posible solución incluso mejor es separar el bus de E/S en dos buses en función de los dispositivos E/S más rápidos y más lentos, y conectar ambos buses por medio de un puente isa.

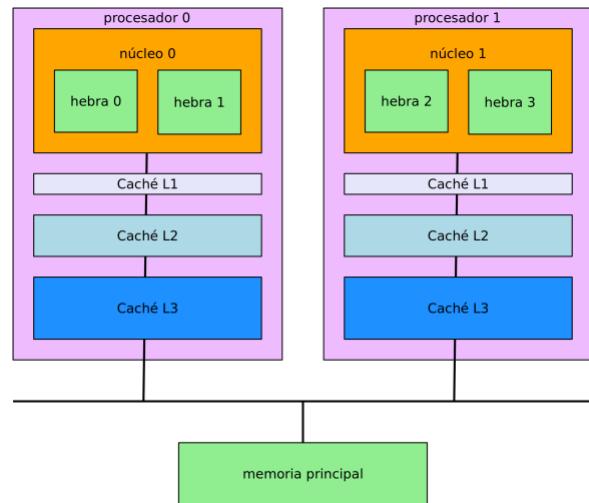


Sistema multiprocesador: multiproceso simétrico. Lo más simple es conectar todos los elementos a un bus común.

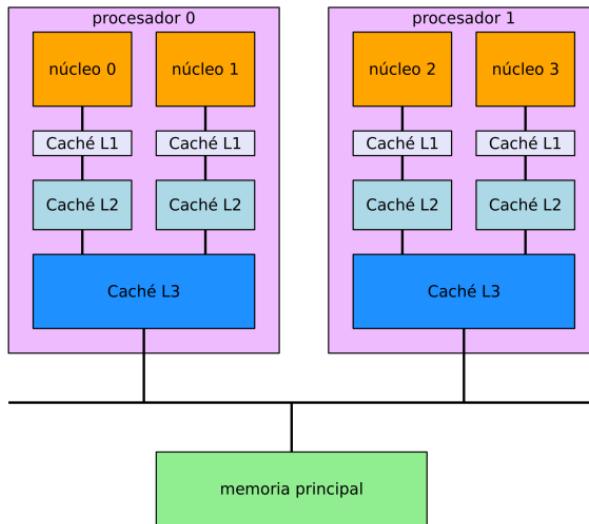
- Ventaja → precio.
- Inconveniente → se agrava la infrautilización de componentes.



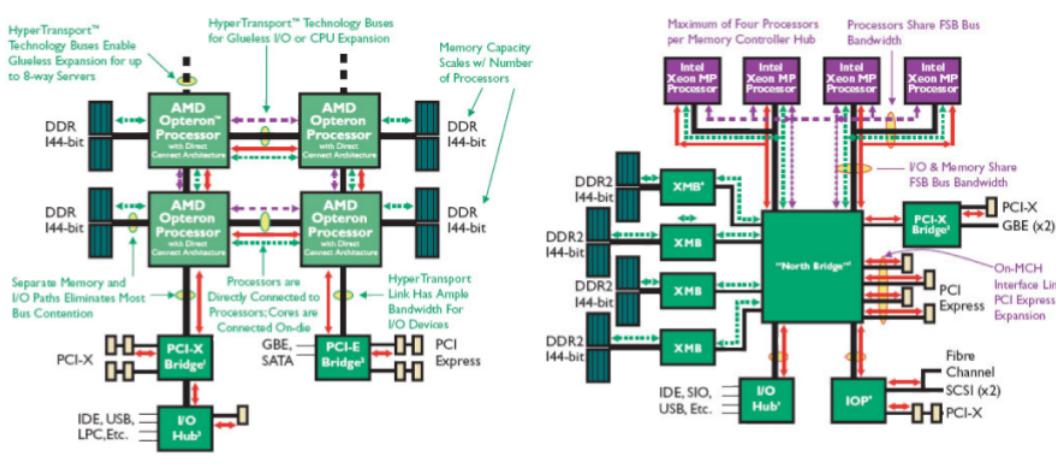
Sistemas multiprocesador: multihebra simultánea



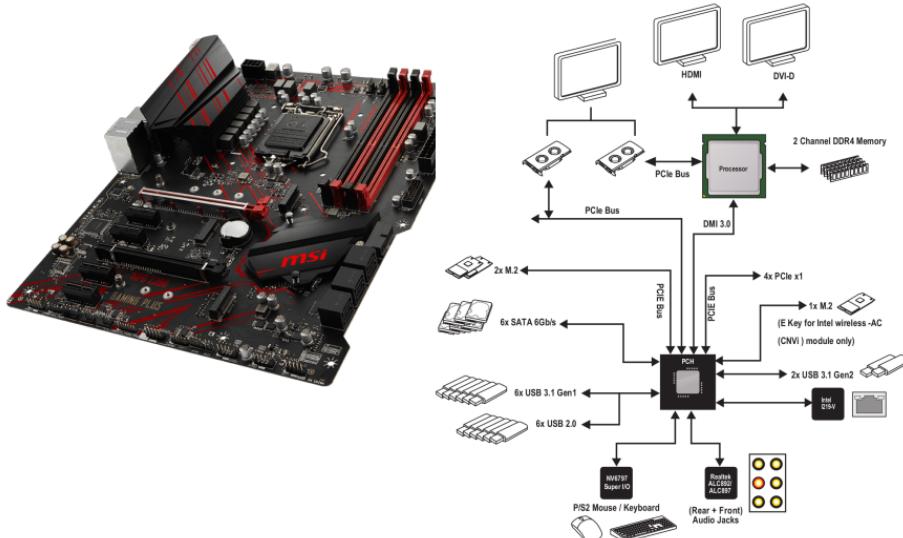
Sistemas multiprocesador: multiproceso simétrico



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.

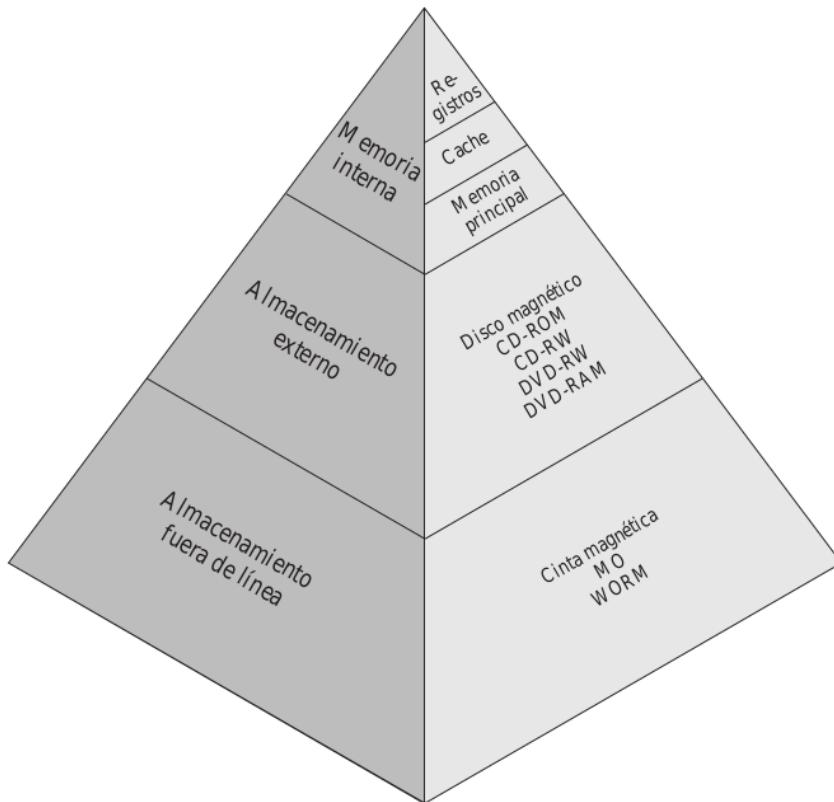


Figura 1.14. La jerarquía de memoria.

- 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, ...

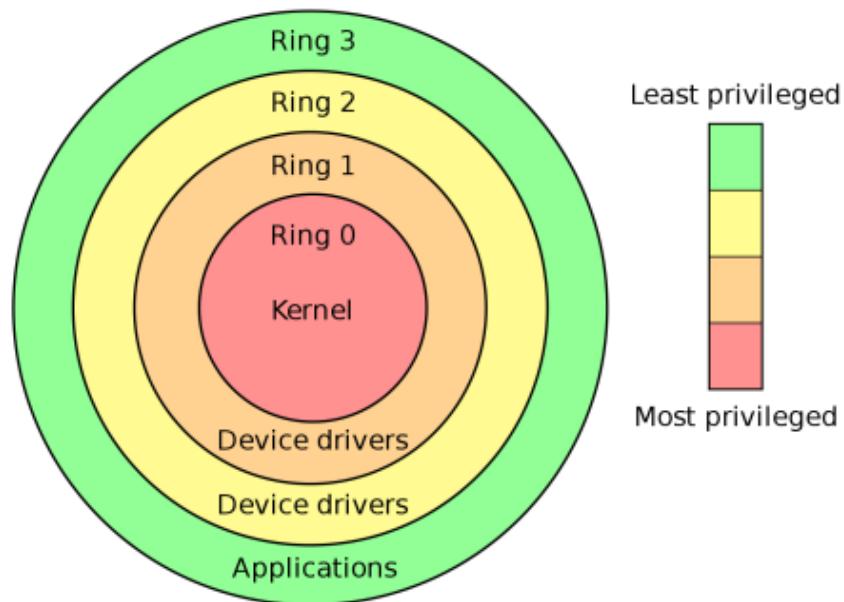
1.2.4 Procesador

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 ⇒ excepción



Registros del procesador: familia x86-64

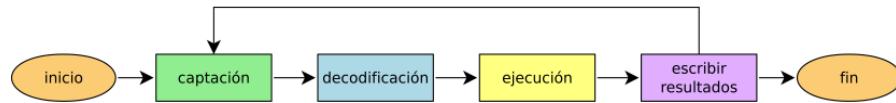
En el modo núcleo (modo privilegiado, modo kernel, ...) se pueden ejecutar instrucciones privilegiadas y se puede acceder a áreas de memoria protegidas. Sólo el núcleo o kernel del sistema operativo puede ejecutar instrucciones en modo privilegiado. Algunos ejemplos de instrucciones privilegiadas son:

- 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) ⇒ RISC (Reduced Instruction Set Computing) ⇒ 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.

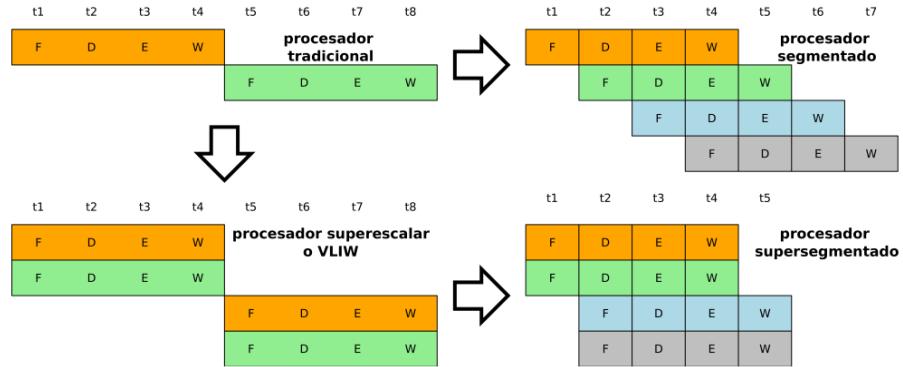
RISC: Como su nombre dice se trata de un diseño de microprocesador. Este diseño contiene instrucciones muy simples y con la combinación de ellas se puede obtener cualquier programa. Si bien su ejecución es rápida, pues son instrucciones muy simples, su tamaño en memoria puede llegar a ser muy grande,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

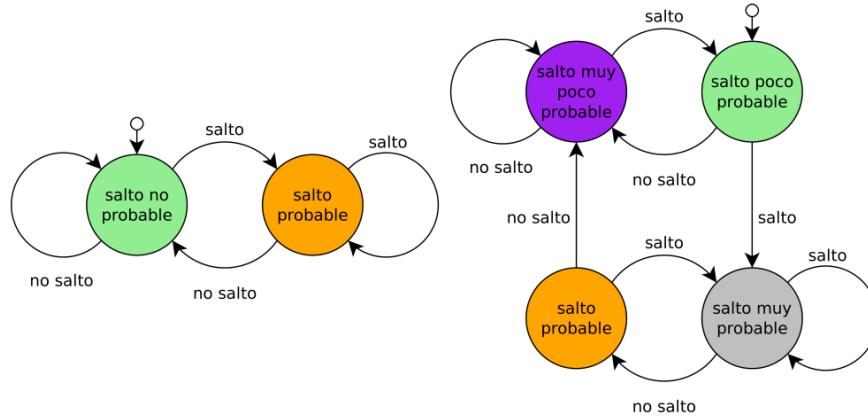
- Ejecución concurrente sobre procesadores: intento de explotación del paralelismo entre instrucciones (ILP). ILP es una medida de cuántas operaciones pueden ejecutarse simultáneamente sin afectar al resultado.

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 estapas, 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.



(a) Algoritmo de predicción de dos estados:
funciona bien en bucles salvo entrada y salida.

(b) Algoritmo de predicción de cuatro estados:
permite recordar estado previo de un bucle.

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

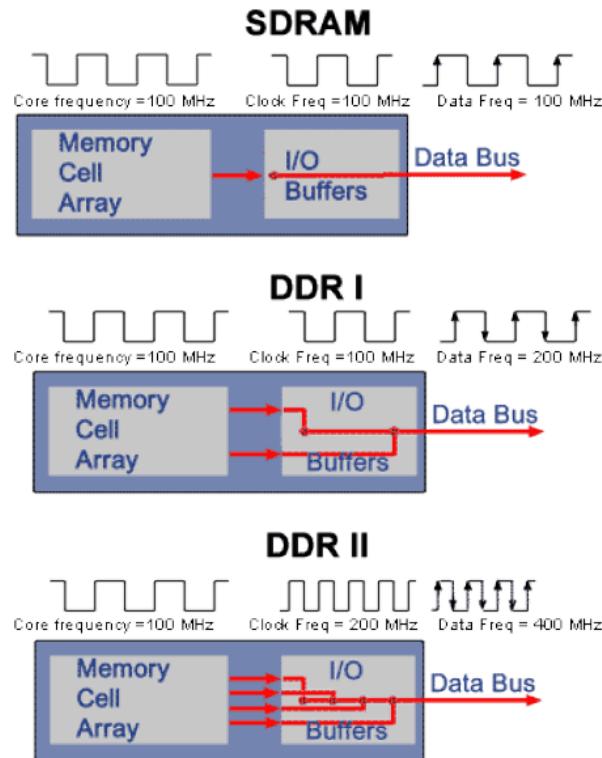
1.2.5 Memoria

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

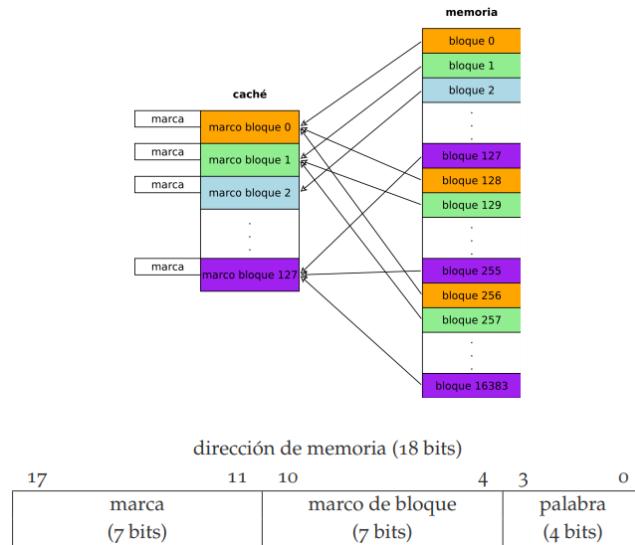
$$\bar{T}_{acceso} = (1 - T_{fallos}) \times T_{cache} + T_{fallos} \times (T_{cache} + T_{ram})$$

Parámetros de diseño de memorias caché

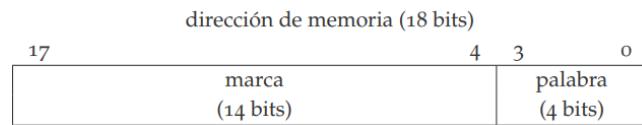
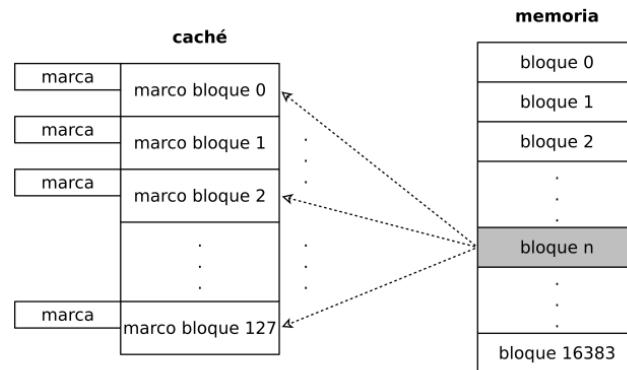
- Tamaño:
 - L1:8,...,256KB
 - L2:1,...,16MB
 - L3:4,...,128MB
- Tamaño de bloque: 32,...,128B
- Tiempo de acceso: 1,...,10ns
- Política de búsqueda:
 - bajo demanda
 - anticipativas
- Política de colocación:
 - correspondencia directa: el bloque B_j de memoria principal se puede ubicar sólo en el marco 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

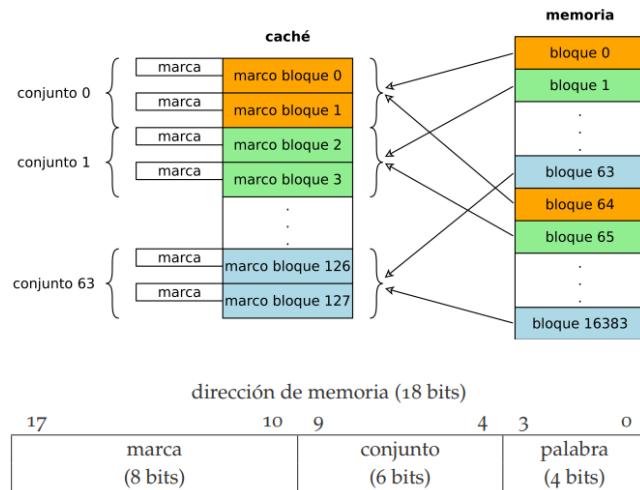
Políticas de colocación: correspondencia directa



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 accesible 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 del programador)	Los segmentos de programa se especifican por el programador al compilador (por ejemplo, la decisión se toma por parte del 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 tabla 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 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 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 ejecute. 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):

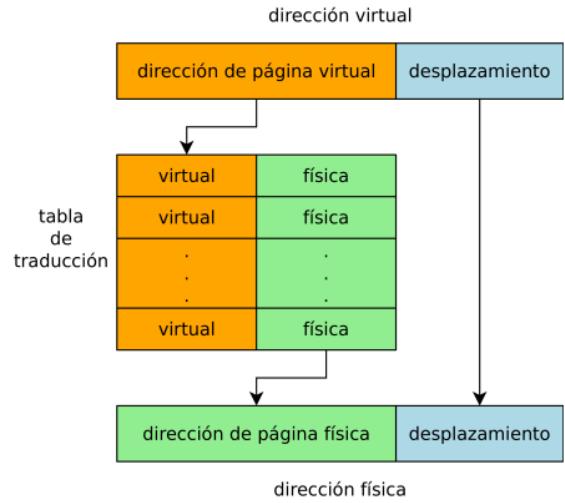
- Ventaja ⇒ protección automática.
- Inconveniente ⇒ dificulta la compartición

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)

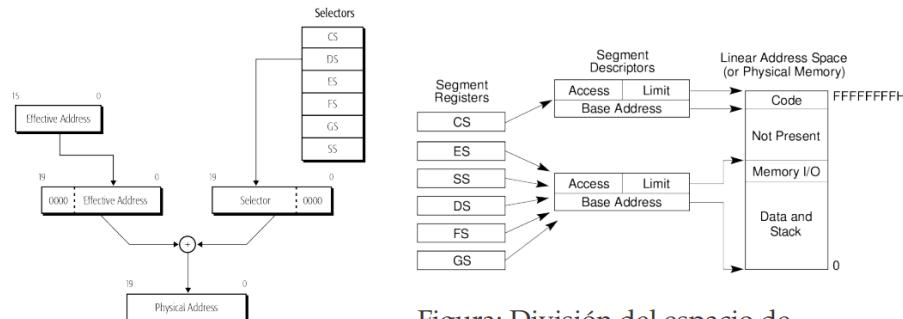


Figura: Cálculo de la dirección física.

Figura: División del espacio de direcciones de un proceso.

Segmentación + paginación (x86_64)

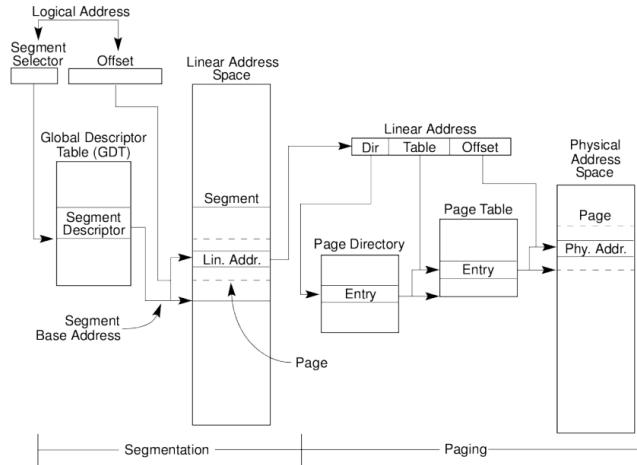
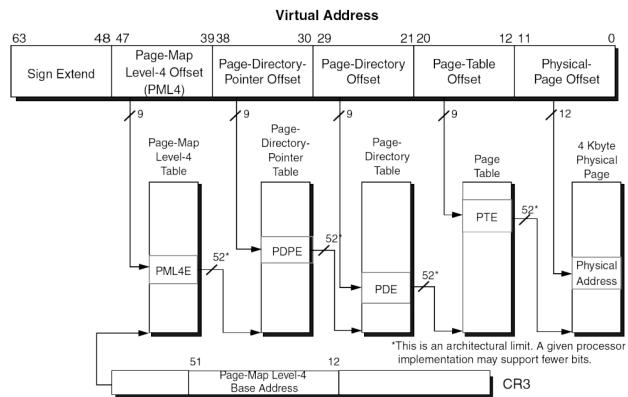


Tabla de páginas multinivel (x86_64)



Estructura habitual tabla de páginas

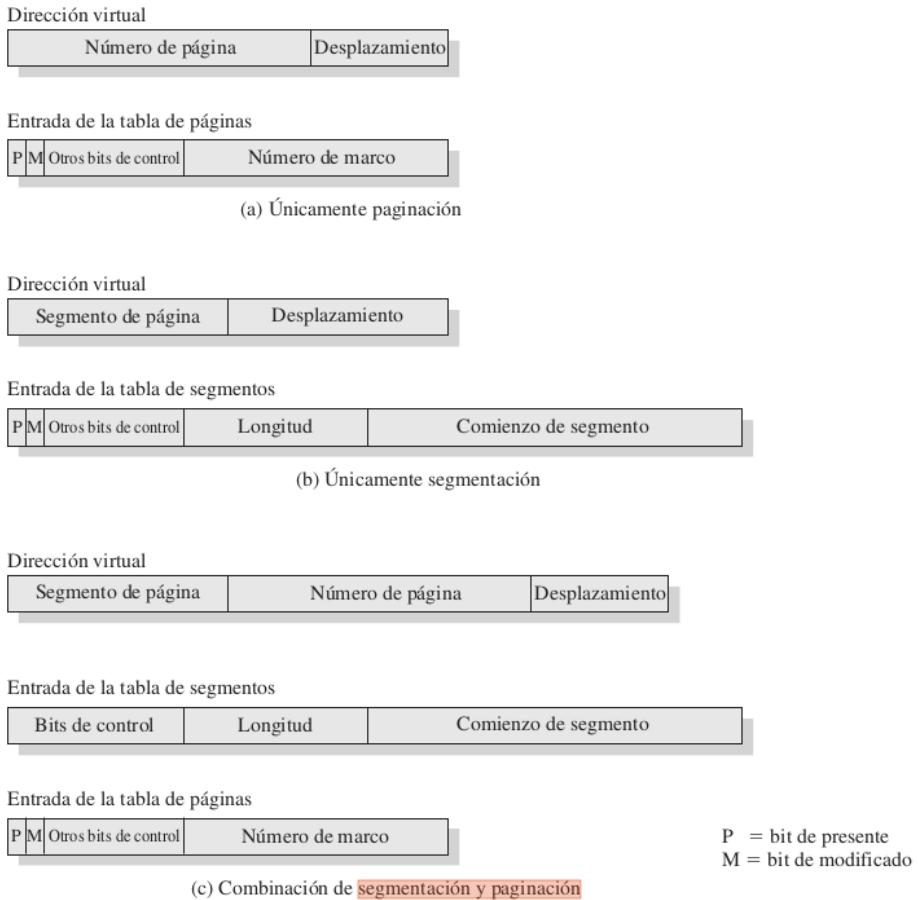


Figura 8.2. Formatos típicos de gestión de memoria.

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:

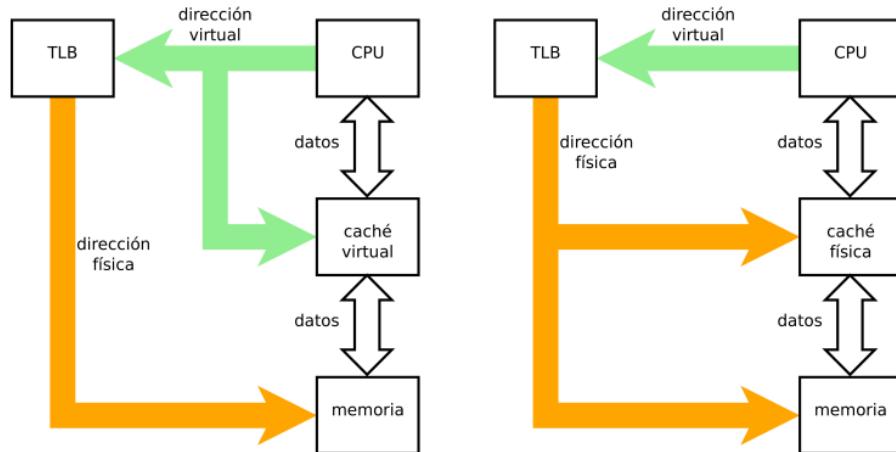
- Contenido: pares de direcciones virtual/física.
- Implementado mediante memorias asociativas.
- Tamaño típico: de 32 a 128 entradas.

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:

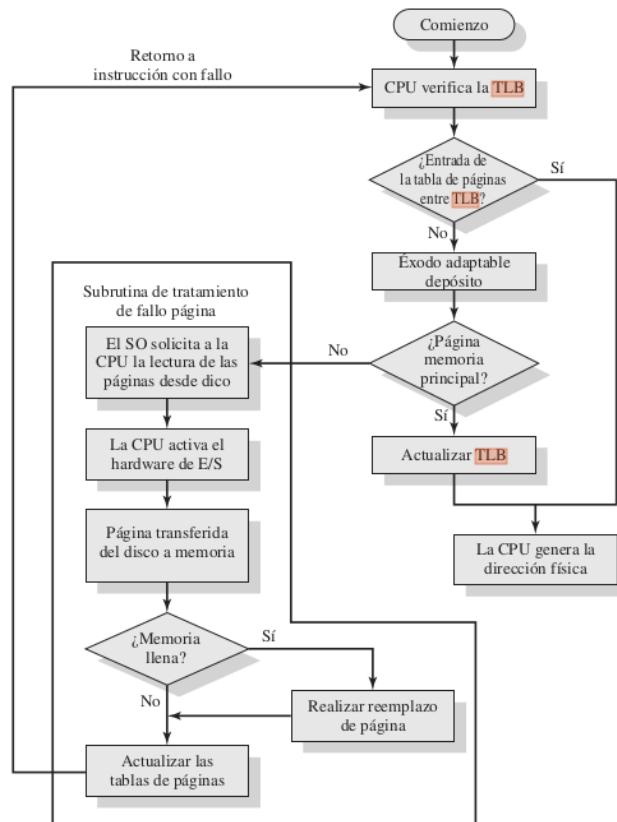
- Si se encuentra la dirección en el TLB se devuelve su traducción.
- Si no se encuentra es necesario calcular la traducción.
 - Hardware: se devuelve la traducción correcta (x86).
 - Software: más lento pero más flexible (PowerPC).
- Es fundamental conocer la interacción TLB/caché.



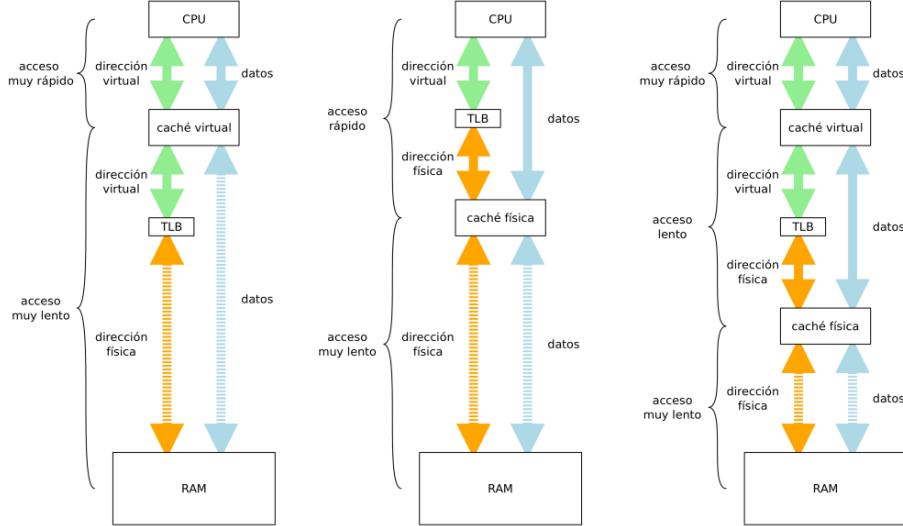
El proceso que sigue generalmente es el siguiente:

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 este 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 recuperarán 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.

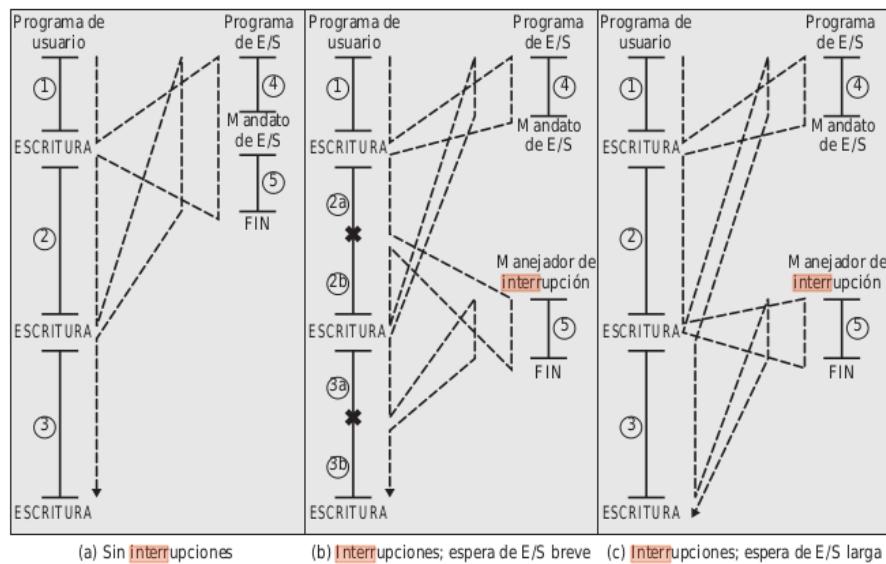


Figura 1.5. Flujo de programa del control sin **interrupciones** y con ellas.

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.

Excepciones síncronas e internas al procesador:	Interrupciones asíncronas y externas al procesador:
<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.	<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.

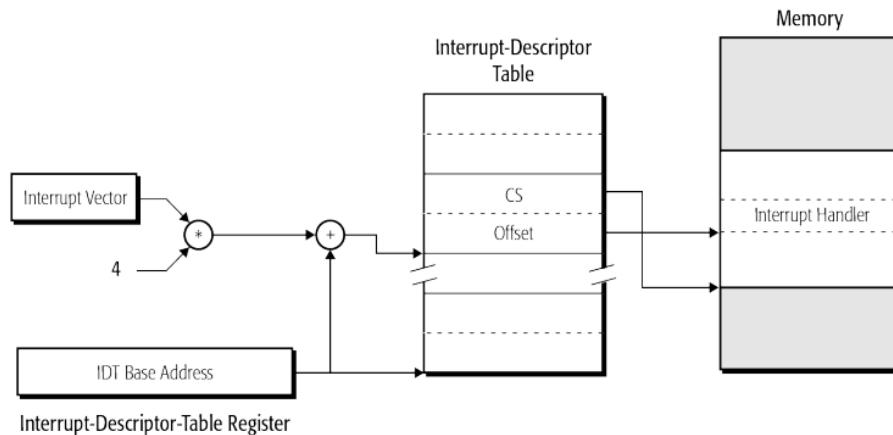


Figura: Vector de interrupciones (procesadores 80x86).

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 si 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 mismo 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 comúnt. 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.

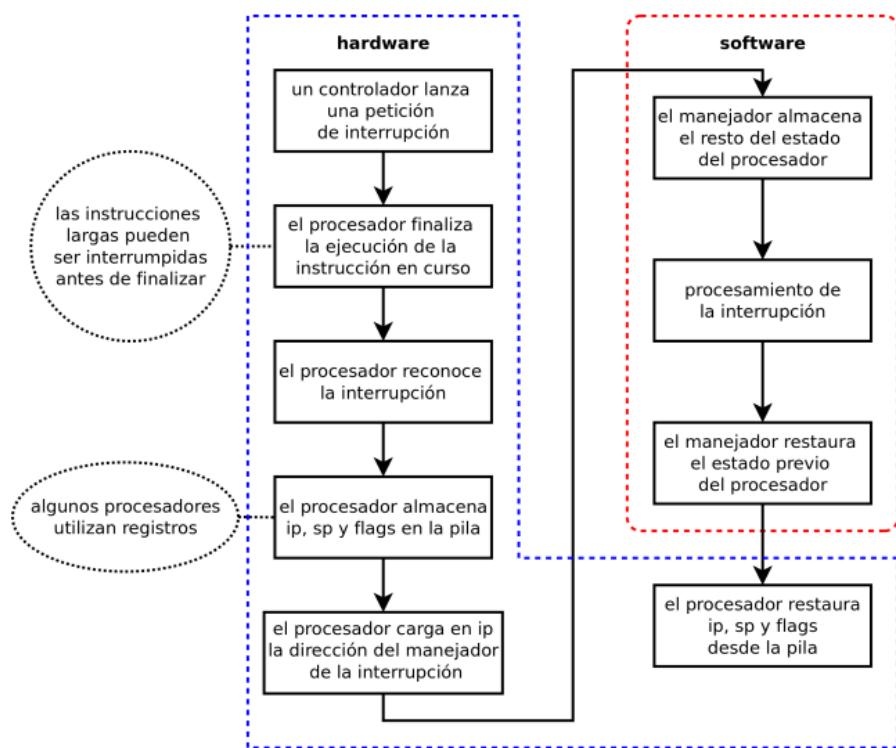
- **tas**: *test and set* (comprueba y establece)
- **cmpxchg**: comparar e intercambiar.
- **ll/sc**: carga enlazada y almacenamiento condicional.

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

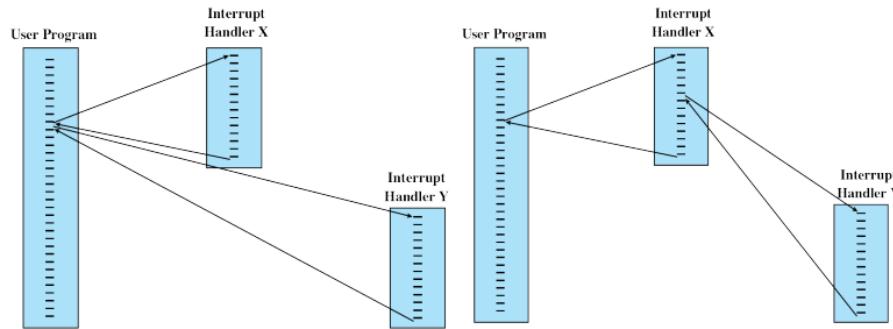


Figura: Procesamiento secuencial (interrupciones deshabilitadas). Figura: Procesamiento anidado (interrupciones con prioridades).

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.

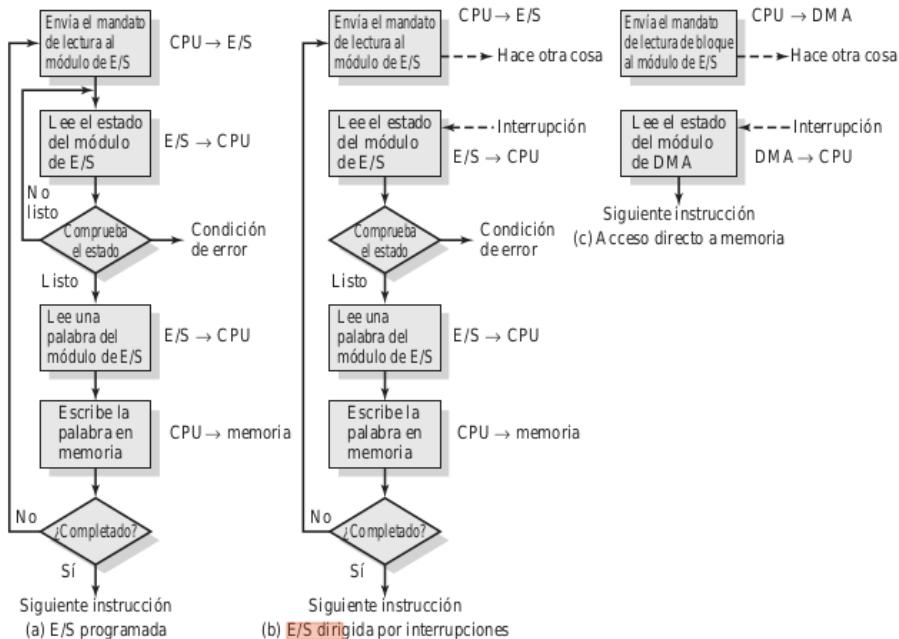
- **E/S dirigida mediante interrupción.** Es una alternativa a la E/S programada. En ella el procesador genera un mandato de E/S para un módulo y, acto seguido, continúa realizando algún otro trabajo útil. El módulo de E/S interrumpirá más tarde al procesador para solicitar su servicio cuando esté listo 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.

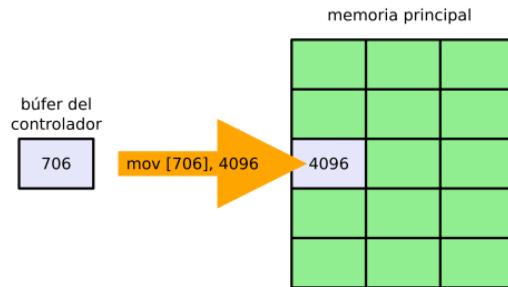
A continuación, el procesador continúa con otro trabajo. Ha delegado esta operación de E/S al módulo de DMA, que se ocupará de la misma. El módulo de DMA transferirá el bloque completo de datos, palabra a palabra, hacia la memoria o desde ella sin pasar a través del procesador. Por tanto, el procesador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



Direccionamiento físico de la memoria principal

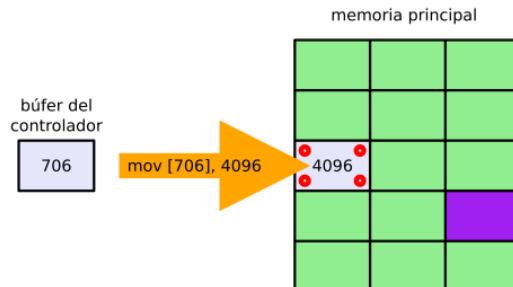
El direccionamiento virtual es algo que sólo existe en el interior del procesador. Para acceder a memoria los controladores de dispositivos y DMA sólo emplean direcciones físicas. Por ejemplo:

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

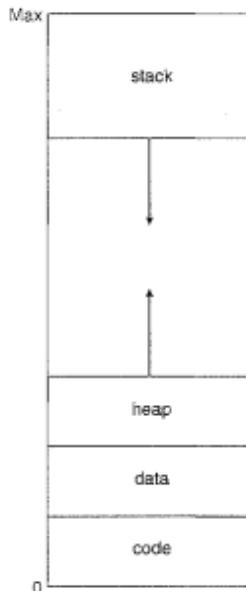


Figure 9.2 Virtual address space.

Procesos y/o hebras:

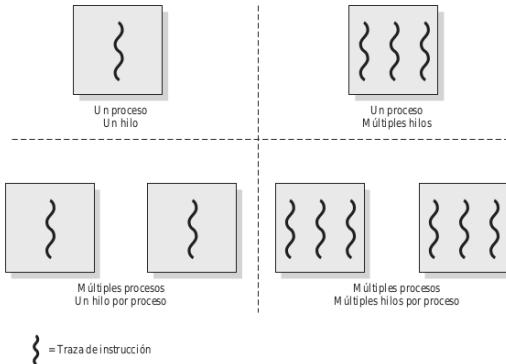
Se trata de una unidad de trabajo. Incluye el contexto del procesador (que contiene el contador del programa y el puntero de pila) y su propia área de datos para una pila (para posibilitar el salto a subrutinas). Un hilo se ejecuta secuencialmente y se puede interrumpir de forma que el procesador pueda 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 procesos 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 utilizando las mismas instrucciones de máquina que se utilizan para leer y escribir en otras partes de su espacio de memoria virtual.

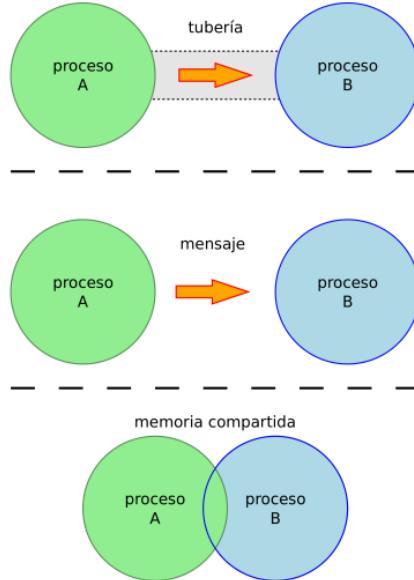
En algunos libros no se considera la tubería como un tipo de comunicación en sí mismo, ya que quería dentro de alguno de los otros dos.

Concurrencia y paralelismo:

Varios procesos pueden ejecutarse,

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

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 relativamente 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.

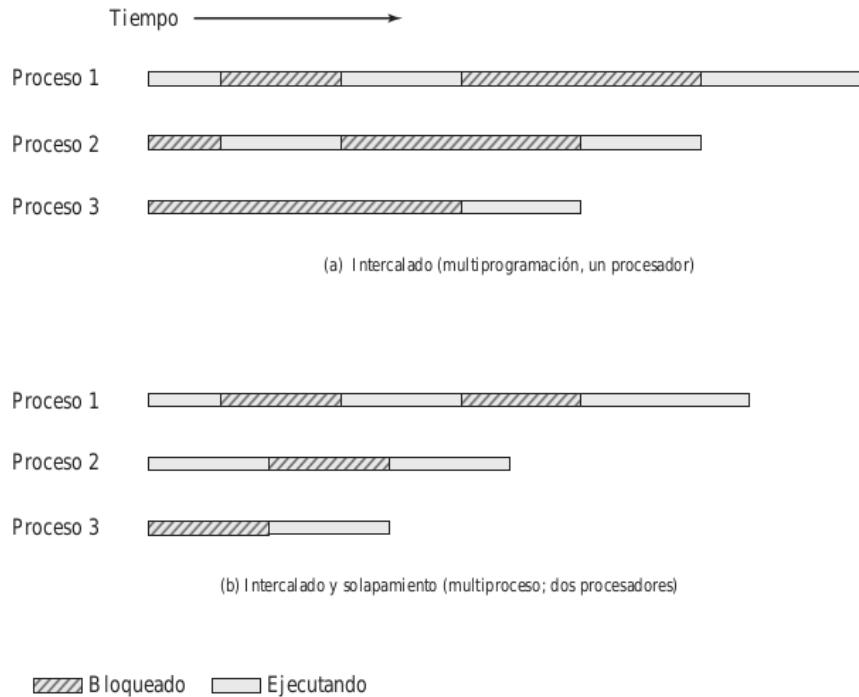
Podrán aparecer "condiciones de carrera" si no manejamos cuidadosamente la concurrencia, lo que implica la necesidad de métodos de sincronización de procesos y hebras.

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 relocalizable. 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 este 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.

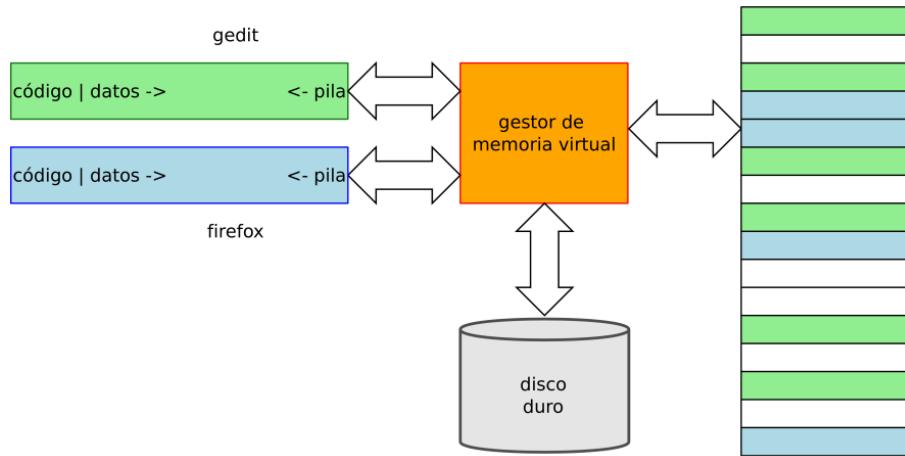
Memoria virtual: Permite al programador direccionar memoria de forma razonable en cuanto a

- 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 (control-

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/explusión
- Paginación \Leftrightarrow reemplazo.
- Interacción \Leftrightarrow comunicación.

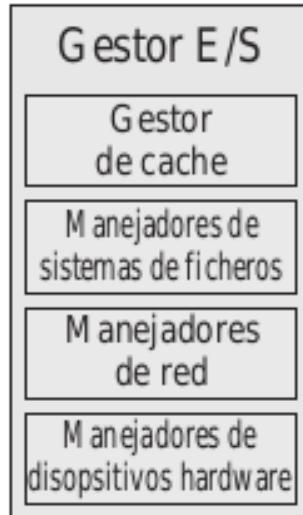
Gestión de E/S:

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 ⇒ 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

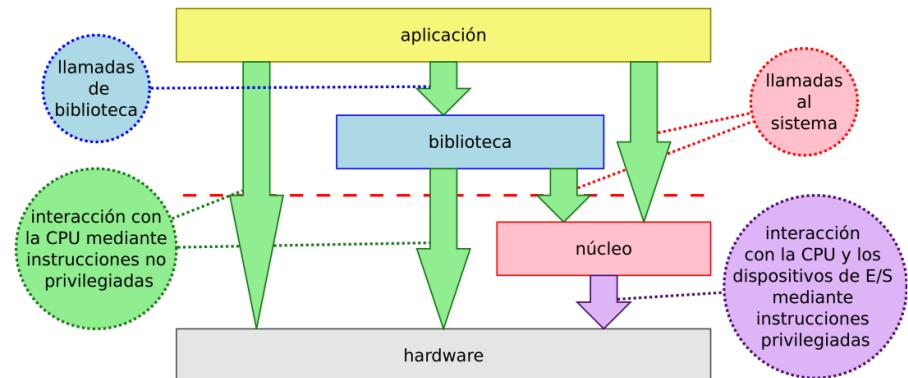
Implementación del almacenamiento persistente o a largo plazo. Las unidades almacenadas persistentemente son objetos como:

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

- Gestión de directorios

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

Algunos de los primeros ordenadores digitales eran binarios, algunos usaban tubos de vacío, otros eran programables, pero todo eran muy primitivos y tardaban segundos en realizar incluso los cálculos más simples.

En esta generación un pequeño grupo de gente (normalmente ingenieros) diseñaba, construía, programaba, operaba y mantenía cada máquina (genios como Aiken, von Neuman o Mauchley). Toda la programación se realizaba en lenguaje máquina o incluso peor, cableando circuitos eléctricos conectando miles de cables a un clavijero para controlar las funciones básicas de la máquina. Por ese entonces los lenguajes de programación eran inexistentes y los sistemas operativos desconocidos.

Se utilizaban para resolver problemas matemáticos y cálculos numéricos, tales como senos y logaritmos o computación de trayectorias de artillería.

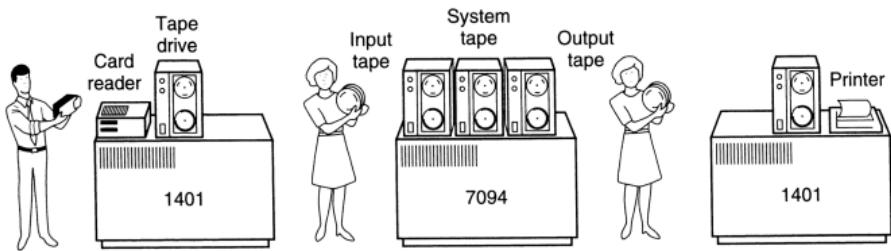
3.2.2 Segunda generación (1955-1965): Transistores y sistemas por lotes

La invención del transistor cambió radicalmente la situación. Los ordenadores se volvieron suficientemente confiables para ser manufacturados y vendidos a consumidores con la esperanza de que funcionen el suficiente tiempo para obtener la consecución algún trabajo útil. Se hace una separación entre diseñadores, constructores, operadores, programadores y personal de mantenimiento.

Estas máquinas se conocen actualmente como mainframes (IBM 1401, IBM 7094, la primera es una máquina para la escritura de cintas y la otra para el cálculo de operaciones). Para ejecutar una tarea el programador primero escribe el programa en papel (en FORTRAN o ensamblador), luego lo escribía en tarjetas, es decir, se programa en ensamblador o lenguajes de alto nivel.

Debido al alto coste del equipo se buscaron formas de reducir el tiempo desperdiaciado. La solución fueron los sistemas por lotes. La idea era recolectar una bandeja de tareas, escribirlas en una cinta magnética, llevarlas a la máquina de cálculo que hace los cálculos y los escribe en otra cinta magnética, y imprimir los resultados. Surge así el procesamiento por lotes.

Los ordenadores de segunda generación se utilizaron principalmente para cálculos cinéticos y de ingeniería, como resolver ecuaciones diferenciales parciales. Los sistemas operativos más típicos eran FMS (el Fortran Monitor System) y IBSYS, el sistema operativo de IBM para el 7094.



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

A principios de los 60, la mayoría de fabricantes de ordenadores tomaron dos líneas de producto distintas e incompatibles. La primera era la orientada a palabras y ordenadores científicos de gran escala como el 7094, los cuales se utilizaron para cálculos numéricos en ciencia e ingeniería a nivel industrial.

La segunda orientada a caracteres, ordenadores comerciales, como el 1401, los cuales eran ampliamente utilizados por bancos y compañías de seguros, en la ordenación de cintas y la escritura.

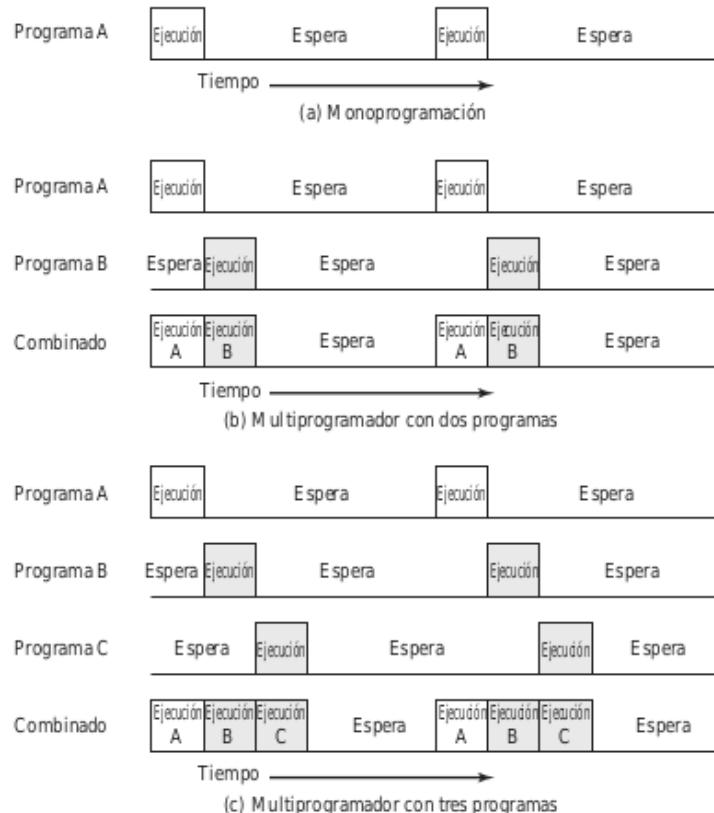
Se empiezan a utilizar circuitos integrados, así se pudo proveer una mayor relación precio/rendimiento respecto a las máquinas de la segunda generación, las cuales se basaban en transistores individuales (IBM 360, GE-645, DEC, PDP-1).

Como logro más destacables tenemos:

- Multiprogramación.
- Spooling (Simultaneous Peripheral Operation On Line). Habilidad de leer trabajos de tarjetas al disco tan pronto se traen a la sala de computación. Así, cuando el trabajo en ejecución termine, el sistema operativo puede cargar un nuevo trabajo desde el disco en la partición vacía y ejecutarlo.

- Tiempo compartido. Variación de la multiprogramación, en la cual cada usuario tiene un terminal online.

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



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

Con el desarrollo de circuitos LSI (Integración de Gran Escala) - chips con cientos de transistores por centímetro cuadrado de silicón - estalla la era el ordenador personal.

Surgen arquitecturas de procesadores como los 8080, Z80, 80x86, Alpha, Ul-traspac, ARM.

Como logros destacables tenemos:

- GUI (Graphic User Interface).
- SO de red. Los usuarios son conscientes de la existencia de múltiples ordenadores y pueden registrarse en cada máquina remota y copiar archivos de una máquina a otra.

- 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 cuales el núcleo proporciona la mayoría de las funcionalidades propias del sistema 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:
 - Micrónúcleo. Asigna unas pocas funciones esenciales al núcleo, incluyendo los espacios de almacenamiento, comunicación entre procesos y planificación básica.
 - Exónucleo.
- Máquina virtual.
- Híbridos

Tendencias:

- Núcleos Extensibles
- Multiservidores sobre un micrónúcleo.
- Núcleos híbridos.

Monolítico:

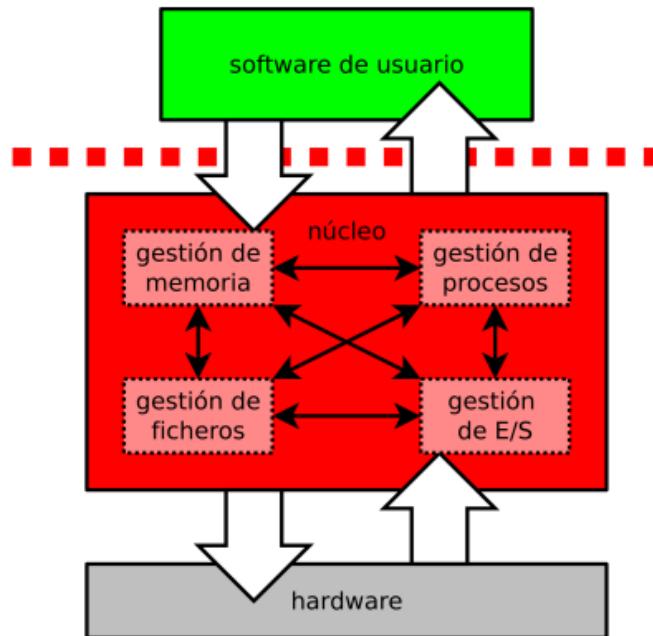
El Sistema operativo completo se ejecuta en modo protegido. Como consecuencia hay una protección nula entre los componentes del mismo, pues cualquier fallo puede afectar a cualquier parte del sistema operativo.

Ventajas:

- La capacidad de llamar a cualquier procedimiento que se quiera es muy eficiente.

Desventajas:

- La falta de protección implica una menor fiabilidad. Tener miles de procedimientos que pueden llamarse entre ellos sin restricción puede llevar a un sistema que es difícil de entender. Además, un fallo en cualquiera de esos procedimientos tumbaría el sistema operativo entero.
- Mal manejo de la complejidad. Es más sencillo escribir 1000 programas de 1000 líneas que uno de 1000000



Capas/Niveles:

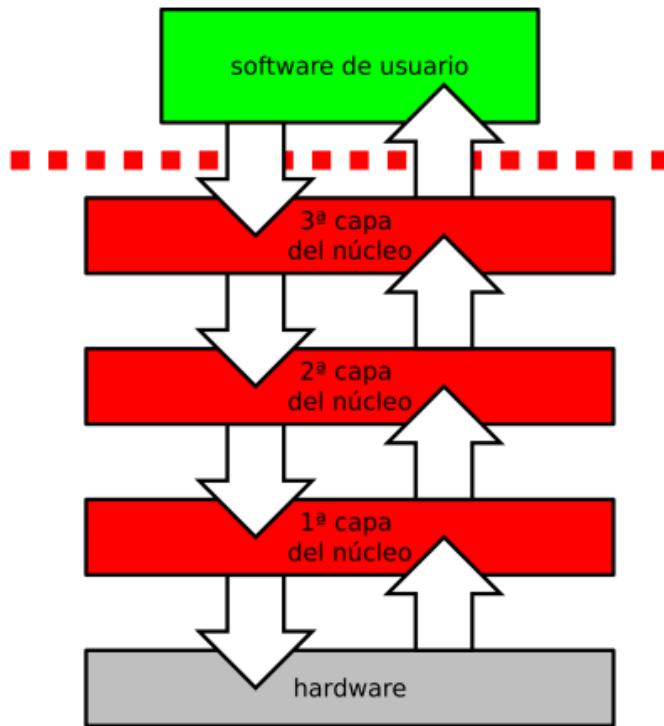
El Sistema Operativo completo se ejecuta en modo protegido. Hay una escasa división entre los componentes.

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:

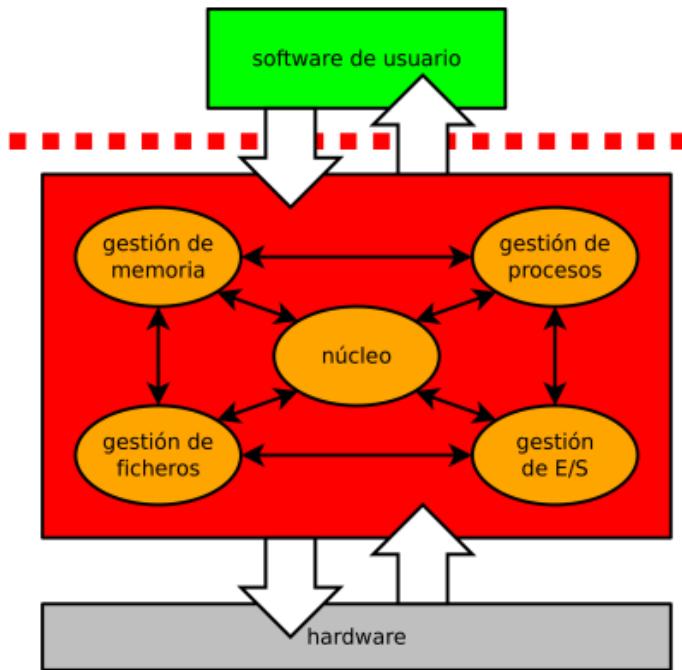
El Sistema Operativo se ejecuta en modo protegido. Una vez más esto implica una escasa protección entre los componentes.

Ventajas:

- De nuevo la econocomías de cambios de contexto lo hace eficiente.
- Tiene una menor complejidad.

Desventajas:

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



Micronúcleo/Microkernel

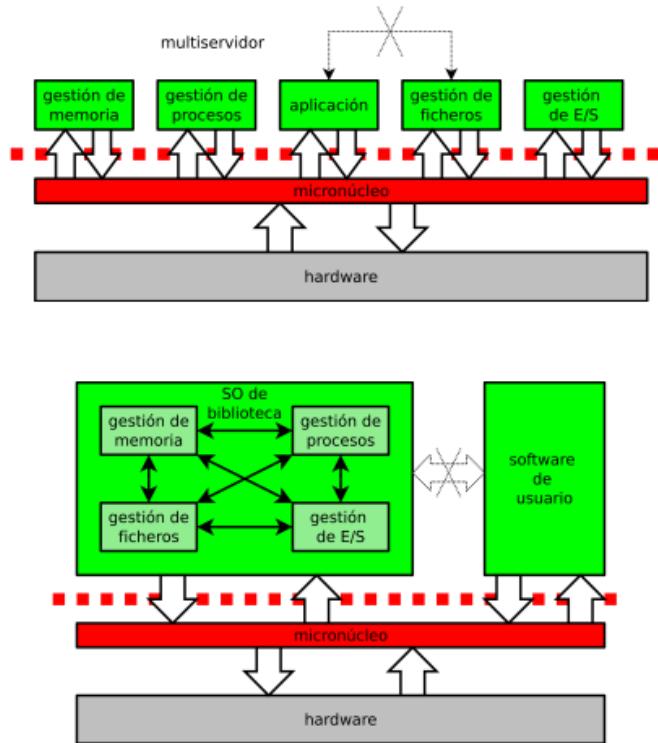
Una mínima parte del Sistema Operativo se ejecuta en modo protegido.

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:

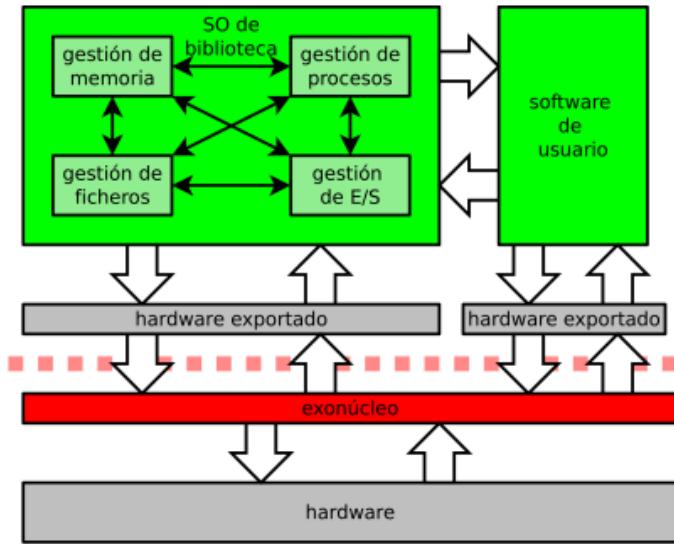
Apenas existe el sistema operativo, sólo un gestor de recursos. En este tipo de sistema operativo dejamos que el software accedan directamente al hardware.

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 rel:

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

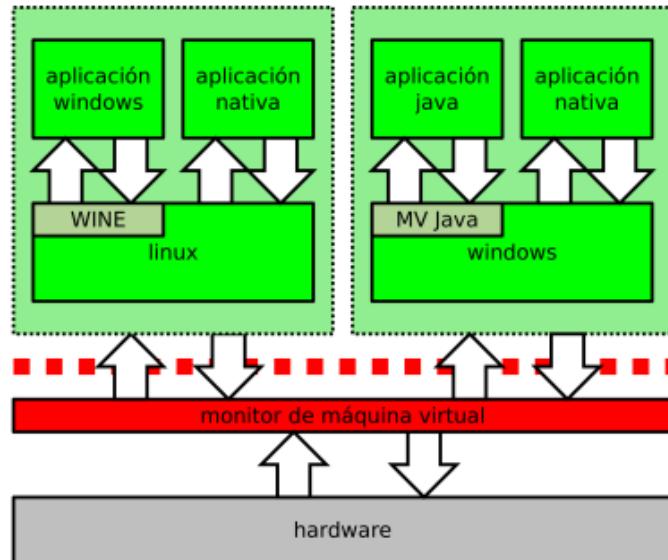
Ventajas:

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

Desventajas:

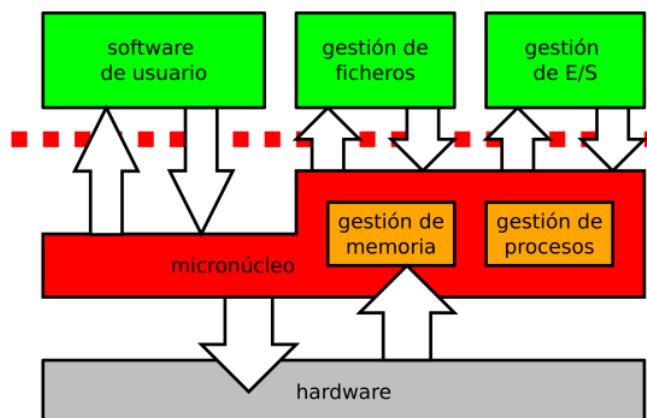
- La simulación del hardware real puede ser costosa, dando lugar a una baja eficiencia.

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



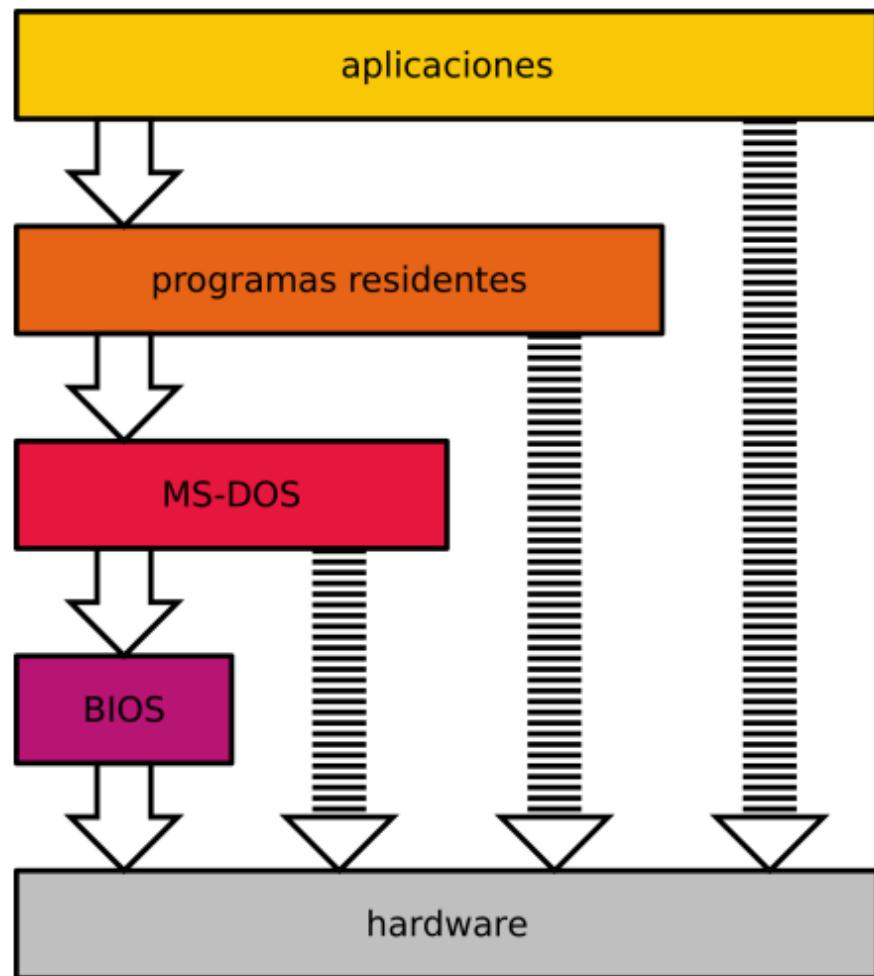
Híbrida

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

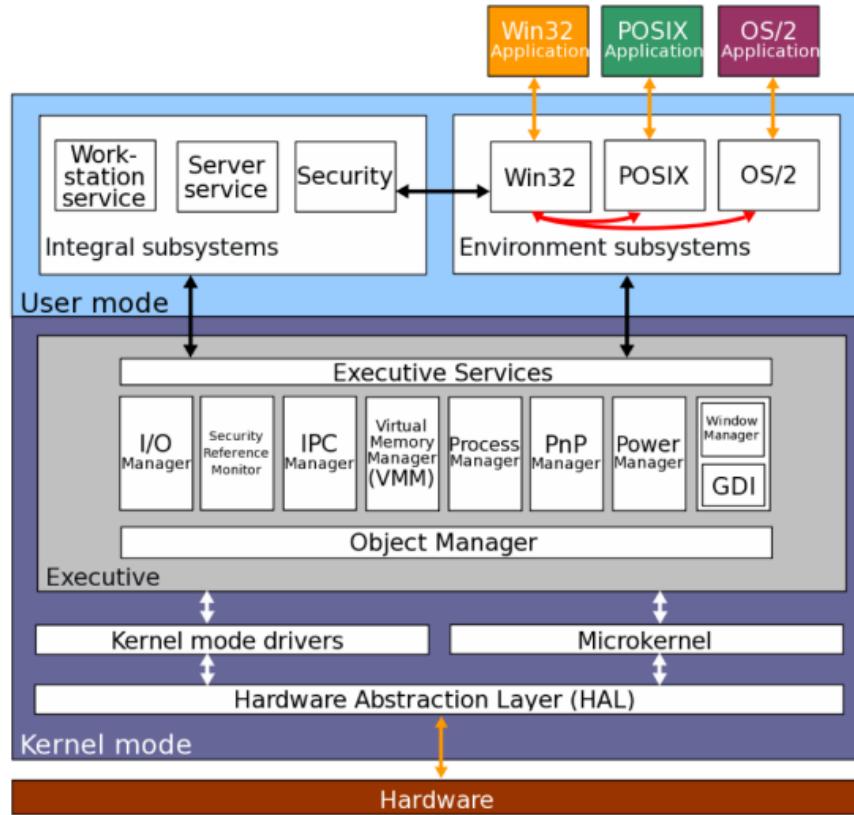


3.4 Ejemplos

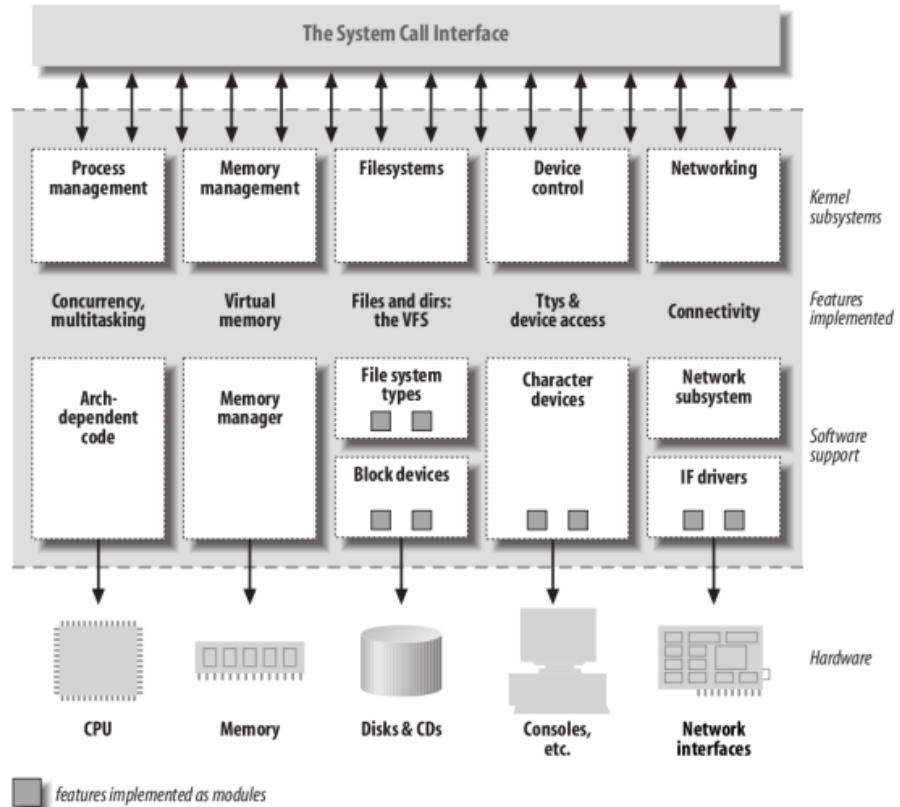
MS-DOS



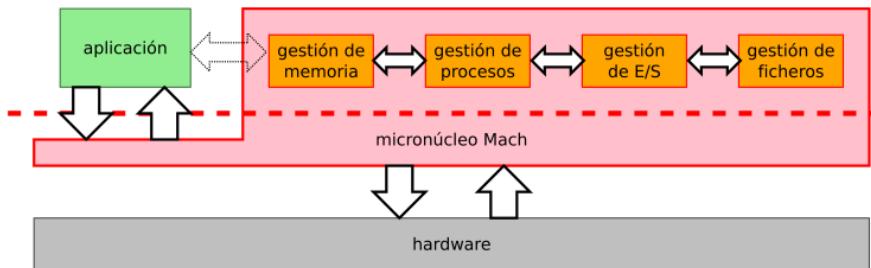
Windows 2000



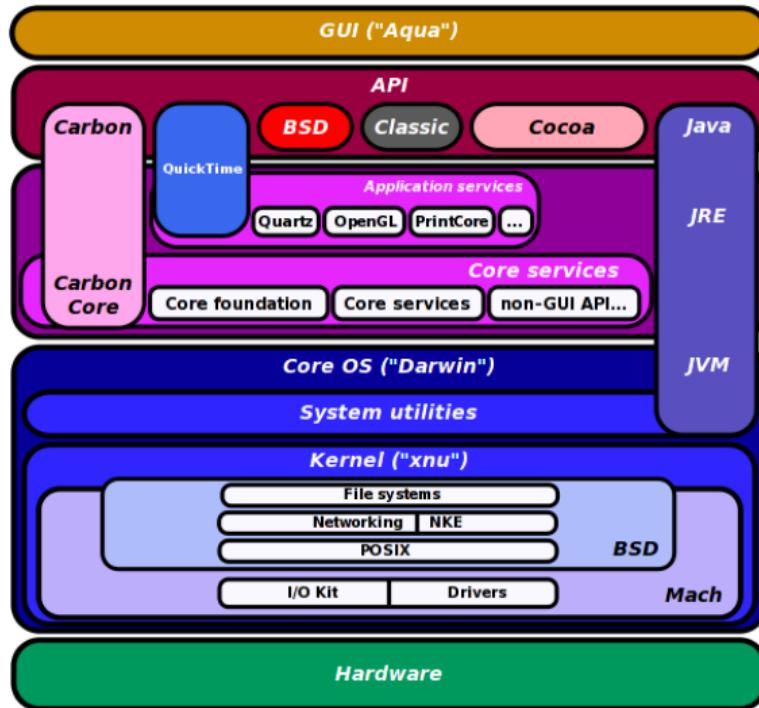
Linux



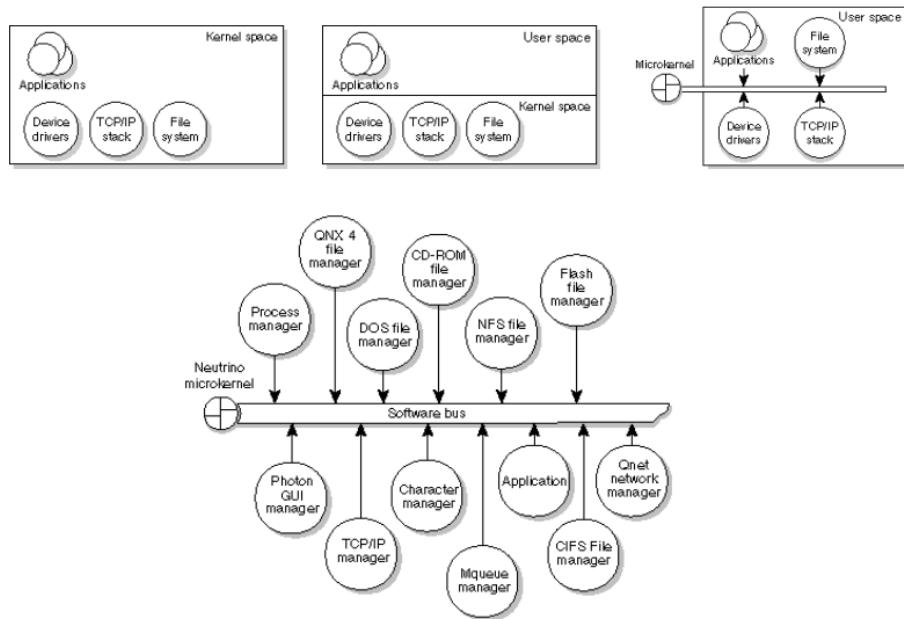
Mach



MacOS X



QNX



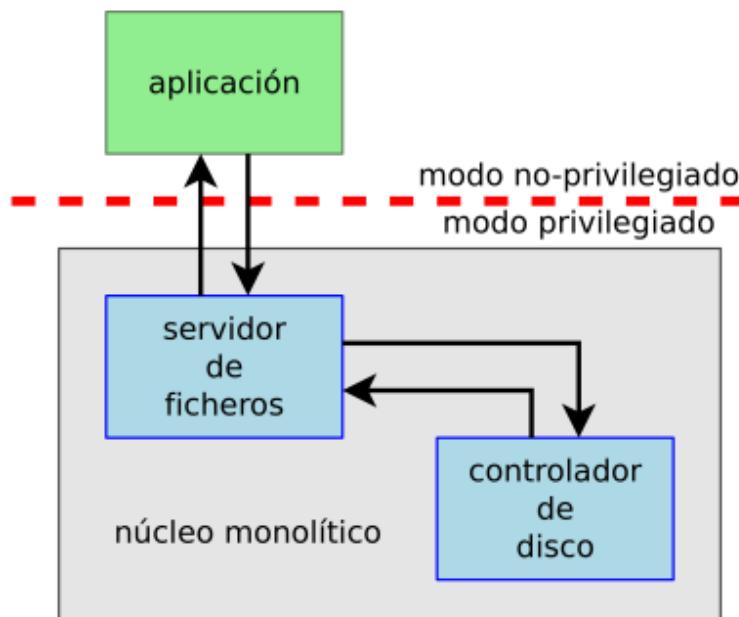
3.5 Comparativa

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.

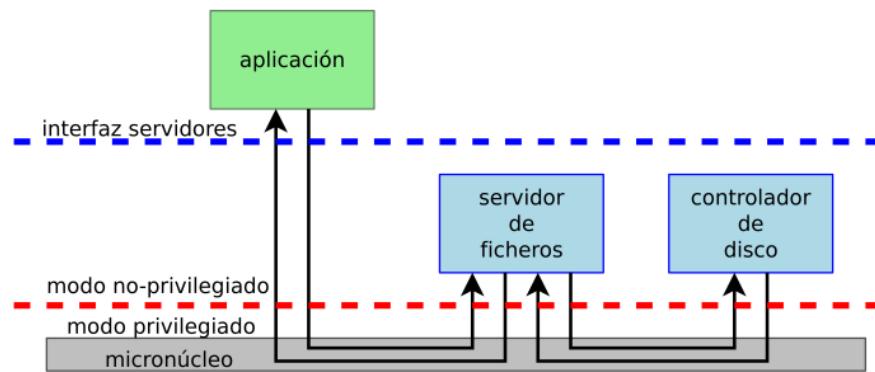
1 llamada a procedimiento: llamada y retorno en el interior del espacio de direcciones del núcleo y pudiendo compartir información.



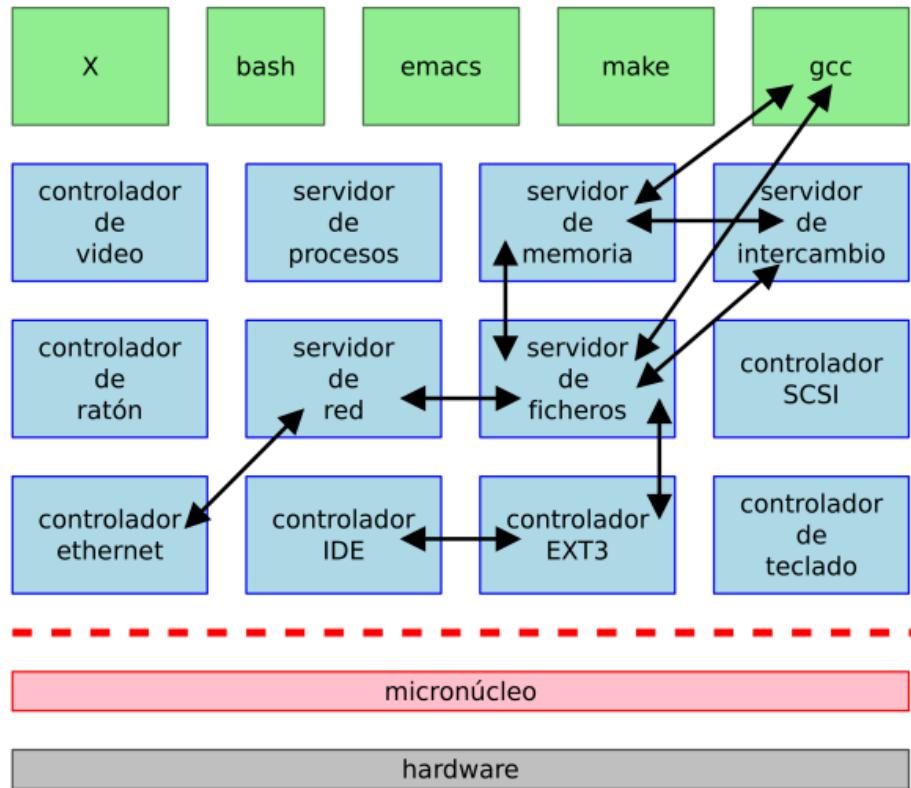
3.5.2 Coste estructural de una llamada al sistema: caso micronúcleo

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



4 Tema 4: Procesos

4.1 Definición

Proceso (definiciones):

- Programa en ejecución
- Entorno de protección
- Algo dinámico

Componentes básicos:

- Hebras/hilos de ejecución
- Espacio de direcciones.

La tarea fundamental de un SO es la gestión de procesos:

- Creación
- Planificación
- Comunicación
- Finalización

Un programa es

- Una lista de instrucciones
- Algo estático
- Varios procesos pueden ejecutar un mismo programa

Puede lanzar, o ser lanzado por, otros procesos.

Modelo de procesamiento:

Todo el software se organiza en forma de procesos, donde

$$proceso = programa + entorno \text{ (procesador + memoria)}$$

Objetivos:

- Multiprogramación: maximizar el uso del procesador, para mantener a este continuamente ocupado y así maximizar el rendimiento.
- Tiempo Compartido: cada proceso cree tener el sistema por completo. Cambian entre ellos con frecuencia en los momentos precisos para que así todos accedan a los recursos de forma equitativa.

Clasificación en función del coste del cambio de proceso:

- Procesamiento pesado: procesos UNIX
 - Hebra de actividad y espacio de direcciones unificado
 - El cambio de proceso implica dos cambios en el espacio de direcciones

$$ED_x \rightarrow ED_{SO} \rightarrow ED_y$$

- Procesamiento ligero: hebras tipo núcleo.
 - Hebra de actividad y espacio de direcciones desacoplados.
 - El cambio de hebra implica uno o dos cambios de espacio de direcciones en función de si las hebras comparten o no.

$$ED_x \rightarrow ED_{SO} \rightarrow ED_y$$

- Procesamiento superligero/pluma: hebras tipo usuario.
 - Hebras de actividad y espacio de direcciones unificados.
 - El cambio de hebra no implica cambio de espacio de direcciones.

$$ED_x \rightarrow ED_y$$

4.2 Control

Estructuras de control del SO Para gestionar procesos y recursos el SO debe disponer de información sobre estos. Para ello el SO mantiene tablas sobre cada elemento que gestiona:

- Tablas de memoria: principal y secundaria, protección, traducción. Se usan para mantener un registro tanto de la memoria principal (real) como de la secundaria (virtual). Parte de la memoria principal está reservada para el uso del SO, mientras que el resto está disponible para el uso de los procesos. Las tablas de memoria pueden contener la siguiente información:
 - Las reservas de memoria principal por parte de los procesos
 - Las reservas de memoria secundaria por parte de los procesos
 - Todos los atributos de protección que restringe el uso de la memoria principal y virtual, de forma que los procesos puedan acceder a ciertas áreas de memoria compartida.
 - La información necesaria para manejar la memoria virtual.
- Tablas de E/S: dispositivos y canales, estado de las operaciones. Información para gestionar los dispositivos de E/S y los canales del computador.
- Tablas de ficheros: existencia, atributos, localización,... Información sobre la existencia de ficheros, su posición en almacenamiento secundario, su estado actual y otros atributos.
- Tablas de procesos: localización y atributos.

Estas tablas no suelen estar separadas, sino entrelazadas. Normalmente estas se inicializan al arrancar el sistema.

Representación física de un proceso:

- Imagen del proceso:
 - Programa a ejecutar.
 - Espacio de direcciones disponible para código, datos, pila, ...
- Bloque de Control del Proceso (PCB) o descriptor de proceso:
 - Atributos para la gestión del proceso por parte del SO.
 - Es la estructura de datos más importante del SO.

Atributos de un proceso:

- Identificación del proceso: identificadores del proceso, proceso padre, usuario.
- Estado del procesador: registros de propósito general, de estado y control (PSW, *program status word*), puntero de pila.

Tabla 3.5. Elementos típicos de un bloque de control del proceso (BCP).

Identificación del proceso
Identificadores
Identificadores numéricos que se pueden guardar dentro del bloque de control del proceso:
<ul style="list-style-type: none"> • identificadores del proceso • identificador del proceso que creó a este proceso (proceso padre) • identificador del usuario
Información de estado del procesador
Registros visibles por el usuario
Un registro visible por el usuario es aquel al que se puede hacer referencia por medio del lenguaje máquina que ejecuta el procesador cuando está en modo usuario. Normalmente, existen de 8 a 32 de estos registros, aunque determinadas implementaciones RISC tienen más de 100.
Registros de estado y control
Hay una gran variedad de registros del procesador que se utilizan para el control de operaciones. Estos incluyen:
<ul style="list-style-type: none"> • <i>Contador de programa</i>: contiene la dirección de la siguiente instrucción a ejecutar. • <i>Códigos de condición</i>: resultan de la operación lógica o aritmética más reciente (por ejemplo, signo, cero, acarreo, igual, desbordamiento). • <i>Información de estado</i>: incluyen los <i>flags</i> de interrupciones habilitadas/deshabilitadas, modo ejecución.
Puntero de pila
Cada proceso tiene una o más pilas de sistema (LIFO) asociadas a él. Una pila se utiliza para almacenar los parámetros y las direcciones de retorno de los procedimientos y llamadas a sistema. Un puntero de pila apunta a la parte más alta de la pila.
Información de control de proceso
Información de estado y de planificación
Esta es la información que el sistema operativo necesita para analizar las funciones de planificación. Elementos típicos de esta información son:

- Información de control del proceso: estado, planificación, estructuración, comunicación y sincronización, privilegios, gestión de memoria, control de recursos y utilización.

Control de procesos

- Modos de ejecución. La mayor parte de los procesadores proporcionan al menos dos modos de ejecución:
 - Modo usuario: permite la ejecución de instrucciones que no afectan a otros procesos.
 - Modo núcleo: permite la ejecución de cualquier instrucción.

Existen otros modos intermedio que solían usarse en controladores de dispositivos y bibliotecas de lenguajes. Un bit en la palabra de estado indica en qué modo se está ejecutando el procesador.

- *Estado del proceso*: indica si está listo o no el proceso para ser planificado para su ejecución (por ejemplo, Ejecutando, Listo, Esperando, Detenido).
- *Prioridad*: uno o más campos que se pueden utilizar para escribir la prioridad de planificación del proceso. En algunos sistemas, se necesitan múltiples valores (por ejemplo, por-defecto, actual, mayor-disponible).
- *Información relativa a planificación*: esto dependerá del algoritmo de planificación utilizado. Por ejemplo, la cantidad de tiempo que el proceso estaba esperando y la cantidad de tiempo que el proceso ha ejecutado la última vez que estuvo corriendo.
- *Evento*: identificar el evento al cual el proceso está esperando para continuar su ejecución.

Estructuración de datos

Un proceso puede estar enlazado con otro proceso en una cola, anillo o con cualquier otra estructura. Por ejemplo, todos los procesos en estado de espera por un nivel de prioridad en particular pueden estar enlazados en una cola. Un proceso puede mostrar una relación padre-hijo (creador-creado) con otro proceso. El bloque de control del proceso puede contener punteros a otros procesos para dar soporte a estas estructuras.

Comunicación entre procesos

Se pueden asociar diferentes *flags*, señales, y mensajes relativos a la comunicación entre dos procesos independientes. Alguna o toda esta información se puede mantener en el bloque de control de proceso (BCP).

Privilegios de proceso

Los procesos adquieren privilegios de acuerdo con la memoria a la que van a acceder y los tipos de instrucciones que se pueden ejecutar. Adicionalmente, los privilegios se pueden utilizar para usar utilidades de sistema o servicios.

Gestión de memoria

Esta sección puede incluir punteros a **tablas** de segmentos y/o páginas que describen la memoria virtual asignada a este proceso.

Propia de recursos y utilización

Se deben indicar los recursos controlados por el proceso, por ejemplo, ficheros abiertos. También se puede incluir un histórico de utilización del procesador o de otros recursos; esta información puede ser necesaria para el planificador.

- El bit puede consultarse como el resto de la palabra de estado.
- Se modifica cuando se produce una llamada al sistema o una interrupción.
- Al retornar de una llamada al sistema o de una interrupción se devuelve el valor original de dicho bit desde la pila.
- Creación de procesos.
- Finalización de procesos.
- Jerarquía de procesos.
- Cambio de procesos.
- Ejecución del sistema operativo.

Funciones típica del núcleo de un SO:

Gestión de procesos	
• Creación y terminación de procesos	• Planificación y activación de procesos
Gestión memoria	
• Reserva de espacio direcciones para los procesos	• <i>Swapping</i>
Gestión E/S	
• Gestión de <i>buffers</i>	• Reserva de canales de E/S y de dispositivos para los procesos
Funciones de soporte	
• Gestión de interrupciones	• Auditoría
• Monitorización	

Creación de procesos.

Salvo sistemas extremadamente simples, todo SO tiene mecanismos para crear nuevos procesos. Posibles causas de la creación de un procesos:

1. Inicialización del sistema:
 - Interactivos / no interactivos
 - Primer /segundo plano
2. Llamada al sistema para crear un proceso.
 - fork() + exec()/CreateProcess()
3. Petición de usuario
 - Lanzamiento de una nueva aplicación desde el interfaz de usuario
4. Inicio de un proceso por lotes
 - Sistemas de colas de trabajos en servidores

Pasos en la creación de un proceso:

1. Asignar un identificador de proceso único. Se añade una nueva entrada a la tabla primaria de procesos, que contiene una entrada por proceso.

2. Reservar espacio para el proceso. El SO debe conocer cuánta memoria se requiere para el espacio de direcciones privado (programas y datos) y para la pila de usuario.
 - Estructuras de datos del SO (PCB).
 - Imagen del proceso.
3. Inicialización del bloque de control del proceso (PCB). Habitualmente se inicializa con la mayoría de entradas a 0, excepto el contador de programa (fijado en el puntero entrada del programa) y los punteros de pila de sistema (fijados para definir los límites de la pila del proceso). La información de control de procesos se inicializa en base a los valores por omisión, considerando también los atributos que han sido solicitados para este proceso. La prioridad se puede fijar, por defecto, a la más baja, a menos que una solicitud explícita la eleve a una prioridad mayor. Inicialmente, el proceso no debe poseer ningún recurso a menos que exista una indicación explícita de ello o haya sido heredado del padre.
 - ppid, estado, ip, sp, prioridad, E/S, ...
4. Establecimiento de enlaces adecuados:
 - Cola de trabajos
5. Creación o expansión de otras estructuras de datos:
 - Auditoría, monitorización, análisis de rendimiento, ...

Finalización de procesos:

Los procesos se crean, se ejecutan y se finalizan. Las causas de finalización de un proceso pueden ser:

- Voluntarias:
 - Terminación normal: la mayoría de los procesos realizan su trabajo y devuelven el control al SO mediante la llamada al sistema exit() / ExitProcess().
 - Terminación por error: falta argumento, ...
- Involuntarias:
 - Error fatal: instrucción privilegiada, excepción de coma flotante, violación de segmento, ...
 - Terminado por otro proceso: mediante la llamada al sistema kill() / TerminateProcess()

Una vez finalizado es necesario, auditoría/contabilidad del uso de recursos y recuperar/reciclar recursos.

Jerarquía de procesos:

UNIX:

- El uso de **fork()** crea una relación jerárquica entre procesos.
- **init/systemd** suele ser el primer proceso del sistema.
- La relación no se puede modificar.
- Si un proceso padre finaliza antes alguno de sus hijos estos pasan a depender del ancestro previo.
- Útil para llevar a cabo operaciones sobre grupos de procesos.

Windows:

- **CreateProcess()** no establece relación entre procesos.
 - Se crea un objeto que permite controlar al nuevo proceso.
 - Se puede traspasar la propiedad de este objeto.

Cambio de proceso:

Se trata de una operación costosa. Por ejemplo, en Linux 2.4.21 un cambio de proceso requiere $5.4 \mu s$ o 13200 ciclos en un Intel Pentium IV a 2.4GHz. Los eventos que pueden provocar un cambio de proceso son:

- Interrupción.
 - Interrupción del reloj. El SO determina si el proceso en ejecución ha excedido o no la unidad máxima de tiempo de ejecución, denominada rodaja de tiempo (time slice).
 - Finalización de operación de E/S o DMA. El SO determina qué acción de E/S ha ocurrido. Si la acción de E/S constituye un evento por el cuál están esperando uno o más procesos, el sistema operativo mueve todos los procesos correspondientes al estado de Listos. El SO puede decidir si reanuda la ejecución del procesos actualmente en estado Ejecutando o si lo expulsa para proceder con la ejecución de un proceso Listo de mayor prioridad.
- Excepción: fallo de página/segmento, llamada al sistema (int/syscall), operación de E/S, ...

Cambio de modo: cambio del modo de privilegio ocn el que se ejecuta el procesador. Es una operación sencilla y poco costosa.

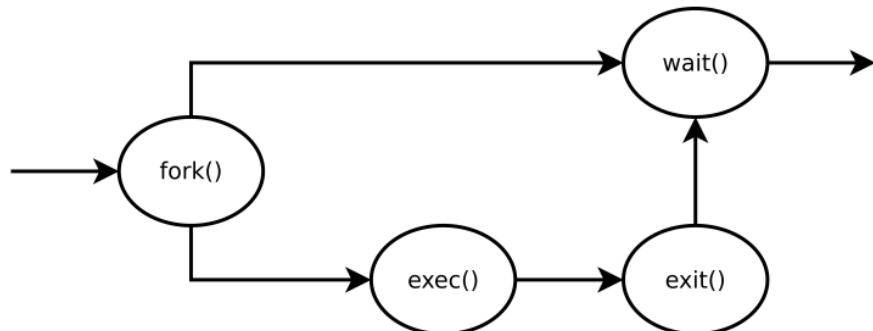
Cambio de contexto es ambiguo.

Pasos a seguir para realizar el cambio de proceso:

1. Salvar el estado del procesador, incluyendo el contador de programa y otros registros.
2. Actualizar el bloque de control del proceso que está actualmente en el estado de Ejecutando. Esto incluye cambiar el estado del proceso a uno de los otros estados. También se tienen que actualizar otros campos importantes, incluyendo la razón por la cual el proceso ha dejado el estado de Ejecutando y demás información de auditoría.
3. Mover el bloque de control de proceso a la cola apropiada.
4. Selección de un nuevo proceso a ejecutar,
5. Actualizar el bloque de control del proceso elegido. Esto incluye pasarlo al estado Ejecutando.
6. Actualizar las estructuras de datos de gestión de memoria. Estos se pueden necesitar, dependiendo de cómo se haga la traducción de direcciones.
7. Restaurar el estado del procesador al que tenía en el momento en el que el proceso seleccionado salió del estado Ejecutando por última vez, leyendo los valores anteriores de contador de programa y registros.

UNIX:fork() + exec() + wait() + exit()

- fork(): crea un nuevo proceso.
- exec(): cambia la imagen de un proceso
- wait(): permite al padre esperar al hijo
- exit(): finaliza el proceso



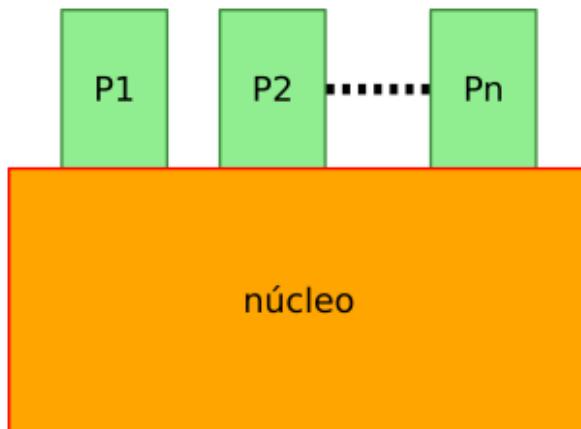
4.3 Ejecución del Sistema Operativo

4.3.1 Núcleo independiente

Es el método más antiguo. El SO no es un proceso (aunque se comporte como uno). Además, dispone de áreas de memoria y pila propias. El sistema operativo puede realizar todas las funciones que necesite y restaurar el contexto del proceso interrumpido, que hace que se retome la ejecución del proceso de usuario afectado. De forma alternativa, el sistema operativo puede realizar la salvaguarda del contexto y la activación de otro proceso diferente. Si esto ocurre o no depende de la causa de la interrupción y de las circunstancias puntuales en el momento.

El concepto de proceso es aplicable sólo a programas de usuario. El código del SO se ejecuta como una entidad independiente que requiere un modo privilegiado de ejecución.

El inconveniente es que cada evento cuesta un cambio de proceso y modo.

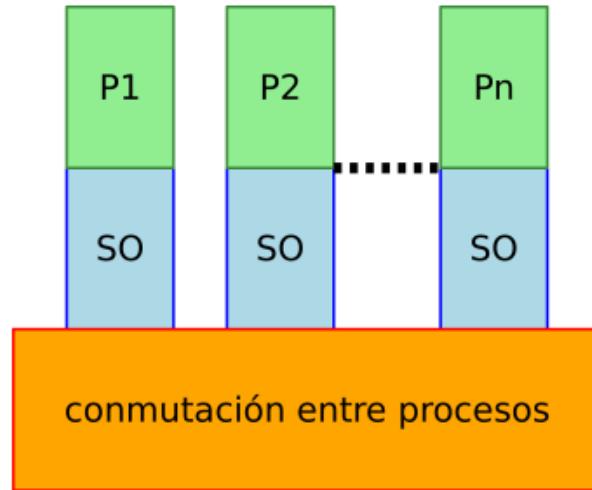


4.3.2 Ejecución dentro de los procesos de usuario

El SO parece un conjunto de subrutinas que el proceso puede invocar desde su espacio de direcciones. A la imagen de cada proceso se une la del SO. Cuando ocurre una interrupción, *trap* o llamada al sistema, el procesador se pone en modo núcleo y el control se pasa al sistema operativo. Para este fin, el contexto se salva y se cambia de modo a una rutina del sistema operativo. Sin embargo, la ejecución continúa dentro del proceso de usuario actual. De esta forma, no se realiza un cambio de proceso, sino un cambio de modo dentro del mismo proceso.

Ventaja: cada evento cuesta sólo un cambio de modo.

Inconveniente: restamos espacio al proceso de usuario.



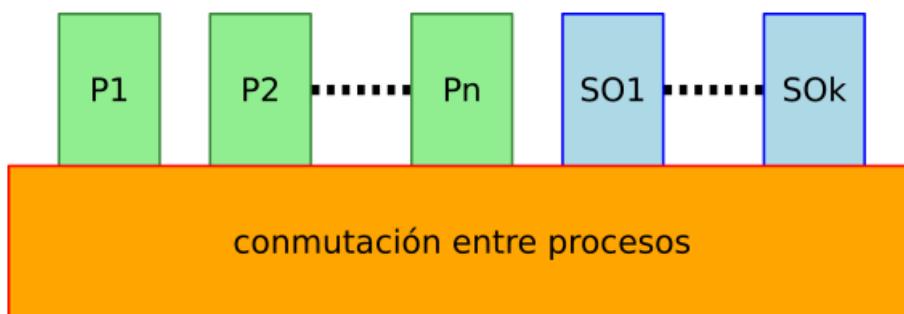
4.3.3 Sistemas operativos basados en procesos

El SO se implementa como una colección de procesos. Debe haber una pequeña cantidad de código para intercambio de procesos que se ejecuta fuera de todos los procesos.

Ventajas:

- Modularidad y facilidad de programación
- Teórica mejora de rendimiento en sistemas multiprocesador.

Inconvenientes: un evento puede costar varios cambios de proceso y modo.



Traza de un proceso.

Comportamiento de un proceso = lista de instrucciones que ejecuta, a la que denominaremos traza.

Activador (dispatcher): programa encargado de cambiar entre los PCBs de los procesos para ejecutar un proceso u otro.

Cola de procesos:

- Creación de un proceso = crear PCB + cargar imagen
- Lista de procesos = lista de PCBs.
- Planificador (*"scheduler"*):
 - Parte del SO que escoge el siguiente proceso a ejecutar.
 - Gestor de las colas de planificación.
- Activador (*"dispatcher"*): parte del planificador que realiza un intercambio de procesos
- Ejecución = encolar + activar.

Modelo de 2 estados

Estados:

- Ejecutando: proceso en ejecución
- No ejecutando: proceso que no se está ejecutando.

Transiciones:

- Ejecutando → no ejecutando: evento o temporización
- No ejecutando → ejecutando: temporización o fin de evento.

Inconvenientes:

- No permite discriminar fácilmente la razón por la que un proceso no se encuentra en ejecución.
- Solución: subdividir el estado "no ejecutando" para reflejar el motivo.

Modelo de 5 estados

Estados:

- Nuevo: el proceso ha sido creado
- Preparado: proceso a la espera de que se le asigne un procesador.
- Ejecutando: proceso actualmente en ejecución.
- Bloqueado: proceso que no puede continuar hasta que finalice un evento.
- Finalizado: proceso finalizado.

Transiciones:

- Nuevo → preparado: se admite un nuevo proceso en el sistema.
- Preparado → ejecutando: el planificador selecciona el proceso para su ejecución.

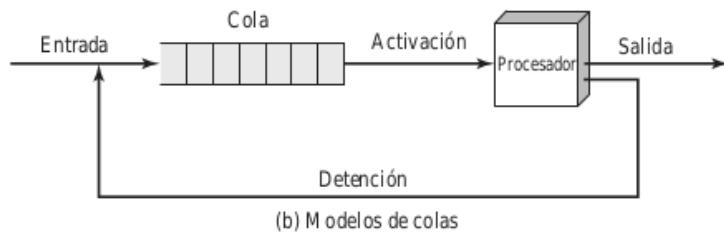


Figura 3.5. Modelo de proceso de dos estados.

- Preparado → finalizado: padre termina hijo.
- Ejecutando → finalizado: proceso finalizado.
- Ejecutando → preparado: tiempo de procesador agotado.
- Ejecutando → bloqueado: se produce un evento.
- Bloqueado → preparado: finalización de evento.
- Bloqueado → finalizado: padre termina hijo.



Figura 3.6. Modelo de proceso de cinco estados.

Modelo de +5 estados

Añaden un nuevo estado, el estado suspendido. El objetivo de la multiprogramación es aprovechar al máximo el procesador por la lentitud de la E/S. Esto no resuelve el problema del modelo de 5 estados.

- Causa: diferencia de velocidad entre CPU y E/S.
- Todos los procesos podrían llegar a estar bloqueados en espera de finalización de un evento.
- Solución: añadir más procesos.
- Problema: falta de memoria.

Se trata de un círculo vicioso difícil de romper:

- Solución cara: añadir más memoria
- Solución barata: memoria de intercambio ("swap")

Intercambio ("swapping"): proceso de expulsión de un proceso de memoria principal a secundaria y viceversa. Se agrava el problema, pues el intercambio requiere E/S.

Modelos de 6 estados



Modelo de 7 estados



Motivación: el reactivar un proceso sólo para descubrir que sigue bloqueado es muy costoso.

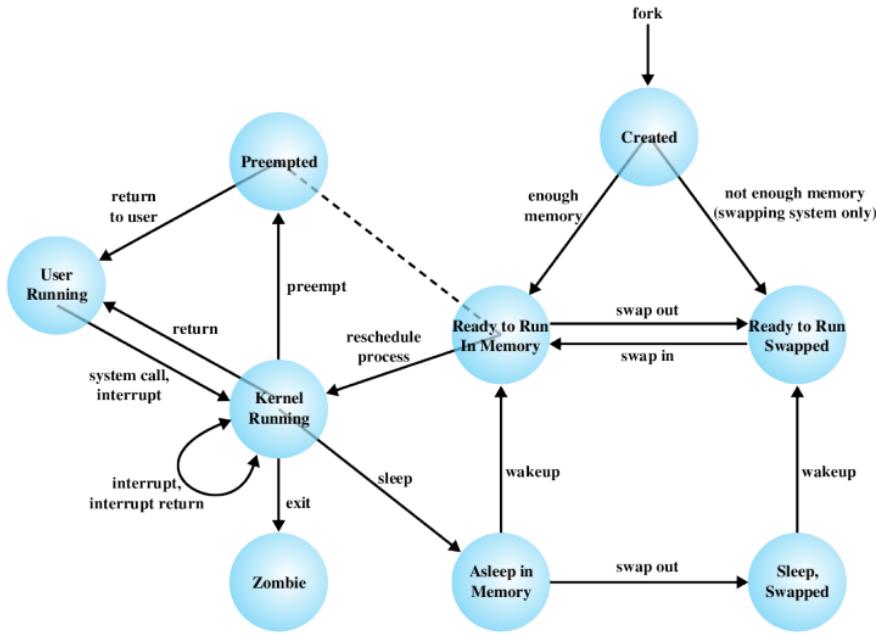
Nuevos estados:

- **Suspendido (bloqueado):** proceso en área de intercambio y esperando un evento.
- **Suspendido (preparado):** proceso en área de intercambio y a la espera de espacio en memoria principal.

Nuevas transiciones:

- **Bloqueado → suspendido (bloqueado):** no hay procesos preparados o estos consumen demasiada memoria.
- **Suspendido (bloqueado) → suspendido (preparado):** sucede el evento por el que estaba bloqueado.
- **Suspendido (preparado) → preparado:** no quedan procesos preparados o tiene mayor prioridad que los preparados.
- **Preparado → suspendido (preparado):** liberar memoria o dejar sitio para un proceso bloqueado de mayor prioridad.
- **Nuevo → suspendido (preparado):** control de carga del sistema.
- **Suspendido (bloqueado) → bloqueado:** queda memoria libre o proceso de alta prioridad.
- **Ejecutando → suspendido (preparado):** un proceso agota su tiempo y hay que liberar memoria para un proceso suspendido de mayor prioridad.
- **X → finalizado:** un proceso elimina a otro.

Diagrama de transiciones entre estados en UNIX



Planificación

Los procesos pueden cambiar varias veces de cola de planificación a lo largo de su vida. La parte del SO encargada de realizar esos cambios es el planificador.

Tipos de planificadores:

- Corto plazo: selecciona entre los procesos preparados para ejecutar.
 - Ejecución muy frecuentemente, ej: cada 10,...,100ms.
- Medio plazo: decide que procesos pasar al área de intercambio y así controla el grado de multiprogramación.
- Largo plazo: selecciona que procesos poner en ejecución, ej: sistema por lotes.
 - Ejecución en función de la carga del sistema, cada varios minutos o cuando finaliza un proceso.

Comunicación entre procesos

Los procesos pueden ser:

- Independientes: no afecta ni es afectado por otros procesos (no comparten datos).

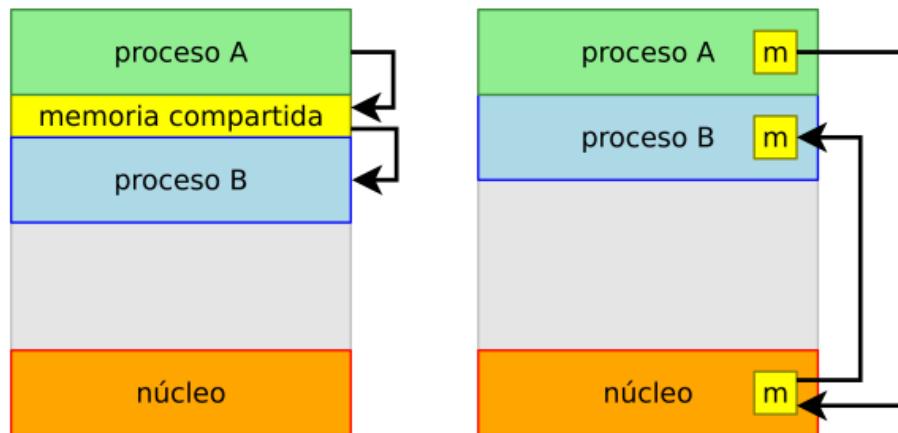
- Cooperatnes: puede afectar y ser afectados por otros procesos (si comparten datos).

El SO debe proporcionar métodos para crear, comunicar y terminar procesos.
Los motivos para cooperar son:

- Compartir información: capacidad de acceso y economía de recursos.
- Acelerar los cálculos: realizar las tareas más rápidamente.
- Modularidad: facilidad de creación de programas.
- Conveniencia: multitarea

Métodos de comunicación (ya explicado):

- Memoria compartida.
 - Los procesos comparten un área de memoria
 - La comunicación es responsabilidad de los procesos.
- Paso de mensajes.
 - Los procesos intercambian mensajes.
 - Comunicación responsabilidad del sistema operativo.



5 Tema 5: Hebras

5.1 Definición

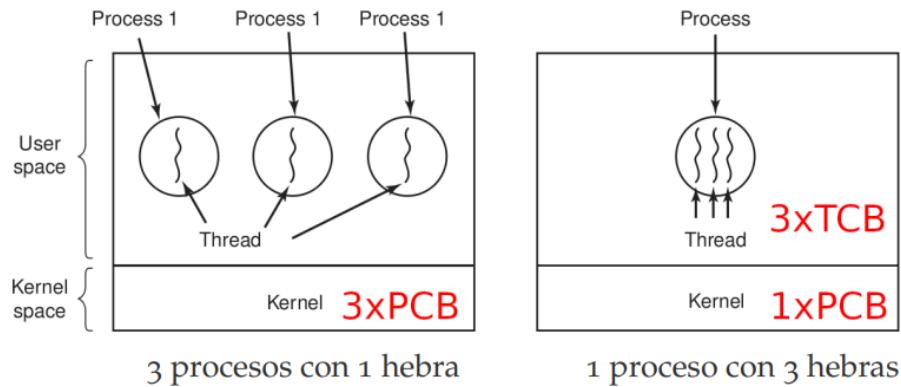
- Proceso en un SO "tradicional" = espacio de direcciones + hebra de control.

- Proceso = contenedor de recursos + hebra de ejecución.
- Una hebra se ejecuta dentro de un proceso.
- Proceso y hebra son conceptos diferentes y suelen ser tratados por separado.
 - La función de un proceso es agrupar recursos.
 - Las hebras son entidades planificables para su ejecución sobre un procesador.
- Las hebras añaden a los procesos la capacidad de ejecutar varios conjuntos de instrucciones de forma concurrente dentro de un mismo entorno de procesamiento.
- Ejecutar varias hebras en paralelo es parecido a ejecutar varios procesos salvo porque comparten sus recursos.
- También se denominan procesos ligeros o subprocesos.

5.2 Comparativo Procesos/Hebras

Dos formas de ejecutar 3 hebras:

- En un caso necesitamos 3 PCB y en el otro 1.
- En un caso necesitamos 0 TCB y en el otro 3.



5.3 Ejecución multihebra

Se conoce como multihebra a la ejecución concurrente de hebras en el interior de un proceso.

- En un sistema con un único procesador las hebras se alternan en el uso del procesador.

- En un sistema multiprocesador tantas hebras como procesadores pueden ejecutarse en paralelo.

Ventajas de la multihebra

- Sensibilidad. Hacer una aplicación interactiva multihilo puede permitir a un programa correr incluso si parte del mismo está bloqueado o realizando una operación larga, incrementando así la respuesta dada al usuario.
- Compartición de recursos: los procesos solo comparten recursos a través de técnicas como memoria compartida o paso de mensajes, las cuales se tienen que declarar explícitamente por el programador. Las hebras comparten memoria y recursos del proceso al que pertenecen por defecto.
- Economía: reservar memoria y recursos para la creación de un proceso es costoso. Ya que las hebras comparte los recursos del proceso al que pertenecen es más económico crear y cambiar de contexto hebras.
- Escalabilidad: los beneficios de la programación multihilo pueden verse gratamente incrementados por una arquitectura multiprocesador, donde las hebras pueden correr en paralelo en distintos procesadores.

Ejemplos de uso:

- Trabajo simultáneo en primer y segundo plano (ej: procesador de texto, servidor de ficheros, ...)
- Procesamiento asíncrono (ej: copia de seguridad)
- Velocidad de ejecución: descomposición de problemas y procesamiento paralelo (ej: solapamiento entre obtención de datos y cálculo).
- Mejor modularidad: fácil subdivisión de tareas complejas en otras más simples.

Las diferentes hebras de un proceso no son independientes como las que se encuentran en diferentes procesos. Esto se debe a que las hebras de un proceso comparten sus recursos: espacio de direcciones, variables globales, ficheros abiertos, ...

No hay protección entre hebras ya que:

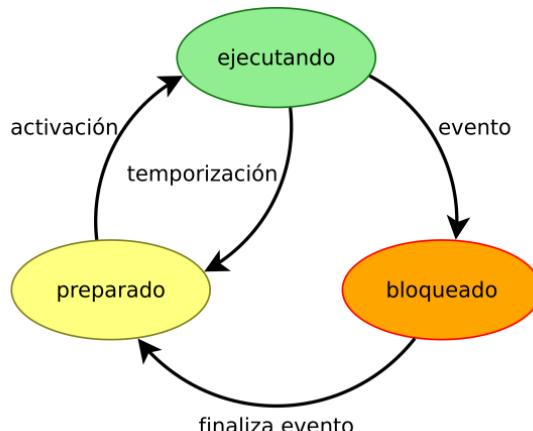
- Es imposible, pues comparten recursos.
- No es necesario, pues diferentes procesos pueden pertenecer a diferentes usuarios, en cambio, todas las hebras de un proceso pertenecen al mismo.

Recursos necesarios

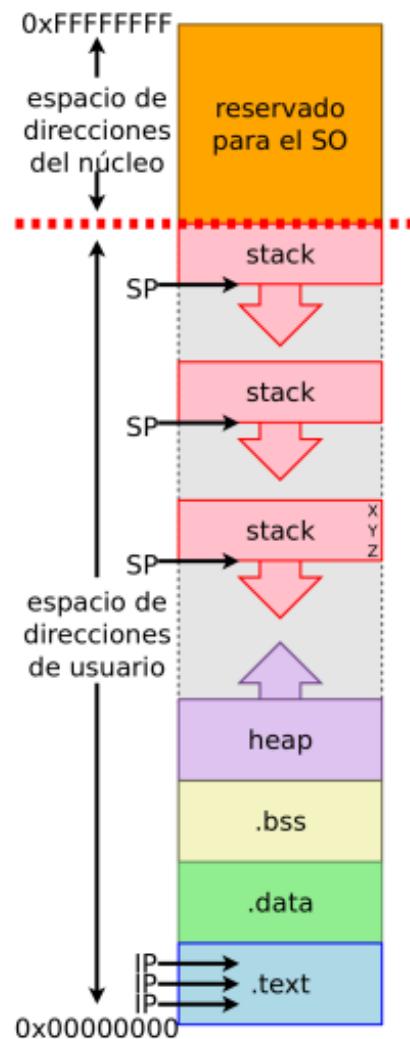
por proceso	por hebra
identificador de proceso espacio de direcciones variables globales ficheros abiertos procesos hijos alarmas pendientes señales y manejadores de señales información contable ...	identificador de hebra registros pila estado datos privados

Estado de una hebra

Al igual que los procesos, una hebra puede pasar por diferentes estados a lo largo de su ejecución.



Espacio de direcciones Cada hebra necesita su propia pila. Cada pila contiene un marco de función por cada función llamada y de la que todavía no se ha retornado. Si en una hebra ejecutamos el procedimiento X, este llama al Y, y este llama al Z, durante la ejecución de Z la pila contendrá los marcos de X, Y y Z.



Gestión de hebras Para gestionar las hebras, al igual que sucede con los procesos, necesitamos una serie de llamadas al sistema. Los procesos multihebra comienzan su ejecución con una única hebra y crean más en caso de necesitarlas:

```
pthread_create()/std::thread::thread()
```

Las hebras pueden mantener una relación jerárquica o ser creadas todas iguales. Cada hebra suele tener un identificador único, recogido en el TCB. Es necesario una llamada al sistema para terminar con una hebra

```
pthread_exit()/std::thread::~thread()
```

Al igual que los procesos, podemos hacer que las hebras sincronicen su funcionamiento:

```
pthread_join()/std::thread::join()
```

Otra habitual es

```
thread_yield()/std::this_thread::yield()
```

, permite a una hebra ceder voluntariamente el procesador.

- Muy importante porque puede que la interrupción de reloj no las afecte o no se pueda imponer el tiempo compartido.
- Las hebras deben cooperar.

Las hebras introducen nuevos problemas en el sistema, cuya mejor solución es no duplicar hebras.

Tipos de hebras

- Hebras tipo usuario: En las hebras de tipo usuario el parquete de hebras se coloca enteramente en el espacio de usuario. El núcleo no tiene información sobre las misma, y lo que a él respecta está gestionando procesos ordinarios. La primera y principal ventaja es que la hebras a nivel de usuario se pueden implementar en sistemas operativos que no soporan hebras. Todos los sistemas operativos solían caer en esta categoría, y algunos lo siguen haciendo. Con esta aproximación las hebras se implementan por medio de librerías.

Cada hebra tiene su propia tabla de hebras privada para mantener un registro de las hebras en el proceso. Esta tabla es análoga a la tabla de procesos del kernel, con la única diferencia de que solo registra información de las propiedades de las hebras, como el contador de programa, registro de pila, registros, estado, etc. Esta tabla es gestionada por el sistema de tiempo de ejecución.

Las principales ventajas de las hebras de tipo usuario son:

- Se puede implementar en cualquier SO.
- La commutación entre hebras es varios órdenes más rápida que la de procesos.
- Cuando una hebra termina de ejecutarse (pthread_exit()/pthread_yield()), puede cambiarse sin involucrar al núcleo con lo que ahorramos cambios de procesos y modo.
- Cada proceso puede utilizar un algoritmo de planificación personalizado.

Los principales inconvenientes de las hebras de tipo usuario son:

- Como se implementan las llamadas al sistema bloqueantes. Si una hebra hace una llamada al sistema detendrá a todas las hebras. Como soluciones a este problema se tienen:
 - * Modificar el SO para hacer llamadas no lboqueantes. No tiene sentido, pues cambiar el SO no es atractivo, además se supone que una de las ventajas es que es implementable en cualquier SO. Además, cambiar las semánticas de estas requeriría demasiados programas de usuario.
 - * Otra alternativa es comprobar si las llamadas bloqueantes tendrán éxito antes de ejecutarlas. Este método es ineficiente y poco elegantes, pero no quedan muchas opciones.
- Como no existe interrupción de reloj no podremos ejecutar una hebra hasta que otra deje libre el procesador voluntariamente. La solución sería que el sistema de ejecución mande una interrupción periódicamente para cambiar el control, pero es demasiado liso de programar. Además el overhead sería bastante sustancial.
- En sistemas multiprocesador no podremos asignar más de un procesador por proceso.
- Hebras tipo núcleo. El núcleo conoce la existencia de las hebras y las gestiona. No se requiere un sistema de gestión de hebra ni tablas de hebras en cada proceso. En su lugar el kernel mantiene una tabla de hebras que mantiene un historial de todas las hebras del sistema. La gestión de las hebras se hace por medio de llamadas al sistema, lo que implica un costo considerablemente mayor.

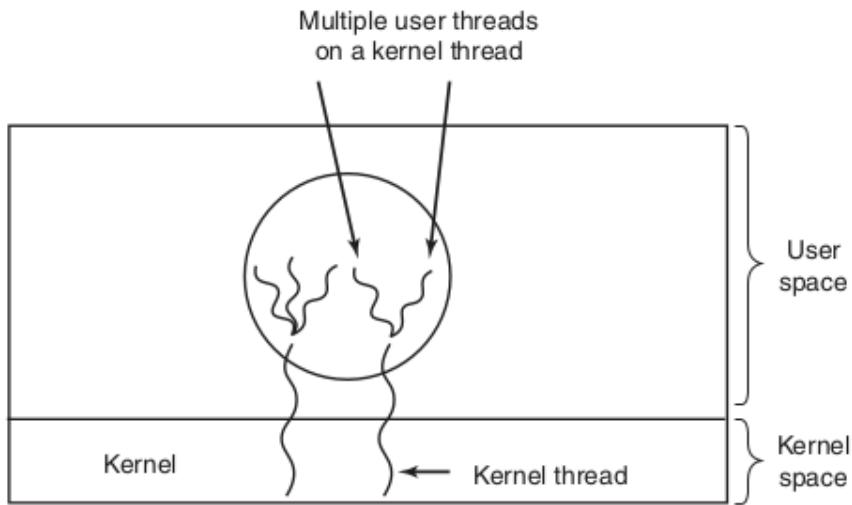
Entre las ventajas de las hebras encontramos:

- No requieren llamadas al sistema no bloqueantes nuevas.
- Si una hebra es blqoeada o excede su tiempo de ejecución otra puede ser ejecutada.
- En sistemas multiprocesador tantas hebras de un mismo proceso como procesadores existan se pueden ejecutar en paralelo.

Entre los inconvenientes encontramos:

- El coste de las llamadas al sistema es sustancial, de forma que las operaciones con hebras implicarán un gran overhead.
 - Importante el reciclaje de hebras: finalizado implica nuevo.
- Hebras híbridas. Estas intentan combinar las ventajas de las hebra de usuario y las de núcleo. Una forma es usar hebras de núcleo y multiplexar hebra de usuario con alguna o todas ellas. Detalles de implementación.
 - Es necesario implementar hebras tipo núcleo en el SO.
 - Utiliza hebras tipo usuario pero multiplexadas sobre hebras tipo núcleo.
 - Cada hebra del núcleo ejecuta un conjunto de hebras de usuario que se turnan en su utilización.

Con esta aproximación, el núcleo es consciente solo de los hebras a nivel de kernel y planifica las mismas.



Activaciones del planificador

Método propuesto por Thomas E. Anderson. En este el núcleo asigna cierto número de procesadores virtuales o procesos ligeros a cada proceso.

- Inicialmente suele asignarse un procesador virtual.
- Cada proceso puede solicitar y devolver procesadores virtuales.
- El núcleo puede conceder y retirar procesadores virtuales.

Cuando el núcleo detecta que una hebra se bloquea avisa al sistema de gestión de hebras en espacio de usuario mediante una llamada directa ("up-call"). Una vez avisado un planificador a nivel usuario puede seleccionar otra hebra para ejecutar. Cuando la hebra se desbloquea de nuevo el núcleo avisa al sistema de gestión de hebras en espacio de usuario.

Ventajas de este enfoque:

- Dos formas de manejar las llamadas al sistema bloqueantes:
 - Una llamada bloqueante en una hebra de usuario sólo bloquea su correspondiente hebra del núcleo hasta que finalice la causa del bloqueo mientras que el resto de hebras del proceso continúan ejecutándose.
 - La hebra del núcleo almacena la causa del bloqueo y avisa al planificador a nivel usuario para que cambie a otra hebra de usuario hasta que finalice la causa del bloqueo.

Por otro lado los principales inconvenientes son:

- El SO debe disponer de hebras de tipo núcleo.
- Hay dos niveles de planificación:
 - El núcleo planifica las hebras del núcleo.
 - Las hebras de usuario son planificadas por una biblioteca.
 - Ambos planificadores deben cooperar.
- Problema: las llamadas directas violan los principios de los sistemas por capas.

Representación de hebras

Para poder gestionar hebras necesitamos estructuras de datos que las representen. Este es el bloque de control de hebra (TCB, Thread Control Block)

TCB mínimo
identificador de hebra (TID)
puntero de instrucción (IP)
puntero de pila (SP)
estado (flags)