**Resumen del libro “Sistemas operativos, aspectos internos y principios de diseño” 5ta edición de William Stallings**

**La parte de virtualización está media floja. Lo que está en verde no sé si está bien**

Contenido

[Capítulo 1: Introducción (pág. 9) 8](#_30j0zll)

[Elementos básicos 9](#_1fob9te)

[Ejecución de instrucciones 9](#_3znysh7)

[Búsqueda y Ejecución de una instrucción 9](#_2et92p0)

[Tipos de instrucciones: 9](#_tyjcwt)

[Interrupciones 9](#_3dy6vkm)

[Clases de interrupciones: 9](#_1t3h5sf)

[Una interrupción suspende la secuencia normal de ejecución: 9](#_4d34og8)

[Controlador/manejador de interrupciones o rutina de servicio de interrupción (ISR): 9](#_2s8eyo1)

[Interrupción simple: 9](#_17dp8vu)

[Interrupciones múltiples: 9](#_3rdcrjn)

[Multiprogramación 9](#_26in1rg)

[Jerarquía de la memoria 9](#_lnxbz9)

[Técnicas de comunicación de E/S 9](#_35nkun2)

[E/S programada: 9](#_1ksv4uv)

[E/S dirigida por interrupciones. 9](#_44sinio)

[Acceso directo a memoria (DMA) 9](#_2jxsxqh)

[Capítulo 2: Objetivos y funciones del SO (pág. 53) 9](#_z337ya)

[Objetivos: 9](#_3j2qqm3)

[Servicios provistos por el sistema operativo (7) 9](#_1y810tw)

[El sistema operativo como administrador de recursos (eficiencia) 9](#_4i7ojhp)

[Capacidad de evolución de un sistema operativo 9](#_2xcytpi)

[Evolución de los sistemas operativos 9](#_1ci93xb)

[1) Procesamiento en serie 9](#_3whwml4)

[2) Sistema simple de lotes (Sistema en lotes sencillos) 9](#_2bn6wsx)

[3) Sistemas de lotes con multiprogramación: 9](#_qsh70q)

[4) Sistemas de tiempo compartido (multiusuario): 9](#_3as4poj)

[Principales logros del sistema operativo (5) 9](#_1pxezwc)

[1)](#_49x2ik5) Procesos 9

[2)](#_2p2csry) Manejo de la Memoria (5) 9

[3)](#_147n2zr) Protección y seguridad de la información (5) 9

[4)](#_3o7alnk) Planificación y manejo de recursos (3) 9

[5)](#_23ckvvd) Estructura del sistema 9

[Capítulo 3: Procesos (pág. 105) 9](#_ihv636)

[Principales requisitos de un sistema operativo 9](#_32hioqz)

[Proceso 9](#_1hmsyys)

[Bloque de control de proceso (BCP) 9](#_41mghml)

[Identificación del proceso 9](#_2grqrue)

[Información de estado del procesador 9](#_vx1227)

[Información de control del proceso (8) 9](#_3fwokq0)

[Imagen de un proceso 9](#_1v1yuxt)

[Motivos de creación de un proceso (4) 9](#_4f1mdlm)

[Pasos para crear un proceso (5) 9](#_2u6wntf)

[Cambio/conmutación de procesos 9](#_19c6y18)

[Sucesos para que el SO tome control (3) 9](#_3tbugp1)

[Motivo de terminación de un proceso (9) 9](#_28h4qwu)

[Estado de un proceso (3 modelos) 9](#_nmf14n)

[Modelo de 2 estados de proceso 9](#_37m2jsg)

[Modelo de 5 estados de proceso 9](#_1mrcu09)

[Modelo con múltiple suspendidos, 7 estados 9](#_46r0co2)

[Estructura de datos del SO 9](#_2lwamvv)

[Tablas (4) 9](#_111kx3o)

[Modo de Ejecución 9](#_3l18frh)

[Funciones típicas de un núcleo de sistema operativo 9](#_206ipza)

[Cambio de modo (5) 9](#_4k668n3)

[Cambio de proceso (7) 9](#_2zbgiuw)

[Tipos de ejecución del SO 9](#_1egqt2p)

[Kernel sin procesos: 9](#_3ygebqi)

[Ejecución dentro de los procesos de usuario: 9](#_2dlolyb)

[Sistema operativo basado en procesos: 9](#_sqyw64)

[Capítulo 4: Hilos, SMP y micronúcleos (pág. 157) 9](#_3cqmetx)

[Procesos e Hilos (Threads) 9](#_1rvwp1q)

[Multihilo 9](#_4bvk7pj)

[Ejemplo: 9](#_2r0uhxc)

[Recursos que comparten los hilos de un proceso: 9](#_1664s55)

[Contenido de un hilo (5) 9](#_3q5sasy)

[Beneficios de los hilos: 9](#_25b2l0r)

[Estados de los hilos: 9](#_kgcv8k)

[Sincronización de hilos 9](#_34g0dwd)

[Hilos de nivel usuario y de núcleo (Pág. 165) 9](#_1jlao46)

[Hilos de nivel de usuario (ULT, user level threads) 9](#_43ky6rz)

[Hilos de nivel de núcleo (KLT, kernel level threads) o procesos ligeros: 9](#_2iq8gzs)

[Propuestas combinadas de hilos (6): 9](#_xvir7l)

[Procesos Windows 9](#_3hv69ve)

[Estado hilos: 9](#_1x0gk37)

[Solaris 9](#_4h042r0)

[Linux 9](#_2w5ecyt)

[Estado hilos (6) 9](#_1baon6m)

[Capítulo 5: Concurrencia y Exclusión mutua (pág. 201) 9](#_3vac5uf)

[Concurrencia 9](#_2afmg28)

[La concurrencia es fundamental en 9](#_pkwqa1)

[Se presenta en 3 contextos: 9](#_39kk8xu)

[Dificultades (3) 9](#_1opuj5n)

[Por la concurrencia, el SO tiene que tener en cuenta (4): 9](#_48pi1tg)

[Interacción entre los procesos concurrentes 9](#_2nusc19)

[Requisitos de exclusión mutua (6) 9](#_1302m92)

[Herramientas para la concurrencia 9](#_3mzq4wv)

[Exclusión mutua por soporte de hardware 9](#_2250f4o)

[Deshabilitar interrupciones 9](#_haapch)

[Instrucciones de máquina especiales 9](#_319y80a)

[Semáforos 9](#_1gf8i83)

[Semáforo no-binario o general: 9](#_40ew0vw)

[Semáforo binario: 9](#_2fk6b3p)

[Problema Productor/Consumidor (con semáforos no binarios) 9](#_upglbi)

[Monitores 9](#_3ep43zb)

[Paso de mensajes 9](#_1tuee74)

[Sincronización 9](#_4du1wux)

[Direccionamiento 9](#_2szc72q)

[Formato de menaje 9](#_184mhaj)

[Seudocódigo 9](#_3s49zyc)

[Capítulo 6: Interbloqueo e inanición (pág. 257) 9](#_279ka65)

[Tipos de recursos 9](#_meukdy)

[Recursos reutilizables 9](#_36ei31r)

[Recursos consumibles 9](#_1ljsd9k)

[Grafo de asignación de recursos 9](#_45jfvxd)

[Interbloqueo 9](#_2koq656)

[Ejemplo 9](#_zu0gcz)

[Condiciones para el interbloqueo 9](#_3jtnz0s)

[Estrategias para tratar el interbloqueo 9](#_1yyy98l)

[∙](#_4iylrwe) Prevención (antes de y sin saber que pasará) 9

[∙](#_2y3w247) Predicción (mientras sucede busco el camino bueno) 9

[Algoritmo del banquero 9](#_1d96cc0)

[Ventajas 9](#_3x8tuzt)

[Desventaja 9](#_2ce457m)

[∙](#_rjefff) Detección 9

[Estrategias de recuperación 9](#_3bj1y38)

[Criterios de selección de procesos (5) 9](#_1qoc8b1)

[Capítulo 7: Gestión de memoria (pág. 305) 9](#_4anzqyu)

[Diccionario 9](#_2pta16n)

[Requisitos (5) 9](#_14ykbeg)

[∙](#_3oy7u29) Reubicación 9

[∙](#_243i4a2) Protección 9

[∙](#_j8sehv) Compartición 9

[∙](#_338fx5o) Organización lógica 9

[∙](#_1idq7dh) Organización física 9

[Técnica de gestión de memoria 9](#_42ddq1a)

[1.](#_2hio093) Particionamiento fijo 9

[2.](#_wnyagw) Particionamiento dinámico 9

[2.1](#_3gnlt4p) Estrategia/sistema buddy (colegas): 9

[3.](#_1vsw3ci) Paginación sencilla 9

[4.](#_4fsjm0b) Segmentación sencilla 9

[5.](#_2uxtw84) Paginación con memoria virtual 9

[6.](#_1a346fx) Segmentación con memoria virtual 9

[Capítulo 8: Memoria virtual (pág. 339) 9](#_3u2rp3q)

[Diccionario 9](#_2981zbj)

[Ejecución de un programa 9](#_odc9jc)

[Ventajas de fragmentar 9](#_38czs75)

[Hiperpaginación (thrashing) 9](#_1nia2ey)

[Principio de cercanía/proximidad (anticipo) 9](#_47hxl2r)

[Soporte necesario para memoria virtual 9](#_2mn7vak)

[Paginación con memoria virtual 9](#_11si5id)

[Traducción de direcciones en sistemas de paginado (Se evalúa) 9](#_3ls5o66)

[Cache TLB (pág. 349) 9](#_20xfydz)

[Tamaño de página 9](#_4kx3h1s)

[Segmentado con memoria virtual 9](#_302dr9l)

[Tabla de segmentos: 9](#_1f7o1he)

[Traducción 9](#_3z7bk57)

[Sistema combinado de paginación/segmentación 9](#_2eclud0)

[Traducción 9](#_thw4kt)

[Políticas que lleva el SO para poder gestionar la memoria virtual (6) 9](#_3dhjn8m)

[Conjunto de trabajo 9](#_1smtxgf)

[1.](#_4cmhg48) Política de lectura/recuperación 9

[2.](#_2rrrqc1) Política de vaciado 9

[3.](#_16x20ju) Política de ubicación 9

[4.](#_3qwpj7n) Política de reemplazo 9

[Marcos que no se deben reemplazar: 9](#_261ztfg)

[Algoritmos de reemplazo 9](#_l7a3n9)

[Buffering de páginas. 9](#_356xmb2)

[5.](#_1kc7wiv) Política de asignación 9

[Asignación fija 9](#_44bvf6o)

[Asignación variable 9](#_2jh5peh)

[Algoritmo de frecuencia de fallo de página 9](#_ymfzma)

[6.](#_3im3ia3) Control de carga (pág. 376) 9

[Suspensión de procesos (6) 9](#_1xrdshw)

[Capítulo 9: Planificación (pág. 399) 9](#_4hr1b5p)

[Nivel de planificación 9](#_2wwbldi)

[Largo plazo 9](#_1c1lvlb)

[Mediano plazo 9](#_3w19e94)

[Corto plazo (Despachador) 9](#_2b6jogx)

[Criterios de planificación a corto plazo 9](#_qbtyoq)

[Prioridades 9](#_3abhhcj)

[Función de selección 9](#_1pgrrkc)

[Modo de decisión 9](#_49gfa85)

[Tipos de planificación a corto plazo 9](#_2olpkfy)

[1.](#_13qzunr) FIFO (FCFS) 9

[2.](#_3nqndbk) Round Robin 9

[3.](#_22vxnjd) Primero el más corto (SPN) 9

[4.](#_i17xr6) Menor tiempo primero (SRT) 9

[5.](#_320vgez) Primero el de mayor tasa de respuesta (HRRN) 9

[6.](#_1h65qms) Realimentación 9

[7.](#_415t9al) Distribución justa 9

[Tema extra 1: Raid 9](#_2gb3jie)

[Conector o Interfaz física 9](#_vgdtq7)

[Interfaz lógica 9](#_3fg1ce0)

[Tiempo de acceso 9](#_1ulbmlt)

[Niveles de un disco 9](#_4ekz59m)

[Raid 9](#_2tq9fhf)

[Software 9](#_18vjpp8)

[Hardware 9](#_3sv78d1)

[Tipos de raid 9](#_280hiku)

[Tema extra 2: Virtualización 9](#_n5rssn)

[Videos 9](#_375fbgg)

[Diccionario 9](#_1maplo9)

[Virtualización 9](#_46ad4c2)

[¿Por qué usar virtualización? (7) 9](#_2lfnejv)

[Sistema tradicional 9](#_10kxoro)

[Sistema virtualizado 9](#_3kkl7fh)

[Actores de diferentes implementaciones 9](#_1zpvhna)

[Técnica de virtualización (5) 9](#_4jpj0b3)

[1.](#_2yutaiw) Emulación 9

[2.](#_1e03kqp) Virtualización total clásica (Hypervisor) 9

[Hipervisor 9](#_3xzr3ei)

[3.](#_2d51dmb) Paravirtualización 9

[4.](#_sabnu4) Contenedores 9

[Motor de contenedor (3) 9](#_3c9z6hx)

[Hay 2 sabores de contenedores 9](#_1rf9gpq)

[Ventajas 9](#_4bewzdj)

[A tener en cuenta 9](#_2qk79lc)

[Arquitecturas 9](#_15phjt5)

[Docker 9](#_3pp52gy)

[Kubernetes (orquestador) 9](#_24ufcor)

[5.](#_jzpmwk) Asistencia vía hardware 9

[Arquitecturas de Virtualización: VDI 9](#_33zd5kd)

[Hypervisor PromoxVE 9](#_1j4nfs6)

[Clúster 9](#_434ayfz)

[Donde me quedé 9](#_2i9l8ns)

# Capítulo 1: Introducción (pág. 9)

Un sistema operativo hace de intermediario entre, por un lado, los programas de aplicación, las herramientas y los usuarios, y, por otro, el hardware del computador.

Explota los recursos de hardware de uno o más procesadores.

Ofrece un conjunto de servicios a los usuarios del sistema.

Administra la memoria secundaria y los dispositivos de E/S.

El sistema operativo actúa como una interfaz entre el usuario y el hardware

## Elementos básicos

* Procesador: Controla el funcionamiento del computador y realiza sus funciones de procesamiento de datos. Cuando sólo hay un procesador, se denomina usualmente unidad central de proceso (CPU). Contiene PC (Contador de programa, dirección de la próxima instrucción), IR (Registro de instrucción, contiene la última instrucción leída), PSW (palabra de estado del programa, contiene información de estado y condiciones) y AC (Acumulador).
* Memoria principal: Almacena datos y programas. Esta memoria es habitualmente volátil; es  
  decir, cuando se apaga el computador, se pierde su contenido. En contraste, el contenido de la  
  memoria del disco se mantiene incluso cuando se apaga el computador. A la memoria principal se le denomina también memoria real o memoria primaria.
* Módulos de E/S: Transfieren los datos entre el computador y su entorno externo. El entorno  
  externo está formado por diversos dispositivos, incluyendo dispositivos de memoria secundaria (por ejemplo, discos), equipos de comunicaciones y terminales. Contiene *buffers* (zonas de almacenamiento internas) que mantienen temporalmente los datos hasta que se puedan enviar.
* Bus del sistema: Proporciona comunicación entre los procesadores, la memoria principal y los  
  módulos de E/S.

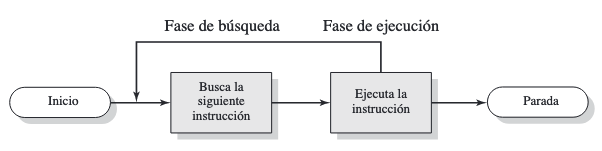
## Ejecución de instrucciones

El procesador ejecuta las instrucciones de un programa.

En su forma más simple, el procesamiento de una instrucción consta de dos pasos: el procesador lee (busca) instrucciones de la memoria, una cada vez, y ejecuta cada una de ellas.

La ejecución del programa consiste en repetir el proceso de búsqueda y ejecución de instrucciones.

Se denomina ciclo de instrucción al procesamiento requerido por una única instrucción.



En la imagen se describe el ciclo de instrucción utilizando la descripción simplificada de dos pasos. A estos dos pasos se les denomina **fase de búsqueda y fase de ejecución.**

### Búsqueda y Ejecución de una instrucción

Al principio de cada ciclo de instrucción:

* El procesador lee una instrucción.
* El PC almacena la dirección de la siguiente instrucción que se va a leer.
* El procesador incrementa el PC después de cada instrucción ejecutada (a menos que se indique otra cosa).
* La instrucción leída se carga en el IR.
* La instrucción contiene bits que especifican la acción que debe realizar el procesador.
* El procesador interpreta la instrucción y lleva a cabo la acción requerida.
* El resultado de la instrucción se guarda momentáneamente en el AC (a menos que se indique otra cosa).
* Se vuelve a empezar con la instrucción que indica el PC.

### Tipos de instrucciones:

* Procesador-memoria: transfiere datos entre procesador y memoria.
* Procesador-E/S: transfiere datos desde o hacia un dispositivo periférico de E/S.
* Procesamiento de datos: operaciones aritméticas o lógicas sobre los datos que realiza el procesador.
* Control: altera secuencia de ejecución (cambia el valor de PC). Bajo nivel y alto nivel. Por ejemplo una instrucción de salto o un condicional

## Interrupciones

* Se interrumpe al procesamiento normal del procesador.
* Mejora la eficiencia del procesamiento.
* Permite al procesador ejecutar otras instrucciones mientras ocurre una operación de E/S.
* Es una suspensión de un proceso causada por un evento externo al procesador y hecha de tal manera que el proceso pueda reanudarse.
* Evita que un programa tome todo el control o que se quede colgado. Otra alternativa, sería poner un Timer para cada cosa.

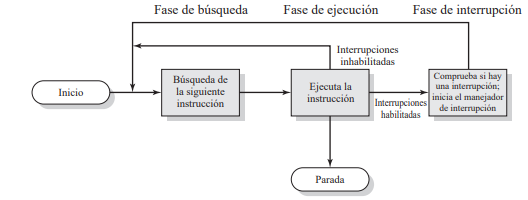
### Clases de interrupciones:

* De programa: Generada por alguna condición que se produce como resultado de la ejecución de una instrucción, tales como una división por cero, un intento de ejecutar una instrucción de máquina ilegal, y las referencias fuera del espacio de la memoria permitido para un usuario.
* Por temporizador: Generada por un temporizador del procesador. Permite al sistema operativo realizar ciertas funciones de forma regular.
* De E/S: Generada por un controlador de E/S para señalar la conclusión normal de  
  una operación o para indicar diversas condiciones de error.
* Por fallo del hardware: Generada por un fallo, como un fallo en el suministro de energía o un  
  error de paridad en la memoria.

### Una interrupción suspende la secuencia normal de ejecución:

Cuando se completa el procesamiento de la interrupción, se reanuda la ejecución. Por tanto, el programa de usuario (seria el que se estaba ejecutando) no tiene que contener ningún código especial para tratar las interrupciones; **el procesador y el sistema operativo son responsables de suspender el programa de usuario y, posteriormente, reanudarlo en el mismo punto.**

Para tratar las interrupciones, se añade a la fase de búsqueda y fase de ejecución una **fase de interrupción** al ciclo de instrucción. En la fase de interrupción, el procesador comprueba si se ha producido cualquier interrupción. Si no hay interrupciones pendientes, el procesador continúa con la fase de búsqueda y lee la siguiente instrucción del programa actual. Si está pendiente una interrupción, el procesador suspende la ejecución del programa actual y ejecuta la rutina del controlador de interrupciones.

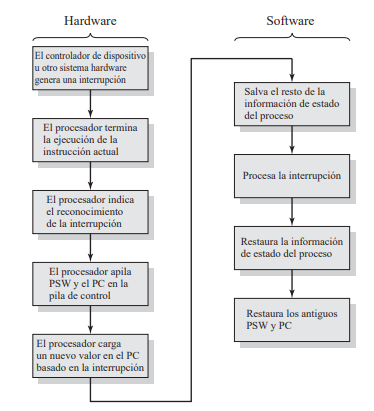


### Controlador/manejador de interrupciones o rutina de servicio de interrupción (ISR):

* Un programa/código que determina la naturaleza de la interrupción y ejecuta la acción que corresponda.
* Se transfiere el control a este programa, en modo núcleo.
* Generalmente parte del sistema operativo.

## Interrupción simple:

CUADRO IMPORTANTE



\*PSW: program status word es un área de la memoria o registro que contiene información sobre el estado de un programa utilizado por el sistema operativo.

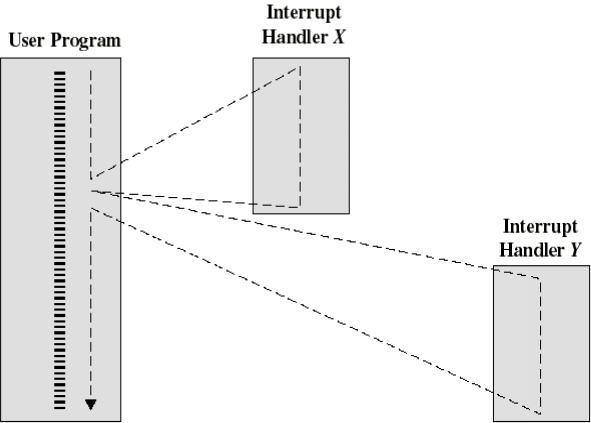
* Se guarda el PSW y PC en la pila de control.
* Se guarda el resto de la información de estado del proceso.

Esto sería salvar el contexto antes de ser interrumpido.

## Interrupciones múltiples:

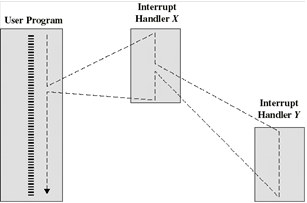
¿Qué pasa si hay una interrupción cuando estoy ejecutando una rutina de interrupción?

Se pueden considerar dos alternativas a la hora de tratar con múltiples interrupciones:

* La primera, **Interrupciones múltiples en orden secuencial, es** inhabilitar las interrupciones mientras que se está procesando una interrupción. Una interrupción inhabilitada significa simplemente que el procesador ignorará cualquier nueva señal de petición de interrupción. Si se produce una interrupción durante este tiempo, generalmente permanecerá pendiente de ser procesada, de manera que el procesador sólo la comprobará después de que se rehabiliten las interrupciones. Por tanto, cuando se ejecuta un programa de usuario y se produce una interrupción, se inhabilitan las interrupciones inmediatamente. Después de que se completa la rutina de manejo de la interrupción, se rehabilitan las interrupciones antes de reanudar el programa de usuario, y el procesador comprueba si se han producido interrupciones adicionales. 

Esta estrategia es válida y sencilla, puesto que las interrupciones se manejan en estricto orden secuencial.

La desventaja de la estrategia anterior es que no tiene en cuenta la prioridad relativa o el grado de urgencia de las interrupciones.

* Una segunda estrategia es definir **prioridades para las interrupciones** y permitir que una interrupción de mayor prioridad interrumpa la ejecución de un manejador de una interrupción de menor prioridad y se almacena el estado en la pila.

Las interrupciones de mayor prioridad hacen esperar a las interrupciones de menor prioridad interrumpen la rutina del controlador de interrupciones de las de menor prioridad.

Un ejemplo: cuando llega una interrupción por la línea de comunicación, necesita ser atendida rápidamente para hacer lugar a más entradas.

## Multiprogramación

Incluso utilizando interrupciones, puede que el procesador siga sin utilizarse eficientemente.

Una solución a este problema es permitir que múltiples programas de usuario estén activos al mismo tiempo.

Se verá más adelante…

## Jerarquía de la memoria

En todo este espectro de tecnologías, se cumplen las siguientes relaciones:  
 • Cuanto menor tiempo de acceso, mayor coste por bit.  
 • Cuanto mayor capacidad, menor coste por bit.  
 • Cuanto mayor capacidad, menor velocidad de acceso.

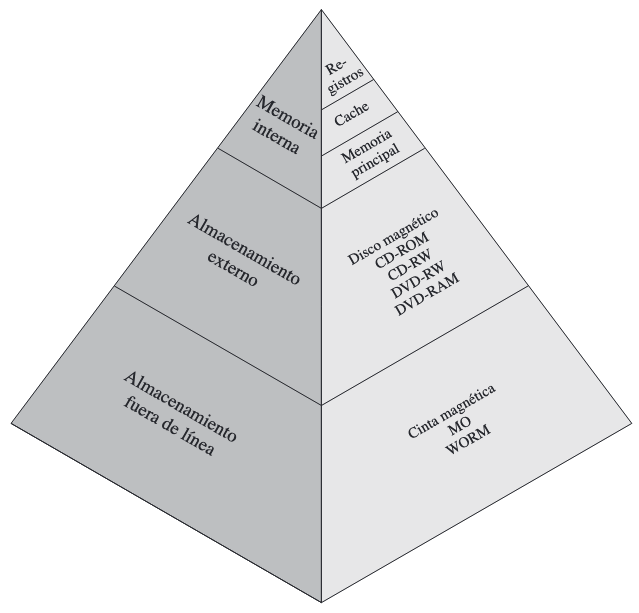
En el compromiso entre las tres características elementales de la memoria, costo, capacidad y tiempo. Podemos afirmar que "cuanto menor costo por bit, mayor tiempo de acceso".

La solución a este dilema consiste en no basarse en un único componente de memoria o en una sola tecnología, sino emplear una **jerarquía de memoria.**

Según se desciende en la jerarquía, ocurre lo siguiente:  
 a) Disminución del coste por bit.  
 b) Aumento de la capacidad.  
 c) Aumento del tiempo de acceso.  
 d) Disminución de la frecuencia de acceso a la memoria por parte del procesador.

Por tanto, las memorias más rápidas, caras y pequeñas se complementan con memorias más lentas, baratas y grandes.

La clave para el éxito de esta organización es el último aspecto: la disminución de la frecuencia de acceso.



El procesador tiene varios niveles de memoria a cuál puede acceder donde el nivel 1 es el más rápido y mayor frecuencia de acceso, pero menor tamaño, mientras que el ultimo es el más lento pero mayor tamaño con poca frecuencia de acceso.

Registros 🡪 Cache (es como Buffer de la M. Principal) 🡪 Memoria principal 🡪 Memoria secundaria (HDD, etc.)

## Técnicas de comunicación de E/S

Hay tres técnicas para llevar a cabo las operaciones de E/S:

### E/S programada:

* Lo realiza el módulo E/S, no el procesador.
* El **procesador está ocupado** chequeando el estado por **cada palabra** hasta completar todos los bloques.
* Coloca los bits apropiados en el registro de estado de E/S.
* No ocurren interrupciones.

### E/S dirigida por interrupciones.

* Se interrumpe al procesador cuando el módulo de E/S está listo para intercambiar datos.
* El **procesador está libre** para hacer otro trabajo hasta que aparezcala **interrupción de E/S** **por cada palabra.**
* No hay espera innecesaria (no comprueba el estado, sino que le avisan).
* Consume mucho tiempo de procesador porque cada palabra leída o escrita pasa a través del procesador.

Instrucciones: Control (Especificar acción a un dispositivo), Estado (Comprobar condiciones) y de Transferencia (leer/escribir)

### Acceso directo a memoria (DMA)

* Es la técnica de E/S más eficiente.
* Transfiere un **bloque** de datos directamente desde o hacia la memoria, obteniendo el control del bus.
* Se envía interrupción cuando se completa la tarea.
* El procesador está comprometido solamente en el principio y final de la transferencia.
* Ocurren intercambios E/S con memoria directamente.
* El procesador autoriza al módulo E/S a escribir o leer de memoria.
* Libera al procesador de la tarea.
* El procesador está libre para hacer otras cosas, solamente pelea para usar el bus de datos.

Es como otro procesador que se encarga de las tareas de E/S.

# Capítulo 2: Objetivos y funciones del SO (pág. 53)

**Trabajo** = Programa de usuario

**Contexto de ejecución** = Datos internos por el cual el sistema operativo es capaz de supervisar y controlar el proceso (Bloque de control y la pila de ejecución).

**Bloque de control** = PC, registros, prioridad, espera de E/S

El sistema operativo **es un programa** que: controla la ejecución de los programas de aplicación, actúa como interfaz entre el usuario y el hardware y enmascara los detalles del hardware

## Objetivos:

* **Facilidad de uso**: Un sistema operativo facilita el uso de un computador.
* **Eficiencia**: Permite que los recursos de un sistema de computación se puedan utilizar de una manera eficiente. Ver “[El sistema operativo como administrador de recursos](#_xevivl)”.
* **Capacidad para evolucionar/adatar**: Se debe construir de tal forma que se puedan desarrollar, probar e introducir nuevas funciones en el sistema sin interferir con su servicio. Ver “[Capacidad de evolución de un sistema operativo](#_3hej1je)”.

## Servicios provistos por el sistema operativo (7)

* **Desarrollo/creación de programas**: El SO proporciona una variedad de utilidades y servicios, tales como editores y depuradores, para asistir al programador en la creación de los programas.
* **Ejecución de programas**: Se necesita realizar una serie de pasos para ejecutar un programa (Cargar en memoria, dispositivos E/S se deben inicializar, etc.), y el sistema operativo los realiza en nombre del usuario.
* **Acceso a dispositivos de E/S**: Cada dispositivo de E/S requiere su propio conjunto peculiar de instrucciones o señales de control para cada operación. El sistema operativo proporciona una interfaz uniforme que esconde esos detalles de forma que los programadores puedan acceder a dichos dispositivos utilizando lecturas y escrituras sencillas.
* **Acceso controlado a los ficheros**: Para acceder a los ficheros, antes el SO operativo tiene que reconocer la naturaleza de los dispositivos de E/S, la estructura de los datos almacenados y el propietario de los mismos. Gracias a esto, el SO puede proporcionar mecanismos de protección para controlar el acceso.
* **Acceso al sistema**: Para sistemas compartidos o públicos, el sistema operativo controla el acceso al sistema completo y a recursos del sistema específicos y proporcionar protección a los recursos y a los datos, evitando el uso no autorizado de los usuarios y resolviendo conflictos de recursos.
* **Detección y respuesta a errores**: El SO debe proporcionar una respuesta que elimine el error con el menor impacto en las aplicaciones (cerrar el programa o informar al mismo). Errores como de hardware (fallo de memoria/dispositivos), de software (desbordamiento aritmético, división por 0, acceso a lugares prohibido de M, etc.) y que el SO no puede conceder el pedido del programa (administrador/sudo/root).
* **Contabilidad**: Recoger estadísticas (para administrar mejor). Monitorear rendimiento. Anticipar mejoras futuras. Cobrar a los usuarios dependiendo sus usos.

## El sistema operativo como administrador de recursos (eficiencia)

* Las funciones del sistema operativo actúan de la misma forma que el resto del software; es decir, **se trata de un programa o conjunto de programas ejecutados por el procesador (consume recursos)**.
* El sistema operativo frecuentemente cede el control y depende del procesador para volver a  
  retomarlo.

Entonces el SO:

* Dirige al procesador en el uso de los otros recursos del sistema y en la temporización de la ejecución de otros programas.
* Comparte el procesador con los otros programas (deja el control al programa y luego lo retoma).

## Capacidad de evolución de un sistema operativo

Un sistema operativo importante debe evolucionar en el tiempo por las siguientes razones:

* **Actualización y renovación de tipos de hardware**
* **Nuevos servicios**: en respuesta a la demanda de los usuarios y gestores del sistema.
* **Resolución de fallos:** Cualquier sistema operativo tiene fallos, se descubren con el  
  transcurso del tiempo y se resuelven, pero esto puede implicar nuevos fallos.

## Evolución de los sistemas operativos

### 1) Procesamiento en serie

* No había sistema operativo.
* Programas en código de máquina.
* Problemas:
* Reserva previa de tiempo de uso.
* La instalación incluía cargar el compilador, el programa fuente, guardar el programa compilado, y luego cargar y enlazar.
* Si había un error, se volvía a empezar.

### 2) Sistema simple de lotes (Sistema en lotes sencillos)

Secuencia de tareas similares.

Las primeras máquinas eran muy caras, y, por tanto, era importante maximizar su utilización. El tiempo malgastado en la planificación y configuración de los trabajos era inaceptable.

La idea central bajo el esquema de **procesamiento en lotes sencillo es el uso de una pieza de software denomina monitor**. Con este tipo de sistema operativo, el usuario no tiene que acceder directamente a la máquina. En su lugar, el usuario envía un trabajo a través de una tarjeta, que crea un sistema por lotes con todos los trabajos enviados y coloca la secuencia de trabajos en el dispositivo de entrada, para que lo utilice el monitor. Cuando un programa finaliza su procesamiento, devuelve el control al monitor, punto en el cual dicho monitor comienza la carga del siguiente programa.

Dos puntos de vista:

* **Punto de vista del monitor**: El monitor controla la secuencia de eventos. Para ello, una gran  
  parte del monitor debe estar siempre en memoria principal y disponible para la ejecución. Esta porción del monitor se denomina monitor residente. El resto del monitor está formado por un conjunto de utilidades y funciones comunes que se cargan como subrutinas (en JCL) en el programa de usuario, al comienzo de cualquier trabajo que las requiera.

El monitor lee de uno en uno los trabajos desde el dispositivo de entrada. Una vez leído el dispositivo, el trabajo actual se coloca en el área de programa de usuario, y se le pasa el control. Cuando el trabajo se ha completado, devuelve el control al monitor, que inmediatamente lee el siguiente trabajo. Los resultados de cada trabajo se envían a un dispositivo de salida para entregárselo al usuario.

* **Punto de vista del procesador**: En un cierto punto, el procesador ejecuta instrucciones de la zona de memoria principal que contiene el monitor. Estas instrucciones provocan que se lea el siguiente trabajo y se almacene en otra zona de memoria principal. Una vez que el trabajo se ha leído, el procesador encontrará una instrucción de salto en el monitor que le indica al procesador que continúe la ejecución al inicio del programa de usuario. El procesador entonces ejecutará las instrucciones del programa usuario (el programa usuario tiene el control) hasta que encuentre una condición de finalización o de error. Cualquiera de estas condiciones hace que el procesador ejecute la siguiente instrucción del programa monitor.

El monitor realiza una función de planificación: en una cola se sitúa un lote de trabajos, y los trabajos se ejecutan lo más rápidamente posible, sin ninguna clase de tiempo ocioso entre medias. Además, el monitor mejora el tiempo de configuración de los trabajos. Con cada uno de los trabajos, se incluye un conjunto de instrucciones en algún formato primitivo de lenguaje de control de trabajos (JCL).

#### Lenguaje de control de trabajos (JCL):

Se trata de un tipo especial de lenguaje de programación utilizado para proveer instrucciones al monitor como: que compilador usar o que datos usar.

#### Características de hardware (5):

El monitor, o sistema operativo en lotes, es simplemente un programa. Éste confía en la habilidad del procesador para cargar instrucciones de diferentes porciones de la memoria principal que de forma alternativa **le permiten tomar y abandonar el control**. Otras características de hardware que son deseables:

* **Protección de memoria**: No permitir que un programa de usuario altere el área de memoria donde está el monitor.
* **Temporizador**: para evitar que una tarea monopolice el sistema, al expedir el tiempo se devuelve el control al monitor.
* **Instrucciones privilegiadas**: instrucciones ejecutadas solamente por el monitor. Hay interrupción cuando un programa prueba estas instrucciones.
* **Interrupciones**: provee flexibilidad para controlar los programas de usuario.

Los programas de usuario se ejecutan en **modo usuario** y el monitor se ejecuta en **modo núcleo** o modo privilegiado. [Ver acá.](#_1wjtbr7)

### 3) Sistemas de lotes con multiprogramación:

Su principal objetivo **es maximizar el uso del procesador.**

Simula paralelismo entre procesos.

La multiprogramación normalmente provee una utilización de los recursos más eficiente que la monoprogramación.

El procesador se encuentra frecuentemente ocioso, incluso con el secuenciamiento de trabajos automático que proporciona un sistema operativo en lotes simple. El problema consiste en que los dispositivos de E/S son lentos comparados con el procesador.

Permite al procesador ejecutar otro programa mientras un programa debe esperar por un dispositivo de entrada salida (**usando el DMA)**. Esto es el tema central de los SO modernos.

Para tener varios trabajos listos para ejecutar, éstos deben guardarse en memoria principal, requiriendo alguna forma de **gestión de memoria**. Adicionalmente, si varios trabajos están listos para su ejecución, el procesador debe decidir cuál de ellos ejecutar; esta decisión requiere un **algoritmo para planificación.**

### 4) Sistemas de tiempo compartido (multiusuario):

**Minimizar el tiempo de respuesta.**

En un sistema de tiempo compartido, **múltiples usuarios acceden simultáneamente** al sistema a través de terminales, siendo el sistema operativo el encargado de entrelazar la ejecución de cada programa de usuario en pequeños intervalos de tiempo o cuantos de computación. Por tanto, si hay “n” usuarios activos solicitando un servicio a la vez, cada usuario sólo verá en media 1/n de la capacidad de computación efectiva, sin contar la sobrecarga introducida por el sistema operativo. Sin embargo, dado el tiempo de reacción relativamente lento de los humanos, el tiempo de respuesta de un sistema diseñado adecuadamente debería ser similar al de un computador dedicado.

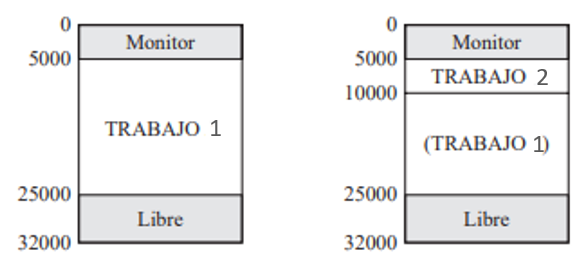
Entonces estos sistemas:

• Usando multiprogramación para manejo de múltiples tareas interactivas.

• Se comparte el tiempo de procesador entre múltiples usuarios.

• Múltiples usuarios acceden simultáneamente al sistema a través de terminales.

Cada vez que se quería hacer un trabajo (2) de otro usuario, este se escribe en memoria principal, mientras que el trabajo (1), que se estaba ejecutando, se escribe en disco. Cuando el trabajo (2) es más pequeño, no se sobrescribe sobre todo el trabajo (1), esto reduce la frecuencia de acceso al disco para escribir el trabajo (1).



## Principales logros del sistema operativo (5)

### Procesos

* Entidad activa del sistema.
* Una instancia de un programa se puede asignar o ejecutar en un procesador.
* Lo que se manifiesta por la existencia de un “bloque de control de proceso”.

Causa de errores:

* **Inapropiada sincronización**: Perdida o duplicación de señales de E/S.
* **Violación de la exclusión mutua**: Programas o usuarios usan simultáneamente un recurso.
* **Operación no determinista de un programa**: Interferencia entre programas que usan los mismos recursos, alterando el resultado final. No debería importar el orden de ejecución.
* **Interbloqueo**: Programas bloqueados entre si esperando una respuesta del otro (parecido a un bucle).

Sus componentes son 3: programa ejecutable, datos asociados, contexto de ejecución.

### Manejo de la Memoria (5)

* **Aislamiento de procesos**: Un proceso no interfiera los datos en memoria de otro.
* **Asignación y manejo automáticos**: Asignación dinámica de memoria en todos sus niveles.
* **Programación modular**: Alternar el tamaño dinámicamente.
* **Protección y control de acceso**: Permiso para que un usuario/proceso modifique la memoria de otro proceso (zona de memoria compartida).
* **Almacenamiento por largos periodos de tiempo**.

#### Memoria Virtual

Permite a los programas direccionar la memoria desde un punto de vista lógico sin tener en cuenta la cantidad de memoria principal real disponible.

Solo una porción del programa y los datos se encuentran en memoria (real) mientras el programa se ejecuta.

Las referencias a memoria se hacen con una dirección virtual donde la “Unidad de gestión de memoria” (hardware) traduce esa dirección en “dirección real” y en “dirección en disco”. Si el contenido se encuentra en disco, entonces una porción de la memoria real (principal) es llevada al disco, y el contenido buscado se llevan a la memoria principal.

### Protección y seguridad de la información (5)

* **Autenticidad/Control de acceso:**

Verifica la identidad del usuario y la validez de los datos.

* **Certificación/Control del flujo de la información:**

Regula flujo de datos dentro del sistema según las especificaciones requeridas y la entrega a los usuarios.

* **Disponibilidad:**

Protección frente a interrupciones.

* **Confidencialidad**:

Que lo usuarios no accedan a datos que no tienen permiso.

* **Integridad de datos:**

Protección de datos frente a modificaciones no autorizadas.

### Planificación y manejo de recursos (3)

* **Equidad** (repartir a todos por igual):

Dar la misma cantidad de recursos a cada proceso con demandas similares.

* **Sensibilidad diferencial** (prioridad al repartir recursos):

Discriminar entre clases de trabajos distintos para tomar decisiones dinámicas en cuanto al reparto de recursos.

* **Eficiencia** (lo mejor posible):

Maximizar transmisión, minimizar tiempo de respuesta y acomodar tantos usuarios como sea posible.

Para un mejor reparto de recursos es recomendable monitorizar el rendimiento y realizar los ajustes correspondientes.

Hay 3 tipos de colas para la espera de recursos: De corto plazo (a punto de usar el procesador), de largo plazo (cuando se va a usar el procesador se pasan a la de corto plazo) y de E/S (varios procesos al mismo dispositivo).

### Estructura del sistema

Los SO pueden llegar a ser extremadamente complejos con millones de líneas de código y son vulnerables a ataques de seguridad, virus, etc.

Para gestionar la complejidad y eliminar estos problemas, se ha puesto énfasis en la estructura de SO:

* **Sistema modular** (SO pequeños): Organizar y detectar errores más fácilmente. Módulos simples y mínimamente conectados entre sí.
* **Vista del sistema como una serie de niveles o estructura jerárquica** (SO grandes): Cada nivel realiza funciones relacionadas con los niveles inferiores (que realizan funciones más primitivas) y proporcionando servicios a los superiores. La modificación de uno no tiene que interferir en los demás.
* Los niveles descomponen un problema dentro de un número más manejable de subproblemas.
* Una de las ventajas de la vista de un sistema en varios niveles es que sirve para facilitar: la programación del sistema, la evolución del sistema y las tareas de diagnóstico.
* Los niveles bajos tratan con una escala pequeña de tiempo, porque se relacionan directamente con el hardware.
* Los niveles altos son los que se comunican con el usuario, invocan mandatos, por lo tanto, tienen una escala de tiempo mayor.
* Nivel 1 – 4: Constituyen el hardware de procesador, pero algunos elementos del SO aparecen (Manejador de interrupciones)
* Nivel 5 – 7: El SO trata únicamente con el procesador.
* Niveles 8 – 13: Dispositivos externos (Redes, periféricos, etc.)

# Capítulo 3: Procesos (pág. 105)

## Principales requisitos de un sistema operativo

La mayoría de los requisitos que un sistema operativo debe cumplir se pueden expresar con referencia a los procesos:

* El SO debe **intercalar la ejecución de varios** **procesos** para maximizar el uso del procesador y minimizar el tiempo de respuesta razonable.
* El SO debe **asignar recursos** a los procesos.
* El SO debe **soportar la comunicación** entre procesos y la creación de procesos por el usuario.

## Proceso

Un proceso es:

* Un programa en ejecución. Entidad activa (programa es una entidad pasiva).
* Tarea o unidad de trabajo.
* La entidad que se puede asignar y ejecutar en un procesador.
* Una unidad de actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual, y un conjunto de recursos del sistema asociados.

## Bloque de control de proceso (BCP)

Cada proceso está asociado a atributos que son utilizados por el SO para controlarlos.

Tres categorías generales de información del BCP:

### Identificación del proceso

* Un **identificador numérico** único asociado a este proceso, para distinguirlo del resto de procesos.
* Identificador de **usuario** responsable del trabajo.
* Identificador del **proceso padre**.

### Información de estado del procesador

* Contenidos de los **registros del procesador** (PC, Acumulador, interrupciones habilitadas/deshabilitadas y Punteros de pila de sistema)
* Palabra de estado de programa (**PSW**): contiene información de estado, modo de ejecución (usuario/root).

### Información de control del proceso (8)

Información adicional necesaria para que el SO controle y coordine los diversos procesos activos.

* + **Estado** (modelo de 7 estados)
  + **Prioridad**
  + **Evento en espera**
  + **Auditoria** (tiempo en espera, tiempo de ejecución).
  + **Comunicación y relaciones** entre procesos
  + **Privilegios** (por lugares de memoria o instrucciones)
  + **Manejo de memoria**: Punteros al código de programa, los datos asociados y a cualquier bloque de memoria compartido con otros procesos.
  + **Utilización de un recurso** (Las peticiones de E/S pendientes, dispositivos de E/S asignados a dicho proceso, una lista de los ficheros en uso por el mismo, etc.)

## Imagen de un proceso

**BCP**, **Datos de usuario**, **Programa a ejecutar** (puede compartirse con otros procesos que estén ejecutando el mismo programa) y **Pilas de sistema** (llamadas al sistema, LIFO).

Se representa la imagen de procesos almacenados en memoria virtual



## Motivos de creación de un proceso (4)

Existen cuatro eventos comunes que llevan a la creación de un proceso:

* **Para controlar una tarea por lotes**: El sistema operativo dispone de un flujo de control de lotes de trabajos, habitualmente una cinta un disco. Cuando el sistema operativo está listo para procesar un **nuevo trabajo**, leerá la siguiente secuencia de mandatos de control de trabajos.
* **Ingreso al sistema por parte de un usuario**: Log on (acceder), no es lo mismo que log in (iniciar sesión).
* **Creado por el SO para proveer un servicio al usuario.**
* **Creado por un proceso existente (Spawning):** Por motivos de modularidad o para explotar el paralelismo, un  
  programa de usuario puede ordenar la creación de un número de procesos (Padre - Hijo).

### Pasos para crear un proceso (5)

Una vez que el SO decide el motivo por el cual crearlo, los pasos que llevan a la creación son:

1. Asignar **IDproceso.**
2. **Reservar memoria** (tamaño basado en el tipo o pude fijarlo el usuario o padre).
3. **Inicialización del BCP**.
4. Establecer los **enlaces apropiados.**
5. Creación o expansión de **archivos de auditoría**.

## Cambio/conmutación de procesos

Puede ocurrir en cualquier instante en el que **el SO obtiene el control**:

### Sucesos para que el SO tome control (3)

#### Interrupciones (Externo al proceso)

* Reloj: Máximo tiempo de ejecución continuo
* E/S
* Fallo de memoria: Traer el bloque de memoria secundaria a principal.

#### Traps/Excepciones (Interno al proceso)

* Error aritmético
* Proceso pasa a estado “Terminado/Saliente”.

#### Llamada al sistema (pedido explicito)

* Rutinas que son parte del SO (abrir un archivo)

## Motivo de terminación de un proceso (9)

Todo sistema debe proporcionar los mecanismos mediante los cuales un proceso indica su finalización, o que ha completado su tarea. Los trabajos deben incluir una instrucción HALT o una llamada a un servicio del SO para su finalización.

Motivos:

* El lote hace la operación de detención.
* El proceso hace la terminación de servicio.
* **El usuario realiza un log off**:Cerrar la sesión
* **El usuario finaliza el proceso**.
* **Pedido de terminación del padre o finalización del padre**.
* **Condiciones de error y falla**: error de protección (recurso no permitido), error aritmético y fallo de E/S.
* **Mas tiempo de ejecución/espera que el que debía.**
* **Intento de acceder a memoria que no debía o que requiere más memoria** que la que se le puede otorgar.
* **Intento de ejecución de instrucción privilegiada/inexistente que no debía.**

## Estado de un proceso (3 modelos)

**Traza del proceso:** lista de la secuencia de instrucciones que se ejecutaron para dicho proceso.

Gracias al BCP, se puede interrumpir y retomar el proceso como si no hubiera pasado nada. El BCP es clave para la multiprogramación en un SO.

**Activador, Planificador a corto plazo o Despachador (**dispatcher**)**: Un programa, parte del SO, que se encarga de elegir y cambiar el proceso que se está ejecutando en el procesador (ocupa procesador cada vez que se cambia de proceso a otro).

**Ejecución con el despachador**: Al proceso A se lo interrumpe por x motivo, se ejecuta el Activador (que es un programa como cualquier otro) para luego darle el control al proceso B, y así repetidamente.

Hay 3 modelos de estados para un proceso:

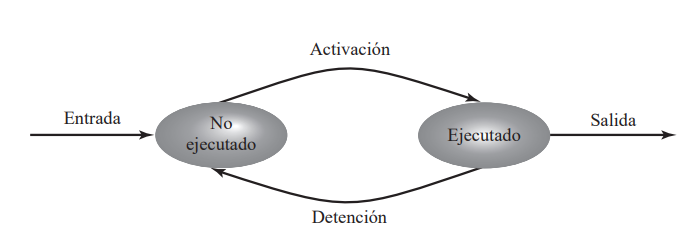
### Modelo de 2 estados de proceso

Se puede construir el modelo más simple posible observando que un proceso puede estar siendo ejecutado por el procesador o no. Un proceso puede estar en dos estados: **Ejecutando** o **No Ejecutando**.

Cuando el sistema operativo crea un nuevo proceso (B), crea su BCP con el estado de “*No Ejecutando*”. El proceso (B) está esperando su oportunidad de ejecutar, cuando el proceso actualmente en ejecución (A) se interrumpa, entonces el activador seleccionará el proceso (B) a ejecutar. El proceso saliente (A) pasará del estado “*Ejecutando*” a “*No Ejecutando*” y el nuevo proceso (B) pasará a “*Ejecutando*”.

Cada proceso debe representarse de tal manera que el sistema operativo pueda seguirle la pista, debe haber información correspondiente a cada proceso, el BCP. Los procesos que no están ejecutando deben estar en una especie de **cola (lista de tipo FIFO)**, cuyas entradas son los punteros a los BCP.

Un proceso que se interrumpe se transfiere a la cola. Si un proceso ha finalizado o ha sido abortado, se descarta (sale del sistema). En cualquier caso, el activador selecciona un proceso de la cola para ejecutar.



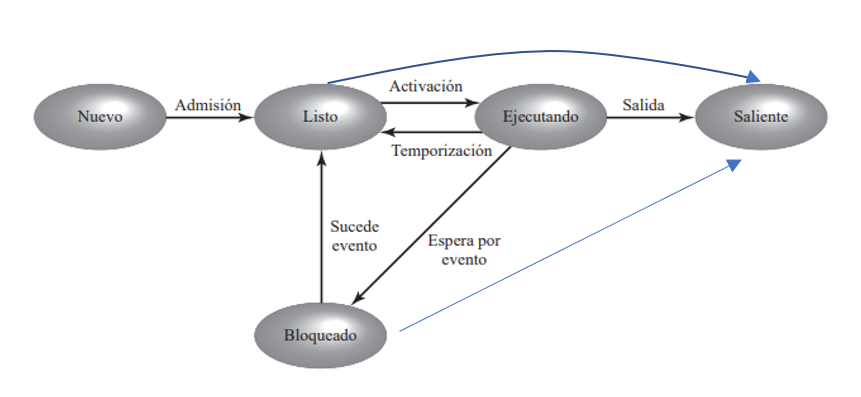
### Modelo de 5 estados de proceso

La cola es una lista de tipo FIFO y el procesador opera siguiendo una estrategia cíclica (**round-robin** o turno rotatorio, cada proceso para **ejecutar tiene un determinado tiempo** y luego regresar de nuevo a la cola, a menos que se bloquee) con todos los procesos disponibles.

Sin embargo, esta implementación es inadecuada: algunos procesos que están en el estado de “No Ejecutando” están listos para ejecutar, mientras que otros están bloqueados, esperando a que se complete una operación de E/S. Por tanto, el activador no puede seleccionar únicamente los procesos que lleven más tiempo en la cola. En su lugar, debería recorrer **la lista buscando los procesos que no estén bloqueados y que lleven en la cola más tiempo**.

La solución es dividir el estado de “No Ejecutando” en dos estados nuevos: “**Listo**” y “**Bloqueado**”. Para gestionarlo correctamente, se han añadido “**Nuevo**” y “**Saliente**”.

* **Nuevo**: Un proceso que se acaba de crear y no está en el grupo de ejecutables. Se trata de un proceso que no ha sido cargado en memoria principal, aunque su BCP si ha sido creado.
* **Listo**: Un proceso que se prepara para ejecutar cuando tenga oportunidad (en espera de procesador).
* **Ejecutando**: El proceso está actualmente en ejecución.
* **Bloqueado**: No puede ejecutar hasta que se cumpla un evento determinado o se complete una operación E/S.
* **Saliente**: Ha sido liberado del grupo de procesos ejecutables, debido a que ha sido detenido o que ha sido abortado por alguna razón. Los datos del proceso se guardan momentáneamente para que programas auxiliares puedan recopilar información (contaduría, estadísticas, tiempos, etc.)



En la lista de “Bloqueados” es recomendable tener una lista para cada evento, esto ayuda a no tener que buscar en una única lista los procesos que esperan x evento.

Si el SO trabaja con prioridades, entonces en la lista de “Listos” es recomendable tener una lista para cada prioridad, es el mismo concepto que el anterior.

#### Las posibles transiciones son: (9)

* **Null 🡪 Nuevo**: Se crea un nuevo proceso para ejecutar un programa.
* **Nuevo 🡪 Listo**: Admisión, pasa a estar preparado para ejecutar un nuevo proceso.
* **Listo 🡪 Ejecutando**: Activación, fue elegido por el planificador para ser ejecutado.
* **Ejecutando 🡪 Listo**: Temporización, el proceso en ejecución alcanzó el máximo tiempo de ejecución de forma ininterrumpida.
* **Ejecutando 🡪 Bloqueado**: Espera de un evento, un proceso se pone en el estado “Bloqueado” si solicita algo por lo cual debe esperar (una llamada al sistema) como acceso a un fichero o algún servicio del SO.
* **Bloqueado 🡪 Listo**: Sucede el evento que estaba esperando.
* **Ejecutando 🡪 Saliente**: Salida, se finaliza la ejecución por parte del SO (se ha completado o fue abortado).
* **Listo 🡪 Saliente**: El proceso padre finaliza a un hijo o finaliza el proceso padre, haciendo finalizar a los hijos.
* **Bloqueado 🡪 Saliente:** El proceso padre finaliza a un hijo o finaliza el proceso padre, haciendo finalizar a los hijos. También puede no suceder el evento que esperaba y el SO lo finaliza.

### Modelo con múltiple suspendidos, 7 estados

NO se está utilizando la memoria virtual.

Dos nuevos estados:

* Bloqueado suspendido
* Listo suspendido

**Swapping** **(memoria de intercambio):** Es el intercambio de proceso completo entre disco y memoria principal porque ambos procesos no entran en la memoria principal, no es lo mismo que memoria virtual donde ahí son partecitas de los procesos en disco y memoria principal.

**Expulsión**: Un proceso (A) quiere un recurso que tiene otro proceso (B), por lo tanto, (A) expulsa a (B) y toma el recurso.

#### Procesos suspendidos

Son procesos que:

* No está inmediatamente disponible.
* Están en disco y no en memoria principal.
* La mayoría son porque esperan/esperaban un E/S.
* La condición de bloqueo es independiente de la suspensión (espera el evento igualmente).
* Un agente (El procesador, el SO o el proceso padre) puede ponerlos en suspensión.
* El mismo agente los tiene que sacar de suspendido.

#### Razones para la suspensión

* **Swapping:** Para liberar espacio en Memoria Principal porque no hay ningún proceso en “Listo” sino en “Bloqueado” (el procesador se encuentra ocioso), esto sucede por la gran diferencia de velocidad entre procesador y E/S.

El SO aplica Swapping para liberar la memoria principal y poder traer del disco un proceso en estado “Listo”. Guardar en disco puede empeorar la situación ya que es E/S, pero suele estar disponible y ser más rápido que una Impresora, entre otros.

* El sistema operativo puede suspender un proceso que se **sospecha puede causar algún problema**.
* **Solicitud del usuario**: Puede desear suspender un proceso para su depuración o porque está utilizando un recurso.
* **Temporización**: Un proceso puede ejecutarse periódicamente (pero pocas veces) y suspenderse mientras espera el siguiente intervalo de ejecución.
* **Solicitud del proceso padre**: Puede querer suspender la ejecución de un hijo para modificarlo o para coordinar la actividad de varios procesos hijos.

#### 7 estados



## Estructura de datos del SO

Para que el SO gestione los procesos y recursos, debe saber su estado actual por lo tanto construye y mantiene **tablas** **de información** para cada entidad gestionada.

### Tablas (4)

#### Tablas de Memoria

Se usan para mantener un registro tanto de la memoria principal (real) como de la secundaria (virtual). Los procesos se mantienen en memoria secundaria utilizando Memoria Virtual o Swapping. Las tablas de memoria deben incluir 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 para accesos a regiones de memoria compartida.
* La información necesaria para manejar la memoria virtual.

#### Tablas de E/S

Se utilizar para gestionar los dispositivos de E/S y los canales del computador. Las tablas de E/S deben incluir:

* Dispositivos de E/S disponibles o asignados.
* Estado de operación de E/S.
* Lugar de memoria principal usado como fuente o destino de la transferencia con E/S.

#### Tablas de Archivos (ficheros)

Estas proporcionan información sobre:

* Existencia de archivos.
* Posición en memoria secundaria.
* Estado actual.
* Atributos.

Esta información la gestiona el sistema de manejo de archivos.

#### Tabla de Procesos

De alguna forma directa o indirecta, esta table se relaciona con las otras 3 y entre ellas también.

Contienen:

* Punteros a la información de cada proceso (Imagen de proceso, Bloque de control de proceso)

## Modo de Ejecución

Muchos procesadores proporcionan al menos dos modos de ejecución. Ciertas instrucciones se pueden ejecutar en modos privilegiados únicamente. Las instrucciones son:

* Lectura y modificación de los registros de control.
* Acceso a ciertas regiones de memoria.

El modo menos privilegiado a menudo se denomina **modo usuario**, se ejecutan normalmente los programas de usuario en este modo.

El modo más privilegiado se denomina **modo sistema, modo control, modo kernel o modo núcleo**. Este último término se refiere al núcleo/kernel del sistema operativo, que es la parte del sistema operativo que engloba las funciones más importantes del sistema.

### Funciones típicas de un núcleo de sistema operativo

#### Administración de procesos:

• Creación y terminación de procesos.

• Planificación y despacho de procesos.

• Conmutación de procesos.

• Sincronización de procesos y soporte para comunicación entre procesos.

• Administración de BCP.

#### Manejo de memoria:

• Asignación del espacio de direcciones a los procesos.

• Swapping.

• Manejo de segmentos y páginas.

#### Manejo de E/S:

• Manejo de buffer.

• Asignación a procesos de canales de E/S y dispositivos.

#### Funciones soportadas:

* Manejo de interrupciones.
* Contabilidad
* Monitoreo.

## Cambio de modo (5)

Cuando hay una interrupción pendiente:

1. **Salva contexto** del proceso: El BCP.
2. **Cambia el** **CP** al que corresponde a la rutina de interrupciones.
3. Cambia a **modo de núcleo**.
4. **Ejecuta rutina de interrupciones**.
5. Se puede reanudar el mismo proceso o cambiar a otro.

Un cambio de modo puede ocurrir sin que se cambie el estado del proceso actualmente en estado “Ejecutando”. La salvaguarda del estado y su posterior restauración comportan sólo una ligera sobrecarga.

## Cambio de proceso (7)

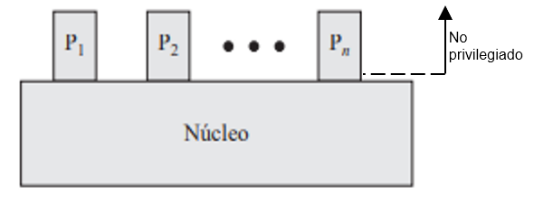
Si el proceso actualmente en estado Ejecutando, se va a mover a cualquier otro estado (Listo, Bloqueado, etc.), entonces el SO debe realizar cambios sustanciales en su entorno. Los pasos que se realizan para un cambio de proceso completo son:

1. Guardar **contexto de procesador** incluyendo PC y otros registros.
2. **Actualizar el BCP (A)** con el nuevo estado.
3. **Mover BCP (A) a la cola apropiada** (“Listo”, “Bloqueado”, etc.).
4. Seleccionar **otro proceso a ejecutar (B)**.
5. **Actualizar el BCP (B)** a “Ejecutando”.
6. **Actualizar el manejo de memoria.**
7. **Restaurar contexto del proceso (B)**.

Por tanto, el cambio de proceso, que implica un cambio en el estado, requiere un mayor esfuerzo  
que un cambio de modo.

## Tipos de ejecución del SO

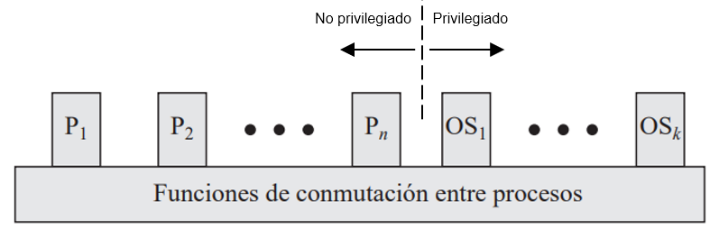
### Kernel sin procesos:

* El código del SO se ejecuta como una **entidad separada (No es un proceso)**.
* El SO opera siempre en **modo privilegiado**.
* Los **únicos procesos son los de programas de usuario**.
* Hace cambio de modo, hace cambio de proceso (a pesar que no es un proceso como tal).

### Ejecución dentro de los procesos de usuario:

* El SO dentro de los procesos de usuario (**A la imagen del proceso se le incluye una pila de núcleo**).
* Proceso se ejecuta en **modo privilegiado cuando se ejecuta código de SO.**
* Hay una **rutina de cambio de proceso** que no forma parte de ningún proceso.
* Hace cambio de modo, no hace cambio de proceso.

### Sistema operativo basado en procesos:

* Funciones principales del **kernel son procesos separados**.
* Hay una **rutina de cambio de proceso** que no forma parte de ningún proceso.
* Alienta a los SO modulares
* Conveniente para múltiples procesadores.
* Hace cambio de modo, hace cambio de proceso.

# Capítulo 4: Hilos, SMP y micronúcleos (pág. 157)

## Procesos e Hilos (Threads)

El sistema operativo trata estas dos características en forma independiente:

* **Pertenencia de recursos**: se le asigna al proceso un espacio de direcciones virtual para contener la imagen del proceso, también la memoria principal y control de otros recursos.
* **Planificación/ejecución**: La ejecución sigue una ruta de ejecución y puede estar intercalada con otros procesos.

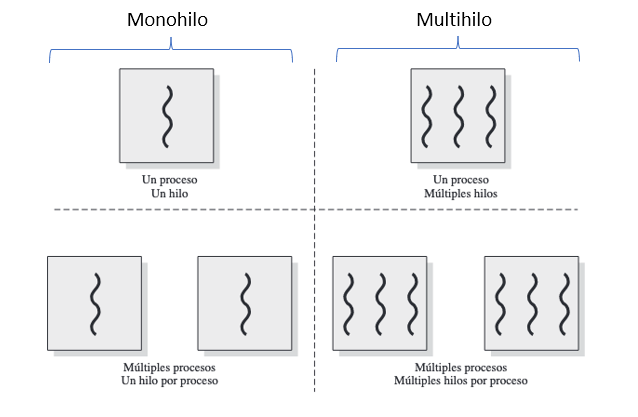
Por lo tanto:

Pertenencia de recursos => **Procesos/tareas**

Planificación/ejecución => **Hilos**

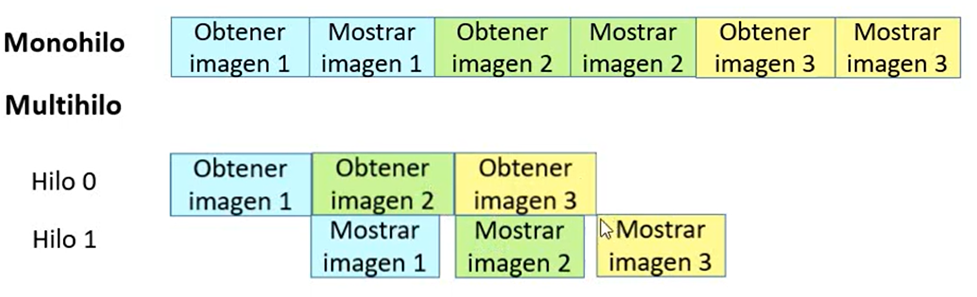
## Multihilo

Se refiere a la capacidad de dar soporte a múltiples hilos de ejecución en un solo proceso.



### Ejemplo:

Un navegador web puede tener un hilo que cargue las imágenes png y otro hilo que las muestre en pantalla.

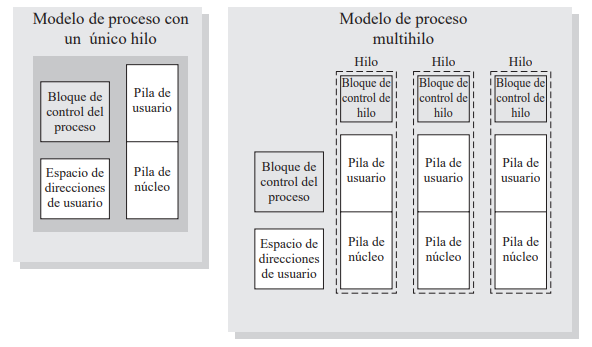


## Recursos que comparten los hilos de un proceso:

* El **espacio de direcciones virtuales** que soporta la imagen del proceso.
* **Afinidad de procesadores, otros procesos** (para comunicación entre procesos), **archivos** y **recursos de E/S** (dispositivos y canales).

## Contenido de un hilo (5)

* Un **estado** de ejecución por hilo (Ejecutando, Listo o Bloqueado.)
* Un **contexto** de hilo; una forma de ver a un hilo es como un contador de programa independiente dentro de un proceso.
* Una **pila de ejecución**.
* Un **espacio de almacenamiento estático** para variables locales.
* **Acceso a la memoria y recursos de su proceso**, compartido con todos los hilos del mismo proceso.



## Beneficios de los hilos:

* Lleva **menos tiempo crear** un nuevo hilo que crear un nuevo proceso.
* Lleva **menos tiempo finalizar** un hilo que un proceso.
* Lleva **menos tiempo cambiar/conmutar entre hilos** dentro del mismo proceso.
* Ya que los hilos dentro del mismo proceso (ULT) comparten memoria y archivos, **se pueden comunicar entre sí sin invocar al kernel.**

De esta forma, si se desea implementar una aplicación con unidades de ejecución relacionadas, es mucho más eficiente hacerlo con un conjunto de hilos que con un conjunto de procesos.

Todos los hilos de un proceso comparten el mismo espacio de direcciones, todos los hilos se suspenden al mismo tiempo y la finalización de un proceso finaliza todos los hilos de ese proceso.

## Estados de los hilos:

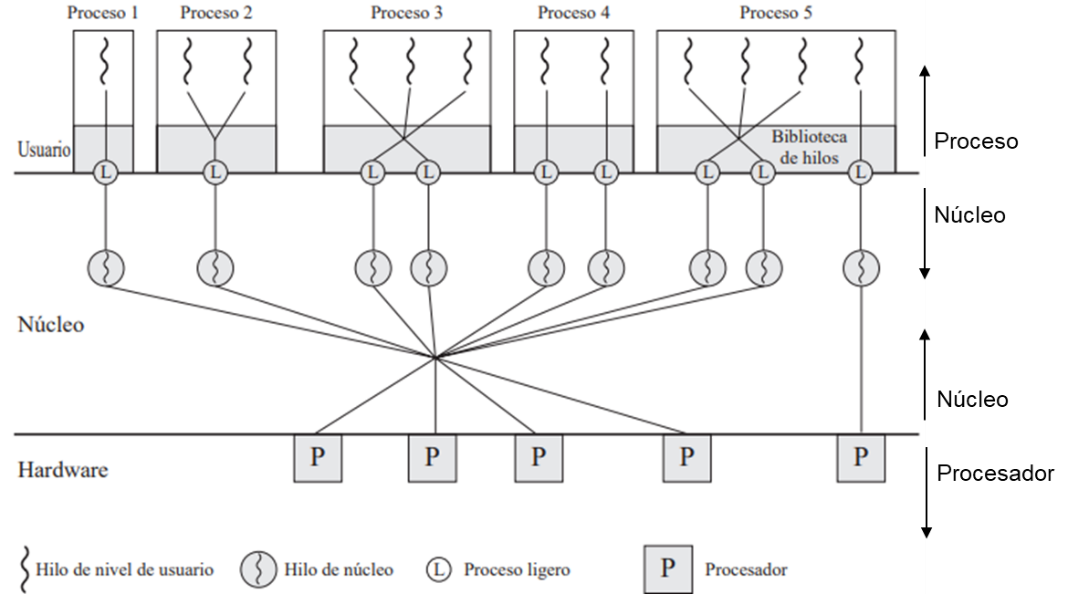
Los estados de los hilos son: Ejecutando, Listo y Bloqueado. No tiene sentido aplicar estados de suspensión a un hilo, ya que son conceptos de proceso. Si se expulsa un proceso, todos sus hilos se deben expulsar porque comparten el espacio de direcciones del proceso.

Operaciones básicas: Crear (spawn), Bloquear, Desbloquear (despertar) y Terminar.

## Sincronización de hilos

Todos los hilos de un proceso comparten el mismo espacio de direcciones y otros recursos, por lo tanto, es importante sincronizarlos para que no interfieran entre ellos. Se usan las mismas técnicas de sincronización que para los procesos.

## Hilos de nivel usuario y de núcleo (Pág. 165)



### Hilos de nivel de usuario (ULT, user level threads)

* **Los hilos los maneja la biblioteca de hilos de la aplicación,** el programador**.**
* **El kernel no sabe de su existencia.**

#### Ventajas:

* **El cambio/conmutación de hilos es más rápido,** no hay cambio de proceso, ni de modo**.**
* **Se pueden ejecutar en cualquier sistema operativo.**
* El intercambio de datos no necesita privilegios del kernel.
* La biblioteca reparte los recursos a su criterio.

#### Desventajas:

* **Se bloquea un hilo, se bloquea el proceso.**
* **No aprovecha múltiples procesadores**, un solo hilo en ejecución**.**

### Hilos de nivel de núcleo (KLT, kernel level threads) o procesos ligeros:

* Ejemplos: Windows 2000 y Linux.
* **El kernel tiene todo el contexto de los hilos y del proceso.**

#### Desventajas:

* **Lentos, cada vez que quiera cambiar un hilo, se cambia el contexto.**

#### Ventajas:

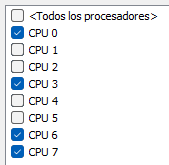
* **Aprovecha los múltiples procesadores y el paralelismo.**
* **Si se bloquea el hilo, el kernel puede activar otro hilo.**

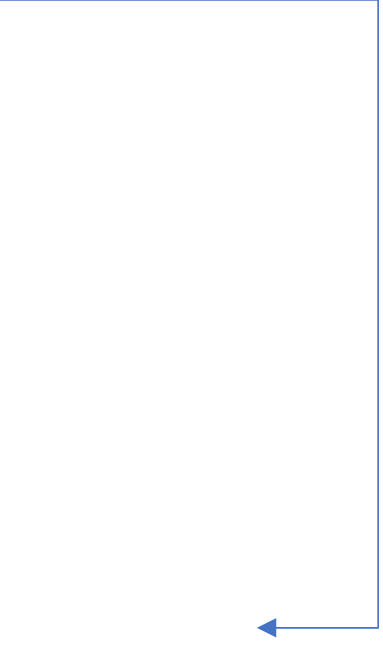
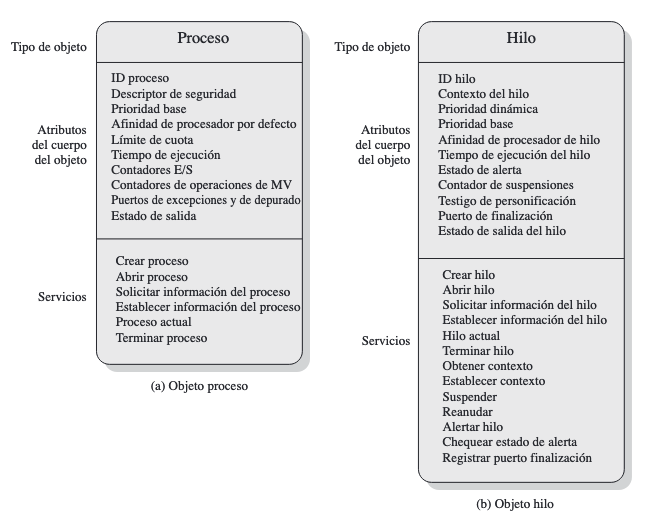
### Propuestas combinadas de hilos (6):

* Ejemplo: Solaris
* **Combina las ventajas y minimiza las desventajas.**
* El grueso de la creación, planificación y sincronización de hilos se hace en el espacio de usuario.
* Aprovecha el paralelismo.
* Si se bloquea un hilo, no se debería bloquear el proceso.

## Procesos Windows

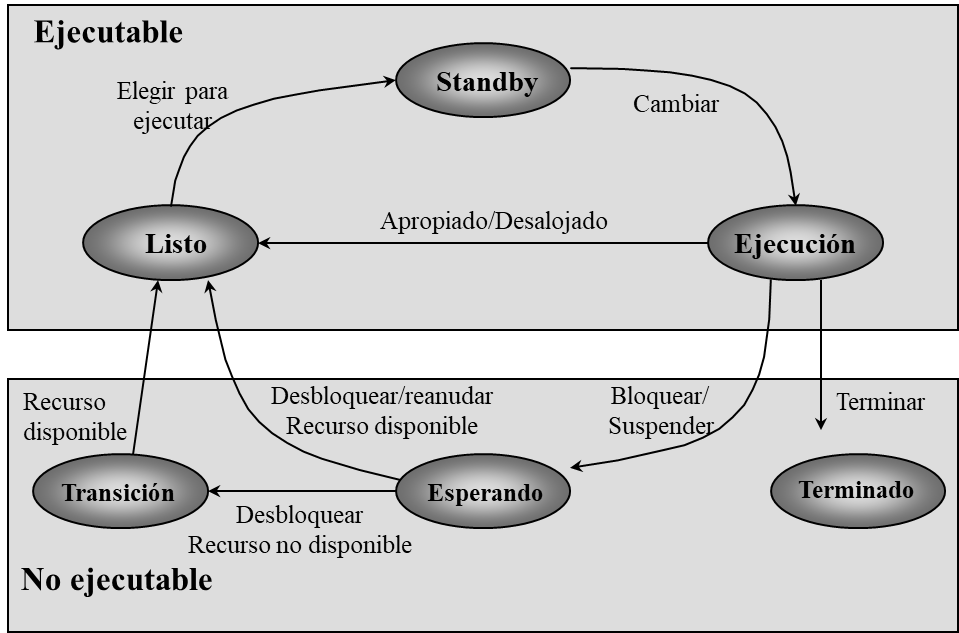
Características importantes de los procesos Windows son las siguientes:

* Los procesos están implementados como **objetos** **(Tipo, Atributo, Servicio)**.
* Un **proceso ejecutable puede contener uno o más hilos.**
* Ambos objetos, procesos e hilos, **tienen capacidades de sincronización incorporadas**.
* Se **puede asignar afinidad de procesador** a cada proceso.
* Se puede asignar afinidad de procesador a los hilos basado en la afinidad del proceso.
* El usuario puede asignar prioridad a cada proceso.



Contexto de hilo: Información para que el hilo se suspenda y reanude (pila, registros, etc.)

### Estado hilos:



* **Listo**: Está todo okey para ejecutar. El activador conoce todos los hilos listos y los planifica en orden de prioridad para mandarlo a Standby.
* **Standby**: En cola para ser ejecutado. Si un hilo sustito (A) tiene mayor prioridad que el hilo en ejecución (B), (A) desalojará a (B).
* **Ejecución**: Una vez que el micronúcleo realiza un intercambio de hilo o proceso, el hilo sustituto pasa al estado de ejecución y ejecuta hasta que es expulsado, finaliza su porción de tiempo, se bloquea o termina. En los dos primeros casos vuelve a la cola de listos.
* **Esperando:** Un hilo pasa a estado esperando cuando:

1. Se bloquea en un evento.
2. Espera voluntariamente por temas de sincronización.
3. Un subsistema manda al hilo a estado de suspendido.

* **Transición:** Cuando ya sucedió el evento que esperaba, pero su contexto no está disponible por x razón.
* **Terminado:** Un hilo se puede finalizar por sí mismo, por otro hilo o cuando su proceso padre finaliza.

## Solaris

Solaris implementa un soporte de hilo multinivel poco habitual, diseñado para proporcionar considerable flexibilidad para sacar provecho de los recursos del procesador.

* Procesos: Incluye el espacio de direcciones del usuario, la pila, el BCP y lista de estructura (reemplazando al bloque de estado del procesador).
* Hilos multiusuario
* Procesos livianos (Estado de ejecución del procesador)
* Hilos a nivel usuario y núcleo.
* Los procesos ligeros van 1 a 1 con los hilos a nivel núcleo.
* Muchos hilos de usuario a muchos procesos livianos.
* Facilidad de multiplexar hilos de usuario con procesos livianos

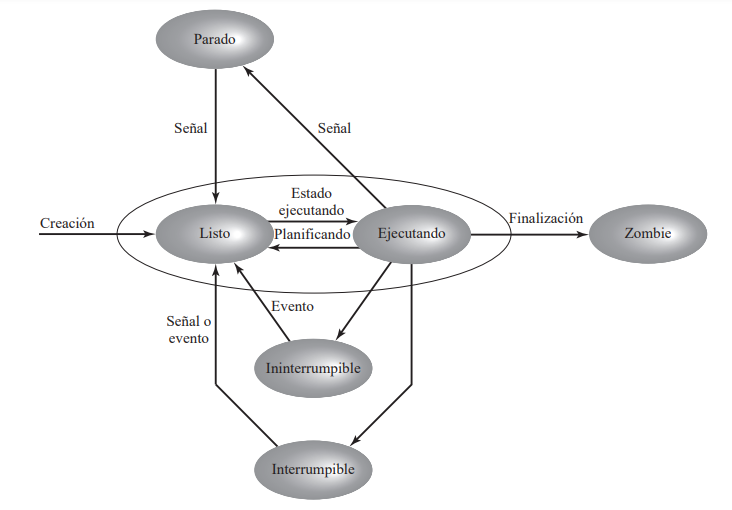
## Linux

Un proceso, o tarea, en Linux se representa por una estructura de datos “task struct” que contiene  
información de diversas categorías (9):

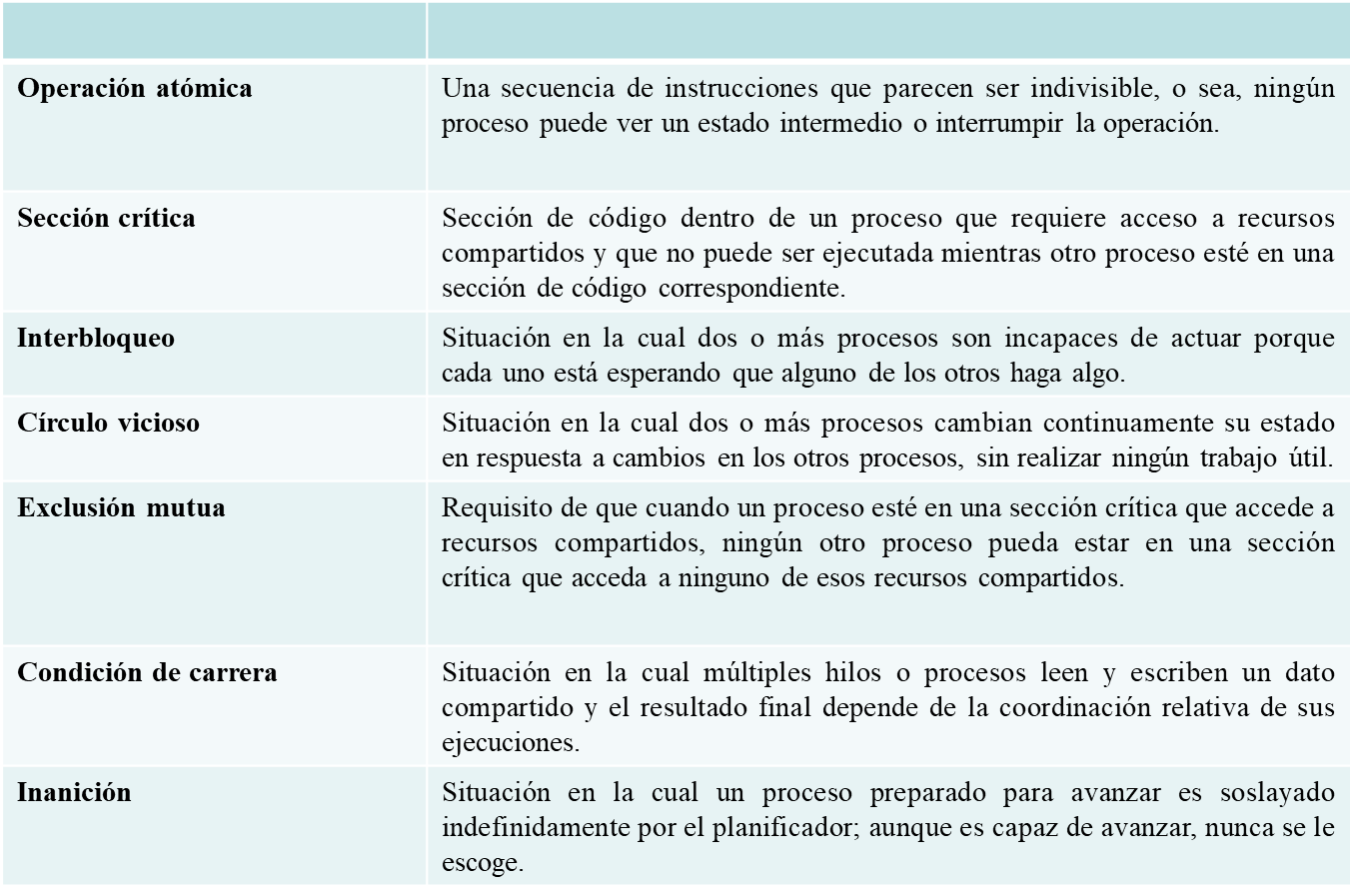
* Estado
* **Información de planificación:** Prioridad.
* **Identificadores**: Cada proceso tiene un identificador único de proceso y de usuario y grupo. El usuario se utiliza para asignar privilegios (Root o FGuarnier).
* Comunicación entre procesos.
* **Enlaces**: Entre el proceso padre, los hermanos (procesos con el mismo padre), y a sus hijos.
* **Tiempos y temporizadores**
* **Sistema de archivos**: Incluye punteros a cualquier archivo abierto por este proceso, punteros a los directorios actual y raíz para este proceso.
* **Memoria virtual**: Define el espacio de direcciones virtual asignado a este proceso.
* **Contexto específico del procesador**: La información de los registros y de la pila que constituyen el contexto de este proceso.

### Estado hilos (6)

* **Listo**: Recién creado o en espera del procesador.
* **Ejecutando**: Este valor de estado se corresponde con dos estados. Un proceso Ejecutando  
  puede estar ejecutando o está listo para ejecutar.
* **Interrumpible**: Es un estado bloqueado, en el que el proceso está esperando por un evento, tal  
  como la finalización de una operación de E/S, la disponibilidad de un recurso o una señal de  
  otro proceso.
* **Ininterrumpible**. Éste es otro estado bloqueado. La diferencia entre este estado y el estado Interrumpible es que en el estado Ininterrumpible un proceso está esperando directamente sobre un estado del hardware y por tanto no manejará ninguna señal. Prioridad muy alta, si se interrumpe retoma lo más rápido la cpu y puede suceder algún error.
* **Detenido**: En espera de una señal de otro proceso.
* **Zombie**: El proceso se ha terminado, pero, por alguna razón, todavía tiene su estructura de tarea en la tabla de procesos.



# Capítulo 5: Concurrencia y Exclusión mutua (pág. 201)



## Concurrencia

Tiene que haber Exclusión Mutua para que haya Concurrencia.

Coincidir en el lugar y/o tiempo.

### La concurrencia es fundamental en

* Procesos e hilos.
* Multiprogramación
* Multiprocesamiento
* Procesamiento distribuido (varios SO)

### Se presenta en 3 contextos:

* **Múltiples aplicaciones**: Multiprogramación
* **Aplicaciones estructuradas**: Una aplicación puede ser un conjunto de procesos concurrentes
* **Estructura del sistema operativo**: Conjunto de procesos o hilos

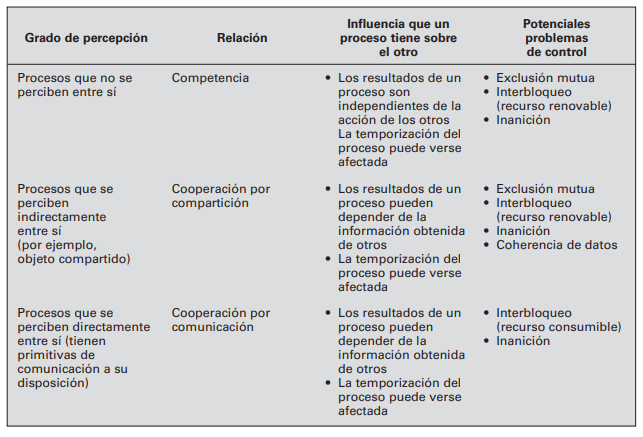
### Dificultades (3)

* **Compartir recursos** globales (variables, disp. E/S).
* Para el sistema operativo es complicado gestionar la **asignación de recursos de manera óptima**.
* **Dificultad de localizar errores** de programación (no se sabe qué proceso cambió el valor de la variable, depende del orden de ejecución).

## Por la concurrencia, el SO tiene que tener en cuenta (4):

* Debe ser capaz de **seguir la pista de los procesos (BCP)**.
* Debe **ubicar y desubicar recursos** (Tiempo de procesador, memoria, archivos y dispositivos de E/S).
* Debe **proteger** los datos y recursos.
* **El funcionamiento de un proceso y el resultado que produzca debe ser independiente de la velocidad a la que suceda su ejecución en relación con la velocidad de otros procesos concurrentes.**

## Interacción entre los procesos concurrentes



## Requisitos de exclusión mutua (6)

* **Sólo un proceso** de todos los demás, puede estar en su sección critica de un mismo recurso compartido.
* Un proceso que se interrumpe en su **sección NO crítica** debe hacerlo **sin interferir con otros** procesos
* Un proceso **no puede ser demorado indefinidamente** (no pude haber interbloqueo o inanición).
* **Si ningún proceso está en su sección crítica, cualquier proceso que solicite entrar debe poder sin demora.**
* **No se deben hacer suposiciones** sobre las velocidades de los procesos ni sobre el número de procesadores.
* Un proceso permanece dentro de su sección crítica sólo por un **tiempo finito** (avisar que salió de la sección critica).

## Herramientas para la concurrencia

* [Exclusión mutua por soporte de hardware.](#_2vor4mt)
* [Semáforos.](#_1au1eum)
* [Monitores.](#_3utoxif)
* [Paso de mensajes.](#_29yz7q8)

## Exclusión mutua por soporte de hardware

### Deshabilitar interrupciones

* En maquina **monoprocesador**, **deshabilitar interrupciones** garantiza la exclusión mutua.
* Se limita la capacidad del procesador, se deshabilita la multiprogramación.
* En maquina es multiprocesador, deshabilitar interrupciones no garantiza la exclusión mutua (se deshabilita solo en un procesador).

### Instrucciones de máquina especiales

* Llevan a cabo acciones sobre una única posición de memoria **con un único ciclo de búsqueda de instrucción (instrucciones atómicas).**
* El acceso a la posición de memoria se le bloquea a toda otra instrucción.

#### Instrucciones

* Test and set (comprueba y establece):

Muestra si el recurso está libre (0), sino está ocupado (1).

Cambia el valor de I (cerrojo) y devuelve un booleano.

Se suele inicializar en 0 para que al primer proceso le devuelve verdadero.

Cuando el proceso termina señaliza al cerrojo en 0 para que lo tome otro proceso.

**boolean testset (int i) {**

**if (i == 0) {**

**i = 1;**

**return true;**

**}**

**else {**

**return false;**

**}**

**}**

* Exchange (Intercambio)

Es un intercambio de llave como los baños en una estación de servicio.

Los procesos comienzan con un valor igual a 1 (k) y el recurso con 0 (b). Cuando van a usar el recurso intercambian k con b. Si ahora k vale 0 puedo usar el recurso. Si otro proceso hace el intercambio cuando se está usando el recurso su k quedará con el valor 1 (es como si no pasó nada) y no podrá usar el recurso.

**void exchange(int registro, int memoria) {**

**int temp;**

**temp = memoria;**

**memoria = registro;**

**registro = temp;**

**}**

El de la caja tiene la frase “Tengo la llave” (0), el primer cliente tiene la frase “No tengo la llave” (1). Entonces realizan un intercambio y el de la caja se queda con “No tengo la llave” (1), cuando venga otro cliente van a hacer el intercambio, pero va a intercambiar “No tengo la llave” (1) y se van a quedar igual.

#### Desventaja

* **Utiliza el procesador** revisando el estado del recurso (**Espera activa**).
* Puede haber **inanición** ya que no hay una lista/fila/cola.
* Puede ocurrir **interbloqueo.**

#### Ventajas

* Es aplicable a **cualquier número de procesos sobre monoprocesador o multiprocesador**.
* Es simple y, por tanto, **fácil de verificar**.
* Puede ser utilizado para dar soporte a **múltiples secciones críticas** (múltiples cerrojos).

## Semáforos

* Es una **variable especial** que se utiliza para señalizar (números enteros o binarios).
* Programa del SO.
* Monoprocesador y multiprocesador.
* Los procesos se ejecutan en modo usuario y hacen llamados al SO para para que cambie el valor de la variable.
* Si un proceso está esperando, el mismo se bloquea (**No hay espera activa, no ocupa al procesador**)
* Cuando un proceso **quiere usar el recurso**, se realiza la **operación** **Wait** (llamada al SO)**, puede usarlo o ser bloqueado**.
* Después del Wait, si el valor es mayor/igual a 0, el proceso toma el recurso.
* Cuando el proceso **deja de utilizar el recurso**, se realiza la **operación** **Signal para despertar** (poner en “Listo”) **a otro proceso**.
* Se utiliza una cola para conocer los procesos esperando por el semáforo. **Semáforo fuerte** donde la cola es FIFO (libre de inanición), y **Semáforo débil** que no especifica como salen los procesos de la lista (posible inanición).

### Semáforo no-binario o general:

La variable tiene un valor entero con solamente 3 operaciones definidas:

* Puede ser **inicializado** a un valor no negativo (si hay 4 ejemplares del recurso, se inicializa con un 4).
* La operación ***semWait***decrementa el valor del semáforo. Si el valor pasa a negativo, el proceso se bloquea.
* La operación ***semSignal*** incrementa el valor del semáforo.

### Semáforo binario:

* Puede ser inicializado con 0 o 1.
* La operación **semWaitB** comprueba el valor. Si es 0, el proceso que consultaba se bloquea. Si el valor es 1, se cambia a 0 y el proceso utiliza el recurso.
* La operación **semSignalB** comprueba si hay un proceso bloqueado. Si lo hay, lo desbloquea. Si no hay, entonces el semáforo se pone en 1.

### Problema Productor/Consumidor (con semáforos no binarios)

**Buffer**: Amortigua las diferencias de velocidades.

* Un proceso está generando algún tipo de datos y **poniéndolos en un buffer***.*
* Un único consumidor está **extrayendo datos de dicho buffer**en un momento dado.
* **Sólo un agente** productor o consumidor **puede acceder al buffer**en un momento dado.
* **Es importante el orden de las preguntas (semáforos),** si primero pido el recurso y luego pregunto si está vacío, nunca se va a agregar un nuevo elemento porque tengo el recurso, pero estoy bloqueado (interbloqueo).

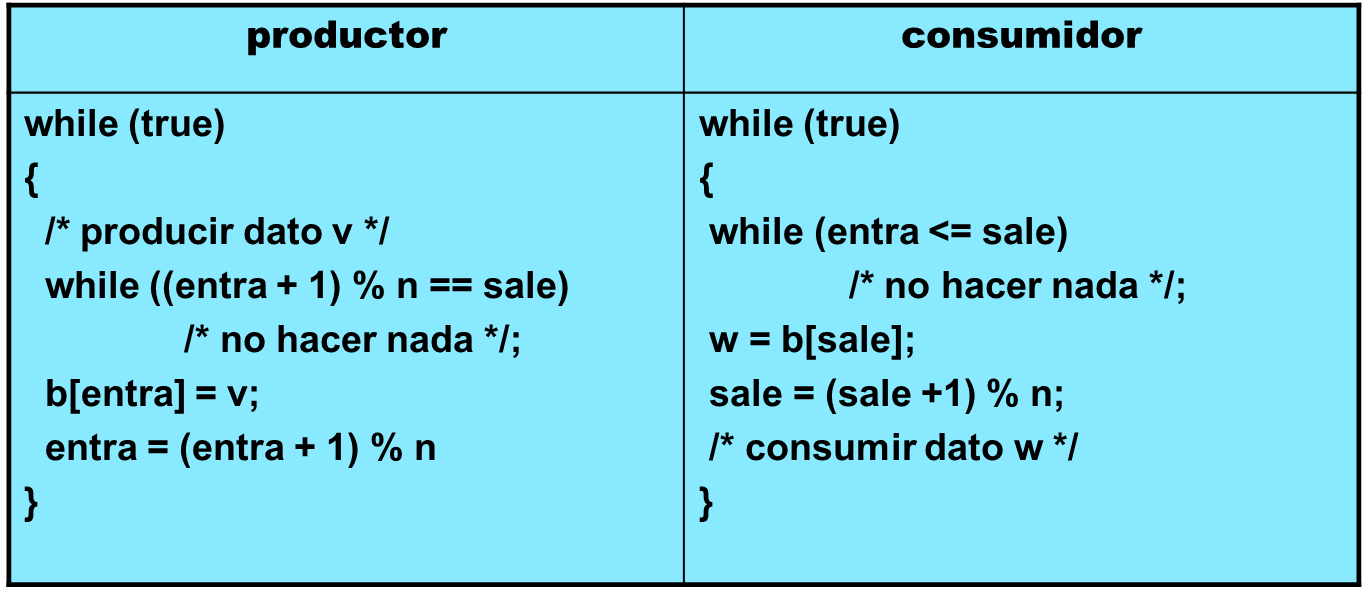
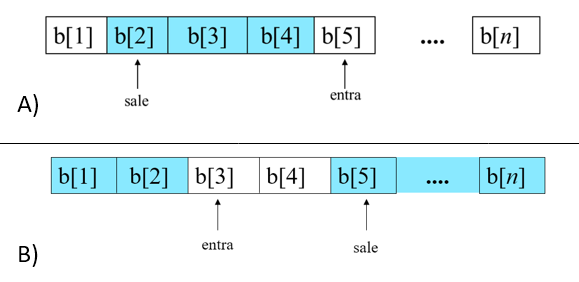
#### Problemas:

1. Tiene que haber exclusión mutua, solo uno de los dos puede estar en el buffer. Porque si no habría problemas con los índices/punteros.
2. El consumidor no puede consumir si el buffer está vacío (n=0).
3. El buffer es finito.

#### Soluciones:

1. Utilización del semáforo no binario “s”, para la exclusión mutua.
2. Utilización del semáforo no binario “n”, para saber si el buffer está vacío (elementos almacenados sin leer).
3. Utilización del semáforo no binario “e”, para saber si hay espacio libre en el buffer.

#### Seudocódigo

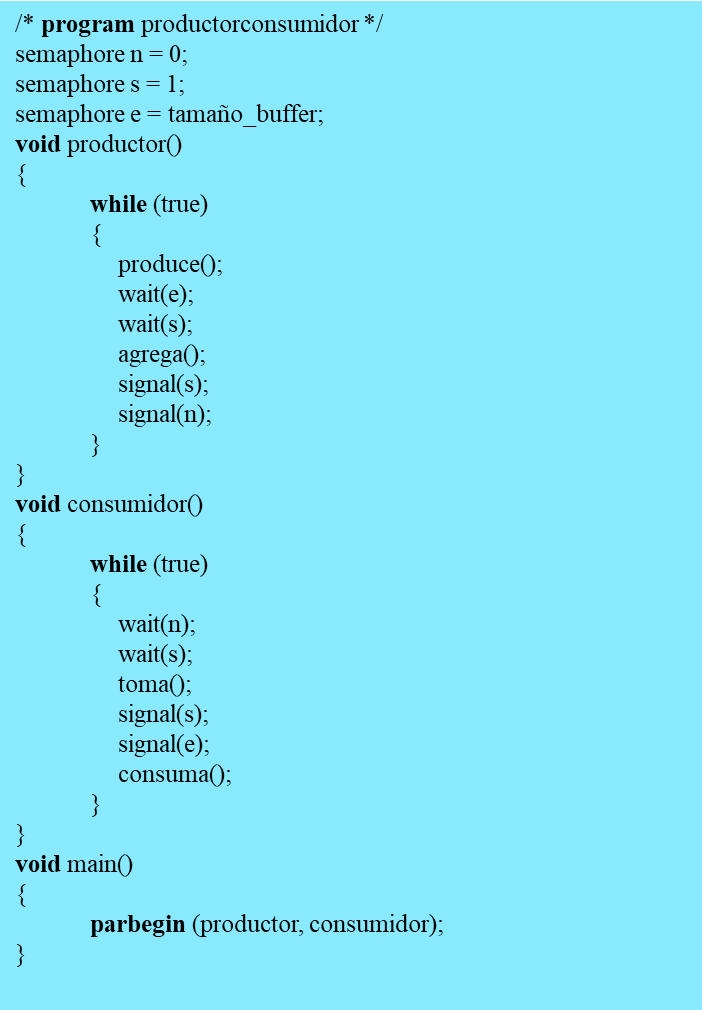
**v =** Elemento a escribir.

**w =** Elemento leído.

**n =** Cantidad de punteros (tamaño del buffer).

**entrada =** Puntero a escribir.

**sale =** Puntero a leer.



## Monitores

Los semáforos son flexibles y potentes, pero puede resultar difícil construir un programa correcto por medio de semáforos, ya que las operaciones semWait y semSignal deben distribuirse por todo el programa.

Una vez que el monitor se programó de forma correcta, puede trabajar con cualquier proceso (Es sencillo de verificar que la sincronización y detectar los fallos). En cambio, para el buen funcionamiento de los semáforos, los procesos tienen que estar bien programados.

Los monitores son una construcción en lenguaje de programación que ofrecen una funcionalidad equivalente a la de los semáforos y son más fáciles de controlar:

* Módulo de software.
* Un proceso entra en el monitor invocando uno de sus procedimientos.
* **Sólo un proceso puede estar ejecutando dentro del monitor al mismo tiempo (exclusión mutua).**
* Si el proceso se bloquea, el monitor es liberado para otro proceso. Luego, el primer proceso puede retomar el monitor justo donde lo dejó.
* **Variables condición**: Variables locales de datos, sólo accesibles por el monitor. Se manipulan con cwait(“variable”) y csignal(“variable”).
* **Función cwait(c):** Suspende al proceso con la condición “c”. El proceso no sale fuera del monitor, sino que pasa a la cola de bloqueados de la condición, que se ubica en la zona de espera del monitor.
* **Función csignal(c):** Retoma la ejecución de algún proceso bloqueado con la condición “c”. Si no hay ningún bloqueado, esta señal se pierde y no se hace nada.
* Las condiciones y las funciones son las que ayudan a la sincronización de los procesos, bloqueando unos y desbloqueado otros (nolleno = falso, entonces el productor se bloquea).

## Paso de mensajes

**Intercambia información**.

Impone la exclusión mutua.

Operación **send (destino, mensaje)**

Operación **receive (origen, mensaje)**

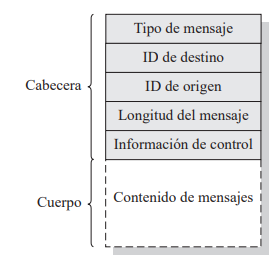
### Sincronización

* **Ambos bloqueantes**: emisor y receptor se bloquean hasta que el mensaje se entrega.
* **Envío no bloqueante, recepción bloqueante:** El receptor se bloquea hasta que el mensaje solicitado llegue. El emisor está libre.
* **Ambos no bloqueantes:** Ninguna de las partes tiene que esperar.

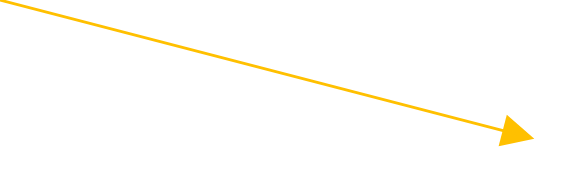
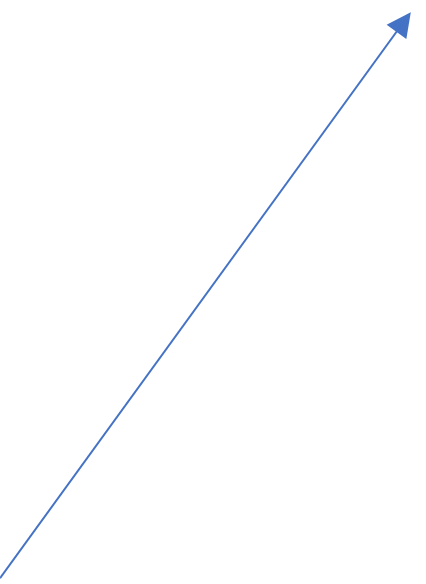
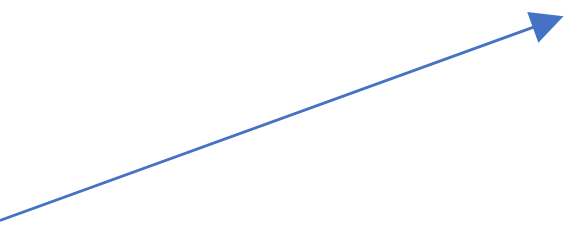
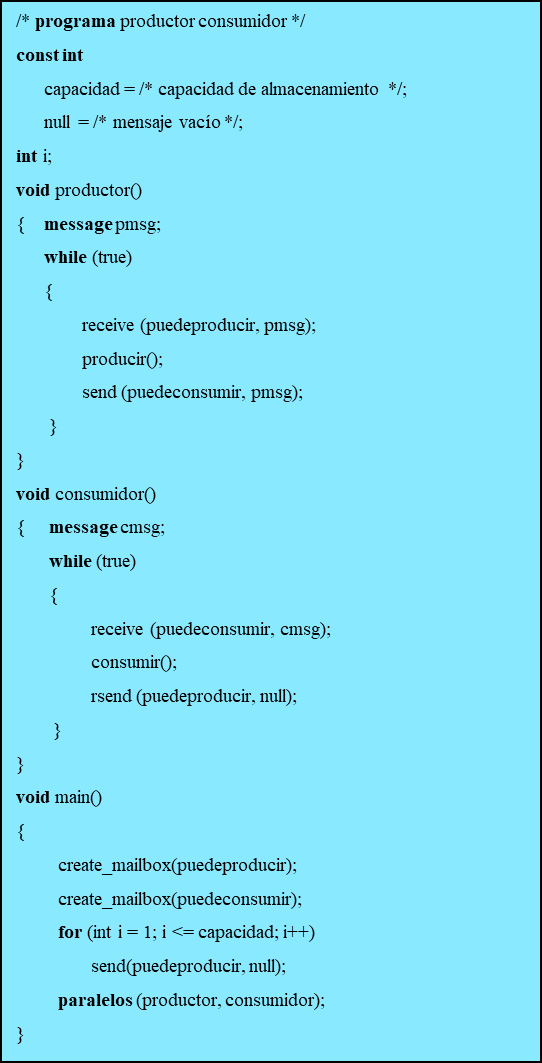
### Direccionamiento

* Direccionamiento directo
  + La primitiva *send* incluye un identificador específico del proceso destinatario.
  + La primitiva *receive* debe conocer con anticipación de qué proceso espera el mensaje.
  + La primitiva *receive* puede usar el parámetro *origen* para devolver un valor, cuando la operación de recepción se completa.
* Direccionamiento indirecto
  + Los mensajes se envían a una estructura de datos compartida, que consiste en colas (se conocen como buzones).
  + Un proceso envía un mensaje al buzón apropiado y otro proceso toma el mensaje del buzón.
  + M:1, 1:1 = Puerto
  + M:N, 1:N = Buzón

### Formato de menaje



### Seudocódigo



Bloqueante, hasta que no tenga un mensaje no produce

El buffer se llena de menajes vacíos, luego “Puede producir” va tomando de a uno y los envía a “Puede producir”. Si no hay más mensajes vacíos, “*receive (puedeproducir, pmsg)*” bloqueará al producto.

La misma lógica para el consumidor.

# Capítulo 6: Interbloqueo e inanición (pág. 257)

## Tipos de recursos

### Recursos reutilizables

* Usados sólo por un proceso en cada momento y no se destruyen después de su uso.
* Los procesos obtienen unidades del recurso que más tarde liberarán para que puedan volver a usarlas otros procesos.
* Procesadores, canales de E/S, memoria principal y secundaria, dispositivos y estructuras de datos como ficheros, bases de datos y semáforos.
* El interbloqueo se produce si cada proceso mantiene un recurso y solicita el otro.

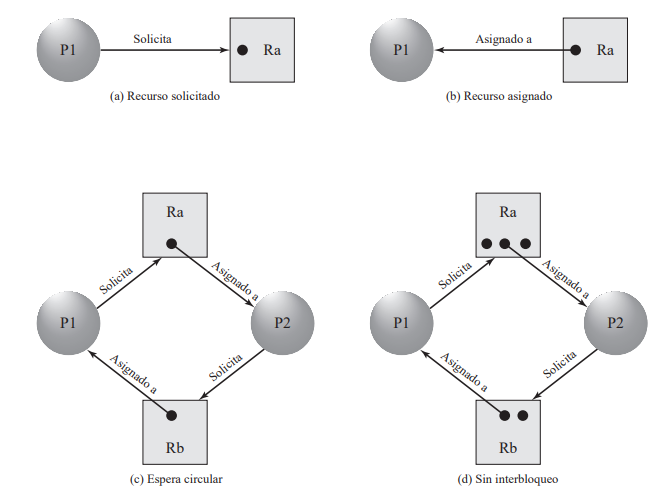
### Recursos consumibles

* Creado (producido) y destruido (consumido).
* Interrupciones, señales, mensajes e información en *buffers* de E/S.
* El interbloqueo puede producirse si la función “Recibe” es bloqueante, en “[Paso de mensaje](#_29yz7q8)”.
* Puede darse una rara combinación de eventos que cause el interbloqueo (por prioridades u otra cosa).

## Grafo de asignación de recursos

Es la representación gráfica del estado del sistema en lo que se refiere a recursos y procesos.

Cada punto en los recursos significa una instancia de los mismos.



## Interbloqueo

* **Es el bloqueo permanentede un conjunto de procesos** que compiten por recursos del sistema o se comunican entre sí.
* No hay una solución eficiente (matar a un proceso).
* Involucran necesidades conflictivas que afectan a los recursos de dos o más procesos.

### Ejemplo

Es un ejemplo mío.

Un electricista realiza una instalación eléctrica de 2 focos, pero comete un error. El foco A para apagarse tiene que prender el foco B, pero el foco B para prenderse se tiene que apagar el foco A.

Otro ejemplo: Experiencia laboral al buscar trabajo.

## Condiciones para el interbloqueo

* **Exclusión mutua** (Condición necesaria y NO suficiente): Tan sólo un proceso puede usar un recurso en cada momento.
* **Retención y espera** (Condición necesaria y NO suficiente):Un proceso puede mantener los recursos asignados mientras espera la asignación de otros procesos.
* **Sin expropiación** (Condición necesaria y NO suficiente):No se puede forzar la expropiación de un recurso a un proceso que lo posee.
* **Espera circular** (Condición necesaria y suficiente):Existe una lista cerrada de procesos, de tal manera que cada proceso posee al menos un recurso necesitado por el siguiente proceso de la lista. Para que esta condición se cumpla, se tienen que cumplir las primeras 3 condiciones.

## Estrategias para tratar el interbloqueo

* [Prevención](#_393x0lu)
* [Predicción](#_1o97atn)
* [Detección](#_488uthg)

## Prevención (antes de y sin saber que pasará)

Poco eficiente, pero bien efectiva.

Técnicas para evitar que aparezca por lo menos 1 de las condiciones:

* **Retención y espera:** *<condición>*
  + Un proceso debe solicitar todos sus recursos a la vez (reduce el rendimiento) *<técnica>*.
* **Exclusión mutua**:
  + Debe existir para la concurrencia.
* **Sin expropiación** (perdida de trabajos)
  + Un proceso debe liberar los recursos que tiene asignado cuando pide nuevos recursos y tiene que esperar.
  + El sistema operativo puede expropiar a un proceso y obligarle a liberar sus recursos para dárselos a otro.

En ambos casos habría que guardar un tipo de contexto de los recursos para ser restaurado cuando se devuelven al proceso original.

* **Espera circular**
  + Define un orden lineal entre los distintos tipos de recursos. Si a un proceso se le asigna un recurso; posteriormente puede pedir solo recursos cuyo orden sea posterior al que ya tiene (no sirve con los recursos consumibles). Muy usada en Unix.

## Predicción (mientras sucede busco el camino bueno)

Se deciden dinámicamente si la petición de un recurso puede causar interbloqueo.

Se requiere el conocimiento de las futuras solicitudes de recursos del proceso.

Son un poco más eficientes que las de prevención.

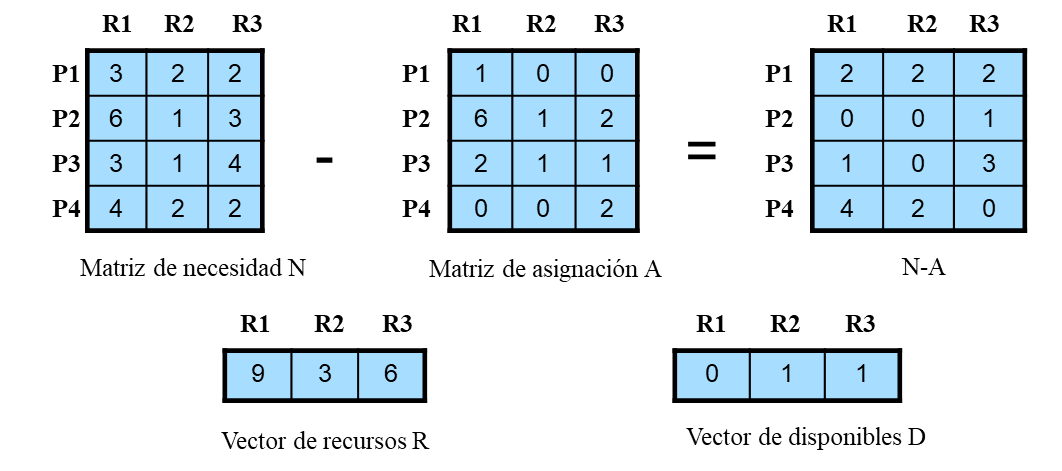
Técnicas para evitar el interbloqueo:

* No iniciar un proceso si sus demandas pudieran llevar al interbloqueo (muy drástico).
* No conceder una petición adicional de un recurso por parte de un proceso si esta asignación pudiera provocar un interbloqueo ([algoritmo del banquero](#_2ne53p9)).

### Algoritmo del banquero

Se ejecuta **cada vez que hay un nuevo pedido de recursos.**

* El estado del sistema refleja la asignación actual de recursos a procesos.
* **Un estado seguro** es aquél en el que hay al menos una secuencia de asignación de recursos a los procesos que no implica un interbloqueo
* **Un estado inseguro** no necesariamente implica un futuro interbloqueo pero da la posibilidad.



**Matriz de necesidad N:** La demanda máxima de los procesos para cada recurso.

**Matriz de asignación A:** Son los recursos que tengo asignados ahora.

**Vector de recursos R:** Total de instancias de cada recurso.

**Vector de disponibilidad D:** Los recursos sobrantes que tengo ahora.

**N-A=** Recursos que falta por asignar. Demanda futura

Para conseguir un estado seguro, tengo que intentar satisfacer a un proceso para que se complete y me libere los recursos y poder asignarlos a otros y así hasta hacerlo con todos. Vendría a ser como una simulación de cómo podrían terminar.

### Ventajas

* No hay expropiación.

Hay más libertades para los procesos.

### Desventaja

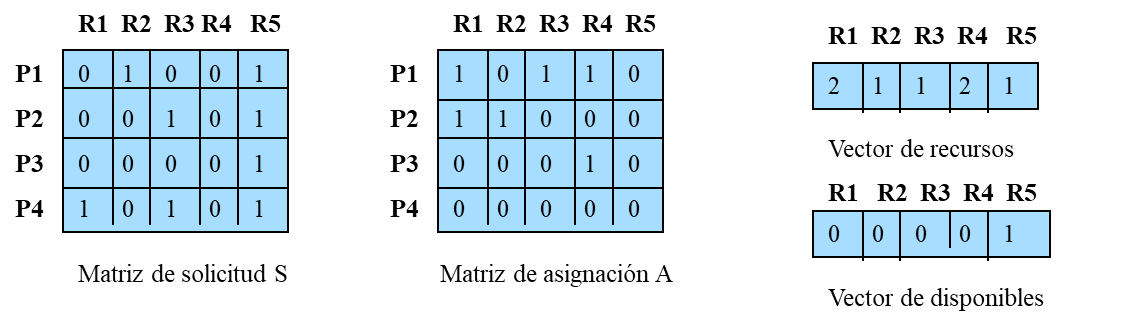
* Deben establecerse por anticipado los requisitos máximos de recursos de cada proceso (problema serio)
* Los procesos involucrados deben ser independientes, es decir, el orden en el que se ejecutan no debe importar.
* Debe haber un número fijo de recursos que asignar (para hacer las matrices).
* Ningún proceso puede terminar mientras mantenga recursos.

## Detección

Encontrar el interbloqueo ya existente.

Es la **más eficiente en rendimiento** del sistema porque a los procesos se le dan los recursos que van pidiendo y cada tanto (de **forma sincrónica**) se ejecuta el algoritmo para solucionar los interbloqueos formados.

El problema que esta estrategia mata a los procesos, por lo tanto, no se puede usar en cualquier sistema. No sería conveniente usarlo en un avión, pero no habría problema en usarlo en Windows.



**Matriz solicitud S**: Es la demanda actual de recursos.

**Matriz de asignación A:** Los recursos asignados hasta ahora.

**Vector recursos**: Recursos totales.

**Vector de disponibles:** Recursos sin asignar.

El proceso 4 no puede estar en el interbloqueo porque no tiene ningún recurso asignado (no cumple con “Retención y espera”).

Se puede satisfacer a P3, puede completarse, por lo tanto, devuelve los recursos que tiene asignado.

El interbloqueo sucede con P1 y P2, ya que los recursos que devuelve P3 al completarse, no alcanza para satisfacer a P1 y P2.

**Es muy parecido al algoritmo del banquero, el procedimiento es el mismo, nada más que se utiliza la peticiones de ahora, no las futuras o las máximas.**

### Estrategias de recuperación

#### Todos los procesos

* **Abortar todos** los procesos involucrados en el interbloqueo (matar a P1 y P2). Hay perdida de trabajo.
* **Retroceder cada proceso en interbloqueo** a algún punto de control, pero el interbloqueo original puede volver a ocurrir.

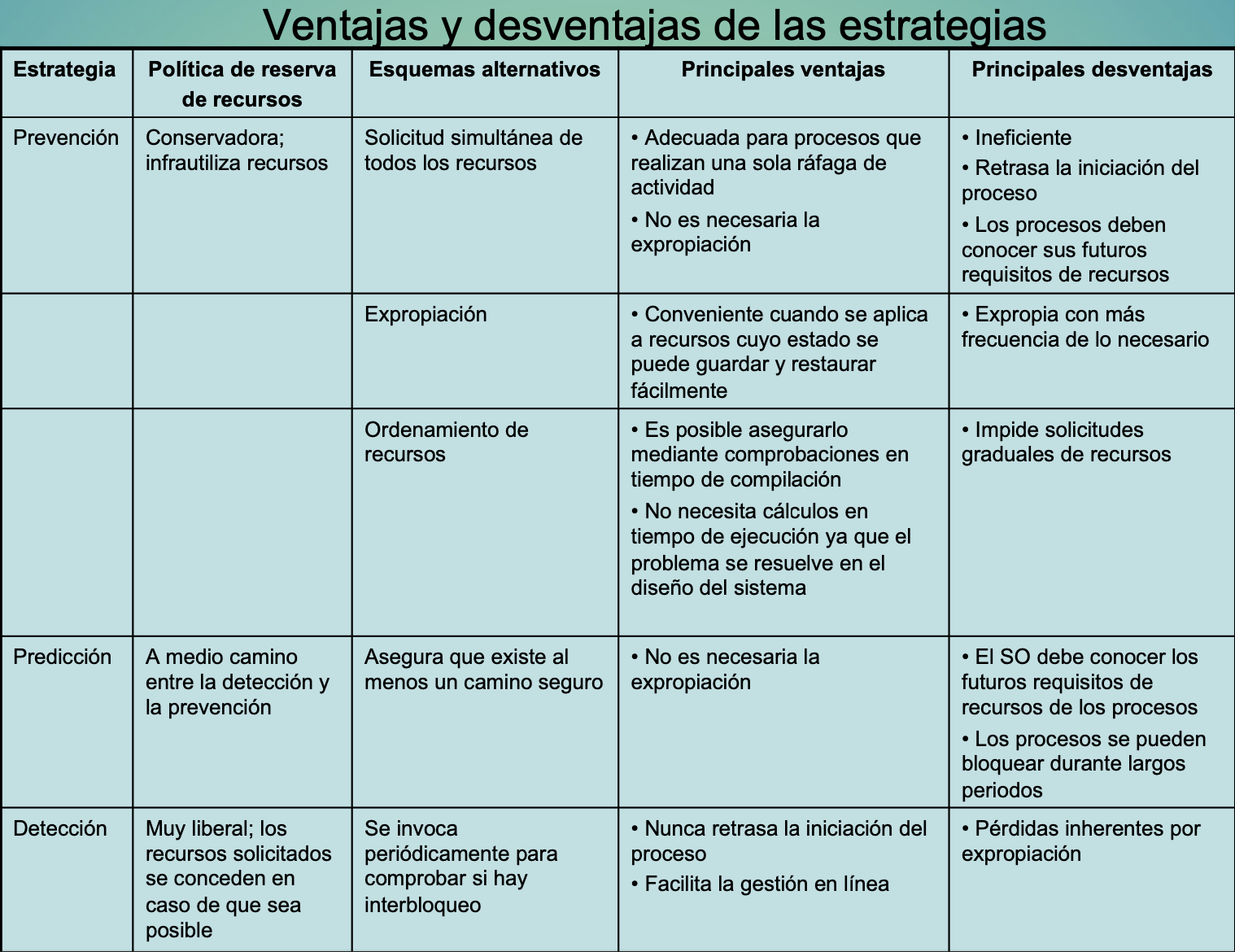
#### Procesos seleccionados

Según los [Criterios de selección de procesos](#_12jfdx2):

* Abortar sucesivamente los procesos en el interbloqueo hasta que éste deje de existir (**matar de a uno**).
* **Expropiar sucesivamente los recursos** hasta que el interbloqueo deje de existir (tienen que volver a algún punto de control).

### Criterios de selección de procesos (5)

* El de menor tiempo de ejecución.
* El de menor cantidad de salidas producida hasta ahora.
* El mayor tiempo restante estimado.
* El menor número total de recursos asignados hasta ahora.
* La menor prioridad.



Ver problema de los “Filósofos comensales”.

# 

# Capítulo 7: Gestión de memoria (pág. 305)

La memoria principal está dividida en 2, parte del monitor/núcleo y otra parte para el usuario. La parte de usuario se subdivide para cada proceso y el encargado es el SO. **Esta tarea se la llama gestión de memoria.**

## Diccionario

**Fragmentación externa**: Es generada cuando durante el reemplazo de procesos quedan huecos entre dos o más procesos de manera no contigua y cada hueco no es capaz de soportar ningún proceso de la lista de espera. Una solución es **la compactación,** pero consume procesador y tiempo.

**Fragmentación interna:** Es generada cuando se reserva más memoria de la que el proceso va realmente a usar. Estos huecos no se pueden compactar para ser utilizados.

**Página:** es el almacenamiento virtual, un bloque de longitud fija que dispone de una dirección virtual y que se transfiere como una unidad entre la memoria principal y la memoria secundaria.

**Marco de Página**: bloque de memoria principal contiguo y de longitud fija que se usa para contener una página.

**Segmento**: en la memoria virtual, un bloque que tiene una dirección virtual. Los bloques de un programa pueden ser de longitud desigual y pueden ser incluso de longitud variable dinámicamente.

## Requisitos (5)

### Reubicación

Cuando un programa se carga en la memoria se determinan las direcciones, con el paso del tiempo pueden ir cambiando:

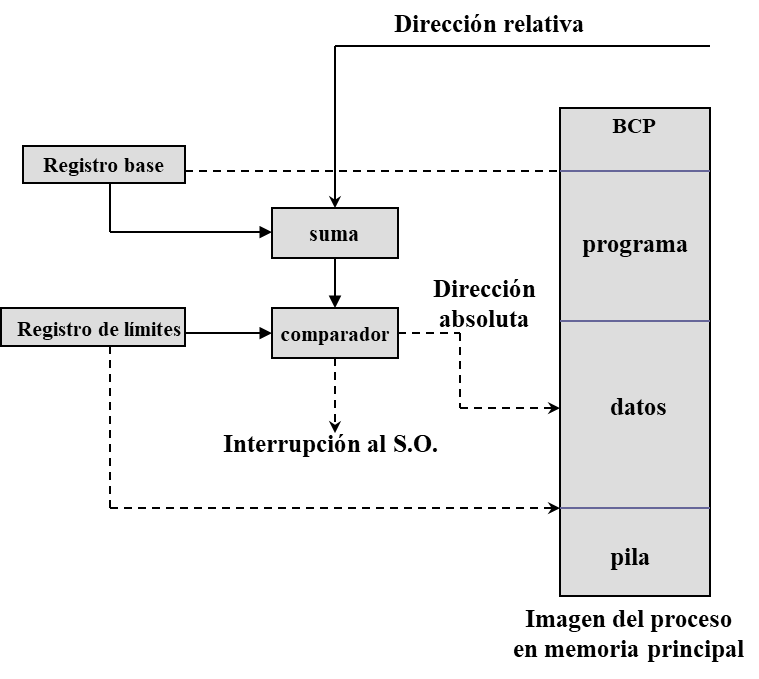
* + Un proceso puede ocupar diferentes particiones durante la ejecución (por el swap, se puede descargar a disco y volver a cargar en una dirección de memoria diferente).
  + La compactación también causa que un programa tenga que ocupar diferentes particiones (se va trasladando para no dejar huecos).

Todo esto se hace en tiempo de ejecución (tiene que ser al instante y muy rápido).

#### Tipos de direcciones (3)

Se necesita un mecanismo de traducción porque los programas hablan con direcciones virtuales/lógicas, mientras que el procesador habla con las direcciones reales/físicas.

* **Direcciones virtuales (lógicas)**
  + Direcciones a las cuales hacen referencia los **procesos** independientemente de la asignación actual
  + Mecanismos de hardware hacen la traducción a direcciones reales (físicas).
* **Direcciones relativas**
  + Direcciones expresadas en relación a algún punto conocido.
  + Para poder traducir entre virtual y real. Si la real empieza en la posición 40 y el programa solicita la posición 12 de la virtual, la dirección relativa 0 es 40 entonces 12 es 52, el registro base es 40 (referencia).
  + Registro de limites: requisito de protección con respecto a que no acceda en memoria de otro proceso.



* **Direcciones reales (físicas)**
  + Direcciones actuales de memoria.

### Protección

* Los procesos no deberían poder referenciar localidades de memoria de otros procesos sin permiso.
* Imposible chequear direcciones dentro de los programas puesto que se pueden reubicar (por el primer puntito de reubicación).
* Se debe chequear durante la ejecución.

### Compartición

* Permitir a varios procesos acceder a la misma porción de memoria
* Ejemplos:
  + Acceder a la misma copia del programa en lugar de tener su propia copia.
  + Acceder a una estructura de datos compartida (procesos del mismo programa o hilos).

### Organización lógica

* Los programas se escriben en módulos.
* Diferentes grados de protección para los distintos módulos de un programa (solo lectura, solo ejecución)
* Compartición de módulos

### Organización física

* Tarea de mover información entre los niveles de memoria: principal y secundaria.
* No es deseable dejar esta responsabilidad al programador. Ejemplo: Cuando no es suficiente la memoria principal disponible para un programa y sus datos, una solución es la técnica de superposición, pero malgasta tiempo del programador.
* En un entorno multiprogramado, el programador no conoce en tiempo de codificación, cuánto espacio estará disponible o dónde.

## Técnica de gestión de memoria

Es posible combinar segmentación con paginación.

Evolución de las técnicas (es en ese orden):

### Particionamiento fijo

En si no existió, pero es la base para el particionamiento dinámico. Consiste en:

* **División de la memoria en particiones de tamaño fijo** (Solo se puede colocar cualquier proceso cuyo tamaño sea menor o igual que el de la partición).
* **La imagen del proceso es contigua** (no se divide en varias direcciones).
* Si todas las particiones están llenas, el SO puede sacar a un proceso de cualquiera de las particiones (swap) y cargar otro.

#### Problemas

* Para un programa más grande que una partición se usaba la técnica de superposición (ir sobrescribiendo el programa).
* Cualquier programa ocupa una partición entera. Esto se llama **fragmentación interna** (espacio asignado pero que no se ocupa en su totalidad)**.** Uso de la memoria ineficiente [*<Diccionario>*](#_3mj2wkv).

#### Mejora: Particiones fijas de diferentes tamaños

Es una mejora, pero el problema no es solucionado. Sigue habiendo fragmentación interna, pero menor cantidad. Es posible cargar programas grandes. Una desventaja es que hay que saber qué cosa ubicar en dónde.

#### Algoritmo de ubicación

* **Particiones del mismo tamaño**:
  + Algoritmo trivial, se asigna cualquier espacio.
* **Particiones de tamaño diferente**:
  + Traducción y carga absoluta (Cola por partición): Asignar a cada proceso a la partición más pequeña dentro de la cual cabe. Minimiza la fragmentación interna. Hay particiones libres de un tamaño mayor mientras que hay procesos esperando por particiones pequeñas (mucha cola para 2MB, pero el de 4MB no tiene nadie).
  + Traducción y carga con reubicación (Cola única): Mayor aprovechamiento en uso de todas las particiones. Mayor fragmentación.

#### Problemas remanentes

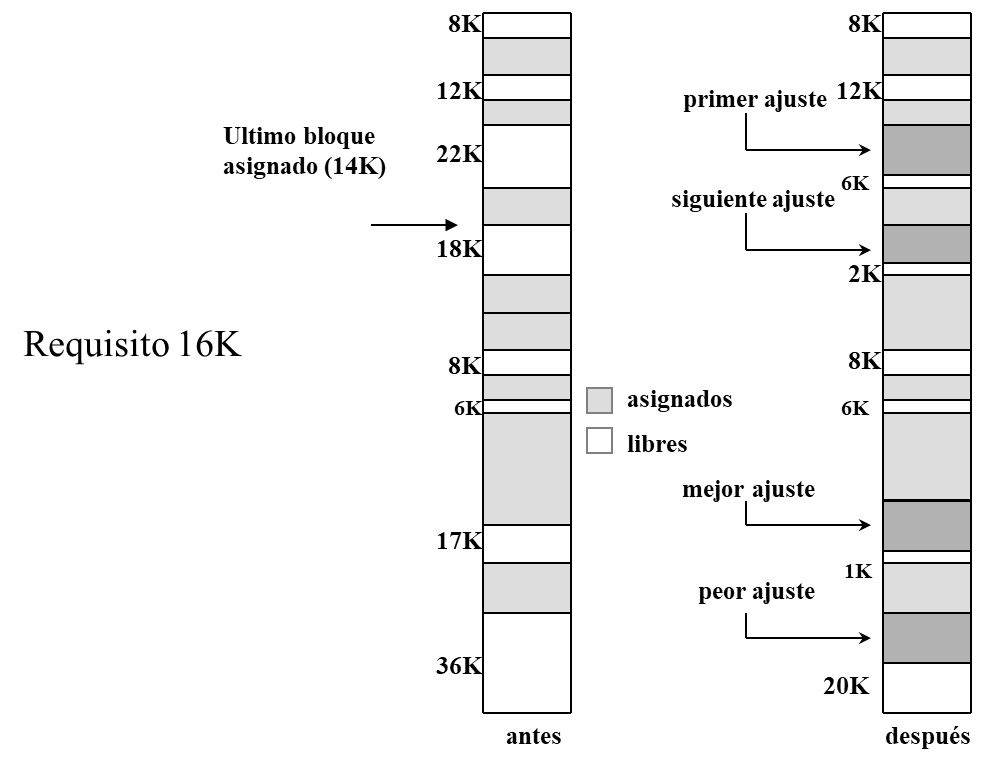
* El número de particiones limita el número de procesos activos.
* Muchos procesos pequeños no utilizan el espacio eficientemente (en todo tipo de particionamiento fijo).

### Particionamiento dinámico

* Las particiones son de tamaño y número variable.
* Los procesos ocupan tanto espacio como necesiten, no hay fragmentación interna.
* Se van generando huecos, esto se llama **fragmentación externa** [*<Diccionario>*](#_3mj2wkv)**.**
* Se debe usar **compactación** que malgasta tiempo de procesador y requiere capacidad de reubicación dinámica para disminuir la fragmentación externa (mueve los programas de su lugar y los amontona a todos).

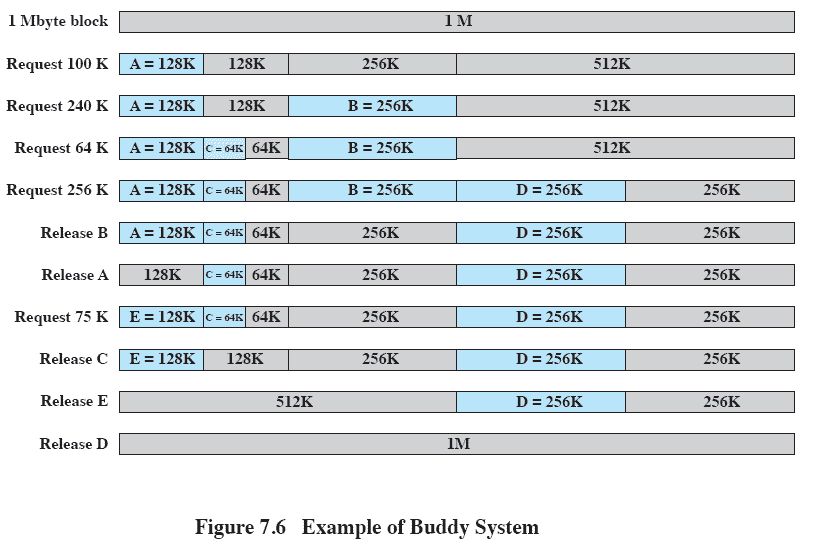
#### Algoritmo de ubicación (4)

* Estrategia del **mejor ajuste**
  + Asigna el menor hueco disponible.
  + Al quedar huecos más pequeños, hay que compactar más frecuentemente.
  + Hay que analizar la memoria para saber los huecos que hay (una lista ordenada por tamaño con la dirección y tamaño del hueco).
* Estrategia del **peor ajuste**
  + Asigna el mayor hueco disponible.
  + Se necesita menos compactación.
  + Hay que analizar la memoria para saber los huecos que hay (una lista ordenada por tamaño con la dirección y tamaño del hueco).
* Estrategia del **primer ajuste**
  + Asigna el primer hueco disponible (FIFO), siempre desde el comienzo.
  + Más rápida al principio, pero cuando comienzan a aparecer huecos, hay que ir viendo uno por uno para saber si entra.
* Estrategia del **siguiente ajuste**
  + Empieza a buscar desde donde terminó la búsqueda anterior, un puntero.
  + Se fragmenta más el gran espacio libre del final de la memoria.
  + Se requiere compactación más frecuente que la anterior.



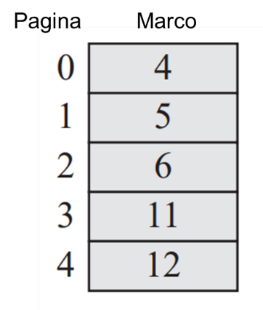
### Estrategia/sistema buddy (colegas):

* Puede ser combinación de particionamiento fijo y dinámico.
* La misma lógica que Subnetting (divida las direcciones IP).
* Aprovecha la ventaja del sistema binario.
* Cuando hay un pedido de asignación se va dividiendo la memoria a la mitad (todos múltiplos de 2, a esto se los llaman colegas) hasta que el tamaño sea el mínimo (70kb en 128kb y no en 256kb).
* Cuando se desocupa un espacio y el colega está desocupado, se unen para hacer uno más grande (128 y 128 pasarían a uno solo de 256). Esto no se hace siempre porque requiere de procesador, se puede hacer cada x tiempo o un pedido de gran tamaño.

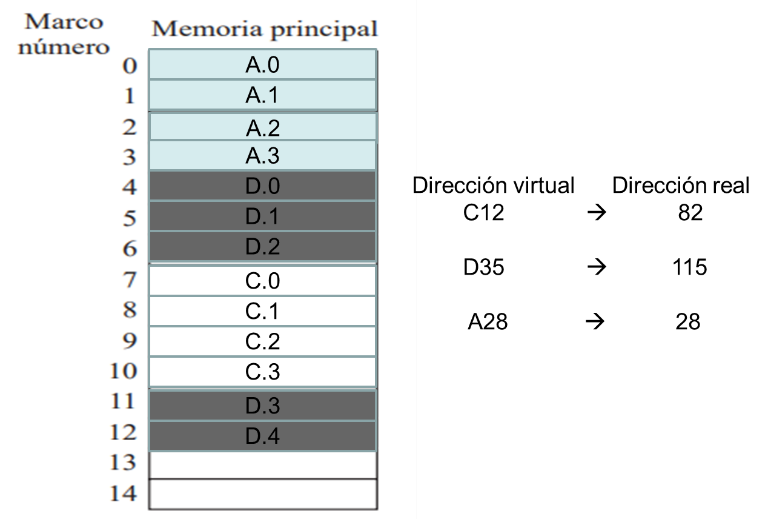
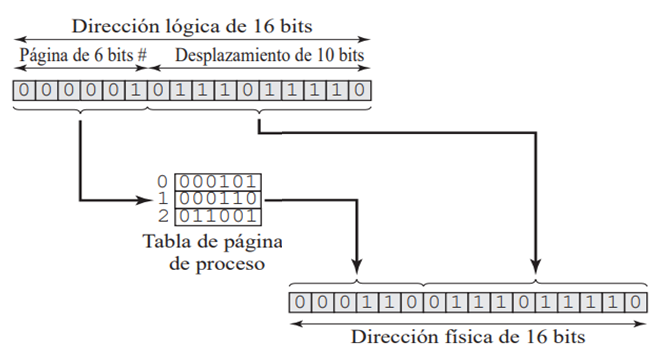


### Paginación sencilla

* La **memoria principal** se divide en un número de **marcos** relativamente pequeños de igual tamaño [*<Diccionario>*](#_3mj2wkv).
* Cada **proceso** se divide en un número de **páginas** del mismo tamaño que los marcos [*<Diccionario>*](#_3mj2wkv).
* Un proceso se carga colocando todas sus páginas en marcos disponibles, no necesariamente contiguos.
* **Se disminuye la fragmentación interna** (proceso de 15kb, los marcos de 4kb por lo tanto las páginas son 4kb, 4kb, 4kb y 3kb, entonces voy a tener 3 marcos al 100% y uno que me sobra 1kb, pero no se puede asignar a otro proceso).
* **El requisito protección está incluido, porque el tamaño de página, que sería el desplazamiento, es el mismo para todas, por lo tanto, no te podés pasar al espacio de otro proceso (no te dan los bits).**
* Contra más páginas y marcos de menor tamaño menor será la fragmentación interna, pero aumentará el computo al crear y mantener tablas de mayor tamaño.
* **No existe fragmentación externa.**
* El sistema operativo mantiene **una lista con los marcos libres.**
* El sistema operativo mantiene **una tabla de páginas por cada proceso**: Contiene el número de marco correspondiente a cada página en el proceso.



* La dirección de memoria para el proceso (**virtual**) está dada por: el número de página y el desplazamiento dentro de la página. Para la traducción se utiliza la tabla de páginas, y gracias al sistema binario la bases de cada marco empiezan con 00000… .

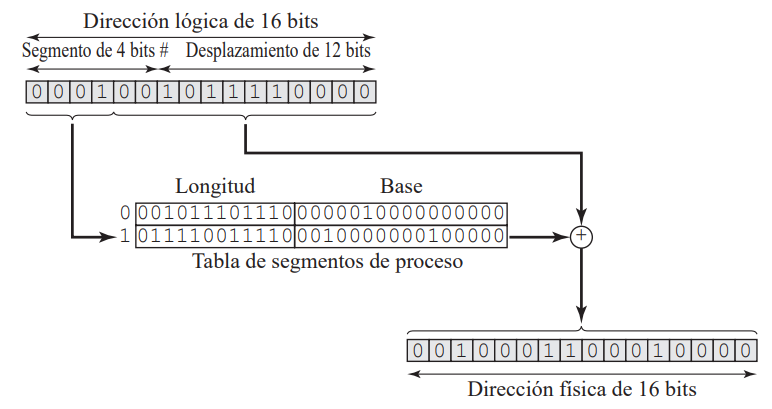
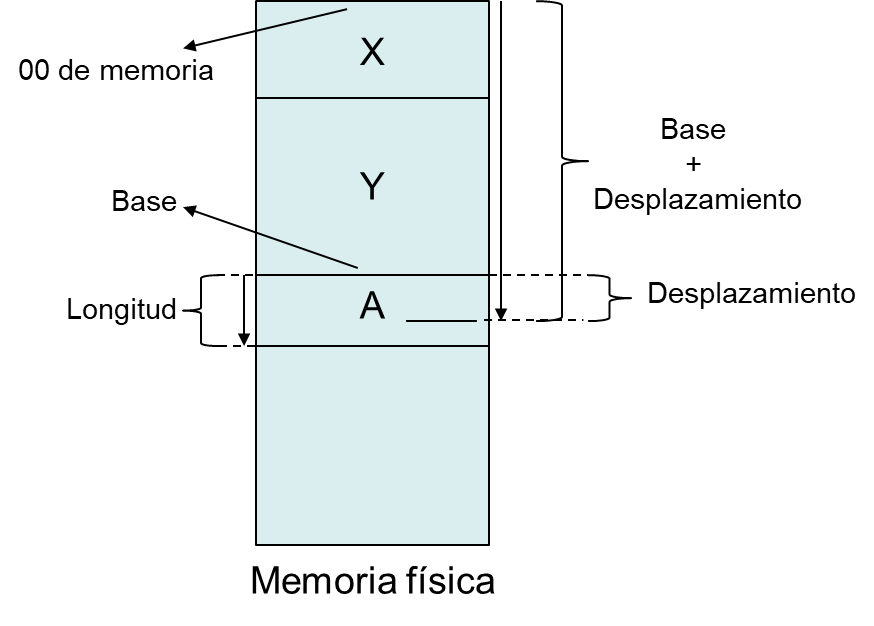
 

Para hacer la traduccion solo hay que reemplazar con los valores de la tabla (es rápido), el desplazamiento siempre será el mismo.

Un ejemplo puede ser un libro digital: Las paginas del PDF son la direccion virtual y las paginas del libro son la direccion virtual. Las paginas de los libros suelen empezar por la 8 port el indice, tapa, bibliografia, etc. Direccion relativa es la 8.

### Segmentación sencilla

* La **memoria no se divide**, es como un único hueco.
* **Cada proceso se divide en un número de segmentos** que no necesariamente tienen que ser del mismo tamaño [*<Diccionario>*](#_3mj2wkv).
* Similar a particiones variables/dinámico, pero no necesitan estar en forma contigua.
* La dirección se expresa como número de segmento y desplazamiento.
* **La tabla de segmentos para cada proceso está formada por**: número de segmento (índice), longitud (tamaño del segmento) y la base (la dirección en donde empieza el segmento en la memoria física).
* Supongo que hay una tabla con las direcciones libres.

* Para traducir tengo que:
  1. Buscar el segmento en la tabla.
  2. Comparar que el desplazamiento no sea mayor que la longitud (para no salirme e ir a otro proceso), esto cumple con el requisito de protección.
  3. Sumar la “base” más el “desplazamiento” (donde empiezo y cuanto me muevo).

### Paginación con memoria virtual

[Capítulo 8](#_gtnh0h)

### Segmentación con memoria virtual

[Capítulo 8](#_30tazoa)

# Capítulo 8: Memoria virtual (pág. 339)

## Diccionario

**Conjunto residente**: Porción inicial del programa y de datos sobre la cual acceden las primeras instrucciones y que se encuentran en memoria principal.

**Fallo de página:** Se origina cuando se busca una pág. en la tabla, pero esta está en M. virtual, o cuando se busca una página en la tabla y el bit de presencia está en 0.

## Ejecución de un programa

1. El sistema operativo trae a memoria principal el **conjunto residente** [*<Diccionario>*](#_21od6so).
2. Se genera una interrupción cuando no se encuentra la dirección lógica en memoria principal. Fallo de acceso a memoria o **fallo de página**).
3. Se realiza la petición de E/S para traer el fragmento de disco a memoria y se bloquea el proceso.
4. Mientras, se ejecuta otro proceso.
5. Una vez finalizado la E/S el proceso se pone en Listo.

## Ventajas de fragmentar

Ya sea paginación o segmentación:

* **Más procesos se pueden mantener en memoria principal** porque solamente se cargan algunos fragmentos de cada proceso.
* **Utilización más eficiente del procesador** porque hay más procesos en Listo.
* **Un proceso puede ser más grande que toda la memoria principal.** Pude ser tan grande como la memoria de disco.
* **Se cargan en memoria los fragmentos que se necesitan.**
* **Se ahorra tiempo porque no se cargan fragmentos que no se usan.**
* **Asignación no contigua.**
* No es necesario que todos los segmentos o páginas estén en memoria durante la ejecución.

## Hiperpaginación (thrashing)

El procesador gasta la mayor cantidad de su tiempo intercambiando los bloques en lugar de ejecutando las instrucciones.

* Intercambio hacia afuera de un bloque de un proceso justo antes de que se necesite, por lo tanto, hay que llamarlo nuevamente.

## Principio de cercanía/proximidad (anticipo)

Para que la memoria virtual funcione:

* Las referencias a datos y programas dentro de un proceso tienden a agruparse.
* Solamente algunas partes de un proceso se necesitarán por un periodo breve de tiempo.
* Es posible adivinar inteligentemente cuáles bloques se necesitarán en el futuro, hay una gran posibilidad de que sean las primeras páginas.

* **Localidad espacial:** tendencia a referenciar localidades cercanas entre sí (Secuencias lineales de código, recorridos de vectores, definiciones de variables afines cercanas unas a otras).
* **Localidad temporal:** tendencia a referenciar la misma posición varias veces durante breves intervalos (ciclos, subrutinas, pilas, variables utilizadas para cuenta y totalización).

## Soporte necesario para memoria virtual

* El hardware debe soportar paginado y/o segmentado (ej: registro del origen de la tabla de páginas, interpretar interrupción por fallo de página).
* El SO debe ser capaz de manejar los movimientos de páginas y/o segmentos entre memorias primaria y secundaria.

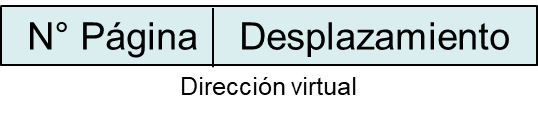
## Paginación con memoria virtual

La asignación de marco para la página es totalmente trivial (son todos iguales no importa donde lo pongo).

**Cada proceso tiene su propia tabla de páginas** que contiene el número de marco de la página correspondiente en memoria principal:

* Pueden ser muy grandes y necesitar mucha memoria.
* Las tablas, si son muy grandes, también se pueden almacenan una parte en memoria virtual.
* Cuando un proceso se ejecuta, parte de su tabla está en memoria principal.
* **Bit de Presencia (P)**: Se necesita un bit para indicar si la página está (1) o no (0) en memoria principal. Si es 0, provoca una interrupción de fallo de página (mecanismo de hardware).

* **Bit de Modificación (M)**: Se necesita un bit para indicar si la página ha sido modificada desde la última vez que fue cargada en memoria. Si no ha habido cambio, la página no necesita volver a escribirse en el disco cuando se quita de memoria principal, se utiliza la que ya está escrita y evito hacer una escritura en disco que es muchísimo más lenta.



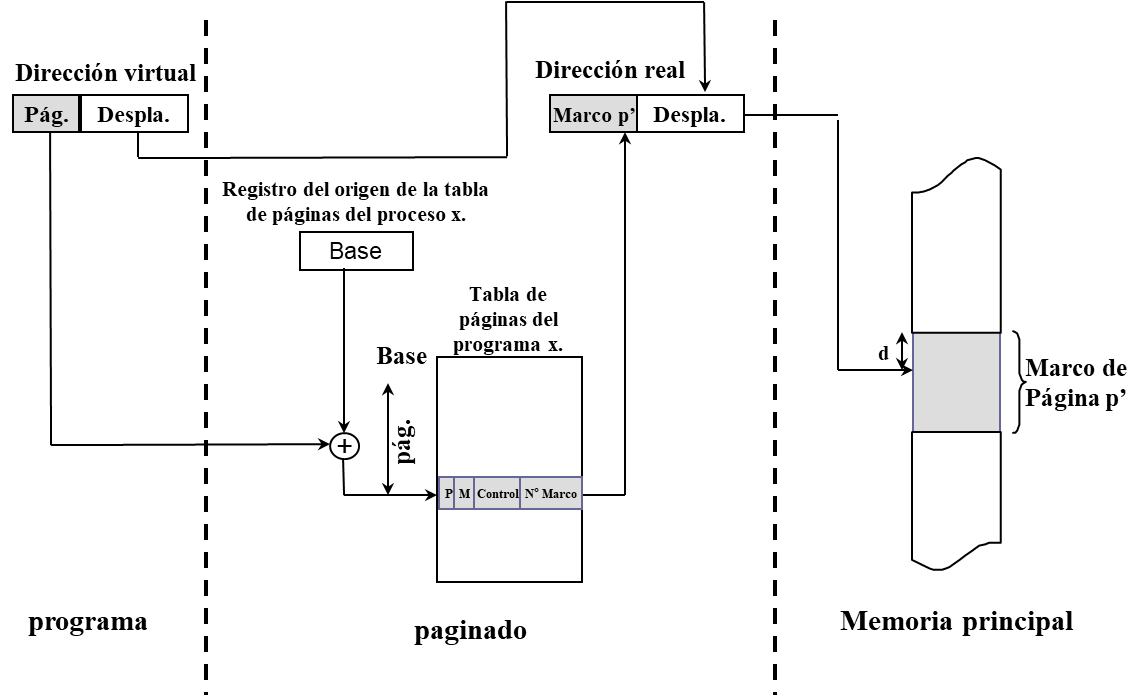


* En la tabla habrá: Cantidad pág. >= cantidad marco (se utiliza el bit P).

Otras aclaraciones:

* Hay un registro que contiene la dirección de comienzo de una tabla y que dicha tabla se encuentra en memoria principal. El registro se almacena en el BCP.
* Cada referencia a una dirección virtual puede causar dos accesos a memoria física.
  + Uno para leer la tabla.
  + Uno para leer el dato.
* Para mejorar esto se usa una cache especial para las entradas de las tablas llamada [TLB (Translation Lookaside Buffer)](#_1fyl9w3).

### Traducción de direcciones en sistemas de paginado (Se evalúa)



Se busca la base de la tabla de páginas en el registro y luego se suma la base y la pág. para encontrar la referencia. Primero compruebo el bit de presencia; si está en 1, puedo mirar el marco porque sé que está en memoria, y ahora puedo buscar con la dirección real. Si el bit de presencian fuese 0, se levanta una interrupción de fallo de página y haría todo lo de E/S dicho en títulos anteriores.

Esta traducción tiene que ser extremadamente rápida ya que se está accediendo 2 veces a memoria.

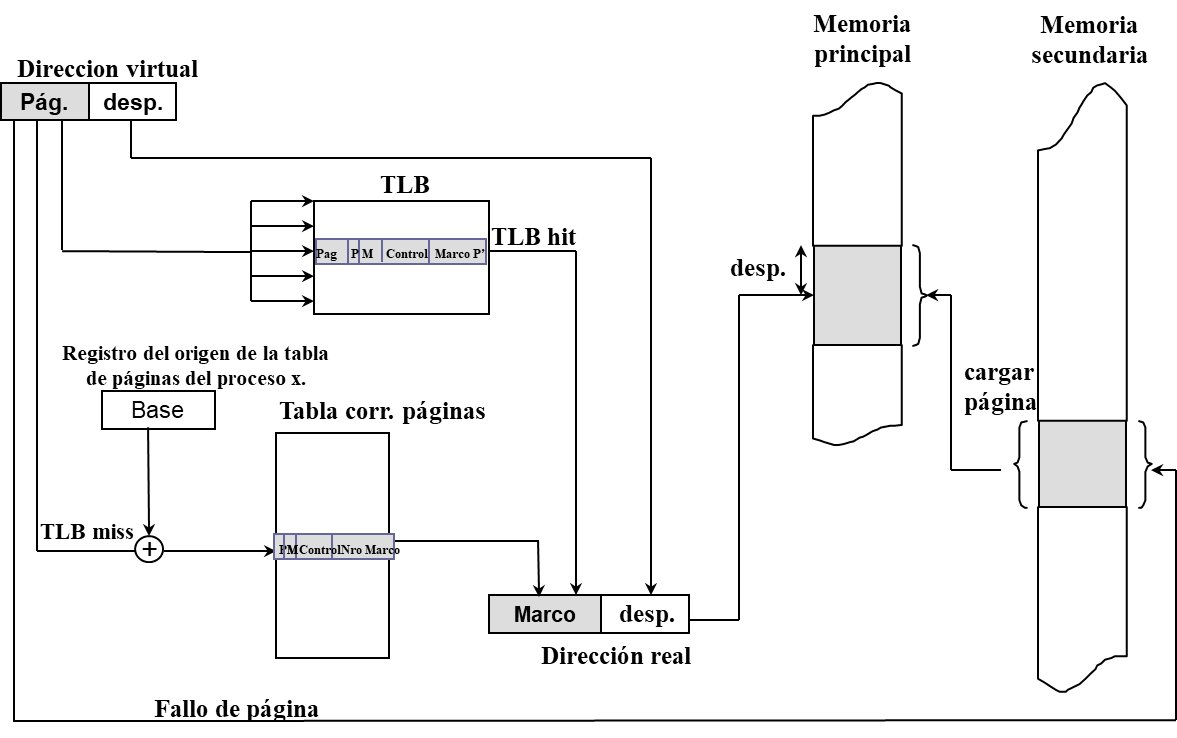
## Cache TLB (pág. 349)

También llamado Buffer de traducción anticipada:

* **Contiene las entradas de la tabla de páginas que han sido usadas más recientemente¸** al momento de reemplazar una referencia de pág., se elige la menos usada recientemente**.**
* Trabaja parecido a la cache de memoria principal.

Pasos de funcionamiento:

1. Dada una dirección virtual, el procesador examina el TLB.
2. La lectura de pág. es en paralelo y no indexada (una por una).
3. Si se encuentra la entrada de la página (hit), se obtiene el número del marco y se forma la dirección real.
4. Si no se encuentra la entrada de la página (miss), se usa el número de página para buscar en la tabla de páginas del proceso.
5. Comprueba si la página está en memoria principal (bit de presencia), si no, se avisa un fallo de página.
6. Si estaba en memoria principal, se actualiza el TLB para incluir la nueva entrada.



## Tamaño de página

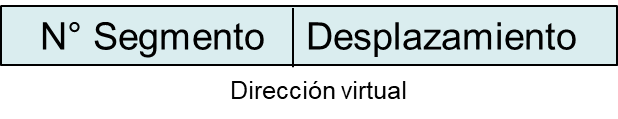
* **A selección del SO** (cant. De memoria reserva para cada proceso) **y del hardware** (tamaño de pág.).
* **A menor tamaño:** 
  + Menor fragmentación interna, más páginas por lo tanto tablas más grandes y posiblemente trasladadas a memoria virtual y provoca más fallos de páginas.
  + Mayor número de páginas se encontrarán en memoria principal. A medida que sigue la ejecución, las páginas en memoria contendrán porciones del proceso traído recientemente a memoria (principio de cercanía), por lo tanto, disminuyen los fallos de página.
* **Mayor tamaño:**
  + La memoria secundaria está diseñada para transferir grandes bloques de datos eficientemente, por lo tanto, un mayor tamaño de páginas sería mejor al realizar una E/S.
* Hay SO que tienen **múltiples tamaños** de página que proveen la flexibilidad necesaria para usar efectivamente el TLB:
  + Páginas grandes se pueden usar para instrucciones de programa
  + Páginas pequeñas se pueden usar para hilos

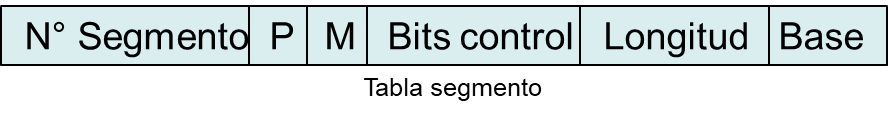
## Segmentado con memoria virtual

* **Puede ser de tamaño diferente y dinámico.**
* Permite crecimiento de estructuras, modularidad y soporte para compartir y proteger.
* El programador la percibe**.**

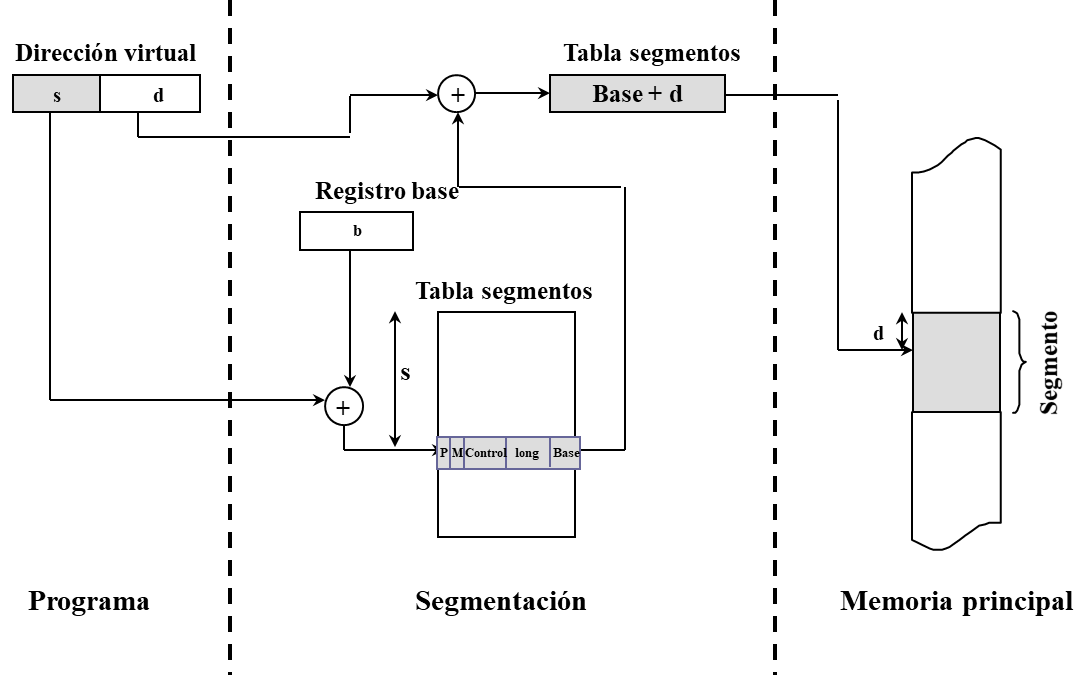
### Tabla de segmentos:

* **Se necesita un bit** (P) para determinar si el segmento está o no en memoria principal.
* **Se necesita un bit** (M) para determinar si el segmento ha sido modificado desde la última vez que se cargó.
* **Bits de control**: protección (lectura, escritura)
* Cada entrada contiene la **longitud** del segmento.
* Cada entrada contiene **la dirección de comienzo** del correspondiente segmento en memoria principal.



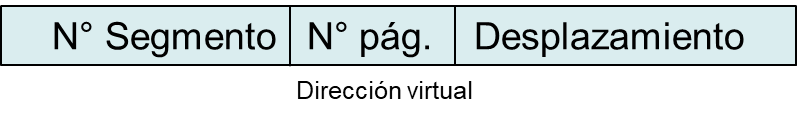


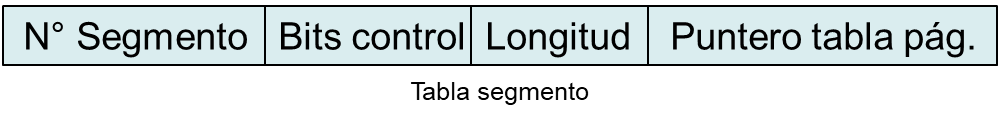
### Traducción



## Sistema combinado de paginación/segmentación

* La memoria principal está dividida en marcos.
* Cada segmento está formado por páginas de tamaño fijo (tamaño de los marcos). Elemento A01: Proceso A, Marco 0, Página 1.
* La paginación es transparente al programador (no es visible, por lo que habíamos dicho antes de paginación).
* La paginación elimina fragmentación externa.
* La segmentación reduce la **fragmentación interna** (solo habrá en la última pág. de cada segmento, por lo tanto, es mínima).
* La segmentación es visible al programador, permite crecimiento de estructuras, modularidad y soporte para compartir y proteger.
* Se necesitan 2 tablas por proceso: Segmento y Pagina.





\*Puntero a la base de la tabla de páginas del proceso



### Traducción



## Políticas que lleva el SO para poder gestionar la memoria virtual (6)

Políticas que lleva el SO para poder gestionar la memoria virtual:

1. [**Lectura**](#_2f3j2rp)

Cuando se debe transferir una página/segmento.

1. [**Vaciado**](#_u8tczi)

Cuando sacar un proceso.

1. [**Ubicación**](#_3e8gvnb)

Dónde colocar la nueva página/segmento.

1. [**Reemplazo**](#_1tdr5v4)

Cuál página/segmento desalojar de memoria.

1. [**Asignación**](#_4ddeoix)

Qué cantidad de memoria real se asigna a cada proceso activo.

1. [**Control de carga**](#_2sioyqq)

Grado de multiprogramación.

## Conjunto de trabajo

* Cantidad de páginas que necesita un proceso en una ventana de tiempo determinada.
* Basado en el principio de cercanía.
* Es difícil de calcular, es dependiendo como tomemos la ventana de tiempo.
* Se mide la cantidad de fallos de página.
* Los conjuntos de trabajo de un proceso son transitorios.
* El tamaño óptimo de la ventana no se conoce.
* Ventana:

W(t,w)

t: Tiempo determinado (Inicio)

w: Espacio de tiempo (Duración)

## Política de lectura/recuperación

Determina cuándo cargar una página a memoria

* Por demanda:
  + El paginado por demanda trae solo las páginas cuando se hacen las referencias.
  + Garantiza que las únicas páginas que se transfieren son las requeridas.
  + Esperas más costosas.
* Paginación anticipada (principio de cercanía):
  + Trae más páginas que las que se necesitan.
  + Acelera tiempos de ejecución de un proceso.
  + Más eficiente cuando las páginas son contiguas en el disco, se usa en la carga inicial.
  + No se puede predecir el uso de estas páginas.

Se suele hacer una combinación: En la carga inicial se traen varias páginas y luego se van trayendo por demanda.

## Política de vaciado

* De memoria principal a disco.
* Vaciado por demanda: una página se escribe en disco solamente cuando ha sido seleccionada para reemplazo.
* Prevaciado: Se realiza [buffering](#_17nz8yj) de páginas, las páginas se escriben en disco por lotes. Se han marcado por alguna razón.

## Política de ubicación

* Determina dónde ubicar un bloque en memoria real.
* Irrelevante en el caso de paginación (trivial).
* Iguales estrategias que en particiones variables para segmentación
  + mejor ajuste
  + peor ajuste (no estaba escrito, no sé si irá)
  + primer ajuste
  + siguiente ajuste

## Política de reemplazo

Elección de la víctima, cual página voy a sacar/matar:

### Marcos que no se deben reemplazar:

* + El kernel, estructuras de control del sistema operativo, buffers I/O.
  + Bit de bloqueo asociado a cada marco.

### Algoritmos de reemplazo

Se busca conseguir la menor cantidad de fallos de página.



#### Óptimo

* + Selecciona aquella página que tardará más tiempo en volver a ser utilizada.
  + Imposible predecir el futuro. Sería la mejor de todas.

#### LRU (menos recientemente utilizada):

* + Matar al que hace mucho que no se usa.
  + Lista con el orden que se usaron. Muy difícil y costoso de implementar.

#### NUR (no utilizada recientemente)

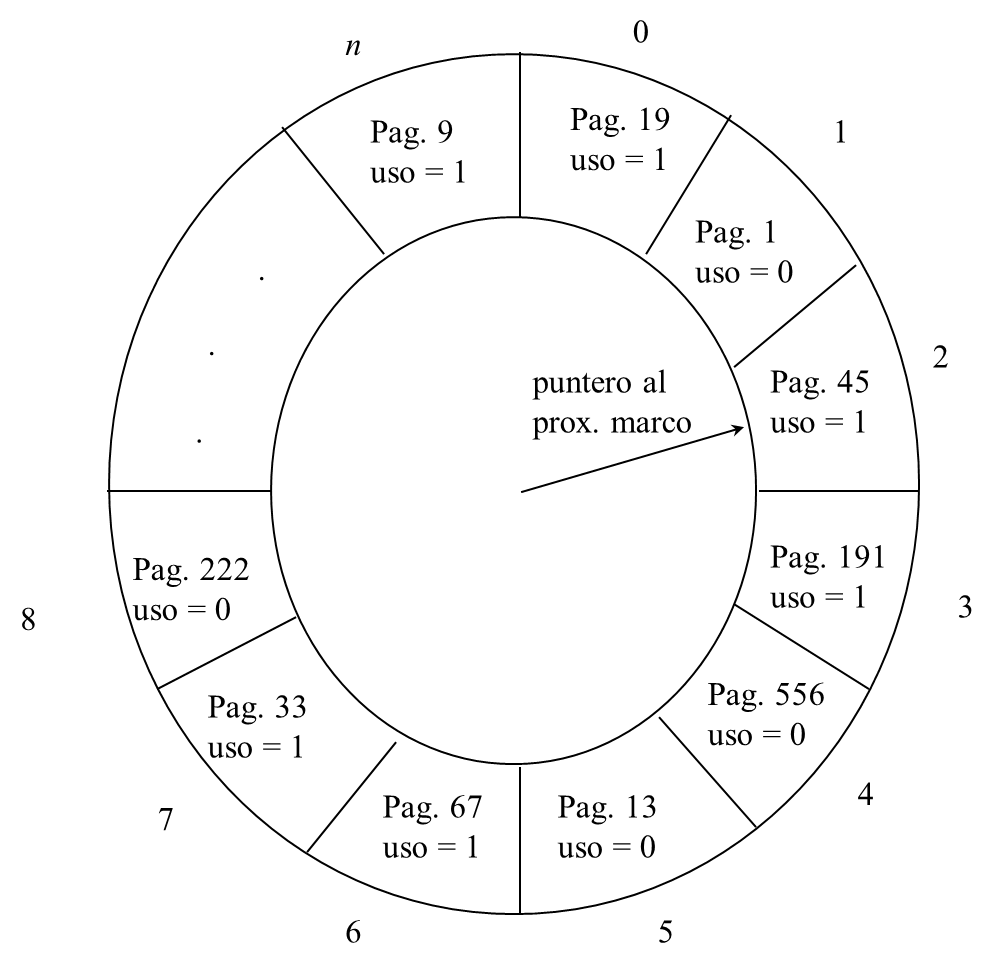
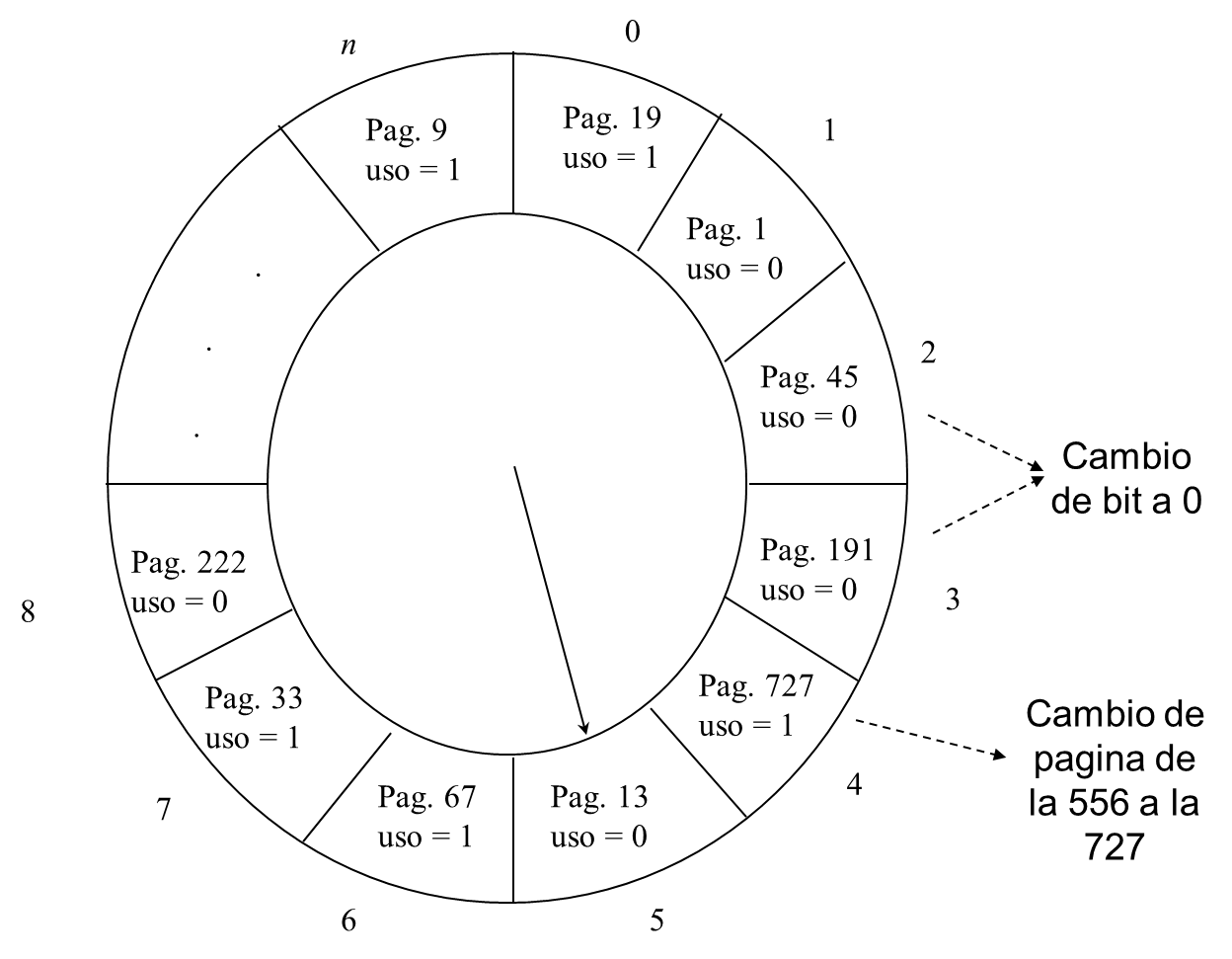
* + Matar al que no usé.
  + Fácil de implementar, un bit para saber si se utilizó recientemente.

#### FIFO

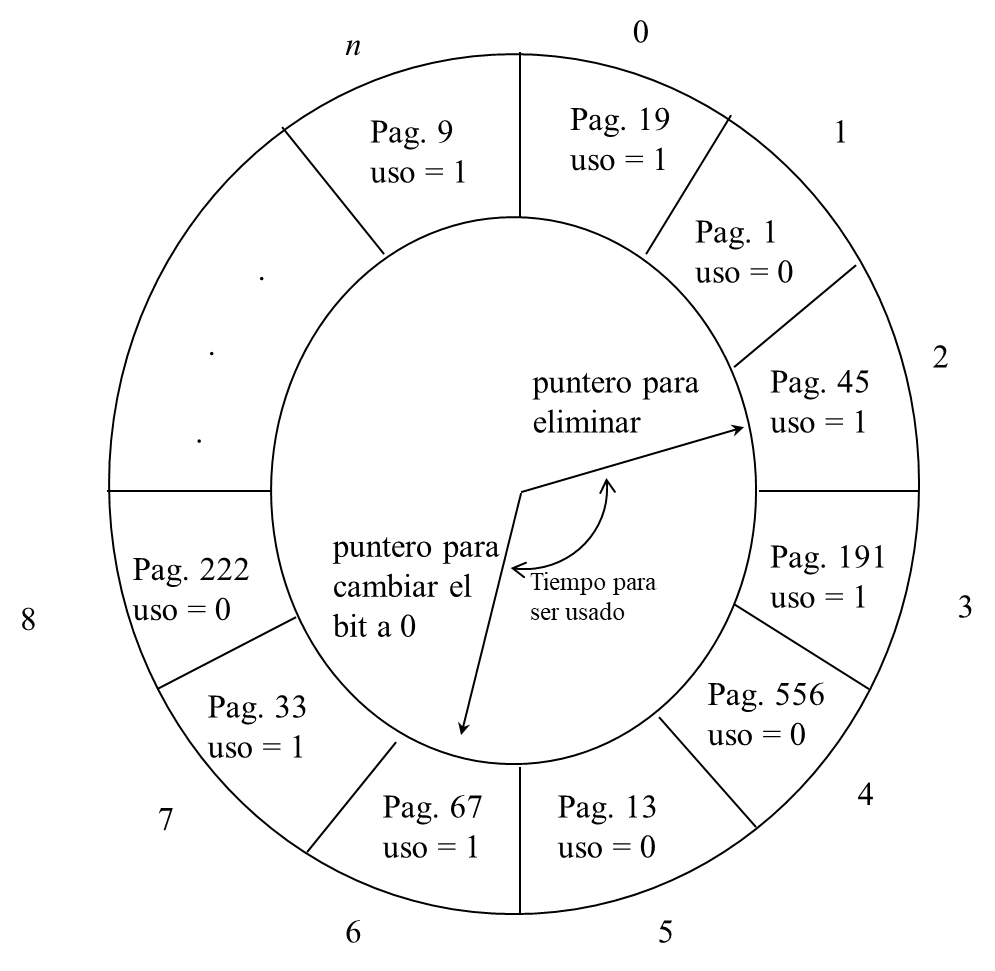
* + Lista de marcos ocupados.
  + Fácil de implementar, se refresca por cada fallo de página (avanza el puntero). Problema, la más antigua puede ser la más usada.

#### Reloj o Segunda oportunidad:

* + Combinación del LRU y FIFO.
* Se reemplaza el primer marco cuyo bit de uso sea 0.
* Cuando va pasando la aguja (puntero) si el bit de uso está en 1 se cambia a 0 (segunda oportunidad) pero no se reemplaza.
* Cuando se carga una nueva pág. su bit está 1.
* Si todos los marcos tenían el bit en 1, hay que dar toda una vuelta para recién buscar si hay alguno en 0. Hay una variante de este método con dos agujas que reduce el tiempo para que los marcos sean usados (bit en 1).

Método de una aguja



Método de 2 agujas, una va adelantada.

Contra más chico es el tipo de adelanto, más se parece al método FIFO.

### Buffering de páginas.

* Utiliza el algoritmo FIFO.
* Las páginas no se reemplazan inmediatamente, se agregan a una de las dos listas:
  + Lista de marcos libres si la página no ha sido modificada (no hace falta cargarla en disco porque ya está, es prácticamente instantáneo).
  + Lista de páginas modificadas (hay que escribirlas en disco, es más lento).
* Las listas funcionan como cache: puedo descargar las páginas en conjunto, cada x tiempo, de forma anticipada (como la política de lectura/vaciado). Conviene mover bloques grandes cuando se realiza una operación con el disco.
* Las páginas sin modificar pueden usarse si se referencian nuevamente o perderse si se asigna su marco a otra página.
* Esta técnica se utiliza en [política de vaciado](#_26sx1u5).

## Política de asignación

* Cuanto lugar se le va a dar a los procesos.
* Factores a tener en cuenta
  + Menor cantidad de memoria para un proceso, mayor cantidad de procesos residentes. Disminuye el tiempo perdido en intercambios.
  + Pequeña cantidad de páginas residentes, aumento de tasa de fallos de páginas.
  + Después de una cierta cantidad de marcos asignados no hay efecto en la tasa de fallos de página

### Asignación fija

* Número fijo de marcos asignados a un proceso en forma anticipada.
* Página a reemplazar debe ser elegida entre los marcos asignados a ese proceso (estrategias de reemplazo se hacen en forma local).
* Desventajas:
  + Asignación muy pequeña: alto grado de fallos de página.
  + Asignación muy grande: bajo grado de multiprogramación.

### Asignación variable

* Número variable de marcos asignados a un proceso, dependiendo de sus necesidades.
* Reemplazo puede ser:

Local:

* + Asignación de un cierto número de marcos al proceso.
  + Cuando se produce un fallo se selecciona un marco del proceso, difícil aplicar las políticas de reemplazo.
  + De vez en cuando se vuelve a evaluar la asignación otorgada (cantidad de marcos).

Global:

* + Fácil de implementar. Reemplazo es sencillo.
  + Al reemplazar, se puede quitar marcos de otros procesos activos.
  + Se puede usar combinada con “Política de reemplazo buffering de páginas” para contrarrestar problemas.

Basadas en [conjuntos de trabajo](#_3zy8sjw) (Local):

* + Intentan mantener el conjunto de trabajo de los procesos activos en memoria real.
  + Si no se mantiene:
    - Sobreutilización de la memoria real (cantidad de marcos > conjunto de trabajo) (hay paginas cargadas innecesariamente).
    - Hiperpaginación (cantidad de marcos < conjunto de trabajo) (muchos fallos de página).

### Algoritmo de frecuencia de fallo de página

* Frecuencia/tiempo entre fallas de página.
* Se establecen 2 limites:
  + Tiempo > límite superior:
    - Liberar páginas que no se referencian últimamente (hay páginas que no se usan).
  + Tiempo < límite inferior:
    - Asignar más cantidad de marcos ([política de asignación variable](#_3rnmrmc)).

## Control de carga (pág. 376)

* Depende el **grado** **de** **multiprogramación** que se quiera.
* Determina el número de procesos que estarán residentes en la memoria principal.
  + Muy pocos procesos: muchas ocasiones en que el procesador estará desocupado porque no hay procesos listos.
  + Demasiados procesos activos: hiperpaginación. Demasiada multiprogramación, hay que matar programas, no solo páginas.

### Suspensión de procesos (6)

Criterio de elección para los procesos que voy a sacar, NO UNA PAGINA:

* **Prioridad más baja**.
* **Con muchos** **fallos de páginas**: no tiene bien su conjunto de trabajo en memoria, por lo tanto, se bloqueará de cualquier manera.
* **Último proceso activado**: Este proceso es el que tiene menos posibilidades de tener su conjunto de trabajo residente en memoria (es el que posiblemente tiene menos por perder).
* **Con el conjunto residente más chico**: Este proceso requiere el menor esfuerzo futuro para volver a cargarse.
* **Proceso más grande:** Se obtiene la mayor cantidad de marcos libres.
* **Mayor ventana de ejecución restante:** Le queda mucho tiempo de ejecución.

# Capítulo 9: Planificación (pág. 399)

<https://drive.google.com/file/d/1IYJldh-BSPU-nQhGmoh8lnFgpv3kj20T/view>

Como se van a ir tomando las decisiones dependiendo de los objetivos del SO (Tiempo respuesta, productividad, eficiencia CPU).

## Nivel de planificación

### Largo plazo

Agregar procesos a “Listo”.

Crear procesos nuevos.

Grado de multiprogramación (más procesos, menos tiempo de ejecución).

### Mediano plazo

Cargar o descargar de memoria, Swapping (suspender/reactivar).

Grado de multiprogramación.

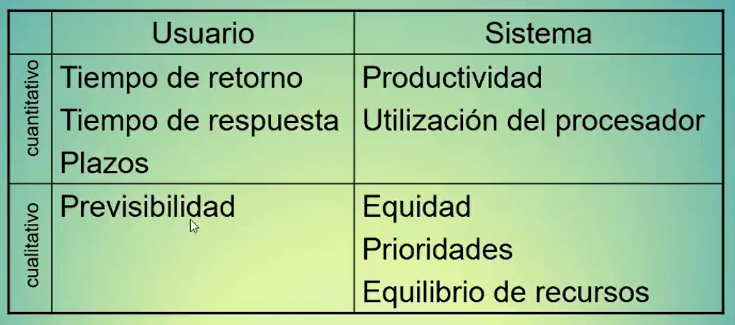
### Corto plazo (Despachador)

Qué proceso pasa de “Listo” a “Ejecución”.

El despachador es invocado cuando el proceso en ejecución deja de estar en ejecución (interrupción, timer, etc.)

Tener en cuenta que van apareciendo procesos con el tiempo, por lo tanto, hay que estar bien atento a corto plazo.

## Criterios de planificación a corto plazo



## Prioridades

El despachador siempre elegirá al de mayor prioridad.

Hay múltiples colas para cada prioridad.

Los de baja prioridad pueden sufrir inanición.

## Función de selección

Determina que proceso se elige.

Basada en prioridades, necesidades de recursos o características.

### Modo de decisión

Instantes donde se aplica la función de selección.

* Sin expulsión (no apropiativa): El proceso puede monopolizar el procesador, solo sale cuando se bloquea.
* Con expulsión (apropiativa): Interrupciones y timers.

## Tipos de planificación a corto plazo

Hay que tener en cuenta el tiempo de cambio de contexto de los procesos.

Tener en cuenta que van apareciendo procesos con el tiempo, por lo tanto, hay que estar bien atento a corto plazo.

1. FIFO (FCFS)
2. Round Robin
3. Primero el más corto (SPN)
4. Menor tiempo primero
5. Primero el de mayor tasa de respuesta (HRRN)
6. Realimentación
7. Distribución justa



## FIFO (FCFS)

Primero que entra primero en salir.

Es de modo sin expulsión

No importan las prioridades.

Favorece a los procesos largos (es preferible que un cirujano esté todo el tiempo en la misma operación antes que ir cada 10 minutos a otra operación).

## Round Robin

Darle rodajas de igual tiempo de ejecución a cada proceso (timer).

Ejemplo UNIX (de 1 segundo).

Es de modo con expulsión.

## Primero el más corto (SPN)

Manda a ejecutar el proceso mas corto que esté en la lista de “Listos”.

Los procesos largos pueden sufrir de inanición, ya que pueden estar apareciendo todo el tiempo procesos cortos.

Es de modo sin expulsión, excepto que, si el tiempo estimado es más largo, el SO lo puede sacar.

## Menor tiempo primero (SRT)

El que menos le queda tiene prioridad. Esto se evalúa cada vez que aparece un proceso nuevo (si se está ejecutando un proceso que le quedan 2 de 10 unidades de tiempo y aparece uno de 3 unidades, se sigue ejecutando el de 10 ya que le queda menos).

Es de modo con expulsión.

## Primero el de mayor tasa de respuesta (HRRN)

Elige al que tiene mayo taza de respuesta, mediante la siguiente formula:



Tiene prioridad los procesos cortos pero los largos no tienen inanición ya que el que más espera más prioridad tiene.

Es de modo sin expulsión.

## Realimentación

Como realmente no se conoce el tiempo que le falta a los procesos, contra más tiempo se estuvo ejecutando menos prioridad tendrá.

Los procesos largos pueden llegar a tener inanición.

Procesos largos son castigados.

Es de modo con expulsión.

## Distribución justa

Los procesos están en grupos, contra más ejecución de un proceso aumenta el valor de prioridad (seria reducir la prioridad) al grupo y al proceso. Por lo tanto, se elige a los grupos y procesos con valor de prioridad más bajo.

# Tema extra 1: Raid

**Raid**: Matriz de discos redundantes baratos

**Disponibilidad**: Tener un camino alternativo ante una posible falla, ya tener un plan B listo para ejecutarse.

## Conector o Interfaz física

**M.2**: SATA o NVMe

**SATA**: SATA

## Interfaz lógica

**SATA**: Se conecta mediante el bus al puente sur (chip que hace de interfaz con los dispositivos).

Ancho de banda: 400-600 MB/s

**NVMe**: Se conecta al bus PCI express (PCIe).

Ancho de banda: 2000MB/s

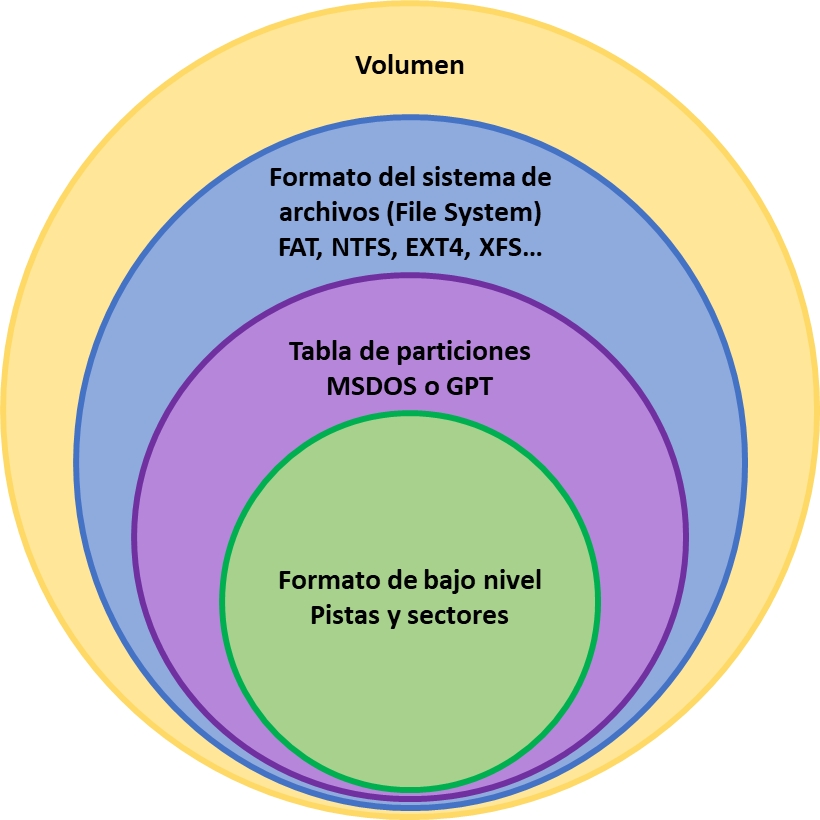
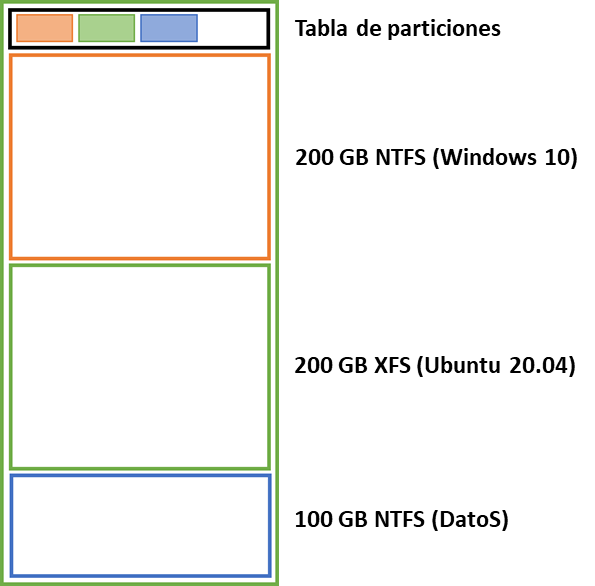
## Tiempo de acceso

**Memoria principal**: 7 nanosegundos

**Discos**: 7 o más milisegundos

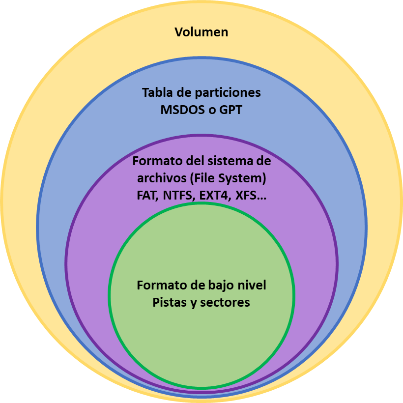
1.000.000 de veces más lento.

## Niveles de un disco

En la foto de la izquierda, para mí, “Tabla de particiones” va antes de “Formato del sistema de archivos”. Porque las particiones individuales son sistemas de archivos, como sale en la foto de la derecha.

Quedaría así, para mí:



## Raid

El raid crea un volumen nuevo formado por 2 o más discos con capacidad y velocidad dependiendo del raid. Usualmente son interoperables, no se puede sacar los discos de una controladora y llevarlos a otra, a pesar que utilice estándares.

Puede realizarse mediante:

### Software

* El SO lo realizará con la placa controladora de disco, nivel de drivers.
* Provoca mucha carga computacional sobre el procesador al utilizar Raids con paridades.
* No aumentan los costos económicos ya que no se necesita comprar una controladora.
* Da la posibilidad de portar el raid sin perder los datos.

### Hardware

* Placa controladora (controladora de RAID y de discos).
* La placa puede fallar y suelen no dar la posibilidad de portar un raid, a pesar de que la controladora sea de la misma marca y modelo que la original.
* Pueden soportar la carga al calcular paridades.

## Tipos de raid

[Hoja de cálculo de Google](https://docs.google.com/spreadsheets/d/1YLaGuV1Kkm_hrXQEMOTaXG4FODXdedeD6gc6Vtgoe9o/edit?usp=sharing)

# Tema extra 2: Virtualización

## Videos

1- <https://drive.google.com/file/d/1a17oztYpb51Ol8S1YNmbJmSgKvHM5X7I/view> 1:41:00 ✅

2- <https://drive.google.com/file/d/1zgK7RoYPkAJJ6UG2QAffiEtVUBlmqdi9/view> 2:05:00✅

3- <https://drive.google.com/file/d/1qZSp2E96J0JhW0yNCSN7-qG6rRTxv8Bi/view> 34:00✅

1- <https://drive.google.com/file/d/1qdZRZJAijWIshY-cWWf93HypP6UduXpl/view> 1:26:00

2- <https://drive.google.com/file/d/1ys1rwHcUeTNr8FpQqAcQFsHhKmoJ5q1-/view> 1:01:00

3- <https://drive.google.com/file/d/1T5LvnxblbiBKJkva7TQ2ZyNFhOKUlgMH/view> 1:41:00

<https://www.youtube.com/playlist?list=PLHgcF7CZ8gKeM7lPJkS3fd2XX_EMX1gmU>

## Diccionario

**MV** = Máquina virtual

**Virtualizar** ≠ Emular

**Sistemas operativos guests** = Máquina virtual

**VMM =** Administrador de máquinas virtuales, Hypervisor o Hipervisor.

**Nodo =** Computadora física o MV

**Clúster** = Sistema distribuido de granjas de computadoras unidos entre si

## Virtualización

Genera una capa de simulación de hardware diferente al que tiene el equipo realmente. Siempre el hardware simulado tiene que ser menor o igual al que tengo físicamente (no puedo virtualizar 32gb de RAM cuando tengo 12gb).

Otorga la posibilidad de trasladar la máquina virtual a otra maquina física sin ningún tipo de problemas, mientras que el virtualizador sea el mismo y esté configurado igual. Al final el SO virtualizado estará viendo siempre el mismo hardware.

¿Porque se puede virtualizar?

Porque me sobra potencia de proceso.

### ¿Por qué usar virtualización? (7)

* **Uso de hardware heredado**: Para SO o programas incompatibles con nuevo hardware. Ej: Usar Windows 95, no lo actualizo porque funciona bien, pero en una maquina nueva porque las viejas ya no se consiguen.

Todo esto se debe a la necesidad de confiabilidad al 100%. Lo viejo está muy probado, lo nuevo aún no se sabe si puede fallar.

* **Implementación rápida:** Copiar una máquina virtual y se lleva a otras máquinas físicas.
* **Versatilidad:** Ahora se puede aprovechar el hardware que no se usaba al 100%.
* **Consolidación y agregación:** Una maquina reemplaza a varias (una física con varias virtuales a la vez).
* **Dinámica:** Facilidad de agregar/cambiar hardware para la máquina virtual.
* **Facilidad en administrar:** Se puede hacer toda la prueba de un sistema para una empresa en una sola maquina física.
* **Mayor disponibilidad:** Varios hosts, si hay algún problema de hardware están los otros que lo reemplazan momentáneamente.

Niveles desde abajo hacia arriba en:

### Sistema tradicional

1. **Hardware**: Lo físico.
2. **SO**: Interfaz y administrador del hardware.
3. **Librerías/bibliotecas**: Para ciertas funciones.
4. **Aplicaciones**: Procesador de texto, etc.

### Sistema virtualizado

1. **Hardware**: Lo físico.
2. **Software de virtualización:** Hipervisor, el encargado de virtualizar para los SO. Es la encargada de transformar los recursos físicos en lógicos para las máquinas virtuales.
3. **SO:** Pueden ser varios SO y diferentes
4. **Librerías/bibliotecas**: Para ciertas funciones.
5. **Aplicaciones**: Procesador de texto, etc.

## Actores de diferentes implementaciones

Windows: Hyper-V

Linux: KVM

VMware

Xen

LXC (Contenedor)

Docker

## Técnica de virtualización (5)

1. [Emulación](#_35xuupr)
2. [Virtualización total](#_1l354xk)
3. [Paravirtualización](#_452snld)
4. [Contenedores](#_2k82xt6)
5. [Asistencia vía hardware](#_zdd80z)

## Emulación

No entra en las técnicas de virtualización, pero sería como el escalón más alto en lo que es simular algo.

Contiene virtualización.

Se puede emular otra arquitectura (ARM, etc.).

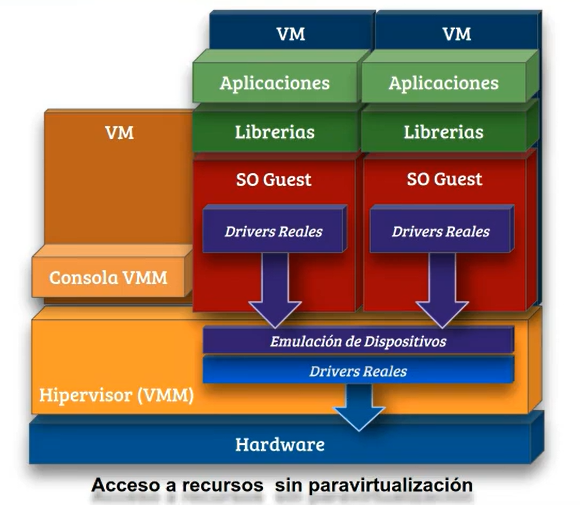
## Virtualización total clásica (Hypervisor)

Misma arquitectura de host.

Traducción bit a bit. Cada instrucción del SO virtualizado pasa por el **Hipervisor** (el SO host), si es una instrucción no privilegiada se envía al procesador, si es una instrucción privilegiada se realiza una subrutina, pero no se envía al procesador.

Le presento a una máquina virtual un hardware totalmente virtualizado, emulando distintos tipos de placas, para poder usar el SO como en una maquina real.

El SO no sabe que está siendo virtualizado.



Como el SO guest NO sabe que está siendo virtualizado, cada dispositivo guest tiene drivers reales que envían información al hardware emulado, donde este realiza la traducción y la envía al hardware físico.

### Hipervisor

Un Virtual Machine Monitor (VMM) o Hypervisor tiene **3 desafíos** (para las arquitecturas x86):

* **Administración de instrucciones:** Distinguir entre Inst. privilegiadas y no privilegiadas.
* **Administración de memoria:** Hace 2 veces la traducción de memoria (2 direcciones lógicas).
* **Como acceder al hardware virtual:** Como el hardware virtual genera salidas sobre el hardware real.

#### Funciones (5)

* **Ejecución Inst. privilegiadas por el Hypervisor.**
* **Adm. Ejecución de MV:** Aislamiento entre MV, cambio de contexto de los procesadores, prioridades de MV.
* **Gestión de ciclo de vida de la MV:** Inicio, pausa, apagarla.
* **Interfaz para que el usuario administre el Hypervisor.**
* **Emular dispositivos y el control de acceso**: De red, controladores, etc.

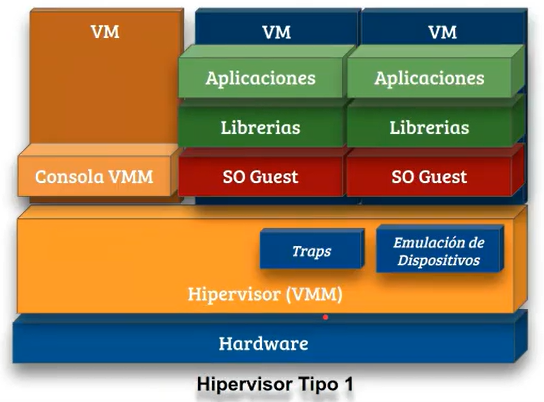
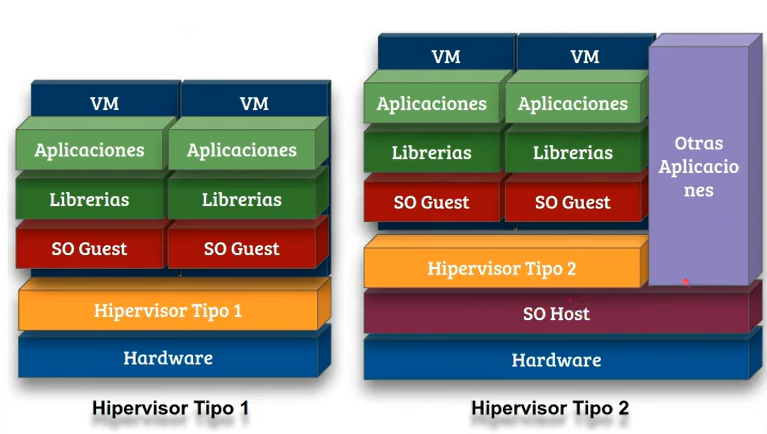
#### Tipo de Hypervisor

**Tipo 1:** El Hypervisor es como el SO host de uso especifico, se utiliza en servidores. Es bastante eficiente, solo se centra en las 5 funciones y 3 desafíos anteriores, el resto de la potencia es dedicada a las MV. Por los recursos solo compiten las MV. **Es poco vulnerable.**

Realiza emulación de hardware virtual y traducción **bit a bit** para la **virtualización total**.

* **Tipo 2:** Hypervisor que corre arriba del SO host (es como otra app más instalada en el SO host). **Hay pelea por los recursos** entre las apps y las MV, disminuye el rendimiento. **Es bastante vulnerable**, hay peligro de que un virus afecte al SO host y por lo tanto se pierdan las MV.

Realiza emulación de hardware virtual y traducción **bit a bit** para la **virtualización total**.

## Paravirtualización

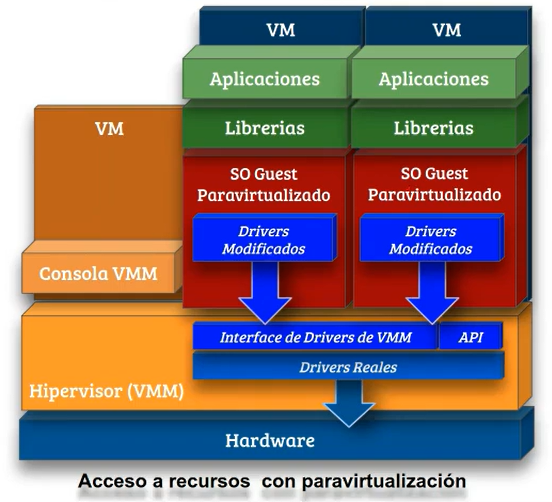
Mas eficiente que virtualización total.

El **SO guest sabe que está siendo virtualizado** (alguien modificó los drivers), por lo tanto, las instrucciones privilegias se realizan con el llamado **(Hypercall) a la API del hipervisor** que las traduce en instrucciones que simulan lo mismo en la MV.

El hipervisor tiene menos trabajo porque ya no tiene que emular hardware.

Los drivers modificados formatean las instrucciones para que las reciba la “Interface de Drivers del Hypervisor” y de ahí las pasa a los “Drivers Reales” que comandan el hardware.

Menos portable.

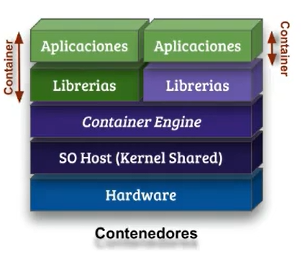


## Contenedores

* Poner los ejemplos: LXC, [Docker](#_3jd0qos),
* Están **vinculados y son dependientes del kernel** del SO host. Por lo tanto, son **vulnerables** a los ataques hacia el SO host.
* **No es muy portable**, pero son más eficientes (menos memoria y menos trabajo del hipervisor).
* Son prácticamente una instancia de un proceso.
* No se busca hacer una máquina virtual, sino correr las aplicaciones que correría en una máquina virtual.
* El conteiner inicializa directamente en el kernel del SO host y en base a esto, puede correr las diferentes aplicaciones con sus librerías correspondientes.
* Cada conteiner empieza con un “init” y luego todas las remas del árbol.
* Se instancian a partir de una Imagen.
* Para mantener aislados cada contenedor, deben tener su FileSystem propio.
* Para guardar información se utiliza COW (Copy on Write), quiere decir que se lee el archivo original (compartido entre los contenedores) pero al momento de realizar un cambio, se realiza una copia con la modificación para dejar la original sin editar.

Se basan en cgroups (control groups) que proporciona:

* Limitación de recursos.
* Priorización
* Contabilidad
* Control



### Motor de contenedor (3)

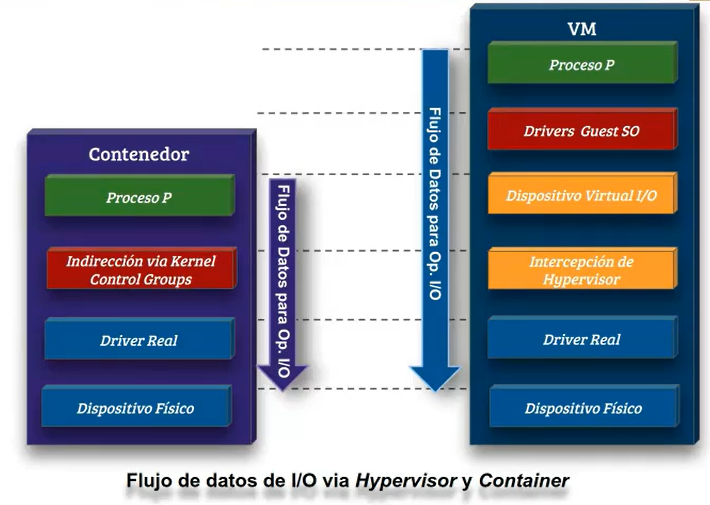
Es como el Hypervisor.

El motor de contenedores configura cada contenedor como una instancia aislada al solicitar recursos dedicados del SO para cada contenedor.

Cumple las siguientes funciones:

* Crear el proceso para el contenedor.
* Adm. los puntos de montaje del Sistema de Archivos (espacio para cada contenedor).
* Solicitar los recursos necesarios al kernel.

El último punto, antes se hacía 2 veces: una el SO guest y otra el Hypervisor:



### Hay 2 sabores de contenedores

**Gruesos**

Cada contenedor tiene una copia de las librerías.

**Finos/livianos**

Comparte librerías con otros contenedores casi un ejecutable (aplicación sola o casi sola).

Ej: Docker

### Ventajas

* No hay necesidad de un SO guest.
* La capa de Adm. es sencilla y portable (ver a tener en cuenta).

### A tener en cuenta

* Solo son portables en SO host con el mismo kernel.
* Si se necesita una configuración especial del kernel, no se pueden usar contenedores sino una MV.
* Si se actualiza el kernel, hay que recrear la imagen.
* Tiene menos aislamiento que una MV (por la dependencia con el kernel).

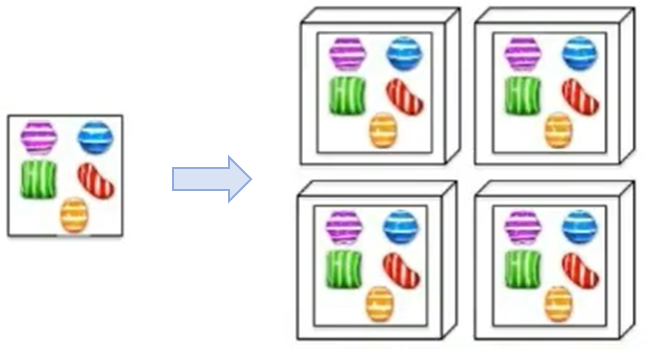
### Arquitecturas

#### Monolítica

Los caramelos son funciones, los cuadrados son contenedores y las cajas son servidores.

Una app monolítica pone todas sus funcionalidades dentro de un único proceso.

Cuando hay más carga, se podría escalar poniendo una máquina más grande, pero esto no siempre es viable, entonces se escala replicando lo monolítico en múltiples servidores.



La consecuencia es que se multipliquen funciones que no hacían falta multiplicar.

Ej: Yo necesitaba más caramelos verdes (función de sacar la raíz cuadrada pongámosle), entones multipliqué por 4 a ese con todo los otros. Los otros 4 caramelos van a estar al pedo por decir.

#### Microservicios

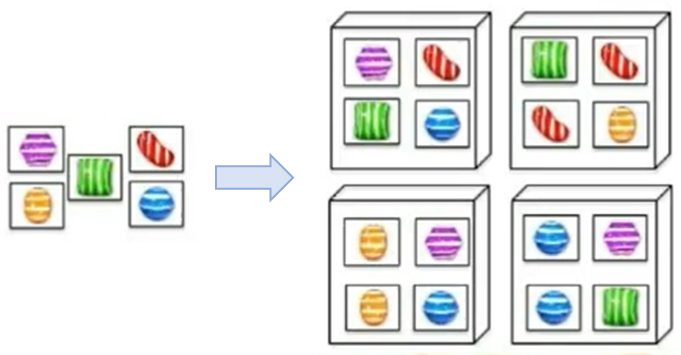
Los caramelos son funciones, los cuadrados son contenedores y las cajas son servidores.

Una Arquitectura Microservicios pone cada elemento de funcionalidad en un servicio separado.

La escala es elástica, quiere decir, se multiplican y distribuyen dichos servicios a través de los servidores, según se requiera. Cuando sobren servicios, se pueden ir matando para reducir los costos en el alquiler del server.

Los servicios están separados, son dependientes y se realizan llamadas entre ellos.

Cada servicio independiente es un contenedor. Cada uno tiene sus propios parámetros para poder comunicarse.



Se separan los servicios individualmente (los caramelos), escalo/multiplico los servicios que necesito y hago llamadas entre los servicios.

Para poder utilizar los servicios duplicados se necesita **un balanceador de cargas (LB, load balancer)** que distribuyen las cargas entre los diferentes contenedores en los diferentes servidores.

### Docker

Facilita la implementación de los microservicios.

Mete en un contenedor todo lo que la app necesita para que se ejecute.

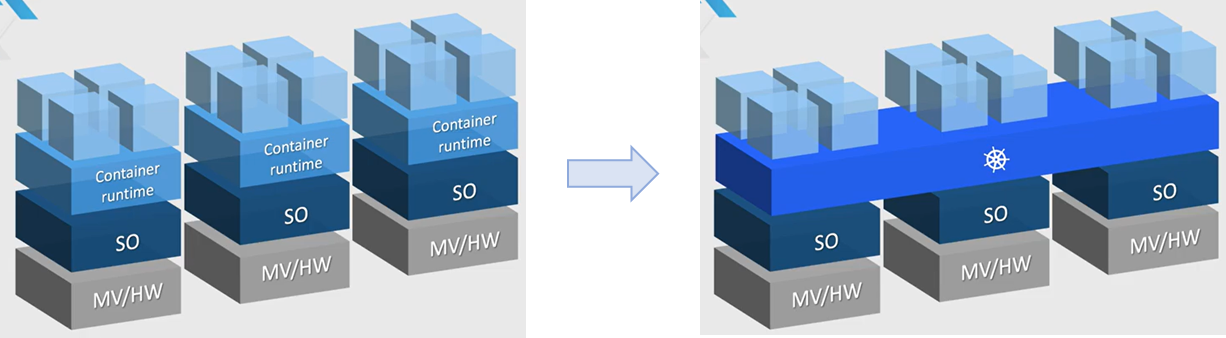
Dockerfile contiene la receta de lo que tiene que hacer el Docker Engine para armar la imagen, que cuando se corre aparece el microservicio.

Docker aísla, agrupa y despliega los servicios. Pero no sirve para producción por cuestiones de seguridad. Tampoco permite un monitoreo de servicio, pero Kubernetes si lo permite.

### Kubernetes (orquestador)

Administra cargas de trabajos basadas en contenedores y crear automatizaciones.

Es como un motor de contenedor que se encontraría en un nodo <[Diccionario](#_ly7c1y)>, pero Kubernetes es el orquestador de los contenedores en los distintos nodos:



Kubernetes no ve contenedores sino POD que son los contenedores en ejecución con características de monitoreo propias de Kubernetes para poder integrarlas en un nodo.

Realiza un monitoreo a los microservicios para:

* Equilibrio de cargas.
* Despliegue y revisiones.
* Auto reinicios.
* Optimización de recursos (distribuir entre los distintos nodos).

Para mantener todo anidado se utilizan los Clúster <[Diccionario](#_ly7c1y)>.

Ya no se utiliza Docker en los Kubernetes porque se podía poner Docker dentro de un POD.

El objetivo de todo esto es la respuesta ante fallas en los clústeres/nodos

## Asistencia vía hardware

Técnicas de hardware (BIOS y placa madre) nuevo para apoyar las técnicas de virtualización anteriores. Replantea la virtualización total.

La mayoría de las maquinas modernas soportan las siguientes extensiones (Intel y AMD):

* “VT-x” y “AMD-V” (Aceleración de virtualización de MV)
* “VT-d” y “AMD-Vi” (Virtualización de dispositivos E/S para una única MV)
* “VT-c” y “???????” (Virtualización de dispositivos E/S para múltiples MV)

#### Modelos de virtualización de dispositivos de E/S:

* **Emulación**: El VMM emula un dispositivo existente, lo cual da compatibilidad, pero, sacrifica rendimiento. El Hypervisor interviene totalmente.
* **Interfaces Sintéticas**: Similar a la emulación, pero se expone al Guest un dispositivo nuevo (que no existe en realidad), diseñado para mejorar el rendimiento y es paravirtualizado (driver de ese dispositivo nuevo sabe que es virtualizado y espera la interrupción del Hypervisor). Hay que generar drivers específicos para ese hardware y SO Guest, por lo tanto, su compatibilidad es menor pero más eficiente.
* **Asignación Directa (VT-d)**: El Guest ejecuta el driver directamente, y el dispositivo sólo puede estar asignado a la VM, el SO host no puede usar/manejar ese dispositivo. Prácticamente es eficiente como si fuera SO host. Sirve para utilizar una placa que solo es compatible con el SO Guest y no con el Host. El Hypervisor no interviene ya que hay comunicación directa.

Requiere de soporte en hardware: PCI-Passthrough.

* **Distribución de I/O en Dispositivo** **(VT-c)**: El dispositivo posee soporte de múltiples interfaces funcionales, que pueden asignarse a diferentes VM. El Hypervisor tiene una mínima injerencia en la transmisión de datos.

Requiere de soporte en hardware: SR-IOV

#### VT-x o AMD-V

Recordemos que el Hypervisor tenía que hacer la Adm. de las instrucciones que se ejecutan en la CPU (ver si son privilegiadas o no).

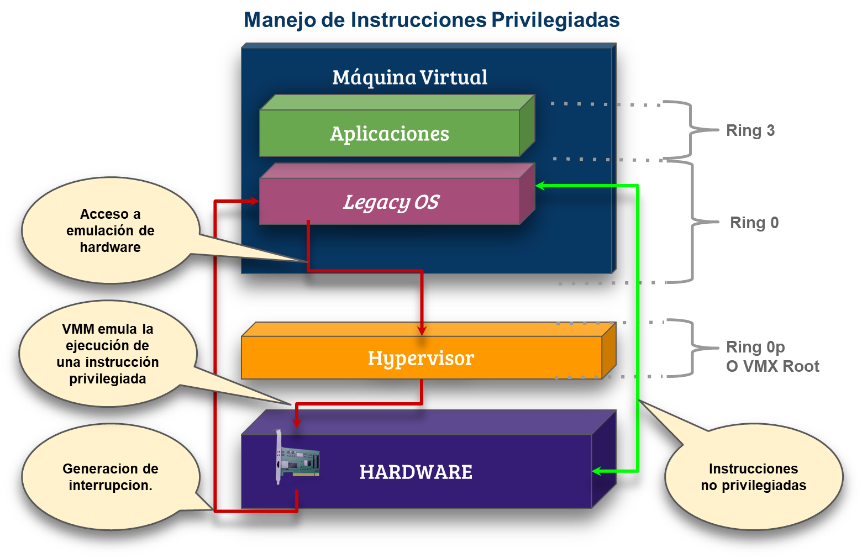
**Uso normal (sin virtualizar):** La arquitectura x86 implementa 4 anillos (0-3) de prioridades donde el 0 es el que puede con instrucciones privilegiadas y el 3 el no (como un anillo de usuario). Tener en cuenta que esto es hardware.



**Virtualización total con hipervisor tipo 1:** Como la VM está un anillo más arriba que el Hypervisor, la VM tiene menos prioridad por lo tanto todas las instrucciones (como mapeo de memoria, interrupciones, etc.) de la VM tiene que pasar si o si por el Hypervisor.



**Virtualización con soporte de hardware:** Se agrega un anillo 0 privilegiado (0p) o VMX Root donde solo se ocupa de las instrucciones privilegiadas y el 0 puede realizar las instrucciones no privilegiadas (hacer mapeo de memoria, interrupciones). Por lo tanto, no necesita la intervención del Hypervisor instrucción por instrucción. El rendimiento es muy parecido a como si estuviera corriendo en forma nativa.

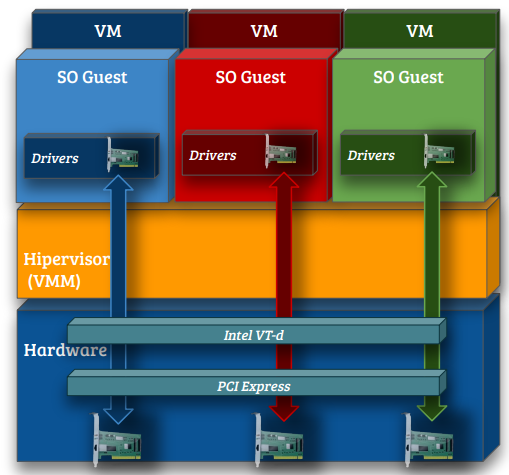
#### VT-d o AMD-Vi

VT-d proporciona rendimiento, seguridad y flexibilidad adicionales al proporcionar al VMM las siguientes funcionalidades:

* Asignación de dispositivos de E/S (Una tarjeta o puerto sea accesible directamente por un SO virtualizado) a través de PCI-Passthrough.
* Remapeo/Reasignación de DMA
* Reasignación de interrupciones

El VT-d se encuentra en el Puente Norte, donde este conecta DMA, Dispositivos integrados y PCIe Root Ports. Por lo tanto, solo funciona con dispositivos PCIe, y no en dispositivos Legacy (USB, etc.) conectados al Puente Sur.

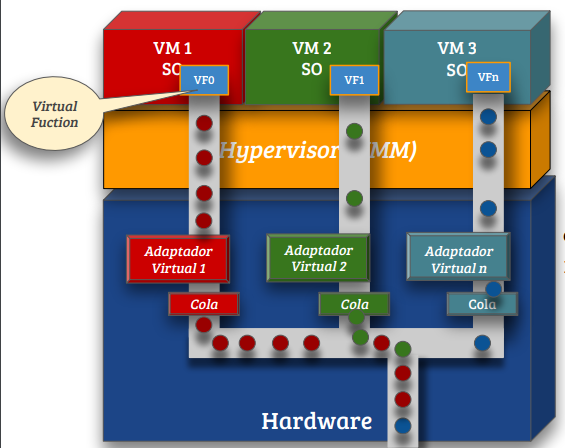
El VT-d realiza el remapeo de direcciones de los dispositivos hacia la MV que están asignados, evitando que interactúen con otras MV, por lo tanto, no pasa por el Hypervisor, es mucho más veloz, se puede utilizar hardware que no se puede emular.



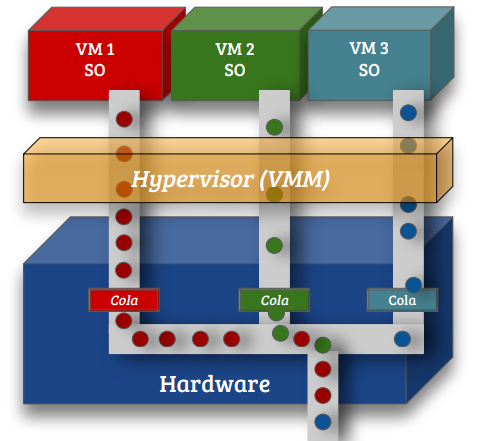
#### VT-c o ??????

Por un lado, **tenemos la técnica SR-IOV**, suele ser para placas de servidores. Permite asignar Funciones Virtuales (FV) que son interfaces de E/S de cada MV con el dispositivo.

Aumentan los costos y quitan un poco de portabilidad ya que al trasladar una MV con VF1 a otro clúster, pude ser que VF1 ya esté ocupado.



Por otro lado, **tenemos a VMDq**, que asigna una interrupción a cada Cola (como las VF, pero menos cantidad) y clasifica las tramas por MAC o VLAN.



## Arquitecturas de Virtualización: VDI

VDI: Virtual Desktop Infrastructure

En este modelo, los recursos de las computadoras de escritorio están virtualizados (CPU, Memoria, Almacenamiento).

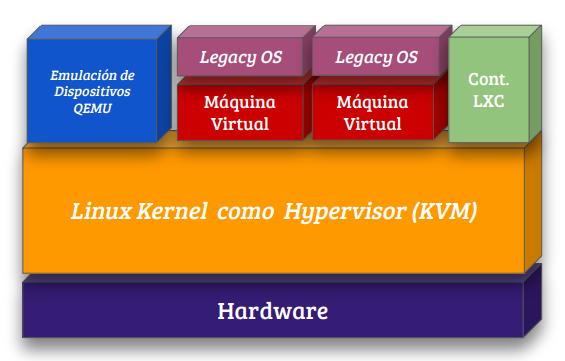
Cada usuario se conecta a una MV en un clúster, por lo tanto, la computadora del usuario solo funciona como un teléfono fijo (no tiene datos y procesa nada más que la imagen de la MV). Esto hace que las máquinas de escritorio no se pongan tan viejas ya que el ciclo de vida seria el del clúster (que es mucho más largo).

## Hypervisor PromoxVE

El Hypervisor es el kernel de Linux.

QEMU: Modulo para emular, en este caso se emula hardware.

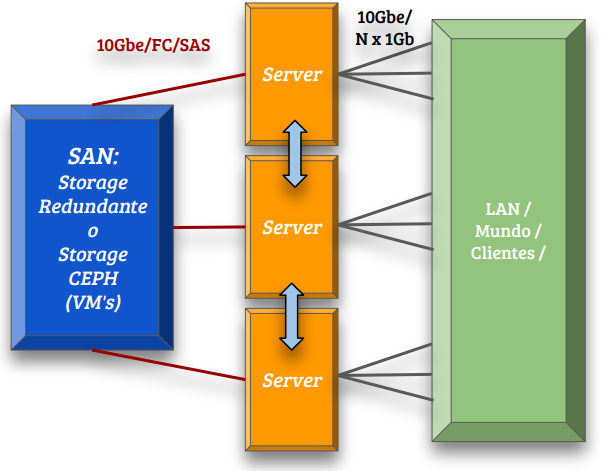
Virtualiza SO si/no modificados y contenedores de Linux (LXC).



## Clúster

Está formado por SAN (almacenamiento) y los Servers (procesamiento).

Las máquinas virtuales están almacenadas en los SAN, donde estos se conectan a los Servers a través de 10Gb o más. Los múltiples Servers están comunicados entre sí, por lo tanto, si uno falla, los otros pueden retomar las MV que están el los SAN. Cada servidor está conectado al mundo (usuarios) a través de 10Gb total o 1Gb por usuario.



# Donde me quedé

<https://shad3xx.angelfire.com/tarea1.htm> Respuestas cap. 1 y 2

<https://shad3xx.angelfire.com/tarea2.htm> Respuestas cap. 3

<https://shad3xx.angelfire.com/tarea3.htm> Respuestas cap. 4

<https://shad3xx.angelfire.com/tarea4.htm> Respuestas cap. 7 y 8.

<https://shad3xx.angelfire.com/tarea5.htm> Respuestas cap. 12.