

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

<https://spark-iodine-aa8.notion.site/Sistemas-Operativos-e976ec4410c342fb9b438459baf74886>

Capítulo 1: Introducción (pág. 9)

CAP 1 TINCHO

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

Se encarga de maximizar la utilización de los recursos disponibles y proporcionar una interfaz fácil de usar para que los usuarios interactúen con el hardware y ejecuten sus programas y tareas de manera efectiva.

Elementos básicos (4)

- **Procesador:** Controla el funcionamiento de la computadora y realiza 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 de los programas que se están utilizando en el momento (volátil); 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 la computadora y su entorno externo. El entorno externo está formado por diversos dispositivos, incluyendo dispositivos de memoria secundaria (discos), equipos de comunicaciones(módems, tarjetas de red) y terminales(Terminal de texto:"Command Prompt" en Windows, Terminal gráfica: La Terminal de GNOME en sistemas Linux.). 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.

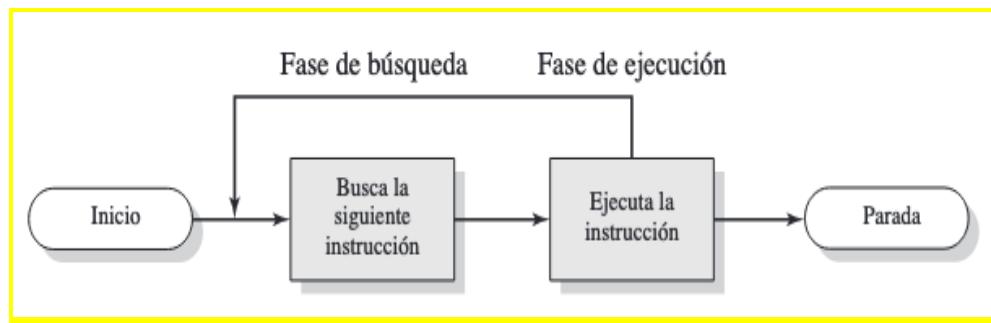
Ejecución de instrucciones

El procesador ejecuta las instrucciones de un programa.

En su forma más simple, el procesamiento de una instrucción consta de dos pasos: el procesador busca y lee instrucciones en la memoria 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

- El procesador lee la instrucción ubicada en la dirección apuntada por el PC.
- La instrucción leída se carga en el IR.
- La instrucción en el IR contiene bits que especifican la acción que debe realizar el procesador.
- El procesador interpreta la instrucción y lleva a cabo la acción requerida.
- El procesador incrementa el PC después de cada instrucción ejecutada (a menos que se indique otra cosa)
- El resultado de la instrucción se guarda momentáneamente en el AC (a menos que se indique otra cosa).
- Se repite el ciclo con la siguiente instrucción apuntada por el PC.

Tipos de instrucciones:

- Procesador-memoria: permiten la transferencia de datos entre procesador y memoria.
- Procesador-E/S: permiten la transferencia de datos desde o hacia un dispositivo periférico de E/S.

- Procesamiento de datos: para realizar operaciones aritméticas o lógicas sobre los datos que realiza el procesador.
- Control: altera la secuencia de ejecución (cambia el valor de PC). Por ejemplo una instrucción de salto o un condicional.

Interrupciones

- Interrumpen el procesamiento normal del procesador.
- Mejora la eficiencia del procesamiento ya que permite al procesador ejecutar otras instrucciones mientras ocurre una operación de E/S o evento externo(cualquier suceso que ocurre fuera del procesador).
- Es la 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.

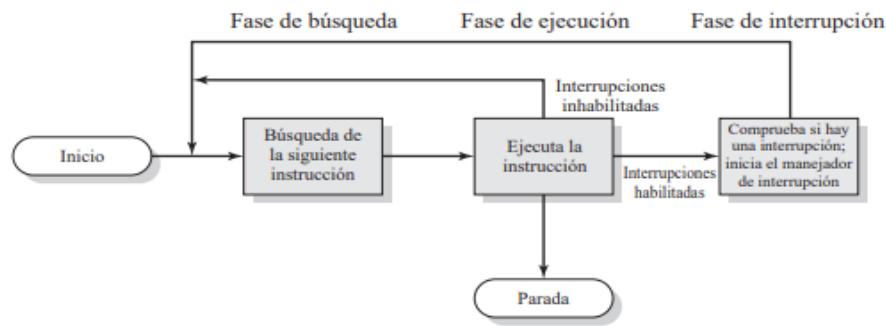
Clases de interrupciones (4):

- 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 cuando ocurre un fallo en el hardware del sistema, 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 (sería 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.

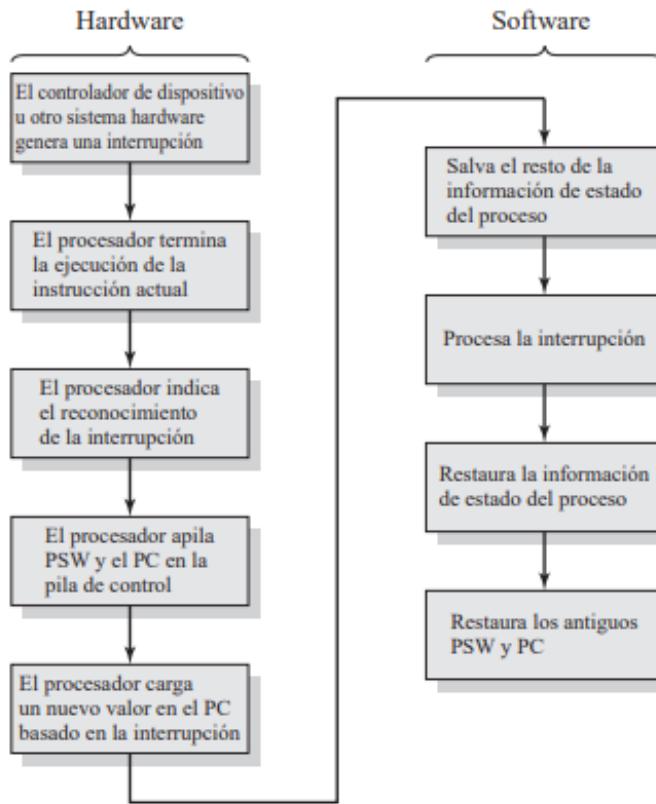


Controlador/manejador de interrupciones o rutina de servicio de interrupción (ISR):

- Un programa que determina la naturaleza de la interrupción y ejecuta las acciones que corresponda.
- Se transfiere el control a este programa, en modo núcleo.
- Generalmente es parte del sistema operativo.

Interrupción simple:

CUADRO IMPORTANTE



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

*Controlador de dispositivo: pequeño software cuya función es la de indicar al sistema operativo el modo en el cual comunicarse con una pieza de hardware.

- 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:

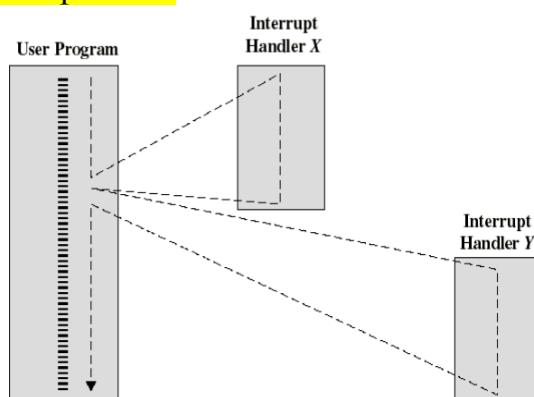
¿Qué pasa si hay una interrupción cuando estoy ejecutando una rutina de interrupción?

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. Despues 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.

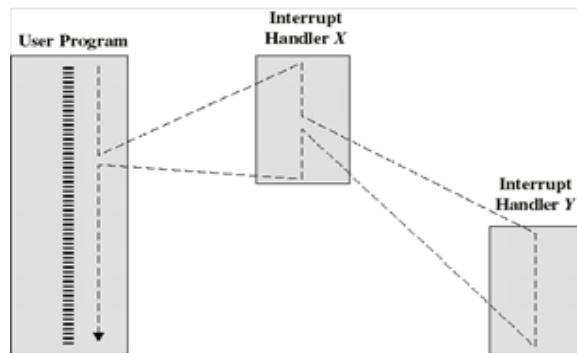
La desventaja de la estrategia anterior es que no tiene en cuenta la prioridad relativa o el grado de urgencia de las interrupciones.



- Una segunda estrategia es definir **prioridades para las interrupciones** y permitir que una interrupción de mayor prioridad interrumpa una interrupción de menor prioridad y que la de menor prioridad se almacene en el estado de 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 básicas de la memoria, costo, capacidad y tiempo.

Podemos afirmar que "cuanto menor costo por bit, mayor capacidad pero mayor tiempo de acceso".

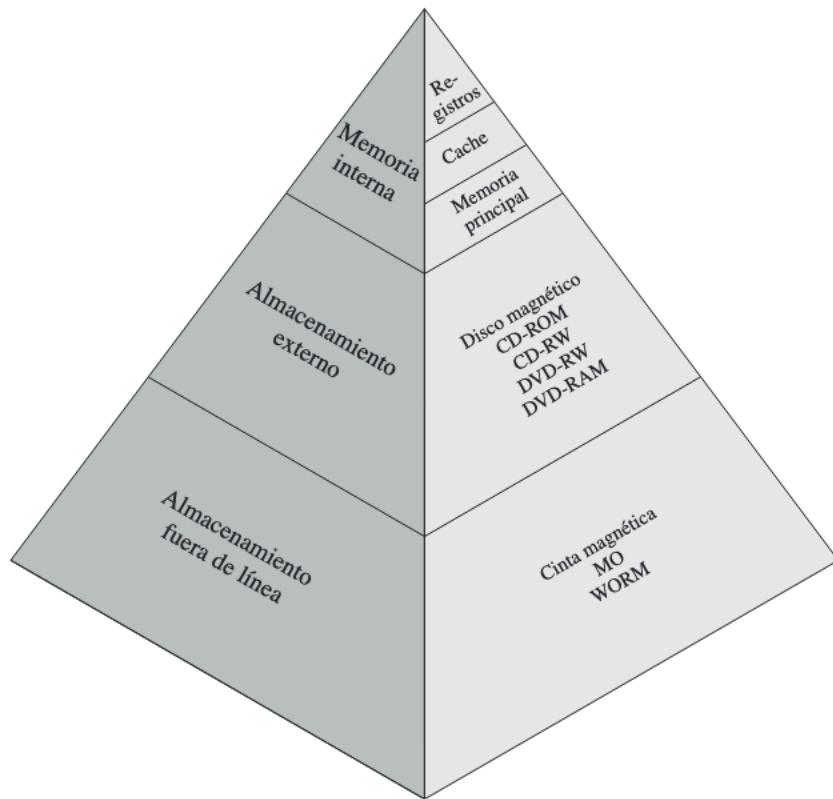
La solución a este dilema consiste en no basarse en un único componente de memoria o en una sola tecnología, sino emplear una **jerarquía de memoria**.

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 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.

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.
- El módulo de E/S actualiza el registro de estado de E/S con información relevante sobre el estado de la operación 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 aparezca la **interrupción de E/S por cada palabra**.

- No hay espera innecesaria (no comprueba el estado, sino que le avisan).
- Consumo mucho tiempo de procesador porque cada palabra leída o escrita pasa a través del procesador.

Instrucciones: Control (permiten al procesador interactuar con los periféricos), Estado (Comprobar condiciones en el procesador o en el sistema) y de Transferencia (leer/escribir entre la memoria y los registros del procesador)

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)

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 control).

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.

El **Sistema Operativo** como administrador de recursos:

- Es un programa.
- Dirige al procesador en el uso de los recursos del sistema
- Dirige al procesador cuando ejecuta otros programas
- Comparte el procesador con los otros programas

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 a los ficheros (archivos).
- **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, evita el uso no autorizado de los usuarios y resuelve 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.

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:

- Debido a la naturaleza secuencial del procesamiento, los usuarios debían reservar un tiempo específico para utilizar el sistema informático. Esto se debía a que solo un usuario podía utilizarlo a la vez, ya que los programas se ejecutaban uno tras otro.
- Antes de poder ejecutar un programa, se requería instalar y cargar el compilador en el sistema. Luego, el programa fuente debía ser cargado y compilado para generar un programa ejecutable.
- Una vez compilado el programa, era necesario guardarla en algún medio de almacenamiento, como un disco magnético, para su posterior carga y ejecución. Esto implicaba una gestión manual de los archivos y la necesidad de contar con suficiente espacio de almacenamiento.
- Si había un error en el programa durante su ejecución, se debía reiniciar todo el proceso desde el principio.

NOTIÓN AGUS

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 denominada 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](#).

3) Sistemas de lotes con multiprogramación: **ACÁ**

Permite al procesador ejecutar otro programa mientras un programa debe esperar por un dispositivo de entrada salida

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.

*DMA: El acceso directo a memoria permite a cierto tipo de componentes de una computadora acceder a la memoria del sistema para leer o escribir independientemente de la unidad central de procesamiento.

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):

Usando multiprogramación para manejo de múltiples tareas

Se comparte el tiempo de procesador entre múltiples usuarios

Múltiples usuarios acceden simultáneamente al sistema a través de terminales

Minimizar el uso del procesador

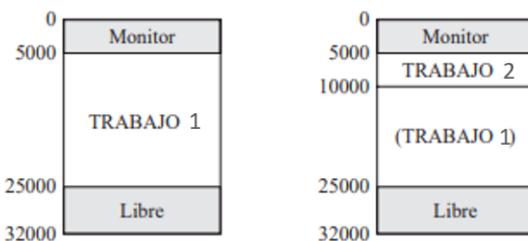
En un sistema de tiempo compartido, **múltiples usuarios acceden simultáneamente** al sistema a través de terminales, siendo el sistema operativo el encargado de entrelazar la ejecución de cada programa de usuario en pequeños intervalos de tiempo o cuantos de computación. Por tanto, si hay “n” usuarios 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)

1) Procesos

- Es un programa en ejecución
- Una instancia de un programa que 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:** Pérdida 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 sí esperando una respuesta del otro (parecido a un bucle).

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

2) Manejo de la Memoria (5)

- **Aislamiento de procesos:** Un proceso no interfiere los datos en memoria de otro.
- **Asignación y manejo automáticos:** Asignación dinámica de memoria en todos sus niveles.
- **Programación modular:** Alternar el tamaño dinámicamente.
- **Almacenamiento por largos períodos 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.

3) Protección y seguridad de la información (5)

- **Autenticidad/Control de acceso:**

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

- **Disponibilidad:**

Protección frente a interrupciones.

- **Confidencialidad:**

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

- **Integridad de datos:**

Protección de datos frente a modificaciones no autorizadas.

4) 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** (establecer 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 (procesos que están listos para ejecutarse), de largo plazo (procesos recién creados que esperan a ser admitidos en el sistema operativo) y de E/S (procesos que han solicitado operaciones de entrada/salida (E/S) y están esperando a que se completen).

5) 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:

- Una entidad activa en un sistema operativo que representa un programa en ejecución. A diferencia del programa en sí, que es una entidad pasiva que reside en el almacenamiento, un proceso es una unidad de trabajo que se ejecuta en un procesador y realiza tareas específicas.
- Una unidad de trabajo 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)

El BCP es un registro especial donde el sistema operativo agrupa toda la información que necesita conocer respecto a un proceso particular. Cada proceso está asociado a atributos que son utilizados por el SO para controlarlos.

Tres categorías generales de información del BCP:

1) Identificación del proceso

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

- Identificador del **proceso padre**. (PPID)

2) Información de estado del procesador

- Registros de CPU: contador del programa, códigos de condición (acarreo, signo, cero, desbordamiento, igualdad) e Información de estado (incluye habilitación/inhabilitación de interrupciones y modo de ejecución). Esta información de los registros debe guardarse, junto con el contador del programa, cada vez que se interrumpe el proceso para que pueda restaurarse cuando el proceso reanude su ejecución.
- Nota: PSW (Palabra de estado del programa). Conjunto de registros que contienen la información de estado.

3) 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.)

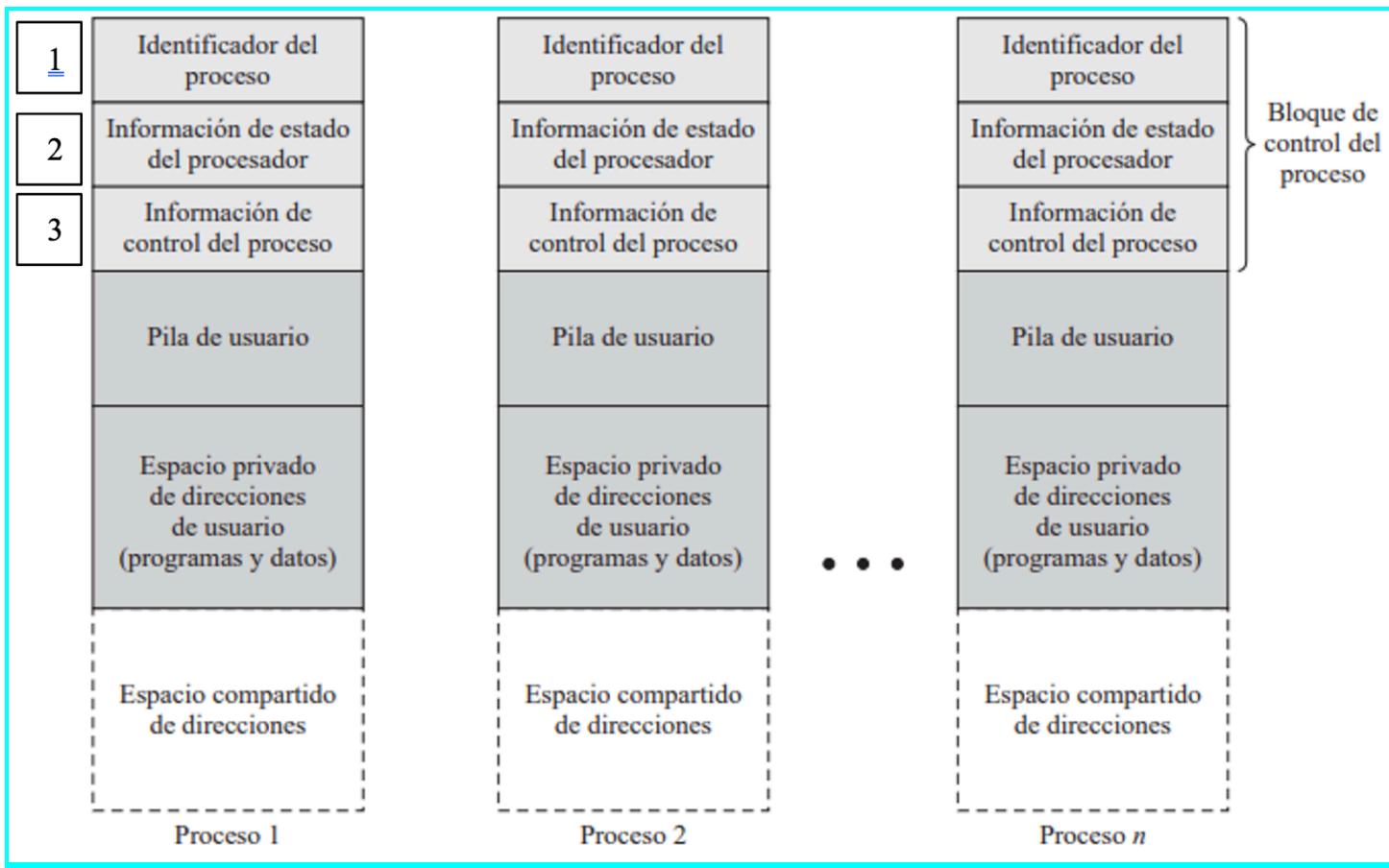
Imagen de un proceso

BCP, Datos de usuario, Programa a ejecutar (puede compartirse con otros procesos que estén ejecutando el mismo programa) y **Pilas de sistema** (para gestionar las llamadas al sistema operativo y los cambios de contexto cuando el proceso necesita acceder a recursos de más bajo nivel o realizar operaciones de E/S., LIFO).

***Pila de sistema:** En un sistema operativo cada proceso tiene un espacio de memoria (pila) para almacenar valores y llamadas a funciones específicas del kernel o núcleo del sistema operativo. La pila del sistema se utiliza para gestionar la ejecución de funciones en modo privilegiado, como las llamadas al sistema y otras operaciones que requieren acceso al hardware y recursos del sistema.

***Pila de usuario:** Es un área en el espacio de usuario que se usa para almacenar información como parámetros, valores de retorno y variables locales que se llaman entre subprogramas del proceso de usuario.

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 crea un nuevo proceso para ejecutar un trabajo que se ha enviado al sistema en forma de lote. Los trabajos por lotes son tareas que se realizan secuencialmente sin interacción del usuario.
- **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 PID.
2. **Reservar memoria** (tamaño basado en el tipo o pude fijarlo el usuario o padre).
3. **Inicialización del BCP** (El sistema operativo crea el BCP para el nuevo proceso).
4. Establecer los **enlaces apropiados**(El sistema operativo establece los enlaces necesarios para integrar el nuevo proceso en las estructuras de datos del sistema operativo, como listas de procesos activos, listas de espera y otras estructuras de administración de procesos.).
5. Creación o expansión de **archivos de auditoría**(para registrar información relevante sobre el nuevo proceso).

*Información de auditoría: Puede incluir la cantidad de tiempo de procesador y de tiempo de reloj utilizados, así como los límites de tiempo, registros contables, etc.

Cambio/conmutación de procesos ESTUDIAR ESTO SI O SI

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(se completa una operación de E/S)
- Fallo de memoria: Traer el bloque de memoria secundaria a principal.

Traps/Excepciones (Internas al proceso)

- Error aritmético(ocurre un error matemático)
- Proceso pasa a estado “Terminado/Saliente”.

Llamada al sistema (pedido explícito)

- Rutinas que son parte del SO (abrir un archivo)(Cuando un programa de usuario necesita acceder a funciones o recursos que solo están disponibles a través del sistema operativo, realiza una llamada al sistema.)

— TODO LO DE ESTADOS DE PROCESOS LEELO NOMAS. SOLO IMPORTA EL DE 7 ESTADOS —

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 Despachador (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:

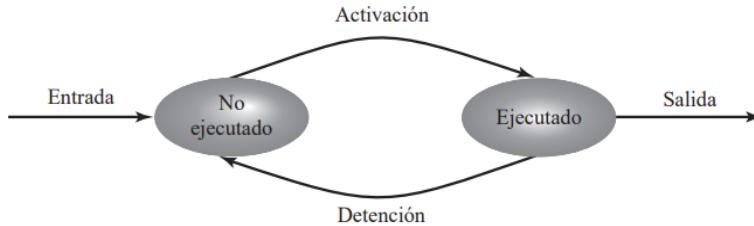
Modelo de 2 estados de proceso

Se puede construir el modelo más simple 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 interrumpe, 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 MITI ACA MITI AGUS

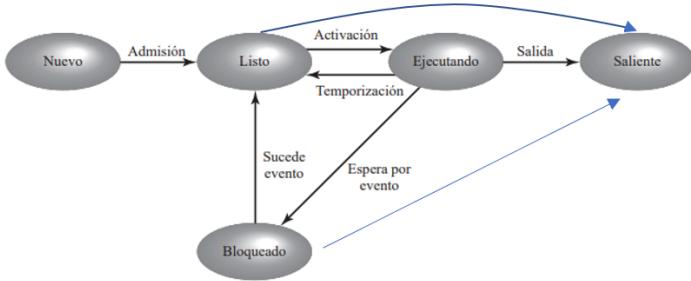
La cola es una lista de tipo FIFO y el procesador opera siguiendo una estrategia cíclica llamada **round-robin** con todos los procesos disponibles.

En round robin cada proceso **tiene un determinado tiempo para ejecutarse** y luego regresar de nuevo a la cola, a menos que se bloquee).

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. Para mejorar la eficiencia de la planificación, se podría implementar una estrategia para que el activador o despachador seleccione procesos no bloqueados que llevan más tiempo en la cola de listos. Esto se logra recorriendo la cola de listos en busca del proceso que cumpla con ambos criterios (no bloqueado y mayor tiempo en la cola)

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 □ Terminado:** Salida, se finaliza la ejecución por parte del SO (se ha completado o fue abortado).
- **Listo □ Terminado:** El proceso padre finaliza a un hijo o finaliza el proceso padre, haciendo finalizar a los hijos.
- **Bloqueado □ Terminado:** 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

Dos nuevos estados:

- **Bloqueado suspendido:** El proceso está en almacenamiento secundario y esperando un evento.
- **Listo suspendido:** El proceso está en almacenamiento secundario pero está disponible para su ejecución tan pronto como sea cargado en 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 una 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.

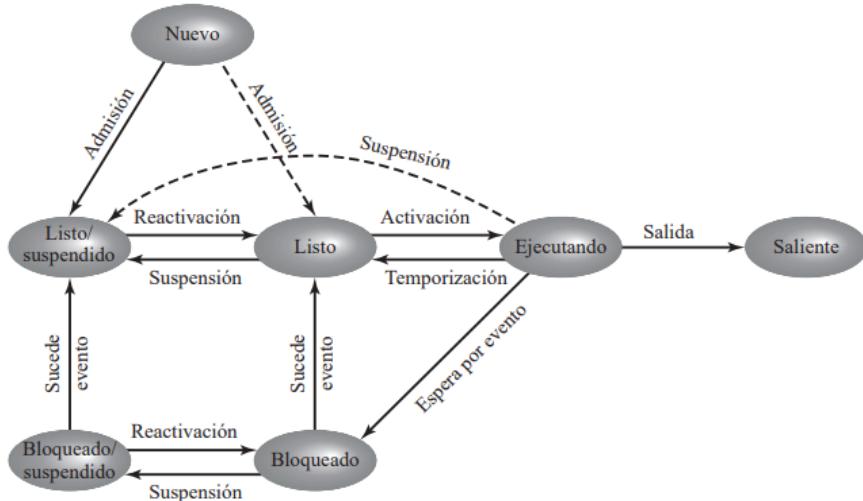
Razones para la suspensión

- **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 ser más rápido que realizar operaciones de E/S en dispositivos periféricos como impresoras o dispositivos de red.

- 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(buscar y resolver problemas) 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

- Las tablas de memoria son estructuras de datos utilizadas por el sistema operativo para mantener un registro de la asignación y uso de la memoria tanto en la memoria principal (RAM) como en la memoria secundaria (disco). Estas tablas son fundamentales para la gestión eficiente de la memoria y para garantizar que los procesos se ejecuten de manera segura y sin interferencias. Deben contener la siguiente información:
 - Las reservas de memoria principal por parte de los procesos.
 - Las reservas de memoria secundaria por parte de los procesos.
 - Todos los atributos de protección para accesos a regiones de memoria compartida.
 - La información necesaria para manejar la memoria virtual.

Tablas de E/S

Se utiliza 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 tabla 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 MÁS O MENOS SABER CUALES SON

Administración de procesos:

- Creación y terminación de procesos.
- Planificación y despacho de procesos.
- Comutación 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.

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:

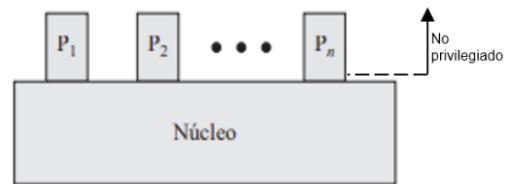
1. Guardar **contexto de procesador** incluyendo PC y otros registros.
2. **Actualizar el BCP (A)** con el nuevo estado.
3. **Mover BCP (A) a la cola apropiada** (“Listo”, “Bloqueado”, etc.).
4. Seleccionar **otro proceso a ejecutar (B).**
5. **Actualizar el BCP (B)** a “Ejecutando”.
6. **Actualizar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

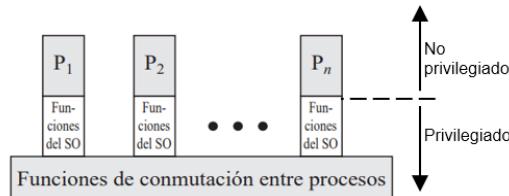
Kernel sin procesos:

- 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).



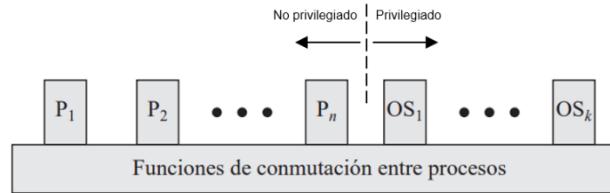
Ejecución dentro de los procesos de usuario:

- El SO dentro de los procesos de usuario (**A la imagen del proceso se le incluye una pila de núcleo, que es una pila separada utilizada por el código del sistema operativo que se ejecuta en modo privilegiado**).
- 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) CASI TODO AGUS, VER QUE DEL TINCHO

Por ejemplo, si yo tengo una computadora con 4 núcleos (puede ejecutar 4 procesos a la vez) y cada núcleo tiene 2 hilos (puede tener 1 o 2 hilos), por lo que si yo tengo 4 núcleos de 2 hilos cada núcleo voy a tener 8 hasta 8 procesos de los cuales solo 4 estarán en estado de ejecución (1 por cada núcleo). Ejemplo Facu: comer porciones de pizzas, cada mano sería un hilo.

Procesos e Hilos (Threads)

El concepto de proceso es más complejo y sutil que el presentado hasta ahora. Engloba dos conceptos separados y potencialmente independientes: uno relativo a la propiedad de recursos y otro que hace referencia a la ejecución.

● **Unidad que posee recursos:** A un proceso se le asigna un espacio de memoria y, de tanto en tanto, se le puede asignar otros recursos como dispositivos de E/S o ficheros.

● **Unidad a la que se le asigna el procesador:** Un proceso es un flujo de ejecución (una traza) a través de uno o más programas. Esta ejecución se entremezcla con la de otros procesos. De tal forma, que un proceso tiene un estado (en ejecución, listo, etc) y una **prioridad** de expedición u origen. La unidad planificada y expedida por el sistema operativo es el proceso.

En la mayoría de los sistemas operativos, estas dos características son, de hecho, la esencia de un proceso. Sin embargo, son independientes, y pueden ser tratadas como tales por el sistema operativo. Esta distinción ha conducido en los sistemas operativos actuales a desarrollar la construcción conocida como **thread**, cuyas traducciones más frecuentes son **hilo**, **hebra** y **proceso ligero**.

Si se tiene esta división de características, entonces:

- la unidad de asignación de la CPU se conoce como hilo
- mientras que a la unidad que posee recursos se le llama proceso.

Dentro de un proceso puede haber uno o más hilos de control cada uno con:

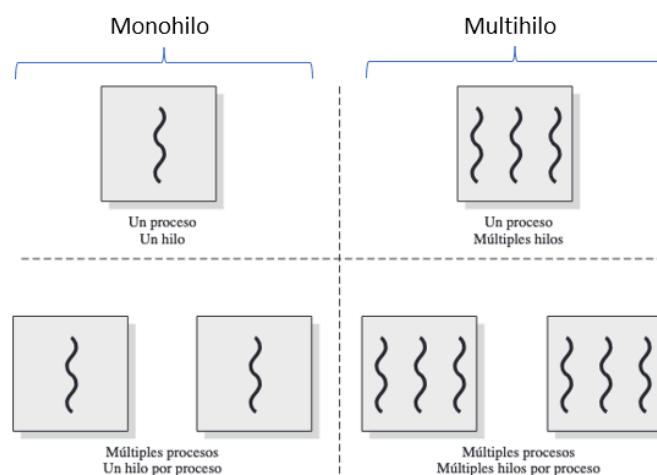
- Un estado de ejecución (en ejecución, listo, bloqueado).
- Un **contexto** de procesador, que se salva cuando no esté ejecutándose.
- Una **pila** de ejecución.
- Algun almacenamiento estático para variables locales.
- Acceso a la memoria y a los recursos de ese trabajo que comparte con los otros hilos.

Los beneficios clave de los hilos se derivan de las implicaciones del rendimiento: se tarda menos tiempo en crear un nuevo hilo de un proceso que ya existe, en terminarlo, y en hacer un cambio de contexto entre hilos de un mismo proceso. Al someter a un mismo proceso a varios flujos de ejecución se mantiene una única copia en memoria del código, y no varias.

Un ejemplo de aplicación que podría hacer uso de los hilos es un servidor de ficheros de una red de área local. Cada vez que llega una solicitud de una operación sobre un fichero, se puede generar un nuevo hilo para su gestión. El servidor gestiona multitud de solicitudes, por tanto, se pueden crear y destruir muchos hilos en poco tiempo para dar servicio a estas peticiones. Si el servidor es un multiprocesador, se pueden ejecutar varios hilos de un mismo proceso simultáneamente y en diferentes procesadores.

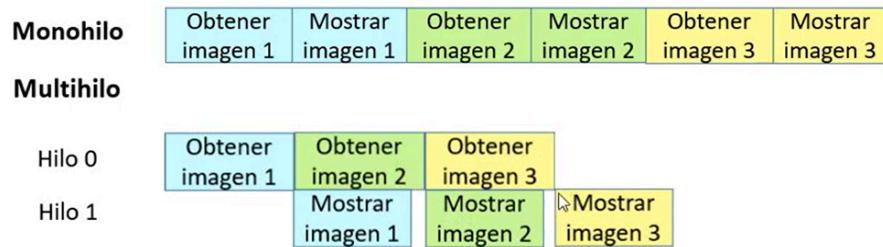
Multihilo

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.

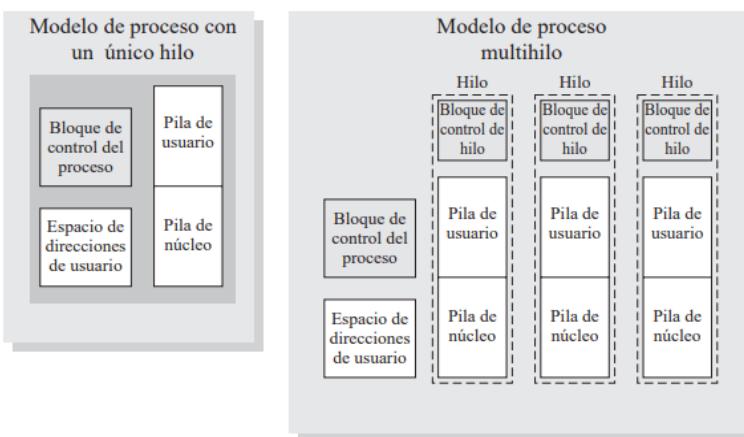


Recursos que comparten los hilos de un proceso:

- El espacio de direcciones virtuales que soporta la imagen del proceso.
- Afinidad de procesadores(pueden ejecutarse en los mismos núcleos), Comunicación entre hilos (los hilos de un proceso pueden comunicarse entre sí) , 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 (contador de programa y otros registros que mantienen el estado de ejecución del hilo)
- Una **pila de ejecución**(almacena la información necesaria para llevar a cabo llamadas a funciones y mantener el flujo de ejecución del hilo).
- 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.

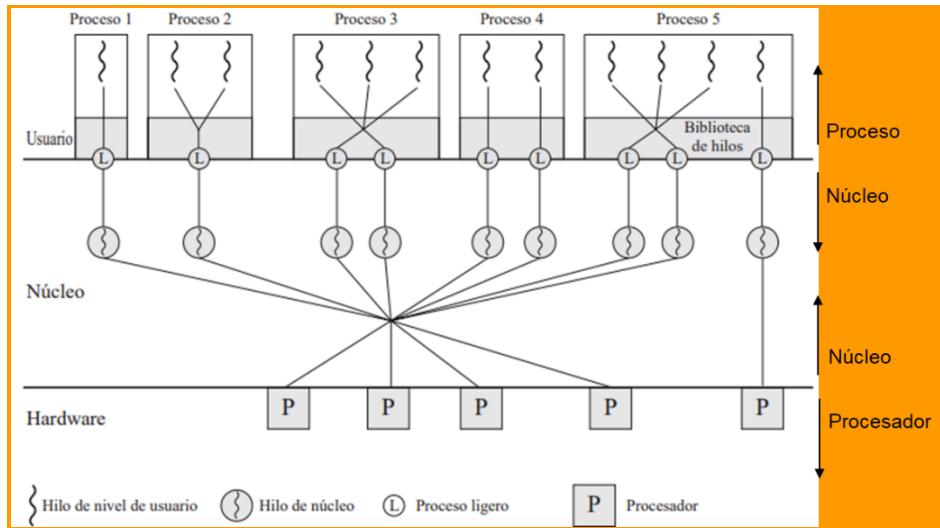
Hay cuatro operaciones básicas relacionadas con los hilos que están asociadas con un cambio de estado del hilo:

- Creación: Cuando se crea un nuevo proceso, también se crea un hilo de dicho proceso. Posteriormente, un hilo del proceso puede crear otro hilo dentro del mismo proceso. Cada hilo tiene su propio registro de contexto y espacio de pila. Después de ser creado, el nuevo hilo se coloca en la cola de hilos listos para su ejecución.
- Bloqueo: Ocurre cuando necesita esperar por algún evento, como una operación de E/S o una señal, y no puede continuar su ejecución hasta que dicho evento se complete. Durante el bloqueo, el sistema operativo debe guardar el contexto del hilo para poder cambiar a otro hilo en la CPU en estado Listo, dentro del mismo proceso o en otro diferente.
- Desbloqueo: Cuando sucede el evento por el que el hilo está bloqueado, el hilo se pasa a la cola de Listos.
- Finalización: Cuando se completa un hilo, el sistema operativo debe liberar los recursos asociados a ese hilo.

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) ESTO DE ACÁ



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 ya que su administración es independiente del kernel.
- 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 (esto se debe a que todos los hilos de nivel de usuario comparten el mismo espacio de direcciones y recursos del 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): EL LICCI SE LA JUGÓ CON ESTO

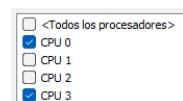
- Ejemplo: Solaris
- Combinan las ventajas y minimizan las desventajas.
- El grueso de la creación, planificación y sincronización de hilos se hace en el espacio de usuario.
- Aprovecha el paralelismo.
- Si se bloquea un hilo, no se debería bloquear el proceso.

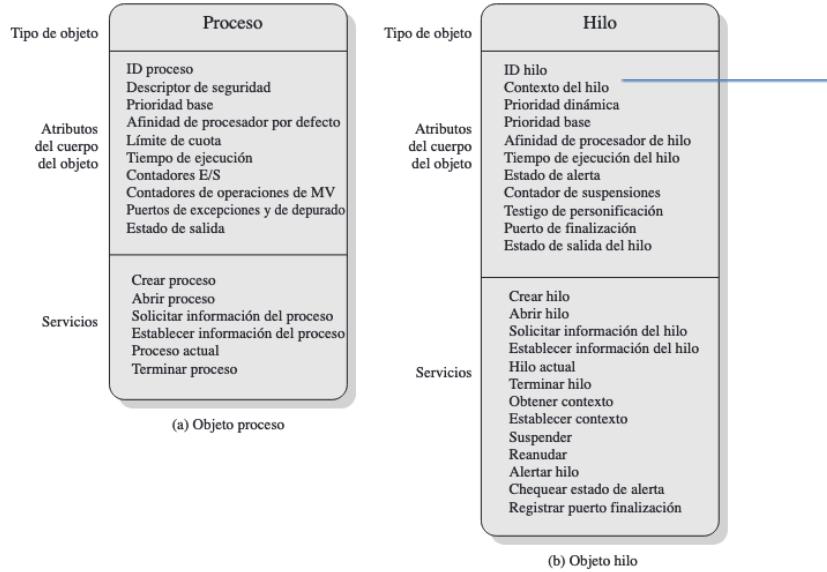
Tenes una idea de lo de abajo (Solaris Windows)

Procesos Windows

Características importantes de los procesos Windows son las siguientes:

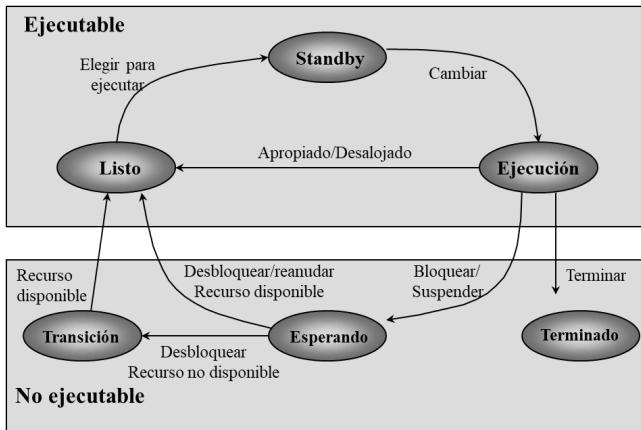
- Los procesos están implementados como objetos (**Tipo, Atributo, Servicio**).
- Un proceso ejecutable puede contener uno o más hilos.
- Ambos objetos, procesos e hilos, tienen capacidades de sincronización incorporadas.
- Se puede asignar afinidad de procesador a cada proceso.
- Se puede asignar afinidad 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 sustituto (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

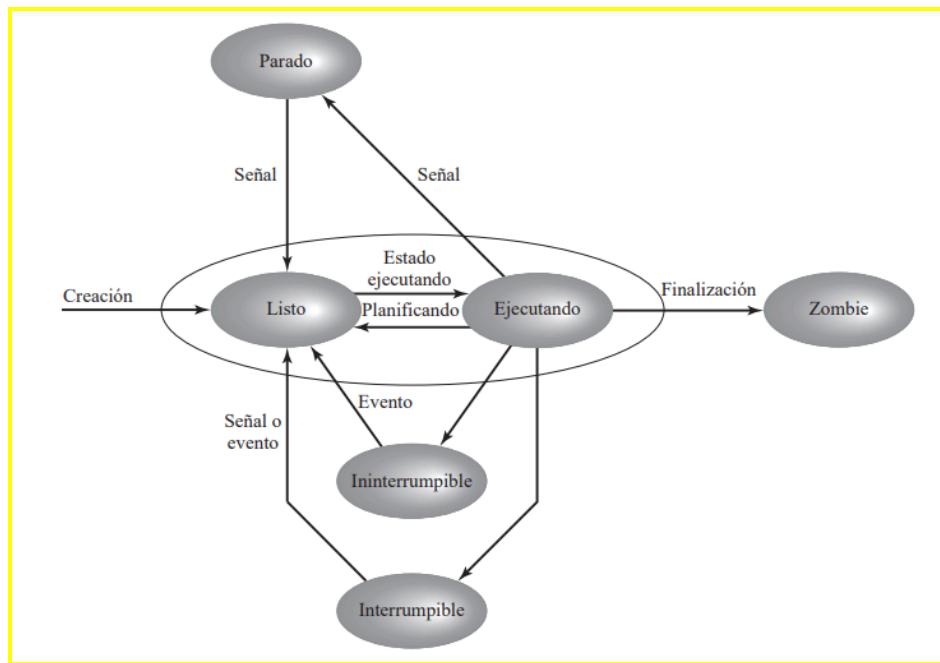
Un proceso, o tarea, en Linux se representa por una estructura de datos “task struct” que contiene información de diversas categorías (9):

- 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)

ESTUDIAR DEL QUE ME PASÓ EL LICCI, ACÁ SE PICA PQ

HAY QUE ENTENDER

Operación atómica	Una secuencia de instrucciones que parecen ser indivisible, o sea, ningún proceso puede ver un estado intermedio o interrumpir la operación.
Sección crítica	Sección de código dentro de un proceso que requiere acceso a recursos compartidos y que no puede ser ejecutada mientras otro proceso esté en una sección de código correspondiente.
Interbloqueo	Situación en la cual dos o más procesos son incapaces de actuar porque cada uno está esperando que alguno de los otros haga algo.
Círculo vicioso	Situación en la cual dos o más procesos cambian continuamente su estado en respuesta a cambios en los otros procesos, sin realizar ningún trabajo útil.
Exclusión mutua	Requisito de que cuando un proceso esté en una sección crítica que accede a recursos compartidos, ningún otro proceso pueda estar en una sección crítica que acceda a ninguno de esos recursos compartidos.
Condición de carrera	Situación en la cual múltiples hilos o procesos leen y escriben un dato compartido y el resultado final depende de la coordinación relativa de sus ejecuciones.
Inanición	Situación en la cual un proceso preparado para avanzar es soslayado indefinidamente por el planificador; aunque es capaz de avanzar, nunca se le escoge.

Concurrencia

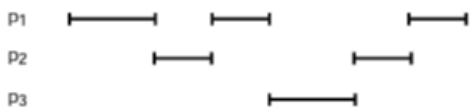
Tiene que haber Exclusión Mutua para que haya Concurrencia.



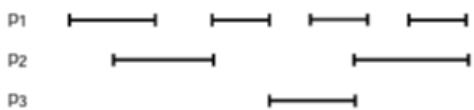
Coincidir en el lugar y/o tiempo.

Concurrencia

- En un sistema multiprogramado ($1 \text{ } \mu\text{P}$), los procesos se intercalan, para dar la apariencia de simultaneidad.



- En un sistema con varios procesadores ($n \text{ } \mu\text{P}$), los procesos se superponen.



6

Concurrencia

- La concurrencia es la simultaneidad de hechos.
- Un programa concurrente es aquel en el que ciertas unidades de ejecución internamente secuenciales (procesos o threads), se ejecutan paralela o simultáneamente.
- Incluye los siguientes aspectos:
 - comunicación entre procesos.
 - compartición y competencia por los recursos.
 - sincronización de la ejecución de varios procesos.
 - asignación del tiempo de procesador a los procesos.
- Surgen en entornos con un solo procesador, con multiprocesadores y proceso distribuido.

3

Concurrencia

- Un programa concurrente está formado por una colección de procesos secuenciales autónomos que se ejecutan (aparentemente) en paralelo.
- Tres formas de ejecutar procesos concurrentes:
 1. Los procesos multiplexan sus ejecuciones sobre un único procesador (multiprogramación).
 2. Los procesos multiplexan sus ejecuciones sobre un sistema multiprocesador de memoria compartida (multipropceso).
 3. Los procesos multiplexan sus ejecuciones en varios procesadores que no comparten memoria (procesamiento distribuido).
- El término **conurrencia** indica paralelismo potencial.

4

La concurrencia es fundamental en: (ESTO ES RE DE MEMORIA)

- 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 des ubicar 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.

Requisitos de exclusión mutua (6)

- Sólo un proceso de todos los demás, puede estar en su sección crítica 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 (aviso que salió de la sección crítica).

Herramientas para la concurrencia

- Exclusión mutua por soporte de hardware
- Semáforos
- Monitores
- Paso de mensajes

Exclusión mutua por soporte de hardware

Deshabilitar interrupciones

- En maquina monoprocesador, deshabilitar interrupciones garantiza la exclusión mutua.
- En maquina multiprocesador, deshabilitar interrupciones no garantiza la exclusión mutua (se deshabilita solo en un procesador).

Instrucciones de máquina especiales

- Realizan una acción sobre una única posición de memoria en un solo ciclo de búsqueda de instrucción, lo que las convierte en instrucciones atómicas.
- El acceso a la posición de memoria se le bloquea a toda otra instrucción.

Desventaja

- 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.

Semáforos (ESTO ENTRA CASI SIEMPRE, EL GRÁFICO DEL NOTION DEL AGUS)

- Es una variable especial que se utiliza para señalizar (números enteros o binarios).
- Programa del SO.
- Los procesos se ejecutan en modo usuario y hacen llamados al SO 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)

Semaforo 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.

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 desbloqueando otros (nolleno = falso, entonces el productor se bloquea).

Un monitor es como un baño público. Sólo una persona puede entrar a la vez. Cierran la puerta con llave para evitar que alguien más entre, hacen sus cosas y luego la abren cuando se van.

Un semáforo es como un lugar de alquiler de bicicletas. Tienen un cierto número de bicicletas. Si intentas alquilar una bicicleta y tienen una libre, puedes tomarla, si no, debes esperar. Cuando alguien devuelva su bicicleta, otra persona puede tomarla. Si tienes una bicicleta, puedes dársela a otra persona para que la devuelva. Al alquiler de bicicletas no le importa quién la devuelva, siempre y cuando la devuelvan.

Capítulo 6: Interbloqueo e inanición (pág. 257) DE LO DEL AGUS, ES LA UNIDAD MÁS FÁCIL DE TODAS

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.

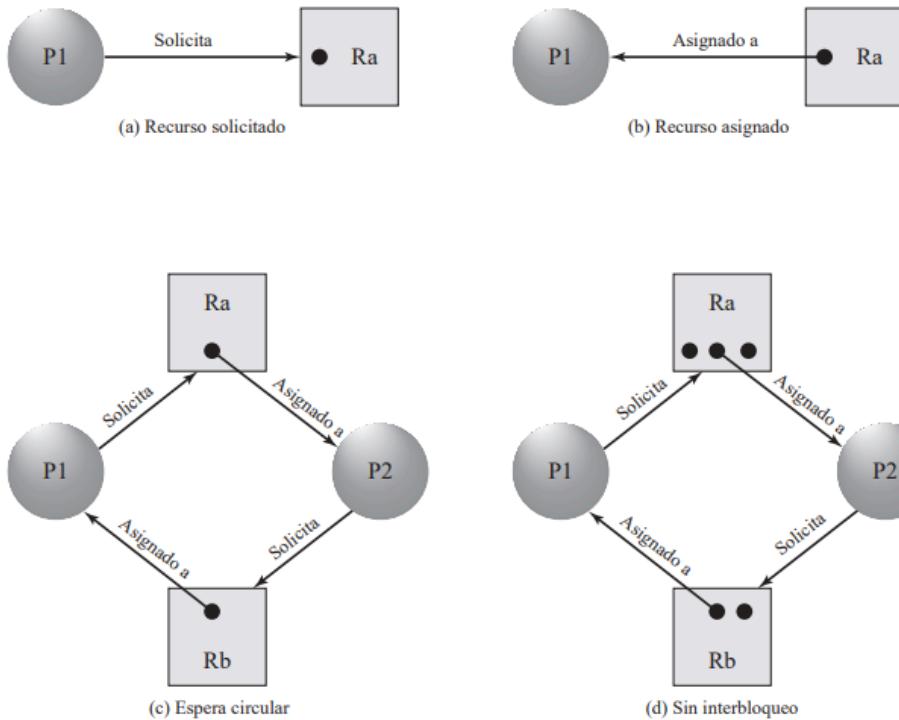
Recursos consumibles

- 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](#)”.
- 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 permanente de un conjunto de procesos que compiten por recursos del sistema
- 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 necesario 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
- Predicción
- Detección

● **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** (pérdida 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](#)).

Algoritmo del banquero (Saber algo de la matriz)

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.

	R1	R2	R3		R1	R2	R3		R1	R2	R3	
P1	3	2	2	-	P1	1	0	0	P1	2	2	2
P2	6	1	3		P2	6	1	2	P2	0	0	1
P3	3	1	4		P3	2	1	1	P3	1	0	3
P4	4	2	2		P4	0	0	2	P4	4	2	0

Matriz de necesidad N Matriz de asignación A N-A

R1	R2	R3
9	3	6

Vector de recursos R

R1	R2	R3
0	1	1

Vector de disponibles D

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.

	R1	R2	R3	R4	R5
P1	0	1	0	0	1
P2	0	0	1	0	1
P3	0	0	0	0	1
P4	1	0	1	0	1

Matriz de solicitud S

	R1	R2	R3	R4	R5
P1	1	0	1	1	0
P2	1	1	0	0	0
P3	0	0	0	1	0
P4	0	0	0	0	0

Matriz de asignación A

	R1	R2	R3	R4	R5
	2	1	1	2	1

Vector de recursos

	R1	R2	R3	R4	R5
	0	0	0	0	1

Vector de disponibles

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:

- 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.

Capítulo 7: Gestión de memoria (pág. 305) SE VUELVE A COMPLICAR UN POCO - Lo más importante es paginación y segmentación

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.**

La memoria principal ofrece un acceso rápido con un coste relativamente alto. Además, la memoria principal es volátil; esto es, no proporciona almacenamiento permanente. La memoria secundaria es más lenta y barata que la memoria principal y, normalmente, no es volátil. De este modo, una memoria secundaria de gran capacidad puede permitir un almacenamiento a largo plazo de programas y datos, al tiempo que una memoria principal pequeña mantiene los programas y datos de uso actual. Leer esto y BORRARLO

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.

Segmento: es una región contigua y lógica de la memoria que puede ser asignada a un proceso. Cada segmento tiene una dirección virtual y puede tener una longitud desigual o variable, lo que significa que puede contener diferentes cantidades de datos según las necesidades del programa.

Superposición: técnica para que un proceso pueda ser mayor que la cantidad de memoria que se le ha asignado. Busca mantener en la memoria sólo las instrucciones y datos que se necesitan en un momento dado. Si se requieren otras instrucciones, se cargan en un espacio que antes estaba ocupado por instrucciones que ya no se necesitan.

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)

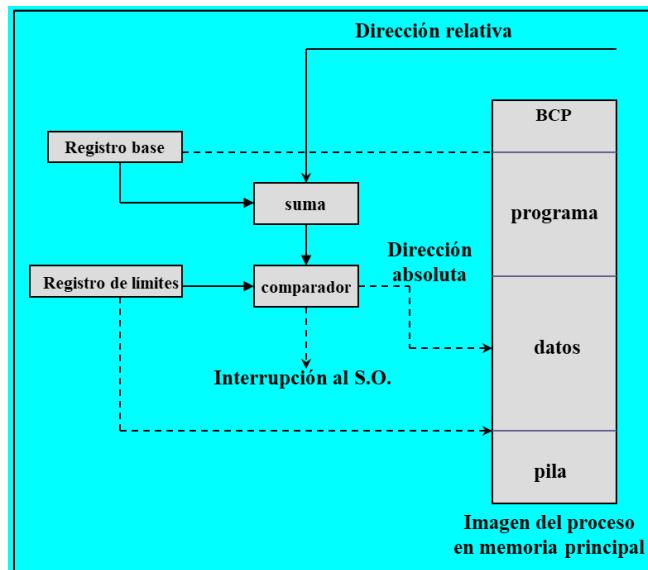
Se necesita un mecanismo de traducción porque los programas hablan con direcciones virtuales/lógicas, mientras que el procesador habla con las direcciones reales/físicas.

• 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).
- Idioma en el que hablan los procesos.

• Direcciones relativas

- Dirección calculada como un desplazamiento a partir de una dirección de base.
- 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 límites: requisito de protección con respecto a que no acceda en memoria de otro proceso.



• Direcciones reales (físicas)

- Direcciones actuales de memoria. Direcciones de la memoria principal, es el idioma en el que habla el procesador.

Siempre que haya reubicación (ya sea por swapping o compactación) hay que hacer un mapeo entre las direcciones virtuales que maneja un proceso y las direcciones reales que maneja el procesador.

• 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):

1. 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](#).

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:**

- 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).

2. Partitionamiento 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>*](#).
- 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**

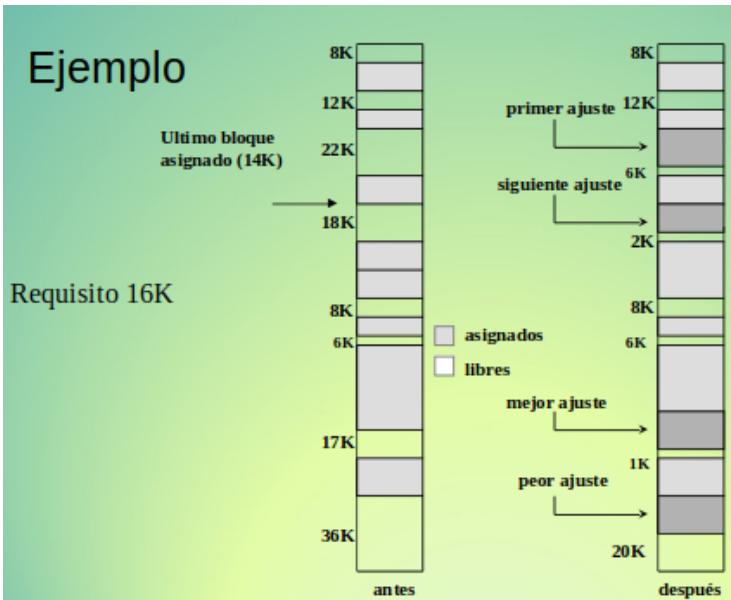
- Asigna el mayor hueco disponible.
- Se necesita menos compactación.
- Hay que analizar la memoria para saber los huecos que hay (una lista ordenada por tamaño con la dirección y tamaño del hueco).

- **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.

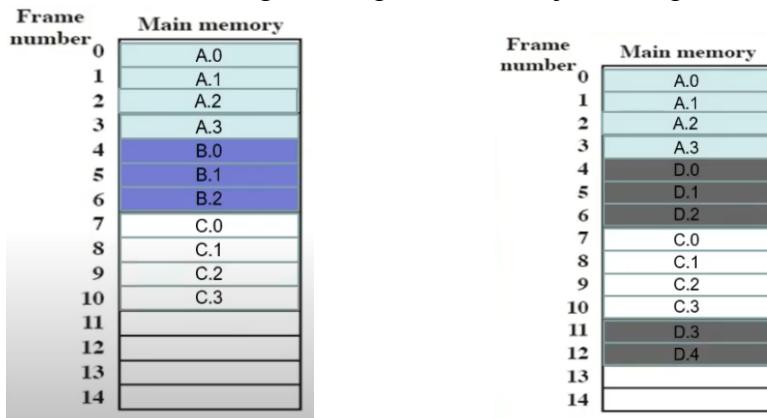
- **Estrategia del siguiente ajuste**

- Variante del primer ajuste, empieza a buscar desde donde terminó la búsqueda anterior
- Se fragmenta más el gran espacio libre del final de la memoria
- Se requiere compactación más frecuente que la anterior



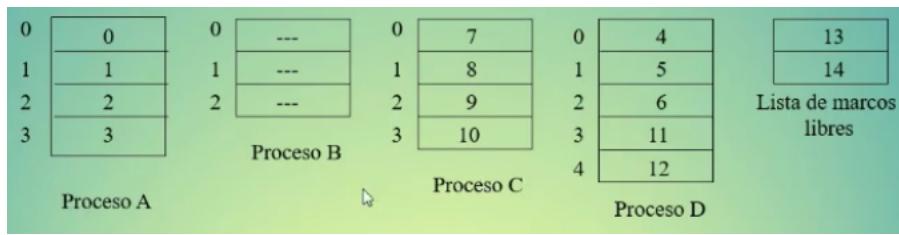
3. Paginación sencilla **IMPORTANTÍSIMO CASI TODO AGUS**

- La **memoria principal** se divide en un número de **marcos** relativamente pequeños de igual tamaño
- Cada **proceso** se divide en un número de **páginas** del mismo tamaño que los marcos
- Un proceso se carga colocando todas sus páginas en marcos disponibles, no necesariamente contiguos. Al no ser necesariamente contiguos se aprovechan mejor los espacios.



Ejemplos 2da tablita: la dirección 28 del Proceso A estaría en la dirección 28 de la memoria principal, la dirección 12 del Proceso C estaría en la dirección 82 de la memoria principal y , la dirección 35 del Proceso D estaría en la dirección 115 de la memoria principal. En este caso esto es así porque cada página es de tamaño 10.

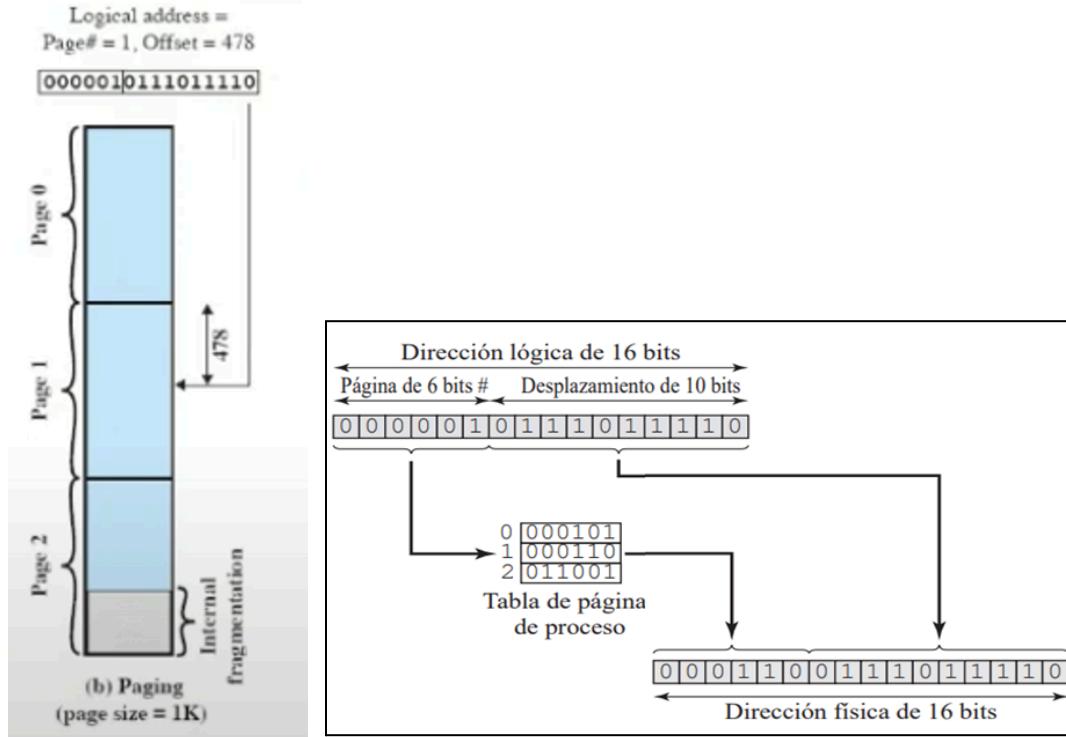
Se debe tener 1 tabla por proceso que indique el marco:



- Dentro de cada marco tengo n desplazamientos. Por ejemplo dirección de memoria 104, sería 10 el número de marco y dentro del marco el desplazamiento 4.
- 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 de protección está incluido.**
- Contra más páginas y marcos de menor tamaño menor será la fragmentación interna, pero aumentará el cómputo 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.
- En la paginación las tablas de cada proceso decía el número de marco en el que se encontraba cada proceso en la memoria principal.

Página	Marco
0	4
1	5
2	6
3	11
4	12

- 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... .

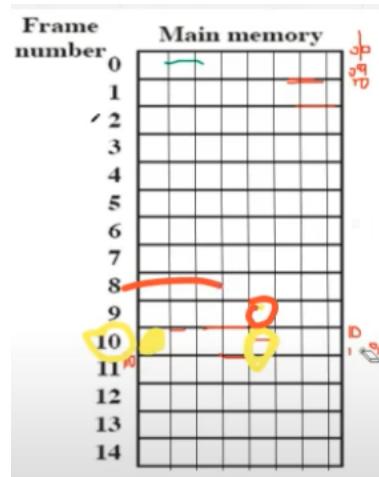


Para hacer la traducción solo hay que reemplazar con los valores de la tabla (es rápido), el desplazamiento siempre será el mismo.

EXPLICACIÓN TRADUCCIÓN PAGINACIÓN SIMPLE:

https://cursos.clavijero.edu.mx/cursos/182_so/modulo3/contenidos/tema3.2.4.html?opc=1

Por ejemplo dirección de memoria 104, sería 10 el número de marco y dentro del marco el desplazamiento 4. Es decir, la dirección absoluta es 104 y la dirección relativa es desplazamiento 4 del marco 10.

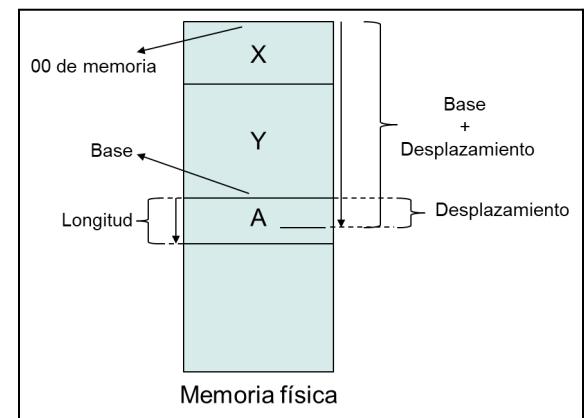
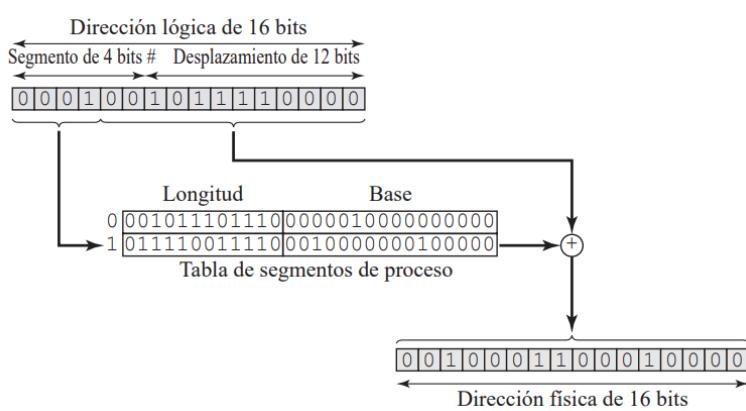


4. Segmentación sencilla AGUS

Acá la memoria no se divide sino que se deja como un solo hueco. Los procesos se dividen en segmentos (para que esos segmentos sean más chicos que todo el proceso completo). A medida que se vaya solicitando se va a ir cargando el segmento de un proceso en el lugar que quepa.

Los segmentos son las diferentes partes o regiones que componen el espacio de direcciones virtuales de un proceso. Cada segmento almacena un tipo específico de información y tiene una función bien definida para facilitar el funcionamiento adecuado del programa y la gestión eficiente de la memoria.

- 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.
- Estos segmentos pueden incluir el código del programa (segmento de código), los datos globales (segmento de datos), la pila de ejecución (segmento de pila), tablas de búsqueda, estructuras de datos compartidos, y otros datos específicos del programa.
- Ofrece seguridad
- Esquema de manejo de memoria es visible para el programador (esto significa que el programador sabe cuántos bits van destinados a la página y cuánto al desplazamiento)
- Similar a particiones variables/dinámicas, 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).
- Lista de bloques libres en memoria principal
- Cuando un proceso pasa al estado Ejecutando, se carga la dirección de su tabla de segmentos en un registro especial del hardware de gestión de memoria.



- 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 cuánto me muevo).

EXPLICACIÓN TRADUCCIÓN SEGMENTACIÓN SIMPLE:

https://cursos.clavijero.edu.mx/cursos/182_so/modulo3/contenidos/tema3.2.5.html?opc=1

5. Paginación con memoria virtual

[Capítulo 8](#)

6. Segmentación con memoria virtual

Capítulo 8

Capítulo 8: Memoria virtual (pág. 339)

Página explicativa:

https://cursos.clavijero.edu.mx/cursos/182_so/modulo3/contenidos/tema3.3.html?opc=2

Diccionario

Conjunto residente: contiene las partes más activamente utilizadas del programa y datos en la memoria principal, lo que mejora el rendimiento y la eficiencia del proceso.

Fallo de página: ocurre cuando un programa informático necesita acceder a una dirección de memoria que no se encuentra en la memoria principal (RAM) actualmente.

Es una técnica que utiliza el sistema operativo para expandir el espacio de memoria disponible para los programas que se ejecutan en la computadora. Permite que un programa utilice más memoria de la que realmente tiene disponible en la memoria RAM. Parte del espacio de almacenamiento en el disco duro o en el SSD se utiliza como una extensión de la memoria RAM. Cuando la memoria RAM se agota, los datos que no se están utilizando activamente se trasladan a la memoria virtual para liberar espacio en la memoria real para otros procesos.

El acceso a la memoria virtual es más lento que el acceso directo a la memoria RAM porque implica el uso del disco duro o SSD, que son dispositivos más lentos. Sin embargo, la memoria virtual permite que la computadora ejecute aplicaciones más grandes o varias aplicaciones al mismo tiempo, incluso si la memoria RAM es limitada.

Ejecución de un programa

1. Cuando un programa comienza a ejecutarse, el sistema operativo trae a la memoria principal (RAM) la parte inicial del programa que se necesita para iniciar su ejecución. Esta parte inicial del programa, junto con cualquier información adicional que el programa pueda requerir inmediatamente, forma el **conjunto residente**.
2. A medida que el programa se ejecuta y necesita acceder a datos o instrucciones que no se encuentran en la memoria principal, se produce un **fallo de página**.
3. Cuando se produce un fallo de página, el sistema operativo toma el control y realiza una solicitud de E/S (entrada/salida) para traer el fragmento de memoria faltante desde el almacenamiento secundario (disco) a la memoria principal (RAM). El proceso que causó el fallo de página se bloquea.
4. El sistema operativo puede cambiar la ejecución a otro proceso que esté listo para ser ejecutado.
5. Una vez que se completa la operación de E/S y el fragmento necesario del programa se encuentra en memoria principal, el proceso que estaba bloqueado debido al fallo de página se coloca nuevamente en estado "Listo".

Ventajas de fragmentar procesos

Ya sea paginación o segmentación:

- **Más procesos se pueden mantener en memoria principal** porque solamente se cargan algunos fragmentos.
- **Utilización más eficiente del procesador** porque hay más procesos en Listo.

- Un proceso puede ser más grande que toda la memoria principal. Puede 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)

- Esto ocurre cuando el sistema está constantemente trayendo páginas o segmentos de memoria al espacio de memoria principal y luego los envía de vuelta al almacenamiento secundario poco después, sin que realmente puedan ser utilizados efectivamente. Esto se produce cuando la memoria disponible en la memoria principal es insuficiente para contener todos los fragmentos necesarios para que los procesos se ejecuten de manera eficiente.
- El problema principal con la hiperpaginación es que consume una cantidad significativa de tiempo de CPU y recursos de E/S (entrada/salida) al realizar continuos movimientos de fragmentos entre la memoria principal y el almacenamiento secundario. Como resultado, el rendimiento general del sistema se ve gravemente afectado, ya que el procesador pasa más tiempo ocupado realizando estas operaciones de intercambio en lugar de ejecutar las instrucciones del usuario.

Principio de cercanía/proximidad (anticipo)

Se utiliza para tratar de prever qué fragmentos de memoria serán más probables de ser utilizados en un futuro cercano por un proceso en ejecución. El objetivo es mejorar el rendimiento y evitar la hiperpaginación o thrashing.

- Las referencias a datos y programas dentro de un proceso tienden a agruparse en ubicaciones contiguas de memoria.
- 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 alta probabilidad de que sean las primeras páginas de memoria, ya que son las que se acceden inicialmente al iniciar un proceso o realizar una transición entre diferentes partes del programa.
- **Principio de Localidad espacial:** tendencia a referenciar localidades cercanas entre sí en la memoria. Esto ocurre cuando un programa accede a una dirección de memoria específica y es probable que también acceda a direcciones cercanas en el espacio de memoria. Las situaciones comunes que aprovechan la localidad espacial incluyen:
 - **Secuencias lineales de código:** Cuando el programa ejecuta instrucciones en orden secuencial, es probable que acceda a las instrucciones cercanas en la memoria.
 - **Recorridos de vectores:** Cuando un programa accede a elementos de un vector o arreglo, es probable que acceda a elementos cercanos, ya que están almacenados contiguamente en la memoria.
 - **Definiciones de variables afines cercanas unas a otras:** Las variables que están relacionadas o se utilizan juntas en un programa tienden a estar almacenadas cerca unas de otras en la memoria.
- **Principio de Localidad temporal:** Se refiere a la tendencia a referenciar la misma posición de memoria varias veces durante breves intervalos de tiempo. Esto significa que los datos o instrucciones que se han utilizado recientemente tienen una alta probabilidad de volver a utilizarse pronto. Situaciones que aprovechan la localidad temporal incluyen:
 - **Ciclos y bucles:** Durante la ejecución de un ciclo, el programa accede repetidamente a las mismas posiciones de memoria.

- **Subrutinas y funciones:** Cuando un programa llama a una subrutina o función, es probable que acceda a las instrucciones y datos de la subrutina varias veces antes de que esta finalice.
- **Pilas:** Las pilas se utilizan para gestionar la ejecución de subrutinas y llamadas a funciones, y se accede repetidamente a las mismas posiciones de la pila durante la ejecución del programa.
- **Variables utilizadas para contar y totalizar:** Las variables que se utilizan para llevar cuentas o sumar valores tienden a ser accedidas varias veces mientras se realiza el conteo o la totalización.

Soprote 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 MITI MITI AGUS Y TINCHO

https://cursos.clavijero.edu.mx/cursos/182_so/modulo3/contenidos/tema3.3.1.html?opc=2

Sólo algunas de las páginas de un proceso pueden estar en memoria principal

Cuando se está ejecutando un proceso en particular, la dirección de comienzo de la tabla de páginas para este proceso se mantiene en un registro. Se almacena en el BCP.

Las tablas, si son muy grandes, también se pueden almacenar una parte en memoria virtual.

Cuando un proceso se ejecuta, parte de su tabla está en la memoria principal.

Cada proceso tiene su propia tabla de páginas:

- Incluye el número de marco para esa página.
- Número de página
- **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.
- **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 evitó hacer una escritura en disco que es muchísimo más lenta.
- Puede haber también **otros bits de control**. Por ejemplo, si la protección o la compartición se gestiona a nivel de página, se necesitarán más bits con tal propósito.

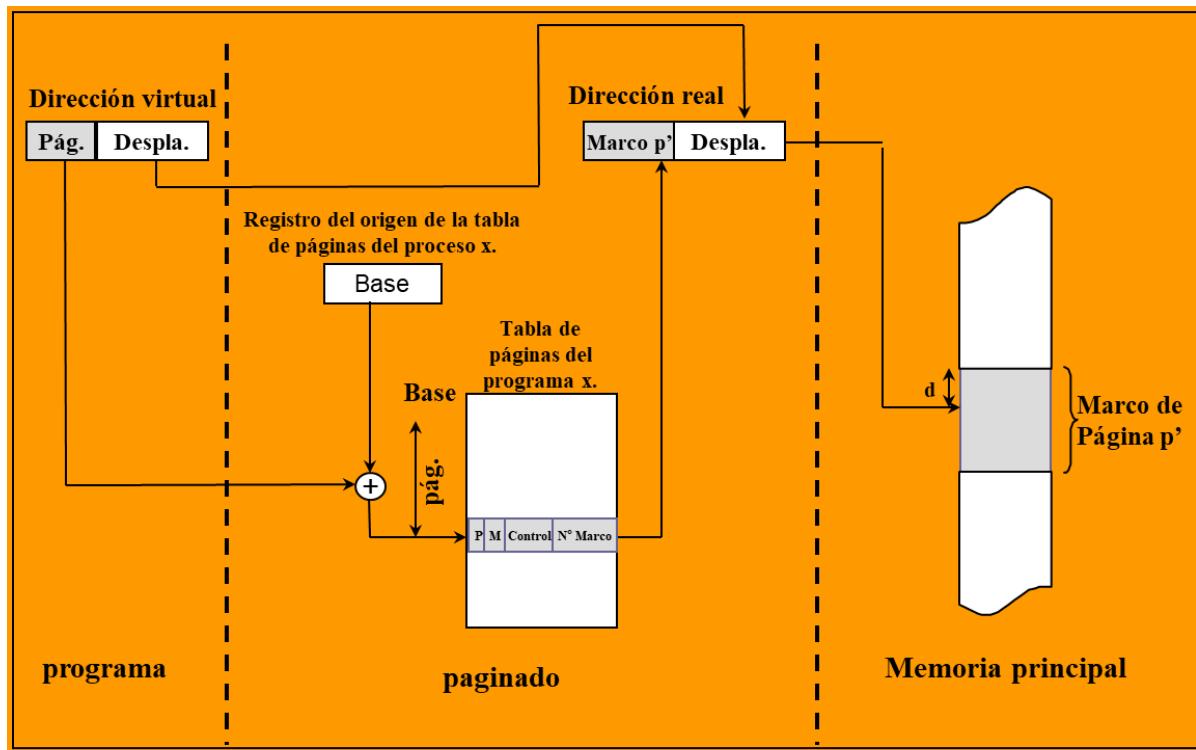




- En la tabla habrá: Cantidad pág. \geq cantidad marco (se utiliza el bit P).
- Cada referencia a una dirección virtual puede causar dos accesos a memoria física.
 - Uno para leer la tabla.
 - Uno para leer el dato.

Esto doblaría el tiempo de acceso a memoria, entonces para mejorar esto se usa una caché especial para las entradas de las tablas llamada TLB (Translation Lookaside Buffer).

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 presencia 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.

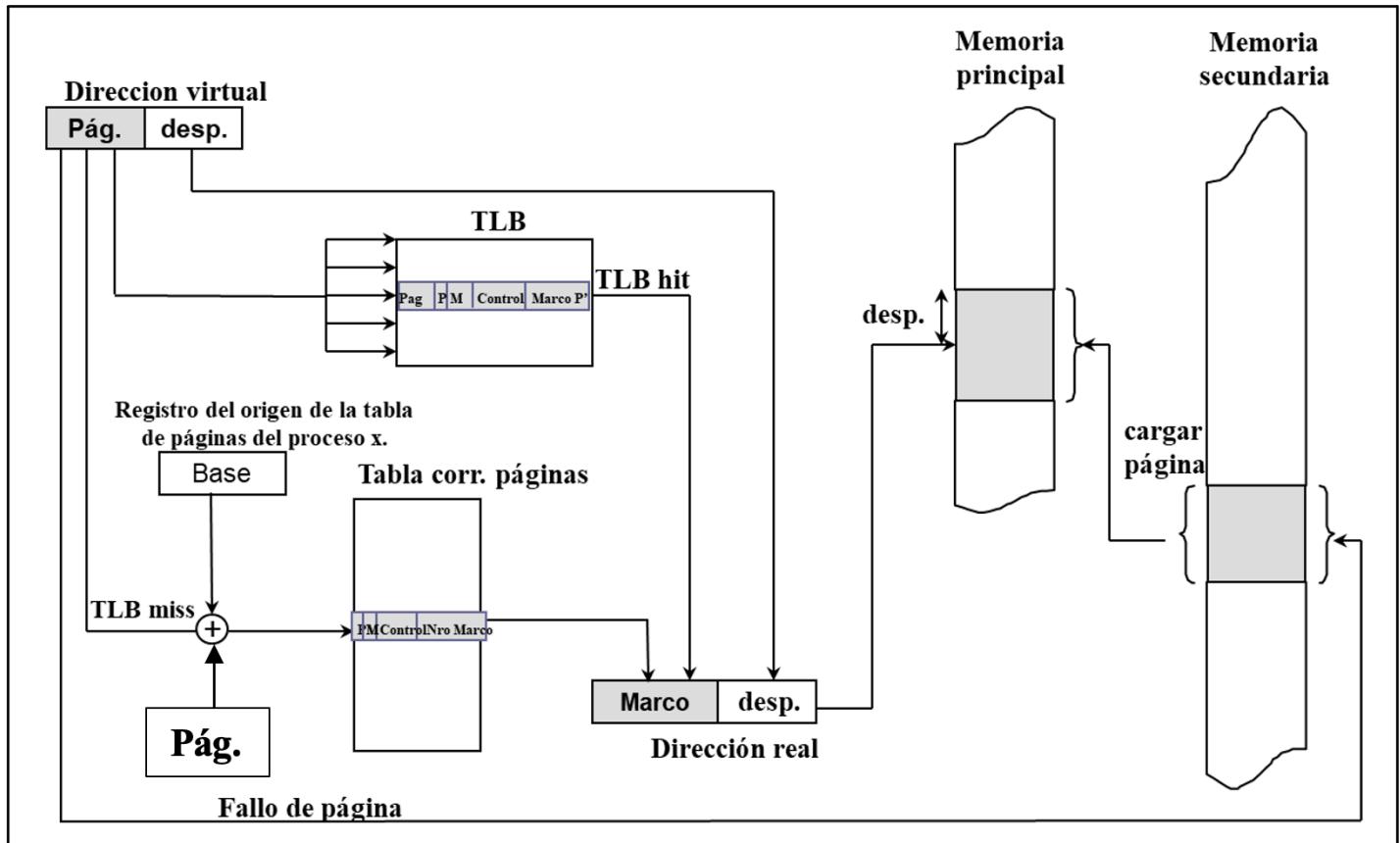
Cache TLB (pág. 349) LO TOMAN, ES RE FÁCIL, (Mirar un poco el grafico)

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 caché de memoria principal.

Dada una dirección virtual, el procesador examinará primero la TLB. Si la entrada de tabla de páginas buscada está presente (un acierto en la TLB), se obtiene el número de marco y se forma la dirección real. Si no se encuentra la entrada de la tabla de páginas buscada (un fallo de TLB), el procesador emplea el número de página para buscar en la tabla de páginas del proceso y examinar la entrada correspondiente de la tabla de páginas. Si se encuentra activo el bit de presencia, es que la página está en memoria principal y el procesador puede obtener el número de marco de la entrada de la tabla de páginas para formar la dirección real. El procesador, además, actualiza la TLB para incluir esta nueva entrada de la tabla de páginas. Por último, si el bit de presencia no está activo, es que la página buscada no está en memoria principal y se produce un fallo en el acceso a memoria, llamado fallo de página. En este punto, se abandona el ámbito del hardware y se invoca al sistema operativo, que carga la página necesaria y actualiza la tabla de páginas.

La lectura de págs. es en paralelo(múltiples páginas pueden ser leídas y escritas en la memoria secundaria (como el disco duro) o en la memoria RAM al mismo tiempo, en lugar de tener que hacerlo de forma secuencial y no indexada (no se leen o escriben una por una de manera secuencial, sino que se accede a ellas mediante la utilización de estructuras de datos que permiten acceder directamente a la página deseada.).



Tamaño de página

- El SO determina la cantidad de memoria reservada para cada proceso y el hardware el tamaño de página.
- **A menor tamaño:**

- Menor fragmentación interna, más páginas por procesos y 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 la 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

Tamaño de página pequeño: Si el tamaño de página es pequeño, cada página en memoria contendrá solo una pequeña porción del proceso. Dado que los programas tienden a acceder a ubicaciones cercanas en un corto período de tiempo (localidad espacial) y acceder repetidamente a las mismas ubicaciones (localidad temporal), la información relevante estará disponible en la memoria principal. Esto significa que la tasa de fallos de página será baja, ya que el sistema no necesitará buscar frecuentemente datos adicionales en el almacenamiento secundario.

Tamaño de página grande: Si el tamaño de página es grande, cada página en memoria contendrá una mayor cantidad de información del proceso y abarcará un rango más amplio de direcciones. Esto disminuye la efectividad de la localidad espacial y temporal, ya que es menos probable que todas las direcciones que el programa necesite estén presentes en la memoria principal. Como resultado, la tasa de fallos de página puede aumentar, ya que el sistema deberá buscar más páginas en el almacenamiento secundario cada vez que el programa intente acceder a una dirección que no está en la memoria principal.

Segmentado con memoria virtual

https://cursos.clavijero.edu.mx/cursos/182_so/modulo3/contenidos/tema3.3.2.html?opc=2

- Los segmentos pueden ser de distintos tamaños, incluso de forma dinámica.
- Permite crecimiento de estructuras(La memoria virtual segmentada es una técnica que asigna a cada estructura de datos su propio segmento de memoria. Esto permite que el sistema operativo expanda o reduzca el segmento de memoria asignado a una estructura según sea necesario.), modularidad(Permite modificar y recompilar los programas independientemente, sin que sea necesario recompilar o volver a montar el conjunto de programas por completo.) , soporte para compartir(Un programador puede situar un programa de utilidades o una tabla de datos en un segmento que pueda ser referenciado por otros procesos.) y proteger(programador o el administrador del sistema podrá asignar los permisos de acceso de la forma adecuada.).
- El programador la percibe.

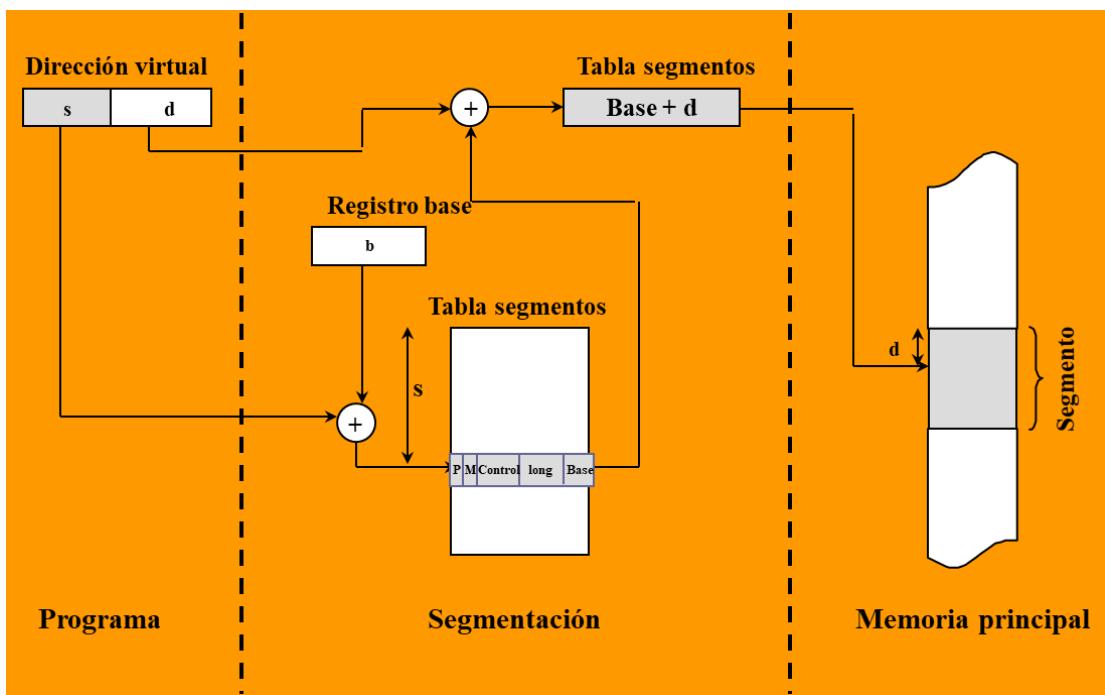
Tabla de segmentos:

- 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: si se gestiona la protección o la compartición a nivel del segmento, se necesitarán bits con tal propósito

- Cada entrada contiene la **longitud** del segmento.
- Cada entrada contiene la **dirección de comienzo** del correspondiente segmento en memoria principal. Sería la **Base**



Traducción



Sistema combinado de paginación/segmentación LICCI ESTUDIO POCO ESTO

https://cursos.clavijero.edu.mx/cursos/182_so/modulo3/contenidos/tema3.3.3.html?opc=2

La paginación es transparente al programador(significa que el programador no necesita preocuparse por la asignación y administración de memoria a nivel de páginas. La administración de páginas es realizada por el sistema operativo.) y **elimina la fragmentación externa**(las páginas pueden ser asignadas de manera no contigua en la memoria física, permitiendo un uso más eficiente de la memoria disponible).

La segmentación sí es visible al programador(significa que el programador debe encargarse de la asignación y administración de memoria a nivel de segmentos), **permite la posibilidad de manejar estructuras de datos que crecen**(cada segmento puede expandirse o reducirse dinámicamente según sea necesario), **modularidad**(cada segmento puede representar una parte independiente y coherente del programa), **dar soporte a la compartición y a la protección**(compartir datos y código entre ellos de manera controlada).

En un sistema con paginación y segmentación combinadas, el espacio de direcciones de un usuario se divide en varios segmentos según el criterio del programador. Cada segmento se vuelve a dividir en varias páginas de tamaño fijo, que tienen la misma longitud que un marco de memoria principal.

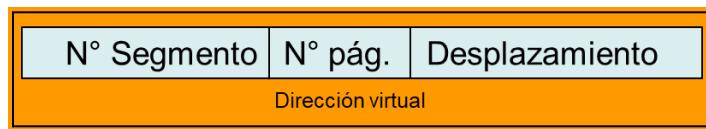
Asociada a cada proceso existe una tabla de segmentos y varias tablas de páginas, una por cada uno de los segmentos.

Si el segmento tiene menor longitud que la página, el segmento ocupará sólo una página.

Desde el punto de vista del programador, una dirección lógica también está formada por un número de segmento y un desplazamiento en el segmento.

Desde el punto de vista del sistema, el desplazamiento del segmento se ve como un número de página dentro del segmento y un desplazamiento dentro de la página.

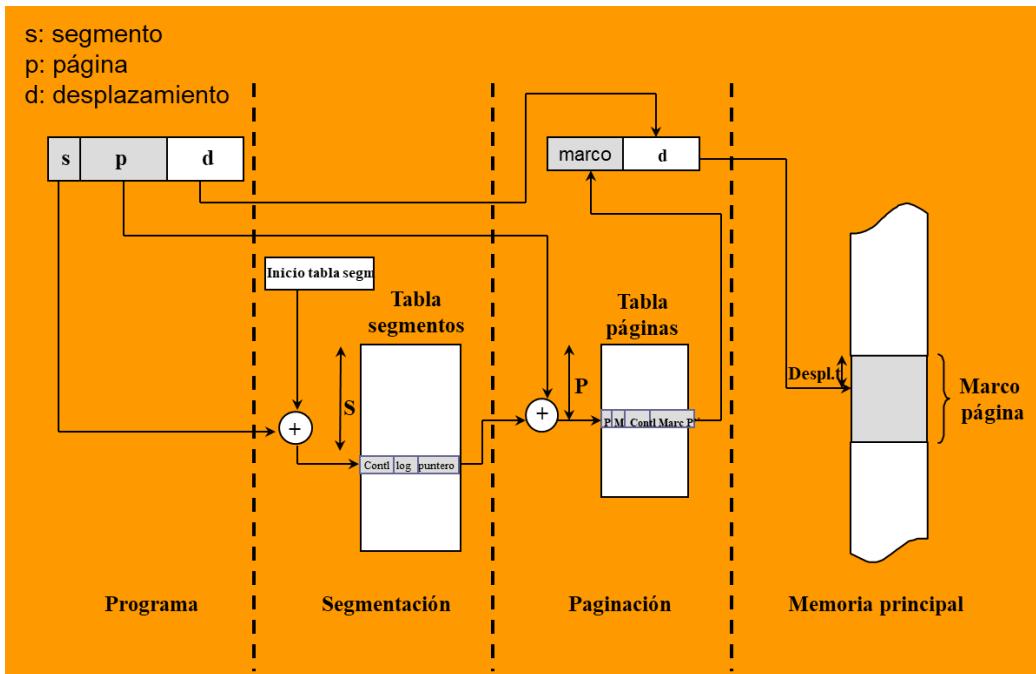
Cuando un proceso determinado está ejecutándose, un registro contendrá la dirección de comienzo de la tabla de segmentos para ese proceso. Dada una dirección virtual, el procesador emplea la parte de número de segmento como índice en la tabla de segmentos del proceso para encontrar la tabla de páginas de dicho segmento. Entonces, la parte de número de página de la dirección virtual se usará como índice en la tabla de páginas para localizar el número de marco correspondiente. Éste se combina con la parte de desplazamiento de la dirección virtual para generar la dirección real deseada.



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



Traducción



Políticas que lleva el SO para poder gestionar la memoria virtual (6) LE TOMARON ESTO AL LICCI

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

1. Lectura

Determina cuándo cargar una página a memoria

2. Vaciado

Cuando sacar un proceso.

3. Ubicación

Dónde colocar la nueva página/segmento.

4. Reemplazo

Cuál página/segmento desalojar de memoria.

5. Asignación

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

6. Control de carga

Grado de multiprogramación.

1. 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.
- Es más costoso, ya que cada vez que se necesita una página que no está en memoria, se produce un fallo de página.

- Paginación anticipada (principio de cercanía):

- Trae más páginas que las que se necesitan. Se asume que si una página está siendo referenciada, es probable que las páginas contiguas también se utilicen pronto debido al principio de cercanía (localidad temporal y espacial).
- Acelera tiempos de ejecución de un proceso. Ya que cuando se requiere una página, es probable que las páginas contiguas ya estén en memoria.
- Más eficiente cuando las páginas son contiguas en el disco, se usa en la carga inicial.
- No se puede predecir con certeza qué páginas serán necesarias, lo que puede resultar en traer páginas innecesarias a la memoria y ocupar más espacio.

En la carga inicial de un proceso, se pueden traer varias páginas a la memoria anticipadamente para aprovechar el principio de cercanía. Luego, a medida que el proceso se ejecuta, las páginas adicionales se traen por demanda según las necesidades reales del programa.

2. Política de vaciado

De memoria principal a disco.

- **Vaciado por demanda:** una página se escribe en disco solamente cuando ha sido seleccionada para reemplazo. Cuando ocurre un fallo de página y se necesita espacio en memoria para cargar una nueva página, la página menos utilizada se selecciona para ser escrita en el disco. Minimiza la cantidad de escrituras en disco ya que solo se escriben las páginas que son reemplazadas debido a que no están actualmente en uso en la memoria.
- **Prevaciado:** Se realiza buffering de páginas, las páginas se escriben en disco por lotes.

3. Política de ubicación

Determina dónde ubicar un bloque en memoria real.

- Irrelevante en el caso de paginación.
- Igualas estrategias que en particiones variables para segmentación
 - mejor ajuste
 - peor ajuste
 - primer ajuste
 - siguiente ajuste

4. Política de reemplazo

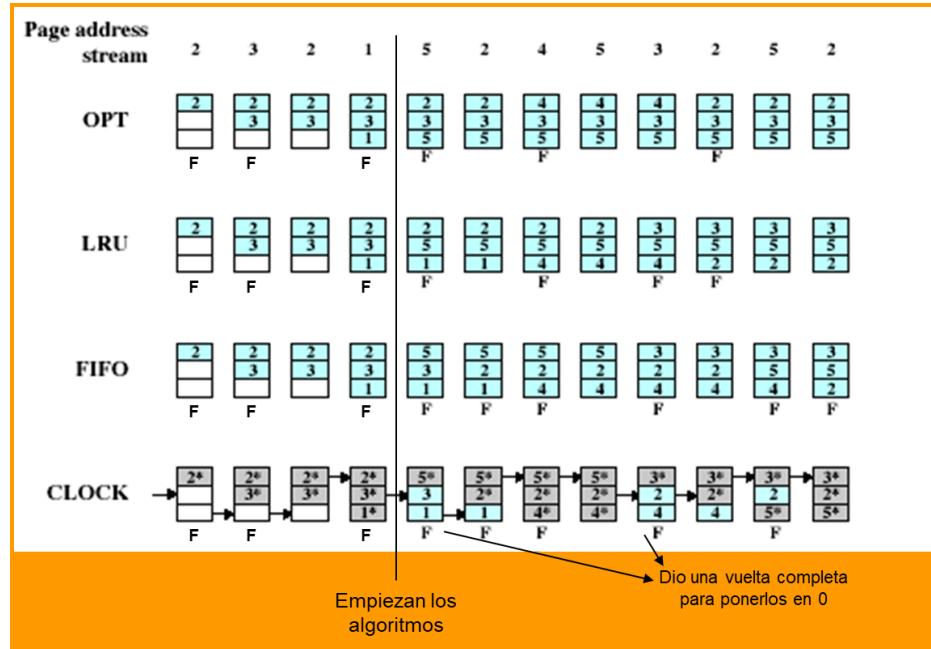
Determina cuál es la página específica que debe elegirse para el reemplazo.

Bloqueo de marcos:

- Cuando un marco está bloqueado, la página actualmente almacenada en dicho marco no puede reemplazarse. Gran parte del núcleo del sistema operativo se almacena en marcos que están bloqueados así como otras estructuras de control claves.
- 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.
- Genera el menor número de fallos de página.
- Imposible de implementar ya que requiere que el SO tenga un conocimiento exacto de los sucesos futuros.

LRU (menos recientemente utilizada):

- Reemplaza la página que no ha sido utilizada por el mayor tiempo
- Se basa en el principio de localidad, que establece que los programas tienden a acceder a las mismas páginas de memoria que han utilizado recientemente.
- La implementación es difícil. El seguimiento de todas las referencias a cada página para determinar cuál fue la menos usada es costoso.
 - Mediante el uso de un contador de tiempo. Cada vez que una página es referenciada, su contador se actualiza con el tiempo actual. Cuando es necesario reemplazar una página, se selecciona aquella que no ha sido utilizada por el mayor tiempo.
 - Mantener una lista ordenada de páginas en memoria, donde las páginas más recientemente utilizadas se mueven al frente de la lista, y las menos utilizadas se encuentran al final. Cuando es necesario reemplazar una página, se selecciona aquella al final de la lista (la menos recientemente utilizada).

NUR (no utilizada recientemente)

- Aproximación de la política LRU.
- Utiliza dos bits de hardware para cada página en la tabla de páginas:
 - Bit de uso (Use bit): Se marca o establece cuando la página es accedida o referenciada, es decir, cuando se produce una lectura en la página.
 - Bit de modificación (Modify bit o Dirty bit): Se marca o establece cuando la página es modificada, es decir, cuando se realiza una escritura en la página.
- Cuando es necesario reemplazar una página, el sistema revisa los bits de uso y modificación de todas las páginas en memoria para determinar cuál será reemplazada.

Clase 0 (Not used, Not modified): Páginas que no han sido utilizadas (bit de uso = 0) y no han sido modificadas (bit de modificación = 0).

Clase 1 (Not used, Modified): Páginas que no han sido utilizadas (bit de uso = 0) pero han sido modificadas (bit de modificación = 1).

Clase 2 (Used, Not modified): Páginas que han sido utilizadas (bit de uso = 1) pero no han sido modificadas (bit de modificación = 0).

Clase 3 (Used, Modified): Páginas que han sido utilizadas (bit de uso = 1) y han sido modificadas (bit de modificación = 1).

La página seleccionada para reemplazo será aquella que pertenezca a la Clase 0, si no hay páginas de Clase 0, se elegirá una de Clase 1, y así sucesivamente.

FIFO

- Reemplaza la página que ha estado más tiempo en almacenamiento primario.
- Las páginas se mantienen en una lista tipo buffer circular, generalmente implementada como una cola FIFO.
- La página más antigua puede ser la más usada
- La más simple de implementar.
- Tiene una limitación conocida como la "anomalía de Belady", donde el aumento de la memoria RAM puede llevar a un aumento en el número de faltas de página en lugar de una disminución

Reloj o Segunda oportunidad: ESTO ES UNA GILADA, EL LICCI NO LO VIO

- Aproximación de FIFO mejorando el rendimiento a través del bit de uso.
- 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.

Buffering de páginas. (Esto me lo preguntaron)

- Utiliza el algoritmo FIFO.
- Las páginas no se reemplazan inmediatamente, se agregan a una de las dos listas:
 - Lista de marcos libres: Esta lista almacena las páginas que se han cargado en memoria principal pero que aún no han sido modificadas. Esto significa que estas páginas están disponibles para su uso inmediato, ya que no se necesita realizar ninguna operación de escritura en el disco para mantener los cambios. Al no tener cambios, también se pueden reemplazar fácilmente si es necesario.
 - Lista de páginas modificadas: En esta lista se mantienen las páginas que se han modificado después de haber sido cargadas en memoria principal. Estas páginas necesitan ser escritas nuevamente en el disco antes de ser reemplazadas o desecharadas. Es decir, cuando una página se ha modificado, se coloca en esta lista para que se pueda guardar en el disco en algún momento futuro, ya que el proceso de escritura puede ser más lento.
- Las listas funcionan como caché: Cuando se realiza una operación de lectura o escritura en el disco, es conveniente mover bloques grandes de páginas en conjunto para aprovechar el acceso secuencial y reducir la sobrecarga de acceso a 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](#).

5. 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 y en cambio, podría afectar el rendimiento general del sistema al reducir la memoria disponible para otros procesos.

Asignación fija

- Número fijo de marcos asignados a un proceso en forma anticipada.
- Solo puede utilizar esa cantidad específica de memoria y no se le permite acceder a marcos de memoria adicionales.
- Se usan los algoritmos de reemplazo visto. Óptimo, LRU (menos recientemente utilizada), NUR (no utilizada recientemente), FIFO, Reloj o Segunda oportunidad
- Desventajas:
 - Asignación muy pequeña: alto grado de fallos de página.
 - Asignación muy grande: bajo grado de multiprogramación.

Asignación variable

- Asigna un número variable de marcos a cada proceso, en función de sus necesidades y del conjunto de trabajo actual (conjunto de páginas que un proceso está utilizando activamente durante un período de tiempo determinado. Estas páginas son las que el proceso necesita tener en memoria para su correcta ejecución).
- Dos enfoques de reemplazo de páginas en la asignación variable:
 - **Reemplazo local:** la página a reemplazar se elige entre los marcos de memoria asignados a ese proceso específico.
 - Asignación de un cierto número de marcos al proceso.
 - Cuando se produce un fallo se selecciona un marco del proceso.
 - De vez en cuando se vuelve a evaluar la asignación otorgada.
 - **Reemplazo global:** la página a reemplazar se elige entre todos los marcos de memoria disponibles en el sistema, independientemente del proceso al que estén asignados.
 - Ventaja: Fácil de implementar (no es necesario preocuparse por transmitir la información a través de diferentes partes del programa.)
 - Desventaja: Quita marcos de otros procesos activos (Si múltiples partes del código acceden y modifican la misma variable global, pueden surgir conflictos y comportamientos inesperados, dificultando el mantenimiento y la depuración del código.)
 - Se puede usar combinada con buffering de páginas para contrarrestar problemas (implica mantener un mecanismo de control para la modificación de la variable global, evitando que varios procesos intentan acceder y modificar su valor simultáneamente. Con el buffering de páginas, las actualizaciones en la variable global se almacenan temporalmente en un área de memoria reservada y se sincronizan en momentos adecuados para evitar conflictos y asegurar que el valor sea consistente en todas las partes del programa.)

6. Control de carga (pág. 376)

- Depende del 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 PÁGINA:

- **Procesos con la prioridad más baja**(son generalmente menos críticos).
- **Procesos con fallos de páginas**: este proceso no tiene 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
- **Procesos 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) MITI MITI. Esto me lo tomaron

La clave de la multiprogramación está en la planificación (varios procesos en MP para optimizar el uso de CPU), ya que en sí, la planificación es una “gestión de las colas” que minimice el tiempo de espera, de respuesta y de ejecución y maximice el rendimiento del entorno (uso de CPU y tasa de procesamiento).

Nivel de planificación

Los nombres de los planificadores hacen referencia a la frecuencia con la que estas funciones se ejecutan.

Largo plazo

- Determina qué programas son admitidos por el sistema para procesar.
- Se realiza al tomar la decisión de crear un proceso nuevo.
- Controla el grado de multiprogramación (cantidad de procesos en ejecución)
- A más procesos, menor porcentaje de tiempo para ejecutar cada proceso.

Mediano plazo

- Responsable de administrar el movimiento de procesos entre la memoria principal y la memoria secundaria(swapping)
- Basada en la necesidad de manejar multiprogramación

Corto plazo (Despachador)

- Determina qué proceso pasará del estado listo al estado en ejecución
- El despachador es invocado cuando el proceso en ejecución deja de estar en ejecución (interrupción, timer, etc.)

Entrada / Salida

- Determina cual de todas las solicitudes de espera de E/S será atendida por un dispositivo disponible



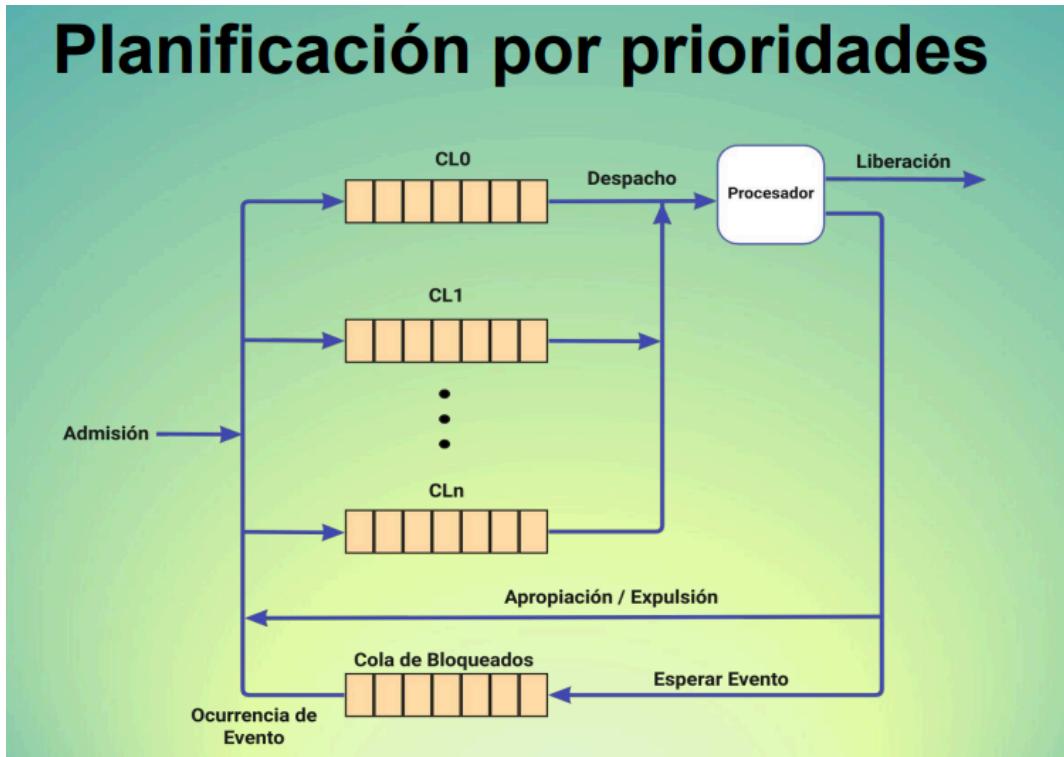
Criterios de planificación a corto plazo

	Usuario	Sistema
cuantitativo	Tiempo de retorno Tiempo de respuesta Plazos	Productividad Utilización del procesador
	Previsibilidad	Equidad Prioridades Equilibrio de recursos
cuantitativo		

Prioridades

- El despachador siempre elegirá al de mayor prioridad.
- Hay múltiples colas que representan cada nivel de prioridad.
- Los de baja prioridad pueden sufrir inanición.
 - Entonces puede permitir a un proceso cambiar su prioridad basado en su antigüedad o historial de ejecución

Planificación por prioridades



Función de selección

- Determina qué proceso se elige.
- Basada en prioridades, necesidades de recursos o características de ejecución de los procesos.

Modo de decisión

Especifica los instantes de tiempo en que se aplica la función de selección. Hay dos categorías:

- **Sin expulsión (no apropiativa):** Una vez que el proceso pasa a estado de ejecución, continúa ejecutando hasta que termina o hasta que se bloquea por E/S.
- **Con expulsión (apropiativa):** El proceso ejecutándose actualmente puede ser interrumpido y llevado al estado de listo por el sistema operativo. Permite mejor servicio ya que ningún proceso monopoliza el procesador por mucho tiempo.

Tipos de planificación a corto plazo

1. Primero en llegar, primero en ser servido (FCFS)
2. Round Robin (turno rotatorio)
3. Primero el más corto (SPN)
4. Tiempo restante más corto (SRT)
5. Primero el de mayor tasa de respuesta (HRRN)
6. Realimentación
7. Distribución justa

	Función de selección	Modo de Decisión	Rendimiento	Tiempo de respuesta	Sobre-carga	Efecto sobre los procesos	Inanición
FCFS	max[w]	no expulsiva	no especificado	puede ser alto especialmente si hay mucha diferencia entre los tiempos de ejecución de los procesos	mínima	penaliza procesos cortos; penaliza procesos con mucha e/s	no
Turno rotatorio(RR)	constante	expulsiva (por rodajas de tiempo)	puede ser mucho si la rodaja es demasiado pequeña	proporciona buen tiempo de respuesta para procesos cortos	mínima	tratamiento justo	no
SPN	min[s]	no expulsiva	alto	proporciona buen tiempo de respuesta para procesos cortos	puede ser alta	penaliza procesos largos	posible
SRT	max[s-e]	expulsiva	alto	proporciona buen tiempo de respuesta	puede ser alta	penaliza procesos largos	posible
HRRN	max[(w+s)/s]	no expulsiva	alto	proporciona buen tiempo de respuesta	puede ser alta	buen equilibrio	no
Realimentación	(ver texto)	expulsiva (por rodajas de tiempo)	no especificado	no especificado	puede ser alta	puede favorecer procesos con mucha e/s	posible

1. Primero en llegar, primero en ser servido (FCFS)

- Primero que entra, primero en salir.
- Es de modo sin expulsión
- No importan las prioridades.
- Favorece a los procesos con carga de CPU.
 - Procesos de E/S deben esperar hasta que los de carga de CPU se terminen
- Cuando el proceso actual termina de ejecutarse, se elige el proceso más antiguo de la cola de listos.
- Un proceso corto puede tener que esperar mucho tiempo antes de ser ejecutado.
- No se producen muertes por inanición.

2. Round Robin (turno rotatorio)

- Usa apropiación basada en reloj.
- Se determina la cantidad de tiempo fija que cada proceso usa el procesador. (ej: qtiempo=1)
- Es de modo con expulsión.
- No se producen muertes por inanición.

3. Primero el más corto (SPN)

- Política no apropiativa (sin expulsión).
- Se elige el proceso con el menor tiempo de procesamiento. Si hay 2 iguales se elige el primero que entró.
- Los procesos más cortos son favorecidos y ejecutados antes que los procesos más largos.
- Se reduce la previsibilidad de los procesos más largos.
- Si el tiempo estimado para el proceso no es correcto, el sistema operativo puede abandonarlo
- Posibilidad de inanición para procesos más largos.

4. Menor tiempo restante (SRT)

- Versión apropiativa de la política de primero el más corto
- Selecciona el proceso con el tiempo de ejecución restante más corto.
- Da prioridad a los procesos que requieren menos tiempo para terminar.
- Debe estimarse el tiempo de procesamiento.

- Posibilidad de inanición para procesos más largos.

5. Primero el de mayor tasa de respuesta (HRRN)

- Elige al que tiene mayor tasa de respuesta, mediante la siguiente fórmula:

$$\frac{\text{Tiempo consumido esperando} + \text{tiempo de servicio esperado}}{\text{tiempo de servicio esperado}}$$

- 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.
- No se producen muertes por inanición.

6. Realimentación

- Penaliza los trabajos que han estado ejecutándose por más tiempo
- 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.
- Los procesos largos son castigados.
- Es de modo con expulsión.

—RAID APUNTES DE LA CÁTEDRA Y VIRTUALIZACIÓN NOTION

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.

Conektor 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

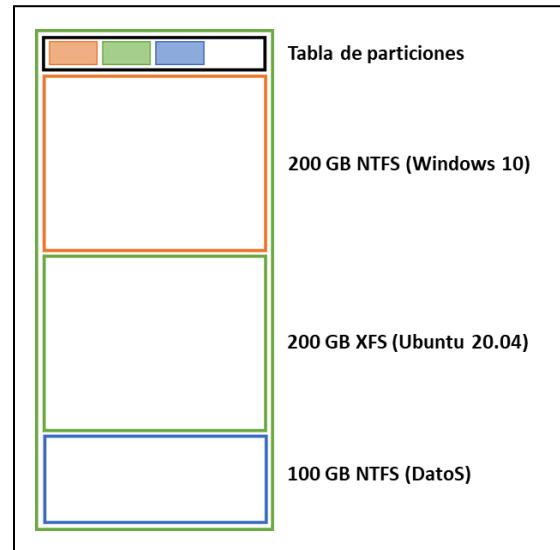
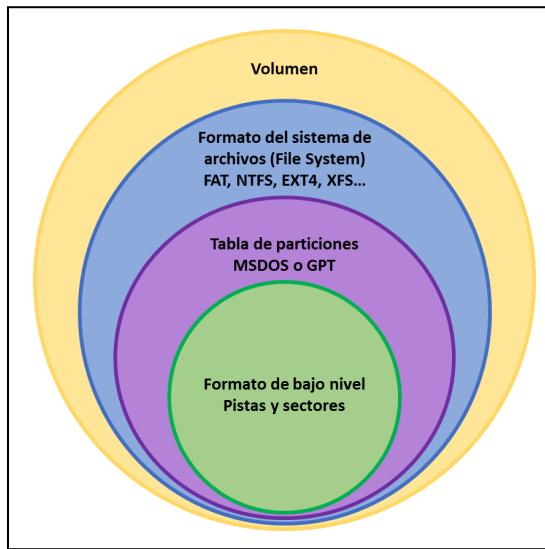
Tiempo de acceso

Memoria principal: 7 nanosegundos

Discos: 7 o más milisegundos

1.000.000 de veces más lento.

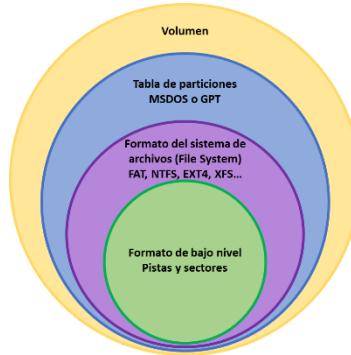
Niveles de un disco



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.

*Un **volumen de disco** es una porción de **disco** duro que funciona como si fuese un **disco** físico independiente. El **volumen** puede estar formado por una sola porción de un **disco** duro, por varias porciones del mismo **disco**, o varias porciones de varios **discos** duros

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

- RAID 0
 - No se usa nunca
- RAID 1
 - Se usa en discos de booteo de los servidores
 - Velocidad de Lectura puede ser en paralelo dependiendo de la controladora(El doble de un disco)
- RAID 4
 - Si se me rompe un disco que no sea el de paridad mi velocidad de lectura va a ser 3 veces más lenta.
 - leer A – leer Pab – operación lógica para obtener B
 - Siempre el disco de paridad falla primero por eso no se usa raid 4.
 - La velocidad de lectura depende de la controladora, pero nunca va a ser menor que la velocidad de un disco. Casi siempre es la velocidad de un disco.
- RAID 5
 - Es el que más se usa.
 - Como la paridad se distribuye en los diferentes discos, la probabilidad de falla de los discos es la misma.
 - Se utiliza para datos.
 - Velocidad de Lectura puede ser en paralelo dependiendo de la controladora(El doble de un disco)
 - Bases de datos pequeñas donde la redundancia es importante, pero no se requiere un rendimiento extremadamente alto.
- RAID 6
 - Si solo me quedan las paridades, operó la paridad Pab con Qab y obtengo un dato, y si invierto como las opero obtengo el otro dato..
 - Pab y Qab no son iguales, no son la misma operación. Velocidad de Lectura puede ser en paralelo dependiendo de la controladora
 - Velocidad de Lectura puede ser en paralelo dependiendo de la controladora(El doble de un disco)
 - Es útil en entornos de almacenamiento críticos, como sistemas de almacenamiento empresarial, grandes bases de datos, sistemas de video vigilancia.
- RAID 0+1
 - No se usa porque solo puede fallar un disco y es lo mismo que raid 10
- RAID 10
 - Se usa mucho en virtualización
- RAID 50
 - Aumenta la velocidad de escritura y lectura del RAID 5. Funcionan los raid 5 en paralelo.
 - Mientras mas RAID 5 mejor velocidad y disponibilidad.
 - Se encuentra en entornos de bases de datos grandes donde se necesita una combinación de rendimiento y redundancia.

- RAID 60
 - Entornos como sistemas de almacenamiento empresarial críticos y servidores que requieren una alta disponibilidad.

Preguntar: Raid 1 → Velocidad de Lectura

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/1T5LvnxbibiBKJkva7TQ2ZyNFhOKUlGMH/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 máquina 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 máquina 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](#)
2. [Virtualización total](#)
3. [Paravirtualización](#)
4. [Contenedores](#)
5. [Asistencia vía hardware](#)

1. 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.).

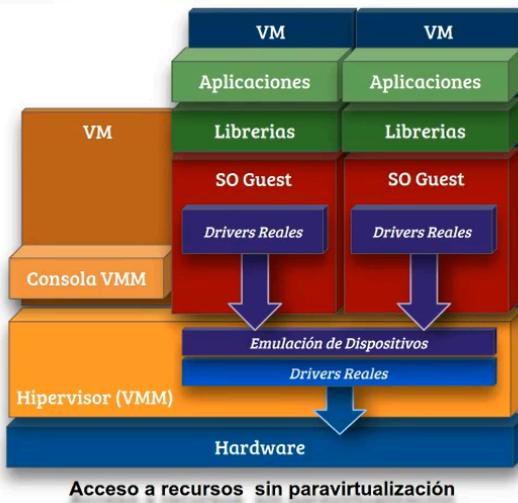
2. Virtualización total clásica (Hypervisor)

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)

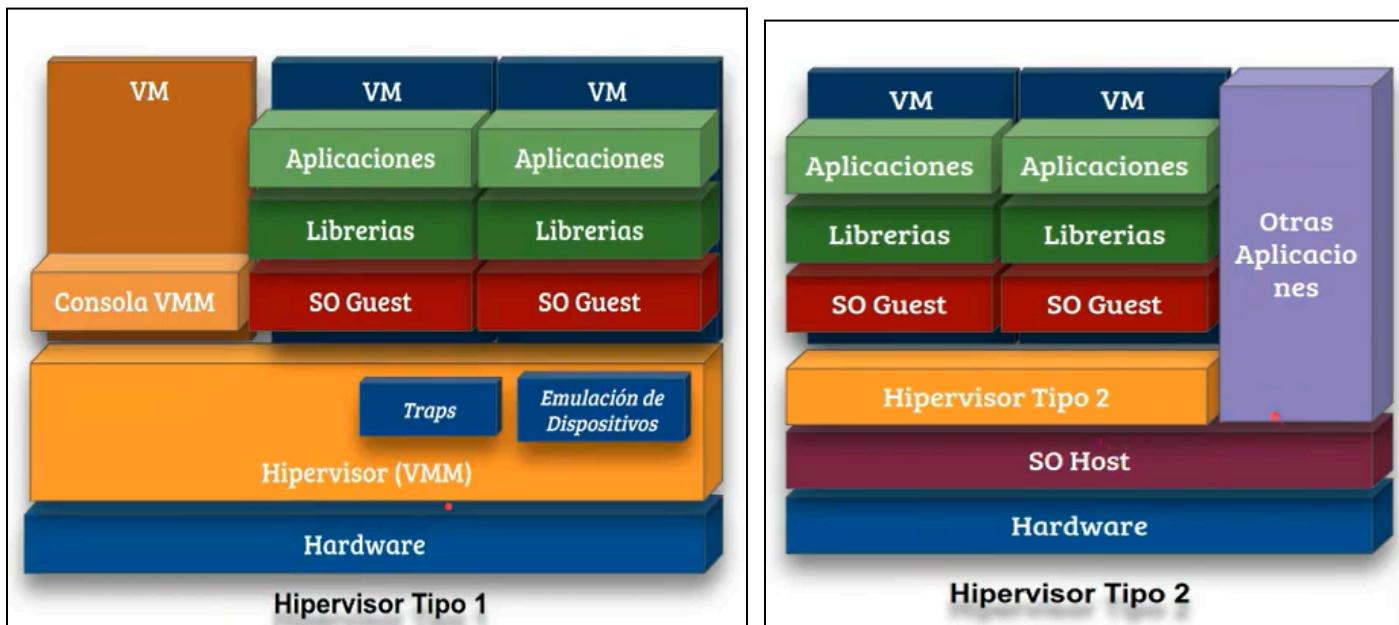
- **Ejecución Inst. privilegiadas por el Hypervisor.**
- **Adm. Ejecución de MV:** Aislamiento entre MV, cambio de contexto de los procesadores, prioridades de MV.
- **Gestión de ciclo de vida de la MV:** Inicio, pausa, apagarla.
- **Interfaz para que el usuario administre el Hypervisor.**
- **Emular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

Tipo de Hypervisor

Tipo 1: El Hypervisor es como el SO host de uso específico, 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**.



3. Paravirtualización

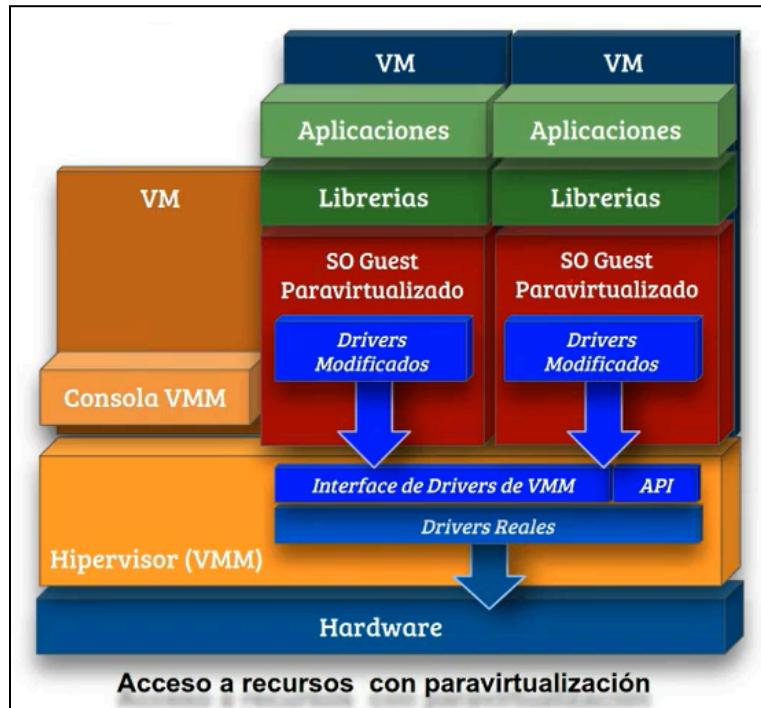
Más eficiente que virtualización total.

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.



4. Contenedores

- Poner los ejemplos: LXC, [Docker](#),
- 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 container inicializa directamente en el kernel del SO host y en base a esto, puede correr las diferentes aplicaciones con sus librerías correspondientes.
- Cada container empieza con un “init” y luego todas las ramas del árbol.
- Se instancian a partir de una Imagen.
- Para mantener aislados cada container, 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 contenedores (3)

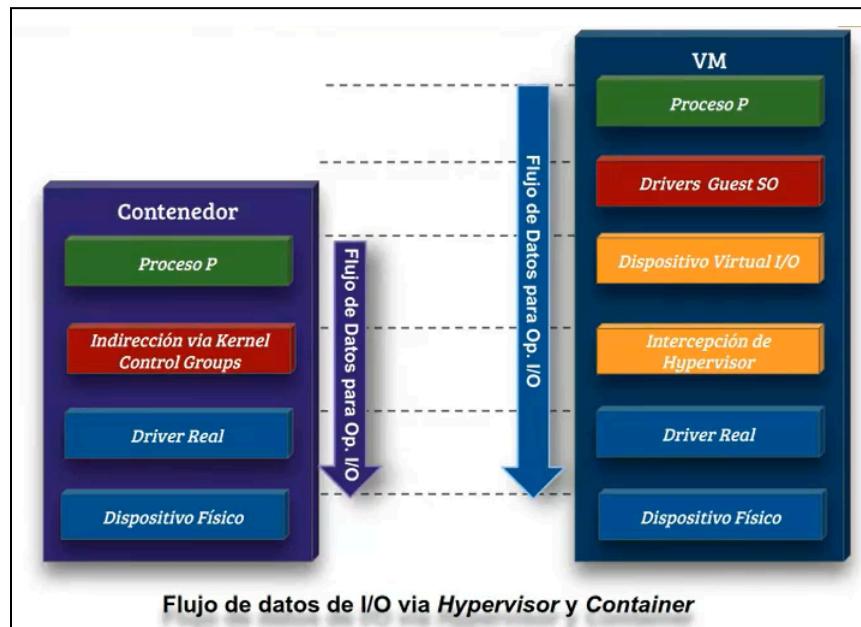
Es como el Hypervisor.

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 pongámole), entonces multipliqué por 4 a ese con todo los otros. Los otros 4 caramelos van a estar al pedo por decir.

Microservicios

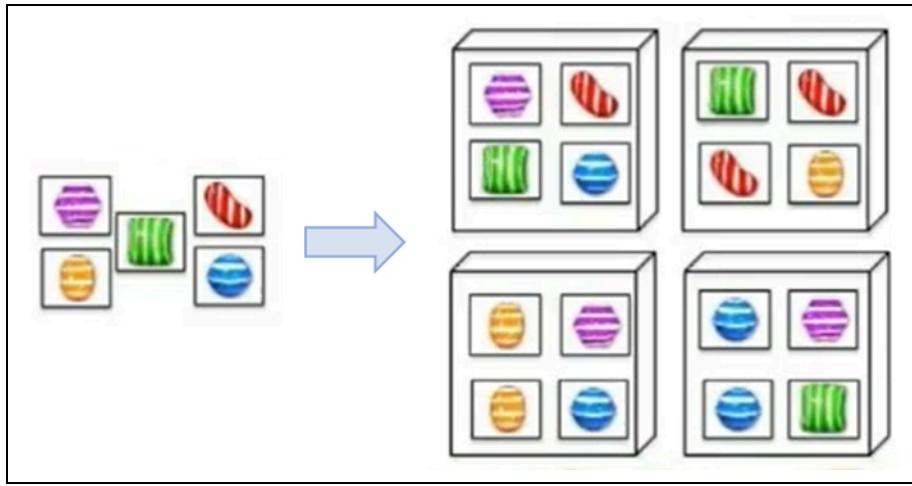
Los caramelos son funciones, los cuadrados son contenedores y las cajas son servidores.

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.

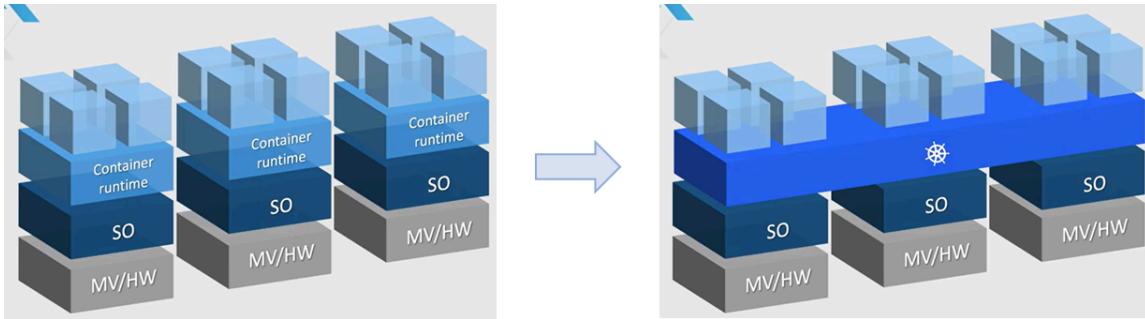
Dockerfile contiene la receta de lo que tiene que hacer el Docker Engine para armar la imagen, que cuando se corre aparece el microservicio.

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](#)>, 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](#)>.

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](#)

5. 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).

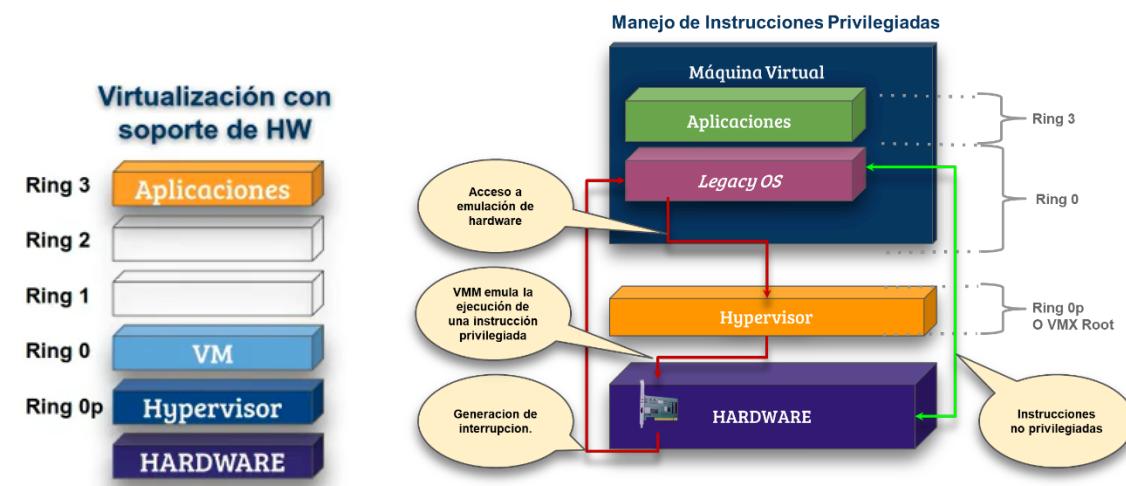
Uso normal (sin virtualizar): La arquitectura x86 implementa 4 anillos (0-3) de prioridades donde el 0 es el que puede con instrucciones privilegiadas y el 3 el no (como un anillo de usuario). Tener en cuenta que esto es hardware.



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 0p 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.



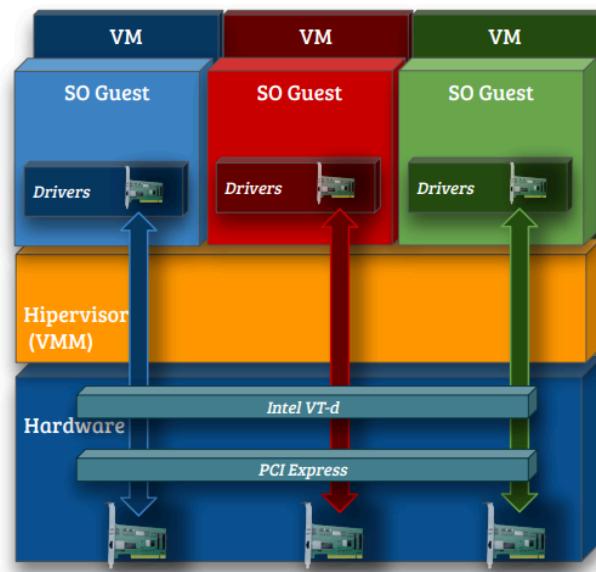
VT-d o AMD-Vi

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.

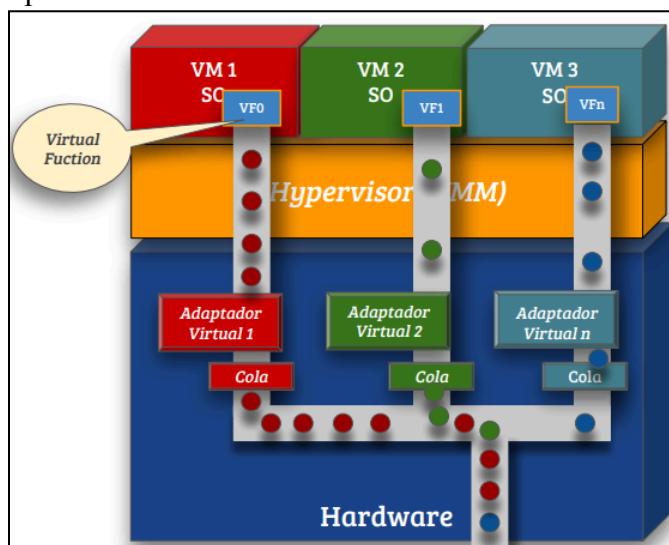
El VT-d realiza el remapeo de direcciones de los dispositivos hacia la MV que están asignados, evitando que interactúen con otras MV, por lo tanto, no pasa por el Hypervisor, es mucho más veloz, se puede utilizar hardware que no se puede emular.



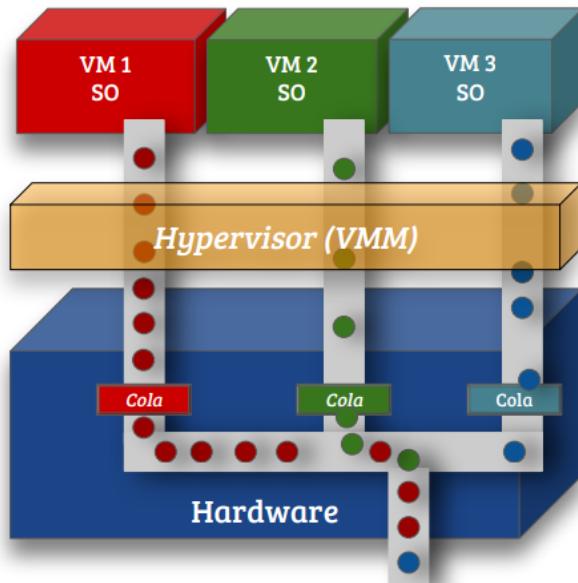
VT-c o ??????

Por un lado, **tenemos la técnica SR-IOV**, suele ser para placas de servidores. Permite asignar Funciones Virtuales (VF) 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).

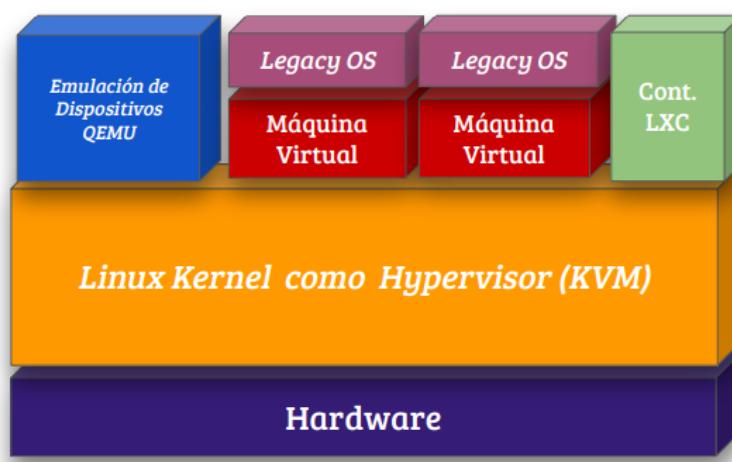
Cada usuario se conecta a una MV en un clúster, por lo tanto, la computadora del usuario solo funciona como un teléfono fijo (no tiene datos y procesa nada más que la imagen de la MV). Esto hace que las máquinas de escritorio no se pongan tan viejas ya que el ciclo de vida sería el del clúster (que es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

Hypervisor PromoxVE

El Hypervisor es el kernel de Linux.

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

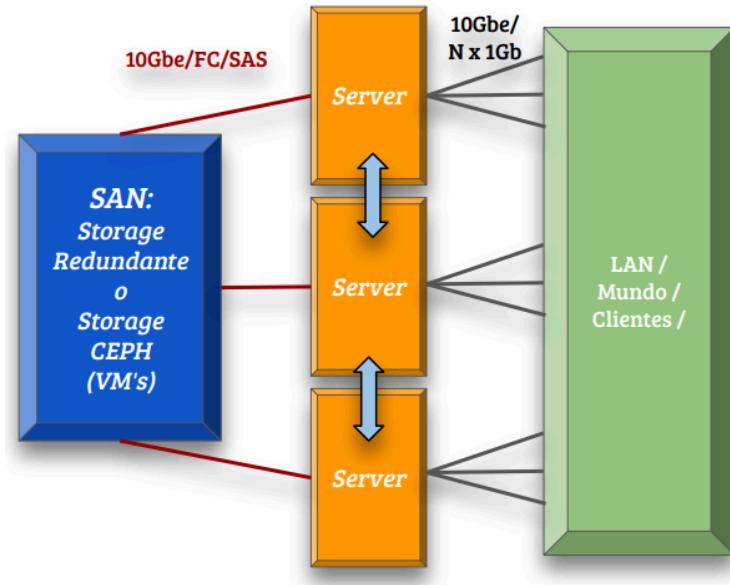
Virtualiza SO si/no modificados y contenedores de Linux (LXC).



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 en 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.