

UNIDAD 1

Tema: SISTEMA DE CÓMPUTOS

¿Cuáles son los 3 elementos básicos de un sistema operativo?

Son 3 elementos principales que están interconectados para ejecutar programas:

- **Procesador**

Controla las operaciones de la computadora y lleva a cabo funciones de programación de datos, Cuando hay un sólo procesador, se suele denominar **unidad central de procesamiento (CPU)**.

- **Memoria principal**

Almacena los datos y los programas. Esta memoria es normalmente volátil, también se le conoce como **memoria real** o **memoria primaria**.

- **Interconexión de sistemas**

Ciertos mecanismos y estructuras que permiten la comunicación entre procesadores, memoria principal y módulos de E/S.

¿De qué registros hace uso el procesador para intercambiar datos con la memoria? ¿Qué es un módulo de memoria?

El procesador intercambia datos con la memoria, para este proceso se lleva a cabo a través de dos **registros**.

- **MAR (Memory Address Register):** El cuál especifica la dirección en memoria de la próxima lectura o escritura.
- **MBR (Memory Buffer Register):** Que contiene datos que van a ser escritos a memoria o que fueron leídos de la misma.

De manera similar existen registros de direcciones de E/S.

- **IOAR (Input-Output Address Register):** Especifica un dispositivo particular de E/S.
- **IOBR (Input-Output Buffer Register):** Se utiliza para intercambiar datos entre un módulo de E/S y el procesador.
- **Módulo de memoria**

Es un conjunto de ubicaciones definidas por direcciones enumeradas secuencialmente. Cada una contiene un número binario que puede ser interpretado como dirección o dato.

¿Qué tipos registros se encuentran dentro del procesador? Registro visible de usuario – registro de control y estado.

- **Registros visibles de usuario**

Un registro visible es aquel que puede ser referenciado por medio del lenguaje de máquina que ejecuta el procesador y que por lo general **es accesible para todos los programas**.

Dentro de esta categoría se encuentran los:

- **Registros de datos:** Registros de propósito general
- **Registros de dirección:** Registros que contienen direcciones en la memoria, se encuentra el:
 - **Registro índice:** Registro que indica una dirección en memoria
 - **Registro de segmento:** Registro que hace referencia a un segmento en memoria
 - **Puntero de pila:** Registro destinado a señalar la cima de la pila
- **Códigos de condición:** Registros que también son denominados **indicadores o flags**. Son bits activados por el hardware del procesador como resultado de determinadas operaciones.
- **Registros de control y estado**

Son varios registros que se emplean para controlar las operaciones del procesador, estos no son visibles para el usuario en su mayoría.

Además del **MAR, MBR, IOAR y IOBR** los siguientes registros son esenciales en la ejecución de instrucciones:

- **Contador de programa (PC):** Contiene la dirección de la instrucción a ser leída
- **Registro de instrucción:** Contiene la última instrucción leída. Todos los procesadores incluyen además un registro o conjunto de registros conocidos como **palabra de estado del programa (PSW)** que contiene información del estado. Entre las más comunes se incluyen: **Signo, Cero, Acarreo, Igualdad, Desbordamiento, Habilitar/Inhabilitar interrupciones, Modo supervisor**

¿En qué consiste el ciclo de lectura y ejecución de instrucciones?

1. El procesador extrae la instrucción de la memoria
2. El contador de programa conserva la dirección de la próxima instrucción a ser extraída.
3. El contador de programa se incrementa después de cada extracción
3. La instrucción extraída se coloca en el registro de instrucción

Estos pasos se llaman **ciclo de lectura (fetch)** y **ciclo de ejecución**. La ejecución del programa sólo se detiene si se apaga la máquina u ocurre algún error.

El procesamiento requerido para una instrucción simple se llama **ciclo de instrucción**.

¿Cuáles son las funciones de E/S?

Son módulos que pueden intercambiar datos directamente con el procesador e iniciar una lectura o escritura en memoria.

¿Qué son las interrupciones? ¿De qué tipo existen?

Las interrupciones son **vías para mejorar la eficiencia del procesamiento**. Básicamente se interrumpen (pausan) las tareas para darle lugar a otras, existen 4 tipos:

- **De programa:** Generadas como instrucción por algún programa
- **De reloj:** Sirven para regular las funciones del sistema operativo
- **De E/S:** Generadas por algún periférico
- **Por fallo de hardware:** Cuando se te corta la luz

¿Qué es un controlador de interrupciones? ¿Y un ciclo de interrupciones?

El controlador de interrupciones es un programa que por lo general es parte del sistema operativo, determina la naturaleza de la interrupción y ejecuta la acción correspondiente.

Un **ciclo de interrupciones** es un ciclo que se añade al ciclo de lectura y escritura. Consiste en 3 pasos:

1. El procesador pregunta por interrupciones
2. Si no hay interrupciones extrae la próxima instrucción del programa actual
3. Si hay una interrupción pendiente, suspende la ejecución del programa actual y ejecuta el manejo de interrupciones

También puede haber **interrupciones múltiples**, que consisten en:

1. Inhabilitar interrupciones para que el procesador pueda completar la tarea
2. Las interrupciones siguen pendientes hasta que el procesador las habilita
3. Luego de completarse la rutina de manejo de interrupciones, el procesador chequea por interrupciones adicionales

En estas interrupciones las instrucciones de **mayor prioridad** interrumpen a las de **menor prioridad**

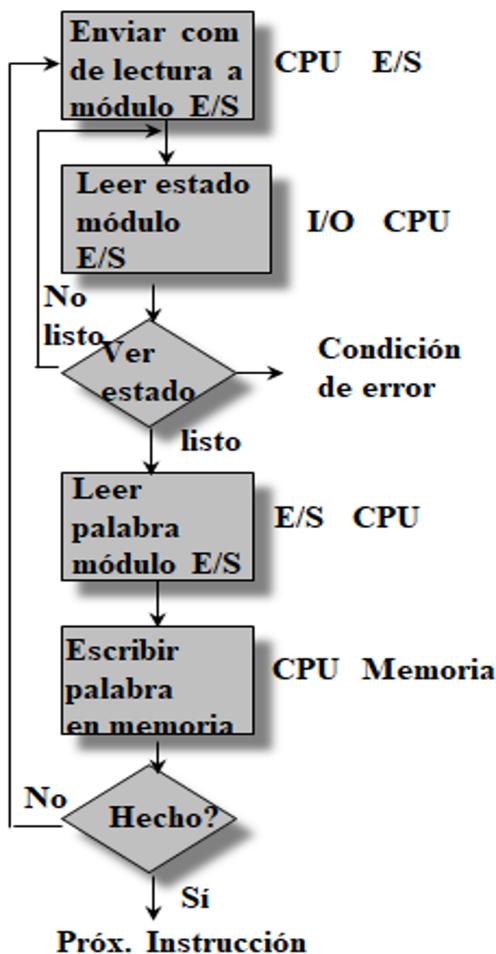
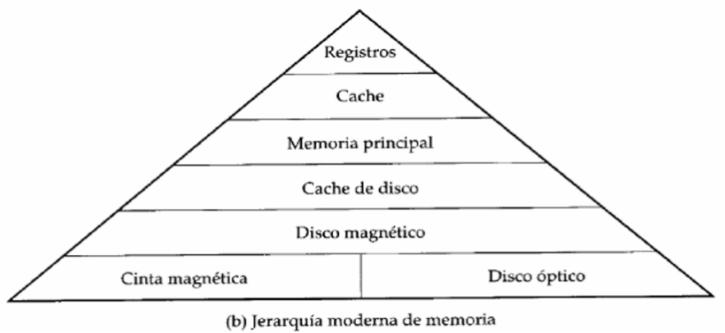
¿Qué jerarquías de memoria existen?

Hay que tener en cuenta 3 elementos: Coste, capacidad y tiempo de acceso, en base a estas se cumplen 3 relaciones:

- A menor tiempo de acceso, mayor coste por bit
- A mayor capacidad, menor coste por bit
- A mayor capacidad, mayor tiempo de acceso

En base a esto se plantea una jerarquía de memoria

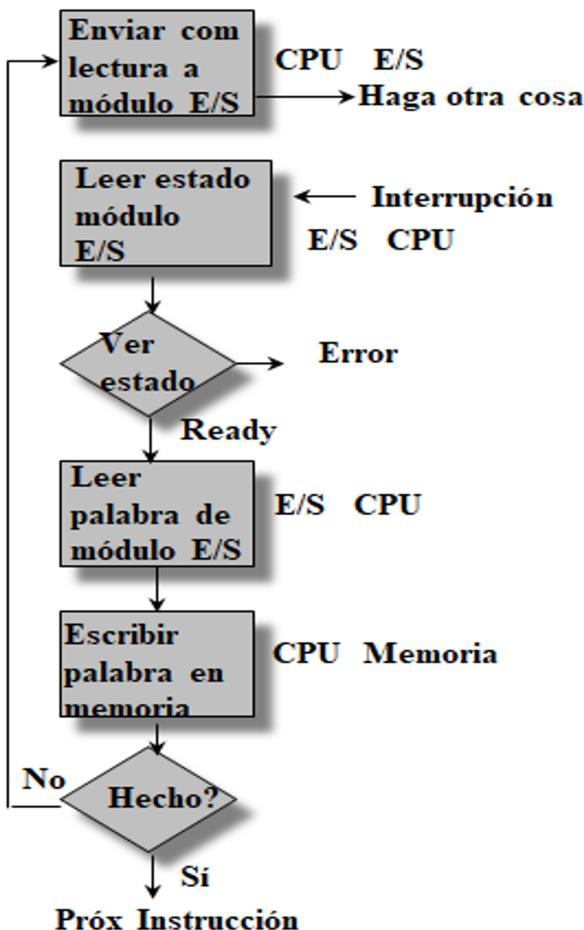
1. Disminución del coste por bit.
2. Aumento de la capacidad.
3. Aumento del tiempo de acceso.
4. Disminución de la frecuencia de acceso a la memoria por parte del procesador



¿Cuáles son las técnicas de comunicación de E/S?

E/S programada

- Lo realiza el módulo E/S, no el procesador
- Coloca los bits apropiados en el registro de estado de E/S
- No ocurren interrupciones
- El procesador está ocupado chequeando el estado

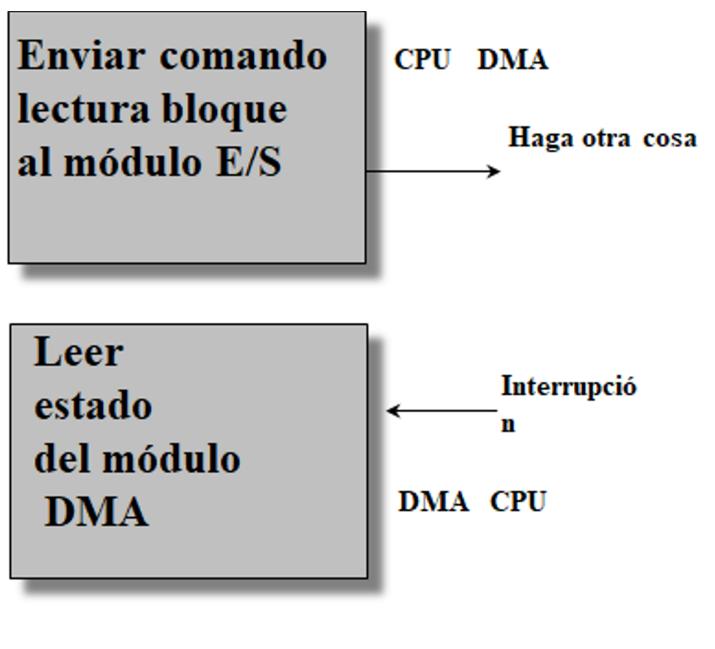


E/S manejada por interrupciones

- Se interrumpe al procesador cuando el módulo de E/S está listo para intercambiar datos
- El procesador está libre para hacer otro trabajo
- No hay espera innecesaria
- Consumo mucho tiempo de procesador porque cada palabra leída o escrita pasa a través del procesado

Acceso directo a memoria (DMA)

- Transfiere un bloque de datos directamente desde o hacia la memoria
- 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



TEMA 2: Sistemas Operativos

¿Qué es un sistema operativo?

Un **sistema operativo** es un programa que:

- Controla la ejecución de los programas de aplicación.
- Actúa como interfaz entre el usuario y el hardware.
- Enmascara los detalles del hardware.

Sus objetivos son:

- **Conveniencia:**
 - ✓ Hacer que una computadora sea más conveniente para usar
- **Eficiencia:**
 - ✓ Permitir un uso eficiente de los recursos
- **Capacidad de adaptación:**
 - ✓ Permitir desarrollo, pruebas e introducción de nuevas funciones sin interferir con el servicio

¿Cuáles son las funciones del sistema operativo? Servicios provistos por el **Sistema Operativo**:

- **Creación de programas: Editores y depuradores**
- **Ejecución de programas**
- **Acceso a dispositivos de E/S**
- **Acceso controlado a archivos**
- **Acceso al sistema**
- **Detección de errores y respuesta:**
 - ✓ **Errores de hardware internos y externos:**
 - Error de memoria
 - Fallo de dispositivo
 - ✓ **Errores de software:**
 - Overflow aritmético
 - Acceso prohibido a lugares de memoria
 - El sistema operativo no puede conceder el pedido de la aplicación
- **Contabilidad**
 - ✓ **Recolectar estadísticas**
 - ✓ **Monitorear rendimiento**
 - ✓ **Usada para anticipar mejoras futuras**
 - ✓ **Usada para facturar a los usuarios**

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

¿En qué se basa la evolución de un sistema operativo? Parte 1

La capacidad de evolución de un **Sistema Operativo** se basa en:

- Actualización y renovación de tipos de hardware
- Nuevos servicios
- Ajustes

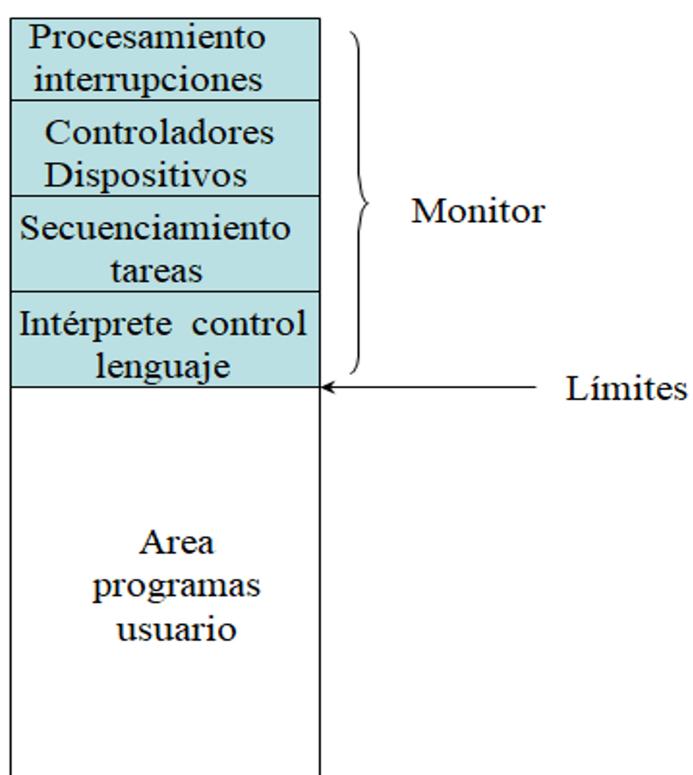
La línea evolutiva de los sistemas operativos empieza con el procesamiento en serie, luego siguen evolucionando hasta llegar a lo que conocemos hoy en día.

- **Procesamiento en serie:**

- No había sistema operativo
- Programas en código de máquina
- Problemas
 - ✓ Reserva previa de tiempo de uso
 - ✓ La instalación incluía cargar el compilador, el programa fuente, guardar el programa compilado, y luego cargar y enlazar

- **Sistema simple de lotes:**

- **Monitores**
 - ✓ Software que controla los programas en ejecución
 - ✓ Sistema operativo por lotes
 - ✓ Los trabajos se agrupan en lotes
 - ✓ El monitor está residente en la memoria principal y disponible para ejecutarse
 - ✓ Las utilidades del monitor se cargan cuando se necesitan



- **Lenguaje de control de trabajos (JCL)**
 - ✓ **Tipo especial de lenguaje de programación**
 - ✓ **Provee instrucciones para el monitor:**
 - Qué compilador usar
 - Qué datos usar
- **Otras características de hardware deseables**
 - ✓ **Protección de memoria**
 - No permitir alteraciones del área de memoria donde está el monitor
 - ✓ **Timer**
 - Evitar que una tarea monopolice el sistema
 - Al expirar el tiempo hay una interrupción
 - ✓ **Instrucciones privilegiadas**
 - Ejecutadas solamente por el monitor
 - Hay interrupción cuando un programa prueba estas instrucciones
 - ✓ **Interrupciones**
 - Provee flexibilidad para controlar los programas de usuario

¿En qué se basa la evolución de un sistema operativo? Parte 2

Sistema de lotes - multiprogramación

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



Ejemplo:

	TAREA1	TAREA2	TAREA3
Tipo de tarea	Mucho cálculo	Mucha E/S	Mucha E/S
Duración	5 min.	15 min.	10 min.
Mem requerida	50K	100 K	80 K
Necesita disco?	No	No	Sí
" terminal	No	Sí	No
" printer?	No	No	Sí

Efecto:

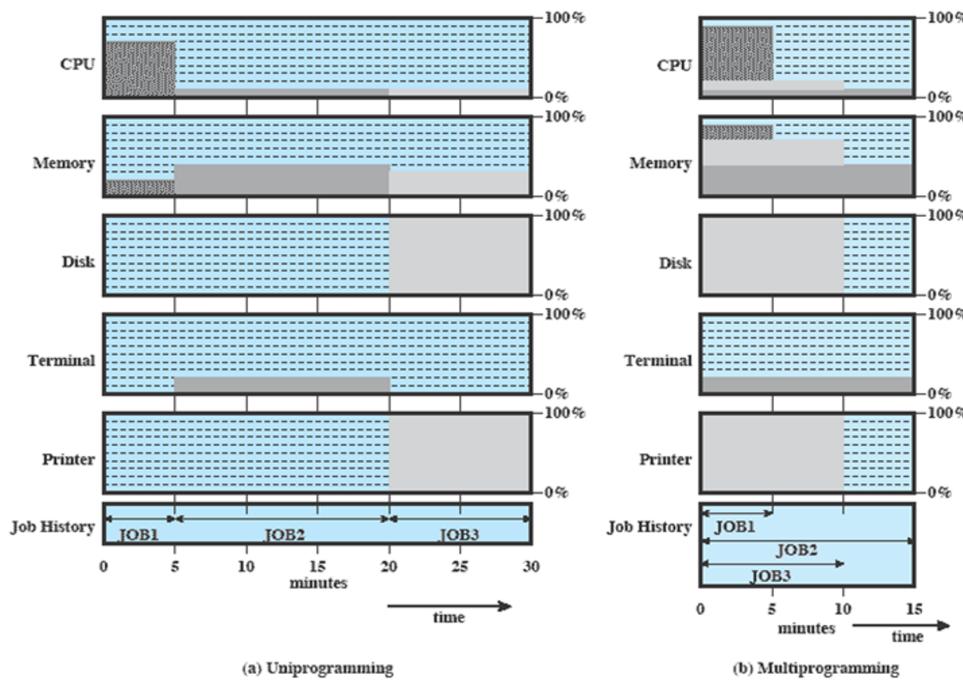


Figure 2.6 Utilization Histograms

Sistema de tiempo compartido

- 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

Multiprogramación por lotes vs Tiempo compartido

	Multiprogramación por lotes	Tiempo compartido
Principal objetivo	Maximiza el uso del procesador	Minimiza el uso del procesador
Fuente de directivas al sistema operativo	Comandos de lenguaje de control de trabajo proporcionados con el trabajo	Comandos ingresados en la terminal

¿Cuáles son los principales logros de los sistemas operativos? Parte 1

- **Procesos**
 - ⌚ Es un programa en ejecución
 - ⌚ La entidad que se puede asignar y ejecutar en un procesador
 - ⌚ Es el “espíritu animado” de un programa
 - ⌚ Lo que se manifiesta por la existencia de un “bloque de control de proceso” en el sistema operativo

- **Gestión de memoria**

- **Manejo de memoria**

- Aislamiento de procesos
- Asignación y manejo automáticos
- Apoyo para programación modular
- Protección y control de acceso
- Almacenamiento por largos periodos de tiempo

- **Memoria virtual**

- Permite a los programas direccionar la memoria desde un punto de vista lógico sin tener en cuenta la cantidad de memoria 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 a la memoria virtual que puede ser tanto
 - Un espacio de direcciones lineal
 - Un conjunto de segmentos (bloques de tamaño variable)
- El hardware debe mapear (traducir) la dirección virtual en dirección real

- **Sistema de archivos**

- Implementa almacenamiento permanente
- Se almacena información en objetos con nombre denominados archivos

¿Cuáles son los principales logros de los sistemas operativos? Parte 2

- **Seguridad y protección de la información**

- Disponibilidad
- Confidencialidad
- Integridad de los datos
- Autenticidad

- **Planificación y manejo de recursos**

- **Equidad**

- Dar acceso justo y equiparado a todos los procesos Sensibilidad diferencial
- Discriminar entre clases de tareas distintas

- **Eficiencia**

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

- **Estructura del sistema**

- Vista del sistema como una serie de niveles
- Cada nivel ejecuta un subconjunto relacionado de funciones
- Cada nivel cuenta con el nivel próximo más bajo para realizar funciones más primitivas
- Esto descompone un problema dentro de un número más manejable de sub-problemas (Sub-tareas)

Jerarquía de diseño de un SO (Gráfico)

TABLA 2.4 Jerarquía de Diseño de un Sistema Operativo

Nivel	Nombre	Objetos	Ejemplos de Operaciones
13	Shell	Entorno de programación del usuario	Sentencias de un lenguaje de shell
12	Procesos de usuario	Procesos de usuario	Salir, eliminar, suspender, reanudar
11	Directorios	Directorios	Crear, destruir, conectar, desconectar, buscar, listar
10	Dispositivos	Dispositivos externos tales como impresoras, pantallas y teclados	Crear, destruir, abrir, cerrar, leer, escribir
9	Sistema de archivos	Archivos	Crear, destruir, abrir, cerrar, leer, escribir
8	Comunicaciones	Tubos (<i>pipes</i>)	Crear, destruir, abrir, cerrar, leer, escribir
7	Memoria Virtual	Segmentos, páginas	Leer, escribir, traer (<i>fetch</i>)
6	Almacenamiento secundario local	Bloques de datos, canales de dispositivos	Leer, escribir, asignar, liberar
5	Procesos Primitivos	Procesos primitivos, semáforos, colas de procesos listos	Suspender, reanudar, esperar, señalizar
4	Interrupciones	Programas de tratamiento de interrupciones	Invocar, enmascarar, desenmascarar, reintentar
3	Procedimientos	Procedimientos, pila de llamadas, visualización	Marcar la pila, llamar, retornar
2	Conjunto de instrucciones	Evaluación de la pila, intérprete de microprogramas, vectores de datos y escalares	Cargar, almacenar, sumar, restar, bifurcar
1	Circuitos electrónicos	Registros, puertas, buses, etc	Borrar, transferir, activar, complementar

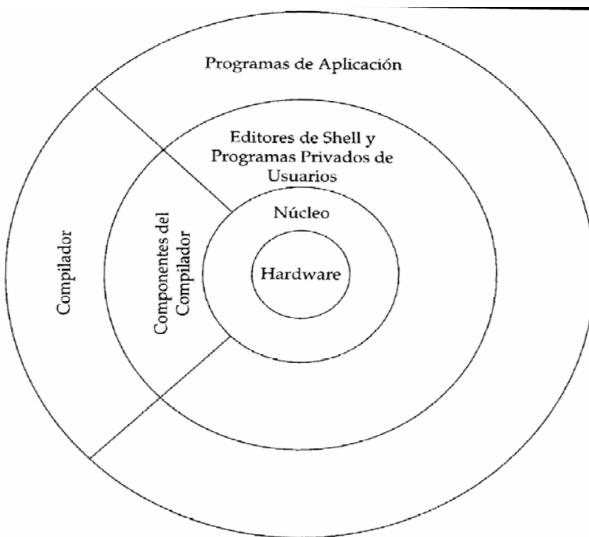
Características de Windows NT

- Sistema operativo **monousuario** y **multitarea**
- **Monousuario:** Orientado a dar soporte a un sólo usuario
- **Multitarea:** Gracias a algunas características de los principales logros de los sistemas operativos aumentó la velocidad con respecto a sus antecesores. Y una capacidad para ejecutar múltiples tareas al mismo tiempo, como una hoja de cálculo, una hoja de texto y un programa de dibujo (Me imagino usando uno de estos SO me corto la piña eso recién era nuevo seguro no corrían ni el GTA 3)
- Creció el proceso **cliente/servidor** que se basa en un host en un área local y otras computadoras conectadas a este en LAN (Como si hoy abrieras un puerto en minecraft lan y todos se conecta a tu pc para jugar, sólo que mucho más primitivo.)

- Otras características a las que no les voy a dar mucha pelota:
 - Independencia de sistema operativo de 4 niveles:
 - Capa de abstracción de hardware
 - Núcleo
 - Subsistemas
 - Servicios del sistema
 - Cada subsistema protegido proporciona una interfaz para los programas (API)
 - Hilos
 - Multiproceso simétrico
 - Objetos de Windows NT:
 - Encapsulamiento
 - Clases e instancias

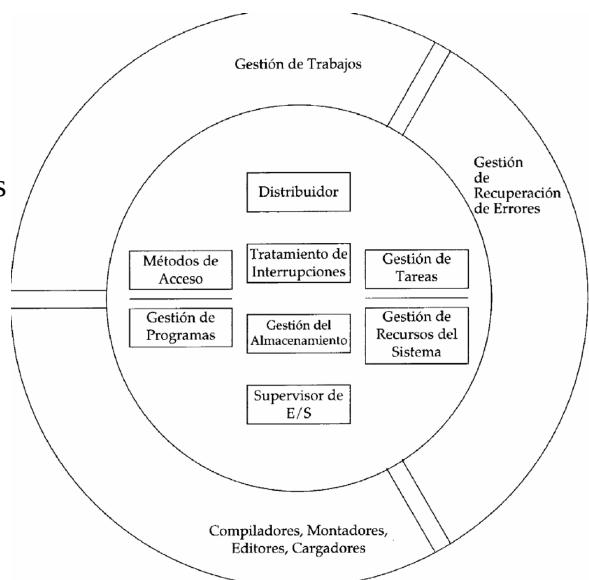
Características de UNIX

- Este sistema operativo introduce el concepto de **kernel (O núcleo del sistema)**
- Trae equipado servicios de usuario e interfaces, se pueden agrupar en un **shell (Terminal)**, otros softwares de interfaz y componentes de compilador de C (**El lenguaje de programación**)
- Parece ser un predecesor simple de linux o algo por el estilo



Características de MVS

- Requiere un mínimo de 2MB de almacenamiento residente (Era el sistema más pesado en la fecha).
- Requería 6MB como recomendado
- Trajo características nuevas como:
 - Soporte para trabajos interactivos y por lotes
 - Almacenamiento virtual de hasta 32GB por trabajo o usuario.
 - Multiproceso fuertemente acoplado
 - Asignación sofisticada de recursos y servicios de supervisión para lograr un uso eficiente de la gran memoria del sistema, múltiples procesadores y estructura compleja de canales de E/S
- También posee un **shell externo** que posee características nuevas que más o menos puede hacer linux hoy en día para resumir
- Es altamente probable que a este sistema operativo no lo conozca ni el creador (Totalmente fantasma)



UNIDAD 2

TEMA: Procesos

¿Qué es un proceso?

Informalmente un proceso es un programa en ejecución. Es la unidad de ejecución más pequeña planificable.

Un proceso está formado por 3 componentes:

1. **Un programa ejecutable.**
2. **Datos asociados necesarios para el programa (variables, espacio de trabajo, buffers, etc.).**
3. **Contexto de ejecución o estado del proceso (contenido en el PCB).**

El contexto de ejecución o estado del proceso es el componente más esencial, ya que incluye toda la información que el SO necesita para administrar el proceso y que el procesador necesita para ejecutarlo correctamente.

¿Cuál es la diferencia entre proceso y programa?

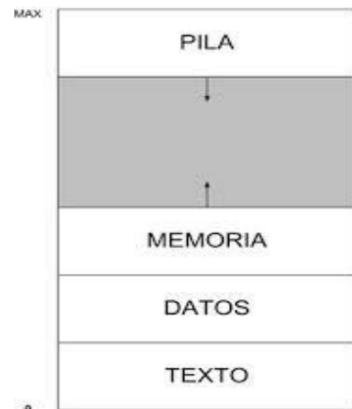
- **Proceso** como entidad activa y dinámica (asignación dinámica de memoria).
- **Programa** entidad pasiva y estática.

Un **programa** se convierte en un **proceso** cuando se carga en memoria su archivo ejecutable.

La relación entre proceso y programa no es de uno a uno. Es decir dentro de un proceso pueden ejecutarse tanto un programa de usuario como uno del sistema operativo.

Proceso en memoria

- **Pila del proceso** → Datos temporales (Direcciones de retorno, parámetros de funciones, variables locales)
- **Memoria (HEAP)** → Cúmulo de memoria. Es la memoria que se le asigna dinámicamente al proceso en tiempo de ejecución.
- **Sección de datos** → Variables Globales.
- **Sección de Texto** → Código del programa.



PCB : Bloque de Control De Proceso

Cada proceso se representa en el SO mediante un bloque de control de proceso. Este contiene un conjunto de atributos utilizados por el SO para el control/administración del proceso.

La información contenida en el **PCB** se puede agrupar en 3 categorías:

- **1. Identificación del proceso**

Identificador del proceso. Identificador del proceso Padre. Identificador del usuario.

• 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.

Nota: El contador del programa (PC) contiene la dirección de la siguiente instrucción que va a ejecutar el proceso.

• 3. Información de control del proceso

Información de planificación de CPU (prioridad del proceso, etc.). **Información de gestión de memoria**. **Información contable** (uso de CPU, etc.). **Información del estado de E/S** (lista de archivos abiertos, dispositivos de E/S asignados al proceso, etc.).

El PCB es la estructura más importante del SO. El conjunto de PCBs define el estado del SO. El PCB es parte de la imagen del proceso (Datos de usuarios, programa de usuario, pila del sistema y PCB). La ubicación de la imagen del proceso depende del esquema de gestión de memoria utilizado. En particular, en sistemas que utilizan memoria virtual, toda la imagen de un proceso activo está siempre en memoria secundaria. Cuando una imagen se carga en memoria principal, esta se copia en vez de moverse.

¿En qué consiste el control de procesos?

Se utilizan 2 modos de ejecución:

- **Modo Usuario** → No se permite la ejecución de instrucciones privilegiadas. El intento de ejecución de una instrucción privilegiada en este modo produce una excepción.
- **Modo Kernel** → Permite ejecución de instrucciones privilegiadas (llamadas al sistema, tratamiento de interrupciones, asignación de espacio de memoria a los procesos, etc.)

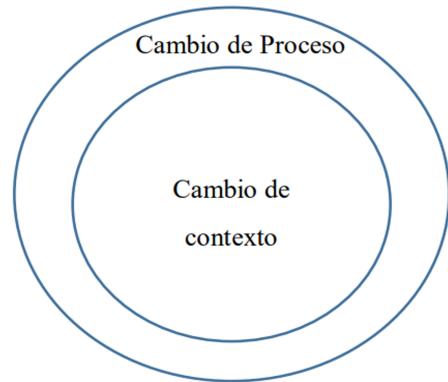
La razón por la que se usan estos 2 modos, se debe a que es necesario proteger al SO y a las estructuras del mismo, como los PCB, de las injerencias de los programas de usuarios.

Nota: ¿Cómo sabe el procesador en qué modo va a ejecutar? Hay un bit en el PSW (Palabra de estado del programa) que indica el modo de ejecución.

Nota: ¿Cómo se cambia de modo? El bit cambia como respuesta a ciertos sucesos. El control se encuentra inicialmente en manos del SO (modo Kernel). Cuando se da el control a una aplicación de usuario se pasa a modo usuario. El control finalmente se tiene que devolver al SO a través de una excepción o interrupción.

CAMBIO DE PROCESOS – PARTE 1

- **Cambio de proceso** → Cambio del uso de la CPU de un proceso de otro.
- **Cambio de contexto** → Cuando un proceso pasa de memoria a CPU o de CPU a memoria.
- **Cambio de modo** → Cuando se cambia de modo Kernel a modo usuario o viceversa.



Cada vez que se produce una interrupción o llamada al sistema, se debe **guardar** el estado del proceso que se está ejecutando para luego poder **reanudar** la *ejecución*. Se trata la interrupción o llamada al sistema y luego se carga nuevamente un *PCB*. Este puede ser el *PCB* del proceso que se estaba ejecutando o el *PCB* de otro proceso.

Por lo tanto, un cambio de proceso, implica inevitablemente un cambio de contexto. En cambio, un cambio de contexto no implica siempre un cambio de proceso, ya que se puede volver a cargar el proceso que se estaba ejecutando.

Un cambio de contexto **NO** involucra siempre un cambio de modo.

Un cambio de modo **NO** involucra siempre un cambio de contexto. Se puede cambiar el modo sin cambiar el estado del proceso (página 134 del Stallings).

Esto se debe a que se puede ejecutar casi todo el software del SO en el contexto de un programa de usuario (el SO se ejecuta dentro del proceso de usuario). Esto sirve para recordar que hay una distinción entre el concepto de programa y proceso y que la relación entre los 2 no es de uno a uno.

Es decir dentro de un proceso pueden ejecutarse tanto un programa de usuario como uno del sistema operativo y los programas del SO que se ejecutan en los diferentes procesos de usuarios son idénticos.

Explicación de este enfoque:

- Cuando se produce una interrupción, se salva el contexto de procesador y tiene lugar un cambio de modo hacia una rutina del SO. Sin embargo, la ejecución continúa dentro del proceso de usuario en curso. De esta manera no se ha llevado a cabo un cambio de proceso (y por ende de contexto), sino un cambio de modo dentro del mismo proceso. Debido al cambio de modo usuario a modo **Kernel**, el usuario no interrumpe las rutinas del SO, aun cuando estas estén ejecutándose en el entorno de proceso de usuario. Cuando el SO termina su trabajo, determina que el proceso en curso debe continuar ejecutándose, entonces un cambio de modo continua el programa interrumpido del proceso en curso (aunque puede determinar cambiar el proceso).
- **Ventaja de este enfoque:** Un programa de usuario se interrumpe para emplear alguna rutina del SO, luego se reanuda y todo se produce SIN LA PENALIZACIÓN DE 2 CAMBIOS DE PROCESO (sacar el proceso en ejecución, cargar la rutina del SO, sacar la rutina del SO, reanudar/ejecutar un proceso de usuario).

- **Observación:** Otro enfoque trata al SO como entidad separada que opera en modo privilegiado. El concepto de proceso se aplica solo a los programas de usuario.

CAMBIO DE PROCESO – PARTE 2

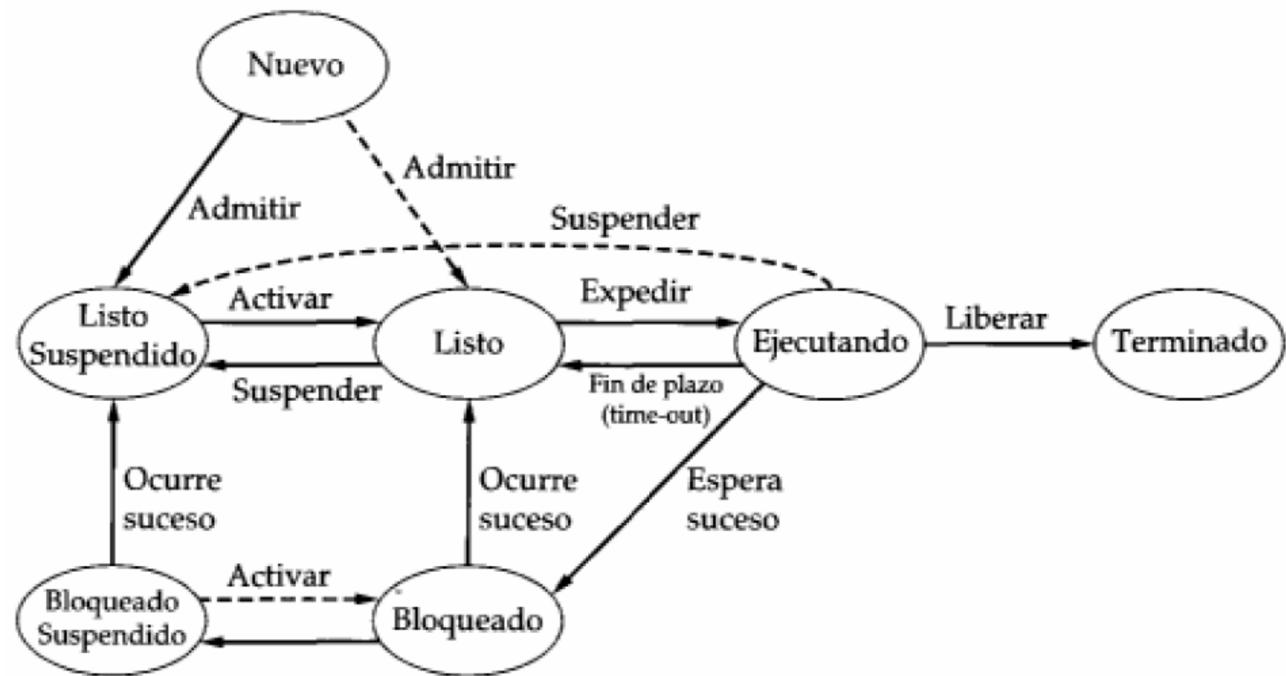
Nota: ¿Cuándo se puede producir un cambio de proceso? En cualquier momento en el que el SO haya tomado el control a partir del proceso que está actualmente ejecutándose. Los sucesos que pueden darle el control al SO son:

- **Interrupción Hardware externa:** originada por algún tipo de suceso que es externo e independiente del proceso que está ejecutándose. Interrupción de E/S, Interrupción de reloj.
- **Interrupción Hardware interna (Excepciones):** causada por una condición de error generada dentro del proceso que está ejecutándose (división por 0, el desbordamiento, el acceso a una posición de memoria no permitida, acceso ilegal a un archivo, etc.) o por condiciones anómalas (Fallo de página, etc.)
- **Interrupción Software (Llamadas al sistema):** Generadas por el programa en ejecución. Llamadas al sistema para que el SO ejecute instrucciones privilegiadas en nombre del programa.

Nota: Sobre el cambio de modo. Cada vez que se produce una interrupción: Se salva el contexto del programa que está ejecutándose. Se asigna al contador del programa la dirección del programa de tratamiento de la interrupción. Finalmente cambia de modo de usuario a modo **Kernel** para poder ejecutar las instrucciones privilegiadas del programa de la interrupción.

Nota: ¿Qué constituye salvar el contexto del programa? Se debe salvar cualquier información que pueda alterarse por la ejecución de la rutina de tratamiento de la interrupción y todo lo necesario para reanudar el programa interrumpido (Debe salvarse la parte del **PCB** que se denomina Información de estado del procesador).

Modelo de los 7 estados



1. **Nuevo** → el proceso se acaba de crear, pero aún no es admitido por el SO como proceso ejecutable (aún no está cargado en la MP. Se mantiene en MS).
2. **Listo** → el proceso está preparado para poder ejecutarse (en MP y listo para ejecutarse).
3. **Ejecutando** → el proceso está actualmente en ejecución
4. **Bloqueado** → el proceso no se puede ejecutar hasta que se produzca cierto suceso (en MP esperando un suceso)
5. **Terminado** → por alguna razón el SO saca al proceso del grupo de procesos ejecutables.
6. **Bloqueado y suspendido** → el proceso está en MS esperando un suceso.
7. **Listo y suspendido** → el proceso está en MS pero está disponible para su ejecución tan pronto que se cargue en MP

Nota: MP (Memoria Principal), MS (Memoria Secundaria).

La figura indica los tipos de sucesos que producen cada transición de estados para un proceso.

Algunas aclaraciones:

- **Nulo** → **Nuevo**: Las razones para la creación de un proceso son, Trabajo por lotes, Conexión interactiva (el usuario crea un proceso desde un terminal), el SO crea un proceso para dar un servicio y un proceso crea otro proceso.
- **Ejecución** → **Varios**: Por espera de un suceso, por fin de Quantum, Yield (proceso que cede voluntariamente el procesador y fin de proceso).
 - Importante: Un proceso puede pasar de Ejecución → Listo, únicamente, si abandona involuntariamente la CPU (Preemptive /Apropiativo /Con desalojo), ya sea por una Interrupción de Reloj o porque se pasó a preparado un otro proceso con mayor prioridad.
 - Si la planificación es Non-Preemptive/ No Apropiativo / sin desalojo) el proceso se ejecuta hasta que termina o hasta que tenga que hacer una E/S (pasa a bloqueados), por lo que NO es posible que un proceso pase del estado “en ejecución” a listo.
- **Varios** → **Terminado**: Un proceso puede terminar en cualquier momento. Por eso están las transiciones de Listo a Terminado, Bloqueado a Terminado, etc.
- **Bloqueados**: Es conveniente tener una cola de bloqueados por cada suceso, ya que si hay 1 sola cola de bloqueados, el SO debería recorrer toda la lista en busca de aquellos procesos que esperaban que ocurra el suceso. En cambio, si se tiene una cola por suceso, directamente la lista entera del suceso puede pasar al estado de listo.

Procesos suspendidos (necesidad de Intercambio)

1. Por más que haya **multiprogramación** (varios procesos en MP), el procesador podría estar desocupado la mayor parte del tiempo. Esto se debe a que el procesador es mucho más rápido que cualquier operación de E/S, lo cual podría generarse una situación en que todos los procesos estén en estado bloqueado y el procesador no tenga procesos para ejecutar.

- Una solución a este problema, es el **intercambio (swap)**, que significa mover una parte del proceso o todo el proceso de la MP a MS. De esta manera se baja el grado de multiprogramación, permitiendo el ingreso de un nuevo proceso listo para ejecutar (depende del diseño se puede traer un Nuevo proceso o uno Suspenido y listo). Por lo general se suspende aquel proceso bloqueado que hace tiempo está esperando un suceso.

2. Otros motivos de suspensión: cambios en los requisitos de memoria, para mejorar la estabilidad del sistema, el proceso puede ser causante de un problema, por solicitud del proceso padre, por solicitud de un usuario, un proceso temporal que se ejecuta cada cierto tiempo, etc.

Nota: El intercambio, en sí, es una operación de E/S a disco por lo que existe la posibilidad de que el problema empeore en vez de mejorar (aunque la E/S de disco es la más rápida del sistema y por lo general suele mejorar el rendimiento).

Importante: Incluso en un sistema de Memoria Virtual, el SO siempre tendrá que expulsar de vez en cuando algunos procesos, de forma explícita y completa (mediante intercambio), con el objetivo de mejorar el rendimiento, ya que el rendimiento de un sistema de memoria virtual puede desplomarse si hay un número suficientemente grande de procesos activos.

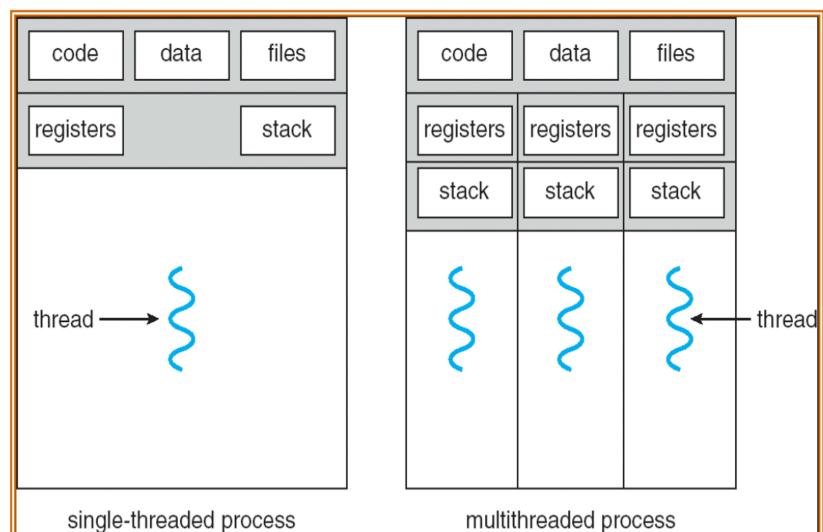
¿Qué es un hilo?

Un hilo es un proceso ligero, un proceso ahora está compuesto por 1 o más hilos; cada hilo puede realizar una tarea.

¿En qué se diferencia un proceso mono-hilo de un multi-hilo?

En un modelo multi-hilo:

- Los hilos de un mismo proceso comparten el código, datos y los recursos del proceso (comparten el mismo espacio de direcciones).
- Cada hilo tiene su propia pila y su bloque de control de hilo (además del bloque de control de proceso).



¿Qué ventajas tienen los hilos?

Sus beneficios se derivan de las implicaciones de rendimiento:

1. **Economía:** Se consume mucho menos tiempo en crear/finalizar y realizar cambios entre hilos debido a que comparten recursos del proceso al que pertenecen. (PRINCIPAL VENTAJA DE LOS HILOS).
2. **Compartir recursos:** Permite que una aplicación tenga varios hilos que realizan una actividad diferente dentro del mismo espacio de memoria.
3. **Capacidad de respuesta:** Una aplicación puede realizar varias tareas a la vez e inclusive seguir ejecutándose aunque parte de ella este bloqueada.
4. **Utilización de multiprocesador:** Las hebras pueden ejecutarse en paralelo en cada procesador incrementando el grado de concurrencia.
5. **Los hilos de un mismo proceso pueden comunicarse entre si, sin invocar al núcleo.**

Estados de un proceso:

- **Listo**
- **Ejecución**
- **Bloqueado**

Ventajas y desventajas de los ULT (User level threads/Hilos a nivel de usuario)

En una aplicación ULT: USER LEVEL THREADS

- La gestión de hilos la realiza la aplicación (biblioteca de hilos). El núcleo no es consciente de la existencia de hilos (el SO solamente ve un proceso con un hilo, no sabe de la existencia de hilos). Por defecto un proceso comienza su ejecución con un único hilo cuyo proceso es gestionado por el SO. El proceso puede crear un nuevo hilo que se ejecuta dentro del mismo proceso. **La creación se lleva a cabo por la biblioteca de hilos. La planificación de los hilos también está a cargo de la biblioteca de hilos.** Cuando el control pasa a la biblioteca, se salva el contexto del hilo actual, que se restaura cuando el control pasa de la biblioteca al hilo. Todas las operaciones descriptas, se llevan a cabo en el espacio de usuario dentro de un mismo proceso. El núcleo no tiene conocimientos de ellas. El núcleo continúa planificando el proceso como una única unidad y asignándole un único estado.
- Si un hilo realiza una llamada al sistema, se bloquea todo el proceso (la biblioteca el SO, el ULT sigue “ejecutando”, por lo tanto cuando el proceso vuelve a ejecutar, sigue ejecutando el mismo ULT). Lo mismo pasa si al proceso se le acaba el quantum de tiempo, para la biblioteca, el hilo que estaba corriendo, sigue en estado ejecutando (aunque verdaderamente no esté haciendo uso de la CPU). Cuando el SO le devuelve a CPU al proceso, continua en ejecución el hijo que estaba ejecutando antes de que se bloqueara el proceso.
- Si un hilo A solicita a otro hilo B del mismo proceso que realice alguna acción. El hilo A se pone en bloqueado y el hilo B pasa a Ejecutando. Importante darse cuenta que el paso de un proceso a otro, lo realiza la biblioteca de hilos y que el proceso sigue en estado ejecutando.

Ventajas de los ULT:

- El intercambio de hilos no necesita de los privilegios del modo Kernel, por estos comparten el espacio de direcciones de usuario de un mismo proceso. NO se necesitan 2 cambios de modo: de usuario a Kernel y de Kernel a usuario.
- Planificación específica: cada aplicación planifica sus hilos mediante la biblioteca de hilos según le convenga.
- Los ULT pueden ejecutarse en cualquier SO. No es necesario realizar cambios en el núcleo subyacente.

Desventajas de los ULT:

- La mayoría de las llamadas al sistema son bloqueantes, por lo cual cada vez que un ULT realiza una llamada al sistema, bloquea todo el proceso.
- No aprovecha el multiprocesamiento. El SO puede asignar un solo proceso a un solo procesador cada vez. Por lo tanto se puede ejecutar un hilo de cada proceso en cada instante.

Nota: se puede solucionar mediante la técnica de jacketing (recubrimiento) que convierte una llamada bloqueadora en una llamada no bloqueadora.

Ventajas y desventajas de los KLT (Kernel level threads/Hilos a nivel de núcleo)

En una aplicación KLT:

- El SO conoce la existencia de los hilos y se encarga de realizar la gestión de los mismos.

Ventajas de los KLT:

- Resuelve los problemas de los ULT:
- Si se bloquea uno de los hilos, no bloquea a todo el proceso. El núcleo puede planificar otro hilo del mismo proceso.
- Aprovecha el multiprocesamiento. El SO puede planificar diferentes hilos de un mismo proceso en diferentes procesadores.

Desventaja de los KLT:

- El cambio de un hilo a otro requiere de cambio de modo.

¿Qué tipos de relación existe entre los ULT y los KLT?

- **Muchos a uno:** Múltiples hilos de usuario a uno de Kernel. Desventaja que no hay concurrencia ya que un solo hilo ULT puede acceder al núcleo a la vez (no hay multiprocesamiento). Además una llamada bloqueante bloquea a todo el proceso.
- **Uno a Uno:** Soluciona los 2 problemas de Muchos a uno, aunque el problema es que por cada UTL hay que crear un KLT, por lo que esto repercutirá en el rendimiento. Se debe restringir el número de hilos soportados por el sistema.
- **Muchos a Muchos:** Soluciona el problema de uno a uno.

TEMA 2 PLANIFICACIÓN

¿Para qué se usa la planificación de la CPU?

Casi todos los recursos de la PC se planifican antes de usarlos.

La CPU, es uno de los principales recursos de la PC, así que su correcta planificación resulta crucial en el diseño del SO.

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).

Planificadores del procesador

- **PLP (Planificador de largo plazo):** Determina el grado de multiprogramación (cuantos procesos pueden estar en MP). A su vez debe lograr la estabilidad del sistema (un equilibrio entre procesos CPU BOUND e IO BOUND).
- **PMP (Planificador de Mediano plazo):** Controla el grado de multiprogramación. Encargado de realizar el intercambio (swap) para bajar el grado de multiprogramación.
- **PCP (Planificador de corto plazo):** Controla el grado de multiprocesamiento (cantidad de procesos que se puede estar ejecutando a la vez). Encargado de determinar qué proceso se va a ejecutar. También conocido como distribuidor (dispatcher).
- **PELP (Planificador extra largo plazo):** En base a un histórico de procesos ejecutados puede cambiar el algoritmo de planificación o el grado de multiprogramación.



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

Algoritmos de planificación a corto plazo

1. Llamada al sistema (Ejecución → Espera).
2. Interrupción de Reloj (Ejecución → Preparado).
3. Interrupción de E/S (Espera → Preparado).
4. Terminación del proceso (Ejecución → Terminado).

Nota: Cuando la planificación solo tiene lugar en las situaciones 1 y 4, se dice que el esquema de planificación es **Non-Preemptive / No Apropiativo / Sin Desalojo / Cooperativo** ya que el SO no tiene la capacidad de quitarle la CPU al proceso hasta que este termine o ceda voluntariamente la CPU con una llamada al sistema.

Si la planificación también tiene lugar en las situaciones 2 y 3, se dice que el esquema de planificación es **Preemptive / Apropiativo / Con Desalojo** ya que el SO tiene la capacidad de quitarle la CPU al proceso mediante un fin de quantum o debido a que pasa a preparado un proceso con más prioridad que el proceso que se está ejecutando.

Los diferentes algoritmos de planificación de la CPU tienen distintas propiedades y la elección de un algoritmo en particular puede favorecer a una clase de procesos sobre otros. Los criterios son los siguientes:

- Rendimiento / Tasa de procesamiento (\uparrow): Cantidad de procesos que se procesan por unidad de tiempo.
- Utilización de la CPU (throughput) (\uparrow): No desperdiciar el uso de la CPU.
- Tiempo de espera (waiting time) (\downarrow): Tiempo que un proceso tarda en ejecutarse (tiempo que está en la cola de listos).
- Tiempo de respuesta (response time) (\downarrow): Tiempo desde que el procesador se pone en la cola de listos hasta que genera una E/S.

Minimizar la varianza del tiempo de respuesta, significa que si se vuelve a ejecutar el mismo proceso, su tiempo de respuesta debería ser el mismo. Esto es una diferencia con Minimizar el promedio del tiempo de respuesta. En sistemas interactivos conviene minimizar la varianza del tiempo de respuesta (ya que el tiempo de respuesta es predecible. Siempre tarda lo mismo).

- Tiempo de Retorno (turnaround time) (\downarrow): Tiempo que tarda un proceso de pasar de listo → terminado (es el tiempo total que está en el sistema).

Nota: Maximizar (\uparrow) / Minimizar (\downarrow)

Los algoritmos de planificación a corto plazo son

1. FCFS
2. SJF
3. Por Prioridades
4. Por Turnos (RR, round robin)
5. Colas Multinivel
6. Colas Multinivel Realimentadas

¿Qué es la planificación FCFS? (Sin desalojo)

Primero en llegar, primero en ser atendido.

Ventajas:

- No se producen muertes por inanición.

Desventajas:

- Tiempo de espera medio largo y se modifica notablemente según el orden en que lleguen los procesos.
- Problemático para sistemas de tiempo compartido por no ser con desalojo.
- Rinde mejor con procesos CPU bound que con I/O bound. Puede provocar un uso ineficiente de la CPU y de los dispositivos de E/S.

Efecto convoy: muchos procesos I/O bound esperan mucho tiempo a que un proceso CPU bound se termine de ejecutar. Esto hace que la utilización de la CPU y los dispositivos sea menor que si se ejecutaría primero los I/O bound

¿Qué es la planificación SJF? (Con o sin desalojo)

Primero ejecuta el más corto (Ordena por la siguiente ráfaga de CPU más corta). Si 2 procesos tienen la misma ráfaga de CPU, se desempata por FCFS.

Ventajas:

- Tiempo de espera medio disminuye y se mantiene por más que cambie el orden de llegada de los procesos.

Desventajas:

- Se puede producir muerte por Inanición.
- Mucho OVERHEAD. No se puede saber cuál es la siguiente ráfaga de CPU. Solo se puede aproximar, lo que requiere de cálculos extras.

Se usa frecuentemente como mecanismo de planificación a largo plazo.

SJF en sí, es un algoritmo por prioridades → Cuanto más larga sea la ráfaga de CPU, menor prioridad.

Tiempo estimado de la próxima Ráfaga → $T_{n+1} = T_n \alpha + R_n (1 - \alpha)$ siendo $0 \leq \alpha \leq 1$

- T_n tiempo estimado de la ráfaga anterior
- R_n tiempo real de la ráfaga anterior
- Si α tiende a 0 se le da poca bola al estimado anterior. Si α tiende a 1 poca bola al real anterior.

Observación: SPN → sin desalojo, SRT → con desalojo. Observación: SRT a diferencia de RR no genera interrupciones adicionales. También debería devolver un mejor tiempo de retorno ya que atiende *inmediatamente* a los procesos de menor duración.

¿Qué es la planificación por Prioridades? (Con o sin desalojo)

Primero se atiende el de mayor prioridad. Se asume que 0 es la prioridad más alta.

Ventajas:

- Tiempo de espera medio disminuye y se mantiene por más que cambie el orden de llegada de los procesos.

Desventajas:

- Se puede producir muerte por Inanición

¿Qué es la planificación HRRN? (Sin desalojo)

Primero el de mayor tasa de respuesta. (Basado en prioridades) Tasa de Respuesta $\rightarrow R = W + S / S$ Siendo W tiempo de espera y S tiempo de ráfaga esperado. Cuanto mayor R, mayor prioridad. Tiene en cuenta la edad del proceso (por W, tiempo de espera). Por lo tanto elimina el problema de inanición.

¿Qué es la planificación RR (Round robin)? (Con desalojo)

Igual que FCFS pero con desalojo por fin de quantum. **Si Q es muy grande:** Termina siendo igual que FCFS **Si Q es muy chico:** Produce mucho OVERHEAD. **Importante:** Ante simultaneidad de evento se atiende primero:

1. Excepción
2. Interrupción Q
3. Interrupción E/S
4. Llamada al sistema

UNIDAD 3 – CONCURRENCIA

Tema: Exclusión mutua y sincronización.

¿Para qué sirve la concurrencia?

La concurrencia es fundamental en la:

- Multiprogramación
- Multiprocesamiento
- Procesamiento distribuido (gestión de varios procesos, ejecutándose en sistemas de computadores múltiples y distribuidos).

La concurrencia comprende un gran número de cuestiones de diseño del SO como es la compartición de recursos, comunicación entre procesos, asignación de recursos a los procesos, etc

Principios generales de la concurrencia

En un sistema multiprogramado con un único procesador, los procesos se intercalan en el tiempo para dar la apariencia de ejecución simultánea, a diferencia de un sistema multiprocesamiento en el que consigue procesamiento paralelo real. Aunque intercalación y superposición representen formas de ejecución diferente, ambas técnicas pueden contemplarse como ejemplos de procesamiento concurrente y ambas plantean los mismos problemas. **Las dificultades que presenta la concurrencia son:**

1. Compartir recursos globales está lleno de riesgos. Si dos procesos hacen uso al mismo tiempo del mismo recurso, el orden en que se ejecuten las lecturas y escrituras, es crítico.
2. Es difícil gestionar la asignación óptima de recursos.
3. Resulta difícil localizar un error de programación.

El problema de concurrencia se presenta en la multiprogramación debido a que una interrupción puede detener la ejecución de instrucciones de un proceso en cualquier momento y dar lugar a otro sin que este halla finalizado.

En el caso de un sistema multiprocesador, se tiene la misma condición y además el problema puede ser causado por 2 procesos que se estén ejecutando simultáneamente y que intenten ambos acceder a la misma variable global (Recordemos que la velocidad relativa de ejecución de los procesos es impredecible.)

Sin embargo, la solución al problema de la concurrencia para ambos tipos de sistemas es la misma: CONTROLAR EL ACCESO AL RECURSO COMPARTIDO.

Moraleja: La exigencia básica para soportar la concurrencia de procesos es la posibilidad de **hacer cumplir la exclusión mutua**, es decir la capacidad de prohibir a los demás procesos realizar una acción cuando un proceso haya obtenido el permiso (un solo proceso puede hacer uso de la región critica a la vez)

Nota: Un sistema operativo no Apropiativo está esencialmente libre de condiciones de carrera en lo que respecta a las estructuras de datos del Kernel (ejemplo: lista de todos los archivos abiertos) ya que hay solo 1 proceso activo en el Kernel en cada momento.

¿Cómo es la interacción entre procesos?

Los procesos que se ejecutan **concurrentemente** pueden ser procesos independientes o procesos cooperativos:

- **Procesos independientes** → Aquel que NO puede afectar o verse afectado por los restantes procesos que se encuentran en el sistema. Cualquier proceso que NO comparte datos con ningún otro proceso es un proceso independiente.
- **Procesos cooperativos** → Aquel que puede afectar o verse afectado por los restantes procesos que se encuentran en el sistema. Cualquier proceso que comparte datos con otros procesos es un proceso cooperativo. Los procesos cooperativos pueden compartir un espacio de memoria directamente (**cooperación por compartimiento**) o compartir datos solo a través de archivos o mensajes (**cooperación por comunicación**). El primer caso se consigue mediante el uso de hilos.

En el caso de los procesos independientes, aunque no comparten datos, el sistema operativo tiene que encargarse de la competencia por los recursos (CPU, impresora, memoria, ETC). En este caso, los resultados de un proceso son independientes de las acciones de los otros, pero la ejecución de un proceso puede influir en el comportamiento de los procesos que compiten (los tiempos de los procesos pueden verse afectados)

Cuando hay procesos en competencia por un mismo recurso, se deberán solucionar 3 problemas de control:

1. **Exclusión Mutua** (hacer cumplir la exclusión mutua crea 2 problemas de control adicionales, 2 y 3)
2. **Interbloqueo**
3. **Inanición**

Nota: Hacer cumplir la exclusión mutua hace que aparezcan los problemas 2 y 3. Uno es el interbloqueo (un proceso P1 está esperando que un proceso P2 libere un recurso y el P2 está esperado a que P1 libere un recurso. Ni P1 ni P2 liberan el recurso hasta que el otro libere el recurso que necesita) y la inanición (un proceso espera indefinidamente a que se le asigne el acceso al recurso).

Nota: El control de la competencia siempre involucra al SO porque es quien asigna los recursos
Sección crítica → Segmento de código que hace uso de un recurso crítico

Recurso critico → Cualquier recurso compartido (variables, archivos, impresora, lista, etc.) El caso de **cooperación por compartimiento** los procesos pueden tener acceso a variables compartidas, archivos o base de datos compartidos. No saben que procesos acceden a los datos, pero son conscientes de que estos otros pueden tener acceso a los mismos datos.

Como los datos se guardan en recursos (dispositivos, memoria, etc.) también presentan los problemas de exclusión mutua, interbloqueo e inanición por competir por recursos críticos. Además se debe introducir un nuevo requisito: **Integridad/Cohesión** de los datos.

En el caso de **cooperación entre procesos por comunicación**, la comunicación es una manera de sincronizar o coordinar las distintas actividades. Por lo que no es necesario el control de la mutua exclusión para este tipo de cooperación. Sin embargo, los problemas de inanición e interbloqueo siguen presentes.

¿Qué es la condición de carrera?

Cuando varios procesos manipulan y acceden **concurrentemente** a los mismos datos y el resultado de la ejecución depende del orden en que se produzcan los accesos.

Para protegerse frente a este tipo de condiciones, se necesita garantizar que solo un proceso cada vez puede acceder a la sección crítica.

Protocolo de la región crítica

1. **Exclusión mutua:** Cuando un proceso usa la región crítica, ningún otro proceso, que tenga región crítica por el mismo recurso u objeto compartido, puede entrar en ella.
2. **Progreso:** Cuando un proceso está fuera de la región crítica, no pueden impedir a otros procesos que la usen.
3. **Espera limitada:** Cualquier proceso que quiere entrar a la región critica lo debe poder hacer luego de un número finito de intentos.
4. **Velocidad de los procesos:** Todos los procesos pueden usar la región crítica las veces que quieran y el tiempo que quieran. **Nota:**
 - **Espera activa** → Cuando un proceso que no obtuvo permiso para entrar en su sección critica, se queda comprobando periódicamente la variable hasta que pueda entrar. Consuma tiempo de procesador (está activo) mientras espera su oportunidad.
 - **Espera no activa** → Se bloquea al proceso y se lo desbloquea cuando puede entrar

¿Qué soluciones existen para satisfacer los requisitos del protocolo de región crítica?

1. Solución por software.
2. Solución por hardware.
3. Soporte del SO / lenguaje de programación.

Soluciones por software (Algoritmo de Dekker 1º y 2º intento)

- **Algoritmo de Dekker (1º intento)**

Cualquier intento de exclusión mutua debe depender de algunos mecanismos básicos de exclusión en el hardware. El más habitual es que sólo se puede acceder a una posición de memoria en cada instante, teniendo en cuenta esto se reserva una posición de memoria global llamada turno. Un proceso que desea ejecutar su sección crítica primero evalúa el contenido de turno. Si el valor de turno es igual al número del proceso, el proceso puede continuar con su sección crítica. En otro caso el proceso debe esperar. El proceso en espera, lee repetitivamente el valor de turno hasta que puede entrar en su sección crítica. Este procedimiento se llama espera activa. Después de que un proceso accede a su sección crítica y termina con ella, debe actualizar el valor de turno para el otro proceso.

Proceso 0	Proceso 1
...	...
/*esperar*/	/*esperar*/
while (turno!=0);	while (turno!=1);
/*sección crítica*/	/*sección crítica*/
...	...
turno=1;	turno=0;
...	...

Esta solución garantiza el problema de la exclusión mutua pero tiene 2 inconvenientes:

- Deben alternarse de forma estricta en el uso de sus secciones críticas, por lo que el ritmo de ejecución depende del proceso más lento.
- No cumple la condición de progreso → Si un proceso falla fuera o dentro de la sección crítica, el otro proceso se bloquea permanentemente.

Algoritmo de Dekker (2º intento)

Cada proceso debe tener su propia llave de la sección crítica para que, si uno de ellos falla, pueda seguir accediendo a su sección crítica; para esto se define un vector booleano señal. Cada proceso puede evaluar el valor de señal del otro, pero no modificarlo. Cuando un proceso desea entrar en su sección crítica, comprueba la variable señal del otro hasta que tiene el valor falso (indica que el otro proceso no está en su sección crítica).

Asigna a su propia señal el valor cierto y entra en su sección crítica. Cuando deja su sección crítica asigna falso a su señal

Proceso 0	Proceso 1
...	...
/*esperar*/	/*esperar*/
While (señal[1]);	While (señal[0]);
señal[0] = cierto;	señal[1]=cierto;
/*sección crítica*/	/*sección crítica*/
...	...
señal[0] = falso;	señal[1] = falso;

Se solucionan los problemas anteriores, sin embargo, ahora surgen otros nuevos:

- Si uno de los procesos falla dentro de su sección crítica el otro quedará bloqueado permanentemente.
- No se garantiza la Exclusión Mutua como vemos en la secuencia:
 1. P0 ejecuta el While y encuentra señal [1] a falso.
 2. P1 ejecuta el While y encuentra señal [0] a falso.
 3. P0 pone señal [0] a cierto y entra en su sección crítica.
 4. P1 pone señal [1] a cierto y entra en su sección crítica. Ambos procesos están en su sección crítica, esto se debe a que esta solución no es independiente de la velocidad de ejecución relativa de los procesos.

Algoritmo de Dekker (3º intento)

Una vez que un proceso ha puesto su señal en cierto, el otro no puede entrar a su sección crítica hasta que el primero salga de ella. Se garantiza la EM, sin embargo, se generará interbloqueo si ambos procesos ponen su señal a cierto antes de ambos hayan ejecutado el While. Además, si un proceso falla en su sección crítica, el otro queda bloqueado permanentemente.

Proceso 0	Proceso 1
...	...
señal[0] = cierto;	señal[1]=cierto;
/*esperar*/	/*esperar*/
<u>While</u> (señal[1]);	<u>While</u> (señal[0]);
/*sección crítica*/	sección crítica*/
...
señal[0] = falso;	señal[1] = falso;

Algoritmo de Dekker (4º intento)

En el tercer intento, un proceso fijaba su estado sin conocer el estado del otro. Se puede arreglar esto haciendo que los procesos activen su señal para indicar que desean entrar en la sección crítica pero deben estar listos para desactivar la variable señal y ceder la preferencia al otro proceso.

Existe una situación llamada bloqueo vital, esto no es un interbloqueo, porque cualquier cambio en la velocidad relativa de los procesos rompería este ciclo y permitiría a uno entrar en la sección crítica.

Recordando que el interbloqueo se produce cuando un conjunto de procesos desean entrar en sus secciones críticas, pero ninguno lo consigue. Con el bloqueo vital hay posibles secuencias de ejecución con éxito.

Algoritmo de Dekker (Solución correcta) resuelve el problema de la exclusión mutua.

Combinación entre intento 1 y 4: while (true)

{

interesado[0] = TRUE;

while (interesado[1]) {

if (turno ≠ 0) {

interesado[0] = FALSE; while (turno ≠ 0); interesado[0] = TRUE; }

}

SECCIÓN CRÍTICA

turno = 1; interesado[0] = FALSE;

SECCIÓN RESTANTE}

Algoritmo de Peterson

Resuelve también el problema de la exclusión mutua, pero a diferencia del algoritmo de Dekker, mediante una solución simple y elegante.

Soluciones por hardware

- **Inhabilitación de interrupciones** En un sistema Apropiativo, un proceso continuará ejecutándose hasta que solicite una llamada al sistema o hasta que sea interrumpido. Por lo tanto, para garantizar la exclusión mutua, es suficiente con impedir que un proceso sea interrumpido.

Problema 1: Valido solamente para sistemas monoprocesador. Para sistemas multiprocesador, es posible que haya más de un proceso ejecutándose al mismo tiempo.

Problema 2: Pierde eficiencia el sistema ya que se limita la capacidad del procesador para intercalar procesos.

- **Instrucciones de maquina**

- **Test and set (TLS):** Instrucción Hardware que lee y modifica atómicamente una variable (por ser atómica esta instrucción no puede ser interrumpida). Además, si nos encontramos en un sistema multiprocesador, ninguno de los demás procesadores tendrá acceso a la variable hasta que termine de ejecutarse la instrucción.
- **Swap (intercambio):** De forma atómica sobre 2 palabras.

Ventajas de la solución con instrucciones de máquina:

- Valido para sistema monoprocesador y multiprocesador.

Desventajas:

- Emplea espera activa.
- Puede producir inanición.
- Puede producir interbloqueo.

En las soluciones software, los procesos deben coordinarse unos con otros para cumplir la exclusión mutua, sin ayuda por parte del lenguaje de programación o del sistema operativo.

Las soluciones por software son propensas a errores y a una fuerte carga de proceso. Las soluciones por hardware reducen la sobrecarga, pero no son interesantes como solución de carácter general.

Debido a estos inconvenientes, tanto de las soluciones por software como por hardware, es necesario buscar otros mecanismos.

Soluciones con soporte del lenguaje de programación o del SO

Semáforos Son variables especiales que tienen un valor entero sobre el que se definen 3 operaciones:

1. Un semáforo puede iniciarse con un valor **no** negativo
2. La operación wait disminuye el valor del semáforo. Si el valor se hace negativo, el proceso que ejecuta el wait se bloquea.
3. La operación signal incrementa el valor del semáforo. Si el valor no es positivo, se desbloquea un proceso bloqueado por una operación wait.

Son las únicas 3 operaciones posibles para manipular semáforos

WAIT(S)	SIGNAL(S)
Semáforo--	Semáforo++
If (semáforo<0)	If (semáforo<=0)
{	{
Poner este proceso en la cola del semáforo;	Quitar un proceso de la cola del semáforo;
Bloquear este proceso	Poner el proceso en la cola de listos;
}	}

En cualquier instante el valor del semáforo puede interpretarse de la siguiente forma:

- **Semáforo** ≥ 0 : Es el número de procesos que puede ejecutar un wait(s) sin bloquearse.
- **Semáforo** < 0 : Cantidad de procesos bloqueados en la cola del semáforo

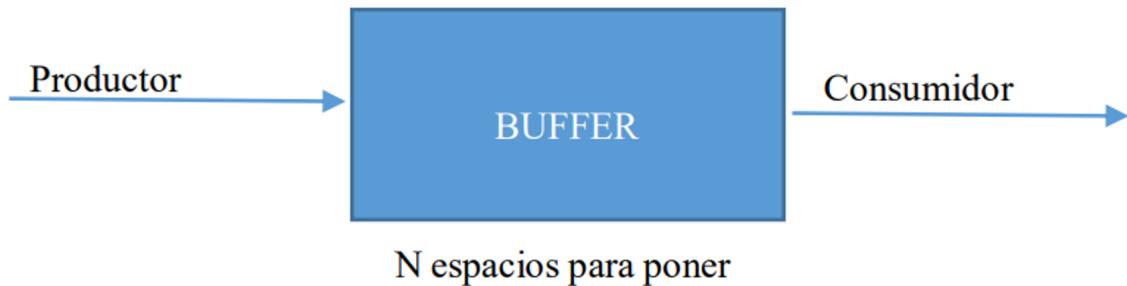
Las primitivas wait(s) y signal(s) son atómicas. Son relativamente cortas, por lo que la cantidad de espera activa que se obtiene será menor. Los semáforos emplean una cola para mantener los procesos que están esperando en el semáforo. Si el semáforo incluye la estrategia de cómo se retiran los procesos de la cola, se denomina **semáforo robusto** (Ejemplo: se retira por FIFO). En caso contrario, **semáforo débil**. Los semáforos robustos garantizan la inexistencia de inanición, pero no así los semáforos débiles

Nota:

Semáforos binarios

WAITB(S)	SIGNALB(S)
If (semáforo = 1)	If (Cola.semáforo.esvacia)
Semáforo = 0;	Semáforo = 1;
Else	Else
{	{
Poner este proceso en la cola del semáforo;	Quitar un proceso de la cola del semáforo;
Bloquear este proceso;	Poner el proceso en la cola de listos;
}	}

Problema de productor y consumidor



Uno o más productores generan cierto tipo de datos y los sitúan en un buffer. Un único consumidor saca elementos del buffer de uno en uno. El sistema debe impedir la superposición de operaciones sobre el buffer. Es decir un solo agente consumidor o productor puede acceder al buffer en un instante dado (se logra con un semáforo binario mutex)

- Si el buffer está lleno, el productor debe ser demorado (en el caso que el buffer sea limitado)
→ Semáforo Lugar
- Si el buffer está vacío, el consumidor debe ser demorado → Semáforo Lleno

Buffer ilimitado (color rojo) y buffer limitado (se agrega el semáforo lugar en verde)

Lleno = 0

Mutex = 1

Lugar = N

Productor () {

While (1){

 Msg = producir ()

 Wait (Mutex)

 Wait (Lugar)

 Depositar (msg)

 Signal (Lleno)

 Signal (Mutex)

}

Consumidor () {

While (1) {

 Wait (Lleno)

 Wait (Mutex)

 Msg = retirar ()

 Signal (Mutex)

 Signal (Lugar)

 Consumir ()

}

¿Qué son los monitores?

Los semáforos son flexibles y potentes pero puede resultar **difícil** construir un programa correcto por medio de semáforos, ya que las operaciones wait y signal deben distribuirse por todo el programa y no es fácil advertir el efecto global de estas operaciones sobre los semáforos a los que afectan.

Los monitores ofrecen una funcionalidad equivalente a la de los semáforos y que son más fáciles de controlar. Un **monitor** es un **módulo de software que consta de uno o más procedimientos, una secuencia de inicio y variables locales**. Solo un proceso puede estar ejecutando el monitor a la vez, por lo que ofrece un servicio de exclusión mutua fácilmente.

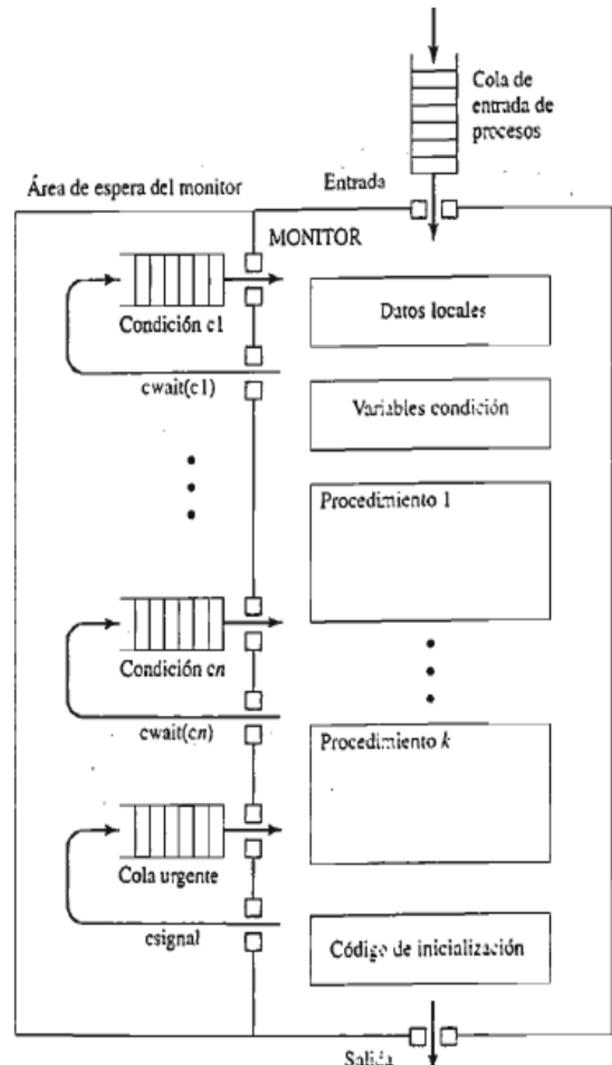
La ventaja que brinda, es que al almacenar los procedimientos dentro del módulo monitor automáticamente **se garantiza la exclusión mutua**, ya que solamente un proceso a la vez puede acceder al monitor y por ende a los procedimientos. De esta manera se desliga al programador de hacer cumplir la mutua exclusión. En cuanto a la sincronización de procesos, al igual que en semáforos, es responsabilidad del programador mediante las herramientas de sincronización que brindan los monitores (csignal y cwait).

En cambio en los semáforos, la exclusión mutua como la sincronización son responsabilidades del programador.

Otra de las ventajas de los monitores sobre los semáforos, es que es sencillo de verificar que la sincronización se ha realizado correctamente y detectar los fallos, ya que todas las funciones de sincronización están confinadas dentro del monitor.

Nota: Cwait y csignal son diferentes de las de los semáforos. Si un proceso de un monitor ejecuta un csignal y no hay procesos esperando en la variable condición, el csignal se pierde (como si nunca se hubiera ejecutado).

Nota: Una vez que un proceso está dentro del monitor, puede suspenderse a sí mismo temporalmente bajo la condición x ejecutando cwait(x); entonces se sitúa en una cola de procesos que esperan volver a entrar al monitor cuando la condición cambia (es decir, 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. Ver imagen).



TEMA 2: Interbloqueo o bloqueo mutuo.

¿Qué es el interbloqueo?

El interbloqueo se puede definir como el **bloqueo permanente** de un conjunto de procesos que compiten por los recursos del sistema o bien se comunican unos con otros. Todos los interbloqueos suponen necesidades contradictorias de recursos por parte de 2 o más procesos.

Ejemplo: Proceso P: Obtener A → Obtener B → Liberar A → Liberar B... Proceso Q: Obtener B → Obtener A → Liberar B → Liberar A...

Podría pasar por ejemplo:

- Q obtiene B y después A y a continuación libera B y A. Cuando P se ejecute podrá obtener ambos recursos.
- O que, Q obtiene B y después P obtiene A. Es inevitable el interbloqueo porque al continuar la ejecución Q se bloqueara por A y P se bloqueara por B.

Por lo tanto, que se produzca o no el interbloqueo depende tanto de la dinámica de la ejecución como de los detalles de la aplicación. Por ejemplo si P tuviese la siguiente forma: Proceso P:
Obtener A → Liberar A → Obtener B → Liberar B Independientemente de la ejecución de cada proceso, no puede darse interbloqueo.

Recursos reutilizables

Un recurso reutilizable es aquel que puede ser usado con seguridad por un proceso y **no se agota** con el uso. Como ejemplo de recursos reutilizables se tienen los procesadores, canales de E/S, memoria principal y secundaria, dispositivos y estructuras de datos tales como archivos, base de datos, semáforos, etc.

Recurso consumible

Un recurso consumible es aquél que puede ser creado (producido) y **destruido** (consumido).

Normalmente no hay límites en el número de recursos consumibles de un tipo en particular. Cuando un proceso adquiere un recurso, este deja de existir. Como ejemplo de recursos consumibles están las interrupciones, señales, mensajes e información de buffers de E/S.

¿Cuáles son las condiciones necesarias para producir un interbloqueo?

Para producirse un interbloqueo deben darse 3 condiciones:

1. **Exclusión Mutua:** Solo un proceso puede usar un recurso a la vez.
2. **Retención y espera:** Un proceso puede retener unos recursos asignados mientras espera que se le asignen otros.
3. **No apropiación:** Ningún proceso puede ser forzado a abandonar un recurso que tenga.

Puede existir interbloqueo con estas 3 condiciones, pero puede NO existir con solo estas 3 condiciones. Es decir son condiciones necesarias pero NO suficientes para que exista interbloqueo. Se necesita una cuarta condición para que se produzca interbloqueo:

4. Espera Circular (circulo vicioso de espera): Existe una cadena cerrada de procesos, cada uno de los cuales retiene, al menos un recurso que necesita el siguiente proceso de la cadena.

La cuarta condición es, en realidad, una consecuencia potencial de las 3 primeras. Es decir, dado que se producen las 3 primeras condiciones, puede ocurrir una secuencia de eventos que desemboque en un círculo vicioso de espera irresoluble. Un círculo de espera irresoluble es, de hecho, la definición de interbloqueo. En resumen, las 4 condiciones en conjunto constituyen una condición necesaria y suficiente para el interbloqueo. (1 a 3 son decisiones estratégicas, mientras que la condición 4 depende de la secuencia de solicitudes y liberación de los procesos).

Nota: Condiciones Coffman. (?)

¿Cómo prevenir un interbloqueo?

Consiste en diseñar un sistema de manera que esté excluida la posibilidad de interbloqueo. Los métodos para prevenir el interbloqueo son de 2 tipos:

- **Métodos indirectos:** impedir la aparición de alguna de las 3 condiciones necesarias.
- **Métodos directos:** evitar la aparición del cirulo vicioso de espera.

Exclusión mutua: Esta condición NO puede anularse

Retención y espera: Esta condición puede prevenirse exigiendo que todos los procesos soliciten todos los recursos que necesiten a un mismo tiempo y bloqueando el proceso hasta que todos los recursos puedan concederse simultáneamente. Ineficiente desde cualquier punto de vista:

- Un proceso puede permanecer mucho tiempo suspendido hasta que obtenga todos los recursos que necesita.
- Los recursos asignados pueden permanecer mucho tiempo sin ser utilizados por el proceso que los obtuvo.
- Un proceso puede NO conocer por adelantado todos los recursos que necesitará.

No apropiación: Esta condición puede prevenirse de varias formas:

- Si a un proceso que retiene varios recursos se le deniega una nueva solicitud, dicho proceso deberá liberar sus recursos anteriores y solicitarlos de nuevo.
- Si un proceso solicita un recurso que es retenido por otro, el SO puede solicitarle al segundo que lo libere (si los 2 procesos no tienen la misma prioridad).

Esta técnica solo sirve cuando se aplica a recursos cuyo estado puede salvarse y restaurarse más tarde de forma fácil, como el procesador.

Espera circular: Esta condición puede prevenirse definiendo una ordenación lineal de los tipos de recursos. Ineficiente como en retención y espera.

¿Cómo predecir un interbloqueo?

Con la predicción del interbloqueo, se puede alcanzar las 3 condiciones necesarias, pero se realizan elecciones acertadas para asegurar que nunca se llegue al punto de interbloqueo.

Se decide dinámicamente si la petición actual de asignación de un recurso podría llevar potencialmente a un interbloqueo. Se necesita conocer las peticiones futuras de recursos.

Dos enfoques de predicción:

- No iniciar un proceso si sus demandas pueden llevar a interbloqueo: un proceso comenzará solo si puede asegurarse la demanda máxima de recursos de todos los procesos actuales más la del nuevo proceso.
- No conceder una solicitud de recurso si esta asignación puede llevar a un interbloqueo (algoritmo del banquero): El algoritmo decide si le asigna recursos a los procesos que los solicitan para mantener el estado seguro (estado en el que existe al menos una secuencia que no lleva a interbloqueo, es decir todos los procesos pueden ejecutarse hasta el final)

Nota:

La predicción permite más concurrencia y es menos restrictiva que la prevención (se pueden alcanzar las 3 condiciones necesarias, pero se hace la asignación de manera a que nunca se llegue al interbloqueo).

Nota:

No predice el interbloqueo exactamente, sino que anticipa la posibilidad de interbloqueo y asegura que nunca exista esa posibilidad.

Para poder predecir el interbloqueo hay una serie de restricciones:

- Los procesos a considerar deben ser independientes.
- No debe haber un número variable de procesos y recursos, sino un número fijo.
- Los procesos no pueden finalizar mientras retengan recursos.
- Se debe presentar la máxima demanda de recursos por anticipado.

¿Cómo detectar un interbloqueo?

El algoritmo de detección de interbloqueo:

1. Marcar los procesos que no tengan ningún recurso asignado (fijándose en la matriz de asignación que la fila sea todo cero)
2. Crear un vector W con los recursos disponibles
3. Marcar los procesos en los que las solicitudes sean menor o igual a W y sumarlos a W.

Si al finalizar el algoritmo de detección hay algún proceso NO marcado, indica que hay interbloqueo.

Luego de detectar el interbloqueo, hace falta alguna técnica de recuperación del interbloqueo.

Algunas de ellas son:

- Abortar todos los proceso interbloqueados. Es la estrategia más común adoptada por el SO.
- Retroceder cada proceso interbloqueado hasta un punto en que no lo haya.

Estrategia integrada del interbloqueo

Es eficiente usar diferentes estrategias en diferentes situaciones:

- Espacio intercambiable → Prevención de retención y espera.
- Recursos de procesos → Predicción
- Memoria Principal → Prevención por apropiación.
- Recursos internos → Prevención por ordenación de recursos.

UNIDAD 4 – MEMORIA

TEMA: Gestión de la memoria

¿En que consiste la Gestión de memoria?

En un sistema **mono-programado**, la memoria principal se divide en 2 partes: una para el SO (normalmente en posiciones bajas de la memoria) y otra para el programa que se ejecuta en ese instante (en posiciones altas de memoria).

En cambio, en un sistema **multi-programado**, la parte de usuario de la memoria debe subdividirse aun mas para hacer lugar a varios procesos. La tarea de subdivisión la lleva a cabo dinámicamente el SO y se conoce como **gestión de memoria**.

En un sistema multiprogramado es vital repartir eficientemente la memoria para poder introducir tantos procesos como sea posible y que el procesador este la mayor parte del tiempo ocupado.

¿Cuáles son los 5 requisitos para la gestión de memoria?

1. Reubicación
2. Protección
3. Compartición
4. Organización lógica
5. Organización física

¿En que consiste la Reubicación?

El sistema busca cargar y descargar los procesos activos en la memoria principal para maximizar el uso del procesador, manteniendo una gran reserva de procesos listos para ejecutarse. Una vez que se descargó el programa a disco, cuando vuelve a ser cargado se puede necesitar **reubicar** el proceso en un área distinta de la memoria.

EL SO debe conocer la ubicación del **PCB**, de la **pila** y el punto de partida del **programa** del proceso.

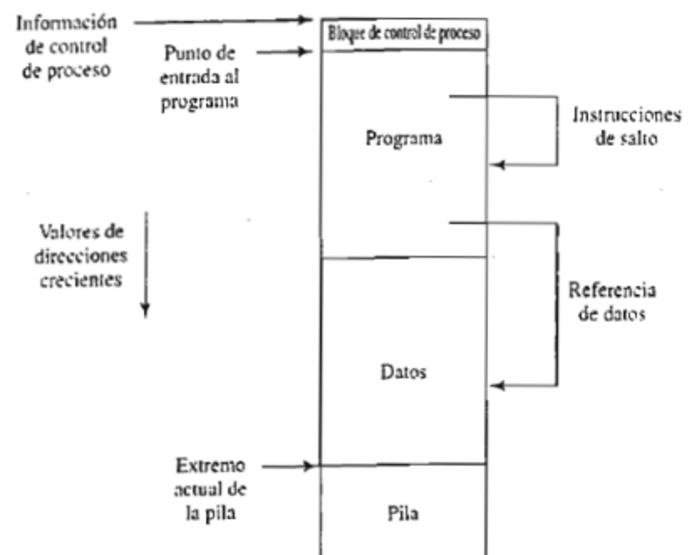


Figura 7.1. Requisitos de direccionamiento para un proceso.

El procesador y el SO deberán ocuparse de las referencia a la memoria dentro del programa, siendo que “ocuparse” hace referencia a que deberán traducir las referencias a memoria en el código del programa a las direcciones físicas reales que reflejen la posición actual del programa en MP.

- **Instrucciones de bifurcación** → Referencia a la instrucción que se va a ejecutar a continuación.
- **Referencia a los datos** → Deben contener la dirección del byte o de la palabra de datos referenciada.

El procesador y el SO deberán ocuparse de las referencia a la memoria dentro del programa, siendo que “ocuparse” hace referencia a que deberán traducir las referencias a memoria en el código del programa a las direcciones físicas reales que reflejen la posición actual del programa en MP.

- **Instrucciones de bifurcación** → Referencia a la instrucción que se va a ejecutar a continuación.
- **Referencia a los datos** → Deben contener la dirección del byte o de la palabra de datos referenciada.

¿En qué consiste la Protección?

El código de un proceso no puede hacer referencia a posiciones de memoria de otros procesos, **sin permiso**.

Al desconocerse la ubicación de un programa en memoria principal, es imposible comprobar las **direcciones absolutas** durante la compilación para asegurar la protección. Por lo tanto, todas las referencias a memoria generadas por un proceso deben comprobarse durante la ejecución para asegurar que las instrucciones solo hagan **referencia al espacio de memoria destinado a dicho proceso**. El **procesador** es el que debe satisfacer las exigencias de protección de memoria, ya que el SO no puede anticiparse a todas las referencias de memoria que hará el programa (y si la anticipación fuera posible, consumiría mucho tiempo proteger por adelantado a cada programa de posibles violaciones de referencia a la memoria).

¿En qué consiste la Compartición?

La protección debe permitir el acceso de varios procesos a la misma zona de la memoria principal. Los mecanismos para respaldar la re-ubicación forman parte básica de las capacidades de compartimiento.

¿En qué consiste la Organización lógica?

Si el SO y el hardware pueden tratar a los programas de usuario y los datos en forma de módulos, se conseguirá una serie de ventajas. La herramienta que satisface esta necesidad es la **segmentación**.

¿En qué consiste la Organización física?

La memoria secundaria puede permitir un almacenamiento a largo plazo de programas y datos, al tiempo que una memoria principal mantiene los programas y datos de uso actual.

En este esquema de 2 niveles, la organización del flujo de información entre la memoria principal y la memoria secundaria debe ser responsabilidad del sistema. **La responsabilidad de este flujo NO puede ser asignada al programador**, por dos razones:

- En un **sistema multiprogramado**, el programador no conoce durante la codificación cuánto espacio de memoria habrá disponible o donde estará este espacio.
- La memoria principal para un programa puede ser insuficiente. En este caso el programador debe emplear una práctica conocida como **superposición** (varios módulos asignados a la misma región de memoria, intercalándose entre sí según se necesite). La programación superpuesta malgasta el tiempo del programador.

¿Cuáles son las técnicas de gestión de memoria?

1. Partición estática
2. Partición dinámica
3. Paginación simple
4. Segmentación simple
5. Memoria virtual paginada
6. Memoria virtual segmentada

¿En qué consiste la partición estática?

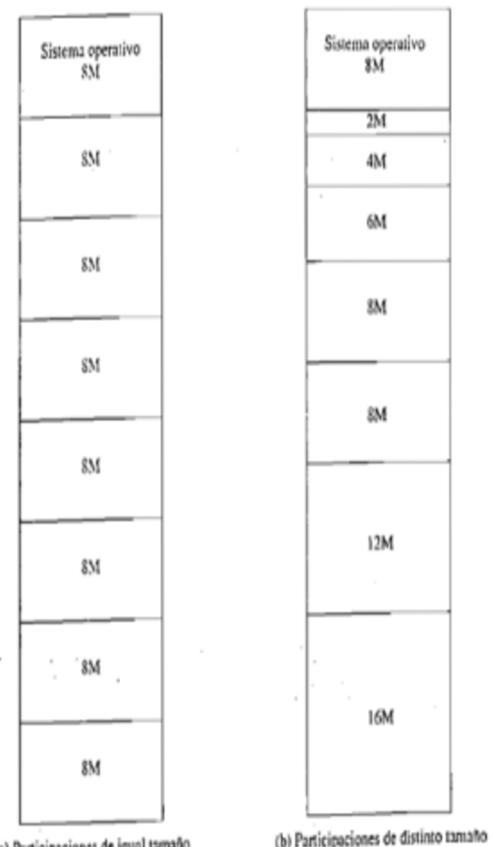
La MP se divide en un conjunto de particiones estáticas durante la generación del sistema (en el momento de arranque del SO). Un proceso se puede cargar en una partición de menor o igual tamaño. Hay dos alternativas de partición estática:

1. Emplear particiones estáticas de igual tamaño
2. Emplear particiones estáticas de distinto tamaño

Las particiones estáticas de igual tamaño plantean dos dificultades:

- El programa puede ser demasiado grande para caber en la partición. El programador deberá crear el programa mediante superposiciones.
- Uso de la MP extremadamente ineficiente. Cualquier programa, sin importar lo pequeño que sea, ocupará una partición completa. Esto genera fragmentación interna.

Ambos problemas pueden reducirse, aunque no eliminarse, por medio del empleo de **particiones estáticas de distinto tamaño**.



¿Qué es la fragmentación?

La fragmentación es el **desperdicio** de memoria. Hay 2 tipos de fragmentación:

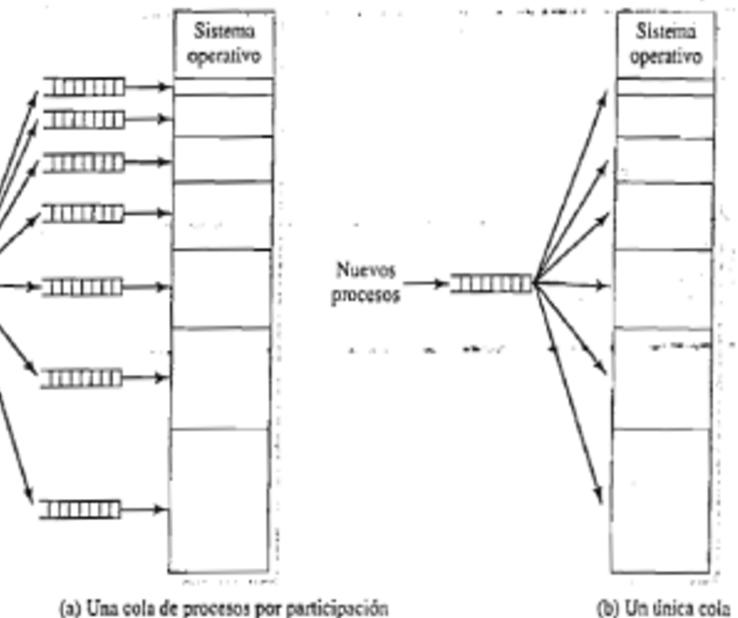
- **Fragmentación interna:** Es el desperdicio de memoria que se produce cuando un programa ocupa una partición de memoria más grande de la que necesita.
- **Fragmentación externa:** Es el desperdicio de memoria que se produce cuando hay memoria disponible, pero esta no se puede usar, porque es menor de la que requiere el programa.

¿Cómo funciona el algoritmo de ubicación de la partición estática?

Con particiones del mismo tamaño, la ubicación de un proceso en la memoria es trivial. Mientras haya alguna partición libre, puede cargarse un proceso en esa partición y no importa la partición que se use (ya que todas son del mismo tamaño). Si están todas las particiones ocupadas, sacar a un proceso de ella es una cuestión de planificación de procesos.

Con particiones de distinto tamaño hay 2 formas de asignar los procesos a las particiones:

1. La más fácil, es asignar cada proceso a la partición más pequeña que quepa, de forma que los procesos siempre están asignados de forma que se minimiza la memoria desaprovechada dentro de cada partición. Sin embargo, esto no es conveniente porque pueden haber particiones sin usar, y que dicha partición no sea asignada al proceso porque esta no es la partición más pequeña en la que quepa.
2. Asignar a cada proceso la partición más pequeña disponible que quepa.



Resumen:

Partición Estática	Ventajas	Desventajas
La MP se divide en un conjunto de particiones estáticas durante la generación del sistema. Un proceso se puede cargar en una partición de menor o igual tamaño	Sencillo de implementar. Poca sobrecarga del SO.	Fragmentación interna El número de procesos activos es fijo.

¿En qué consiste la partición dinámica?

Las particiones se crean dinámicamente, de forma que cada proceso se carga en una partición de exactamente el mismo tamaño que el proceso. Es decir se le asigna tanta memoria como necesita y no más.

Este método comienza bien, pero finalmente, desemboca en una situación en la que hay un gran número de [huecos pequeños de memoria](#). Conforme pasa el tiempo, la memoria comienza a estar más fragmentada y su rendimiento decae. Se produce **fragmentación externa**.

Una técnica para vencer a la fragmentación externa, es la **compactación**. De vez en cuando, el sistema operativo desplaza los procesos para que estén contiguos, de forma que toda la memoria libre quede junta en un bloque. El problema es que es un procedimiento que consume tiempo, por lo que desperdicia tiempo del procesador.

¿Cómo funciona el algoritmo de ubicación de la partición dinámica?

En el esquema de particiones dinámicas se pueden considerar 3 algoritmos. Los 3 se limitan a elegir entre los bloques de memoria libres que son mayores o iguales que el proceso a cargar.

- **Mejor ajuste (best-fit)** → Elige el bloque de tamaño mas próximo al solicitado.
- **Primer ajuste (first-fit)** → Recorre la memoria desde el principio y escoge el primer bloque disponible suficientemente grande.
- **Siguiente ajuste (next-fit)** → Recorre la memoria desde la ultima ubicación y elige el siguiente bloque disponible que sea suficientemente grande.

El mejor método aplicable dependerá de la secuencia exacta de intercambios de procesos que ocurran y del tamaño de estos procesos.

<i>Primer ajuste</i>	<i>Siguiente Ajuste</i>	<i>Mejor ajuste</i>
Sencillo. El mejor y más rápido.	<p><i>Genera peores resultados que el primer ajuste.</i></p> <p><i>Necesita una compactación más frecuente que el primer ajuste ya que el bloque de memoria libe más grande, que suele aparecer al final del espacio de memoria, se divide rápidamente en fragmentos pequeños.</i></p>	<p><i>Proporciona en general los peores resultados</i></p> <p>Al elegir el bloque de tamaño más aproximado al tamaño del proceso, genera rápidamente huecos muy pequeños. Así pues debe compactar con más frecuencia que los otros 2 algoritmos.</p>

¿En qué consiste el sistema de colegas?

Los bloques de memoria disponibles son de tamaño 2^L K\$, para valor de K tal que $L \leq K \leq U$ y donde:

- 2^L K\$ = tamaño de bloque mas pequeño asignable.
- 2^U K\$ = tamaño de bloque mas grande assignable (generalmente es el tamaño de la memoria entera disponible para asignar)

1 bloque de 1 Mbyte	1M				
Solicitar 100K	A = 128K	128K	256K		512K
Solicitar 240K	A = 128K	128K	B = 256K		512K
Solicitar 64K	A = 128K	C = 64K	64K	B = 256K	512K
Solicitar 256K	A = 128K	C = 64K	64K	B = 256K	D = 256K
Liberar B	A = 128K	C = 64K	64K	256K	D = 256K
Liberar A	128K	C = 64K	64K	256K	D = 256K
Solicitar 75K	E = 128K	C = 64K	64K	256K	D = 256K
Liberar C	E = 128K	128K		256K	D = 256K
Liberar E			512K		D = 256K
Liberar D				IM	

Figura 7.6. Ejemplo de sistema Buddy.

La figura anterior muestra un ejemplo utilizando un bloque inicial de 1MB. La primera solicitud A de 100KB necesitara un bloque de 128KB. El bloque inicial se divide en 2 colegas de 512KB. El primero de estos se divide en 2 de 256KB y el primero de estos se divide en 2 de 128K.

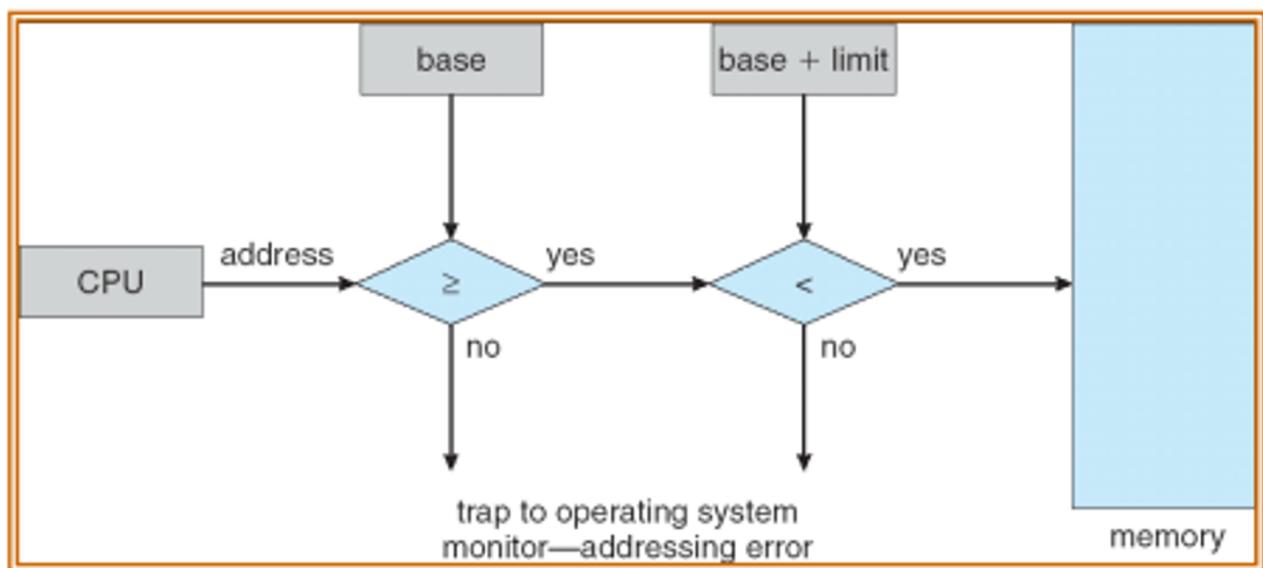
El sistema de colegas es un equilibrio razonable para superar las desventajas de los esquemas de partición fija y variable.

Aclaraciones de la reubicación

Aclaraciones:

- La dirección absoluta no se calcula hasta que se ejecute una instrucción. Para asegurar que esta función no degrade el rendimiento, debe realizarse por medio de un hardware especial del procesador, en vez de por software. Este hardware se denomina **unidad de gestión de memoria** (MMU, memory management unit).
- El programa de usuario maneja direcciones lógicas y el hardware de conversión de memoria convierte esas direcciones lógicas en direcciones físicas.

- La vinculación de instrucciones y datos a direcciones de memoria puede realizarse en tres etapas diferentes:
 - **Compilación:** Si se conoce a priori la posición que va a ocupar un proceso en la memoria se puede generar código absoluto con referencias absolutas a memoria; si cambia la posición del proceso hay que recomilar el código.
 - **Carga:** Si no se conoce la posición del proceso en memoria en tiempo de compilación se debe generar código reubicable; el compilador no generará direcciones reales de memoria principal sino direcciones relativas a algún punto conocido, como el comienzo del programa. Al momento de la carga del proceso a memoria principal, se convierten todas las direcciones relativas a direcciones absolutas. El problema surge que en un sistema multiprogramado, el proceso puede ser suspendido luego de la carga en memoria, y vuelto a ser cargado en otra posición diferente.
 - **Ejecución:** Si no se conoce la posición del proceso en memoria en tiempo de compilación se debe generar código reubicable. Es lo que se conoce como carga dinámica en tiempo de ejecución.
- Las direcciones lógicas y físicas son iguales en los esquemas de vinculación en tiempo de compilación y de carga; pero difieren en el esquema de vinculación en tiempo de ejecución.
- Un par de registros base y límite definen el espacio de direcciones lógicas. También se llaman **registros de reubicación** y se usan para proteger los procesos de usuario unos de otros.



¿En qué consiste la paginación simple?

Mediante la paginación simple, la memoria principal se encuentra dividida en trozos iguales de tamaño fijo, denominados **marcos**. A su vez, cada proceso está dividido también en trozos de tamaño fijo y del mismo tamaño que los de memoria, denominados **páginas**. El término marco se utiliza porque un marco puede mantener o encuadrar una página de datos. Es decir, cuando se introduce un proceso en la memoria, se cargan todas sus páginas en los marcos libres.

El espacio desaprovechado en la memoria para cada proceso por fragmentación consta solo de una fracción de la última página del proceso. Además, no hay fragmentación externa.

El sistema operativo mantiene una lista de los marcos libres. Cuando llega el momento de cargar un proceso en memoria, el SO busca los marcos libres necesarios para cargar todas las páginas del proceso. No necesariamente los marcos deben ocupar una posición contigua de memoria. Para lograr esto, el SO operativo mantiene una **tabla de páginas** para cada proceso. Cada tabla de página contiene una entrada por cada página del proceso. Y en cada entrada se encuentra el número de marco de la memoria principal que alberga la página correspondiente.

Al igual que la partición simple, el procesador también realiza la traducción de direcciones lógicas a físicas. En el caso de la partición simple el programa contenía direcciones relativas al comienzo del programa. En la paginación, el programa contiene direcciones lógicas que constan de un número de página y de un desplazamiento dentro de la página. Para hacer la traducción, el procesador debe acceder a la tabla de páginas del proceso actual. Dada una dirección lógica (número de página, desplazamiento), el procesador emplea la tabla de páginas para obtener una dirección física (número de marco, desplazamiento).

La diferencia entre la paginación simple y la partición estática, es que con la paginación, las particiones son algo más pequeñas (por lo que la fragmentación interna será menor) y que un programa puede ocupar más de una partición y estas no tienen por qué ser contiguas.

Aplicación correcta del esquema de paginación - **IMPORTANTE**

El tamaño de la página y por lo tanto, el tamaño del marco, debe haber una potencia de 2. Un ejemplo donde se emplean direcciones de 16 bits y el tamaño de página es de 1K = 1024 bytes. Con un tamaño de página de 1K, se necesitan 10 bits para el campo desplazamiento ($2^{10} = 1024$ bytes). De este modo, un programa puede estar formado por un máximo de 26 = páginas de 1KB cada una.

Si se tiene una dirección lógica = 0000010111011110, se corresponde con el número de página 1 (000001) y desplazamiento 478 (0111011110). Suponiendo se fija en la entrada 1 de la tabla de página y dicha entrada contiene el marco 6 = 000110 en binario. Entonces la dirección física es el marco 6 y el desplazamiento 478 = 0001100111011110.

¿En qué consiste la segmentación simple?

Otro modo de **subdividir** el programa es la **segmentación**. En este caso, el programa se divide en un conjunto de **segmentos**. No es necesario que todos los segmentos de todos los programas tengan la misma longitud, aunque existe una longitud máxima de segmento. En este esquema, una dirección lógica consta de 2 partes, un número de segmento y un desplazamiento.

Este esquema resulta similar a la partición dinámica. La diferencia, radica en que, con segmentación un programa puede ocupar más de una partición y estas no tienen por qué estar contiguas.

La segmentación no sufre de fragmentación interna, pero, como en la partición dinámica, sufre de fragmentación externa. Aunque esta será menor.

Mientras que la paginación es transparente al programador, la segmentación es visible.

Para la traducción de direcciones lógicas a físicas, al igual que en la paginación, el SO hará uso de una **tabla de segmentos** para cada proceso y una lista de bloques libres en la memoria principal. Cada entrada a la tabla de segmentos tendrá que contener la dirección de comienzo del segmento correspondiente de la memoria principal.

MEMORIA VIRTUAL

¿En qué consiste la memoria virtual?

No es necesario que todas las páginas o todos los segmentos de un proceso se encuentren en la memoria principal durante la ejecución.

Supongamos que se tiene que traer un nuevo proceso de memoria. El sistema operativo comienza trayendo únicamente una o dos porciones, que incluye la porción inicial del programa y la porción inicial de datos sobre la cual acceden las primeras instrucciones acceden. Esta parte del proceso que se encuentra realmente en la memoria principal para, cualquier instante de tiempo, se denomina **conjunto residente** del proceso. Cuando el proceso está ejecutándose, las cosas irán perfectamente mientras que todas las referencias a la memoria se encuentren dentro del conjunto residente. Usando una tabla de segmentos o páginas, el procesador siempre es capaz de determinar si esto es así o no. Si el procesador encuentra una dirección lógica que no se encuentra en la memoria principal, generará una interrupción indicando un fallo de acceso a la memoria. El sistema operativo coloca al proceso interrumpido en un estado de bloqueado y toma el control. Para que la ejecución de este proceso pueda reanudarse más adelante, el sistema operativo necesita traer a la memoria principal la porción del proceso que contiene la dirección lógica que ha causado el fallo de acceso. Con este fin, el sistema operativo realiza una petición de E/S, una lectura a disco. Después de realizar la petición de E/S, el sistema operativo puede activar otro proceso que se ejecute mientras el disco realiza la operación de E/S. Una vez que la porción solicitada se ha traído a la memoria principal, una nueva interrupción de E/S se lanza, dando control de nuevo al sistema operativo, que coloca al proceso afectado de nuevo en el estado Listo.

Esta nueva estrategia de no cargar todo el proceso en memoria conduce a mejorar la utilización del sistema ya que:

1. Puede mantenerse un mayor numero de procesos en memoria principal
2. Un proceso puede ser mayor que toda la memoria principal. Con la memoria virtual basada en paginación o segmentación, el sistema operativo automáticamente carga porciones de un proceso en la memoria principal cuando estas se necesitan.

La memoria virtual permite una multiprogramación muy efectiva que libera al usuario de las restricciones excesivamente fuertes de la memoria principal.

¿Qué es la memoria virtual y en qué consiste la Proximidad?

Se puede hacer un mejor uso de la memoria cargando únicamente unos pocos fragmentos. Entonces, si el programa hace referencia a un dato que se encuentra en una porción de memoria que no está en la memoria principal, entonces se dispara una interrupción. Éste indica al sistema operativo que debe conseguir la porción deseada

Así, en cualquier momento, sólo unas pocas porciones de cada proceso se encuentran en memoria, y por tanto se pueden mantener más procesos alojados en la misma. Además, se ahorra tiempo porque las porciones del proceso no usadas no se cargan ni se descargan de la memoria

Sin embargo, el sistema operativo debe saber cómo gestionar este esquema. Cuando el sistema operativo traiga una porción a la memoria, debe expulsar otra. Si elimina una porción justo antes de que vaya a ser utilizada, deberá recuperar dicha porción de nuevo casi de forma inmediata.

Demasiados intercambios de fragmentos lleva a una condición denominada **hiperpaginación** (thrashing): el procesador consume más tiempo intercambiando fragmentos que ejecutando instrucciones de usuario. Para evitar esto, el SO trata de adivinar, en base a la historia reciente, qué porciones son menos probables de ser utilizadas en un futuro cercano. Esto se basa en el **principio de cercanía**. Este principio indica que las referencias al programa y a los datos dentro de un proceso tienden a agruparse. Por tanto, se resume que sólo unas pocas porciones del proceso se necesitarán a lo largo de un periodo de tiempo corto.

¿En qué consiste la paginación?

Para la memoria virtual basada en el esquema de paginación también se necesita una tabla de páginas por proceso. En este caso, debido a que sólo algunas de las páginas de proceso se encuentran en la memoria principal, se necesita un bit en cada entrada de la tabla de páginas para indicar si la correspondiente página está presente (P) en memoria principal o no lo está. Si el bit indica que la página está en memoria, la entrada también debe indicar el número de marco de dicha página.

Otro bit de control necesario en la entrada de la tabla de páginas es el **bit de modificado (M)**, que indica si los contenidos de la correspondiente página han sido alterados desde que la página se cargó por última vez en la memoria principal. Si no ha habido cambios, no es necesario escribir la página cuando llegue el momento de reemplazarla por otra página en el marco de página que actualmente ocupa. Pueden existir también otros bits de control en estas entradas.

Estructura de la tabla de páginas

Debido a que la tabla de páginas es de longitud variable dependiendo del tamaño del proceso, la cantidad de memoria demandada por las tablas de página únicamente puede ser inaceptablemente grande.

Por esto, la mayoría de los esquemas de memoria virtual almacenan las tablas de páginas también en la memoria virtual, en lugar de en la memoria real. Esto representa que las tablas de páginas están sujetas a paginación igual que cualquier otra página. Cuando un proceso está en ejecución, al menos parte de su tabla de páginas debe encontrarse en memoria, incluyendo la entrada de tabla de páginas de la página actualmente en ejecución

Algunos procesadores utilizan un [esquema de dos niveles](#) para organizar las tablas de páginas de gran tamaño.

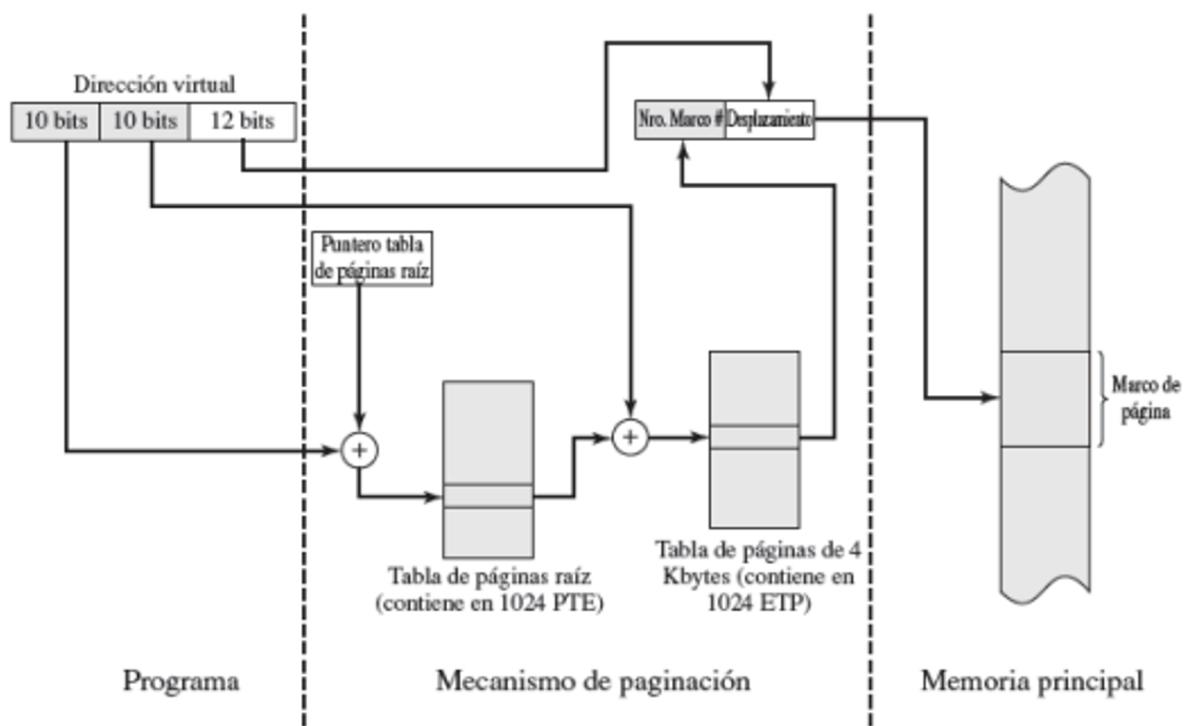


Figura 8.5. Traducción de direcciones en un sistema de paginación de dos niveles.

Un ejemplo de un esquema típico de dos niveles que usa 32 bits (232) para la dirección con tamaño de páginas de 4K, tiene una dirección lógica formada por:

- Un desplazamiento de 2^{12} (4K)
- Un numero de pagina de 20 bits

Ya que la tabla de páginas está paginada y cada entrada de la tabla de páginas ocupa 4 bytes, el número de página es de nuevo dividido en:

- Un numero de pagina de 10 bits
- Un desplazamiento de 10 bits

Por tanto, una dirección lógica tiene el siguiente aspecto:

Donde P1 es un índice a la tabla de páginas raíz y P2 es un desplazamiento en la segunda tabla de páginas.

Una estrategia alternativa al uso de tablas de páginas de uno o varios niveles es el uso de la estructura de tabla de **páginas invertida**.

número de página	desplazamiento
p_1	p_2

10 10 12

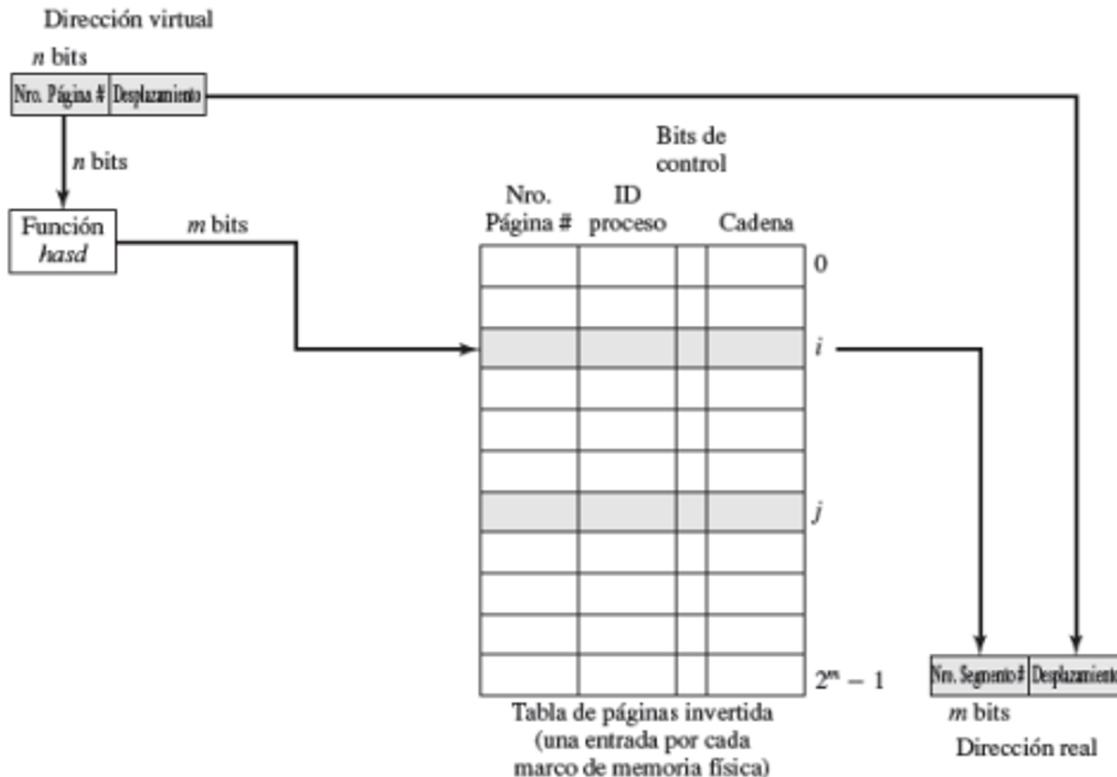


Figura 8.6. Estructura de tabla de páginas invertida.

La tabla de páginas invertida contiene un registro por cada marco de página real en lugar de un registro por cada página virtual. La estructura de la tabla de páginas se denomina invertida debido a que se indexan sus entradas de la tabla de páginas por el número de marco en lugar de por el número de página virtual. Para un tamaño de memoria física de 2^m marcos, la tabla de páginas invertida contiene 2^m entradas, de forma que la entrada en la posición i -esima se refiere al marco i .

Las entradas contienen la dirección virtual de la página almacenada en el marco con información sobre el proceso que la posee.

Disminuye la memoria necesaria para almacenar cada tabla de páginas.

¿En qué consiste el Buffer de traducción adelantada y el tamaño de pagina en la paginación?

En principio, **toda referencia a la memoria virtual puede causar dos accesos a memoria física:** **uno** para buscar la entrada la tabla de páginas apropiada y **otro** para buscar los datos solicitados.

Por esta situación, la mayoría de esquemas de la memoria virtual utilizan una cache especial hardware de alta velocidad para las entradas de la tabla de página **TLB (translation lookaside buffer)**. Esta contiene aquellas entradas de la tabla de páginas que han sido usadas de forma más reciente. Dada una dirección virtual, el procesador primero examina la TLB, si la entrada de la tabla de páginas solicitada está presente (acierto en TLB), entonces se recupera el número de marco y se construye la dirección real. Los estudios sobre la TLB han demostrado que este esquema significa una importante mejora del rendimiento.

Tamaño de pagina

Cuanto menor es el tamaño de la página, menor cantidad de fragmentación interna. Por otro lado, cuanto menor es la página, mayor número de páginas son necesarias para cada proceso. Un mayor número de páginas por proceso significa también que el tamaño de las tablas de páginas será mayor. Esto puede significar que una gran parte de las tablas de páginas de los procesos activos debe estar en memoria virtual y no en memoria principal. Esto puede provocar 2 fallos de página para una única referencia a memoria:

- **Para traer la parte necesaria de la tabla de paginas**
- **Para traer la pagina del proceso**

Por principio de proximidad, si el tamaño de página es muy pequeño habrá gran cantidad de páginas disponibles en la memoria principal para cada proceso. Después de un tiempo, las páginas en memoria contendrán las partes de los procesos a las que se ha hecho referencia de forma reciente. De esta forma, la tasa de fallos de página debería ser baja. A medida que el tamaño de páginas se incrementa, la página en particular contendrá información más lejos de la última referencia realizada. Así pues, el efecto del principio de proximidad se debilita y la tasa de fallos de página comienza a crecer.

Para un tamaño determinado de una TLB, a medida que el tamaño del proceso crece y la proximidad de referencias decrece, el índice de aciertos en TLB se va reduciendo. Bajo estas circunstancias, la TLB se puede convertir en el cuello de botella del rendimiento.

¿Cómo se da la Paginación y Segmentación combinadas?

La paginación es transparente al programador y elimina la fragmentación externa, y por tanto proporciona un uso eficiente de la memoria principal.

La segmentación sí es visible al programador, incluyendo la posibilidad de manejar estructuras de datos que crecen, modularidad, y dar soporte a la compartición y a la protección. A su vez, elimina la fragmentación interna.

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.

¿En qué consiste la Protección y seguridad de la segmentación?

La segmentación proporciona protección y compartición. Cada entrada en la tabla de segmentos incluye la longitud así como la dirección base, por lo que un programa NO puede acceder a una posición de memoria más allá de los límites del segmento. Para conseguir compartición, es posible que un segmento se encuentre referenciado desde las tablas de segmentos de más de un proceso. Los mecanismos están disponibles en los sistemas de paginación. Sin embargo, en este caso la estructura de páginas de un programa y los datos no son visibles para el programador, haciendo que la especificación de la protección y los requisitos de compartición sean más difíciles.

¿Qué son las políticas de lectura?

La política de lectura (fetch) está relacionada con la decisión de cuando se debe cargar una página en la memoria principal:

- **Paginación por demanda** → Una página se trae a memoria sólo cuando se hace referencia a una posición en dicha página. Al principio se producirá un aluvión de fallos de página, pero luego al estar cargadas varias páginas utilizadas recientemente, por el principio de proximidad, la situación se estabilizará y el número de fallos será muy bajo.
- **Paginación previa** → Se traen a memoria también otras páginas, diferentes de la que ha causado el fallo de página. Es mucho más eficiente traer a la memoria un número de páginas contiguas de una vez, en lugar de traerlas una a una a lo largo de un periodo de tiempo más amplio (por la operación de E/S a disco). Por supuesto, esta política es ineficiente si la mayoría de las páginas que se han traído no se referencian a posteriori.

¿Qué son las políticas de ubicación?

La política de ubicación determina en qué parte de la memoria real van a residir las porciones de un proceso. En los sistemas de segmentación puros, la política de ubicación es un aspecto de diseño **muy importante** (políticas del estilo mejor ajuste, primer ajuste). Para sistemas que usan paginación pura o paginación combinada con segmentación, la ubicación es **irrelevante** debido a que el hardware de traducción de direcciones y el hardware de acceso a la memoria principal pueden realizar sus funciones en cualquier combinación de página-marco con la misma eficiencia.

¿Qué son las políticas de reemplazo?

La política de reemplazo determina entre el conjunto de páginas consideradas, cuál es la página específica que debe elegirse para el reemplazo. Todas las políticas de reemplazo tienen como objetivo que la página que va a eliminarse sea aquella que tiene menos posibilidades de volver a ser referenciada en un futuro cercano. Cuanto más elaborada y sofisticada es una política de reemplazo, mayor va a ser la sobrecarga a nivel software y hardware para implementarla.

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.

¿Qué algoritmos de reemplazo encontramos?

Existen ciertos algoritmos básicos que se emplean para la selección de una página a reemplazar:

1 - Óptima

Óptima selecciona para reemplazar la página que tiene que esperar una mayor cantidad de tiempo hasta que se produzca la referencia siguiente. Genera el menor número de fallos de página pero es **imposible de implementar** ya que requiere que el SO tenga un conocimiento exacto de los sucesos futuros.

2 - Usada menos recientemente (LRU)

LRU reemplaza la página de memoria que ha sido referenciada desde hace más tiempo. **LRU** proporciona unos resultados casi tan buenos como la política óptima. El problema con esta alternativa es la dificultad en su implementación. Es una opción costosa en cuanto a la sobrecarga del sistema.

3 - Primero entrar, primero en salir (FIFO)

FIFO trata los marcos de página ocupados como si se tratase de un buffer circular, y las páginas se reemplazan mediante una estrategia cíclica de tipo **round-robin**. Sencillo de implementar pero su rendimiento es relativamente pobre ya que reemplaza la página que lleva en memoria más tiempo y por lo general que estos sean las zonas utilizadas de forma intensiva durante todo el tiempo de vida del proceso.

4- Reloj

Reloj asocia un bit a cada marco, denominado bit de uso. Cuando una página se trae por primera vez a la memoria, el bit de usado de dicho marco se pone a 0. Cuando se hace referencia a la página posteriormente, el bit se pone en 1. El conjunto de marcos a ser reemplazados se considera como un buffer circular con un puntero asociado. Cuando llega el momento de reemplazar una página, el sistema operativo recorre el buffer para encontrar un marco con su bit de usado a 0. Cada vez que encuentra un marco con el bit de usado a 1, se reinicia este bit a 0 y se continúa. Si alguno de los marcos del buffer tiene el bit de usado a 0 al comienzo de este proceso, el primero de estos marcos que se encuentre se seleccionará para reemplazo. Si todos los marcos tienen el bit a 1, el puntero va a completar un ciclo completo a lo largo del buffer, poniendo todo los bits de usado a 0, parándose en la posición original, reemplazando la página en dicho marco. **Véase que esta política es similar a FIFO, excepto que, en la política del reloj, el algoritmo saltará todo marco con el bit de usado a 1.** Se aproxima al rendimiento LRU sin introducir mucha sobrecarga.

El algoritmo del reloj puede hacerse más potente incrementando el número de bits que utiliza (y cuantos menos bits se utilicen se degenera en FIFO). Se asocia un bit de modificado que combinado con el bit de usado, en resumen, el algoritmo de reemplazo de páginas da vueltas a través de todas las páginas del buffer buscando una que no se haya modificado desde que se ha traído y que no haya sido accedida recientemente. Esta página es una buena opción para reemplazo y tiene la ventaja que, debido a que no se ha modificado, no necesita escribirse de nuevo en la memoria secundaria.

¿En qué consiste el almacenamiento intermedio de páginas?

Para mejorar el rendimiento, una página remplazada no se pierde sino que se asigna a una de las dos siguientes listas: la lista de páginas libres si la página no se ha modificado, o la lista de páginas modificadas si lo ha sido. La página no se mueve físicamente de la memoria, en su lugar, se suprime su entrada en la tabla de páginas y se pone en la lista de páginas libres o modificadas.

En efecto, la lista de páginas modificadas y libres actúa como una cache de páginas (la página que se va a reemplazar se mantiene en la memoria).

¿Qué es la gestión de conjunto residente?

- **Asignación Fija** → proporciona un número fijo de marcos de memoria principal disponibles para ejecución. Siempre que se produzca un fallo de página del proceso en ejecución, la página que se necesite reemplazará una de las páginas del proceso.
- **Asignación variable** → permite que se reserven un número de marcos por proceso que puede variar a lo largo del tiempo de vida del mismo. Muchos fallos de páginas, se asignan marcos adicionales al proceso. Pocos fallos de página, se saca marcos que no utiliza. La dificultad de esta estrategia se deben a que el SO debe saber cuál es el comportamiento del proceso activo por lo que produce una sobrecarga en el sistema.

¿Cuál es el alcance de la política de reemplazo?

La política de reemplazo surge cuando se produce un fallo de página. Esta puede ser:

- **Reemplazo de Alcance local** → Se escoge entre las páginas residentes del proceso que originó el fallo.
- **Reemplazo de Alcance Global** → Se escoge entre todas las páginas de la memoria para reemplazo.

NO es posible aplicar asignación fija y una política de reemplazo global.

¿Qué son las políticas de vaciado?

Determina el momento en el que hay que escribir en la memoria secundaria una página modificada.

- **Vaciado por demanda** → Una página se escribirá en la memoria secundaria solo cuando haya sido elegida para reemplazarse.
- **Vaciado previo** → Escribe las páginas modificadas antes de que se necesiten sus marcos, de forma que las páginas puedan escribirse por lotes.

Las 2 políticas son inefficientes, ya que el vaciado por demanda la escritura de una página es anterior a la lectura de una nueva página, por lo que un proceso que sufra una falla de página pueda tener que esperar 2 transferencias de páginas antes de desbloquearse. Con el vaciado previo, las páginas pueden ser nuevamente modificadas luego de ser escritas en memoria secundaria.

Solución: incorporar almacenamiento intermedio de páginas → Se desactivan las políticas de reemplazo y vaciado para dar lugar al almacenamiento intermedio.

¿En qué consiste el control de carga?

Determinar el grado de multiprogramación: pocos procesos en la memoria principal, entonces menor aprovechamiento de la CPU. Si hay demasiados procesos, el tamaño medio del conjunto residente de cada proceso no será el adecuado y se producirán frecuentes fallos de páginas. El resultado es la **hiperpaginación**.

UNIDAD 5 – ENTRADA / SALIDA , ARCHIVOS

TEMA: Gestión de E/S y planificación de discos

¿Qué técnicas de E/S encontramos?

Hay 3 técnicas para realizar la E/S:

- **E/S programada:** el procesador emite una orden de E/S de parte de un proceso a un módulo de E/S; el proceso espera hasta que se complete la operación antes de continuar (espera activa, se queda preguntando al dispositivo de E/S si ya está listo para realizar la E/S - Bit de ocupado/desocupado de la controladora).
- **E/S dirigida por interrupción:** el procesador emite una orden de E/S de parte de un proceso a un módulo de E/S, el procesador continua la ejecución de las instrucciones siguientes mientras el modulo E/S se prepara para la E/S. Cuando está listo para realizar la operación de E/S manda una interrupción y el procesador la capta (entre medio de cada instrucción verifica la línea de interrupciones). El procesador interrumpe lo que estaba haciendo y comienza la transferencia.
- **Acceso directo a memoria:** un módulo de DMA, que es un procesador de propósito especial, controla el intercambio de datos entre la memoria principal y un módulo de E/S. El procesador envía una petición de transferencia al DMA. El DMA interrumpe al procesador solo para indicarle que la transferencia ya terminó.

¿Cómo se da el acceso directo a la memoria?

EL DMA transfiere datos desde y hacia la memoria a través del bus del sistema. Normalmente el DMA debe usar el bus solamente cuando el procesador no lo necesite o debe obligar al procesador a que se suspenda temporalmente su operación. Esta última técnica es más común y se denomina **robo de ciclos de bus** donde el procesador debe esperar un ciclo de bus mientras la unidad de DMA transfiere una palabra. **El efecto final es que el procesador ejecuta más lentamente.** Sin embargo, para una transferencia de varias palabras, el DMA es mucho más eficiente que la E/S programada o la dirigida.

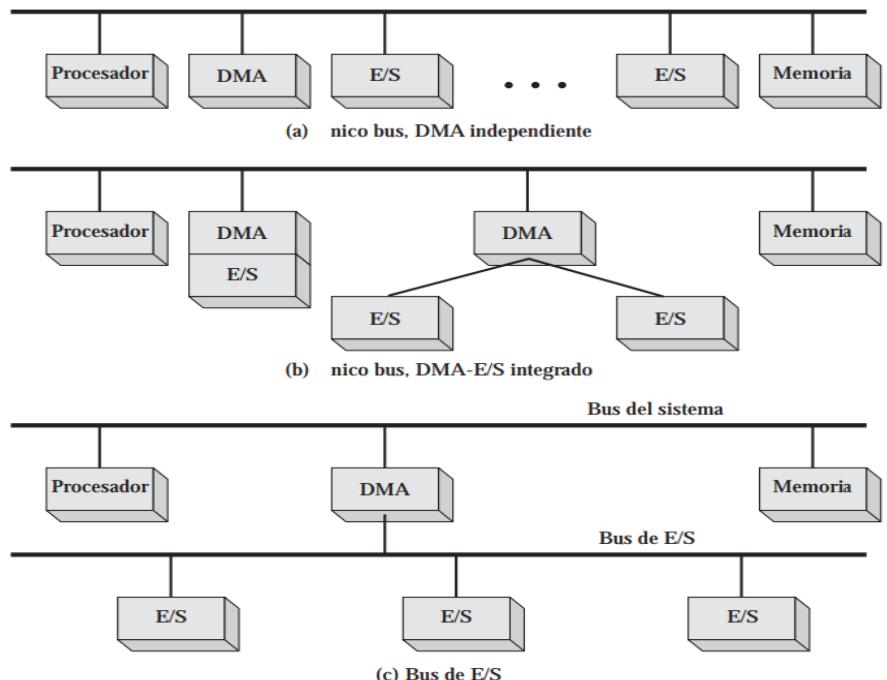


Figura 11.3. Configuraciones de DMA alternativas.

En las alternativas de configuraciones de DMA (imagen de arriba) se ve que:

- **En el caso A,** EL DMA y los dispositivos de E/S utilizan el bus del sistema por lo que la transferencia de una palabra consume 2 ciclos de bus: solicitud de transferencia + la transferencia.
- **En el caso B y C,** se conectan los módulos de E/S al módulo de DMA mediante un bus de E/S, por lo que el bus del sistema solo es utilizado por el DMA para intercambiar datos con la memoria y señales de control con la CPU. El intercambio de datos entre el módulo de DMA y el de E/S tiene lugar fuera del bus del sistema, achicando así el número de ciclos de bus del sistema requeridos.

¿Qué aspectos de diseño se tienen en cuenta en los sistemas operativos?

Las operaciones de E/S constituyen un cuello de botella en los sistemas informáticos ya que la mayoría de los dispositivos de E/S son extremadamente lentos en comparación con la memoria principal y el procesador. Una manera de abordar este problema es con el uso de la multiprogramación, que permite que algunos procesos esperen la E/S mientras otro proceso está ejecutado.

¿En qué consiste el almacenamiento intermedio o buffering? Que es doble buffer?

Un buffer es una memoria intermedia que almacena datos mientras se están transmitiendo entre 2 dispositivos o entre un dispositivo y una aplicación. **Es un amortiguador de velocidades.**

Si un proceso realiza una E/S sin buffer, este debe quedar residente en la memoria principal, no pudiendo ser expulsado a disco. Esta condición reduce la oportunidad de usar el intercambio al fijar como residente parte de la memoria principal, lo que conlleva una disminución del rendimiento global del sistema. Asimismo, el dispositivo de E/S queda asociado al proceso durante la duración de la transferencia, no estando disponible mientras tanto para otros procesos.

Por lo tanto, en un entorno **multiprogramado**, donde hay múltiples operaciones de E/S, el almacenamiento intermedio es una herramienta que puede incrementar la eficiencia del SO y el rendimiento de los procesos individuales.

Doble buffer

Un proceso puede transferir datos hacia o desde una memoria intermedia mientras el SO vacía o rellena el otro.

¿En qué consiste la planificación de discos?

- **Tiempo de búsqueda** → tiempo que se tarda en ubicar la cabeza en la pista.
- **Tiempo de giro** → tiempo que tarda el inicio del sector deseado en llegar hasta la cabeza.
- **Tiempo de acceso** → suma del tiempo de búsqueda + el retardo de giro

El orden en que se leen los sectores del disco tiene un efecto inmenso en el rendimiento de la E/S. Acá entra en juego las políticas de planificación del disco.

Si los pedidos de lectura a disco no llegan en simultáneo, no hay nada que planificar ya que hay un solo pedido a la vez.

Las políticas de planificación son:

- **FIFO (Primero en entrar primero en salir):** Buen rendimiento si hay pocos procesos que requieren acceso y si muchas de las solicitudes son a sectores agrupados de un archivo. Sino mal rendimiento (como si fuese aleatorio).
- **Prioridad:** la planificación queda fuera del control del software de gestión de disco. Esta estrategia no está diseñada para optimizar la utilización del disco sino satisfacer otros objetivos del SO. Favorece a los trabajos cortos, por los que los trabajos mayores pueden tener que esperar excesivamente. Poco favorable para sistemas de bases de datos.
- **LIFO (Último en entrar, primero en salir):** Puede producir inanición. Buen rendimiento en los sistemas de transacciones ya que conceder el dispositivo al último usuario acarrea pocos movimientos de brazos al recorrer un archivo secuencial.

Las siguientes políticas tienen en cuenta la posición actual de la pista.

- **SCAN (subo hasta la última pista del disco y bajo):** Favorece a los trabajos con solicitudes de pistas cercanas a los cilindros más interiores y exteriores; así como a los últimos trabajos en llegar. El primer problema se puede evitar con C-SCAN, mientras el 2do con SCAN N pasos.

¿Qué es un RAID y de qué tipos hay?

RAID (Vector Redundante de Discos Independientes) es un conjunto de unidades de disco físico vistas por el SO como una sola unidad lógica.

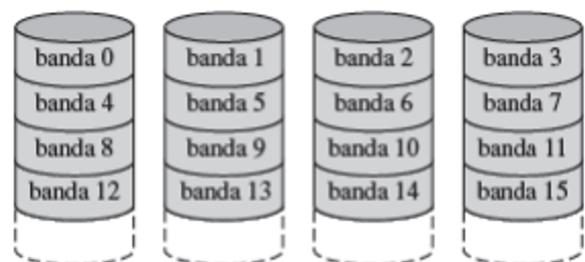
Con múltiples discos que funcionan en forma independiente y en paralelo, las distintas solicitudes de E/S se pueden gestionar en paralelo, siempre que los datos solicitados residan en discos separados. Es más, una única petición de E/S se puede ejecutar en paralelo si el bloque de datos que se pretende acceder está distribuido a lo largo de múltiples discos. Por lo que RAID tiende a mejorar significativamente el rendimiento y fiabilidad del sistema de E/S a disco.

El sistema de RAID consta de 6 niveles posibles, de 0 a 6. Los niveles de 2 a 4 no se comercializan.

RAID 0

El RAID 0 (no redundante). No incluye redundancia. Es un RAID “truco”. Todos los datos están distribuidos a lo largo de todo el vector de discos.

Los datos están distribuidos en bandas a lo largo de los discos disponibles. El RAID 0 se usa normalmente para aplicaciones que requieren alto rendimiento para datos no críticos.

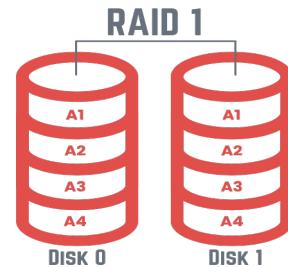


(a) RAID 0 (sin redundancia)

RAID 1

El **RAID 1 (espejo)** se consigue redundancia por el simple medio de duplicar todos los datos. **Mucha fiabilidad. Permite lecturas más rápidas** ya que los datos los puede servir cualquiera de los 2 discos que contienen los datos solicitados. La velocidad de escritura depende de la escritura más lenta de las 2, **pero NO hay penalización de escritura** (escribir información adicional como bit de paridad). La principal desventaja es el elevado costo por tener que duplicar toda la información en otro disco.

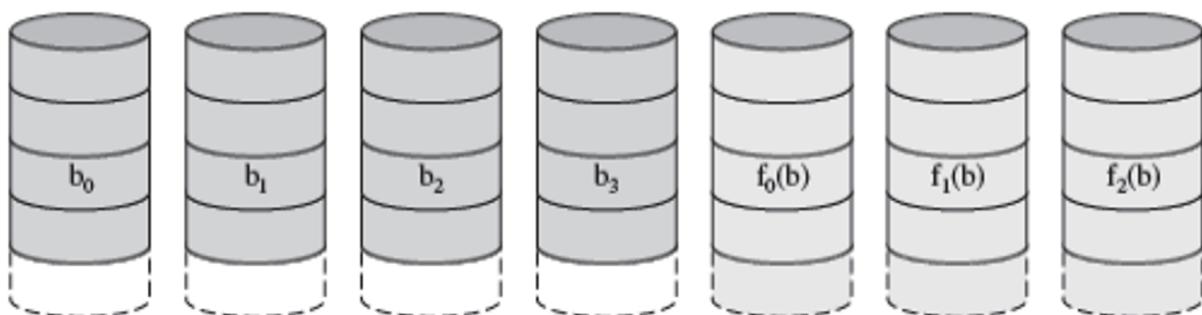
Discos = 2N



RAID 2

El **RAID 2 (redundancia mediante código Hamming - Acceso paralelo)** usa división a nivel de bits. Se utilizan un par de discos para datos y otros para paridad. El número de discos redundantes es proporcional al logaritmo del número de discos de datos.

Discos = N + M



(c) RAID 2 (redundancia mediante código Hamming)

Uno de sus efectos secundarios es que normalmente **no puede atender varias peticiones simultáneas**, debido a que por definición cualquier simple bloque de datos se dividirá por todos los miembros del conjunto, residiendo la misma dirección dentro de cada uno de ellos. Así, cualquier operación de lectura o escritura exige activar todos los discos del conjunto.

Puede conseguir una tasa de transferencia muy alta.

Sirve solamente para un entorno donde se produjeran **muchos errores de disco**.

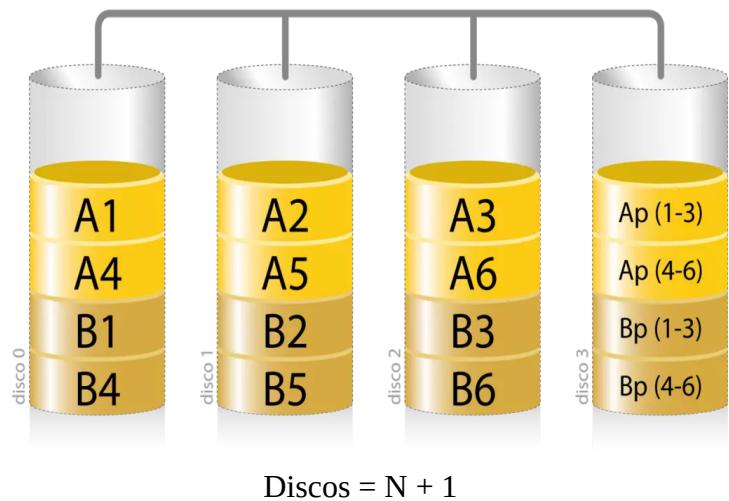
RAID 3

El **RAID 3 (paridad por intercalación de bits - Acceso paralelo)** es idéntico a RAID 2 pero solamente requiere de 1 disco redundante, sin importar la cantidad de discos. En lugar de un código de corrección de errores, se calcula un bit de paridad simple para el conjunto de bits almacenados en la misma posición en todos los discos de datos.

En el ejemplo del gráfico, una petición del bloque «A» formado por los bytes A1 a A6 requeriría que los tres discos de datos buscaran el comienzo (A1) y devolvieran su contenido. Una petición simultánea del bloque «B» tendría que esperar a que la anterior concluyese. Puede conseguir una tasa de transferencia muy alta. No puede atender varias peticiones simultáneas.

Diagrama de configuración RAID 3, Cada nro representa un byte de datos; cada columna, un disco.

RAID 3



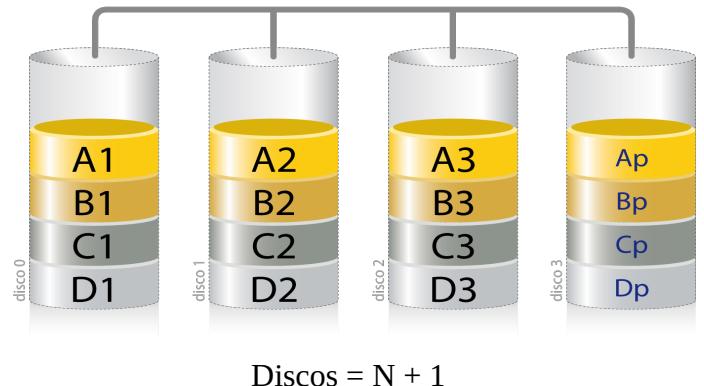
$$\text{Discos} = N + 1$$

RAID 4

El **RAID 4 (paridad por intercalación de bloques - Acceso Independiente)** Usa división a nivel de bloques con un disco de paridad dedicado. Permite acceso independiente.

En ejemplo del gráfico, una petición del bloque «A1» sería servida por el disco 0. Una petición simultánea del bloque «B1» tendría que esperar, pero una petición de «B2» podría atenderse concurrentemente.

RAID 4

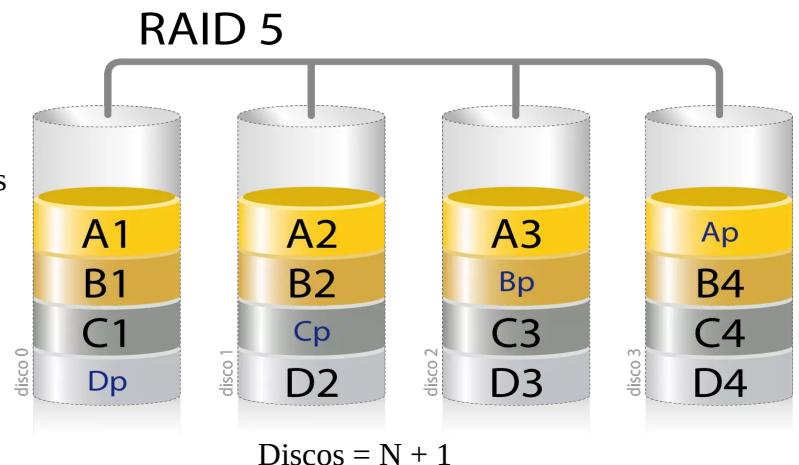


$$\text{Discos} = N + 1$$

RAID 5

El **RAID 5 (paridad por intercalación distribuida de bloques - Acceso independiente)** se organiza de manera similar al RAID 4. La diferencia estriba en que el esquema RAID 5 distribuye las bandas de paridad a través de todos los discos.

El RAID 5 ha logrado popularidad gracias a su bajo coste de redundancia. Además a diferencia de RAID 4 no tiene cuello de botella al no tener un disco dedicado a redundancia.



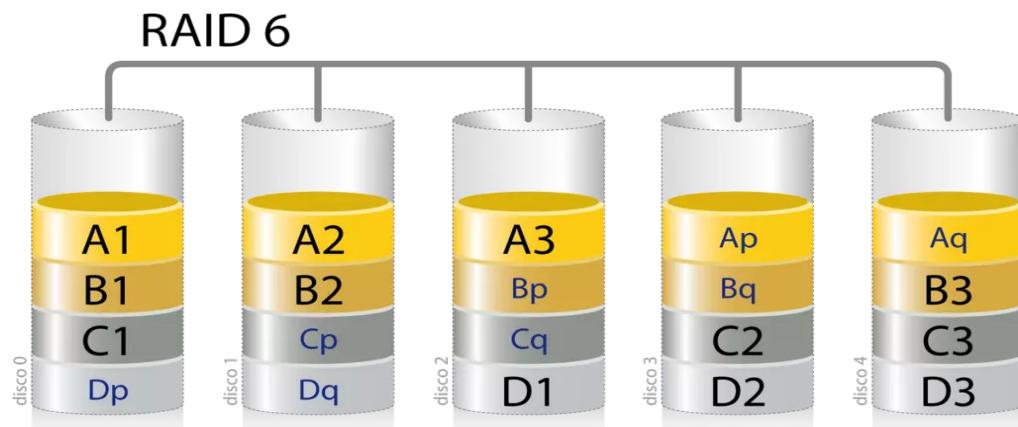
RAID 6

En el **RAID 6 (paridad dual por intercalación distribuida de bloques - Acceso independiente)** se realizan dos cálculos de paridad diferentes, almacenándose en bloques separados de distintos discos. Por tanto, un vector RAID 6, cuyos datos de usuario requieran N discos, necesitará:

$$\text{Discos} = \mathbf{N + 2}$$

La ventaja del esquema RAID 6 es que proporciona una extremadamente alta disponibilidad de datos.

- **Acceso paralelo** → todos los discos participan en la ejecución de cada solicitud de E/S. Más apropiados para aplicaciones que requieran tasas altas de transferencia de datos.
- **Acceso independiente** → cada disco opera independientemente, por lo que se pueden satisfacer en paralelo solicitudes de E/S. Más apropiados para aplicaciones que requieran tasas altas de solicitudes de E/S.



¿Qué es la memoria cache?

Memoria **pequeña** y de **alta velocidad**, más rápida que la memoria principal, y que se sitúa entre esta y el procesador. Dicha memoria cache reduce el tiempo medio de acceso a memoria aprovechándose del principio de la proximidad.

La **cache de disco** contiene una copia de algunos de los sectores del disco. Cuando se hace una petición de E/S solicitando un determinado sector, se comprueba si el sector está en la cache del disco. En caso afirmativo, se sirve la petición desde la cache. Debido al fenómeno de la proximidad de referencias, cuando se lee un bloque de datos en la cache para satisfacer una única petición de E/S, es probable que haya referencias a ese mismo bloque en el futuro.

La **diferencia entre cache y buffer** en que el buffer puede almacenar la única copia existente de un elemento de datos, mientras que la cache almacena una copia de un elemento que reside en otro lugar.

¿Cuáles son los tipos de E/S?

- **Bloqueante y No bloqueante:**

- **Bloqueante** → Se suspende la ejecución de la aplicación que la solicita. Cuando se completa la operación vuelve a la cola de listos.
- **No bloqueante** → No detiene la ejecución. En lugar de ello, la llamada vuelve rápidamente, con un valor de retorno que indica cuantos bytes se han trasferidos.

- **Asincrónicas / Sincrónicas:**

- **Asincrónico** → La llamada vuelve rápidamente, sin ningún valor de retorno. Cuando la E/S termina avisa (consume una cantidad impredecible de tiempo). La diferencia con una no bloqueante, es que una operación read () no bloqueante vuelve inmediatamente con los datos que haya disponibles, mientras que una llamada read () asincrónica solicita una transferencia que se realizará de modo completo pero que completara en algún instante futuro.
- **Sincrónico** → Como voy a seguir con el proceso depende del resultado de E/S (consume una cantidad predecible de tiempo)

SISTEMA DE ARCHIVOS

¿Qué es un archivo?

Un archivo es una **colección de información relacionada**, con un nombre, que se graba en almacenamiento relacionado. Es la unidad lógica más pequeña de almacenamiento secundario (no pueden escribirse datos en el almacenamiento secundario a menos que estos se encuentren dentro de un archivo).

Los atributos de un archivo generalmente son:

- **Nombre** → identifica el archivo para el usuario
- **Identificador** → identifica el archivo dentro del sistema de archivos
- **Tipo**
- **Ubicacion**
- **Tamaño**
- **Proteccion** → informacion de control de acceso al archivo (quien puede leer, escribir, etc.)
- **Fecha, hora e identificacion del usuario**

La información de los archivos se almacena en la estructura de directorios, que también reside en almacenamiento secundario.

¿Qué es el sistema de gestión de archivos?

Un sistema de gestión de archivos es aquel conjunto de software del sistema que proporciona servicios a los usuarios y aplicaciones para el uso de archivos. Los objetivos de un sistema de gestión de archivos son:

- Almacenar datos y operar con ellos
- Proteccion de los datos
- Optimizar la performance del sistema
- Minimizar la posibilidad de perdida de datos
- Brindar soporte para distintos tipos de almacenamiento
- Interface personalizada para aplicaciones de usuarios
- Garantizar que sea posible la coherencia de los datos

¿Cuál es la arquitectura de los sistemas de archivos?

La figura muestra una representación de una organización típica del software (puede variar dependiendo del sistema). Por lo tanto, el sistema de archivos está compuesto, generalmente, de muchos niveles. Cada nivel del diseño utiliza funciones de los niveles inferiores. Desde el nivel más bajo al más alto:

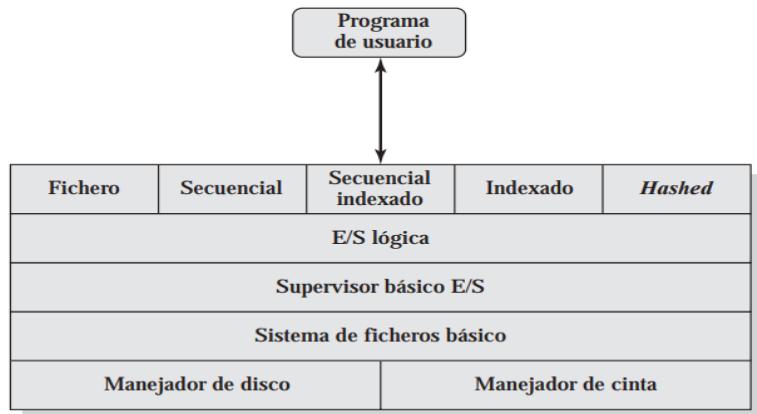
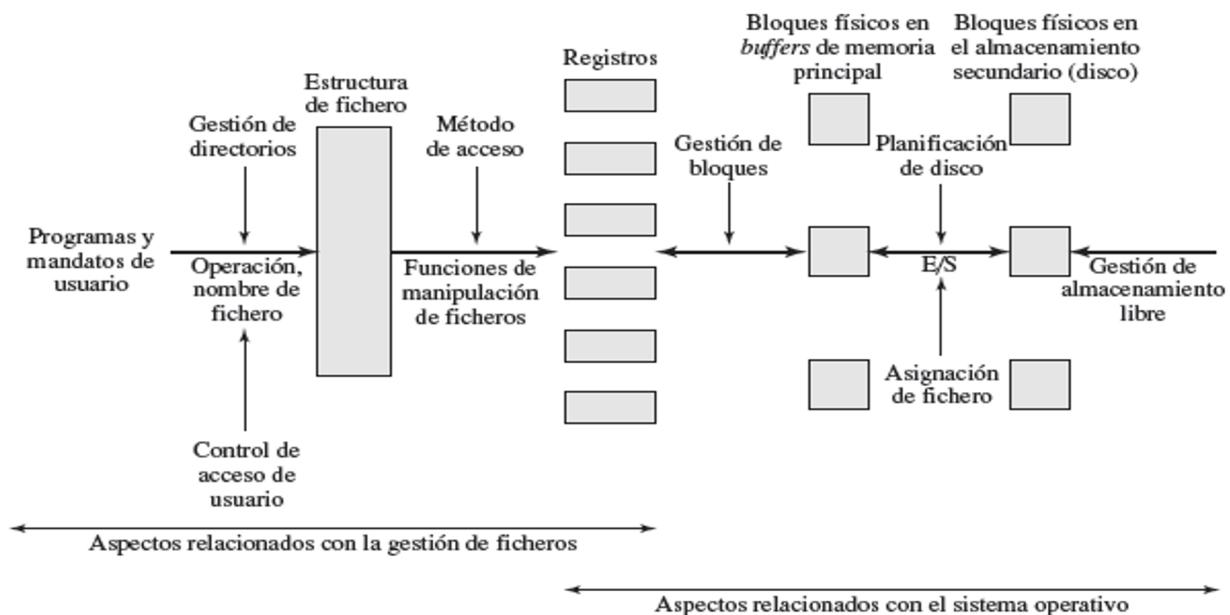


Figura 12.1. Arquitectura software de un sistema de ficheros.

- **Controladores de dispositivo** → Un controlador de dispositivo es el responsable de iniciar las operaciones de E/S de un dispositivo y procesar la finalización de una petición de E/S. Los controladores de dispositivos se consideran normalmente parte del sistema operativo.
- **Sistema de archivos básico** → envía comandos genéricos al controlador de dispositivo apropiado, con el fin de leer y escribir en bloques físicos en el disco. El sistema de archivos básico se considera normalmente parte del sistema operativo.
- **Supervisor básico de E/S** → El supervisor de E/S básico selecciona el dispositivo en el cual se van a llevar a cabo la operación de E/S. También se encarga de la planificación de disco y cinta para optimizar el rendimiento. A este nivel, se asignan los buffers de E/S y se reserva la memoria secundaria. El supervisor de E/S básico es parte del sistema operativo.
- **E/S Lógica** → Permite a los usuarios y a las aplicaciones acceder a los registros.
- **Método de acceso** → Diferentes métodos de acceso reflejan diferentes estructuras de archivos y diferentes formas de acceder y procesar los datos.

Otra forma de ver las funciones de un sistema de archivos se muestra en la siguiente figura:

Sigamos este diagrama de izquierda a derecha:



Los usuarios y programas de aplicaciones interaccionan con el sistema de archivos mediante órdenes/operaciones sobre archivos.

El sistema de archivos debe identificar y localizar el archivo seleccionado. Esto requiere el uso de algún tipo de **directorio** que se utilice para describir la ubicación de todos los archivos.

Adicionalmente se aplica un control de acceso (sólo se permite determinado acceso particular a los usuarios autorizados.)

Las operaciones básicas que puede realizar un usuario o aplicación se realizan a nivel de registro. El usuario o aplicación ve el archivo como una estructura que organiza los registros. Por tanto, para traducir los órdenes de usuario en órdenes de manipulación de archivos, debe emplearse el método de acceso apropiado para esta estructura de archivos.

La E/S se realiza a nivel de bloque. Por tanto, los registros de un archivo se deben traducirse en bloques para la salida y los bloques traducirse a registros después de la entrada.

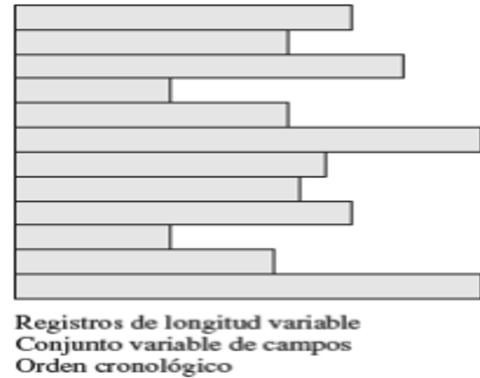
Se debe gestionar el almacenamiento secundario Asignación de archivos a bloques libres de almacenamiento secundario, gestión del espacio libre, planificación de las peticiones de E/S.

En sí, la figura sugiere una división entre las funciones del sistema de gestión de archivos y las funciones del sistema operativo, siendo el punto de intersección el procesamiento de registro

¿Cómo se da la organización y acceso a archivos?

Las organizaciones son: archivo de pila, archivo secuencial, archivo secuencial indexado, archivo indexado, archivos **hash**.

- **Pilas** → Forma más simple de organización de archivos. Los datos se recolectan en el orden en que llegan. Cada registro consiste en una ráfaga de datos. El propósito de la pila es simplemente acumular la masa de datos y guardarlos. El acceso a los datos se hace por búsqueda exhaustiva por no tener una estructura. Fáciles de actualizar. Aprovecha bien el espacio cuando los datos almacenados varían en tamaño y estructura.



- **Archivos secuenciales** → Se emplea un formato fijo para los registros. Todos los registros son de igual tamaño y están compuestos por el mismo número de campos de longitud fija en un orden específico. Las búsquedas se realizan en forma secuencial.



- **Archivos secuenciales indexados** → Solventa las desventajas de los archivos secuenciales. Mantienen las características básicas de los archivos secuenciales. Los registros se organizan en una secuencia basada en un campo clave, pero se añaden 2 características nuevas: un índice del archivo para soportar los accesos aleatorios y un archivo de desbordamiento (overflow). Reducen enormemente el tiempo necesario para acceder a un solo registro sin sacrificar la naturaleza secuencial del archivo.
- **Archivos indexados** → Cuando es necesario buscar por algún otro atributo que no sea el campo clave, ambas formas de archivos secuenciales son inadecuadas. En algunas aplicaciones, esta flexibilidad es deseable. Para lograr esta flexibilidad, se necesita una estructura que emplea múltiples índices, uno por cada tipo de campo que puede estar sujeto a una búsqueda (índice exhaustivo y parcial).
- **Archivos directos o hash** → Los archivos hasheados o directos, hacen uso de la capacidad del disco para acceder directamente a un bloque. Se requiere una clave para cada registro. Sin embargo, en este tipo de archivos no existe el concepto de ordenación secuencial.

El archivo directo hace uso de una función hash sobre un valor clave. Los archivos directos se utilizan frecuentemente cuando se requiere un acceso muy rápido (ejemplo: tabla de precios).

¿Cómo se da la organización de directorios?

El directorio de archivos contiene información sobre los archivos, incluyendo atributos, ubicación y propietario. Gran parte de esta información la gestiona el SO. El directorio es propiamente un archivo, poseído por el sistema operativo.

La información que almacena difiere mucho entre los distintos sistemas, pero algunos elementos claves siempre deben permanecer en el directorio, como el nombre, dirección, tamaño y organización.

Los tipos de operaciones que pueden realizarse con un directorio:

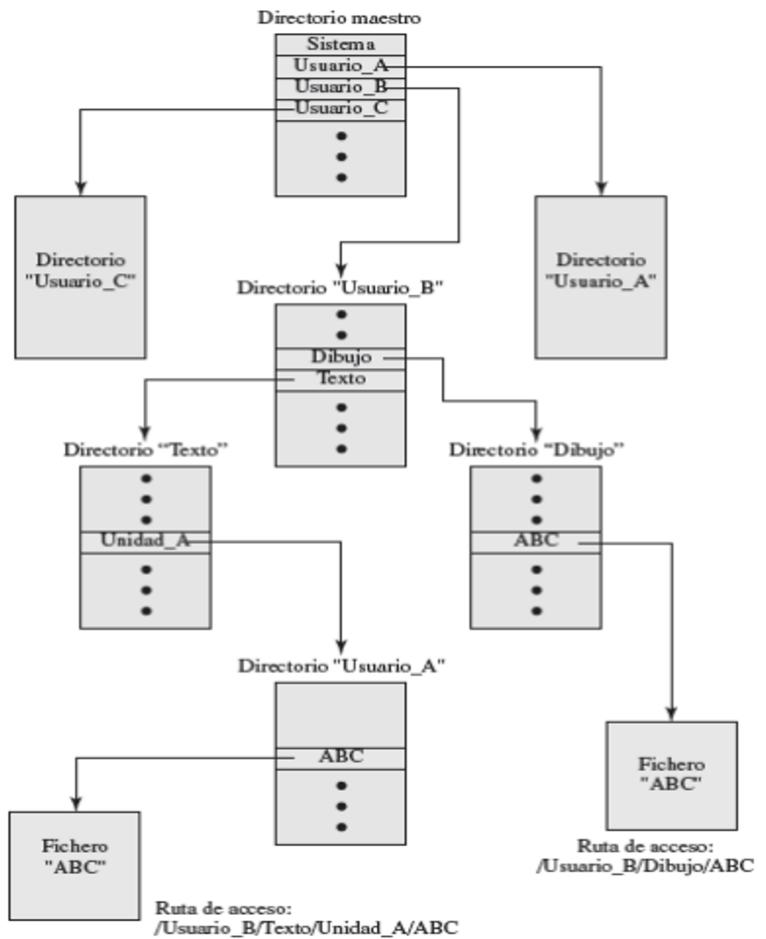
- **Buscar archivo** → cuando un usuario hace referencia a un archivo debe buscarse en el directorio la entrada correspondiente.
- **Crear archivo** → se añade una entrada al directorio.
- **Borrar archivo** → se elimina una entrada del directorio
- **Enumerar archivo** → una lista de todos los archivos poseídos por dicho usuario.
- **Actualizar directorio** → cambio en la entrada del directorio correspondiente por un cambio de algún atributo del archivo.

Una única lista no es adecuada como estructura del directorio para soportar estas operaciones por cuestiones de organización (por ejemplo tener los archivos separados por tipo, por usuario, etc.). Una técnica más potente y flexible, que es casi universalmente adoptada, es utilizar una estructura jerárquica en forma de árbol.

Como en la técnica anterior, hay un directorio maestro, que tiene bajo dicho directorio varios directorios de usuario. Cada uno de estos directorios de usuario, a su vez, podría tener subdirectorios y archivos como entradas. Esto se cumple para todos los niveles. Es decir, en cada nivel, un directorio podría estar formado por subdirectorios y/o archivos.

Cada directorio se almacena como un archivo secuencial. Cuando los directorios tengan un número muy grande de entradas es preferible una estructura hash.

El conjunto de nombres de directorios, finalizando en el **nombre del archivo**, constituye un nombre de camino para el archivo.



¿Cómo se da la agrupación de registros?

Como se vio los registros son la unidad lógica de acceso a los archivos, mientras que los bloques son la unidad de E/S para almacenamiento secundario. Para que la E/S se pueda realizar, los registros se deben organizar como bloques.

- Los bloques en la mayoría de los sistemas son de **longitud fija**. Esto simplifica la E/S.
- Cuanto **mayor** sea el bloque, más registros se transferirán en una operación de E/S. La preocupación viene por el hecho de que los bloques más grandes requieren buffers de E/S mayores, haciendo la gestión de buffers más difícil.
- **Los bloques pueden ser de:**
 - **Tamaño fijo:** Se utilizan registros de longitud fija, guardándose en cada bloque un número entero de registros. Podría haber espacio no utilizado al final de cada bloque. Esto se denomina fragmentación interna.
 - **Bloques de longitud variable con tramos:** Se utilizan registros de longitud variable y se agrupan en bloques sin dejar espacio sin usar. Por tanto, algunos registros deben abarcar dos bloques, con su continuación indicada por un puntero al bloque sucesor
 - **Bloques de longitud variable sin tramos:** Se utilizan registros de longitud variable, pero no se dividen en tramos. En la mayoría de los bloques habrá espacio desperdiciado si el siguiente registro es mayor que el espacio sin usar.

Utilizar bloques fijos es el modo común para archivos secuenciales con registros de longitud fija.

Los **Bloques de longitud variable con tramos** son difíciles de implementar ya que los registros que ocupan 2 bloques requieren dos operaciones de E/S y los archivos son difíciles de actualizar.

Los **Bloques de longitud variable sin tramos** implican espacio malgastado y limitan el tamaño del registro al tamaño de un bloque.

¿Cómo se realizan las transferencias de E/S entre memoria y disco?

En esta segunda parte, se verá los temas relativos al almacenamiento de archivos y al acceso a archivos en el medio más común de almacenamiento secundario, que es el disco.

Las transferencias de E/S entre la memoria y el disco se realizan en unidades de bloques para mejorar la eficiencia de E/S. Cada bloque tiene uno o más sectores. Los sectores varían entre 32 bytes y 4096 bytes; generalmente su tamaño de 512 bytes.

Bloque de control de archivo

(FCB) → Estructura que contiene información acerca de un archivo específico, incluyendo su propietario, los permisos y la ubicación del contenido del archivo. Hay un FCB por archivo.

Cuando se crea un archivo se asigna un nuevo **FCB**.

Permisos del archivo
Fechas del archivo (creación, accesos, escritura)
Propietario del archivo
Tamaño del archivo
Punteros a los bloques de datos del archivo

- **Partición** → división lógica de un dispositivo físico
- **Volumen** → partición formateada con un sistema de archivos
- **Tabla de asignación de archivos (FAT)** → Estructura de datos que se usa para guardar constancia de las secciones asignadas a un archivo.

¿Cómo se asignan espacios de memoria secundaria a los archivos?

El principal problema de un sistema de archivos es como asignar el espacio a los archivos de modo que el espacio de disco se utilice de forma eficaz y que se pueda acceder a los archivos de forma rápida. Hay 3 métodos: asignación contigua, asignación enlazada y asignación indexada.

Tipos de asignación

- **Asignación previa** → Requiere que el tamaño máximo del archivo sea declarado en el momento de crearlo. Es difícil, a veces imposible, estimar el tamaño máximo del archivo. Por lo que habría que sobreestimar el tamaño de lo que supone un desperdicio de la asignación de espacio de almacenamiento.
- **Asignación dinámica** → Asigna espacio a un archivo en secciones a medida que se necesitan.

Tamaño de sección

- **Secciones contiguas variables y grandes** → El mejor rendimiento. Evita malgastar espacio, y las tablas de asignación de archivos serán pequeñas. Espacio difícil de reutilizar.
- **Bloques** → Secciones fijas y pequeñas ofrecen mayor flexibilidad. Tablas de asignación serán grandes y/o complejas. Los bloques se asignan a medida que se necesiten (pueden no ser contiguos).

En el caso de secciones contiguas variables y grandes se asigna previamente un grupo de bloques contiguos.

Esto elimina la necesidad de una tabla de asignación de archivos; todo lo que se requiere es un puntero al primer bloque y el número de bloques asignados.

En el caso de los bloques, todas las secciones requeridas se asignan a la vez. Esto significa que la tabla designación de archivos para el archivo es de tamaño fijo.

Con secciones de tamaño variable, es necesario preocuparse de la fragmentación del espacio libre. Las siguientes estrategias son posibles:

- **Primer ajuste:** elegir el primer grupo de bloques contiguo sin usar de tamaño suficiente.
- **Siguiente ajuste:** elegir el grupo mas pequeño sin que sea de tamaño suficiente.
- **Ajuste mas cercano:** Elegir el grupo sin usar de tamaño suficiente que está más cercano al asignado previamente al archivo para aumentar la cercanía.

¿En que consiste la asignación contigua?

	Contiguos	Encadenado	Indexado	
¿Preasignación?	Necesaria	Possible	Possible	
¿Porciones de tamaño fijo o variable?	Variable	Bloques fijos	Bloques fijos	Variable
Tamaño de porción	Grande	Pequeño	Pequeño	Medio
Frecuencia de asignación	Una vez	Pequeña a alta	Alta	Baja
Tiempo a asignar	Medio	Largo	Corto	Medio
Tamaño de tabla de asignación de ficheros	Una entrada	Una entrada	Grande	Medio

Asignación contigua → Cuando se crea el archivo se asigna un único conjunto contiguo de bloques. Es una estrategia de asignación previa. La FAT contiene sólo una entrada por cada archivo y que muestra el bloque de comienzo y la cantidad de bloques que ocupa.

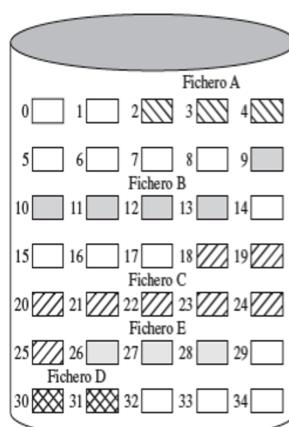


Tabla de asignación de ficheros

Nombre de fichero	Bloque inicial	Longitud
Fichero A	2	3
Fichero B	9	5
Fichero C	18	8
Fichero D	30	2
Fichero E	26	3

Ventajas:

- Requiere menos reposicionamientos del cabezal para escritura y lectura al ser posiciones contiguas.
- Eficiente para acceso directo y secuencial.
- Si se daña un bloque NO pierdo al resto.

Desventajas:

- Fragmentación externa a causa de la asignación previa. A medida que se asignan y borran archivos, el espacio libre del disco se descompone en pequeños fragmentos.
- Sufre demasiada fragmentación interna a causa de la asignación previa.
- Poco óptimo para expandir archivos a causa de la asignación previa.

¿En que consiste la asignación enlazada?

Asigna bloques dinámicamente a medida que se necesiten (aunque es posible la asignación previa). Para poder seguir la cadena de bloques de un archivo, cada bloque contendrá un puntero al siguiente bloque de la cadena. La tabla de asignación de archivos necesita una sola entrada por cada archivo que muestre el bloque de comienzo y la longitud del archivo.

Ventajas:

- No tiene fragmentación externa al asignar un bloque cada vez y pudiendo ser de forma no continua (asignación dinámica).
- Sufre en menor medida la fragmentación interna en comparación a la asignación contigua.
- Óptimo para expandir el archivo.

Desventajas:

- Requiere muchos reposicionamientos del cabezal para escritura y lectura al ser posiciones no contiguas (muy dispersas en el disco).
- No es eficiente para acceso directo. Se debe recorrer la cadena hasta el bloque deseado.
- Si se daña un bloque pierdo al resto.

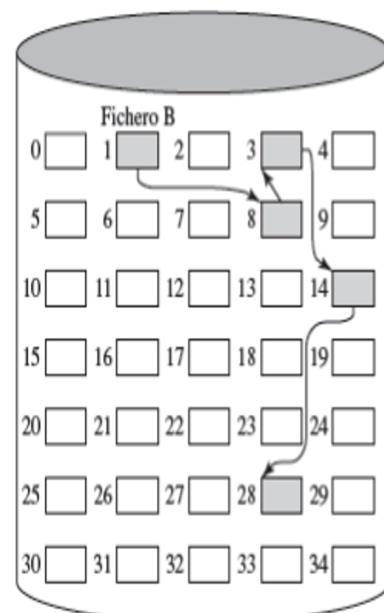


Tabla de asignación de ficheros

Nombre de fichero	Bloque inicial	Largo
•••	•••	•••
Fichero B	1	5
•••	•••	•••

¿En qué consiste la asignación indexada?

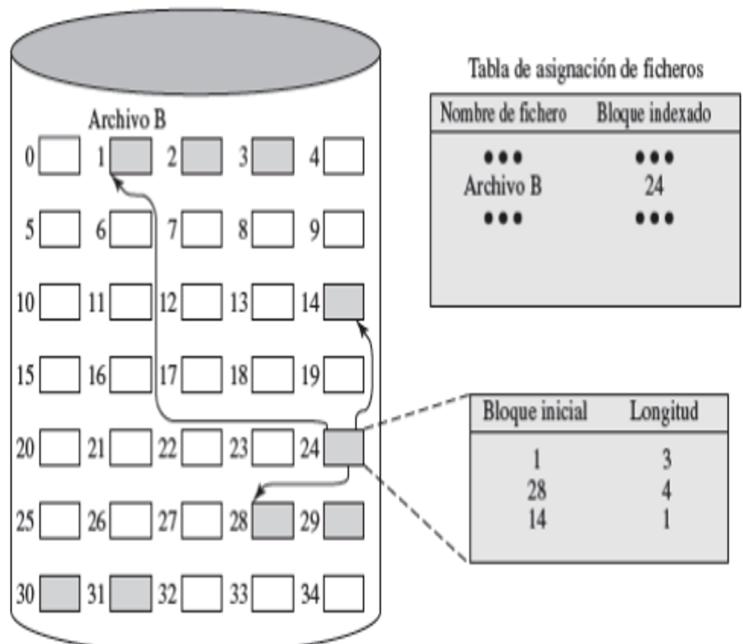
La tabla de asignación de archivos **contiene un índice separado de un nivel para cada archivo**. El índice posee una entrada para cada sección asignada al archivo. El índice del archivo se guardará en un bloque aparte y la entrada del archivo en la tabla de asignación apuntará a dicho bloque.

Ventajas:

- No tiene fragmentación externa al asignar un bloque cada vez y pudiendo ser de forma no continua (asignación dinámica).
- Sufre en menor medida la fragmentación internada en comparación a la asignación contigua.
- Eficiente para acceso directo y secuencial.
- Si se daña un bloque NO pierdo al resto.

Desventajas:

- Requiere muchos reposicionamientos del cabezal para escritura y lectura al ser posiciones no contiguas (muy dispersas en el disco).
- Requiere de espacio para almacenar el índice.
- Requiere mantener actualizado el índice.



¿Cómo se da la gestión del espacio libre? Que son las tablas de bits?

Para llevar a cabo cualquiera de las técnicas de asignación de archivos descritas previamente, es necesario saber qué bloques del disco están disponibles. Por tanto se necesita una tabla de asignación de disco. Tres técnicas son de uso común: las tablas de bits, las secciones libres encadenadas y la indexación.

Tablas de bits

Utiliza un vector que está formado por un bit por cada bloque en el disco. Cada entrada 0 corresponde a un bloque libre y cada 1 corresponde a un bloque en uso.

La cantidad de memoria (en bytes) requerida para un mapa de bits en bloques se puede calcular de la siguiente manera: Tamaño del disco en bytes / (8 x tamaño del bloque en el sistema de archivos).

- **Ventajas**

- Fácil de encontrar un bloque libre o un grupo contiguo de bloques libres.
- Trabaja bien con cualquier método de asignación de archivos.

- **Desventajas**

- Rendimiento inaceptable cuando la tabla de bits es grande. La tabla de bits puede ser grande como para almacenarla en memoria principal y una tabla en disco no es eficiente porque requiere E/S cada vez que se necesite un bloque (parte de la ineficiencia se puede reducir manteniendo estructuras auxiliares que resumen el contenido de subrangos de la tabla de bits).

¿Qué son las secciones libres encadenadas?

Las secciones libres pueden encadenarse mediante un puntero y un valor de longitud en cada sección libre.

- **Ventajas:**

- Sobrecarga de espacio insignificante. No se necesita una tabla de asignación de disco, sino simplemente un puntero al comienzo de la cadena y la longitud de la primera sección.
- Trabaja bien con cualquier método de asignación de archivos.

- **Desventajas:**

- Disco fragmentado después de un tiempo de uso.
- Se ralentiza la creación del archivo al tener que actualizar los punteros del bloque.

¿Qué es la indexación?

La técnica de indexación trata el espacio libre como un archivo y utiliza una tabla de índices tal y como se describió en la asignación de archivos. Por motivos de eficiencia, el índice se debería utilizar en base a secciones de tamaño variable en lugar de bloques. Por tanto, hay una entrada en la tabla por cada sección libre en el disco. Esta técnica proporciona soporte eficiente a todos los métodos de asignación de archivos.

Unidad 6: Virtualización

TEMA: Introducción a la virtualización.

¿Qué es la virtualización?

Es una **capa de abstracción** (traducción) entre el software y el hardware físico.

Permite que el hardware pueda ejecutar varias instancias de SO diferentes.

Teniendo en cuenta el sistema de capas, en un **sistema tradicional** encontramos las siguientes capas:

- Una plataforma de hardware
- Una capa de SO, quien administra y hace interfaz con respecto al resto y al hardware
- Librerías, para garantizar que las aplicaciones utilicen ciertas funciones
- Aplicaciones



Ahora, cuando hablamos de **visualización**,

tenemos las siguientes capas:

- Se tiene la misma plataforma de hardware
- Ya no se tiene un SO de base, si no un software de virtualización, generalmente conocido como **Hypervisor**. Esta es la capa encargada de realizar la abstracción mencionada anteriormente.

Podrían tenerse dos SO distintos con sus librerías y aplicaciones respectivas.

¿Por qué se usa la virtualización?

- **Uso de hardware heredado** → siendo que provee compatibilidad de sistemas de hardware con arquitecturas no compatibles con los nuevos servicios de software o sistemas operativos.
- **Implementación rápida**
- **Versatilidad** → puedo tener múltiples máquinas virtuales trabajando
- **Consolidación y Agregación** → se refiere a mantener la eficiencia de equipos (una máquina virtual fácilmente podría reemplazar a 10 equipos)
- **Dinámica** → puedo cambiar sus parámetros dinámicamente en base a mis necesidades (puedo subir la cantidad de memoria que utiliza fácilmente a partir de la configuración)
- **Facilidad de administración**
- **Mayor disponibilidad**

¿Cuáles son los enfoques de la virtualización?

- **Emulacion** → es decir, permite inclusive emular otras arquitecturas diferentes a la del host. Por ejemplo, otras arquitecturas de procesador u otros sistemas de hardware.
- **Virtualizacion total** → traducción bit a bit. Misma arquitectura del host. Es decir, si tengo una maquina host de 32 bits, no puedo emular una arquitectura de 64 bits.
- **Paravirtualizacion** → llamadas a una API en vez del hardware. Para que el SO guest sepa que esta siendo virtualizado y sus instrucciones sean pasadas al Hypervisor, excepto de aquellas instrucciones privilegiadas que se pasan a la API.
- **Containers** → virtualizacion sobre SO.
- **Soporte en Hardware** → cambio de paradigma, replantea la virtualizacion total.

¿Cuáles son los desafíos de un VMM?

Un **VMM (Virtual Machine Monitor) o Hypervisor** tiene tres desafíos básicos a resolver para poder virtualizar arquitecturas x86:

- **La administración de las instrucciones que se ejecutan en el CPU** → instrucciones que son privilegiadas de las que no son.
- **La administración de la memoria (MMU)** → El hypervisor debe administrar que sectores de memoria le va a dar a cada memoria virtual.
- **El acceso I/O al hardware virtual** → por ejemplo, como comunica la placa de red con la instancia virtual.

¿Qué funciones tiene un Hypervisor?

- **Administración** de ejecución de maquinas virtuales
- **Emulación de dispositivos y control de acceso**, es decir, se fija como mapear las instrucciones de la maquina virtual en los dispositivos físicos.
- **Ejecución de operaciones privilegiadas** por hipervisor para maquinas virtuales invitadas
- **Gestión de ciclo** de vida de las maquinas virtuales (como iniciarla, pausarla, detenerla, apagarla)
- **Administración de la plataforma del hipervisor y el software del hipervisor.**

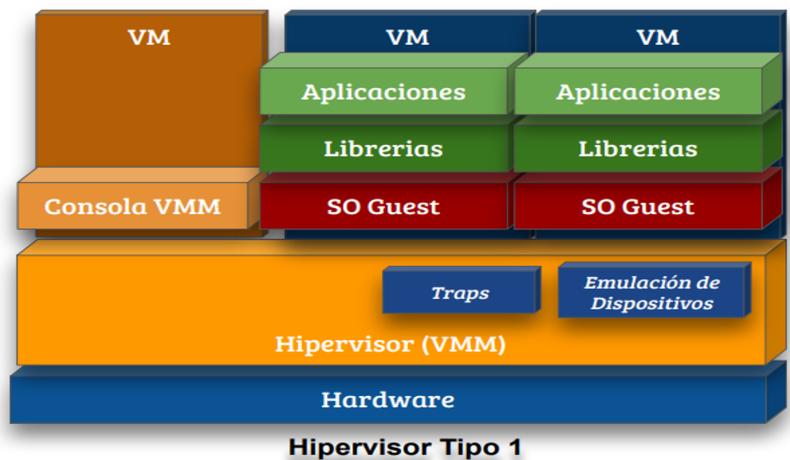
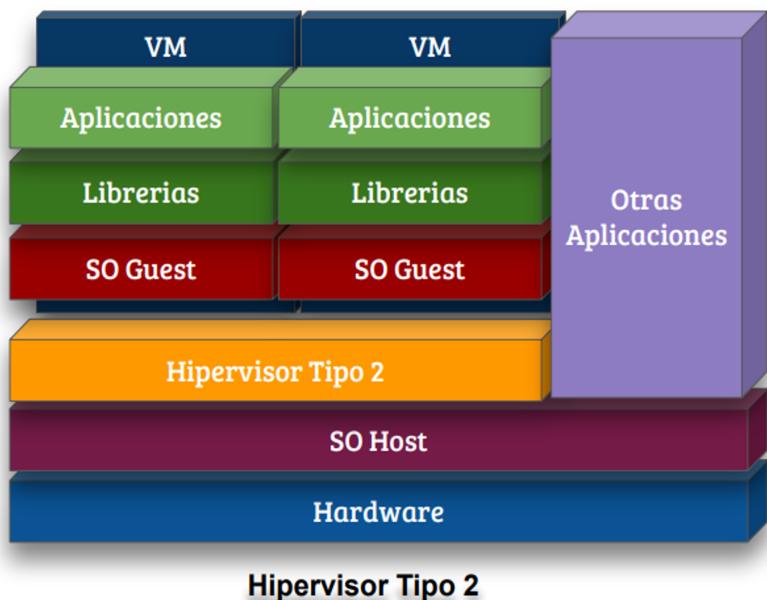
¿Cómo se clasifican los Hypervisores?

Hay dos tipos:

- **Virtualización por software sobre SO (Tipo 2)** → siendo el más usado en el año, en este, el Software de Virtualización realiza la tarea de emular el hardware virtual y la técnica de traducción bit a bit para realizar una virtualización total. El software de virtualización y sus VM's compiten con otras aplicaciones.

Tenemos una base de hardware y un SO Host, por encima el Hipervisor y cada SO Guest respectivo dentro de una máquina virtual (VM).

- **Virtualización sobre Hypervisor (VMM) (Tipo 1)**
 - El Hypervisor realiza la tarea de emular el hardware virtual y la técnica de traducción bit a bit para realizar una virtualización total. Al no competir con un SO por el uso y administración del hardware mejora el rendimiento.



¿Qué es la Paravirtualización?

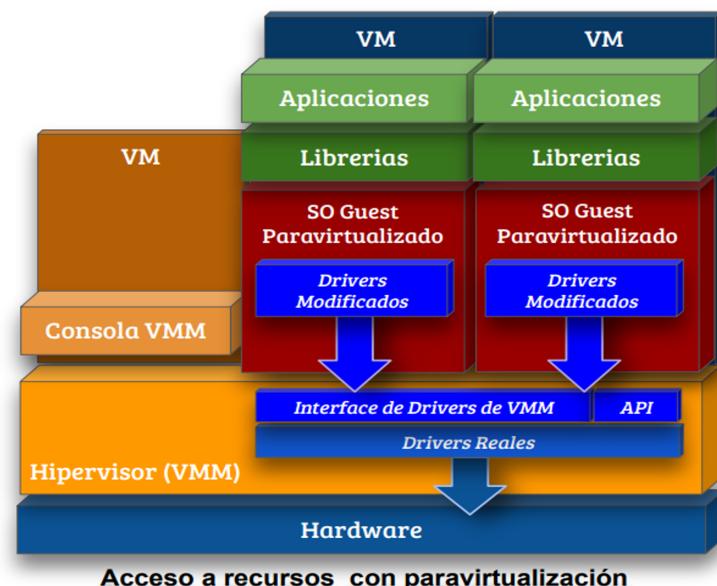
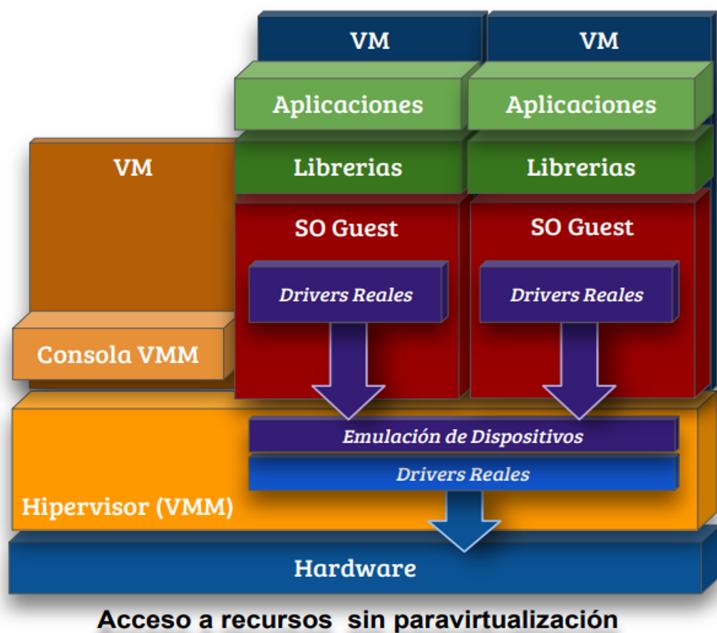
Cada SO Guest tiene drivers reales, que le manda a la capa de emulación de dispositivos del Hipervisor, **la cual traduce las instrucciones**, pasando los datos a los drivers reales que van al hardware.

El Hypervisor tiene una API, que realiza la traducción de las instrucciones privilegiadas a un conjunto de instrucciones que simulan lo mismo en el entorno virtualizado.

El SO Guest, “sabe” que está siendo virtualizado y realiza una **Hypercall** a la API del Hypervisor en vez de ejecutar una instrucción privilegiada.

El resto de las instrucciones se ejecutan directamente sobre el hardware.

Entonces, un SO Guest paravirtualizado, envía unos drivers modificados a la interface de drivers de la máquina virtual, teniendo menos carga, y pasandolo a los drivers reales. Además tenemos la API a la cual el SO Guest hace las Hypercalls.



CONTENEDORES

¿Que es un Container?

Los **contenedores** son unidades ejecutables de software en los que el código de aplicación se empaqueta, junto con sus bibliotecas y dependencias, de forma común para que pueda ejecutarse en cualquier lugar, ya sea en el desktop, en la TI tradicional o en la nube.

Para ello, los contenedores utilizan una forma de **virtualización del sistema operativo (SO)** en la que las características del SO (específicamente los espacios de nombres y las primitivas de cgroups en el caso del kernel de Linux) se aprovechan al máximo para aislar procesos y controlar la cantidad de CPU, memoria y disco a los que dichos procesos tienen acceso.

Los contenedores son pequeños, rápidos y portátiles porque, a diferencia de una máquina virtual, los contenedores no necesitan incluir un SO invitado en todos los casos y, en su lugar, pueden simplemente hacer uso de las características y recursos del sistema operativo host.

¿Que características posee un Container?

- No tiene como objetivo emular servidores físicos
- En líneas generales, existen múltiples instancias de sistema distintas, que comparten una misma instancia del kernel del sistema operativo que se ejecuta en el host.
- Cada contenedor se ejecuta sobre el núcleo del sistema operativo host y proporciona un entorno de ejecución aislado para las aplicaciones.
- Esto resulta en que la capa de virtualización es mucho más liviana, por lo tanto soportan muchas más instancias virtualizadas.
- Se basa en Linux cgroups (control groups), que proporciona:
 - Limitaciones de recursos
 - Priorización → si por ejemplo tengo una base de datos entre otras cosas, puedo priorizar la ejecución de la BD dentro del kernel.
 - Contabilidad → saber cuantos recursos estoy utilizando dentro de los containers.
 - Control → sobre el container.

¿Qué son los Containers Engines?

El motor de contenedor configura cada contenedor como una instancia aislada al solicitar recursos dedicados del sistema operativo para cada contenedor. Funciona en forma similar a un Hipervisor.

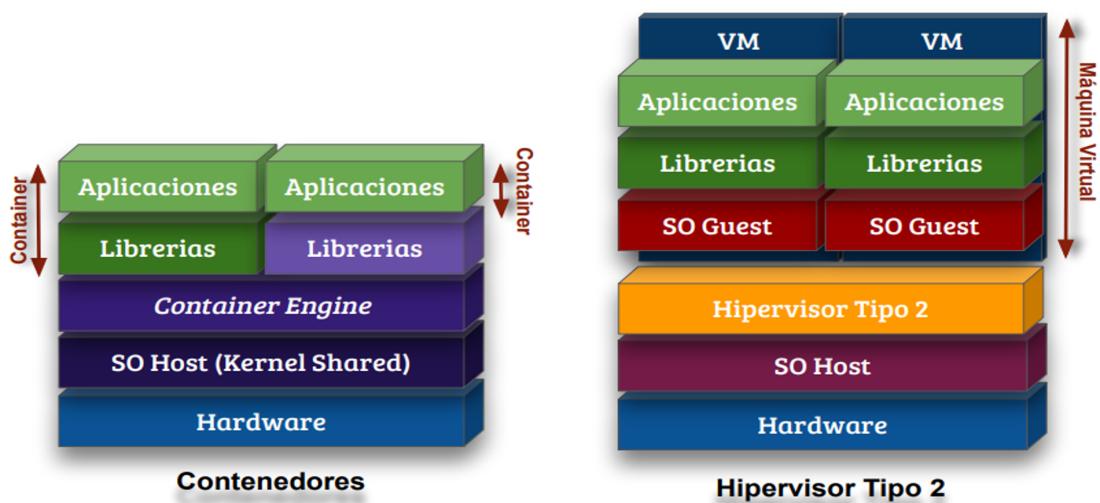
Generalmente cumplen con las siguientes funciones:

- Mantener un entorno de tiempo de ejecución ligero y una cadena de herramientas que gestione contenedores, imágenes y compilaciones.
- Crear un proceso para el contenedor.
- Administrar puntos de montaje del sistema de archivos
- Solicitar recursos del núcleo, como memoria, dispositivos de E/S y direcciones IP.

¿En qué se diferencia un Container de los VMs?

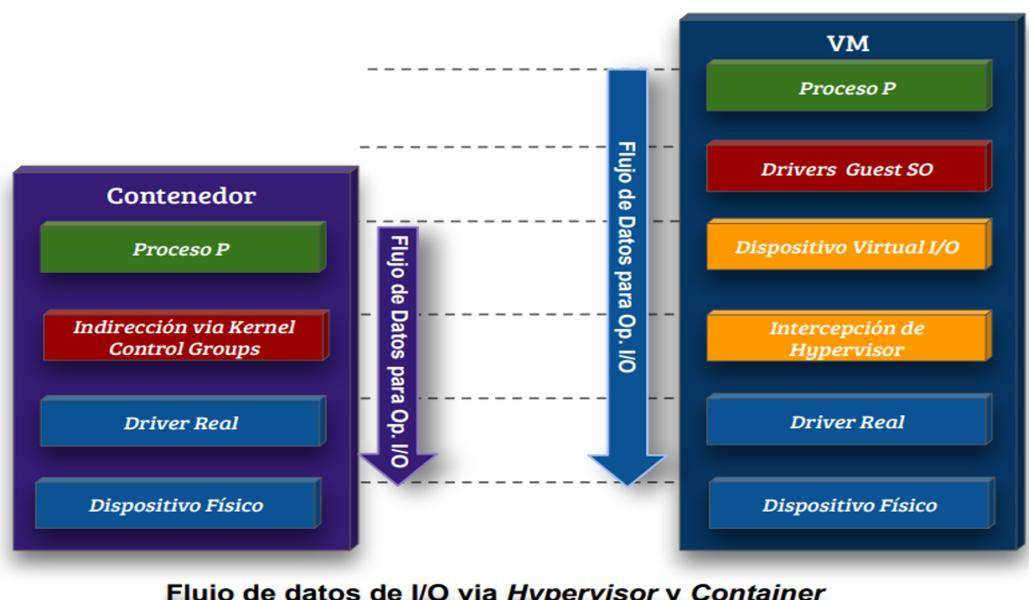
Una forma de entender mejor un contenedor es comprender cómo difiere de una máquina virtual (VM) tradicional. En la **virtualización tradicional**, ya sea local o en la nube, se aprovecha un hipervisor para virtualizar el hardware físico. Cada VM contiene un SO huésped, una copia virtual del hardware que el sistema operativo necesita para ejecutarse, junto con una aplicación y sus bibliotecas y dependencias asociadas.

En lugar de virtualizar el hardware subyacente, los contenedores virtualizan el **sistema operativo** (normalmente Linux) para que cada contenedor individual contenga *solo* la aplicación y sus bibliotecas y dependencias. La ausencia del sistema operativo huésped es el motivo por el que los contenedores son tan ligeros y, por lo tanto, rápidos y portátiles.



En los containers puedo tener copias de librerías completas o solo tener las aplicaciones (como en Docker).

Como se ve en el diagrama, el flujo de datos para operaciones I/O debe pasar por varias instancias antes de llegar a los dispositivos físicos. En cambio, en un conenedor, es mas rápido, porque pasa por menos esados.



¿Qué función tienen las imágenes y filesystem en los Container?

- Los Containers se instancian a partir de Imágenes (ISO) que hay que descargar o crear.
- Para mantener la aislación de cada container, debe tener su filesystem propio aislado. Generalmente son directorios en la máquina host.
- Frecuentemente usan COW (Copy-on-Write), para grabar en el container.

¿Qué beneficios tiene un Container?

- **Ligero:** los contenedores comparten el kernel del sistema operativo de la máquina, lo que elimina la necesidad de una instancia completa del sistema operativo por aplicación y hace que los archivos contenedores sean pequeños y fáciles de usar. Su tamaño más pequeño, especialmente en comparación con las máquinas virtuales, significa que pueden adaptarse rápidamente y soportar mejor las aplicaciones nativas de la nube que escalan horizontalmente.
- **Portátil e independiente de la plataforma:** los contenedores llevan todas sus dependencias con ellos, lo que significa que el software puede escribirse una vez y, después, ejecutarse sin la necesidad de volver a configurarse a través de los portátiles, la nube y entornos informáticos de instalaciones propias.
- **Soporta el desarrollo y la arquitectura modernos:** debido a una combinación de su portabilidad/consistencia de implementación entre plataformas y su tamaño pequeño, los contenedores son ideales para el desarrollo moderno y los estándares de aplicaciones, como DevOps, sin servidor y microservicios, que se crean con implementaciones de código regular en pequeños incrementos.
- **Mejora la utilización:** al igual que las máquinas virtuales, los contenedores permiten a los desarrolladores y operadores mejorar la utilización de la CPU y de la memoria de las máquinas físicas. Uno de los principales beneficios de los contenedores es que, dado que también permiten arquitecturas de microservicio, los componentes de la aplicación se pueden implementar y escalar de manera más granular, una alternativa interesante a tener que escalar una aplicación monolítica completa porque un solo componente tiene dificultades para procesar la carga.

A tener en cuenta:

- Sólo son portables entre sistemas que tienen el MISMO Kernel.
- Si una VM necesita una configuración de kernel diferente, no se puede usar.
- Tiene menor aislamiento que una VM, está entre el SO y la aplicación.

¿Qué es una aplicación Monolítica y que son los microservicios?

Por ejemplo, puedo tener una aplicación monolítica compuesta por python, mysql, apache, etc, y que se replique en base a las necesidades de uso.

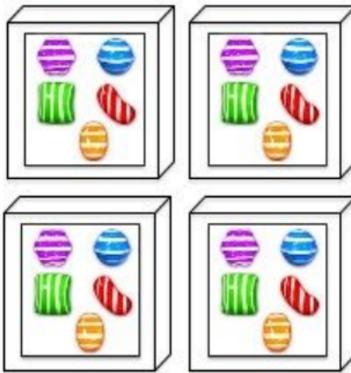
En una arquitectura de microservicios tendríamos cada servicio separados y generariamos llamadas entre cada uno. (Algo similar seria en base de datos, dos tablas que comparten llave, y una posee una llave foranea). MariaDB debería asignar los parámetros de red para que apache pueda comunicarse con los servicios de otro contenedor. Estos van a escalar en base a los requerimientos de sistema, replicando cada servicio individualmente, en base a que tanta carga tenga.

Por ejemplo, si en un container tengo un servicio al 80% de uso de la CPU, se instancia otro container para aligerar la carga repartiéndola en varios servicios del mismo tipo.

Una aplicación monolítica pone todas sus funcionalidades en un único proceso...



...y escala replicando lo monolítico en múltiples servidores

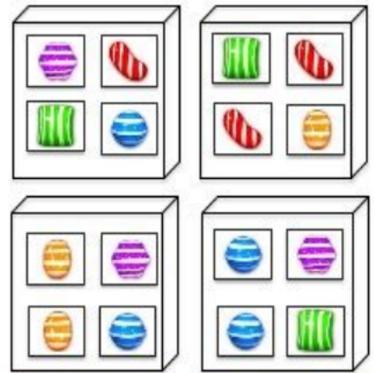


Arquitectura Monolítica

Una arquitectura de microservicios pone cada elemento de funcionalidad en un servicio separado...



...y escala distribuyendo dichos servicios a través de los servidores, según se requiera



Arquitectura Microservicios

¿Qué es LXC?

El término de “**contenedor de Linux**” o **LXC** se refiere a las **aplicaciones virtualizadas sobre la base de Linux**, así como a **la plataforma y la tecnología de contenedores subyacente**.

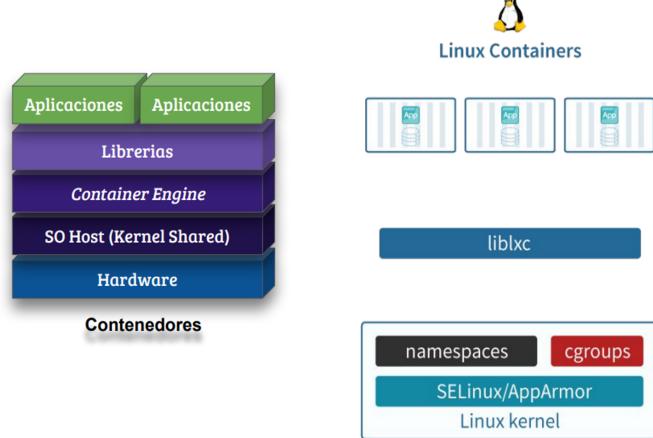
El objetivo de LXC es crear un entorno que **se parezca lo más posible a una instalación estándar de Linux sin la necesidad de un kernel independiente**.

La plataforma de contenedores de Linux actual utiliza las siguientes características del *kernel* para incorporar aplicaciones y procesos en los contenedores:

- **Espacios de nombres del núcleo (ipc, uts, mount, pid, red y usuario)**
- **Perfiles AppArmor y SELinux**
- **Directrices de Seccomp**
- **Chroots (utilizando pivot_root)**
- **Capacidades del kernel**
- **cgroups (grupos de control)**

Los contenedores de Linux deben permanecer compactos. Por lo tanto, solo constan de unos pocos componentes:

- Biblioteca liblxc
- Vinculación de varios lenguajes para la API:
 - Python 3 (soporte a largo plazo en 2.0.x)
 - etc.



¿Qué es Docker?

Docker es un sistema operativo para contenedores. De manera similar a cómo una máquina virtual virtualiza (elimina la necesidad de administrar directamente) el hardware del servidor, los contenedores virtualizan el sistema operativo de un servidor. Docker se instala en cada servidor y proporciona comandos sencillos que puede utilizar para crear, iniciar o detener contenedores.

En el Docker file se ponen los requerimientos a ejecutar por Docker, se genera una imagen (binario de la aplicación), y dicha imagen, una vez que se corre, muestra el microservicio, que es la imagen ejecutándose en dicho container.

Docker: Plataforma de contenedor de software diseñada para desarrollar, enviar y ejecutar aplicaciones aprovechando la tecnología de los contenedores.

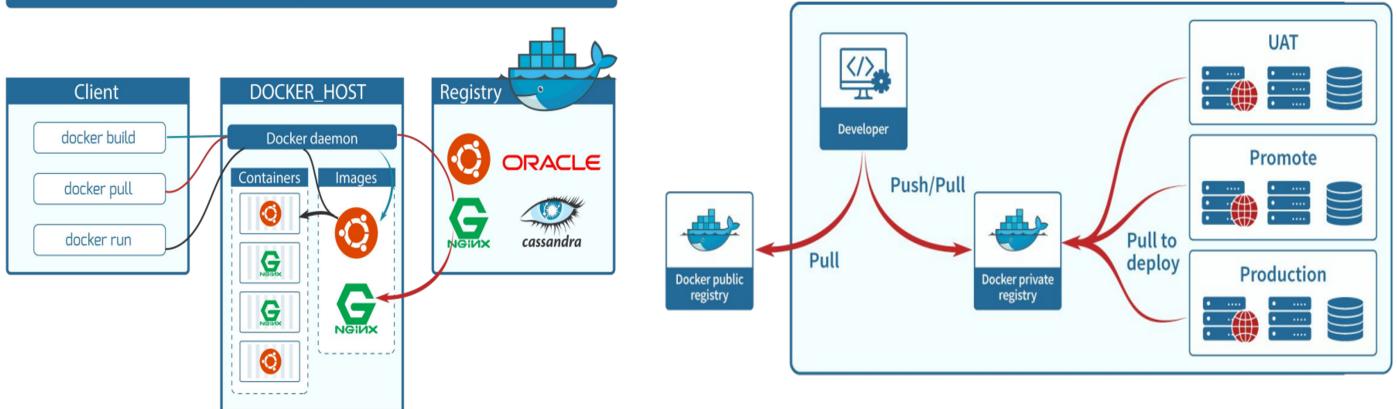
Contenedor: Un contenedor proporciona virtualización ligera a nivel de sistema operativo mediante la abstracción del "espacio del usuario". Los contenedores comparten el núcleo del sistema host con otros contenedores. Un contenedor, que se ejecuta en el sistema operativo host, es una unidad de software estándar que empaqueta código y todas sus dependencias, para que las aplicaciones se puedan ejecutar de forma rápida y fiable de un entorno a otro. Los contenedores no son persistentes y se activan desde imágenes.

Motor de Docker: El software de host de código abierto que crea y ejecuta los contenedores. Los motores de Docker funcionan como la aplicación del servidor del cliente que admite contenedores en varios servidores Windows y sistemas operativos Linux, como Oracle Linux, CentOS, Debian, Fedora, RHEL, SUSE y Ubuntu.

Imágenes de Docker: Colección de software que se ejecutará como un contenedor que incluye un conjunto de instrucciones para crear un contenedor que se pueda ejecutar en la plataforma Docker. Las imágenes no son modificables, de modo que para realizar cambios en una imagen es preciso crear otra nueva.

Docker Registry: Lugar para almacenar y descargar imágenes. Registry es una aplicación de servidor escalable y sin estado que almacena y distribuye imágenes de Docker. Es como el GitHub de los contenedores de docker.

DOCKER COMPONENTS



¿Qué es Kubernetes?

Kubernetes es una plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios. Kubernetes facilita la automatización y la configuración declarativa. Tiene un ecosistema grande y en rápido crecimiento. El soporte, las herramientas y los servicios para Kubernetes están ampliamente disponibles.

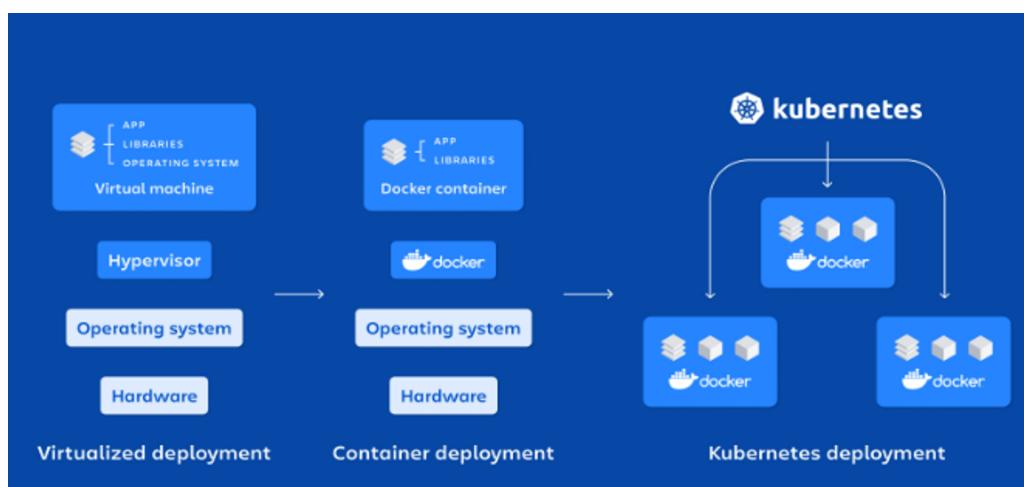
- una plataforma de contenedores
- una plataforma de microservicios
- una plataforma portable de nube

Kubernetes ofrece un entorno de administración **centrado en contenedores**. Kubernetes orquesta la infraestructura de cómputo, redes y almacenamiento para que las cargas de trabajo de los usuarios no tengan que hacerlo.

Puede agrupar en **clústeres** conjuntos de hosts que ejecuten contenedores de Linux, y Kubernetes lo ayudará a gestionarlos con facilidad y eficacia.

Los clústeres de Kubernetes pueden contener hosts locales y en nubes públicas, privadas o híbridas.

Docker es una plataforma de contenedorización y un tiempo de ejecución de contenedores, mientras que Kubernetes es una plataforma para ejecutar y gestionar contenedores a partir de numerosos tiempos de ejecución de contenedores. Kubernetes admite varios tiempos de ejecución de contenedores, incluido Docker.

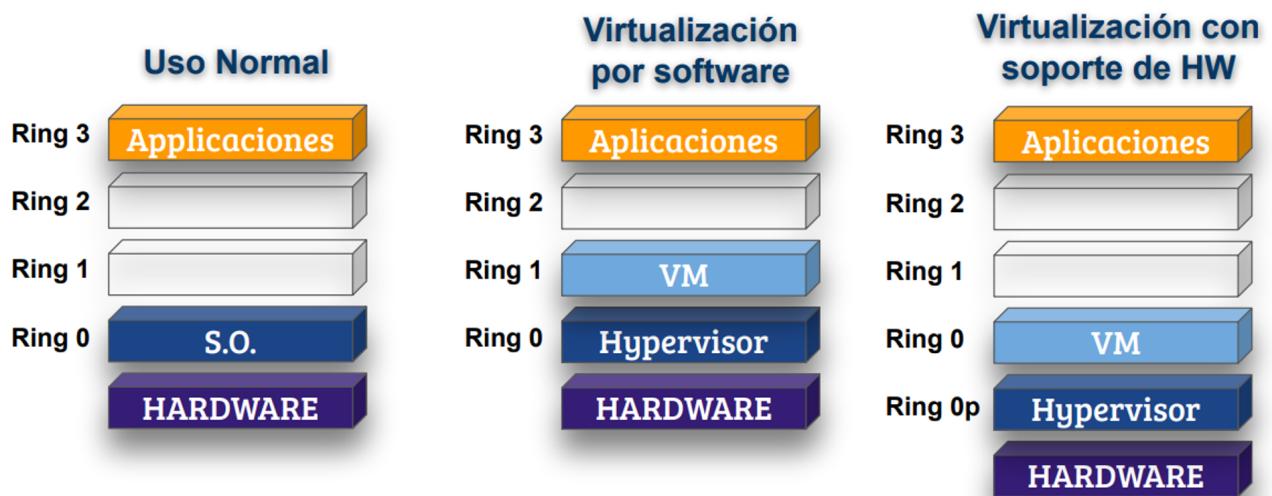


VIRTUALIZACIÓN CON SOPORTE EN HARDWARE

¿Qué es VT-x?

Ring: Anillos de prioridad, sólo la prioridad 0 tiene acceso privilegiado al Hardware. Siendo el Ring 3 la menor prioridad para permisos de usar el hardware

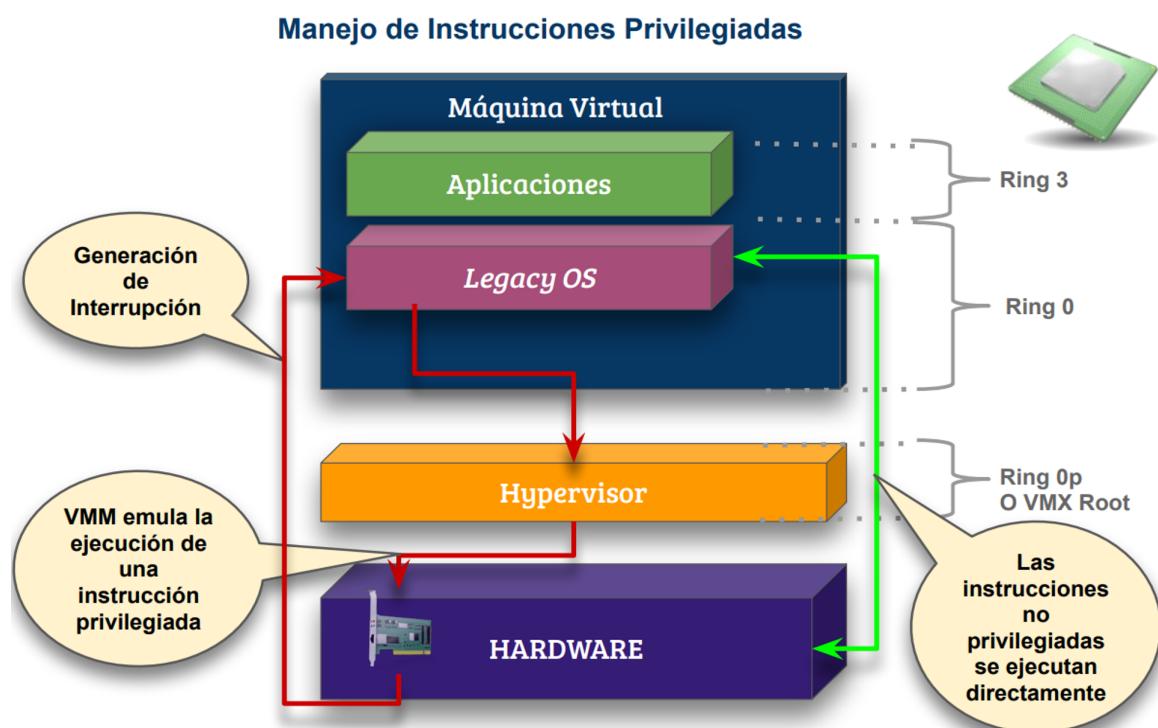
- VT-x es una tecnología implementada en Intel, se basa en utilizar un anillo 0 “desprivilegiado”, esto es porque antes sólo se usaba el anillo 0 y el anillo 3, desaprovechando performance para la Virtual Machine.
- En la virtualización se puede ocupar un anillo más para poder utilizar la Virtual Machine, el problema de esto es que no podía utilizar el hardware de forma directa, teniendo que pasar siempre por el hypervisor, así produciendo errores y bajadas en la performance.
- En la virtualización con soporte de HW (Gracias a VT-x en intel o AMD-v en AMD) se ocupa el anillo 0 con la Virtual Machine, lo que permite a la Virtual Machine interactuar más directamente con los recursos de hardware, logrando hacer lo que haría un sistema operativo pero de forma virtualizada. El hypervisor sigue ocupando el anillo cero, pero ahora de forma “privilegiada” siendo el anillo cero de la Virtual Machine un anillo cero “desprivilegiado” pero con acceso prácticamente directo al software, salvo para instrucciones privilegiadas de alto nivel o simplemente instrucciones privilegiadas.



¿Cómo es la arquitectura de VT-x?

Ahora que el sistema operativo está corriendo en el anillo 0 no privilegiado:

- Cuándo la máquina virtual quiera acceder a una emulación de hardware (Instrucción privilegiada de alto nivel) debe pasar si o sí por el hypervisor, la ventaja de esto es que la instrucción vuelve directamente al sistema operativo (**Flecha roja**) produciéndose una interrupción de forma directa con la máquina virtual
- Las instrucciones no privilegiadas se comunican de forma directa entre el sistema operativo de la máquina virtual y el hardware, lo que agiliza la velocidad de la máquina virtual. (**Flecha verde**) Cómo green arrow jaja dc reference



¿Qué es VT-d?

Siendo el VT-x una tecnología para acelerar la máquina virtual, el VT-d ofrece instrucciones para acelerar tarjetas de comunicaciones virtuales, es decir, controladores de disco duro, controladores de tarjeta de red, etc. El hypervisor entonces puede soportar la virtualización de tareas de Input/Output siguiendo los siguientes modelos:

- **Emulación:** El VMM emula un dispositivo existente, lo cual da compatibilidad pero, sacrifica rendimiento.
- **Interfaces Sintéticas:** Similar a la emulación pero se expone al Guest un dispositivo nuevo, diseñado para mejorar el rendimiento. Puede ser paravirtualizado. Su compatibilidad es menor. (Básicamente emula un componente de hardware que no existe físicamente, como tratar de emular velocidad del procesador con RAM o viceversa, no va a tener el mayor rendimiento pero va a funcionar bien.)

- **Asignación Directa:** El Guest ejecuta el driver directamente, y el dispositivo sólo puede estar definido en esa VM. Generalmente requiere de soporte en hardware. PCI-Passthrough. (Se envían instrucciones al hardware para que otros “anillos” (**Hypervisor y Sistema Operativo**) de prioridades no puedan utilizar el hardware real, así queda solamente asignado a la máquina virtual. Lo que le da mayor rendimiento a la máquina virtual)
- **Distribución de I/O en Dispositivo:** (**I/O Device Sharing**). El dispositivo posee soporte de múltiples interfaces funcionales, que pueden asignarse a diferentes VM's. El VMM tiene una mínima injerencia en la transmisión de datos. Requiere SR-IOV (El Hypervisor va a tener mínima ingerencia entre la repartición de recursos de hardware, lo que permite este modelo es compartir el mismo hardware a varias máquinas virtuales distintas)

¿Cómo es la arquitectura de VT-d?

Nota: **VT-x** significa **V**irtualization **T**echnology for **e**Xecution mientras que **VT-d** significa **V**irtualization **T**echnology for **D**irected input/output

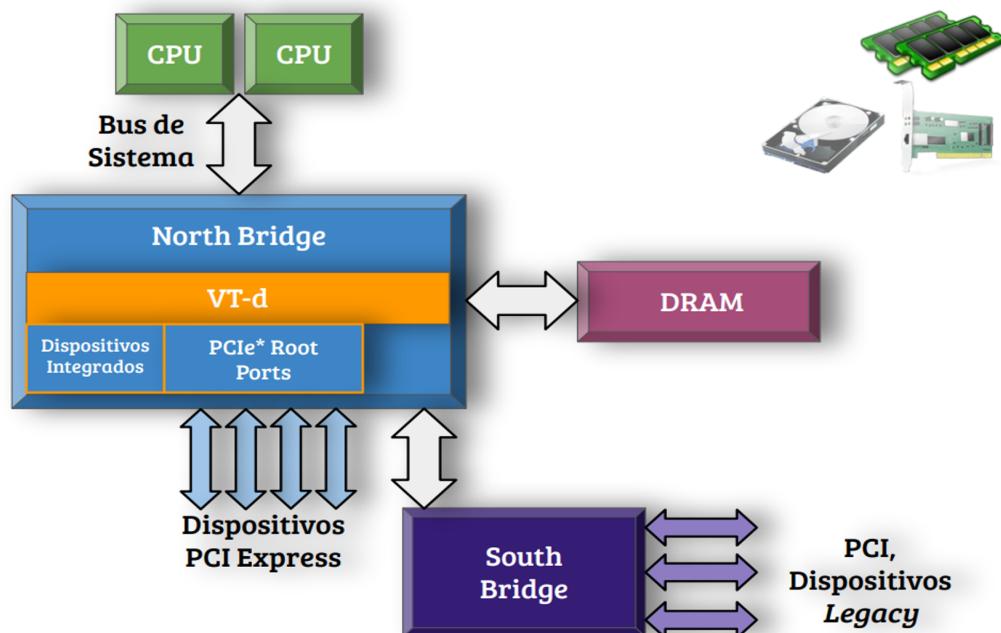
El hypervisor tenía dos desafíos:

- La administración de la memoria (MMU)
- El acceso I/O al hardware virtual

El VT-d y el AMD-Vi (Que es lo mismo pero de AMD) tienen 3 objetivos principales:

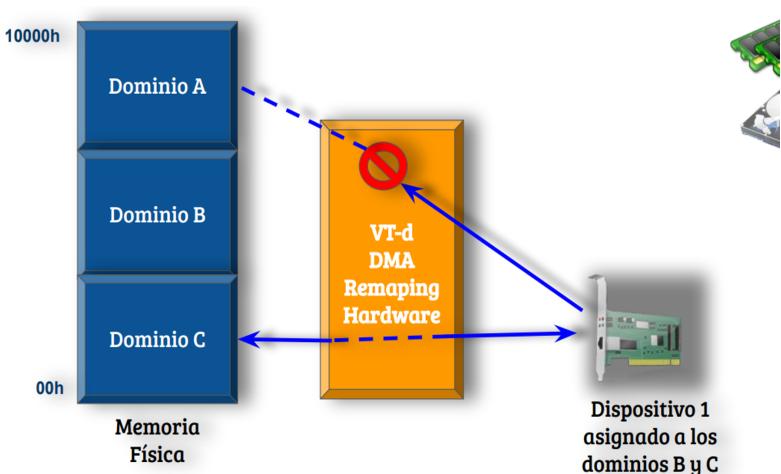
- Remapeo de DMA (Acceso directo a memoria) asistido por hardware
- Asignación directa de dispositivos I/O a los VM's
- Remapeo y Routing de interrupciones

El VT-d se encuentra en el chipset (Más específicamente en el North Bridge) y que la BIOS de la PC tengan soporte para este mismo. El VT-d se conecta de forma directa con la DRAM y con los dispositivos PCI express (Todo lo que está en naranja es lo que tiene conexión directa, por ejemplo no tiene conexión directa con el South bridge).



¿En qué consiste el DMA Remapping?

El dispositivo 1 (Representación de un dispositivo de I/O genérico) cuando trabaje con una máquina virtual en particular, si tiene el buffer casi lleno y tiene manejo de DMA, va a mandar todo lo que tenga en el buffer y una interrupción a la memoria del sistema (Las flechitas que indican al dominio C). La idea es que el dispositivo vaya a dominios específicos con cada máquina virtual, por lo que si el dispositivo quiere entrar a otro dominio de la memoria no pueda (Flechita de arriba).

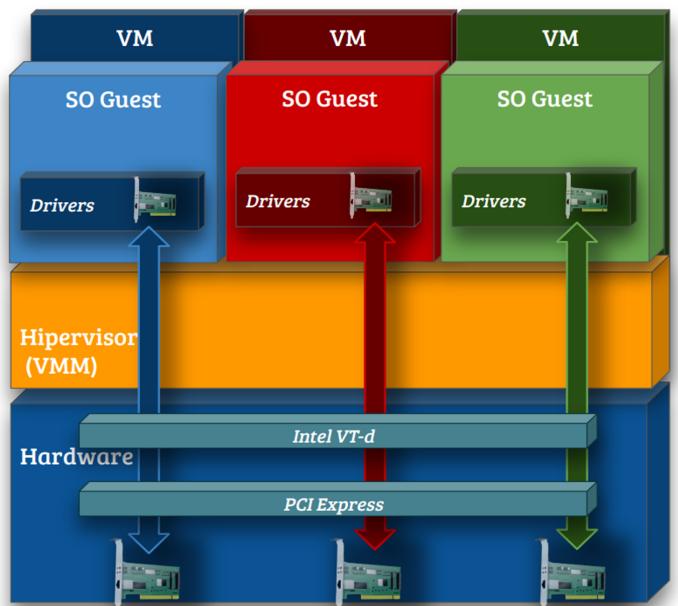


Cuando el dispositivo 1 intenta acceder a la memoria del dominio A, el hardware niega el acceso.

¿En qué consiste la asignación directa?

Pero para asignarle un dispositivo físico a una máquina virtual se le debe hacer una asignación directa para lograr un PCI Passthrough\$^1\$. Entonces para cada máquina virtual se remapea por DMA para asignarle una parte del dominio de la memoria a una máquina virtual. Esto ayuda a la velocidad de conexión y también sirve para los dispositivos físicos los cuales no podemos reemplazar con dispositivos emulados. Esto está pensado para servidores principalmente.

1: El Pablo le dice PCI (Peripheral component interconecct) Passthrough a conectar exitosamente un dispositivo de entrada y salida correctamente a la máquina virtual.

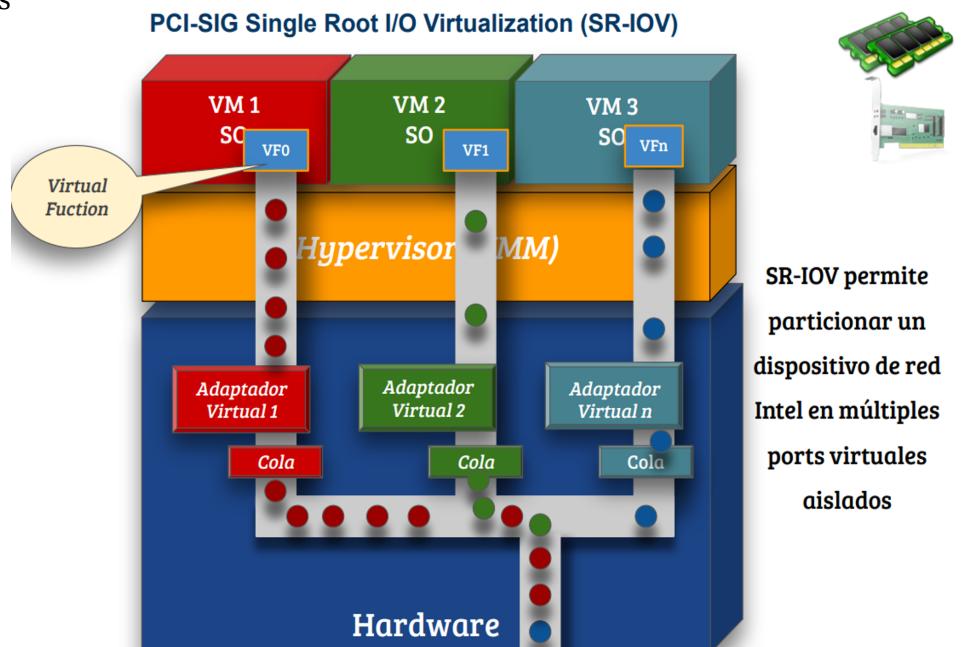


¿Qué es VT-c?

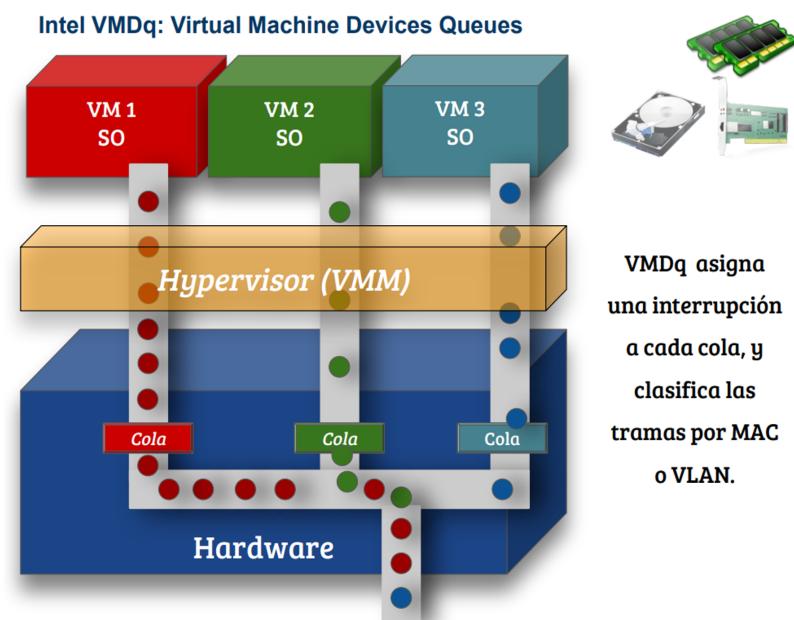
El **VT-c** es **Virtualization Technology for Communication**. Tiene dos formas de tratar de resolver la performance con respecto de la máquina virtual y el hardware.

En la **primera forma** la placa (de servidor) tiene que tener soporte para esta tecnología, el VT-c nos permite definir cuántas funciones virtuales vamos a utilizar (las placas del ejemplo tienen hasta 64 funciones). Se pueden activar en bloques de 2, 4, 8, etc. Es decir, se asigna una parte de la placa de red como recurso para la Virtual Machine, estaríamos “virtualizando” una placa de red a través de un adaptador virtual. (Los círculos

de colores son los paquetes que envía la placa de red, entonces ahora la placa de red sabe a qué máquina virtual enviarle cada paquete.). Todo esto ocurre por encima del hypervisor, es decir, tiene una comunicación directa con las máquinas virtuales.



En la **segunda forma** se asignan colas (terrible) a la placa de red, se mapean colas en particular de la placa de red para cada máquina virtual. La diferencia está en que no se utilizan funciones virtuales en este caso, toma los recursos directamente del hardware. Por lo tanto los paquetes se pueden enviar por VLAN o por MAC. Las tramas que se envían pueden utilizar la misma cola.



¿Qué diferencias hay entre los distintos Hypervisores?

Nota: Una **máquina virtual anidada** es una máquina virtual dentro de otra, por ejemplo, virtualizo windows 10, pero para una máquina virtual en específico quiero virtualizar windows XP y linux (Todo dentro de la misma máquina) La máquina virtual entonces crea un hypervisor “virtual” para emular el Hypervisor del anillo 0p pero sin las tecnologías de VT-x, VT-d ni VT-c. Por lo tanto estas máquinas van a ser lentas.

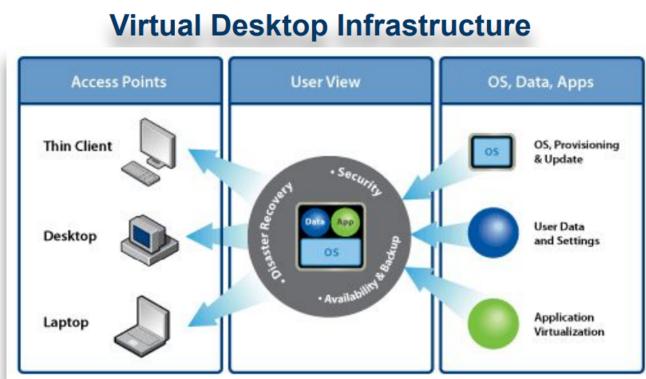
Para poder arreglar esto los hypervisores virtuales deben poder soportar un “puenteo” entre el hypervisor principal (El que se conecta directamente con el hardware, el del anillo 0p) para poder utilizar las tecnologías del procesador casi como si fueran nativas en el hypervisor virtualizado para la virtualización anidada, abajo hay una comparativa entre los distintos hypervisores, son más que nada características, Proxmox es el mejor según el Pablo por ser Open Source y tener todas las características que tienen los otros.

	Proxmox (KVM - LXC)	VMWare Vsphere	Windows Hyper-V	Citrix XenServer
Guest OS	Windows y Linux (KVM). Otros SO.	Windows, Linux, Unix	WIndows Modernos, Linux Limitado	Mayoria de Windows, Linux limitado
Open Source	Si	No	No	Si
Linux Containers (LXC)	Si	No	No	No
High Availability	Si	Si	Requiere MS Failover Clustering. Soporte para guest limitado	Si
Backup en Caliente aka Snapshots	Si	Si	Limitado	Si
PCI-Passthrough	Si	Si	No	Si

¿En qué consiste la arquitectura de virtualizacion VDI?

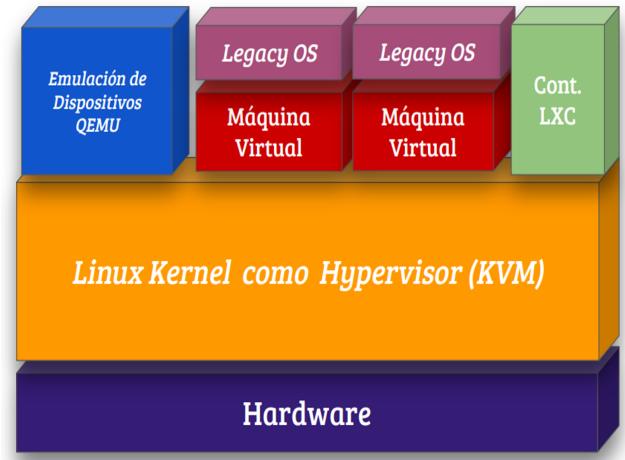
VDI es una arquitectura de virtualización de escritorios. Es como si todas las terminales estuvieran conectadas a un servidor, como ejemplo, pero con la diferencia de que tienen todas las funciones de un sistema operativo completo. Esto tiene ventajas para trabajar, ya que si se rompe la computadora dónde se está corriendo la máquina virtual, esta sólo se debe reemplazar y no se va a perder ningún dato ya que sólo se debe volver a conectar al “servidor” y se seguirá teniendo todo como estaba antes.

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



¿Cómo funciona proxmox? (Hypervisor para virtualización)

Es esta arquitectura de hypervisor el kernel de linux es el mismo hypervisor, QEMU es un emulador de arquitecturas. Proxmox permite varias máquinas virtuales corriendo simultáneamente y es gratis.



¿Cómo funciona el “servidor” de máquinas virtuales para VDI?

El hardware está en un sólo lugar específico, los servidores se conectan a este y estos se conectan a los clientes, si un servidor se quema la máquina virtual que estaba siendo soportada por este se envía a otro servidor que no tenga su capacidad de memoria llena para evitar fallos. Ya que si tuviera la memoria llena, guardaría la máquina virtual en un disco duro, lo que haría que la máquina virtual soportada fuera excesivamente lenta.

