

RESUMEN SISTEMAS OPERATIVOS - LUCAS CROSTA

UNIDAD 1: INTRODUCCIÓN

1.1 ELEMENTOS BÁSICOS

Computadora consta de procesador, memoria y componentes de E/S. Se interconectan para lograr la función principal: ejecutar programas. Elementos principales:

Procesador: Controla el funcionamiento del computador. Si es solo uno = CPU	Memoria principal: Almacena datos y programas. Normalmente volátil. Memoria real/primaria.
Módulo de E/S: Transfieren los datos entre el computador y su entorno externo.	Bus del sistema: Proporciona comunicación entre los procesadores, la memoria principal y los módulos de E/S

Una función del procesador es el intercambio de datos con la memoria, para la cual se usan registros internos:

- RDIM: registro de dirección de memoria. Especifica la dirección de memoria de la siguiente lectura/escritura.
- RDAM: registro de datos de memoria. Contiene los datos que se van a escribir o los leídos.

1.2 EJECUCIÓN DE INSTRUCCIONES

Un programa que va a ejecutarse en un procesador consta de un conjunto de instrucciones almacenadas en la memoria. El procesamiento de una instrucción consta de dos partes: lee/busca (fase de búsqueda) la instrucción de la memoria y ejecuta cada una (fase de ejecución).

Ciclo de instrucción: procesamiento requerido por una sola instrucción.

- Búsqueda y ejecución de una instrucción:

Al principio de cada ciclo de instrucción, el procesador lee una instrucción de la mm.

- PC: almacena la dirección de la siguiente instrucción a leer. Se incrementa luego de leer una.
- IR: se carga la instrucción leída.

La instrucción tiene bits que especifican la acción que debe realizar el procesador. Normalmente son 4 categorías:

Procesador-Memoria: Se pueden transferir datos desde el procesador a la memoria o viceversa.	Procesador-E/S: Se pueden enviar datos a un dispositivo periférico o recibirlos, transfiriendo los entre el procesador y un módulo de E/S.
Procesamiento de datos: operaciones lógicas o aritméticas.	Control: instrucción específica que se altera la secuencia de ejecución.

Formato de instrucción:



(a) Formato de instrucción

4 bits: código de operación.

12 bits: direccionamiento de palabras de mm.

Sistema de E/S:

Se pueden intercambiar datos directamente entre un módulo de E/S y el procesador. Es deseable permitir que los intercambios de E/S se produzcan directamente con la memoria para liberar al procesador de la tarea de E/S.

El procesador concede a un módulo de E/S la autorización para leer o escribir de la memoria, de manera que la transferencia entre memoria y E/S puede llevarse a cabo sin implicar al procesador. Este módulo es llamado DMA (Direct Memory Access)

1.3 INTERRUPCIONES

Todas las computadoras proporcionan un mecanismo por el cual otros módulos (memoria y E/S) pueden interrumpir la secuencia normal del procesador. Las interrupciones constituyen una manera de mejorar la utilización del procesador.

Clases de interrupciones:

De programa: Generada por alguna condición que se produce como resultado de la ejecución de una instrucción. Por ej: división por 0, referencia fuera del espacio permitido de mm.	Por temporizador: Generada por un temporizador del procesador.
De E/S: Generada por un controlador de E/S para señalar la conclusión normal o error.	Por fallo de hardware: Generada por un fallo, suministro de energía.

Interrupciones y el ciclo de instrucción:

Gracias a las interrupciones, el procesador puede ejecutar otras instrucciones mientras que la operación de E/S se lleva a cabo. Cuando se completa el procesamiento de la interrupción, se reanuda la ejecución. Por tanto, el programa de usuario (sería el que se estaba ejecutando) no tiene que contener ningún código especial para tratar las interrupciones; el procesador y el sistema operativo son responsables de suspender el programa de usuario y, posteriormente, reanudarlo en el mismo punto.

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

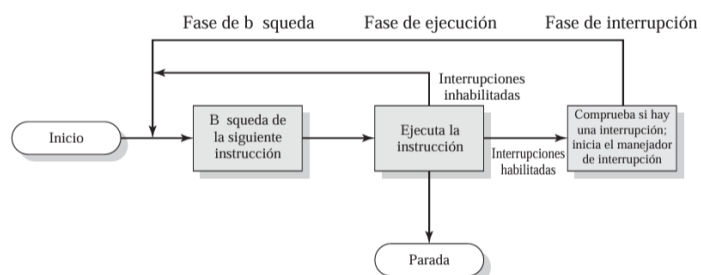


Figura 1.7. Ciclo de instrucción con interrupciones.

Interrupción simple

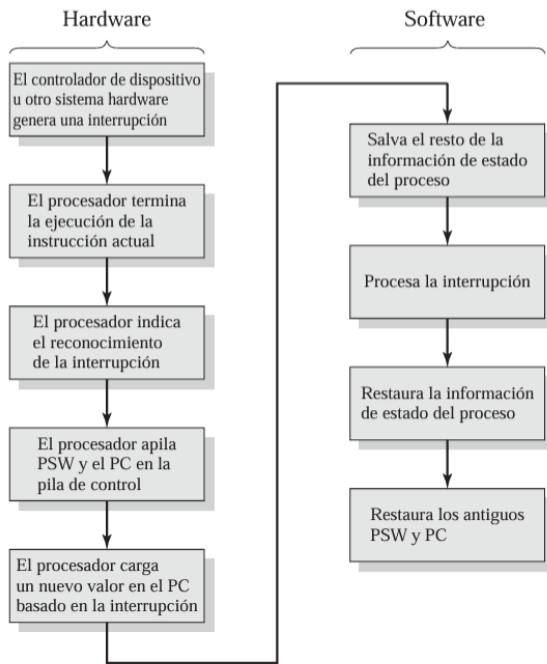


Figura 1.10. Procesamiento simple de interrupciones.

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

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

Salvar el contexto sería guardar el PSW y PC en la pila de control y se guarda el estado del proceso.

Interrupcion multiple

Cuando se produce una interrupción y se está ejecutando otra rutina de interrupción. Alternativas:

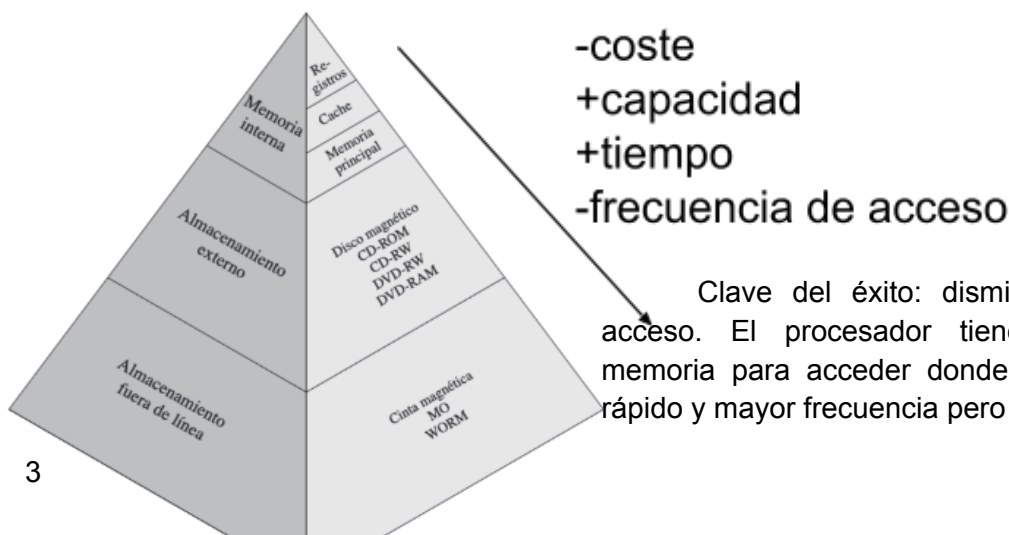
- Inhabilitar interrupciones mientras se procesa una interrupción. El procesador ignorará cualquier señal de interrupción nueva. Permanecerá pendiente y se comprobará luego que se reabiliten antes de continuar con el programa anterior. Orden secuencial.
x: no tiene en cuenta las prioridades o urgencia de las interrupciones.
- Definir prioridades. Define prioridades y permite que una interrupción con mayor prioridad interrumpa la ejecución de un manejador de interrupciones de una de menor.

1.4 JERARQUÍA DE MEMORIA

Siempre se cumple:

- Menor tiempo de acceso, mayor coste por bit.
- Mayor capacidad, menor coste por bit.
- Mayor capacidad, menor velocidad de acceso.

Por esto la solución no es usar solo un componente, si no, una jerarquía de memorias.



Clave del éxito: disminuir la frecuencia de acceso. El procesador tiene varios niveles de memoria para acceder donde el nivel 1 es el más rápido y mayor frecuencia pero más chico.

1.5 TÉCNICAS DE COMUNICACIÓN DE E/S

3 técnicas para llevar a cabo operaciones de E/S:

- E/S PROGRAMADA:

El procesador ejecuta mandato hacia el módulo de E/S y lo realiza el módulo. No interrumpe al procesador, por lo tanto el procesador debe tener un papel activo para determinar cuando se completa la instrucción de E/S. Comprueba periódicamente el estado del módulo.

Procesador responsable de extraer los datos y almacenarlos. No hay interrupciones.

x: consume tiempo del procesador innecesariamente.

- E/S DIRIGIDA POR INTERRUPCIONES:

Se interrumpe al procesador cuando el módulo de E/S está listo para intercambiar datos. Desde el punto de vista del módulo:

Módulo recibe un mandato, lee los datos de un periférico. Una vez que los datos estén en el registro, módulo genera petición de interrupción al procesador, esperando a que el procesador pida sus datos.

Punto de vista del procesador:

Genera un mandato, salva el contexto. Al final de cada ciclo de instrucción, revisa si hay peticiones. Cuando se produce la interrupción de E/S, lee la palabra de datos de E/S y la almacena. Luego restaura el contexto y reanuda la ejecución.

x: consume tiempo del procesador por cada palabra de datos.

- ACCESO DIRECTO A MEMORIA (DMA):

Es la técnica más eficiente. Se usa DMA, que lleva a cabo un módulo separado conectado en el bus del sistema o módulo de E/S. Cuando el procesador quiere leer/escribir un bloque, genera un mandato al DMA, enviando la siguiente información (es lectura o escritura; dirección del dispositivo involucrado; posición inicial para leer o almacenar; número de palabras a leer o escribir).

El procesador continúa con otro trabajo, el DMA realiza todo. Procesador sólo involucrado al principio y final. El procesador a veces puede pelear por el bus de datos ya que lo puede estar usando el DMA.

1.6 FUNCIONES Y OBJETIVOS DE LOS SISTEMAS OPERATIVOS

Sistema operativo es un programa que controla la ejecución de aplicaciones y programas y que actúa como interfaz entre las aplicaciones y el hardware. Tiene 3 objetivos:

- facilidad de uso: facilita el uso de la computadora.
- eficiencia: permite que los recursos de un sistema se utilicen de forma eficiente.
- capacidad para evolucionar: se debe poder desarrollar nuevas funciones.

Servicios que proporciona:

- desarrollo de programas: proporciona servicios y utilidades para asistir al programador en la creación de programas.
- ejecución de programas: el SO realiza una serie de operaciones para poder ejecutar un programa.

- Acceso a dispositivos de E/S: SO proporciona una interfaz para todos los dispositivos de E/S.
- Acceso controlado a los ficheros: SO proporciona mecanismos de protección para controlar el acceso a los ficheros.
- Acceso al sistema: SO controla el acceso al sistema completo y a recursos del sistema.
- detección y respuesta de errores: SO debe proporcionar una respuesta que elimine el error con el menor impacto posible.
- contabilidad: recoger estadística para administrar mejor.

Gestor de recursos:

SO gestiona los recursos. Las funciones del sistema operativo actúan de la misma forma que el resto del software; es decir, se trata de un programa o conjunto de programas ejecutados por el procesador (consume recursos). También suele ceder el control y depende del procesador para volver a tenerlo.

Una parte del SO se encuentra en la memoria principal, incluyendo al kernel (funciones del SO más utilizadas).

1.7 PRINCIPALES LOGROS DE LOS SISTEMAS OPERATIVOS

1. PROCESOS:

Concepto fundamental en la estructura de SO. Un proceso es: un programa en ejecución, una instancia de un programa ejecutándose, etc. Sus 3 componentes son: programa ejecutable, datos asociado que necesite el programa, contexto de ejecución del programa (es por el cual el sistema es capaz de controlar al proceso).

2. GESTIÓN DE LA MEMORIA:

Los gestores del sistema necesitan un control eficiente y ordenado de la asignación de recursos. Para satisfacerlo, el SO tiene 5 responsabilidades:

- aislamiento de procesos: un proceso no interfiere los datos en memoria de otro.
- asignación y gestión automática: asignación dinámica de memoria por demanda.
- soporte a la programación modular: alterar los módulos dinámicamente.
- Protección y control de acceso: permiso para que un usuario/proceso modifique la memoria de otro proceso en memoria compartida.
- almacenamiento a largo plazo.

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. Las referencias a memoria se hacen con dirección virtual y la Unidad de Gestión de Memoria realiza la traducción en dirección física/real.

3. PROTECCIÓN Y SEGURIDAD DE INFORMACIÓN:

- Autenticidad/Control de acceso: Verifica la identidad del usuario y la validez de los datos.
- Certificación/Control del flujo de la información: Regula flujo de datos dentro del sistema según las especificaciones requeridas y la entrega a los usuarios.
- Disponibilidad: Protección frente a interrupciones.
- Confidencialidad: Que los usuarios no accedan a datos que no tienen permiso.
- Integridad de datos: Protección de datos frente a modificaciones no autorizadas.

4. PLANIFICACION Y GESTION DE RECURSOS:

Gestionar varios recursos disponibles y planificar su uso. Para esto se debe tener en cuenta:

- Equidad (repartir a todos por igual): Dar la misma cantidad de recursos a cada proceso con demandas similares.
- Sensibilidad diferencial (establecer prioridad al repartir recursos): Discriminar entre clases de trabajos distintos para tomar decisiones dinámicas en cuanto al reparto de recursos.
- Eficiencia (lo mejor posible): Maximizar transmisión, minimizar tiempo de respuesta y acomodar tantos usuarios como sea posible.

Para un mejor reparto de recursos es recomendable monitorizar el rendimiento y realizar los ajustes correspondientes. Hay 3 tipos de colas para la espera de recursos: De corto plazo (a punto de usar el procesador), de largo plazo (cuando se va a usar el procesador se pasan a la de corto plazo) y de E/S (varios procesos al mismo dispositivo).

5. ESTRUCTURA DEL SISTEMA:

El SO puede ser muy complejo. Para gestionar la complejidad y eliminar estos problemas, se ha puesto énfasis en la estructura del SO:

- Sistema modular (SO pequeños): organizar y detectar errores más fácilmente. Módulos simples y mínimamente conectados entre sí.
- Serie de niveles (SO grandes): cada nivel realiza funciones relacionadas con los niveles inferiores (funciones más primitivas). La modificación de uno no interfiere con los demás.

Los niveles descompone un problema dentro de problemas más chicos y manejables.

Sirve para facilitar la programación del sistema, evolución y tareas de diagnóstico.

Niveles bajos tratan con una escala pequeña de tiempo, ya que se relacionan con el hardware.

Niveles altos se comunican con el usuario.

Nivel 1 – 4: Constituyen el hardware del procesador, pero algunos elementos del SO aparecen (Manejador de interrupciones).

Nivel 5 – 7: El SO trata únicamente con el procesador.

Niveles 8 – 13: Dispositivos externos (Redes, periféricos, etc.)

UNIDAD 2: PROCESOS

2.1 DESCRIPCIÓN

Principales requisitos de un sistema operativo se pueden expresar en referencia a los procesos:

- SO debe intercalar la ejecución de varios procesos para maximizar el uso del procesador.
- SO debe asignar recursos a los procesos mediante alguna política.
- SO debe dar soporte a la comunicación entre procesos y creación de procesos.

2.2 PROCESO

Programa → ejecutable

Programa en ejecución → proceso (es cuando el programa usa los recursos).

La diferencia es el contexto de ejecución.

Proceso puede ser: programa en ejecución, instancia de un programa ejecutado, entidad que se puede asignar y ejecutar, unidad de actividad que se caracteriza por la ejecución de una secuencia de instrucciones usando recursos.

Componentes de un proceso son: código de programa (puede compartirse con otros procesos del mismo programa) y conjunto de datos. Además, el proceso está compuesto por el bloque de control de proceso (BCP) que almacena (identifica a cada proceso):

Identificador: identificador único asociado al proceso.	Estado: estado del proceso.	Prioridad: nivel de prioridad del proceso.
PC: dirección de la siguiente instrucción del programa que se ejecutará.	Punteros a memoria: punteros al código del programa y datos asociado al proceso.	Datos de contexto: datos presentes cuando está corriendo.
Información de estado de E/S: peticiones de E/S pendientes, dispositivos asignados.	Información de auditoría: puede incluir la cantidad de tiempo de procesador y de reloj utilizados.	

2.3 ESTADOS DE LOS PROCESOS

Para que un programa se ejecute se debe crear un proceso para dicho. Se puede caracterizar el comportamiento de un determinado proceso listando la secuencia de instrucciones que se ejecutan para él, esta lista se llama traza. Se puede caracterizar el comportamiento de un procesador mostrando cómo se enlazan las trazas de varios procesos.

Debe haber información del proceso incluyendo su estado actual y localización en memoria (BCP). Los procesos que no se están ejecutando "NO EJECUTANDO" estarán en la cola esperando, y los que estén en "EJECUTANDO" se están ejecutando. Un proceso que se interrumpe se transfiere a la cola, si se finaliza o aborta sale del sistema. El activador es el encargado de seleccionar algo de la cola para ejecutar.

- CREACIÓN DE PROCESOS:

Cuando se añade un nuevo proceso a los que se están gestionando, el SO construye las estructuras necesarias de cada proceso. Razones por las que el SO puede crear un proceso nuevo:

- nuevo proceso de lotes
- Sesión interactiva: usuario entra al sistema desde terminal.
- Creado por el SO: para realizar alguna función que necesite de él.
- Creado por proceso existente (spawning). El que lo lanza es llamado proceso padre y el otro proceso hijo. Necesitan comunicación.

- TERMINACIÓN DE PROCESOS:

Todo sistema debe proporcionar los mecanismos mediante los cuales un proceso indica su finalización. Razones para terminar un proceso:

-finalización normal, limite de tiempo excedido, memoria no disponible, violaciones de frontera (memoria sin acceso permitido), errores, limite de tiempo (espero demasiado tiempo por un evento), fallo de E/S, instrucción no válida, intervención del SO, terminación del proceso padre, solicitud proceso padre.

- MODELO DE 5 ESTADOS:

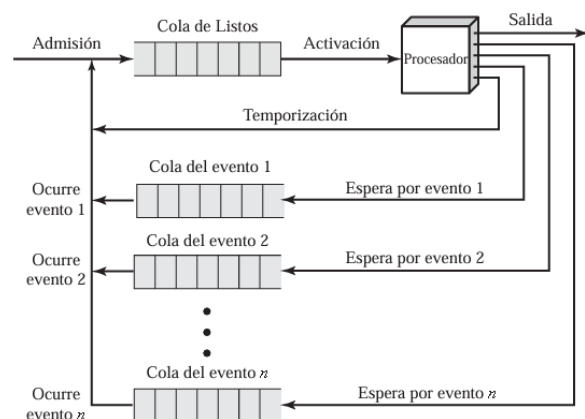
Usando una única cola el activador no puede seleccionar únicamente los procesos que lleven más tiempo en la cola (porque pueden estar bloqueados). Debería recorrer la misma.

Para hacer esto se divide el estado No Ejecutando en: Listo y Bloqueado. Ejecutando, listo, bloqueado, nuevo, saliente.

Bloqueado: un proceso no se puede ejecutar hasta que ocurra algún evento determinado.



Al tener una cola de bloqueados por evento es mucho más eficiente. Cuando sucede el evento de una fila, todos los procesos de esa fila pasarán automáticamente a Listos.



- PROCESOS SUSPENDIDOS:

La necesidad del swapping. La memoria principal puede expandirse para acomodar nuevos procesos, pero es caro y los programas ocupan más espacio. **La solución eficaz es el swapping: implica mover parte o todo el proceso de memoria principal al disco.**

Cuando ninguno de los procesos en la mm se encuentre en Listo, el SO intercambia uno de los bloqueados a disco en la cola de Suspendidos (lista de procesos existentes que han sido expulsados de la mmpp o suspendidos). El espacio liberado puede usarse para traer otro proceso.

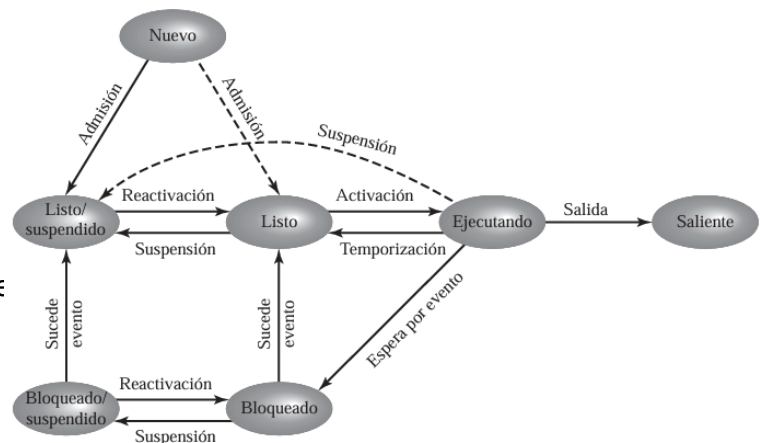
Cuando realiza el swap el SO puede seleccionar dos opciones para seleccionar un nuevo proceso: nuevo proceso que se haya creado o un proceso en suspendido. Pero todos los suspendidos anteriormente estaban en bloqueado, entonces puede pasar:

- Listo: proceso en mm para ejecutar
- Bloqueado: proceso en mm bloqueado
- Bloqueado/Suspendido: proceso en disco y bloqueado
- Listo/Suspendido: proceso en disco, ya se produjo el evento. La información de estado debe ser accedida por el SO.

- DIAGRAMA DE 7 ESTADOS:

Razones de suspensión:

- Swapping (liberar memoria).
- Otras razones (problemas).
- Solicitud interactiva del usuario.
- Temporización (espera el siguiente intervalo de ejecución).
- Solicitud del proceso padre.



2.4 DESCRIPCIÓN DE PROCESOS

Se piensa en el SO como entidad que gestiona los recursos del sistema por parte de los recursos.

SO se encarga de la gestión de procesos y recursos, debe tener información sobre el estado de cada proceso y recurso. Por eso construye y mantiene tablas de información de cada entidad que gestiona.

- Tablas de memoria: guardan un registro de la memoria principal y secundaria, incluyen: reservas de mm pp y secundaria por parte de los procesos, todos los atributos de protección e información para manejar la memoria virtual.
- Tablas de E/S: gestionar los dispositivos y canales del computador.
- Tablas de fichero: información sobre ficheros, posición en almacenamiento y otros atributos.
- Tablas de procesos: gestiona los procesos, examina los requisitos que tienen las tablas de proceso

- ESTRUCTURAS DE CONTROL:

El SO debe conocer donde están localizados los procesos y los atributos de los procesos.

- Localización de los procesos:

Un proceso debe incluir un programa o un conjunto de ellos. Estos tienen asociados posiciones de memoria para las variables. La ejecución de un programa incluye una pila, registra las llamadas a procedimientos y parámetros.

Cada proceso está asociado a un número de atributos usados por el SO para controlar al proceso, al conjunto de estos atributos se le llama bloque de control del proceso (BCP). Conjunto de programa, datos, pila y atributos = **imagen del proceso**, que sus elementos son:

Datos del usuario: datos del programa, área de pila de usuario y programas que puedan ser modificados.	Programa de usuario: programa a ejecutar.
Pila de sistema: proceso tiene una o más pilas de sistema para almacenar parámetros y direcciones.	BCP: datos necesarios para que el sistema operativo pueda controlar los procesos.

Normalmente se mantiene como un bloque de memoria contiguo y en memoria secundaria y para gestionar el proceso se debe cargar la imagen en memoria principal/virtual

- Atributos de proceso:

SO requiere mucha información para manejar cada proceso, la cual reside en el BCP. La información que está en el BCP se divide en 3 categorías:

IDENTIFICACIÓN DE PROCESO Identificadores: identificador del proceso (índice), identificador del proceso padre, identificador del usuario responsable del trabajo
INFORMACIÓN DEL ESTADO DEL PROCESADOR (indica los contenidos de los registros del procesador) Registros visibles por el usuario Registros de estado y control: PC, códigos de condición, información de estado. Puntero de pila
INFORMACIÓN DE CONTROL DE PROCESO Información adicional necesaria para que el SO controle y coordine los procesos.

El BCP es la estructura más importante del SO. Contiene toda la información sobre un proceso que necesita el SO. El conjunto de BCP define el estado del SO. EL BCP es parte de la imagen.

2.5 CONTROL DE PROCESOS

- MODOS DE EJECUCIÓN:

Antes de continuar sobre cómo se gestiona el proceso se deben distinguir entre los modos de ejecución de un procesador.

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

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

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

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

El motivo del uso de los otros modos es porque se debe proteger al SO y a las tablas como BCP. En Kernel el software tiene control completo sobre el procesador y de sus instrucciones, registros y memoria; no es recomendable para nivel de programas de usuario. Para determinar el modo de ejecución hay un bit en el estado de programa (PSW). Este bit puede cambiar frente a diferentes eventos: cuando un usuario realiza una llamada a un servicio del sistema operativo o cuando una interrupción dispara la ejecución de una rutina del sistema operativo, el modo se cambia a modo núcleo y luego vuelve al modo inicial.

- FUNCIONES TÍPICAS DE UN KERNEL:

<i>Administración de procesos</i> <ul style="list-style-type: none">• Creación y terminación de procesos.• Planificación y despacho de procesos.• Conmutación de procesos.• Sincronización de procesos y soporte para comunicación entre procesos.• Administración de BCP.	<i>Manejo de memoria</i> <ul style="list-style-type: none">• Asignación del espacio de direcciones a los procesos.• Swapping.• Manejo de segmentos y páginas.
<i>Manejo de E/S</i> <ul style="list-style-type: none">• Manejo de buffer.• Asignación a procesos de canales de E/S y dispositivos.	<i>Funciones de soporte</i> <ul style="list-style-type: none">• Manejo de interrupciones.• Contabilidad• Monitoreo.

- CAMBIO DE MODO:

Cuando hay una interrupción pendiente:

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

Luego el procesador busca la primera instrucción del programa de manejo de interrupción. El contexto del proceso que se interrumpe se guarda en el BCP. Luego de atender la interrupción puede reanudar el proceso o elegir otro. Generalmente la existencia de una interrupción no implica necesariamente un cambio de proceso.

- CAMBIO DE PROCESO:

Puede ocurrir en cualquier momento en el que el SO obtiene el control sobre el proceso en ejecución. Los mecanismos de interrupción son:

- Interrupción: externa a la ejecución del proceso. Reacción ante un evento externo asíncrono.
- Trap: asociada a la ejecución de la instrucción actual. Manejo de error.
- Llamada al sistema: solicitud explícita. Llamada a una función del SO.

El cambio de proceso que implica un cambio del estado, requiere un mayor esfuerzo que un cambio de modo.

2. 6 PROCESOS E HILOS

Por ejemplo, si yo tengo una computadora con 4 núcleos (puede ejecutar 4 procesos a la vez) y cada núcleo tiene 2 hilos (puede tener 1 o 2 hilos), por lo que si yo tengo 4 núcleos de 2 hilos cada núcleo voy a tener hasta 8 procesos de los cuales solo 4 están en estado de ejecución (1 por cada núcleo).

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

- Unidad que posee recursos: A un proceso se le asigna un espacio de memoria y, de tanto en tanto, se le puede asignar otros recursos como dispositivos de E/S o ficheros.
- Unidad a la que se le asigna el procesador: Un proceso es un flujo de ejecución (una traza) a través de uno o más programas. Esta ejecución se entremezcla con la de otros procesos. De tal forma, que un proceso tiene un estado (en ejecución, listo, etc) y una prioridad de expedición u origen. La unidad planificada y expedida por el sistema operativo es el proceso.

En la mayoría de los sistemas operativos, estas dos características son, de hecho, la esencia de un proceso. Sin embargo, son independientes, y pueden ser tratadas como tales por el sistema operativo.

Esta distinción ha conducido en los sistemas operativos actuales a desarrollar la construcción conocida como thread.

Si se tiene esta división de características, entonces:

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

Dentro de un proceso puede haber uno o más hilos de control, CONTENIDO DE HILO:

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

-MULTIHILO:

Capacidad de dar soporte a múltiples hilos de ejecución en un solo proceso.

Todos los hilos de un mismo proceso comparten el estado y recurso de ese proceso. Mismo espacio de direcciones y acceso a los mismos datos.

Ventajas: Lleva menos tiempo crear o finalizar un hilo que un proceso nuevo; lleva menos tiempo cambiar entre 2 hilos; los hilos mejoran la eficiencia de la comunicación (no necesitan del Kernel).

Existen acciones que afectan a los hilos de un proceso y que el SO debe gestionar. Ya que todos los hilos están conectados, todos se suspenden al mismo tiempo, al finalizar el proceso, finaliza todos los hilos.

- ESTADOS Y FUNCIONES DE LOS HILOS:

Si se expulsa un proceso, todos sus hilos se expulsan ya que comparten el espacio de direcciones del proceso. No tiene sentido suspender un hilo porque es para procesos. Sus estados son: ejecutando, listo y bloqueado. Sus funciones asociadas a un cambio de estado son:

- Creación: se crea un nuevo proceso, también un hilo. Luego un hilo puede crear otro dentro.
- Bloqueo: El procesador puede ejecutar otro hilo por estar bloqueado dentro del mismo proceso o no.
- Desbloqueo: se desbloquea el hilo.
- Finalización: completa un hilo

- HILOS DE NIVEL USUARIO Y KERNEL:

- Hilos nivel de usuario:

La aplicación gestiona todo el trabajo de los hilos y el *núcleo no es consciente de la existencia de ellos*. Los hilos los maneja la biblioteca de la app.

Ventajas: cambio de hilo no requiere al núcleo, planificación por parte de la app, cambio de hilo es más rápido, se ejecuta en cualquier SO

Desventajas: si se bloquea un hilo, se bloquea el proceso. No aprovecha múltiples procesadores.

- Hilos nivel kernel:

Núcleo gestiona todo el trabajo de hilos.

Ventajas: aprovecha múltiples procesadores y paralelismo.

Desventajas: lentos, cada vez que se quiere cambiar de hilo se cambia el modo.

2. 7 PLANIFICACIÓN

El objetivo de planificación de procesos es asignar procesos a ser ejecutados por el procesador o procesadores a lo largo del tiempo, de forma que se cumplan los objetivos del sistema tales como el tiempo de respuesta, rendimiento y eficiencia del procesador. La planificación afecta al rendimiento del sistema porque determina qué proceso esperará y cual progresará.

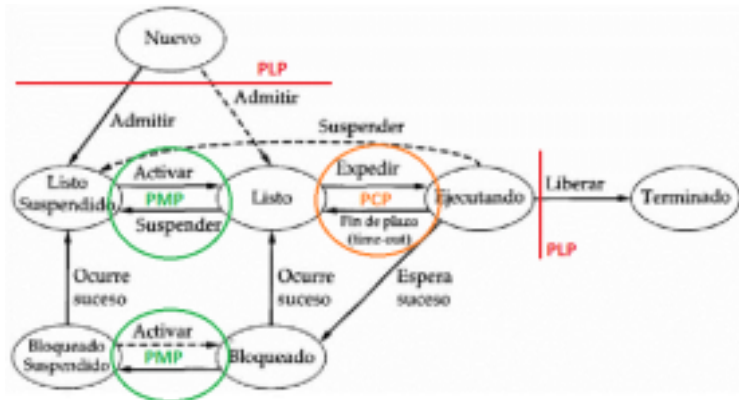
- Planificación a largo plazo: determina qué programas se admiten en el sistema para su procesamiento. Controla el grado de multiprogramación (distintos procesos en una máquina). Una vez admitido se añade a la cola del planificador a corto plazo. Actúa sobre la cola de nuevo.

La decisión de cuándo crear un nuevo proceso se toma dependiendo del grado de multiprogramación deseado. Mientras más procesos se crean, menos es el tiempo para cada uno.

- Planificación a medio plazo: parte del swapping. La decisión del intercambio se basa en la necesidad de gestionar el grado de multiprogramación. Un sistema que no utiliza la

memoria virtual la gestión de la memoria es un aspecto a tener en cuenta. Actúa sobre colas de suspendido, listo y bloqueado.

- Planificación a corto plazo (activador, despachador): toma las decisiones de grano fino sobre qué proceso ejecutar al siguiente. Se invoca siempre que ocurre un evento que puede conllevar el bloqueo del proceso actual y que puede proporcionar la oportunidad de expulsar al proceso actualmente en ejecución en favor de otro (interrupción de reloj o E/S, llamada al sistema, señales).



2. 8 ALGORITMOS DE PLANIFICACIÓN

- USO DE PRIORIDADES:

A cada proceso se le asigna una prioridad y el planificador siempre elegirá el proceso de mayor prioridad y en lugar de una sola cola de listos, hay múltiples colas de listo, una para cada prioridad. Cuando se va a realizar una selección en la planificación comienza con la cola de listos de prioridad más alta. Si hay uno o más procesos en la cola, se selecciona uno utilizando alguna política de planificación.

El problema de la prioridad es que los de prioridad más baja pueden sufrir inanición. Igualmente, la prioridad de un proceso puede cambiar con su antigüedad o historial de ejecución.

- POLÍTICAS DE PLANIFICACIÓN:

La función de selección determina qué proceso (entre los listos) se va a ejecutar. Puede estar basada en prioridades, requisitos sobre los recursos o características de ejecución del proceso. Para esta última es necesario saber:

- w = tiempo usado en el sistema hasta ese momento.
- e = tiempo usado de ejecución hasta ese momento.
- s = tiempo total requerido por el proceso.

El modo de decisión especifica los instantes de tiempo en que se ejecuta la función de selección. Dos categorías:

- Sin expulsión: una vez que el proceso está en estado Ejecutando, continua hasta que **termina o se bloquea**.
- Con expulsión: proceso Ejecutando puede ser interrumpido y pasado a listo por el SO. Tienen mayor sobrecarga, pero proporcionan mejor servicio.

- TIPOS DE PLANIFICACIÓN A CORTO PLAZO:

Apropiativa = Expulsión

<p>1. Primero en llegar, primero en servirse (FCFS):</p> <p>Primero que entra primero en salir. No importan las prioridades (a menudo se combinan). Favorece a los procesos largos. Problemático para sistemas de tiempo compartido por no ser con desalojo. Favorece procesos de uso CPU.</p> <p><i>No hay inanición.</i></p> <p><i>Sin expulsión</i></p>	<p>2. Turno rotatorio (Round Robin):</p> <p>Darle igual tiempo de ejecución a cada proceso (timer). El proceso pasa a listo y se selecciona el próximo con FCFS. Sobrecarga debido al manejo del reloj y planificación.</p> <p><i>No hay inanición.</i></p> <p><i>Con expulsión.</i></p>
<p>3. Primero el proceso más corto (SPN):</p> <p>Manda a ejecutar el proceso más corto que esté en la lista de "Listos" (se coloca en el frente de la cola). Un problema es la necesidad de saber el s de cada proceso.</p> <p>Favorece procesos E/S.</p> <p><i>Puede haber inanición para procesos largos.</i></p> <p><i>Sin expulsión.</i></p>	<p>4. Menor tiempo restante (SRT):</p> <p>El que menos le queda de s tiene prioridad. Esto se evalúa cada vez que aparece un proceso nuevo (si se está ejecutando un proceso que le quedan 2 de 10 unidades de tiempo y aparece uno de 3 unidades, se sigue ejecutando el de 10 ya que le queda menos). Sobrecarga almacenar los tiempos. Favorece a procesos E/S</p> <p><i>Riesgo inanición para procesos largos.</i></p> <p><i>Con expulsión.</i></p>
<p>5. Primero el de mayor tasa de respuesta (HRRN):</p> <p>Elige al que tiene mayor tasa de respuesta, mediante la siguiente fórmula: $R = (w+s)/s$ Tiene prioridad los procesos cortos pero los largos no tienen inanición ya que el que más espera más prioridad tiene. Favorece procesos E/S.</p> <p><i>No hay inanición.</i></p> <p><i>Sin expulsión.</i></p>	<p>6. Retroalimentación (feedback):</p> <p>Si no se puede conocer el tiempo s no se pueden utilizar los otros. Contra más tiempo se estuvo ejecutando menos prioridad tendrá (castiga los más largos). Cada vez que se expulsa un proceso, baja de prioridad. Ya en la última se queda ahí (turno rotatorio).</p> <p><i>Los procesos largos pueden llegar a tener inanición.</i></p> <p><i>Con expulsión.</i></p>

UNIDAD 3: CONCURRENCIA

La concurrencia abarca la comunicación y compartición o competencia por recursos, la sincronización de actividades de múltiples procesos y la reserva de tiempo de procesador para los procesos. Aparece en 3 contextos:

- Múltiples aplicaciones, aplicaciones estructuradas y estructura del SO.

Requisito básico para que exista la concurrencia: ***hacer imperar la exclusión mutua*** (capacidad de impedir a cualquier otro proceso realizar una acción mientras se le haya permitido a otro).

Conceptos básicos:

Sección crítica: parte del programa donde se declara el uso de un recurso.	Interbloqueo: situación donde dos o más procesos son incapaces de actuar porque cada uno está esperando que el otro haga algo.
Condición de carrera: múltiples hilos o procesos leen y escriben un dato compartido y el resultado final depende del orden de las ejecuciones.	

- DIFICULTADES DE MONO Y MULTIPROCESADOR:

- Compartición de recursos globales.
- Complicado para el SO gestionar la asignación de recursos de manera óptima.
- Complicado localizar errores de programación.

- ASPECTOS QUE SURGEN POR LA CONCURRENCIA:

- SO debe seguir la pista de varios procesos con el BCP.
- SO debe ubicar y desubicar varios recursos para cada proceso activo. Recursos son: tiempo de procesador, memoria, ficheros y dispositivos de E/S.
- SO debe proteger datos y recursos físicos frente a interferencias de otros procesos.
- Funcionamiento de un proceso y el rdo que produzca debe ser independiente a la velocidad de su ejecución en relación con procesos concurrentes.

- INTERACCIÓN ENTRE PROCESOS:

Grado de percepción	Relación	Influencia que un proceso tiene sobre el otro	Potenciales problemas de control
Procesos que no se perciben entre sí	Competencia	<ul style="list-style-type: none">• Los resultados de un proceso son independientes de la acción de los otros• La temporización del proceso puede verse afectada	<ul style="list-style-type: none">• Exclusión mutua• Interbloqueo (recurso renovable)• Inanición
Procesos que se perciben indirectamente entre sí (por ejemplo, objeto compartido)	Cooperación por compartición	<ul style="list-style-type: none">• Los resultados de un proceso pueden depender de la información obtenida de otros• La temporización del proceso puede verse afectada	<ul style="list-style-type: none">• Exclusión mutua• Interbloqueo (recurso renovable)• Inanición• Coherencia de datos
Procesos que se perciben directamente entre sí (tienen primitivas de comunicación a su disposición)	Cooperación por comunicación	<ul style="list-style-type: none">• Los resultados de un proceso pueden depender de la información obtenida de otros• La temporización del proceso puede verse afectada	<ul style="list-style-type: none">• Interbloqueo (recurso consumible)• Inanición

3.1 EXCLUSIÓN MUTUA REQUISITOS

Cualquier técnica que otorgue exclusión mutua debe cumplir:

1. La exclusión mutua debe cumplirse.
2. Un proceso en su sección no crítica, no debe interferir con otros procesos.
3. No puede producirse interbloqueo e inanición cuando se solicite entrar a la SC.
4. Si no hay ningún proceso en SC, no debe haber demoras cuando alguno lo pida.
5. No se deben suponer velocidades.
6. El proceso puede permanecer en su SC de forma finita.

Herramientas para la concurrencia / conseguir exclusión mutua: hardware, semáforos, monitores, paso de mensajes.

3.2 EXCLUSIÓN MUTUA: HARDWARE

El software es fácil que tenga una alta sobrecarga de procesamiento. Soluciones hardware a la exclusión mutua:

- DESHABILITAR INTERRUPTACIONES:

La máquina monoprocesador garantiza la exclusión mutua. **Se limita la capacidad del procesador y se deshabilita la multiprogramación. No funciona en multiprogramación.**

- INSTRUCCIONES MÁQUINA ESPECIALES:

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

Test and set. Se inicializa en $i = 0$:

Proceso que entra a su sc: Aquel que encuentra ese $i == 0$. Al entrar lo establece en $i = 1$. Al salir lo establece en $i == 0$.	Proceso que no encuentra $i == 0$: Espera activa: no puede hacer nada pero sigue ejecutando instrucciones para chequear la variable. Cuando un proceso sale entra solo uno (quien ejecuta testset próximo).
--	---

Exchange.

Intercambio de llaves como los baños de una estación de servicio.

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

Si el cerrojo = 0 -> no hay procesos en su SC.

Si el cerrojo = 1 -> hay proceso en su sección crítica, el que tenga la variable llave = 1.

Desventajas Espera activa del procesador revisando el estado del recurso. Puede haber inanición ya que no hay cola. Puede haber interbloqueo	Ventajas Aplicable a cualquier número de procesos sobre mono o multi. Simple. Para múltiples secciones críticas.
--	--

3.3 EXCLUSIÓN MUTUA: SEMÁFORO

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

- SEMÁFORO GENERAL/NO BINARIO:

Tiene definido 3 operaciones:

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

- SEMÁFORO BINARIO (0 y 1):

Tiene definido 3 operaciones:

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

```

/* programa exclusión mutua */
const int n = /* numero de procesos */;
semaphore s = 1;
void P(int i)
{
  while (true)
  {
    semWait(s);
    /* sección crítica */;
  }
}

```

```

semSignal(s);
/* resto */;
}
}
void main()
{
  parbegin (P(1), P(2), . . . , P(n));
}

```

3.4 PROBLEMA PRODUCTOR/CONSUMIDOR

Se utiliza un buffer para amortiguar las diferencias de velocidades.

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

Problemas Debe haber exclusión mutua, sólo uno puede estar en el buffer. El consumidor no puede consumir si el buffer está vacío $n=0$. Buffer finito.	Soluciones Semáforo no binario s para exclusion mutua Semáforo no binario n , para saber si hay elementos almacenados sin leer. Semáforo no binario e , para saber si hay espacio libre en el buffer.
---	---

```

/* program productorconsumidor */
semaphore n = 0;
semaphore s = 1;
semaphore e = tamaño_buffer;
void productor()
{
  while (true)
  {
    produce();
    wait(e);
    wait(s);
    agrega();
    signal(s);
    signal(n);
  }
}

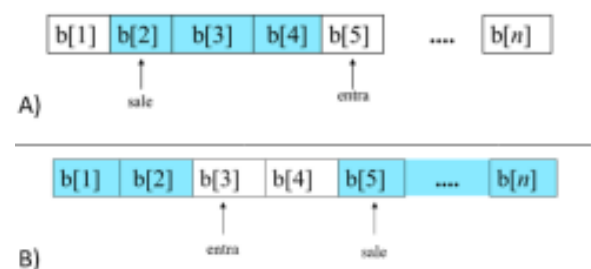
void consumidor()
{
  while (true)
  {
    wait(n);
    wait(s);
    toma();
    signal(s);
    signal(e);
    consume();
  }
}

void main()
{
  parbegin (productor,
            consumidor);
}

```

buffer acotado general

productor	consumidor
<pre> while (true) { /* producir dato v */ while ((entra + 1) % n == sale) /* no hacer nada */; b[entra] = v; entra = (entra + 1) % n } </pre>	<pre> while (true) { while (entra <= sale) /* no hacer nada */; w = b[sale]; sale++; /* consumir dato w */ } </pre>



3.5 EXCLUSIÓN MUTUA: MONITORES

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

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

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

- Módulo de software.
- Un proceso entra en el monitor invocando uno de sus procedimientos.
- Sólo un proceso puede estar ejecutando dentro del monitor al mismo tiempo (exclusión mutua).
- Si el proceso se bloquea, el monitor es liberado para otro proceso. Luego, el primer proceso puede retomar el monitor justo donde lo dejó.
- Variables condición: Variables locales de datos, sólo accesibles por el monitor. Se manipulan con cwait() y csignal().
- ❖ Función cwait(c): Suspende al proceso con la condición "c". El proceso no sale fuera del monitor, sino que pasa a la cola de bloqueados de la condición, que se ubica en la zona de espera del monitor.
- ❖ Función csignal(c): Retoma la ejecución de algún proceso bloqueado con la condición "c". Si no hay ningún bloqueado, esta señal se pierde y no se hace nada.

Las condiciones y las funciones son las que ayudan a la sincronización de los procesos, bloqueando unos y desbloqueando otros (no lleno = falso, entonces el productor se bloquea).

```

/* program productorconsumidor */
monitor bufferacotado;
char buffer[N]; /* espacio para N items */
int nextin, nextout; /* punteros buffer */
int cont; /* número de items en buffer */
cond nolleno, novacio; /* para sincronización*/
void agrega (char x)
{
    if (cont == N)
        cwait(nolleno); /* buffer lleno; espere */
    buffer[nextin] = x;
    nextin = (nextin + 1) % N;
    cont++; /* un item más en buffer */
    csignal(novacio); /* reanudar consumidor que
    espera */
}

void productor()
{
    char x;
    while (true)
    {
        produce(x);
        agrega(x);
    }
}

void consumidor()
{
    char x;
    while (true)
    {
        extrae(x);
        consume(x);
    }
}

void main()
{
    parbegin (productor, consumidor);
}

```

3.6 EXCLUSIÓN MUTUA: PASO DE MENSAJES

Procesos que se comunican entre sí. Cuando lo hacen se deben satisfacer dos requisitos: sincronización (para exclusión mutua) y comunicación (intercambiar información). Enfoque que proporciona ambas: paso de mensajes. Se puede implementar en sistemas distribuidos, multiprocesadores y monoprocesadores. Hay colas, la más simple es por FIFO pero se puede especificar la prioridad de un mensaje o dejar que el receptor revise la cola y él elija.

Tiene 2 operaciones primitivas:

- send(destino, mensaje)

- receive(origen, mensaje)

- SINCRONIZACIÓN:

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

- DIRECCIONAMIENTO:

Es necesario tener una manera de especificar en send qué procesos deben recibir el mensaje. Esquema que usan las dos primitivas:

- DIRECTO: send incluye un id específico del proceso destinatario.
 - receive pueden ser dos maneras:
 - proceso deba designar explícitamente un proceso emisor (explícito, eficaz para procesos concurrentes operantes)
 - parámetro origen toma el valor devuelto por la op cuando se completa (implícito)
- INDIRECTO: mensajes no se envían directamente entre procesos. Son enviados a una estructura de datos compartida que son colas que pueden tener mensajes temporalmente (mailboxes). Un proceso envía un mensaje al buzón y el otro lo toma de ahí. Mayor flexibilidad. Un mailbox es un puerto si hay un único receptor.

- ASOCIACIÓN DE PUERTOS Y PROCESOS:

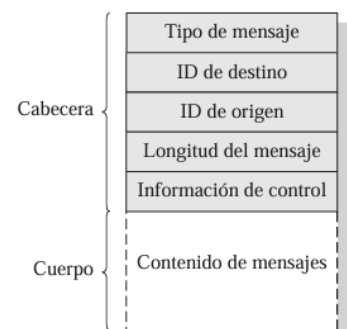
Puede ser estática o dinámica.

- Estática: puertos se asocian con proceso en particular (se crea y se asigna a un proceso permanente).
- Dinámica: muchos emisores Propiedad del buzón.
- Puerto: creado y propiedad del proceso receptor. Destruye proceso, destruye puerto.
- Buzón: propiedad del proceso que lo crea o del SO. Mandato para destruirlo.

- FORMATO DE MENSAJE:

Depende de los objetivos de la facilidad de mensajería. Si se van a transferir muchos datos, los datos pueden estar en archivos y el mensaje puede indicar el archivo. O permitir mensajes de longitud variable.

- Cabecera: información acerca del mensaje (identificación del origen y destinatario, campo de longitud y campo de tipo para discriminar entre tipos de mensajes). También información de control (punteros para crear lista de mensajes, prioridad, o número de secuencia para llevar la cuenta)
- Cuerpo: contenido real del mensaje.



```

/* programa productor consumidor */
const int
    capacidad = /* capacidad de
almacenamiento */;
    null = /* mensaje vacío */;
int i;
void productor()
{
    message pmsg;
    while (true)
    {
        receive (puedeproducir, pmsg);
        producir();
        send (puedeconsumir, pmsg);
    }
}

void consumidor()
{
    message cmsg;
    while (true)
    {
        receive (puedeconsumir, cmsg);
        consumir();
        send (puedeproducir, cmsg);
    }
}

void main()
{
    create_mailbox(puedeproducir);
    create_mailbox(puedeconsumir);
    for (int i = 1; i <= capacidad; i++)
        send(puedeproducir, null);
    paralelos (productor, consumidor);
}

```

Sigue → }

3.7 INTERBLOQUEO

Interbloqueo: bloqueo permanente de un conjunto de procesos que compiten por el mismo recurso o se comunican entre sí. Cuando cada proceso del conjunto está bloqueado esperando un evento que genera otro proceso que se bloquee.

Es permanente y no hay soluciones eficientes. Involucran necesidades conflictivas que afectan los recursos de dos o más procesos.

- Recursos reutilizables
Solo lo utiliza de forma segura un proceso en cada momento y no se destruye después de su uso, (procesadores, canal e/s, mm pp y secundaria, dispositivos, ficheros, semáforos y bd). Interbloqueo se produce si cada proceso mantiene un recurso y solicita el otro.
- Recursos consumibles
Puede crearse y destruirse, no hay límite. (interrupciones, msjs, información en buffers de e/s).
- Grafos de recursos
Herramienta para caracterización de asignación de recursos a los procesos. Grafo representa el estado del sistema en recursos y procesos. Cada proceso y cada recurso se representa con un nodo. (puntos por cada ejemplar del recurso).



3.8 CONDICIONES NECESARIAS PARA INTERBLOQUEO

1. **Exclusión mutua.**
2. **Retención y espera:** proceso puede mantener los recursos asignados en espera de otros.
3. **Sin expropiación:** no se puede forzar la expropiación de un recurso.
4. **Espera circular:** lista cerrada de procesos, de tal manera que cada proceso posee al menos un recurso necesitado por el siguiente en la lista.

Región fatal: región en la que una vez que los procesos ingresan a esta región se verán involucrados en un interbloqueo. Existe si se cumplen las 3 primeras.

Para que exista interbloqueo no solo debe haber una región fatal si no una secuencia de peticiones de recursos que conduzca a la región fatal (4).

1-3 necesarias y no suficientes. 4 suficiente y necesaria.

Estrategias para tratar: prevención, predicción, detección.

3.9 INTERBLOQUEO: PREVENCIÓN

Poco eficiente, pero efectiva (antes y sin saber que pasara).

Diseñar sistema de manera que excluya la posibilidad de interbloqueo, restringiendo las solicitudes de recursos. Pueden ser dos categorías:

- Método indirecto: impedir la aparición de una de las 3 condiciones necesarias.
- método directo: impide la 4 condición.

Técnicas relacionadas con las 4 condiciones:

- exclusión mutua: **no puede eliminarse**. SO debe proporcionar.
- retención y espera: proceso debe solicitar al mismo tiempo todos sus recursos necesarios, bloqueandolo hasta que se le puedan conceder todos. *Inanición, recursos asignados pueden permanecer inutilizados.*
- sin expropiación: proceso que tiene varios recursos se le niega la petición posterior, debe liberar todos los recursos y luego puede solicitarlos de nuevo con el adicional. Si solicita uno que lo esté usando otro, el SO puede expropiar al segundo obligándolo a liberar los recursos (solo si no tienen misma prioridad).
- espera circular: definiendo un orden lineal entre los distintos tipos de recursos. Si se asignaron recursos tipo R solo puede pedir luego recursos de orden posterior.

3.10 INTERBLOQUEO: PREDICCIÓN

Permite las 3 condiciones necesarias pero toma decisiones razonables para asegurarse que nunca se llegue el interbloqueo. Se decide dinámicamente si una petición de reserva de un recurso podría causar un interbloqueo. Requiere conocimiento de las futuras solicitudes de un recurso. Dos técnicas:

- no iniciar proceso si sus demandas pueden llevar a un interbloqueo.
- no conceder una petición (algoritmo de banquero).

- ALGORITMO DE BANQUERO:

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

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

Desventajas Procesos deben establecer anticipado los recursos necesarios (muy malo). Procesos deben ser independientes. Debe haber un número fijo de recursos. Ningún proceso puede terminar mientras tenga recursos	Ventajas No hay expropiación Más libertades
---	--

3.11 INTERBLOQUEO: DETECCIÓN

No limita el acceso a los recursos. Encuentra el interbloqueo ya existente. Es la más eficiente en rendimiento del sistema porque a los procesos se le dan los recursos que van pidiendo y cada tanto (de forma sincrónica) se ejecuta el algoritmo para solucionar los interbloqueos formados.

El problema es que esta estrategia mata a los procesos, por lo tanto, no se puede usar en cualquier sistema.

- RECUPERACIÓN DE PROCESOS:

Una vez detectado se necesita una estrategia de recuperacion (sofisticacion creciente):

1. Abortar todos los procesos del interbloqueo.
2. Retroceder cada proceso en interbloqueo a un checkpoint, puede repetirse el interbloqueo.
3. Abortar sucesivamente los procesos en el interbloqueo hasta que deje de existir. Orden de elección de procesos debe ser según a un criterio que implique un costo mínimo.
4. Expropiar sucesivamente los recursos hasta que el interbloqueo deje de existir. Selección basada en el coste, el proceso expropiado debe retroceder.

Criterios de selección de procesos (para 3 y 4).

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

UNIDAD 4: MEMORIA

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

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

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

Fragmentación externa: Es generada cuando durante el reemplazo de procesos quedan huecos entre dos o más procesos de manera no contigua y cada hueco no es capaz de soportar ningún proceso de la lista de espera. Una solución es la compactación, pero consume procesador y tiempo.	Fragmentación interna: Es generada cuando se reserva más memoria de la que el proceso va realmente a usar. Estos huecos no se pueden compactar para ser utilizados.	Superposición: técnica para que un proceso pueda ser mayor que la cantidad de memoria que se le ha asignado. Busca mantener en la memoria sólo las instrucciones y datos que se necesitan en un momento dado. Si se requieren otras instrucciones, se cargan en un espacio que antes estaba ocupado por instrucciones que ya no se necesitan.
---	---	---

4.1 REQUISITOS GESTIÓN DE MEMORIA

1. 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 vuelva a ser cargado se puede necesitar reubicar el proceso en un área distinta de la memoria.

El SO debe conocer la ubicación del BCP, de la pila y el punto de partida del programa del proceso (parte de la imagen el proceso).

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

- direcciones virtuales/lógicas: direcciones a las cuales hacen referencia los procesos independientemente de la asignación actual. Lo hace el hardware junto al software.
- direcciones relativas: dirección calculada como un desplazamiento a partir de una dirección de base. Si la real empieza en la posición 40 y el programa solicita la posición 12 de la virtual, la dirección relativa 0 es 40 entonces 12 es 52, el registro base es 40.

Registro de límites: requisito de protección para que no acceda en memoria de otro proceso.

2. PROTECCIÓN:

- Los procesos no deberían poder referenciar localidades de memoria de otros procesos sin permiso.
- Imposible chequear direcciones dentro de los programas puesto que se desconoce la ubicación en memoria principal, es imposible chequear durante la compilación.
- Se debe chequear durante la ejecución por el procesador.

3. COMPARTICIÓN:

- Permitir a varios procesos acceder a la misma porción de memoria

Ejemplo: acceder a una estructura de datos compartida (procesos del mismo programa o hilos o instrucción).

4. ORGANIZACIÓN LÓGICA:

La mayoría de los programas se organizan en módulos, con diferentes grados de protección (sólo ejecución, solo lectura). Si el SO y hardware pueden tratar a los programas y datos en forma de módulos, se consiguen varias ventajas. La herramienta que mejor satisface esto es la segmentación.

5. 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 cuanto 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, intercalando entre sí según se necesite). La programación superpuesta malgasta el tiempo del programador.

4.2 PARTICIONAMIENTO DE MEMORIA

1. PARTICIONAMIENTO FIJO

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:

★ Igual tamaño:

Presentan dos dificultades: el programa puede ser demasiado grande para caber en la partición; y hay un uso de la MP muy ineficiente, ya que cualquier programa por más pequeño que sea, ocupará toda una partición. **Generando fragmentación interna.**

★ Distinto tamaño:

Disminuye los problemas anteriores pero no los elimina. Aparece el problema de dónde ubicar cada programa.

- Algoritmo de ubicación:

Para particiones del mismo tamaño es aleatorio. Si están todas ocupadas, el proceso a echar es problema de la planificación de procesos. En caso de que sean de distintos tamaños hay 2 formas:

- Cola por partición: asignar a cada proceso a la partición más pequeña dentro de la cual cabe. Minimiza fragmentación interna. Desventaja: una cola puede estar colapsada y otra no tener ninguna.
- Cola única: asignar a cada proceso la partición más pequeña disponible que quepa. Mayor fragmentación interna.

Partición Fijo	Ventajas	Desventajas
La MP se divide en un conjunto de particiones estáticas. 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.

2. PARTICIONAMIENTO DINÁMICO

Las particiones se crean dinámicamente (tamaño y cantidad), cada proceso se carga en una partición exactamente del tamaño que necesita, no más, sin fragmentación interna. El problema es que al final se encuentra una situación en la que hay varios huecos pequeños, si pasa el tiempo, se fragmenta más, esto se llama **fragmentación externa**.

Algo que se utiliza para disminuir la fragmentación se llama compactación, pero malgasta tiempo del procesador. Lo que hace es desplazar los procesos para que queden contiguos, de forma que toda la memoria disponible quede junta.

- Algoritmos de ubicación:

Eligen 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 más justo al solicitado.
- ★ Primer ajuste (first-fit): analiza la memoria desde el principio y selecciona el primer bloque disponible que sea suficientemente grande.
- ★ Siguiente ajuste (next-fit): recorre la memoria desde la última ubicación y elige el siguiente bloque disponible que sea suficientemente grande.
- ★ Peor ajuste: asigna el más grande.

El método depende de la secuencia de intercambio de procesos y del tamaño de ellos.

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

3. SISTEMA BUDDY

Se usa porque los dos particionamiento tienen desventajas. Acá, cuando hay un pedido de asignación se va dividiendo la memoria a la mitad (todos múltiplos de 2, colegas) hasta que el tamaño sea mínimo (64kb).

Cuando se desocupa un espacio y el colega está desocupado, se unen para volver al doble. No se

hace todo el tiempo porque requiere de procesador, se hace cada cierto tiempo o por pedido de gran tamaño.

4.3 PAGINACIÓN DE MEMORIA

La MP se encuentra dividida en trozos iguales de tamaño fijo, se llaman marcos. También, cada proceso está dividido en trozos de tamaño fijo y del mismo tamaño que los marcos, se llaman páginas. Cuando se introduce un proceso en la memoria, se cargan todas sus páginas en los marcos libres.

El SO mantiene una lista de los marcos libres. Cuando hay que cargar el proceso, el SO busca los marcos libres necesarios para cargar todas las páginas, no deben ser contiguos necesariamente, para lograrlo el SO mantiene una tabla de páginas para cada proceso. Cada tabla de página tiene una entrada por cada página del proceso, donde está el número del marco donde se encuentra esa página.

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).

- Disminuye la fragmentación interna (proceso de 15kb, los marcos de 4kb por lo tanto las páginas son 4kb, 4kb, 4kb y 3kb, entonces voy a tener 3 marcos al 100% y uno que me sobra 1kb, pero no se puede asignar a otro proceso).
- No hay fragmentación externa.
- Requisito de protección está incluido, ya que por el tamaño de páginas, no dan los bits para pasarse.

4.4 SEGMENTACIÓN DE MEMORIA

En este caso, la división del programa se realiza con segmentos, no es necesario que todos tengan la misma longitud, pero hay una longitud máxima. Acá la dirección lógica tiene 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 porqué estar contiguas.

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

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.

4.5 MEMORIA VIRTUAL

Conjunto residente: parte del proceso que se encuentra realmente en la MP para cualquier instante de tiempo.	Fallo de página: cuando se busca una página en la tabla, pero se encuentra en la MV.
--	--

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 (conjunto residente).

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 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. Pueden mantenerse un mayor número 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 éstas se necesitan.

4.6 PROXIMIDAD Y MEMORIA VIRTUAL

Se puede hacer un mejor uso de la memoria cargando únicamente unos pocos fragmentos, usando los fallos de página para ello. Pero, el sistema operativo debe saber cómo gestionar este esquema. Cuando el sistema operativo traiga una porción a la memoria, debe expulsar otra.

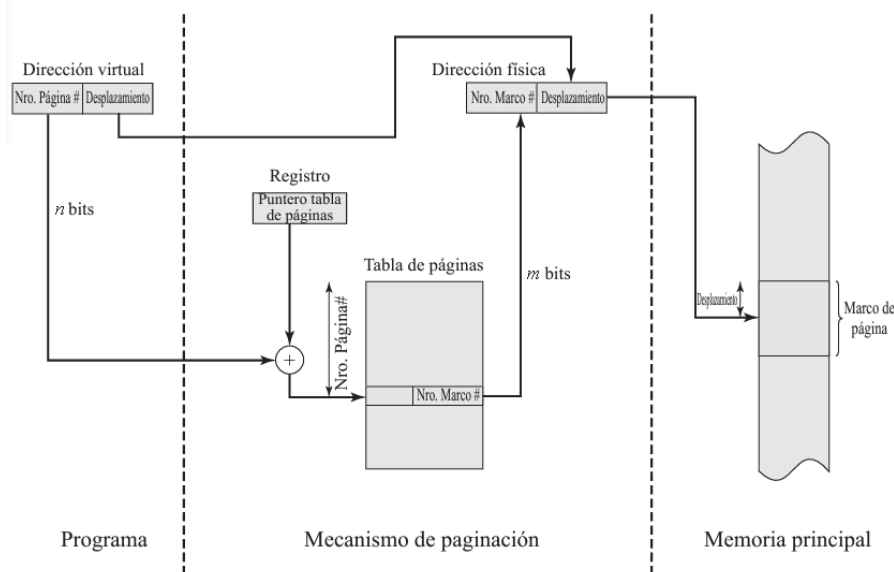
Si elimina una porción justo antes de que vaya a ser utilizada, deberá recuperar dicha porción de nuevo casi de forma inmediata. Demasiados intercambios de fragmentos lleva a una condición denominada **hiperpaginación (thrashing)**: el procesador consume más tiempo intercambiando qué ejecutando instrucciones del usuario.

Para evitarlo, el SO intenta adivinar en base a lo reciente, que porciones son las menos probables que se usen. Esto es el **principio de cercanía**. Esto indica que las referencias al programa y a los datos de un proceso tienden a agruparse y es posible adivinar inteligentemente cuales bloques se necesitaran en el futuro.

4.7 PAGINACIÓN CON MEMORIA VIRTUAL

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.



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 muy 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.

Cada referencia a una dirección virtual puede causar dos accesos a memoria física:

- uno para leer la tabla
- uno para leer el dato

Para mejorar esto se usa una caché especial para las entradas de las tablas:

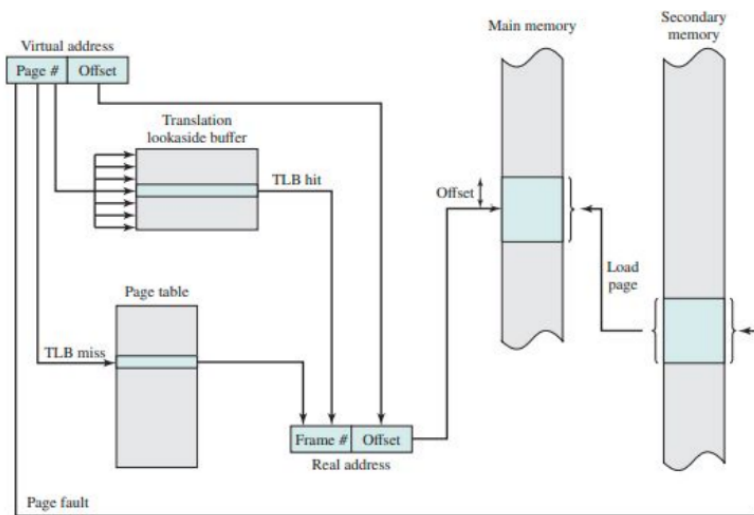
- llamado TLB (Translation Lookaside Buffer)

- TLB:

Es un buffer de traducción anticipada. Contiene las entradas de la tabla de páginas que han sido usadas recientemente.

Dada una dirección virtual, el procesador examinará primero la TLB. Si la entrada de tabla de páginas buscada está presente (un acierto en la TLB), se obtiene el número de marco y se forma la dirección real. Si no se encuentra la entrada de la tabla de páginas buscada (un fallo de TLB), el procesador emplea el número de página para buscar en la tabla de páginas del proceso y examinar la entrada correspondiente de la tabla de páginas.

Si se encuentra activo el bit de presencia, es que la página está en memoria principal y el procesador puede obtener el número de marco de la entrada de la tabla de páginas para formar la dirección real. El procesador, además, actualiza la TLB para incluir esta nueva entrada de la tabla de páginas. Por último, si el bit de presencia no está activo, es que la página buscada no está en memoria principal y se produce un fallo en el acceso a memoria, llamado fallo de página.



- TAMAÑO DE PÁGINA:

A selección del SO y hardware.

A menor tamaño:

- Menor fragmentación interna.
- Más páginas por procesos y por lo tanto tablas más grandes y posiblemente trasladadas a MV con más fallos.
- 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 mayor tamaño:

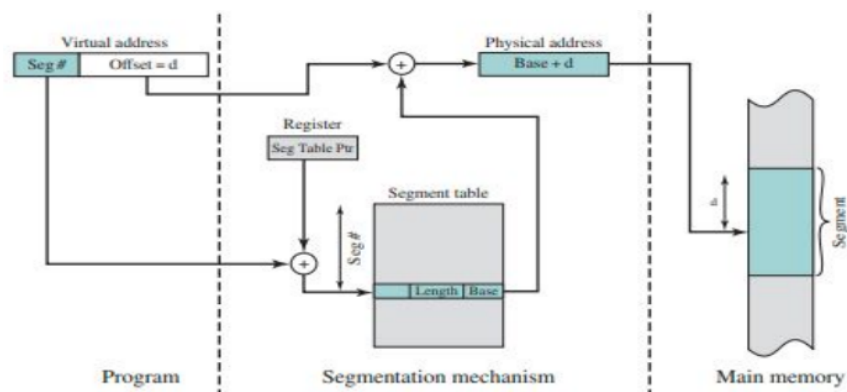
- La página en particular contendrá información más lejos de la última referencia realizada. Así pues, el efecto del principio de proximidad se debilita y la tasa de fallos de página comienza a crecer.

4.8 SEGMENTACIÓN CON MEMORIA VIRTUAL

Puede ser de tamaño diferente y dinámico. Permite el crecimiento de estructuras, modularidad y soporte. Se usa para compartir datos entre segmentos.

- TABLA DE SEGMENTOS:

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



4.9 SEGMENTACIÓN Y PAGINACIÓN CON MEMORIA VIRTUAL

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.

4.10 POLÍTICAS DE GESTIÓN DE LA MV POR EL SO

1. LECTURA/RECUPEARCIÓN:

Decisión de cuándo se debe cargar una página en MP.

- Por demanda: una página se trae a memoria sólo si se hace referencia a una posición en dicha página (es decir, fallo). Al principio habrán muchos fallos de páginas, luego por tener varias páginas cargadas y el principio de proximidad, el número de fallos baja.
- Paginación anticipada: se traen más páginas que las que se han referenciado. Trae páginas contiguas, ya que acelera los tiempos. No se puede predecir el uso, por lo tanto es ineficiente si no se usan.

Se suele usar una combinación, al principio usando paginación anticipada y luego por demanda.

2. UBICACIÓN:

Determina en qué parte de la memoria real van a estar las porciones de un proceso. En paginación y paginación con segmentación es trivial, pero en segmentación se usan best-fit, etc.

3. REEMPLAZO:

Determina entre un conjunto de páginas, cuál es la página que debe elegirse para reemplazarla. Todas las políticas de reemplazo tienen como objetivo que la página que va a eliminarse sea la que menos posibilidades tiene de uso. Mientras más elaborada sea la política, más sobrecarga habrá.

- BLOQUEO DE MARCOS:

Gran parte del núcleo del SO se almacena en marcos que están bloqueados o estructuras de control claves. No se puede eliminar una página de un marco bloqueado.

- ALGORITMOS DE REEMPLAZO:

★ OPTIMA:

Selecciona la página que tardará más tiempo en volver a ser utilizada. Genera el menor número de fallos pero es imposible de implementar ya que el SO no sabe los sucesos futuros.

★ LRU (least recently used):

Reemplaza la página que menos ha sido referenciada en el tiempo. Proporciona buenos resultados. Hay que mantener una lista con todos los usos, muy costosa y difícil de implementar.

- VARIANTE NUR (not recently used):

Utiliza dos bits, de modificación y de uso, reemplaza basándose en el uso con el bit.

★ FIFO:

Trata a los marcos de página ocupados como un buffer circular y las páginas se reemplazan mediante una estrategia cíclica de tipo round-robin. Fácil de implementar, aumenta el puntero por cada fallo de página, el problema es que la más antigua puede ser la más usada. *Básicamente elimina la más antigua.*

★ RELOJ O SEGUNDA OPORTUNIDAD:

Asocia un bit a cada marco, denominado bit de uso. Cuando una página se trae por primera vez a la memoria, el bit de uso 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 uso a 0. Cada vez que encuentra un marco con el bit de uso a 1, se reinicia este bit a 0 y se continúa.

Si alguno de los marcos del buffer tiene el bit de uso 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 todos los bits de uso a 0, parándose en la posición original, reemplazando la página en dicho marco.

Esta política es similar a FIFO, excepto que, en la política del reloj, el algoritmo saltará todo marco con el bit de uso a 1. Se aproxima al rendimiento LRU sin introducir mucha sobrecarga.

- BUFFERING DE PÁGINAS:

Para mejorar el rendimiento, una página reemplazada 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.

Las listas funcionan como caché: puedo descargar las páginas en conjunto, cada x tiempo, de forma anticipada (como la política de lectura/vaciado). Conviene mover bloques grandes cuando se realiza una operación con el disco.

Las páginas sin modificar pueden usarse si se referencian nuevamente o perderse si se asigna su marco a otra página.

4. POLITICA DE ASIGNACION / CONJUNTO RESIDENTE:

Determina cuanto lugar se les va a dar a los procesos. Factores a tener en cuenta:

- Menor cantidad de memoria para un proceso, mayor cantidad de procesos residentes. Disminuye el tiempo perdido en intercambios.
- Pequeña cantidad de páginas residentes, aumento de tasa de fallos de páginas.
- Después de una cierta cantidad de marcos asignados no hay efecto en la tasa de fallos de página.

→ ASIGNACIÓN FIJA: número fijo de marcos asignados a un proceso en forma anticipada. La página a reemplazar debe ser elegida entre los marcos asignados a ese proceso. Desventajas: si la asignación es pequeña: alto grado de fallos de páginas; si es grande: bajo grado de multiprogramación.

→ ASIGNACIÓN VARIABLE: número variable de marcos dependiendo de sus necesidades. 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 debe a que el SO debe saber cuál es el comportamiento del proceso activo por lo que produce una sobrecarga en el sistema.

- ALCANCE 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.
- Basadas en conjuntos de trabajo:

Intentan mantener el conjunto de trabajo de los procesos activos en memoria real.

Si no se mantiene:

- Sobreutilización de la memoria real (cantidad de marcos > conjunto de trabajo) (hay páginas cargadas innecesariamente).
- Hiperpaginación (cantidad de marcos < conjunto de trabajo) (muchos fallos de página).

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

5. 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 memoria secundaria sólo 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.

Ambas son ineficientes, 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.

La solución es el buffering de páginas, las páginas en la lista modificadas se escriben por lotes periódicamente y las páginas sin modificar pueden usarse si se referencian nuevamente o perderse si se asigna su marco a otra página.

6. CONTROL DE CARGA:

Cantidad de procesos en memoria principal. 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.

- SUSPENSIÓN DE PROCESOS:

Criterio de selección para los procesos que se van a sacar:

- Prioridad más baja.
- Con muchos fallos de página.
- Último proceso activado: menos posibilidad de tener el conjunto residente en memoria.
- Conjunto residente más chico: requiere el menor esfuerzo para volver a cargarse.
- Proceso más grande: se obtienen la mayor cantidad de marcos libres.
- Mayor ventana de ejecución restante.

UNIDAD 5: ENTRADA SALIDA Y ARCHIVOS

5.1 RAID

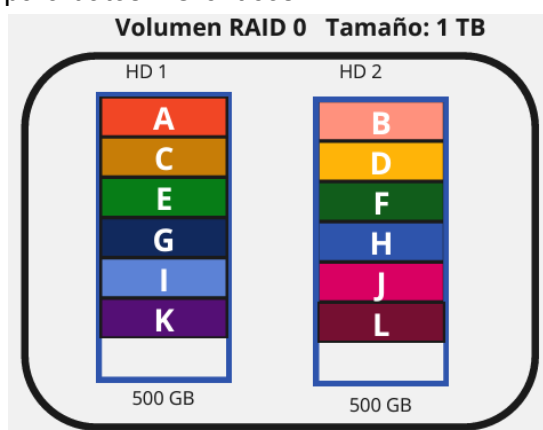
RAID (Vector Redundante de Discos Independientes) es un conjunto de unidades de disco físico vistas por el SO como una sola unidad lógica.

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.

5.2 TIPOS DE RAID SIMPLES

- RAID 0:

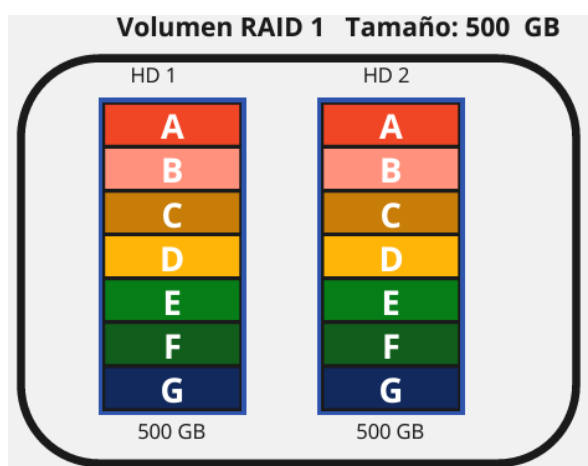
No incluye redundancia. Se usa normalmente para aplicaciones que requieren alto rendimiento para datos NO críticos.



RAID 0 Stripping		Ejemplo Vhd= 100 MB/s
Cantidad de Discos Mínima [N]:	2	
Cantidad de Discos que pueden fallar:	0	
Velocidad de Escritura Relativa a un disco:	$N \times V_{Disco}$ [MB/s]	200 MB/s
Velocidad de Escritura Ponderada [1-10] : (5 es la Velocidad de 1 disco)	10	
Velocidad de Lectura Relativa a un disco:	$N \times V_{Disco}$ [MB/s]	200 MB/s
Velocidad de Lectura Ponderada [1-10] : (5 es la Velocidad de 1 disco)	10	
Costo [\$\$-\$\$\$]	\$	

- RAID 1 (Mirroring):

Hay redundancia porque se duplican los datos. Mucha fiabilidad. Permite lecturas más rápidas ya que cualquiera de los discos que contengan los datos solicitados pueden entregarlos. La velocidad de escritura depende de la escritura más lenta, NO hay penalización por escritura. Se puede usar un Hot Spare en el caso de que uno de los discos falle, automáticamente se utiliza ese.

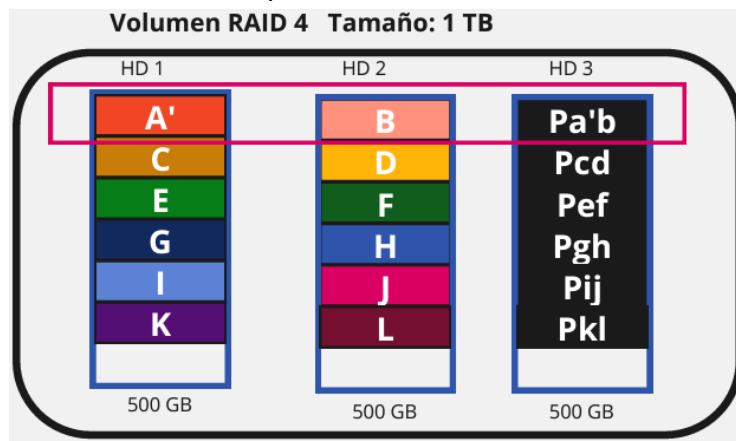


RAID 1 Mirroring		Ejemplo vhd= 100 MB/s
Cantidad de Discos Mínima:	2	
Cantidad de Discos que pueden fallar:	1	
Velocidad de Escritura Relativa a un disco:	$\sim V_{Disco}$ [MB/s]	100 MB/s
Velocidad de Escritura Ponderada [1-10] : (5 es la Velocidad de 1 disco)	5	
Velocidad de Lectura Relativa a un disco: (Siempre el peor caso)	$\sim V_{Disco}$ [MB/s]	100 MB/s
Velocidad de Lectura Ponderada [1-10] : (5 es la Velocidad de 1 disco)	5 (puede ser más por la controladora)	
Costo [\$\$-\$\$\$]	\$\$	

- RAID 4:

Del 0-3 no se usan y el 4 tampoco (porque el disco de paridades es el que más desgaste tiene, por lo que se crea el 5). No se usa porque la carga en la unidad de paridad se puede convertir en un cuello de botella.

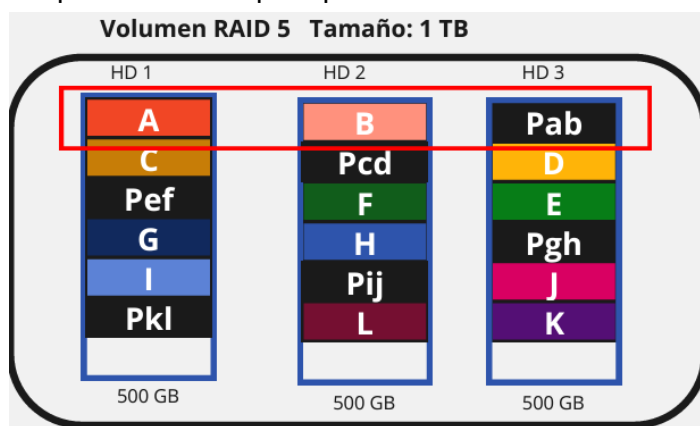
La controladora graba A' (escribe), luego lee B, calcula la paridad P(A'B) con XOR y luego la escribe en el tercer disco. Si se rompe uno de los discos, se pueden recalcul los datos con XOR mediante las paridades.



RAID 4 Block Level Striping with dedicated parity (No se Usa por falla prematura de HD)		Ejemplo Vhd= 100 MB/s
Cantidad de Discos Mínima [N]:	3	
Cantidad de Discos que pueden fallar:	1	
Velocidad de Escritura Relativa a un disco: (Cantidad de Operaciones: [N+1])	Aprox: VDisco / (Cant. Oper.) [MB/s]	25 MB/s
Velocidad de Escritura Ponderada [1-10] : (5 es la Velocidad de 1 disco)	2	
Velocidad de Lectura Relativa a un disco: (Siempre el peor caso)	VDisco [MB/s]	100 MB/s
Velocidad de Lectura Ponderada [1-10] : (5 es la Velocidad de 1 disco)	5	
Costo [\$-\$\$\$]	\$\$	

- RAID 5:

Mientras más discos, más lento. Las paridades se distribuyen equitativamente, no como el 4, mediante Round Robin. Es lento en escritura. Si tengo un disco fallado y no puedo reponerlo, en vez de alojar datos en ese disco, se hace en la memoria los cálculos de las paridades y se guarda la paridad en el disco correspondiente. Nunca va a existir el dato, pero si la paridad, por lo que se puede calcular para poder leerlo.

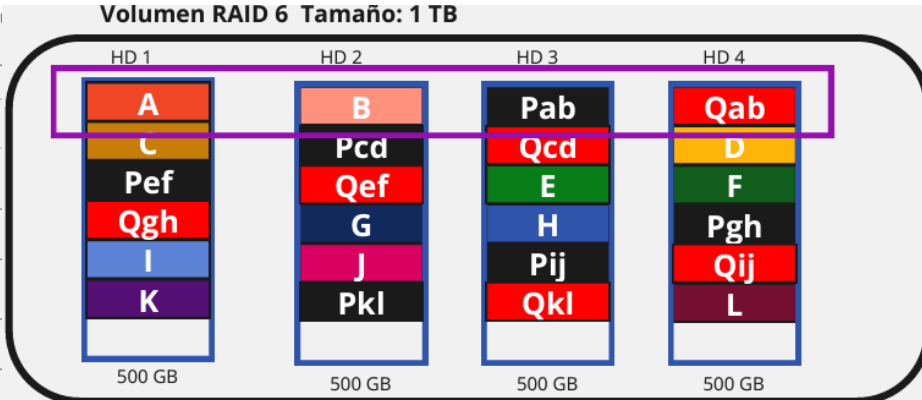


RAID 5 Block Level Striping with distributed parity		Ejemplo Vhd= 100 MB/s
Cantidad de Discos Mínima [N]:	3	
Cantidad de Discos que pueden fallar:	1	
Velocidad de Escritura Relativa a un disco: (Cantidad de Operaciones: [N+1])	Aprox: VDisco / (Cant. Oper.) [MB/s]	25 MB/s
Velocidad de Escritura Ponderada [1-10] : (5 es la Velocidad de 1 disco)	2	
Velocidad de Lectura Relativa a un disco: (Siempre el peor caso)	VDisco [MB/s]	100 MB/s
Velocidad de Lectura Ponderada [1-10] : (5 es la Velocidad de 1 disco)	5	
Costo [\$-\$\$\$]	\$\$	

- RAID 6:

Se utilizan dos paridades diferentes. Es muy lento para escritura, pero tiene mucha redundancia. Para Q se usa Reed Solomov y P XOR

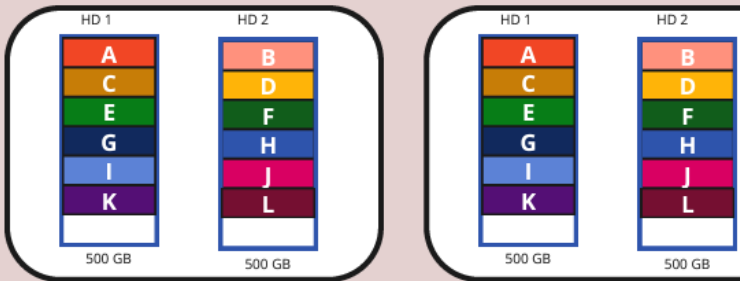
RAID 6 Block Level Striping with double distributed parity		Ejemplo Vhd= 100 MB/s
Cantidad de Discos Mínima [N]:	4	
Cantidad de Discos que pueden fallar:	2	
Velocidad de Escritura Relativa a un disco: (Cantidad de Operaciones: [N+2])	Aprox: VDisco / (Cant. Oper.) [MB/s]	16,66 MB/s
Velocidad de Escritura Ponderada [1-10] : (5 es la Velocidad de 1 disco)	2	
Velocidad de Lectura Relativa a un disco: (Siempre el peor caso)	VDisco [MB/s]	100 MB/s
Velocidad de Lectura Ponderada [1-10] : (5 es la Velocidad de 1 disco)	5	
Costo [\$-\$\$\$]	\$\$	



5.3 TIPOS DE RAID ANIDADOS

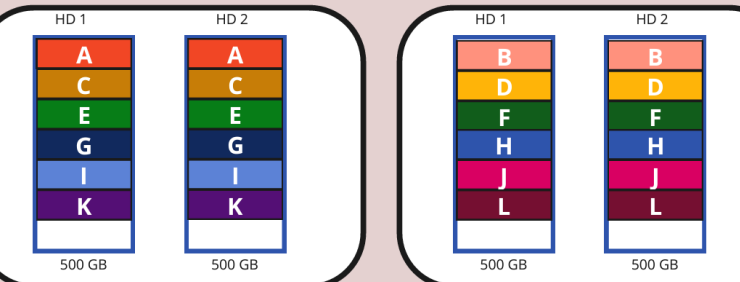
RAID XY: primero se aplica el X y después el Y.

- RAID 0+1:

Volumen RAID 0 + 1 Tamaño: 1 TB		RAID 0+1 Stripping + Mirroring	Ejemplo Vhd= 100 MB/s
		Cantidad de Discos Mínima [N]:	4
		Cantidad de Discos que pueden fallar:	1
		Velocidad de Escritura Relativa a un disco:	$(N/2) \times VDisco \text{ [MB/s]}$
		Velocidad de Escritura Ponderada [1-10] : (5 es la Velocidad de 1 disco)	10
		Velocidad de Lectura Relativa a un disco:	$(N/2) \times VDisco \text{ [MB/s]}$
		Velocidad de Lectura Ponderada [1-10] : (5 es la Velocidad de 1 disco)	10
		Costo [\$-\$\$\$]	\$\$\$

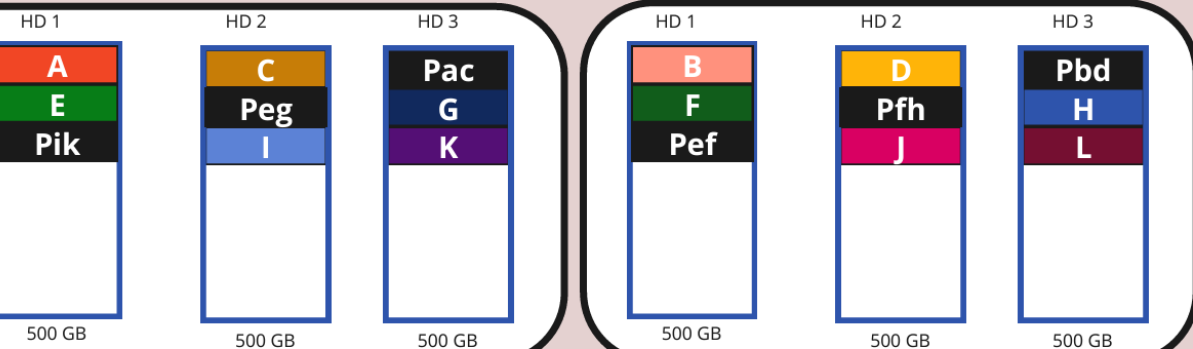
- RAID 10:

Se usa para virtualizar.

Volumen RAID 10 Tamaño: 1 TB		RAID 10(1+0) Mirroring + Stripping	Ejemplo Vhd= 100 MB/s
		Cantidad de Discos Mínima [N]:	4
		Cantidad de Discos que pueden fallar:	2, pero no 2 cualquiera
		Velocidad de Escritura Relativa a un disco:	$(N/2) \times VDisco \text{ [MB/s]}$
		Velocidad de Escritura Ponderada [1-10] : (5 es la Velocidad de 1 disco)	10
		Velocidad de Lectura Relativa a un disco:	$(N/2) \times VDisco \text{ [MB/s]}$
		Velocidad de Lectura Ponderada [1-10] : (5 es la Velocidad de 1 disco)	10
		Costo [\$-\$\$\$]	\$\$\$

- RAID 50:

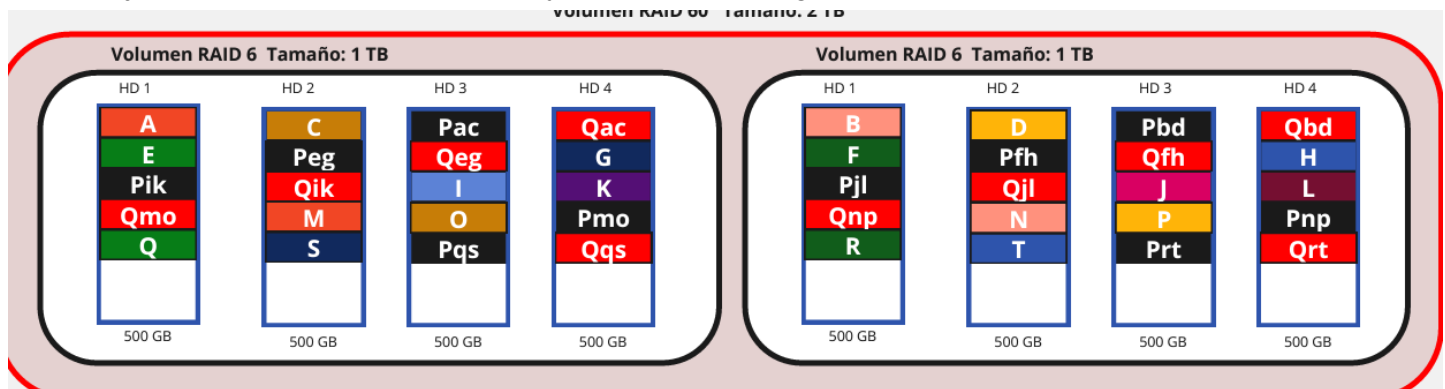
Costoso en potencia.

Volumen RAID 50 Tamaño: 2 TB		RAID 50 Block Level Striping with distributed parity + Stripping	Ejemplo Vhd= 100 MB/s
		Cantidad de Discos Mínima [N]:	6
		Cantidad de Discos que pueden fallar:	2, uno de cada RAID 5
		Velocidad de Escritura Relativa a un disco: (Cantidad de Operaciones: [N+1]) (Cantidad de Elementos RAID 0: M)	Aprox: $(VDisco / (Cant. Oper.)) \times M \text{ [MB/s]}$
		Velocidad de Escritura Ponderada [1-10] : (5 es la Velocidad de 1 disco)	3
		Velocidad de Lectura Relativa a un disco: (Siempre el peor caso)	$VDisco \times M \text{ [MB/s]}$
		Velocidad de Lectura Ponderada [1-10] : (5 es la Velocidad de 1 disco)	8
		Costo [\$-\$\$\$]	\$\$\$

- RAID 60:

Se necesita una controladora muy potente. Por cada conjunto se necesita calcular paridades.

Ventaja: pueden fallar muchos discos y es casi el más seguro (compite con el 10).



RAID 60 Block Level Stripping with double distributed parity + Stripping		Ejemplo Vhd= 100 MB/s
Cantidad de Discos Mínima [N]:	8	
Cantidad de Discos que pueden fallar:	4, dos de cada RAID6	
Velocidad de Escritura Relativa a un disco: (Cantidad de Operaciones: [N+2] (Cantidad de Elementos RAID 0: M)	Aprox: (VDisco / (Cant.Oper.)) * M [MB/s]	33,33 MB/s
Velocidad de Escritura Ponderada [1-10] : (5 es la Velocidad de 1 disco)	2	
Velocidad de Lectura Relativa a un disco:	M x VDisco [MB/s]	200 MB/s
Velocidad de Lectura Ponderada [1-10] : (5 es la Velocidad de 1 disco)	5	
Costo [\$-\$\$\$]	\$\$\$	

UNIDAD 6: VIRTUALIZACIÓN

6.1 VIRTUALIZACIÓN

Es una capa de abstracción (traducción) entre el software y el hardware físico. Permite que el hardware físico pueda ejecutar varias instancias de SO diferentes.

Genera una capa de simulación de hardware diferente al que tiene el equipo realmente. Siempre el hardware simulado tiene que ser menor o igual al físico, no podemos virtualizar 32GB de RAM si en el equipo hay 16.

Otorga la posibilidad de trasladar la máquina virtual a otra máquina física sin ningún tipo de problemas, mientras que el virtualizador y configuración sea la misma. Se puede virtualizar porque nos sobra potencia de proceso.

- ¿PARA QUÉ VIRTUALIZAMOS?

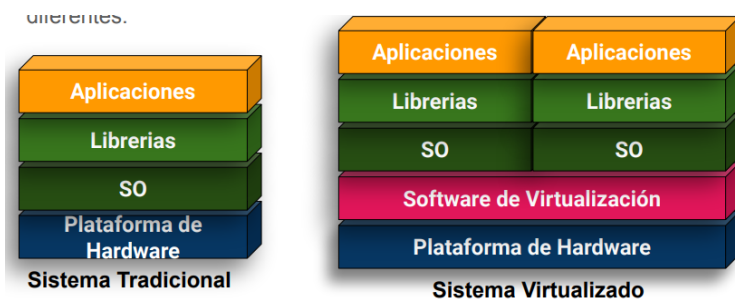
- Uso de hardware heredado: poder usar SO que necesiten hardware viejo.
- Implementación rápida: copiar una máquina virtual y se lleva a otras máquinas físicas.
- Versatilidad: se puede aprovechar todo el hardware.
- Consolidación y agregación: una máquina reemplaza a varias (una física con varias virtuales).
- Dinámica: facilidad de agregar/cambiar hardware para la máquina virtual.
- Facilidad de administración: se puede hacer toda la prueba de un sistema para una empresa en una sola máquina física.
- Mayor disponibilidad: varios hosts, si hay algún problema de hardware están los otros que lo reemplazan momentáneamente (cluster).

Sistema tradicional

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

Sistema virtualizado

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



6.2 ENFOQUES DE VIRTUALIZACIÓN

1. EMULACIÓN:

No es una técnica de virtualización pero es el escalón más alto en lo que es simular algo. Contiene virtualización. Se puede emular otra arquitectura diferente a la de la máquina.

2. VIRTUALIZACIÓN TOTAL (HYPERVISOR):

Virtualizamos la misma arquitectura del host. Traducción de instrucciones bit a bit. Cada instrucción pasa por el Hipervisor (el SO host), si es una instrucción no privilegiada se envía al

procesador (suma), si es una privilegiada (apagar la máquina) se realiza una subrutina y se simula, pero no se envía al procesador.

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

- HYPERVISOR:

Una Virtual Machine Monitor (VMM) o Hipervisor tiene 3 desafíos:

- Administración de instrucciones: distinguir entre instrucciones privilegiadas o no privilegiadas.
- administración de memoria: hace 2 veces la traducción de memoria. Que sector de memoria le va a dar a cada MV.
- Cómo acceder al hardware virtual: como el hardware virtual genera salidas sobre el hardware real. Por ejemplo: mando paquete en una red virtual, como hace el hipervisor para mapearlo a una real.

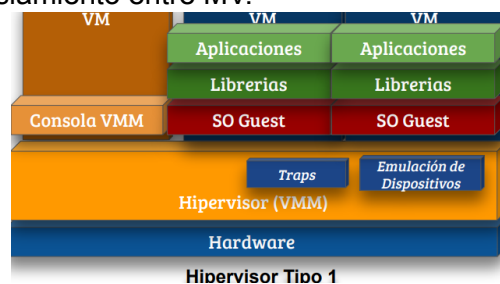
funciones:

- ejecución de instrucciones privilegiadas.
- Administración de ejecución de máquinas virtuales: aislamiento entre MV.
- gestión ciclo de vida de MV.
- interfaz para la administración del mismo.
- emular dispositivos y el control de acceso.

- TIPOS HYPERVISOR:

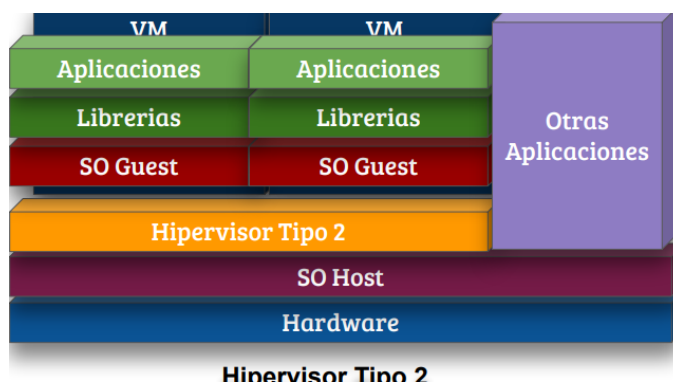
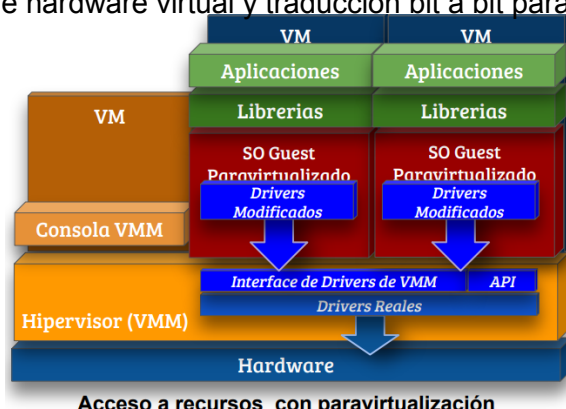
- TIPO 1:

El hipervisor es como el SO host de uso específico, se utiliza en servidores. Eficiente. Se centra en las funciones y desafíos. El resto de la potencia es dedicada a las MV, las MV compiten por los recursos y es poco vulnerable. Realiza la emulación del hardware virtual y traducción bit a bit para la virtualización total. No corre sobre un SO, es el SO, se instala en el hardware directo. No se adapta a ningún SO para sus funciones. **No corre aplicaciones que no sean virtualizar MV.**



- TIPO 2:

Hypervisor que corre arriba del SO host (es como otra app más instalada en el SO host). Hay pelea por los recursos entre las apps y las MV, disminuye el rendimiento. Es bastante vulnerable, hay peligro de que un virus afecte al SO host y por lo tanto se pierdan las MV. Realiza emulación de hardware virtual y traducción bit a bit para la virtualización total.



3. PARAVIRTUALIZACIÓN:

Más eficiente que la virtualización total. **El SO guest sabe que está siendo virtualizado** (modificando los drivers y necesita tener cierta vinculación con el SO host), por esto, las instrucciones privilegiadas (en vez de pasar por el Hipervisor) se realizan con el llamado (Hypercall) a la API del Hipervisor que las traduce en instrucciones que simulan lo mismo en la MV y las no privilegiadas directamente al procesador, por esto el hipervisor tiene mucho menos trabajo (no tiene que ver instrucción por instrucción).

Cada MV tiene drivers (red por ejemplo) que mandan a la capa de emulación del dispositivo del hipervisor. Esta capa traduce las cosas y envía todo a los drivers reales.

En vez de tener una capa de emulación de hardware y traps (para atrapar las privilegiadas) un SO guest que sabe que está siendo virtualizado, tiene mucha menos sobrecarga ya que no hay que emular hardware; y además de tener la API para las privilegiadas. Se MODIFICA EL SO GUEST.

4. CONTENEDORES:

No tienen como objetivo una máquina virtual, si no, trata de correr las apps que correrían en ellas. Como comparten kernel, no están tan aisladas como las MV, por lo tanto son vulnerables a los ataques hacia el SO host.

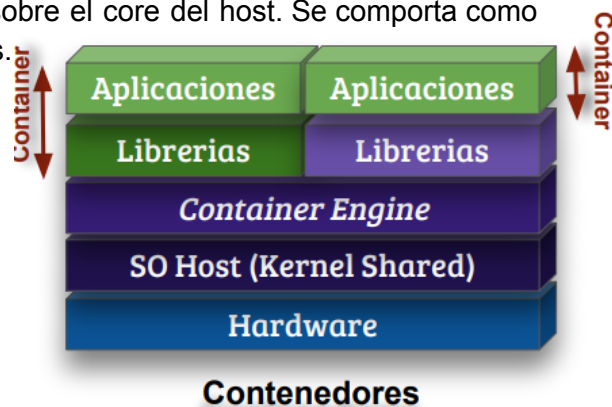
No es muy portable, pero son más eficientes (menos memoria y menos trabajo del hipervisor). No tiene como objetivo emular servidores físicos. No se busca hacer una máquina virtual, sino correr las aplicaciones que correrían en una máquina virtual. El container inicializa directamente en el kernel del SO host y en base a esto, puede correr las diferentes aplicaciones con sus librerías correspondientes.

cgroups: característica de kernel, permite que los procesos se organicen en jerarquía y monitorear los recursos (de cada container).

Se basan en cgroups (control groups) que proporciona para administrar los containers:

- Limitación de recursos: limitar los recursos de cada container.
- Priorización: darle prioridades a cada container (por ejemplo una base de datos).
- Contabilidad: cuantos recursos está utilizando el container.
- Control: poder de decisión sobre el container (apagar, etc.)

Un SO guest dentro del host es mucho más pesado en ejecución y carga de memoria. Podemos tener muchos más containers, cada container inicializa sobre el core del host. Se comporta como una MV, pero no tiene la idea de emular servidores físicos. Es más rápido y liviano que las MV.



- MOTOR DE CONTENEDORES:

Es como el Hypervisor de la MV.

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

Cumple las siguientes funciones:

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

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

Ventajas container:

- No hay necesidad de un SO guest.
- La capa de adm es sencilla y portable.

A tener en cuenta en container:

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

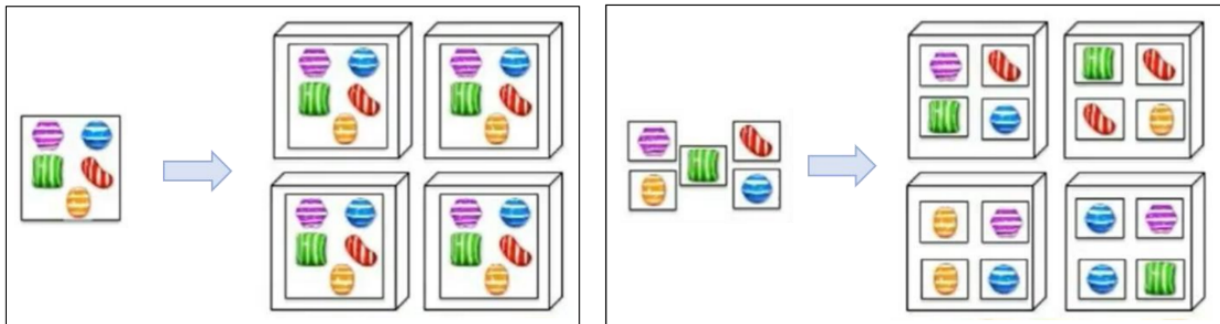
- ARQUITECTURAS:

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

- MONOLÍTICA:

Una app monolítica pone todas sus funcionalidades dentro de un único proceso. Cuando hay más sobrecarga en uno de las funciones, se podría escalar poniendo mas máquinas. El problema es que al escalar lo mismo y capaz solo necesitabas una función ahora sobran todos los demás.

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



- MICROSERVICIOS:

Pone cada elemento de funcionalidad en un servicio separado (conteneirisa cada función). La escala es elástica (se multiplican y distribuyen dichos servicios a través de los servidores, según se requiera). Cuando sobran servicios, se pueden ir eliminando para reducir los costos en el alquiler del server.

Los servicios están separados, son independientes pero conectados y se realizan llamadas entre ellos (cada uno tiene lo necesario para comunicarse con el otro). Cada servicio independiente es un contenedor. Para poder utilizar los servicios duplicados se necesita un balanceador de cargas (LB, load balancer) que distribuyen las cargas entre los diferentes contenedores que se instancian. A medida que la carga disminuya, se eliminan containers, por esto es elástico.

- DOCKER:

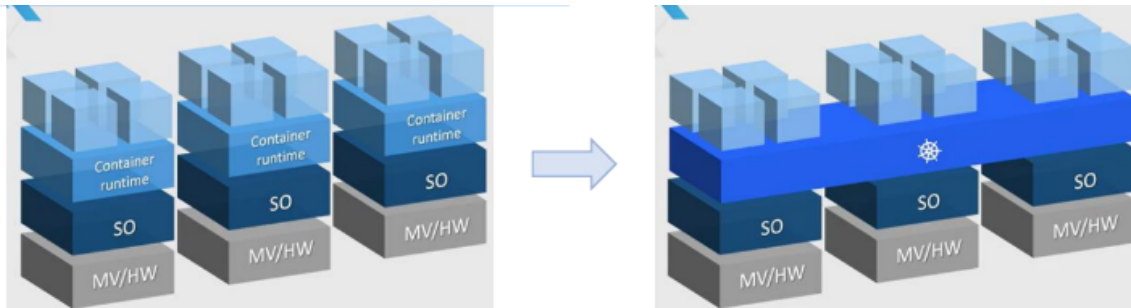
Facilita la implementación de los microservicios. Mete en un contenedor todo lo que la app necesita para que se ejecute. Dockerfile contiene la receta de lo que tiene que hacer el Docker Engine armando la imagen, que cuando se corre la imagen, aparece el microservicio (dockerfile le dice lo que tiene que hacer docker engine).

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

- KUBERNETES:

Administra cargas de trabajo basadas en contenedores y crea automatizaciones.

Es como un motor de contenedor que se encontraría en un nodo, pero Kubernetes es el orquestador de los contenedores en los distintos nodos:



Kubernetes no ve contenedores, sino POD que son los contenedores en ejecución, con características de monitoreo propias de Kubernetes para poder integrarlas en un nodo (computadora física o MV).

Realiza un monitoreo a los microservicios para:

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

Para mantener todo anidado se utilizan los Clúster (sistema distribuido de computadoras unidas entre sí). En kubernetes describo el estado en el que quiero que este cada POD y otras especificaciones.

- ASISTENCIA VÍA HARDWARE:

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

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

● “VT-x” y “AMD-V” (Aceleración de virtualización de MV):

Las instrucciones no privilegiadas se ejecutan directamente. Cuando hay una privilegiada se produce una interrupción.

● “VT-d” y “AMD-Vi” (Virtualización de dispositivos E/S para una única MV):

Modelos de virtualización:

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

● “VT-c” (Virtualización de dispositivos E/S para múltiples MV)



1. 4 niveles de prioridad. En el 3 corren las apps, en el 0 el SO. El problema es que se usaba el 0 y 3 que son de prioridad. Si ejecuto una instrucción privilegiada, no vamos a tener permiso
2. La VM corre en el 1, y el hipervisor en el 0. Las instrucciones privilegiadas tienen menos prioridad teniendo que pasar por el hipervisor.
3. El soporte de **VT-x** crea un anillo más (0 privilegiado). El 0 permite todas las instrucciones menos las privilegiadas, aumentando el rendimiento. Cada vez que manda una instrucción privilegiada pasa por el hipervisor. Es como si estuviera corriendo de forma nativa.

- VT-d:

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

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

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

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

- VT-c:

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

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

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