Unidad 1: SISTEMAS OPERATIVOS. CONCEPTOS GENERALES

1.1. Conceptos básicos. Definición de sistema operativo. Objetivos. Funciones.

1.2. Evolución de los sistemas de explotación. Sistema batch serial simple. Sistemas de multiprogramación. Sistemas de tiempo compartido. Sistemas de tiempo real. Sistemas embebidos. Sistemas de máquina virtual. Sistemas para móviles. Introducción a los sistemas de multiprocesamiento.

1.3. Estructura de un sistema operativo.

Respuesta

**1.1. Conceptos Básicos : este si**

Un sistema operativo (SO) es un conjunto de programas de control que gestiona la ejecución de aplicaciones y actúa como interfaz entre el hardware y los usuarios. Su núcleo (kernel) permanece en memoria mientras la computadora está en funcionamiento.

Los principales objetivos del SO son: este si

* **Abstracción del hardware**, facilita la interacción del usuario con el sistema.
* **Operación eficiente**, Un SO se debe construir de tal forma que

administre los recursos (procesador/es, memoria, periféricos y archivos) de la

mejor manera posible.

* **Capacidad de evolución**, permitiendo actualizaciones y mejoras sin afectar el funcionamiento.

Funciones del sistema operativo:Un sistema operativo proporciona un entorno para la ejecución de programas, podemos identificar varias clases comunes, a saber:

* **Ejecución de programas**: Gestiona la carga, ejecución e interrupción de programas.
* **Operaciones de Entrada/Salida (E/S)**: Facilita la comunicación entre programas y dispositivos.
* **Gestión de archivos**: Permite crear, eliminar y manipular archivos dentro del sistema.
* **Comunicación entre procesos**: Coordina el intercambio de información entre programas en ejecución.
* **Detección de errores**: Supervisa y responde a fallos en hardware y software.
* **Asignación de recursos**: Distribuye procesador, memoria y dispositivos según las necesidades de los programas.
* **Contabilidad del sistema**: Registra el uso de recursos para análisis y administración.
* **Protección y seguridad**: Restringe el acceso a información y recursos para garantizar su integridad.
* **1.2 Evolución de los Sistemas Operativos**

El desarrollo de los sistemas operativos ha estado marcado por avances tecnológicos y cambios en la forma en que los usuarios interactúan con las computadoras. A lo largo del tiempo, los sistemas han evolucionado para mejorar la eficiencia del hardware y optimizar la ejecución de programas.

* **Sistema Batch Serial Simple**

Este fue uno de los primeros sistemas operativos y surgió para automatizar la ejecución de trabajos en lote. Antes de su aparición, los programadores operaban manualmente las computadoras, ingresando código de máquina y controlando cada paso de ejecución.

Con el batch processing, se agrupaban trabajos similares y se ejecutaban de manera automática mediante un programa residente en la memoria, llamado *Monitor* o *Supervisor*. Este sistema permitía procesar programas de manera secuencial, aunque solo uno podía ejecutarse a la vez. Para comunicarse con el sistema, los usuarios utilizaban tarjetas de control perforadas.

* **Sistemas de Multiprogramación**

Estos sistemas marcaron una gran mejora al permitir la ejecución concurrente de varios programas en la memoria. Aunque en computadoras con un solo procesador solo podía ejecutarse una instrucción a la vez, la multiprogramación optimizaba los tiempos ociosos de la CPU alternando entre diferentes procesos.

La clave de esta modalidad era que cuando un programa debía esperar una operación de entrada/salida, otro tomaba el control de la CPU para continuar con la ejecución, evitando desperdicio de tiempo de procesamiento.

* **Sistemas de Tiempo Compartido**

La multiprogramación llevó al desarrollo de los sistemas de tiempo compartido, que permitían a múltiples usuarios interactuar con la computadora simultáneamente a través de terminales conectadas. En estos sistemas, la CPU se asignaba a cada usuario por breves intervalos de tiempo llamados *quantum* o *time slice*.

La principal ventaja era que los usuarios podían ejecutar comandos y recibir respuestas inmediatas, en lugar de esperar largos tiempos de procesamiento como ocurría en los sistemas batch. Además, se facilitaba el desarrollo y depuración de software, ya que los errores podían corregirse dinámicamente.

* **Sistemas de Tiempo Real:si va**

Estos sistemas están diseñados para aplicaciones donde el tiempo de respuesta es crítico. Se utilizan en entornos industriales, médicos y de control, donde una respuesta tardía puede comprometer la seguridad o la funcionalidad del sistema.

Los sistemas de tiempo real deben garantizar la ejecución de tareas dentro de tiempos específicos. Si la respuesta se produce fuera de la ventana de tiempo establecida, se considera que el sistema ha fallado.

Ejemplos comunes incluyen el control de tráfico aéreo, sistemas de frenos ABS en automóviles y dispositivos médicos como monitores cardíacos.

* **Sistemas Embebidos:si va**

Un sistema embebido es un sistema operativo diseñado para realizar funciones específicas dentro de un dispositivo.

A diferencia de los sistemas operativos generales, los embebidos tienen un hardware y software optimizado para su tarea, muchas veces incluyendo firmware en lugar de un SO completo. Estos sistemas son comunes en electrodomésticos, automóviles, teléfonos móviles y maquinaria industrial.

* **Sistemas de Máquina Virtual:si va**

La virtualización permite ejecutar múltiples sistemas operativos en un mismo hardware mediante un software llamado *Hypervisor* o *Virtual Machine Monitor (VMM)*.

Existen dos tipos principales de virtualización:

**Tipo 1**: Se ejecuta directamente sobre el hardware, sin depender de otro sistema operativo. Ejemplo: VMware vSphere, Microsoft Hyper-V.

**Tipo 2**: Se ejecuta sobre un sistema operativo anfitrión.

Los sistemas de máquina virtual permiten la consolidación de servidores, la prueba de nuevos sistemas operativos y la separación de entornos de trabajo en una misma computadora.Ejemplo: VirtualBox

* **Sistemas para Móviles**

Los sistemas operativos móviles han evolucionado desde sistemas embebidos hacia plataformas completas que ofrecen conectividad, almacenamiento y ejecución de aplicaciones avanzadas.

Actualmente, dominan el mercado **Android** y **iOS**, diseñados para optimizar la gestión de recursos en dispositivos con pantallas táctiles y múltiples sensores.

Estos sistemas están orientados a la interactividad, consumo de multimedia y conectividad inalámbrica, y utilizan un **núcleo híbrido**, combinando elementos de kernel monolítico y microkernel para mejorar rendimiento y estabilidad.

* **Introducción a los Sistemas de Multiprocesamiento:si va**

Los sistemas de multiprocesamiento utilizan múltiples procesadores para mejorar el rendimiento y la confiabilidad de los sistemas de computación. Existen dos tipos principales:

**Multiprocesamiento débilmente acoplado**: Cada procesador tiene su propio sistema operativo y hardware, conectados a través de una red.

**Multiprocesamiento fuertemente acoplado**: Varios procesadores comparten la misma memoria y trabajan en paralelo bajo un solo sistema operativo.

Estos sistemas se han vuelto fundamentales en la era moderna, ya que el crecimiento de la velocidad de los procesadores individuales ha alcanzado su límite físico, impulsando el uso de procesadores múltiples y núcleos multicore.

**1.3. Estructura de un Sistema Operativo:si va**

Los SO pueden tener diferentes estructuras:

**Kernel Monolítico**: Integra todos los servicios en el núcleo, ofreciendo alta velocidad de ejecución.Un SO monolítico se caracteriza por **implementar en el núcleo** a los cuatro componentes de administración fundamentales, que son la planificación de procesos, la administración de la memoria, la gestión de periféricos de E/S y la administración de archivos.Como inconveniente, este tipo de SO dispone de un alto número de líneas de código ejecutándose en modo privilegiado o supervisor

* **Microkernel**: Separa los servicios del núcleo, mejorando confiabilidad pero sacrificando rendimiento.
* **Núcleo Híbrido**: Combina elementos de ambos, adaptándose al hardware y necesidades del sistema.,

**Unidad 2: SISTEMAS OPERATIVOS. PROCESOS E HILOS**

2.1. Procesos. Definición. Estados de los procesos. Creación y terminación de

procesos. Modelo de proceso de seis estados. Descripción de procesos.

Estructuras de control del SO. Localización y atributos de proceso. El BCP.

2.2. Hilos (threads) de ejecución. Definición. Multihilo. Funcionalidades de los hilos.

Estados y sincronización de los hilos.

Aquí tienes un resumen basado en el contenido de tu documento:

### Unidad 2: ****Sistemas Operativos – Procesos e Hilos****

#### ****2.1 Procesos****

Un **proceso** es una unidad de ejecución dentro de un sistema operativo. Se distingue de un "trabajo" o "job" en que es una entidad interna, mientras que un trabajo es externo al sistema. Además, un proceso es la ejecución de un programa, lo que lo hace dinámico en comparación con la naturaleza estática del código de un programa.

Los procesos pasan por varios **estados** durante su ejecución:

* **Nuevo**: se ha creado pero aún no está en ejecución.
* **Listo**: espera turno para ejecutarse.
* **Activo**: está siendo ejecutado por el procesador.
* **Bloqueado**: detenido, a la espera de un evento externo.
* **Suspendido**: movido a memoria secundaria.
* **Terminado**: ha completado su ejecución.

La **creación** y **terminación** de procesos ocurren por diversas razones, desde sesiones interactivas de usuario hasta errores de ejecución o decisiones del sistema operativo.

El **modelo de seis estados** define un flujo de transición entre estos estados, con eventos como interrupciones o solicitudes de E/S afectando el cambio entre ellos.

El sistema operativo usa **estructuras de control** para gestionar los procesos, organizando información clave en una **Tabla de Bloques de Control de Procesos (TBCP)**. En esta tabla, cada proceso tiene su **Bloque de Control de Proceso (BCP)**, donde se almacena información relevante como identificadores, registros de CPU, estado del proceso, prioridades y recursos asignados.

#### ****2.2 Hilos (Threads) de ejecución****

Un **hilo** es una unidad de ejecución dentro de un proceso. Todos los hilos comparten los mismos recursos del proceso (memoria, archivos abiertos, etc.), lo que permite ejecutar múltiples tareas de manera concurrente.

El **multihilo** es la capacidad del sistema operativo de soportar múltiples hilos dentro de un proceso, mejorando eficiencia y comunicación. Cada hilo tiene:

* Un estado de ejecución independiente.
* Su propio contexto de ejecución.
* Una pila exclusiva.
* Acceso compartido a la memoria y recursos del proceso.

Los **estados de los hilos** son similares a los de los procesos: **nuevo, listo, activo, bloqueado y terminado**. Sin embargo, los hilos no pueden ser suspendidos individualmente.

La **sincronización de hilos** es esencial para evitar interferencias entre ellos, garantizando la correcta ejecución y acceso a los recursos compartidos dentro de un proceso.

Este resumen cubre los puntos clave de la unidad 2 sobre procesos e hilos en sistemas operativos. Si necesitas detalles adicionales o explicaciones más específicas, dime en qué parte quieres profundizar.

### 1 - ****Procesos****

Un proceso es una unidad de ejecución dentro del sistema operativo, compuesta por un conjunto de instrucciones en ejecución, su estado, y los recursos asignados por el sistema.

**DTE de proceso (Diagrama de Transición de Estado)**  
Los procesos pasan por los siguientes estados en un sistema operativo:

* **Nuevo**: Se ha creado pero aún no está en ejecución.
* **Listo**: Está preparado para ejecutarse, pero espera turno en la cola de planificación.
* **Activo**: Está siendo ejecutado por el procesador.
* **Bloqueado**: Detenido, esperando que se cumpla un evento externo (como una operación de E/S).
* **Suspendido**: Se encuentra en memoria secundaria, esperando ser reactivado.
* **Terminado**: Ha finalizado su ejecución y el sistema operativo libera sus recursos.

**Diferencia con un hilo de ejecución**  
Un proceso es una entidad completa con su propio espacio de memoria y recursos asignados. Un hilo, en cambio, es una subdivisión dentro de un proceso, que comparte recursos con otros hilos dentro del mismo proceso y permite ejecutar múltiples tareas concurrentemente.

### 2 - ****Hebra (Hilo de ejecución)****

Un hilo es una unidad de ejecución dentro de un proceso. Permite dividir tareas dentro de un mismo proceso, compartiendo la misma memoria y recursos del sistema.

**DTE de Hebra (Diagrama de Transición de Estado)**  
Los hilos siguen estados similares a los procesos, excepto que no pueden ser suspendidos individualmente:

* **Nuevo**: Se ha creado pero aún no está en ejecución.
* **Listo**: Está preparado para ejecutarse, pero espera turno en la cola de planificación de hilos.
* **Activo**: Se está ejecutando en el procesador.
* **Bloqueado**: Espera un evento externo para continuar su ejecución.
* **Terminado**: Ha finalizado su ejecución y el sistema operativo libera sus recursos.

**Diferencia con un proceso**  
Mientras un proceso tiene su propio espacio de memoria y recursos asignados, un hilo comparte estos recursos con otros hilos dentro del mismo proceso, lo que permite mayor eficiencia y rapidez en la ejecución de tareas concurrentes. Los hilos pueden comunicarse entre sí sin necesidad de invocar al núcleo del sistema, lo que los hace más ligeros que los procesos.

**Unidad 3: CONCURRENCIA**

3.1.Principios de la concurrencia. Condición de carrera. Interacción de procesos.

Competencia entre procesos por recursos. Cooperación entre procesos. Requisitos

para la exclusión mutua.

3.2.Exclusión mutua: Soporte hardware: Deshabilitar interrupciones e Instrucciones

hardware especiales. Soporte software: Semáforos binarios y Semáforos

contadores, Monitores y Paso de mensajes

### ****3.1 Principios de la concurrencia : esto si va****

**Concurrencia**: es cuando varios procesos acceden (real o aparentemente de manera simultánea) a un recurso no compartible. Esto genera situaciones como:

* 1. Cambio de contexto entre procesos.
  2. Protección de operaciones frente a interferencias.
  3. Sincronización entre tareas dependientes.

**Condición de carrera**: aparece cuando el resultado depende del orden en que los procesos acceden y modifican datos compartidos. Pequeñas variaciones de tiempo generan distintos resultados, lo que hace que el comportamiento sea no determinista.

**Interacción entre procesos**:

* 1. **Competencia**: cuando dos procesos compiten por un mismo recurso sin conocerse. Se requiere exclusión mutua.
  2. **Cooperación vía memoria compartida**: cuando los procesos comparten datos comunes sin comunicarse explícitamente. Se deben sincronizar para preservar la coherencia.
  3. **Cooperación vía comunicación**: cuando los procesos intercambian mensajes directamente. No comparten memoria, pero deben coordinarse.

**Requisitos para exclusión mutua**:

* 1. Un solo proceso en la sección crítica.
  2. Sin espera indefinida.
  3. Acceso inmediato si está libre.
  4. Independiente de la velocidad y cantidad de procesadores.
  5. Permanencia finita en la sección crítica.

### ****3.2 Exclusión mutua : esto si va****

**Soporte por hardware**:

* + **Deshabilitar interrupciones**: válido solo en monoprocesadores. No es escalable ni seguro.
  + **Instrucciones especiales (ej. Test-and-Set)**: operaciones atómicas para reservar recursos de forma segura, evitando condiciones de carrera.

**Soporte por software**:

* + **Semáforos binarios**: variables que indican si un recurso está libre (0) u ocupado (1). Se gestionan con WAIT y SIGNAL.
  + **Semáforos contadores**: extensión de los binarios, permiten conocer cuántos procesos están esperando. Se inicializan con INIT.
  + **Monitores**: módulo que encapsula la lógica de acceso exclusivo. Garantiza que solo un hilo ejecute código dentro del monitor. Más seguros y legibles.
  + **Paso de mensajes**: comunicación explícita usando primitivas send y receive. Útiles en sistemas distribuidos o sin memoria compartida.

### ****1. Conceptos básicos****

**Concurrencia**: Es la existencia de múltiples procesos que se ejecutan “simultáneamente” (de forma real o intercalada) y compiten por recursos que no pueden ser compartidos. Implica que el sistema debe gestionar accesos, sincronización y cambios de contexto entre tareas.

**Condición de carrera**: Ocurre cuando varios procesos acceden y modifican datos compartidos, y el resultado final depende del orden en que se intercalan sus instrucciones. Este tipo de error es difícil de reproducir y puede generar resultados incorrectos.

**Sección crítica**: Parte del código de un proceso que accede a recursos compartidos. Para evitar interferencias con otros procesos, es crucial que solo un proceso esté ejecutando su sección crítica en un momento dado.

**Exclusión mutua**: Conjunto de mecanismos que garantizan que solo un proceso puede acceder a una sección crítica determinada al mismo tiempo. Es la clave para evitar condiciones de carrera.

### ****2. Exclusión mutua: Monitores y Paso de mensajes****

**Monitores**:

* + Son estructuras de alto nivel que combinan sincronización y exclusividad.
  + Un monitor encapsula variables y procedimientos, y solo permite que un proceso acceda a sus procedimientos a la vez.
  + Dentro del monitor, la exclusión mutua es automática: los procesos que quieren entrar mientras otro está dentro, quedan bloqueados.
  + Son útiles para entornos multihilo, y permiten mantener la integridad de los datos compartidos sin tener que gestionar manualmente semáforos.

**Paso de mensajes**:

* + Es una técnica de sincronización y comunicación que **no requiere memoria compartida**.
  + Usa primitivas como send(destino, mensaje) y receive(origen, mensaje).
  + Ambos procesos deben coordinarse para intercambiar información.
  + Es fundamental en sistemas distribuidos, donde los procesos no tienen acceso a la misma RAM.
  + Se puede configurar como bloqueante (espera confirmación) o no bloqueante (continúa sin esperar).

### ****3. Exclusión mutua: Semáforos binarios****

* **Semáforo binario**:
  + Es una variable compartida entre procesos que toma dos valores: 0 (libre) y 1 (ocupado).
  + Si un proceso necesita usar un recurso, **ejecuta** WAIT(X): si X = 0, lo asigna (X := 1); si X = 1, espera.
  + Cuando termina, **ejecuta** SIGNAL(X) para liberar el recurso (X := 0) y despertar a un proceso que estuviera esperando.
  + El SO administra colas de espera asociadas a cada semáforo.
  + Este mecanismo es eficiente y más simple que contar procesos, pero puede ser vulnerable a errores si las operaciones no son bien implementadas.

Unidad 4

### ****1. Fundamentos del interbloqueo****

Un **interbloqueo** (o deadlock) es el bloqueo permanente de un conjunto de procesos que están esperando recursos o eventos que sólo pueden ser liberados por otros procesos igualmente bloqueados. Es una situación en la que nadie puede avanzar. El clásico ejemplo es el de los autos que se bloquean entre sí en un cruce porque cada uno espera que otro avance.

### ****2. Condiciones para el interbloqueo (Condiciones de Coffman)****

Para que ocurra un interbloqueo deben cumplirse estas cuatro condiciones:

1. **Exclusión mutua**: Un recurso sólo puede ser usado por un proceso a la vez.
2. **Retención y espera**: Un proceso puede retener recursos mientras espera otros.
3. **Sin desalojo**: No se puede obligar a un proceso a liberar recursos.
4. **Espera circular**: Hay una cadena cerrada de procesos, cada uno esperando por un recurso que posee otro proceso de la cadena.

### ****3. Estrategias del SO para tratar el interbloqueo****

#### ****3.1 Omisión****

El sistema ignora los interbloqueos, asumiendo que son poco probables. Es una estrategia simple, pero riesgosa, ya que no se hace nada para evitarlos o solucionarlos.

#### ****3.2 Prevención****

Evita que las condiciones necesarias para el interbloqueo ocurran:

* Se puede eliminar la espera circular imponiendo un orden a la solicitud de recursos.
* También se puede forzar a los procesos a pedir todos los recursos de una vez, evitando la retención y espera.
* Sin embargo, no siempre es eficiente y puede desaprovechar recursos.

#### ****3.3 Predicción o evitación****

El sistema **evita llegar a estados inseguros**, conociendo de antemano las necesidades máximas de cada proceso. Si se anticipa que una asignación podría llevar al interbloqueo, se niega. Aquí entra el concepto de **estado seguro**, en el que se garantiza que todos los procesos podrán terminar si solicitan todos sus recursos máximos en ese momento.

#### ****3.4 Detección y recuperación****

El SO permite la asignación libre de recursos y **detecta interbloqueos periódicamente**. Si se detectan, se aplican estrategias para resolverlos:

* Cancelar procesos involucrados.
* Hacer rollback a puntos de control anteriores.
* Terminar procesos uno por uno, según criterios como prioridad, uso de CPU o recursos asignados.

Unidad 5

¡Vamos al grano, alan! Aquí tenés el resumen solicitado sobre planificación de procesos, dividido por los puntos que mencionaste:

### ****1. Algoritmos de planificación monoprocesador****

#### ****FCFS o FIFO (First-Come, First-Served)****

Es el más simple: los procesos se atienden en orden de llegada. No hay desalojo: un proceso sigue hasta que termine o se bloquee. Puede generar tiempos de espera altos para procesos cortos si llegan detrás de otros largos.

#### ****Más corto primero sin desalojo (SJF - Shortest Job First)****

Selecciona el proceso con la **ráfaga de CPU más corta estimada**. Es muy eficiente en tiempo de respuesta, pero necesita estimar la duración de los procesos, lo cual no siempre es fácil. No hay desalojo: si un proceso comienza, se mantiene hasta que se bloquee o finalice.

#### ****Más corto primero con desalojo****

También llamado **Shortest Remaining Time First (SRTF)**. Se compara el tiempo restante de ejecución del proceso actual con el de un nuevo proceso que llega. Si el nuevo tiene menor tiempo estimado, se interrumpe al que está ejecutándose. Mejora el turnaround, pero añade complejidad en la implementación.

#### ****Administración por prioridades****

Cada proceso tiene una prioridad asignada, y el SO siempre elige el de mayor prioridad. Puede ser con o sin desalojo. Problema clásico: **inanición** de procesos de baja prioridad. Solución: incrementar su prioridad con el tiempo (aging).

#### ****Turno rotatorio (Round Robin)****

Cada proceso recibe un **quantum** o tiempo fijo de CPU. Al finalizar su quantum, si no terminó, se manda al final de la cola. Es justo y eficiente para entornos compartidos. La elección del tamaño del quantum es clave: si es muy pequeño, hay sobrecarga; si es muy grande, se comporta como FCFS.

#### ****Colas múltiples****

Se utilizan varias colas para distintos tipos de procesos: interactivos, batch, en tiempo real. Cada cola puede tener su propio algoritmo y prioridad. Permiten organizar mejor los recursos según la naturaleza del proceso. Problema: el “**efecto residual**”, donde un proceso puede quedar indefinidamente en una cola de baja prioridad.

### ****5.2. Planificación multiprocesador****

#### ****Granularidad de sincronización****

Refleja la frecuencia de comunicación/sincronización entre procesos o hilos:

* **Grano grueso**: mínima sincronización, ideal para procesos independientes.
* **Grano medio**: coordinación frecuente entre hilos (como en una misma app).
* **Grano fino**: sincronización constante, uso intensivo de paralelismo.

#### ****Aspectos de diseño de planificación multiprocesador****

Involucra:

* **Asignación de procesos a procesadores**: puede ser estática (cada proceso se asigna a un procesador fijo) o dinámica (se usa una cola global y se asigna al procesador disponible).
* **Uso de multiprogramación en cada procesador**: sobre todo en arquitecturas con muchos núcleos.
* **Activación de procesos**: elección del próximo proceso/hilo a ejecutar. En multiprocesadores se tiende a usar esquemas más simples para reducir overhead.

#### ****Planificación de hilos de ejecución****

Hay varios enfoques:

1. **Compartición de carga**: una cola global de hilos, y cada procesador toma uno disponible.
2. **Planificación en grupo (gang scheduling)**: se planifican varios hilos del mismo proceso a la vez, útil para procesos altamente sincronizados.
3. **Asignación de procesador dedicado**: cada hilo recibe un procesador específico todo el tiempo. Poco eficiente, pero útil en sistemas con muchos núcleos.
4. **Planificación dinámica**: el sistema reajusta cuántos procesadores tiene un proceso durante su ejecución, según necesidades y disponibilidad.

Unidad 6

¡Por supuesto, alan! Acá te paso un resumen detallado y ordenado de los temas que pediste de la Unidad 6 sobre administración de la memoria:

### ****6.1. Introducción a la Memoria Principal****

* **Fundamentos**: La RAM es el único espacio de almacenamiento que el procesador puede usar directamente. Todo programa debe cargarse ahí antes de ejecutarse. Es responsabilidad del SO manejarla eficientemente y protegerla entre procesos.
* **Hardware**: Los sistemas modernos incluyen una **Unidad de Manejo de Memoria (MMU)** para mapear direcciones lógicas a físicas y proteger accesos.
* **Resolución de direcciones**: Pasa por fases: desde direcciones simbólicas (compilación) hasta direcciones absolutas (ejecución). El SO puede ubicar un proceso en cualquier parte de la RAM, independientemente de su dirección lógica.

### ****6.2. Asignación Particionada Fija****

* Se divide la memoria en bloques de tamaño fijo cuando arranca el sistema.
* Los procesos se cargan en la partición que les alcance. Esto genera **fragmentación interna**.
* Fácil de implementar pero **limita el tamaño de los procesos** y el número de procesos en memoria.

### ****6.3. Asignación Particionada Dinámica****

* Se crean particiones a medida que los procesos se cargan.
* **Técnicas de asignación de espacio libre**:
  + Primer ajuste: primer hueco suficientemente grande.
  + Mejor ajuste: hueco más ajustado posible.
  + Peor ajuste: hueco más grande disponible.
* **Fragmentación externa**: huecos dispersos que no pueden ser reutilizados eficientemente.
* **Compactación**: reorganización de memoria para unificar huecos libres contiguos. Costosa en tiempo.
* **Swapping**: se mueve un proceso a almacenamiento secundario temporalmente para liberar espacio.

### ****6.4. Segmentación Simple****

* La memoria del proceso se divide en **segmentos** lógicos: código, datos, pila, etc.
* Cada segmento puede tener distintos permisos.
* El SO usa **tablas de segmentos** para mapear las direcciones lógicas a físicas.
* **Evita la fragmentación interna**, pero persiste la externa.

### ****6.5. Paginación Simple****

* La memoria física se divide en **marcos** y los procesos en **páginas** del mismo tamaño.
* Usa una **tabla de páginas** para hacer el mapeo.
* **Evita la fragmentación externa**, pero puede haber **fragmentación interna**.
* Tamaño de página: potencia de dos, balancea entre desperdicio y sobrecarga administrativa.
* Utiliza **TLB (Translation Lookaside Buffer)** para acelerar traducción de direcciones.

### ****6.6. Memoria Virtual****

* Técnica para simular más memoria RAM usando almacenamiento secundario.
* Permite que los procesos crean que están completamente en RAM aunque no sea así.
* Se apoya en la paginación para mover páginas entre memoria principal y disco.

### ****6.7. Paginación por/Bajo Demanda****

* Las páginas se cargan **sólo cuando se necesitan**, mediante un mecanismo **perezoso (lazy)**.
* Usa un **bit de validez** por página para saber si está en memoria.
* Si no está, se produce un **fallo de página**, el SO la carga desde disco y actualiza las estructuras.

#### ****Algoritmos de sustitución de páginas****:

1. **FIFO**: la más antigua se reemplaza. Simple, pero puede causar la anomalía de Belady.
2. **Óptimo (OPT)**: reemplaza la que no se usará en el futuro. Teórico.
3. **LRU**: menos recientemente usada. Buena aproximación al óptimo.
4. **MFU/LFU**: más/menos frecuentemente usada. Poco utilizados en práctica.
5. **LRU con bit de referencia**: simplificación de LRU usando bits de acceso.
6. **Algoritmos con buffers**: el sistema prepara espacio libre anticipadamente.

#### ****Hiperpaginación (Thrashing)****

* Ocurre cuando hay **demasiados fallos de página seguidos**.
* El sistema pasa más tiempo cargando páginas que ejecutando procesos.
* Se evita **monitoreando la tasa de fallos** y ajustando la cantidad de marcos asignados por proceso.