Resumen Teorico - Modulo 1

Introduccion

Que es un SO

→ Programa de **software** que necesita memoria y procesador para ejecutarse (hardware).

Perspectiva de usuario (aplicaciones)

- Abstrae (oculta y administra) complejidades de las **arquitecturas de las computadoras**.
    - Ejemplo: Archivos.

Perspectiva del Hardware

- Administra Recursos (memoria, uso del bus, …) de uno o mas `procesos`.

Resumen: *Es un intermediario entre el usuario de una computadora y el hardware de la misma.*

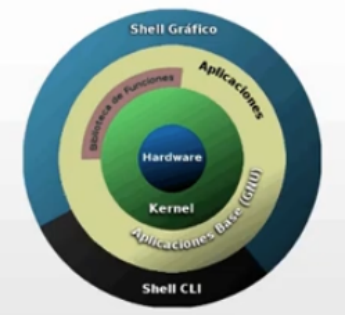
Objetivos de un SISTEMA OPERATIVO

Administrar eficientemente el uso de **la CPU (**evitar apropiaciones de CPU) **y la Memoria (**proteger espacios de direcciones)**.** Brindar asistencia para las acciones de E/S.

Componentes de un SISTEMA OPERATIVO

Kernel: Primer nivel de interaccion entre HW y SW.

- Se encuentra en **memoria principal.**
- Administracion de recursos (memoria, cpu).
- Gestion de servicios (E/S, Concurrencia, Procesos)



Shell: Forma de interactuar con el SO. **Grafica (Graphic User interface)** o con **comandos (Command line interface)**.

Herramientas “Extra”: Servicios no esenciales que ofrecen los SO.

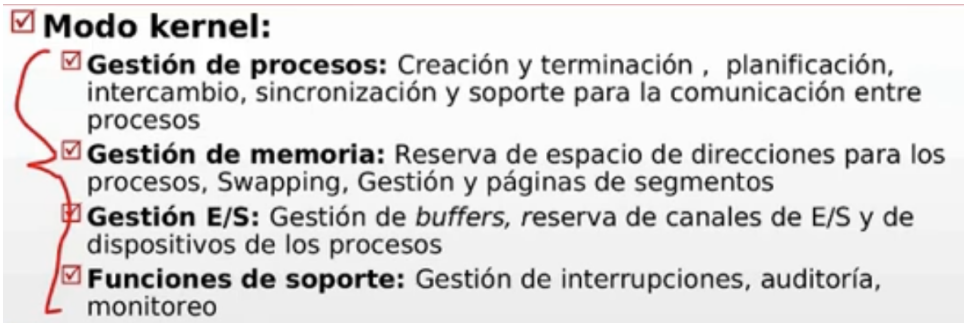
Apoyo del Hardware al SO

1. Se definen 2 **modos de ejecucion** que limitan las posibles acciones de los procesos.
2. Se deben evitar **apropiaciones de cpu** y surge la idea de **interrupcion por clock.**
3. Se definen **espacios de direcciones** donde puede acceder cada proceso.

Modo de ejecucion

La CPU tiene un **bit de modo** que dice en que modo se esta ejecutando.

- Kernel o supervisor: Procesos deben acceder a estructuras de kernel o direcciones que no le pertenecen (rutinas de atencion). *Como lo dice su nombre* ***los modulos del kernel se ejecutan en modo kernel.***



- Usuario: Proceso puede acceder solo a su **espacio de direcciones.**

Cambio de modo: Sistema comienza en modo kernel y al ejecutar el primer proceso de usuario se pone en modo usuario. Para volver al modo kernel debe haber un **trap** o una **interrupcion .** Las **rutinas de gestion** solo estan disponibles en **modo kernel.**

BSOD: Ocurre cuando hay un bloqueo en modo kernel.

Interrupcion por clock

Clock o contador para evitar que procesos se aduenien de la CPU.

Proteccion de la memoria

Registro base y **registro limite** para proteger los **espacios de direcciones.**

System calls

Las **system calls** son rutinas a traves de las cuales los **procesos** pueden acceder a los servicios del SO. La llamada se hace en modo usuario con ciertos parametros y luego la ejecucion del servicio se hace en modo kernel. EJ: I/O.

- La llamada puede ser a traves de una **interrupcion por software,** poniendo en un registro que servicio queremos.

Tipos de kernel

Monolitico

Tareas propias de un SO se ejecutan en **modo kernel.**

> ✅ Menos tiempo “perdido” en cambios de modo

Microkernel

Se busca disminuir el tamanio y las responsabilidades del **kernel ,s**e delegan al modo usuario.

> ✅ Se evitan problemas como el BSOD.

Procesos

→ Programa en ejecucion.

- Un unico **proceso activo** (en CPU) por cada CPU presente.
- Un proceso a su vez es una **abstraccion** de instrucciones que se estan ejecutando y cuenta con:
  - Instrucciones.
  - Datos.
  - Stacks de datos temporarios (de estado).

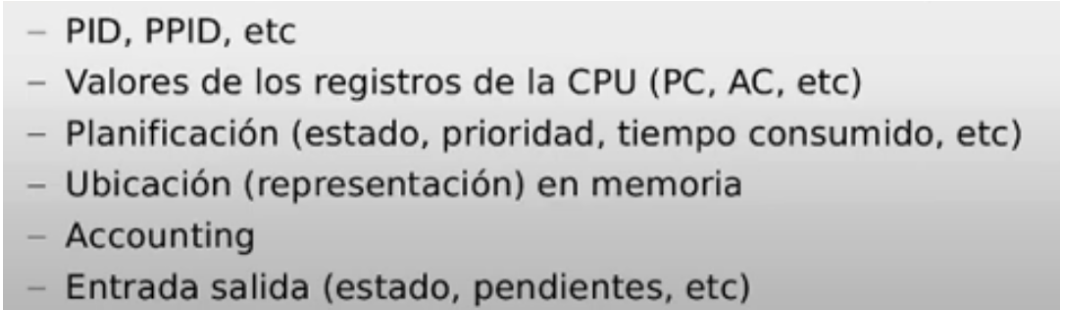
Stacks

- Creacion **automatica** y **dinamica** (tamanio).
- Stack Frame: Parametros de rutina, Stack pointer y Contador de programa del anterior Stack.

PCB (Process Control Block)

- Lo primero que se crea (cuando se crea un proceso) y lo ultimo que se borra cuando termina.

Informacion Contenida



Registros de la cpu para que en caso de ser expulsados, cuando vuelva a su ejecucion lo haga con el mismo estado previo.

Ejemplo PCB (Teorico)

Administración de procesos	Administración de memoria	Administración de archivos
Registros	Apuntador a la información del segmento de texto	Directorio raíz
Contador del programa	Apuntador a la información del segmento de datos	Directorio de trabajo
Palabra de estado del programa	Apuntador a la información del segmento de pila	Descripciones de archivos
Apuntador de la pila		ID de usuario
Estado del proceso		ID de grupo
Prioridad		
Parámetros de planificación		
ID del proceso		
Proceso padre		
Grupo de procesos		
Señales		
Tiempo de inicio del proceso		
Tiempo utilizado de la CPU		
Tiempo de la CPU utilizado por el hijo		
Hora de la siguiente alarma		

Espacio de direcciones (PCB)

- Conjunto de direcciones que **ocupa** un proceso. *En memoria paginada serian los frames que tiene asignados un proceso.*

🚨 NO INCLUYE AL PCB

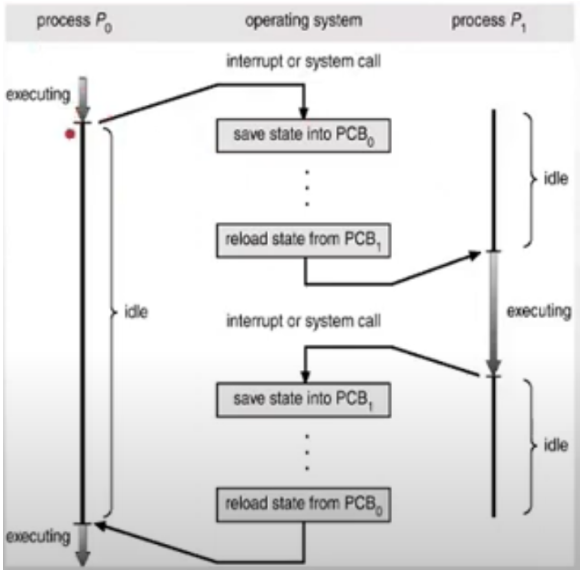
- En **modo usuario** solo se puede acceder al espacio de direcciones propio. En **modo kernel** se puede acceder a espacios de otros procesos.

Contexto de Proceso

Informacion necesaria para la correcta ejecucion (en CPU) de un proceso.

Context Switch: Se guarda contexto de proceso actual (excepto que haya finalizado) para cargar contexto de otro proceso.

- **TIEMPO NO PRODUCTIVO DE CPU.**



Kernel

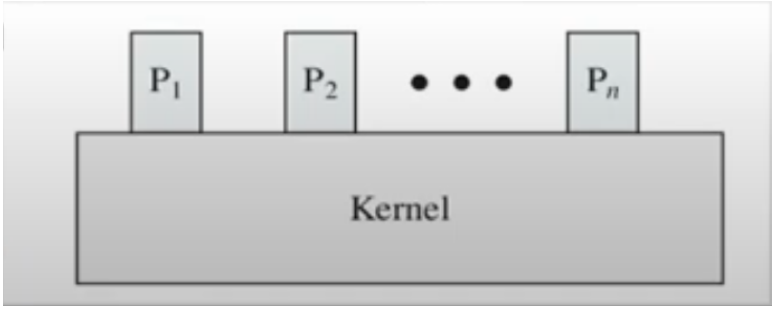
Software que necesita memoria y CPU.

🚨 El Kernel **No es un proceso.**

*Los procesos son los **programas de usuario en ejecucion.***

Kernel como entidad independiente (Enfoque 1)

Por cada **system call (interrupcion)** que se hace se debe hacer un **context switch** para pasar a ejecutar software de Kernel.

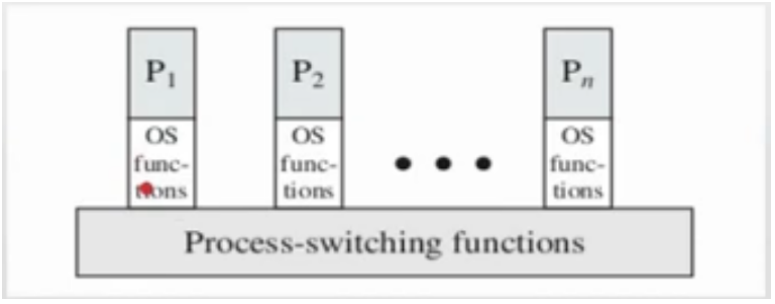


- Kernel tiene su **propio espacio de memoria** y tambien un **stack propio.**

Kernel “dentro” del proceso (Enfoque 2)

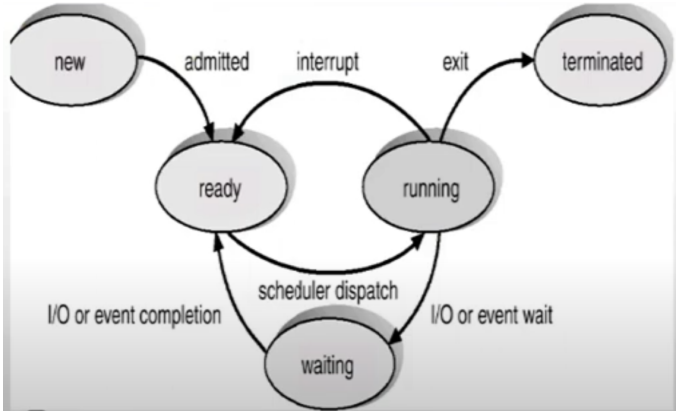
*Proceso en **modo usuario** y Kernel **en modo kernel.***

- El codigo Kernel se encuentra en todos los espacios de direcciones y se puede ver como una coleccion de rutinas que el proceso utiliza.
- No se necesita **context switch** por cada **system call.** Podemos ver las **system calls** como una simple llamada a una rutina y **UN CAMBIO A MODO KERNEL.**



Pilas para modo usuario Pi y Pilas para modo Kernel.

Estados de un proceso



1. New: Se inicializa la estructura de PCB, pero se necesita **memoria** (espacio de direcciones).
2. Ready: Ya se le asigno memoria y tiene todo lo necesario para ejecutarse, pero le falta CPU.
3. Running: Se produce **context switch** y se pasa a estado de **running.**
4. Posibilidades:
    a. Terminated: Se completo la ejecucion. Se borrar todas las estructuras (para liberar memoria).
    b. Waiting: Se pasa a este estado cuando se requiere operacion de E/S. De waiting vuelve a pasar a **ready.**

Planificacion

Las colas de planificacion utilizan al **PCB** como una abstraccion de cada proceso.

Colas de trabajo / procesos: Contienen a todos los PCB.

Colas de listo / ready queue: Contienen PCB de procesos en Memoria principal en estado **ready.**

Colas de dispositivos: PCB de procesos esperando dispositivos I/O.

Modulos de planificacion

Son modulos de **software** asociados a la planificacion **propios del kernel.** Estos modulos son **event-based.**

Los principales son los **schedulers** (planificadores) y se separan por frecuencia de ejecucion.

Long term scheduler: Que procesos se admiten a memoria. Aumento en el grado de multiprogramacion.

- **LOADER:** Carga en memoria el proceso elegido.

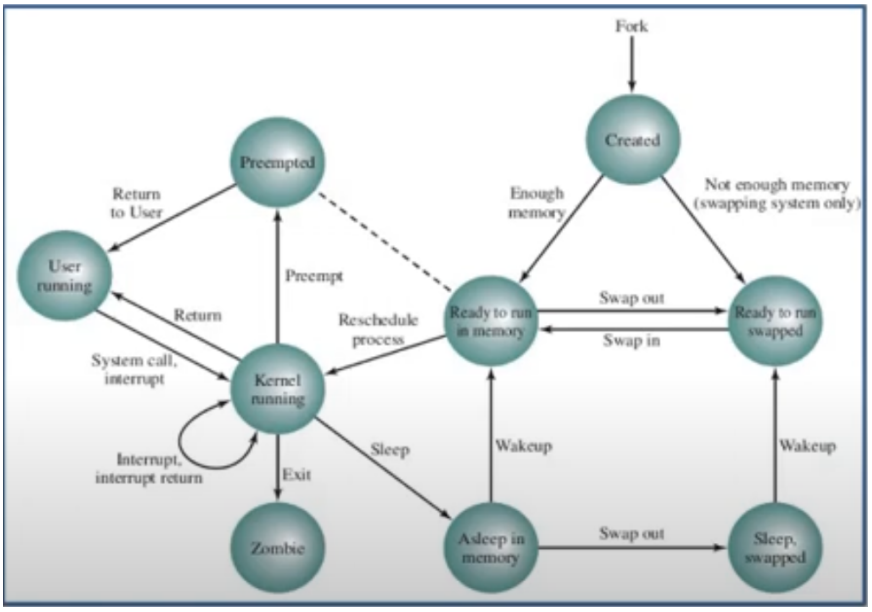
Medium term scheduler: Swapping entre disco (memoria SWAP) y memoria ram. El proceso ya tiene todo para correr pero puede que no haya suficiente espacio o se deba liberar espacio.

- Disminuye grado de **multiprogramacion** (porque libera espacio para mantener equilibrio del sistema).
- SWAP IN y SWAP OUT.

Short term scheduler: Decide que procesos pasan de **ready** a **running.**

- **DISPATCHER**: Se encarga de hacer el **context switch (modo kernel).**

> 💡 Multiprogramacion: Cantidad de procesos tratandose (que se encuentran cargados en memoria) en simultaneo.



Las flechas punteadas indican que hay un proceso que vuelve a memoria mientras que otro (el que se apropia) va a ejecutar a la cpu. El **estado asleep in memory** esta esperando por un evento.

Procesos en tratamiento

Alternan **rafagas de cpu** y **rafagas de E/S.**

- Estos procesos se pueden clasificar en **I/O bound** o **CPU** bound. De acuerdo a lo que mas necesitan.

> 💡 Estado Optimo: Atender en rafagas cortas I/O bound para mantener dispositivos E/S ocupados y mientras tanto atender CPU bound.

Algoritmos de Planificacion

Apropiativos: Proceso puede ser expulsado de la CPU sin haber terminado.

No apropiativos: Solo pueden salir de CPU si **Finalizan** o **Requieren E/S.**

Procesos batch

El usuario no espera una respuesta de estos.

- Se pueden utilizar **SJF (**este resulta mas rapido) **o FCFS.** Ambos no apropiativos.

Procesos interactivos

Necesitamos respuestas rapidas → Rafagas de CPU cortas.

Necesitamos que varios procesos pasen por CPU → Apropiativo.

- Round Robin (Q chicos pero sin tantos context switch), prioridades, colas multinivel(doble planificacion) o SRTF.

> 💡 El mecanismo es el algoritmo(round robin, prioridades, FCFS, etc) y la politica es el parametro (quantum, prioridad, etc) del mecanismo

Creacion de Procesos

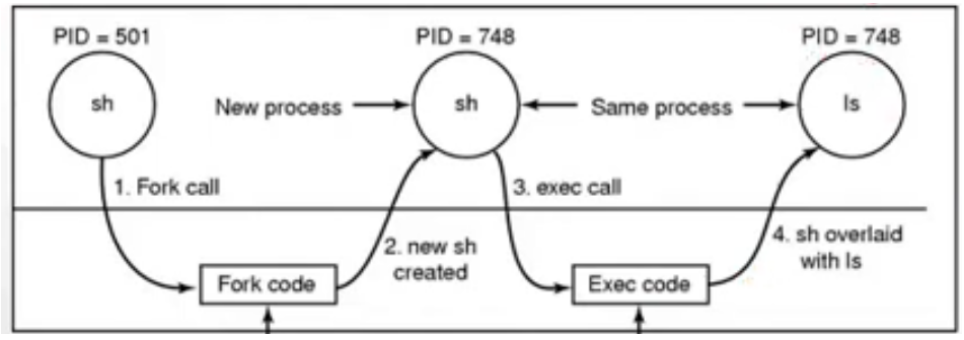
Un proceso es creado por otro proceso (arbol de procesos).

- Se crea el PCB, se le asigna un **PID unico**, se le asigna su espacio de direcciones y se crean las estructuras de datos asociadas. La imagen del proceso que se crea esta en memoria ram.

Espacio de direcciones - Creacion de procesos

Unix: Hijo es un **duplicado** del padre. Ambas de abajo son **system calls.**

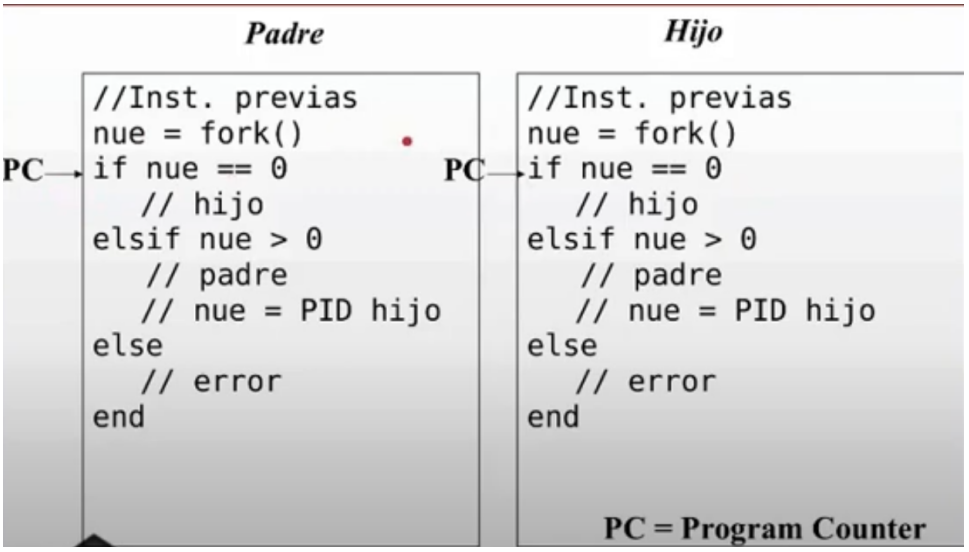
- fork(): Crea proceso hijo que es una copia del padre.
- execve(): Se carga un programa en el espacio de direcciones.



En PID=748 se ejecuta execve(), por lo tanto cambia el contenido (NO SE CREA UN PROCESO NUEVO) y en lugar de tener sh como contenido tendra a ls.

Windows: Se le carga un nuevo programa al proceso hijo.

- CreateProcess(): Crea y carga programa en proceso hijo.



Se copian los registros, pero a partir de ahi no se comparten. Es decir que si modifico una variable en el hijo esta no se modifica en el padre.

Memoria

Las responsabilidades de un SO frente a la administracion de memoria son las de **alocar y desalocar memoria** (para los procesos) y llevar un **registro de la memoria utilizada y** restringir los accesos **inapropiados.**

- Estas responsabilidades son una **abstraccion** para el programador (este no tiene que conocer los manejos de ram).
- El SO debe restringir los accesos a direcciones que no son propias (excepto que tengan permiso - MODO KERNEL).
- Permitir secciones compartidas → Rutinas usadas por multiples procesos (evitar copias de instrucciones).

Espacio de direcciones

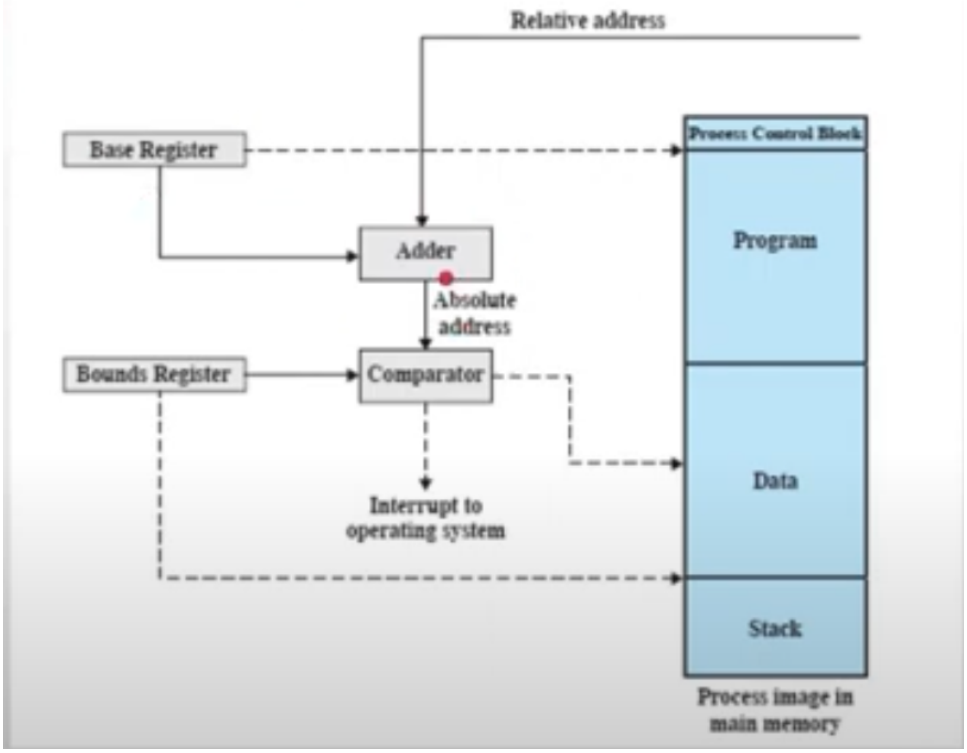
- Direccionamiento logico: `2^^BITS` direcciones logicas referenciables (no quiere decir que realmente existen esas direcciones fisicas.

Direcciones

Logicas: Utilizadas por los procesos.

Fisicas: Direcciones con las que accedemos a la RAM.

- Cada **espacio de direcciones** tiene un **registro base y limite** que se determinan al cargar el espacio de direcciones y dicen en que lugares fisicos estara el espacio de direcciones de nuestro proceso.

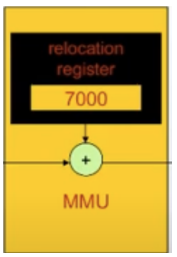


Si al transformar a una direccion fisica se supera el registro limite ocurre un trap y se le otorga el control al SO.

- Este **mapeo** entre **direcciones virtuales** y **direcciones fisicas** se realiza en **tiempo de ejecucion** por hardware (debe ir a la velocidad de la CPU) llamado MMU.

MMU

- Reprogramacion de este unicamente en **modo kernel.**
  Por ejemplo el valor del **relocation register.**



Particiones

✅ Este soporte es brindado por el hardware y el SO se debe adaptar a esto.

Particiones fijas

- Particiones con tamanios definidos y luego cada proceso se lleva a una particion de acuerdo a algoritmos: Best Fit (tamanio que mejor se ajusta), First Fit (primer particion que se ajusta), Worst Fit (la que mas espacio libre deja), Next Fit.

⛔ Espacio libre sin utilizar

Particiones Dinamicas

- Se generan dinamicamente de acuerdo al tamanio justo que necesita un proceso.

⛔ Memoria fragmentada

Fragmentacion

💡 Localidad de memoria que no puede ser utilizada por no encontrarse de forma contigua.

Fragmentacion interna: Memoria libre de una particion (solo ocurre en partciones fijas) que no puede ser utilizada por premisa de ***un proceso una particion.***

Fragmentacion externa: Una vez terminados procesos de particiones pequenias, quedan huecos que no pueden ser utilizados (por tener espacio insuficiente).

- Compactacion: Mover procesos a los extremos para que estos espacios pequenios libres se compacten y formen un bloque mas grande.

Paginacion

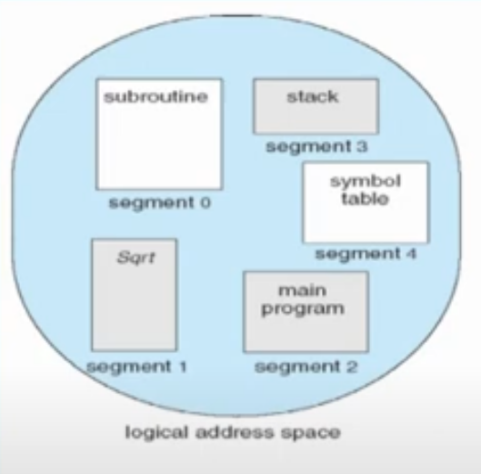
Permite locaciones discontinuas.

Dividir a la memoria en **marcos** y a los procesos en **paginas** (porciones de igual tamanio).

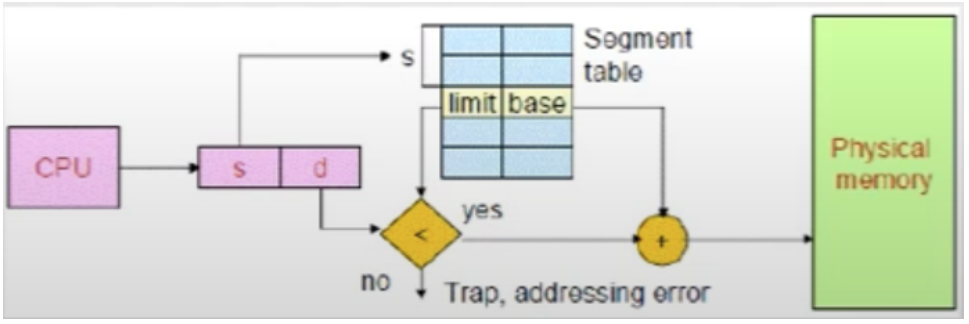
Estructura necesaria → **TABLA DE PAGINAS**

Segmentacion

Segmentos de distintos tamanios, donde cada uno cuenta con informacion de su **longitud** o **limite.** A su vez cada segmento es agrupado de acuerdo a la informacion que contiene (stack, rutina, main program, etc).

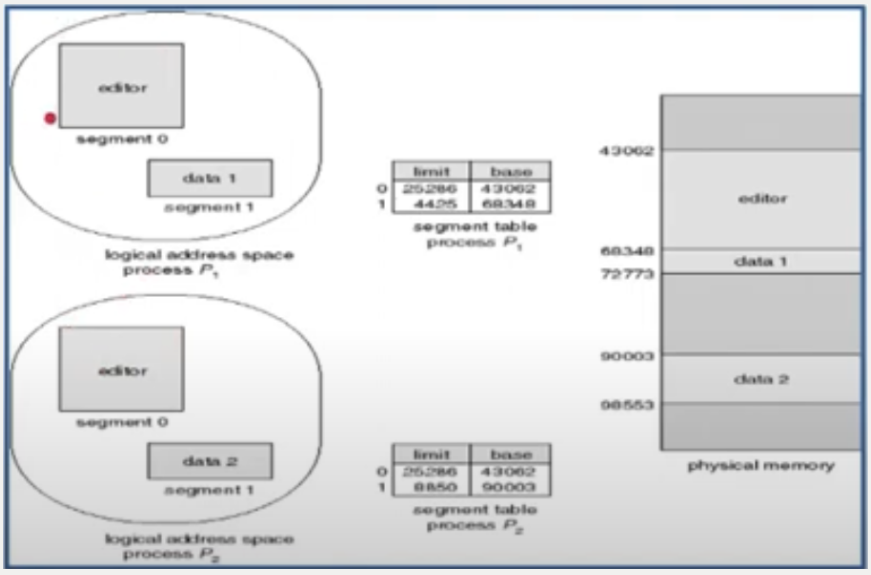


- Esta informacion **limite** nos permite validar que no se haga referencia a un area fuera del **segmento**.



⛔ Puede ocurrir una **fragmentacion externa** debido al tamanio variable de los **segmentos**.

✅ Se puede compartir codigo (segmento de instrucciones) usando datos distintos por cada proceso. Hacemos referencia a un mismo segmento desde **tablas de segmentos de procesos distintos.**



Segmentacion Paginada

Se combina lo mejor de ambos mundos. Primero se divide la informacion de los procesos en segmentos de longitud variable (de acuerdo a sus requerimientos) y luego estos segmentos se dividen en paginas para evitar la fragmentacion externa que aparece en la **segmentacion pura.**