Sistemas Operativos

Administración de memoria

Jerarquía de memoria:

1. Registros CPU del procesador
2. Caché
3. Memoria principal RAM
4. Almacenamiento secundario - Disco (memoria virtual)

Dirección lógica = CPU Dirección física = RAM

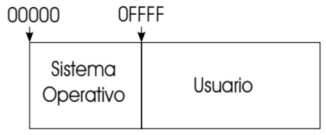
Parte del SO que gestiona la memoria realiza:

- Control de que partes de la memoria estén utilizados o libres.
- Asignar memoria a procesos y liberarla cuando terminan.
- Administrar intercambio entre memoria y disco.

Sistemas monoprogramados

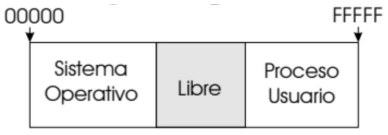
La memoria principal está dividida en dos particiones:

- ➔ Para el usuario (procesos)
- ➔ Para el sistema operativo



Es necesario proteger las particiones entre sí ya que a veces el tamaño del SO puede variar. Para esto se puede:

- Realizar reubicación dinámica del espacio
- Cargar los procesos usuarios en memoria alta



Sistemas multiprogramados

Se debe compartir la memoria entre varios procesos que esperan la asignación de la misma. Es deseable que haya varios procesos en memoria para su ejecución concurrente.

Esquemas de asignación de memoria

1. Contigua: particiones fijas y variables
2. Swapping (intercambio)
3. Paginación
4. Segmentación
5. Segmentación paginada

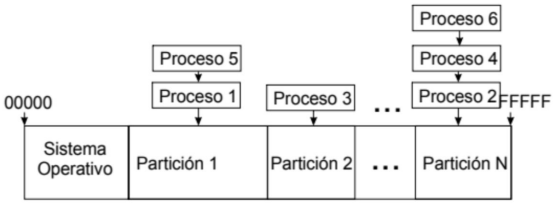
1. Particiones

La memoria está dividida de antemano en espacios (particiones), entonces si un proceso necesita ejecutarse, se le asigna uno de dichos espacios. Cada partición puede contener un único proceso. Pueden ser:

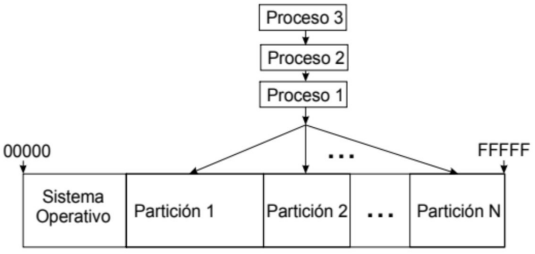
- Particiones fijas
  - **Todas del mismo tamaño**

    - El nivel de multiprogramación está limitado por el número de particiones
    - Hay una cola con procesos que quiere utilizar memoria y ejecutarse
    - Hay una tabla para indicar las particiones ocupadas y las libres

  - **Con diferentes tamaños**: para los procesos que quieren ejecutarse podemos:
    - Tener varias colas: cada proceso se asigna una cola en función de su tamaño



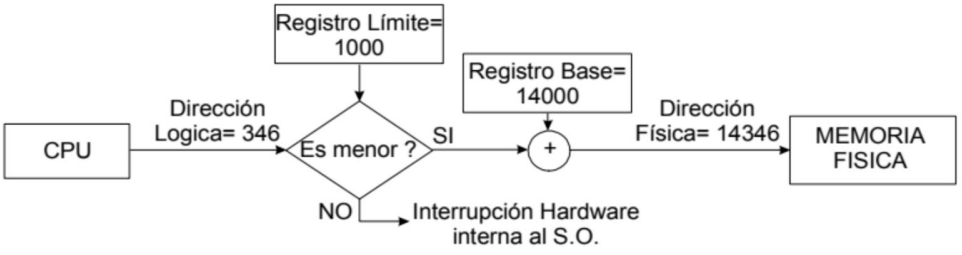
    - Tener una única cola: cuando se libera una partición se le asigna al primer proceso que cabe en ella.



Desventajas

- Existe Fragmentación interna y externa:
  - Interna: Un proceso ocupa menos espacio del que se le asignó.
  - Externa

- Necesidad de protección (en sistemas multiprogramados): que un proceso no acceda al área de memoria del otro. Si la reubicación es dinámica se puede usar **registros base-límite**



- Particiones variables

Funcionamiento:

1. Inicialmente: toda la memoria (salvo la partición de SO) está disponible para los procesos.
2. Llega un proceso: se introduce en un hueco libre y el espacio no ocupado será un nuevo hueco.
3. Cada memoria ocupada es una partición
4. Cuando un proceso termina, libera el espacio, se convierte en hueco y se junta con los huecos adyacentes.

Se conserva una tabla de partes de memoria ocupadas y libres y la cola de entrada de procesos en memoria.

Desventajas

- Fragmentación externa: la memoria puede llegar a quedar dividida en huecos pequeños por lo que si llega un nuevo proceso no entra, a pesar de que la suma del espacio libre es suficiente para ese proceso. Entonces el nuevo proceso no se carga en memoria.

No hay fragmentación interna ya que las particiones se crean con el tamaño solicitado por el proceso.

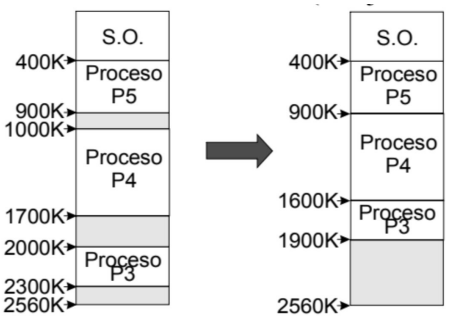
Hay varias estrategias para asignar memoria a un nuevo proceso:

- Primer ajuste: Elegir el primer hueco libre de tamaño suficiente (⅓ mem utilizada)
- Mejor ajuste: Elegir el hueco más pequeño con tamaño suficiente (requiere ver toda la lista si no está ordenada)
- Peor ajuste: Elegir el hueco más grande. Pretende conseguir que los huecos que queden sean más grandes (requiere ver toda la lista si no está ordenada)

Requiere de protección de memoria: se usa código reubicable (dinámico) → se puede usar registros base y límite.

2. Compactación

Intenta solucionar fragmentación externa: consiste en desplazar las particiones ocupadas para que estén juntas en memoria quedando un solo hueco libre de mayor tamaño. Solo es posible si la reubicación es dinámica (en ejecución).



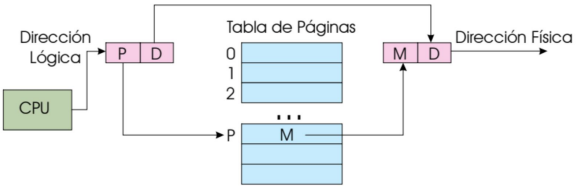
Desventajas

- Consume tiempo: se necesita desplazar zonas de memoria
- Difícil seleccionar una estrategia de compactación óptima

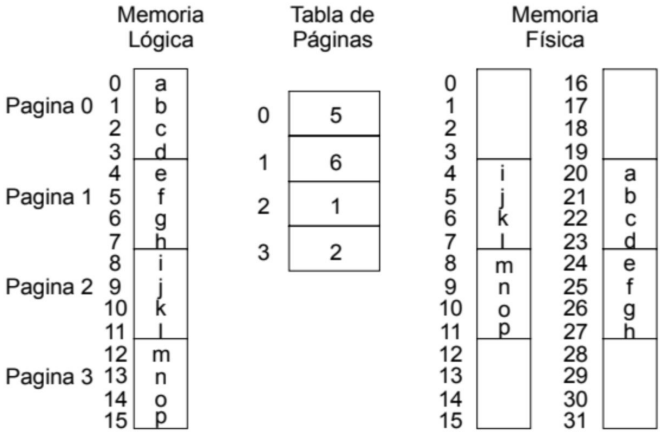
3. Paginación

- Solución a la fragmentación externa ya que permite que la memoria de un proceso no sea contigua
- Hay una distinción entre direcciones lógicas y físicas.
- La **memoria lógica** se divide en bloques de tamaño fijo llamados **páginas**
- La **memoria física** se divide en bloques llamados **marcos,** que tienen el mismo tamaño que las páginas. Ese tamaño es definido por HW
- Las páginas de un proceso se cargan en los marcos de la memoria principal que estén disponibles.
- Utiliza un HW de paginación para la traducción de direcciones

1bloque→ 1proceso→ muchas páginas

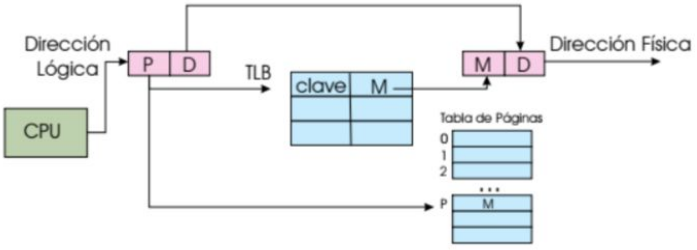


- La dirección lógica generada consta de **número de página** (P) y **desplazamiento dentro de la página / offset** (D).
- La tabla de páginas contiene la dirección base de la memoria física: permite establecer una correspondencia entre el número de página y un número de marco



Normalmente se elige un tamaño de página de potencia 2 ya que es más fácil la traducción de direcciones lógicas a físicas: por ejemplo, si el tamaño de la dirección lógica es: 2^m y el tamaño de la página es 2^n unidades de direccionamiento, entonces m-n bytes de orden alto de la dirección lógica representan el número de página y los n bits de orden bajo el offset de la página.

- Para mejorar la performance se utiliza una caché de acceso rápido para la tabla de páginas: un HW especial llamado TLB (translation look-aside buffers)
  - Los registros contienen sólo unas pocas entradas de una tabla de páginas
  - Contiene una clave (número de página) y un valor (número de marco)
- Compara el valor de la página deseada con todas las claves, si la clave está le da el número de marco asociado, sino accede a la tabla de páginas de memoria.



- El SO traduce direcciones usando una copia de la tabla de páginas en memoria.

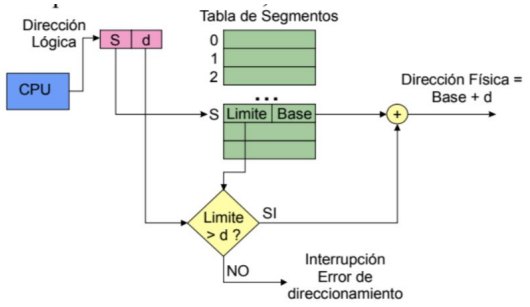
Ventajas

➔ Páginas compartidas: la paginación permite compartir código común entre varios procesos solo si el código es reentrante (no se modifica durante la ejecución).

➔ Protección de memoria: en la tabla de páginas pueden encontrarse unos bits de protección asociados a cada marco que indican si el marco es solo lectura o lectura y escritura. Además se controla que el número de página no supere el total de páginas usadas por el proceso (dirección incorrecta)

4. Segmentación

- El espacio de direcciones lógicas se compone de un conjunto de segmentos de tamaño variable: cada uno tiene un nombre y una longitud *<número de segmento, desplazamiento>*
- Hay una división lógica del proceso en diferentes segmentos
- Utiliza un HW de segmentación mediante una Tabla de segmentos:
  - Establece la correspondencia entre direcciones físicas y lógicas
  - Se busca en la tabla de acuerdo con el número de segmento
  - Cada entrada contiene dos registros: **base** (indica inicio de la tabla de segmentos en memoria) y **límite** del segmento (longitud del segmento)
  - Se compara el límite del segmento con el desplazamiento: si el desplazamiento es válido, se suma la dirección del registro base.



1bloque→ 1proceso→ muchos segmentos

Ventajas

➔ Protección: tiene bits de protección (sólo lectura o lectura y escritura) que se consultan antes de acceder al segmento.

➔ Compartición de código:
- Puede realizarse a nivel segmento (código o datos).
- Cada proceso tendrá una tabla de segmentos.
- Compartir un segmento significa que una entrada de la tabla de segmentos coincide en varios procesos (igual posición física).

Desventajas

- Fragmentación: los segmentos son de tamaño variable por lo que puede haber fragmentación externa: bloques de memoria demasiados pequeños para contener un segmento.

➔ Solución: se puede compactar la memoria (segmentación usa reubicación dinámica).

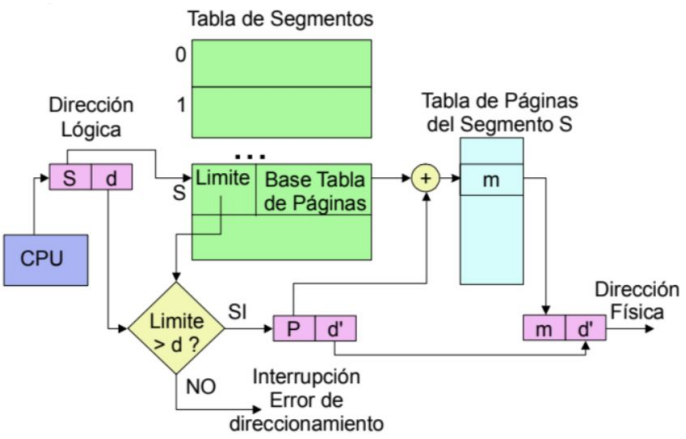
5. Segmentación paginada

Lógica = Páginas
Física = Marcos/Frames

- La memoria lógica está dividida en **bloques** llamados **segmentos** que contienen regiones de un proceso *<número de segmento, desplazamiento>*
- Los segmentos están divididos en **páginas** de igual tamaño que los **marcos** (potencias de 2).
- Las páginas de un proceso se cargan en marcos de la memoria principal.
- Cada segmento tiene asociado una tabla de páginas: se usa un registro límite y base de la tabla de páginas para cada segmento.

Esquema de traducción de direcciones:

- Hay una tabla de páginas por segmento
- S = entrada de la tabla de segmentos:
  - Contiene el límite del segmento
  - Contiene la dirección base de una tabla de páginas
- D = desplazamiento:
  - Un número de página P
  - Un nuevo desplazamiento dentro de la página d'



1. **¿Por qué no es posible forzar la protección de memoria en tiempo de compilación ?**

   Importante: Se debe proteger la memoria porque si varios programas comparten la memoria principal, debe asegurarse que ninguno de ellos pueda modificar el espacio de memoria de los demás.

   Porque en tiempo de compilación, en la mayoría de los SO, se desconoce qué región de memoria será asignada al proceso, por lo tanto no se sabe que direcciones ocupará y no tendrá manera de saber qué región deberá proteger.
   Además, como en TdeC las direcciones físicas son iguales a las lógicas, si asignamos direcciones absolutas al compilar el ejecutable, no podemos asegurar que las direcciones que se asignan invadan el espacio de direcciones de otros procesos usuarios o del SO.

2. **¿Cuál es la diferencia entre la fragmentación interna y externa?**

   La fragmentación externa se da cuando existe suficiente cantidad de memoria para satisfacer los requerimientos de la memoria de un proceso, pero no puede ser asignada al mismo porque no se encuentra de manera contigua.
   La fragmentación interna se da cuando un proceso tiene asignada cierta porción de memoria, pero hay una parte de ella que no está siendo usada por el mismo: se asignó más memoria de la que necesitaba.

   ¿Mecanismos para reducir la fragmentación? ¿Qué tipos de sistemas administrativos posee la fragmentación? ¿Cómo se podría eliminar la fragmentación?

   Compactación: intenta solucionar fragmentación externa: consiste en desplazar las particiones ocupadas para que estén juntas en memoria quedando un solo hueco libre de mayor tamaño. Solo es posible si la reubicación es dinámica (en ejecución).

   Asignación con particiones fijas: consiste en dividir la memoria en espacios (particiones) fijas (ya sea todas del mismo tamaño o de diferentes tamaños). Cuando un proceso quiere ejecutarse, se lo ubica en una cola para luego asignarle uno de dichos espacios.
   Tiene fragmentación interna debido a que al ser particiones fijas, puede que a un proceso se le asigne más espacio de memoria del que necesita, y tambien tiene fragmentación externa.

   Asignación con particiones variables: a diferencia de las particiones fijas, en este caso se le asigna a cada proceso que se quiere ejecutar el espacio que necesita.
   Se soluciona la fragmentación interna ya que se le asigna a cada proceso el espacio de memoria que necesita, pero sigue habiendo fragmentación externa.

   Paginación: en este esquema de asignación, la memoria lógica se divide en bloques de tamaño fijo llamados páginas y la memoria física se divide en bloques del mismo tamaño que las páginas, llamados marcos. El tamaño lo decide HW.
   Permite que el espacio de direcciones lógicas de un proceso sea de manera no contigua, por lo que se elimina la fragmentación externa. Pero la fragmentación interna sigue existiendo ya que el tamaño de las páginas es fijo.

   Segmentación: el espacio de direcciones lógica se compone por un conjunto de segmentos de tamaño variable. Cada uno tiene un número de segmento y un desplazamiento.
   Dado que el tamaño de los segmentos es de tamaño variable, se elimina la fragmentación interna. pero no la externa. Para solucionar la fragmentación externa se puede compactar la memoria para solucionar la fragmentación externa siempre que la reubicación sea dinámica.

3. **Fragmentación en File systems ¿Por qué se produce? ¿Qué problemas ocasiona? ¿Cómo se evita?**

La fragmentación en un sistema de archivos se produce ya que los archivos son creados y eliminados y pueden aumentar o reducir su tamaño. Los archivos en disco ocupan un conjunto de bloques, estos pueden ser asignados de **manera contigua , enlazados entre sí, o utilizando una indexada**.

- Suponiendo que se realiza asignación de bloques contiguos, el problema surge a la hora de saber cuánta memoria (bloques) necesita un archivo, ya que en el momento en que este es creado no se sabe cuanto va a ocupar, por lo que se produce fragmentación interna ya que se le puede haber asignado más de lo que necesitaba.
  → Si se elimina un archivo, éste va a dejar bloques libres y si luego se desea crear uno que necesite una determinada cantidad de bloques, y en disco se posee pero de manera no contigua: se produce fragmentación externa.
- Con asignación indexada o enlazada se puede reducir o eliminar la fragmentación externa pero no la interna.

4. **Explique dos mecanismos para disminuir la fragmentación, realizar conversiones lógicas a físicas y permitir que dos o más procesos accedan a una misma región de memoria.**

Paginación: en este esquema de asignación, la memoria lógica se divide en bloques de tamaño fijo llamados páginas y la memoria física se divide en bloques del mismo tamaño que las páginas, llamados marcos. El tamaño lo decide HW.
Permite que el espacio de direcciones lógicas de un proceso sea de manera no contigua, por lo que se elimina la fragmentación externa. Pero la fragmentación interna sigue existiendo ya que el tamaño de las páginas es fijo.
Las direcciones lógicas se componen por un número de página y un desplazamiento. Este número de página se utiliza como índice en la tabla de página, la cual tiene punteros a las direcciones físicas correspondientes. La tabla de páginas se utiliza para hacer la traducción de direcciones.
Para mejorar la performance se utiliza una cache (TLB) para la tabla de páginas. Esta cache solo contiene unas pocas entradas a la tabla de páginas.
Contiene una clave (nro página) y un valor (nro marco).
Se compara el nro de página deseada con todas las claves: si está en la TLB, le da el nro de marco, sino va a la tabla de página.
La tabla de página es utilizada por la MMU para hacer traducciones.

Segmentación: el espacio de direcciones lógica se compone por un conjunto de segmentos de tamaño variable. Cada uno tiene un número de segmento y un desplazamiento.
Dado que el tamaño de los segmentos es de tamaño variable, se elimina la fragmentación interna. pero no la externa. Para solucionar la fragmentación externa se puede compactar la memoria para solucionar la fragmentación externa siempre que la reubicación sea dinámica.
El mapeo de direcciones lógicas y físicas se realiza a través de la tabla de segmentos. Cada entrada del segmento tiene dos entradas:

- Base: indica inicio de la tabla de segmentos en memoria
- Límite: longitud del segmento.

Se compara el límite del segmento con el desplazamiento, si es válido (L>d), se suma la dirección del registro base (base+desplazamiento).

5. **¿Tiene alguna ventaja diferenciar entre direcciones lógicas y físicas?**

   Una de las ventajas es que se independizan las direcciones de memoria generadas en TdeC con los que se encuentran libres en TdeE.
   La otra es que gracias a esto se puede hacer reubicación, compactación y protección de memoria tanto al SO como a los otros programas usuarios.
   - La **protección** evita que un programa de usuario pueda escribir en el espacio de direcciones Kernel o incluso en el espacio de direcciones de otro programa usuario, ya que con dicha operación el MMU(Unidad de gestión de memoria) generaría una trampa.
   - **Reubicación**: El programador no conoce que otros programas residirán en memoria en momento de ejecución. Mientras se está ejecutando el programa, puede que este se saque de memoria y se ubique en un almacenamiento secundario. La reubicación permite que luego ese programa pueda reubicarse en otro espacio de memoria diferente al que se encontraba anteriormente y se actualice la tabla MMU sin que el programa se entere del cambio.
   - La **compactación** Intenta solucionar fragmentación externa: consiste en desplazar las particiones ocupadas para que estén juntas en memoria quedando un solo hueco libre de mayor tamaño. Solo es posible si la reubicación es dinámica (en ejecución).

6. **¿Por qué es importante que el HW provea algún tipo de soporte para manejar direcciones lógicas y físicas?**

   La verificación de legalidad de cada dirección de memoria generada por la CPU y el mapeo de la misma en una dirección física no puede implementarse eficientemente en SW. Por lo tanto, estamos limitados por el HW disponible. De otra manera se desperdiciaría espacio de memoria para mantener datos administrativos y con ello disminuye la performance.

7. **A pesar de que la diferencia entre direcciones físicas y lógicas es ventajosa, también posee desventajas. Mencione dos de ellas y cómo son tratadas por los sistemas operativos modernos.**

   Respondida abajo.

8. **Las tablas de páginas de un sistema operativo pueden ser muy grandes y ocupar mucho espacio, especialmente si no se ocupa por completo el espacio de direcciones disponible. Describa una técnica para almacenar tablas de forma más eficiente del espacio. Analice cómo impacta esta técnica en tiempo de acceso a memoria.**

   Debido al gran volumen de la tabla de páginas, esta no puede ser almacenada en su totalidad en la memoria caché de la CPU por lo que debe ser almacenada en la RAM. Cuando se quiere tener una página primero debemos ir a MP para buscar la tabla de páginas y luego una vez que se sabe la ubicación de la página ir a buscarla. Esto genera un gran impacto en la performance del sistema.
   Es sabido que un proceso no está utilizando todas sus páginas activamente, sino un subconjunto de estas. Una solución a dicho problema (realizada mediante HW), es proveer una memoria muy rápida para almacenar esas entradas de la tabla que actualmente se están utilizando.
   A esta memoria muy rápida se la llama TLB. Aquí solo estarán aquellas entradas que están utilizando y cuando haya un fallo de TLB, recién ahí buscará dicha entrada en MP y la misma se reemplazará por alguna del TLB que no esté siendo actualmente referenciada, de esta manera si la frecuencia de aciertos de TLB es alta, se lograría mantener la performance como si tuviera toda la tabla de páginas en la caché.

9. La tabla de páginas multinivel se organiza en dos niveles ¿Por qué no más? ¿o menos? ¿Opina que existe algún límite práctico o teórico para el número de niveles posibles?

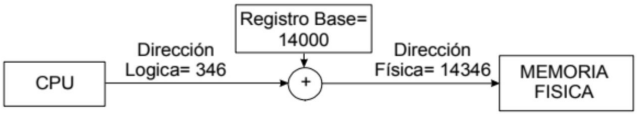
Límite práctico: la tabla de páginas multinivel permite que esta no esté cargada completamente en memoria, y como esta no entra completamente en la caché, se necesita al menos dos niveles de tabla: uno en la caché y otro en la RAM. Si queremos agregar más niveles se necesita una caché más grande con lo cual el procesador tendría un mayor costo.
Además existe el límite teórico de la cantidad de bits por página. Si hay muchos niveles, puede llevar a que exista una tabla por bit, lo cual llevaría muchas tablas.

10. ¿Por qué y para qué sirve la reubicación de procesos? Describa y compare dos formas de implementar reubicación.

Reubicación: vinculación del espacio de direcciones lógicas de cada proceso con direcciones físicas concretas (ese registro de reubicación tiene la dirección a partir de la cuál está cargado el programa).
La reubicación de procesos en memoria se utiliza cuando no se cuenta con demasiada memoria para albergar todos los procesos que se están ejecutando y es necesario reubicar los procesos para que otros entren.
La reubicación puede ser estática o dinámica:

- Estática: se puede realizar en tiempo de compilación (TdeC) o de carga. En TdeC el compilador genera código máquina con direcciones absolutas. Durante la carga, el código máquina contiene direcciones relativas al comienzo del programa, que se vinculan a direcciones físicas al comienzo de ejecución.
- Dinámica: permite traducir direcciones lógicas en físicas en TdeE. Para llevar a cabo dicha traducción se necesita de HW (MMU). Cuando el proceso en cuestión está siendo ejecutado, todos las referencias a memoria son reubicadas durante la ejecución antes de acceder a la memoria física.

La reubicación de procesos es útil para la compactación dinámica de memoria. Facilita el soporte de memoria virtual y proporciona una correcta administración de memoria.



11. Mencione métodos de administración de memoria que requieran reubicación para su implementación.

Paginación, Asignación continua, segmentación con paginación, swapping, segmentación.

12. Un sistema no soporta reubicación ¿Es posible soportar compactación de memoria? ¿Y memoria virtual? Justifique

No, la compactación sólo es posible si los procesos pueden reubicarse dinámicamente, debido a que el algoritmo consiste en mover todos los procesos hacia un extremo de la memoria y todos los huecos en la otra dirección.
La memoria virtual es posible, pero más costosa: los procesos que son intercambiados fuera de la memoria pueden volver a ocupar el mismo lugar, pero si este lugar está ocupado se deben escribir dichos bloques en almacén de respaldo, aunque haya otros lugares libres para reubicar el proceso.

13. **¿Cómo reduciría el impacto de la fragmentación de memoria en un sistema que no soporta reubicación? (leer por las dudas, no lo suelen tomar)**

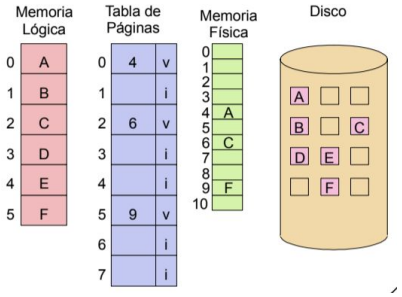
Se puede permitir que el espacio de direcciones lógicas de un proceso sea no contiguo.
Por el lado del SO, cada vez que un proceso solicita memoria se le asigna más de la que necesita. De este modo el proceso podría llegar a reutilizar la memoria que se le asignó o usar el espacio sobrante. Sin embargo con este enfoque se genera fragmentación interna.
Por el lado de la aplicación, lo que se puede realizar es hacer todas las particiones de memoria al principio y en el caso de no saber de antemano acerca de la necesidad de algunos espacios de memoria, lo que puede realizarse es reutilizar la memoria que se pidió con anterioridad, es decir, no devolver la memoria que se asignó sino reutilizarla.

Memoria virtual

- Permite separar la memoria lógica (del usuario) de la memoria física.
- Permite transferencia de información entre memoria principal y secundaria.
- Un proceso en ejecución, si es muy grande para caber en la memoria física, puede estar ocupando memoria virtual y física.
- Usa el disco como dispositivo de intercambio. (HW lo toma como almacenamiento auxiliar)
- Puede implementarse sobre Paginación o Segmentación paginada: se transfieren páginas **bajo demanda**, es decir que se traen a la MP las páginas del proceso que se necesitan.

Paginación por demanda

- Los procesos están divididos en **páginas**.
- Inicialmente las páginas que se usan de un proceso están cargadas en MP y el resto en disco.
- Contiene Tabla de páginas con un bit de validez: (HW se ocupa de marcarlo)
  - 1 si está en MP (v)
  - 0 si no está(i)
- Si el proceso accede a páginas residentes en memoria: la ejecución prosigue normalmente, sino ocurre una interrupción / fallo de página.



➔ Gestión de fallo de páginas

1. Se detecta que la página no está en memoria
2. Se produce interrupción
3. Se busca la página en almacenamiento secundario
4. Se busca un marco libre, se lee la página de almacenamiento secundario y se copia en el marco seleccionado (se pasa a la RAM)
5. Se actualiza la tabla de páginas
6. Reiniciamos en la instrucción interrumpida

- Caso extremo de fallo de páginas: comenzamos con la ejecución de un proceso sin ninguna página cargada en memoria. Se irán produciendo fallos de páginas sucesivamente y cargando las páginas necesarias.

Segmentación Paginada con Paginación por demanda

- No todas las páginas de todos los segmentos estarían en memoria
- Tiene una tabla de páginas asociadas a cada segmento con bits de validez
- El funcionamiento es igual que paginación por demanda

<u>Reemplazo de páginas</u>

Tenemos un proceso que tiene parte de sus páginas cargadas en memoria, pero no existe un frame libre en MP en caso de fallo de páginas.

- Posibles soluciones que aplicaría el SO:
  - Abortar el proceso de usuario (no es una buena solución)
  - Descargar otro proceso y llevarlo a almacenamiento secundario liberando sus marcos (se puede hacer)
  - Reemplazo de páginas: encontramos un marco que no está siendo utilizado y lo liberamos.

- Funcionamiento:
  1. Encontrar un marco que no se esté utilizando en ese momento y lo liberamos.
  2. Liberar un marco escribiendo en disco todo su contenido y modificando la tabla de páginas (y todas las demás tablas) para indicar que la página ya no se encuentra en memoria.
  3. Usar el marco para contener la página por la que falló el proceso.

Se requieren dos accesos a disco: una de entrada y otra de salida. Esto duplica el tiempo de servicio de fallo de página y aumenta el tiempo de acceso.

➔ Solución: utilizar un bit de modificación (dirty bit). Cada página puede tener asociado un bit de modificación en el hardware.
- ◆ Al cargar la página, desde disco a memoria, el HW pone el bit en 0 (no modificado)
- ◆ Si se escribe en la página el bit pasa a 1 (modificado)
- ◆ Si la página se elige como víctima se mira el bit, si fue modificada se salva

Con esto se reduce a la mitad el tiempo de E/S, si la página no ha sido modificada.

- Objetivo: generar la el menor número de fallos de página posible.
  - La tasa de fallos de página dependerá de:
    - Número de páginas de los procesos
    - Números de procesos en memoria
    - Número de marcos disponibles
    - Del algoritmo de reemplazo de páginas que se utilice
  - Estrategias de reemplazo:
    - Reemplazo Global: utilizar los algoritmos de reemplazo de páginas actuando sobre las páginas de todos los procesos
    - Reemplazo local: usar los algoritmos sólo entre las páginas del proceso que necesita un reemplazo de página
  - Algoritmos de reemplazo de páginas:
    - FIFO: reemplaza la página que lleva más tiempo en memoria (el rendimiento no siempre es bueno, pueden sustituirse páginas muy usadas)
    - Óptimo: Reemplaza la página que no se va a usar durante más tiempo (tiene la menor tasa de fallos)
    - LRU: sustituye la página que lleva más tiempo sin ser usada
  - Estrategias de asignación
    - Asignación fija equitativa: nro marcos/ nro procesos (reemplazo local)
    - Asignación fija proporcional: se asignan más marcos a procesos más grandes (reemplazo local)
    - Asignación variable: la cantidad de marcos de cada proceso varía dinámicamente según la necesidad del mismo (reemplazo local o global)

Hiperpaginación - Trashing

Se da cuando el número de fallos de páginas es elevado debido a que el nro de marcos asignados al proceso no es suficiente para almacenar las páginas referenciadas al mismo:

➢ El rendimiento de la CPU decrece debido a que está más tiempo en la cola de servicio de paginación que en CPU

➔ Solución: controlar la tasa de fallo de página: si es demasiado alta, necesita más frames.

1. **¿Qué es trashing y por qué se produce? (Hiperpaginación)**

   Un proceso entrará en trashing si invierte más tiempo implementando los mecanismos de paginación que en la propia ejecución del proceso. (muchos fallos de página por no dar suficientes frames)
   Por ejemplo, si el proceso no tiene el número de frames que necesita para soportar las páginas que están usando activamente, generará rápidamente fallos de páginas. En este momento, deberá sustituir alguna página. Sin embargo, como todos sus páginas se están usando activamente, se verá forzado a sustituir una página que volverá a necesitar enseguida. Como consecuencia, vuelve a generar rápidamente una y otra vez sucesivos fallos de páginas que se ven forzados a recargar inmediatamente.

   ¿Cómo evitarlo? Para evitar el trashing se debe controlar la tasa de fallo de página, si es demasiado alta, necesita más frames. Se tiene dos modelos principales: modelo del working set y modelo de la frecuencia del fallo de página.

2. **¿Qué es la anomalía de Belady? ¿cómo se evita?**

   La anomalía de Belady se da cuando al utilizar un algoritmo de sustitución de páginas, la tasa de fallos de páginas se incrementa a medida que aumenta la cantidad de frames asignados.
   Se evita utilizando el algoritmo de sustitución óptimo o algoritmos que sustituyen la página que no fue utilizada en el periodo más largo como LRU.
   El algoritmo del tipo FIFO puede generar esta anomalía.

3. **Explique el modelo del conjunto de trabajo (working set) y su aplicación para memoria virtual.**

   La técnica del working set surge del principio de la localidad, dicho principio dice que un proceso a lo largo de su vida se mueve en distintas localidades siendo estas también de tamaño variable.
   - El working set busca asignarle a cada proceso un mínimo de páginas que garantice que pueda correr razonablemente. Dado que asignarle más páginas genera un desperdicio de memoria y asignarle menos páginas seguramente trashing.

   Esta técnica impide hiperpaginación (trashing) y aumenta la multiprogramación (más espacio de memoria para más procesos).

4. **Los mecanismos de memoria virtual se basan en una suposición acerca de la forma en que los procesos hacen uso de la memoria con el fin de evitar impactar significativamente en la performance de ejecución de dichos procesos.**

   A) **¿Cuál es la superposición en la que se basan los mecanismos de MV?**

   El hecho de que para ejecutar un programa este deba estar totalmente cargado en memoria, genera la suposición de que como en realidad no se usan todos los datos de un programa, entonces se cargan páginas en memoria que en realidad nunca se leen, por lo tanto se desperdiciaría el espacio que se podría usar para otro proceso. Entonces se deduce la cantidad de páginas que puede ocupar cada proceso y se le asignan ese número a cada uno de los que se van a cargar en memoria.

   B) **¿Qué sucede si dicha suposición no se cumple la mayor parte del tiempo?**

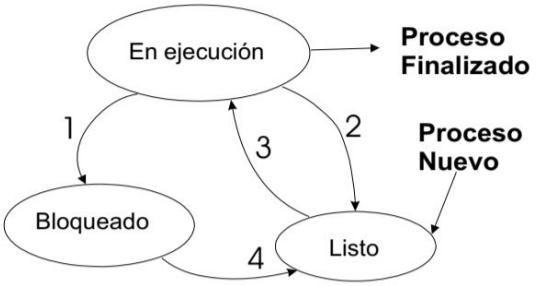
   Genera fallo de página, lo cual hace que deban intercambiarse páginas.

   C) **Esta suposición también se aplica a las técnicas de coaching ¿Por qué?**

   Porque si los fallos de página se generan en los momentos que la memoria esté llena, se comienza a generar sobrepaginación.

Estado de un proceso

1. Pasa a esperar un suceso y se bloquea.
2. Expulsión del proceso a la CPU
3. EL planificador elige otro proceso
4. El suceso que esperaba el proceso acaba.



Round Robin (Turno rotativo)

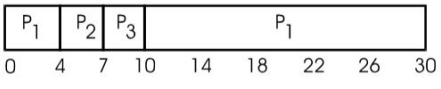
- La cola de procesos es circular (FIFO).
- EL planificador la recorre y **asigna un tiempo máximo de CPU a cada proceso** (Q cuanto tiempo).
- Un proceso puede abandonar la CPU:
  - Libremente si ráfaga de CPU<Q.
  - Después de una interrupción si ráfaga de CPU>Q.
- Diseñado par sistemas de tiempo compartido.
- Es un esquema de planificación expulsiva.

Ejemplo:

Proceso	Duración Ráfaga CPU
P_1	24
P_2	3
P_3	3

Cuanto de timepo Q = 4

$$T_P = \frac{(0+4+7+(10-4))}{3} = 5{,}66$$



SJF apropiativo

Este algoritmo da prioridad a los procesos más cortos a la hora de la ejecución. Primero se ejecutan los procesos que tarden menos ciclos de reloj.

Protección

1. ¿Qué es protección? ¿Para qué sirve? ¿Porque es crucial para el SO con multiprogramación?

La protección es un mecanismo para controlar el acceso de algún programa (proceso o usuario) tanto al sistema operativo como a otros procesos.
→ Un proceso no debería de interferir en la ejecución de otro proceso, por ejemplo modificando a este, y tampoco un proceso podría generar un mal funcionamiento de algún recurso de la computadora, como por ejemplo adueñarse del cpu generando así que los demás procesos no puedan utilizar la cpu rompiendo el paradigma de multiprogramaciòn.
En los sistemas con multiprogramación se deben proveer mecanismos de protección debido a que varios procesos pueden estar alojados en memoria al mismo tiempo, y se debe proveer para cada uno su propio espacio de direcciones y que este no pueda salir de allì, ya que si lo haría podría modificar a otro proceso.

2. ¿Cómo se logra la protección entre procesos?

Se determinan rangos de direcciones legales a los que un programa puede acceder y se protege a la memoria fuera de dicho espacio.

- Para realizar esta protección se utilizan dos registros uno base y uno limite. El registro base contiene la dirección física de memoria menor o inicial válida y el registro límite posee el tamaño del rango.
- Esta protección se realiza mediante el hardware de la CPU comparando cada dirección generada en modo usuario con los registros. Cualquier intento de un programa que se ejecute en modo usuario por acceder a la memoria del monitor o de otros usuarios da como resultado que se active una trampa que se comunica al monitor el cual trata al intento como un error fatal.
- Los registros base y límite sólo pueden ser cargados por el SO el cual emplea un instrucción privilegiada especial.

3. ¿Cómo se logra la protección en HW?

Se logra utilizando distintos modos de operación: modo usuario y modo monitor (o privilegiado):

1. Se crea un bit de modo que es agregado al hardware e indica si se está operando en modo usuario (1) o monitor (0).
2. Cuando inicia el SO el hardware inicia en modo monitor, cuando se carga y comienzan los procesos de usuario se pasa a modo usuario.
3. Cuando ocurre una interrupción el hardware conmuta de modo usuario a modo monitor.

La protección se logra designando como instrucciones privilegiadas a aquellas que puedan causar daño.
El hardware permite que las instrucciones privilegiadas se ejecuten solo en modo monitor. Si se intenta ejecutar en modo usuario se activa una trampa al sistema operativo.

4. ¿Es posible lograr protección de HW sin que el CPU provea algún soporte? ¿Cómo?

Podría construirse el SO como una máquina virtual de tal forma de crear una capa de abstracción entre el hardware y el SO. Para cada instrucción entonces se verificaría si es o no una instrucción permitida. Este enfoque tiene costos en la performance.

Sistemas de archivos

1. **¿Qué significa que un archivo/sistema de archivos posea inconsistencias?**

   Significa que el estado real de algunos archivos no será el que se describe en las estructuras de directorio.

2. **Describa al menos dos inconsistencias típicas y posibles métodos para solucionarlos de forma automática.**

   1. Una posibilidad es que cuando se cree un archivo se le asigne un inodo (nodo indice-arreglo) y se cree el archivo en disco y antes de marcar en el bitmap que ese inodo está en uso, ocurra un fallo en el sistema. Entonces en el bitmap va a figurar que este inodo no está siendo usado, pero en realidad si ya que va a tener información.

   → Solución: cuando se reinicie el sistema, se verifique cuáles son los inodos que están siendo usados y cuales no, y se compare eso con el bitmap. Si se encuentra con que un inodo que tiene información está marcado como no usado, se lo marca como usado.

   2. Otra posibilidad es que en el bitmap un inodo figure como usado pero no pertenezca a ningún archivo porque por alguna razón no se liberó correctamente.

   → Solución: el check-disk genere un archivo en ese bloque para que el usuario lo analice.

   (Existe una tabla de inodos activos que está en memoria (en uso) e incluye un campo que apunta a los bloques de datos en disco).

3. **Explique cómo hacen los SO para detectar y corregir inconsistencias en Sistemas de archivos.**

   Es común que se corra un programa especial (verificador de consistencias) a la hora de volver a arrancar la computadora para verificar y corregir inconsistencias de disco. El verificador de consistencias compara los datos en la estructura de directorios y trata de corregir las inconsistencias que encuentre.
   Los algoritmos de asignación de administración libre determinan qué tipos de problemas pueden encontrarse y el grado de éxito que tendrá la corrección de dichos problemas.

4. **Se desea agrandar o reducir el tamaño en un sistema de archivos. Explique cómo realizar eso en sistemas basados en FAT o INODOS**

   FAT: debido a que la tabla de asignación de archivos se encuentra al principio de la partición y luego se almacenan los archivos, si se desea cambiar el tamaño de dicha partición, primero se deben correr todos los **archivos** hasta el final del disco con el fin de ganar espacio para agregar las nuevas entradas en la FAT. Luego se agregan (o eliminan en caso de querer reducir tamaño) las entradas en la FAT y por último se modifica dicha tabla junto con directores estableciendo los nuevos bloques donde se encuentran almacenados los archivos.

   ¿Qué es FAT? Al principio de una sección del disco hay una tabla FAT (tabla de asignación de archivos), la cual tiene una entrada para cada bloque del disco. Esto mejoró el tiempo de acceso aleatorio.

   INODOS: El cambio de tamaño de una partición con sistema de archivos Unix debe realizarse de manera similar. El arreglo de inodos se mantiene al principio de la partición: al agrandar la misma, se deben correr todos **bloques** de datos y agregar entradas al vector de inodos, ya que el mismo es proporcional al tamaño de partición.

5. **Describa al menos dos inconsistencias que podrian ocurrir en sistemas de archivos FAT. Explique formas de detectar y arreglar esas inconsistencias.**
   - Archivos cruzados: la entrada en el directorio de un archivo apunta al primer clúster de una cadena. Cada cadena debe ser única: ningún cluster debe pertenecer a más de un archivo.
     → Si dos archivos incluyen al mismo cluster, esto indica una inconsistencia y la única forma de resolverla es truncar uno de los archivos en el punto inmediato anterior al cruce.

   - Cadenas perdidas o huérfanas: cuando hay espacio marcado como ocupado en la FAT pero no hay ninguna entrada de directorio haciendo referencia a ella, el espacio está efectivamente bloqueado y desde la perspectiva del usuario: utilizado.

6. **Explique el funcionamiento de los sistemas RAID**

   El sistema RAID combina varios discos en una sola unidad lógica de manera tal de tener seguridad, velocidad y tolerancia a fallos. Distribuye los datos en los discos de distinta forma dependiendo del nivel. Exceptuando el nivel 0, todos tienen tolerancia a fallo por lo que si valla un disco, el sistema continuará trabajando.
   El SO y el usuario solo ven un disco.

   Ventajas : más seguridad, mayor tolerancia a fallos, mayor rendimiento, mayor capacidad

7. **Describa RAID y sus variantes**
   - Raid 0: Este nivel no tiene redundancia o tolerancia a fallos: distribuye de manera equitativa los datos entre 2 o más discos duros utilizando Round Robin
   - Raid 1:Tiene redundancia y tolerancia a fallos: duplica la información en dos discos. Si un disco se estropea, el sistema seguirá funcionando. Aumenta el rendimiento de lectura.
   - Raid 2: No duplica info: divide los datos a nivel bits en lugar de bloques. No muy usado. Utiliza código de Hamming.
   - Raid 3: Guarda los datos a nivel de Bytes y utiliza un disco de paridad: cada dos o más discos se guarda un bit de paridad en un disco extra. Entre los discos que guardan info se intercalan los bytes de los datos.
   - Raid 4: Similar al Raid 3 pero guarda los datos a nivel de bloques.
   - Raid 5: Guarda los datos a nivel de bloques y distribuye la información de paridad entre los discos miembros del conjunto. Mínimo 3 bloques.
   - Raid 6: Amplía el nivel Raid 5 pero tiene unidades de doble paridad: divide los datos a nivel de bloques y distribuye los dos bloques de paridad entre todos los miembros del conjunto.

8. **¿Por qué generalmente Raid se implementa en HW?**

   Porque requiere acceso a varios discos. La implementación mediante SW no sería eficiente dado que este solo ve un disco y no varios. Para ello es necesario dar soporte de HW para dicha implementación. También es necesario un SO que administre dicho HW.

9. **¿Considera a RAID un complemento o un reemplazo de los sistemas de archivo log? ¿Tiene sentido combinar RAID con log? Compare RAID vs log.**

   RAID y log son un complemento. Un sistema con logs brinda confiabilidad a nivel sistema de archivos mientras que RAID a nivel HW.
   Utilizar ambos aumenta la confiabilidad del sistema ya que si solo se posee RAID pero los datos son inconsistentes, se guardará redundancia también inconsistente. Por ejemplo, si se está escribiendo un

dato en los distintos discos de un RAID y se corta la luz, los discos quedarían inconsistentes. Pero si se utiliza el sistema de logs, se reanudaría la tarea que se estaba realizando → Evita inconsistencias.

- ➔ RAID provee tolerancia a fallos sobre los datos del usuario y del sistema, en cambio log provee un mecanismo para mantener siempre consistentes las estructuras de los sistemas de archivos.
- ➔ Log disminuye la performance del sistema debido al procesamiento de las transacciones y RAID lo incrementa.
- ➔ El sistema con logs no tiene más que un costo por la implementación en SW, mientras que RAID tiene alto costo debido a que se necesitan múltiples discos para implementarlos.

Planificación de CPU

1. **Explique qué es un planificador de CPU con desalojo.**

   Es un planificador que permite a los procesos combinar entre estar en ejecución a estar en la cola de espera o de listos. Permite que cuando se produce una interrupción, el proceso deje de ejecutar y pierda el control de la CPU.
   En el caso de estos planificadores, el proceso que está ocupando la CPU no necesariamente es el que la libera. Sino que el desalojo puede ocurrir porque se agotó su tiempo de ejecución o porque llegó un proceso con mayor prioridad.
   Ej: Round robin y SJF apropiativo.

2. **¿Cómo influye en la programación de aplicaciones multihilo un planificador con desalojo con respecto a uno sin desalojo?**

   El planificador **con desalojo** es más costoso ya que hay que coordinar el acceso a datos compartidos. Esto da por resultado secciones críticas de código donde para acceder a ellas, es necesario recurrir a técnicas de sincronización para evitar estados inconsistentes o errores de los datos. Cuando la aplicación se ejecuta, si hace una llamada al sistema, la CPU se va a conmutar con otro thread.
   Si se ejecuta con un planificador **sin desalojo**, la app va a ejecutar más lento porque cuando haga la llamada al sistema la CPU va a quedar ociosa hasta que termine, y el grado de multiprogramación se va a reducir. La solución es que el programador de la aplicación, cuando la escriba, haga llamadas al sistema cada tanto y que en esa llamada al sistema, el SO llame a un scheduler que conmueve los threads.

3. **¿Posee alguna ventaja en los sistemas interactivos? Justifique.**

   Si, en este tipo de sistema por lo general se implementa un round robin porque hay muchas operaciones de I/O. Otra de las ventajas que se presenta es que si tengo dos o más programas de usuario, cuando ejecuto uno en la CPU, si selecciono a otro para que corra en primer plano, se desaloja el anterior y se ejecuta el nuevo pasando a estar en primer plano.

4. **¿Posee alguna ventaja en sistemas de tiempo real?**

   En los sistemas de tiempo real estricto el desalojo no posee ventajas ya que es necesario que el sistema conozca a priori el tiempo que demora cada proceso y con desalojo el tiempo no será determinístico.
   En los sistemas de tiempo real suave, el desalojo tiene sentido ya que la planificación se basa en prioridades.

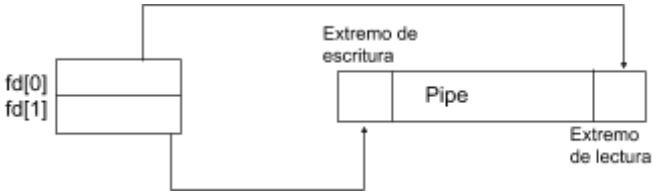
Comunicación entre procesos

1. ¿Qué es un pipe y cómo se utilizan? (punto de vista de la programación)

- Consiste en una cadena de procesos conectados de tal forma que la salida de cada elemento de la cadena es la entrada del próximo.
- Permite comunicación y sincronización entre procesos.
- Permite el flujo confiable de los bytes unidireccional entre dos procesos.
- Su tamaño es fijo, por lo que la lectura es un pipe vacío o la escritura en un pipe lleno hace que el proceso se bloquee hasta que el pipe cambie de estado.
- Todo lo que un proceso lea del pipe se elimina del mismo, es decir, no está disponible para otros procesos lectores.
- Un beneficio del tamaño pequeño de los pipes es que los datos de ellos raras veces se escriben realmente en disco. La caché normal de buffers de bloques generalmente lo mantienen en memoria.

2. Explique la comunicación entre procesos utilizando pipes

Un pipe conecta la salida de un proceso en la entrada del otro. Para realizar esto, el SO asocia un buffer (almacenamiento temporal) al pipe y cuando uno de los procesos quiere leer o escribir del pipe, se lanza una llamada al sistema que chequea el buffer asociado, ve si hay datos en este (en el caso de que se busque leer) o verifica si hay espacio suficiente (si busca escribir).



- Un proceso realiza la llamada al sistema **pipe()**. El SO reserva un espacio de memoria en el espacio de direcciones del kernel y retoma al proceso llamando dos descriptores de archivos que son agregados a la tabla de archivos abiertos del proceso. Los descriptores apuntan a los extremos del pipe de manera que uno es usado para la escritura y otro para la lectura.
- El proceso puede escribir y leer en los descriptores de archivos devueltos por el SO como si lo hiciera con un archivo normal: con los llamados **write() y read()**.
- Todo lo que un proceso lea del pipe se elimina del mismo, es decir, no está disponible para otros procesos lectores.

Desde el punto de vista del SO y las apps, los pipes son un mecanismo para poder comunicar un proceso padre con su hijo:

- Cuando se hace una llamada **fork()**, el hijo hereda la tabla de descriptores de archivo del padre, por lo tanto también hereda el pipe previamente abierto y se crea así un canal de comunicación.

3. Detalle cómo se tratan los problemas de sincronización con pipes

La sincronización se maneja con exclusión mutua sobre el buffer. Mientras el productor escribe, el otro "extremo" está bloqueado para el consumidor. Por su parte mientras el consumidor lee, el productor no puede escribir. Además, la lectura es un pipe vacío o la escritura en un pipe lleno hace que el proceso se bloquee hasta que el pipe cambie de estado.
Desde el punto de vista del SO y las apps, el problema de sincronización se soluciona de la siguiente manera:

- Si un proceso intenta leer un pipe cuyo extremo de lectura ha sido cerrado, la llamada read() retorna 0 indicando fin del archivo.
- Si un proceso intenta escribir en un pipe cuyo extremo de escritura ha sido cerrado, la escritura falla y el SO envía una señal (sigpipe) la cual por defecto, que termina con el proceso de escritura.
- Si un proceso intenta leer de un pipe vacío cuyo extremo de escritura se encuentra abierto, el proceso se duerme hasta que haya elementos en el pipe.
- Si un proceso intenta escribir en un pipe menos bytes de los que este puede almacenar, se garantiza que write() es atómico.
- Si un proceso intenta leer más bytes de los que se encuentran almacenados en el pipe, read() devuelve todo el contenido del pipe y la cantidad de bytes leidos.
- Si un proceso intenta escribir más bytes de los que el pipe puede almacenar, no se garantiza la atomicidad de la escritura.

4. Compare pipes con memoria compartida y pasaje de mensajes

	Pipes	Pasajes de mensajes	Memoria compartida
Velocidad	Para escribir o leer del buffer del pipe se implementa como una llamada al sistema y hay que hacer un cambio de contexto	Para hacer pasaje de mensajes casi siempre hay que hacer una encriptación. Esto suele llevar tiempo	Es más rápida ya que no hacen falta llamadas al sistema y se maneja a nivel aplicación
Uso de memoria	Puede implementarse en memoria o disco. De todas formas depende del tamaño del buffer	Dado el overhead de información que lleva el mensaje, este consume más memoria	Al compartir la memoria no hace falta que se dupliquen los datos para dos o más procesos
Facilidad de uso	Solo hace falta poner el comando "|" entre los procesos	Para comunicar dos procesos hay que implementar las funciones de envío y recepción	Se suelen utilizar librerías que permiten compartir los datos en memoria
Seguridad	Es seguro ya que se comparte información entre dos procesos y solo estos tienen acceso del pipe	Se suele utilizar encriptación debido a posibles interceptores en la comunicación	Solo los procesos que comparten memoria tienen acceso a la región de memoria

Bibliotecas de vinculación dinámica (DDL)

1. **¿Qué es y cómo funciona?**

   Una biblioteca de vinculación dinámica es un archivo ejecutable que actúa como una biblioteca de funciones compartida. La vinculación dinámica proporciona a los procesos una forma de llamar a una función que no forma parte del código ejecutable. Este código está en un archivo DDL, que contiene una o más funciones que se compilan, vinculan y almacenan de forma independiente de los procesos que las utilizan. Los archivos DDL también facilitan el uso compartido de datos y recursos. Distintas apps pueden tener acceso simultáneamente al contenido de un mismo archivo DDL en la memoria.

2. **Ventajas y desventajas respecto a las bibliotecas de vinculación estática.**

   Ventajas
   - Reducen el tamaño de los archivos ejecutables: gran parte del código puede estar almacenado en bibliotecas y no en el propio ejecutable, lo cual genera una mejor modularización. En las estáticas la biblioteca se incluye en el ejecutable.
   - A diferencia de las estáticas, las dinámicas pueden ser compartidas entre varias apps.
   - En los DDL a la hora de la actualizar las librerías, simplemente se reemplazan las que actualmente están en el sistema. En cambio, en la estática se necesita una recopilación del código.

   Desventajas
   - El gasto de cargar cada librería (cosa que no ocurre en el estático). Esta carga se hace en TdeE y hace más lenta la ejecución de la app.

3. **En una biblioteca de vinculación dinámica alguno de sus procedimientos mantienen estado en variables dentro de la biblioteca. La biblioteca se prueba con un único proceso/hilo miles de veces sin problemas, pero cuando se usa con más de un proceso/hilo dichos procedimientos fallan.**
   a. **¿Qué sucede y porqué?**
   b. **¿Cómo solucionaría el problema sin reducir la concurrencia (es decir, sin utilizar mecanismos de exclusión mutua?**

   El problema es que la variable de estado está siendo compartida por más de un proceso pero los estados deberían ser distintos para cada proceso y no manejarse internamente. Lo que hay que hacer es poner la variable de la DDL como parámetro de la función de la DDL.

4. **Suponga que varios procesos comparten una única copia de una biblioteca de vinculación dinámica ¿Qué problemas podrían ocurrir? ¿Cómo los soluciona?**

   Si la biblioteca no es de código rentable podría suceder que dicha biblioteca contenga variables de estados en donde si un proceso modifica dichas variables podría generar que los otros no se comporten de forma correcta al utilizar la biblioteca.
   La forma de solucionar este inconveniente es que dicha biblioteca sea de código rentable (o puro), o realizando la compactación de memoria marcando las páginas de la biblioteca como WOC por lo que cuando alguien modifica alguna página se cree una copia de dicha página en su espacio de direcciones. Esto podría generar problemas a la hora de ver cuál es la última versión modifica de dicha biblioteca para luego volverla a volcar a memoria.

Copy on write

1. **¿Qué es y cómo funciona? ¿Cómo se implementa?**

   COW es una técnica similar al compartimiento de páginas.
   Permite a los procesos padre e hijo inicialmente compartir las mismas páginas.
   Estas páginas son marcadas como páginas COW, lo que significa que si alguno de los dos procesos modifica alguna de estas páginas compartidas se crea una copia de la misma.
   ¿Cómo? Cuando un hijo la quiere modificar a la página, se le asigna una página vacía, se le copia la página del padre y se la mapea al espacio de direcciones del hijo, así este la modifica.
   Las páginas de sólo lectura no es necesario marcarlas ya que las mismas no se pueden modificar y por lo tanto no necesitan ser copiadas.
   La técnica es utilizada para crear procesos de una manera eficiente debido a que una página no se copia hasta que no se modifica. También es útil para disminuir la demanda de página en sistemas con memoria virtual. Este mecanismo resulta útil porque se utiliza para evitar que un proceso escriba en la memoria de otro cuando la misma se está compartiendo.

2. **¿Tiene alguna relación con la llamada del sistema Fork()? Explique**

   Con Fork no tiene relación ya que este crea un duplicado exacto del proceso original y luego del Fork, todas las variables tienen valores idénticos. Dado que los datos del padre se copian para crear el hijo, los cambios subsecuentes en uno de ellos no afectará el otro.

Entrada/Salida

1. **Generalmente las operaciones de I/O a nivel de usuario se realizan de a bytes, mientras que los dispositivos sólo son capaces de leer/escribir de a bloques de tamaño fijo (ej 4KB). ¿Cómo se manejan estas diferencias a nivel sistema? ¿Por qué cree que los dispositivos de almacenamiento operan con bloques en vez de bytes?**

   Los SO proveen un conjunto de interfaces que permiten tratar de una forma estándar y uniforme a los diferentes dispositivos de I/O.
   Por ejemplo: un read() sobre un disco va a tener un bloque, pero un read() sobre un teclado va a traer el carácter que el usuario haya tipeado.
   Los dispositivos de almacenamiento operan con bloques debido a la velocidad de transferencia.
   Si el disco interrumpiera por cada byte que lee del disco al CPU, el CPU estaría continuamente recibiendo interrupciones y no podría atender tareas de los procesos, por eso generalmente las operaciones se realizan a nivel de bloques y generando adecuadas interrupciones cuando dicho bloque se encuentra disponible.

2. **Cuando varios procesos realizan entrada/salida, a veces los SO atienden esos pedidos sin respetar el orden en que fueron realizados ¿Por qué no se respeta el orden? ¿Puede traer problemas?**

   Los SO no respetan el orden para brindar un mejor tiempo de respuesta. Cuando un proceso realiza un pedido de I/O, dicho proceso es puesto en una cola de espera hasta que el dispositivo esté disponible y este atienda el pedido de I/O del proceso.
   Esta cola de espera de dispositivos puede contener varios procesos (pedidos) y se atiende el pedido planificando cual va a mejorar la performance.

Núcleo del SO

1. **¿Qué es un microkernel?**

   Es un método que estructura el SO eliminando todos los componentes no esenciales del kernel e implementandolos como programas a nivel usuario del SO.
   El resultado es un kernel más pequeño. Estos proporcionan una gestión de memoria y de procesos mínima, además de un mecanismo de comunicaciones.

2. **¿Por qué se utilizan?**

   Se utilizan para sistemas de tiempo real debido a su simplicidad que evita que se produzcan variaciones de tiempo inesperadas en la ejecución de procesos.

3. **Sistemas monolíticos y tipo kernel**
   a. **¿Cuál es más fácil de diseñar?**
   El monolítico porque prácticamente no existe diseño: todos los sistemas son un solo programa ya que no existe división de funcionalidades.

   b. **¿Cuál es más fácil de mantener?**
   Los microkernel porque al estar divididas las funcionalidades es fácil de identificar dónde se deben realizar los cambios. Además los cambios tienden a reducirse debido a que el microkernel es un kernel más pequeño.

   c. **¿Cuál es más fácil de extender?**
   Los microkernel porque todos los servicios se agregan al espacio de usuario y en consecuencia no requieren modificación de kernel.

   d. **Compare ventajas y desventajas**
   Los monolíticos proveen muchas funcionalidades en poco espacio con lo cuál sirve para restricciones de HW. En cambio, los microkernel son fáciles de extender, de portar a nuevas arquitecturas de HW, más confiables y seguros. Estos dos últimos porque la mayoría de los servicios están ejecutándose como procesos de usuario.

1. **Generalmente los SO no hacen nada para evitar ni prevenir deadlocks. ¿Se puede hacer algo para que las apps multihilo que uno escribe no posean deadlocks? (sin modificar el núcleo del SO, bibliotecas de hilos ni el lenguaje de programación).**

   Para la prevención de deadlocks se necesita evitar que una de las cuatro situaciones se den en el mismo momento.
   Estas son:
   - Exclusión mútua: es difícil de prevenir porque algunos recursos si o si no son compartibles.
   - Agarrar y esperar: se puede hacer que a un proceso no se le asigne CPU hasta que no pueda disponer de todos los recursos que necesita.
   - Sin desalojo: se necesita que exista desalojo de los recursos que ya han sido asignados. Tiene dos protocolos
     1. Si un proceso que está usando algún recurso, pide otro que no puede ser asignado inmediatamente (osea que el proceso debe esperar), entonces los recursos que tenía en su poder le son quitados: el proceso continuará sólo si se le pueden asignar los viejos recursos que tenía como también los nuevos.
     2. Si un proceso **p** pide recursos, primero se chequea si están disponibles . Si lo están se les asigna. Sino, se chequea si los recursos que necesita están asignados a algún otro proceso que está esperando por otros recursos, luego en este caso se le quita los recursos para asegurarlos al proceso **p**. De lo contrario debe esperar. Mientras, se le pueden sacar los recursos que tiene y para continuar necesita recuperar esos y que se le asignen los nuevos.
   - Espera circular: cada proceso puede pedir los recursos dada una lista de recursos con su respectiva numeración, en un orden creciente de numeración.

2. **“Gracias a la multiprogramación es posible que dos o más procesos se completen en un tiempo menor que si se ejecutan de manera secuencial, incluso en una computadora con un solo procesador”. Explique por qué esto es posible, incluso si el procesador puede ejecutar una sola instrucción en cada instante.**

   En la multiprogramación cuando un proceso realiza un pedido de I/O, este es pasado a una cola de espera, y al procesador se le da un nuevo proceso para que no esté ocioso; luego cuando el pedido se haya completado, el proceso en espera es llevado a una cola de listos para que pueda en algún momento poder seguir en ejecución cuando se le brinde CPU.

   a. **¿Puede ocurrir lo mismo con los hilos en vez de procesos?**

      Esto va a depender de si el SO soporta hilos a nivel kernel. Si es así sí ya que sucedería lo mismo que lo explicado anteriormente pero con hilos: en vez de dormir un proceso, duermo un thread. En cambio, si el SO no soporta hilos a nivel kernel, este no estará consciente de que el proceso contiene hilos y cuando uno de ellos realice un pedido de I/O, el sistema bloqueará a todo proceso, sin dejar avanzar al otro hilo.

3. **Lon SO que soportan multiprogramación típicamente proveen mejor utilización de CPU que los monoprogramados. Sin embargo, los sistemas monoprogramados poseen ventajas y aplicaciones ¿Cuáles?**

   Ventajas
   - Son bastante simples de construir.
   - Poseen un sistema de batch simple.
   - No tienen CPU scheduling.
   - No hay administración de memoria.
   - No hace falta mecanismos de evasión, detección o recuperación de deadlock o sincronización de procesos.
   - No se necesita protección entre diferentes procesos de usuario.
   - Un mismo procesos se ejecutará en el menor tiempo posible ya que nunca será desalojado de la CPU a menos que termine.
   - Son apropiados para ejecutar trabajos grandes que no necesitan interacción con el usuario.

   Aplicaciones
   - Sistemas de tiempo real blandos que requieren mucho uso del CPU. No necesitan esperar para realizar E/S.
   -

4. **¿Cuáles son las diferencias principales entre las operaciones de semáforos wait/signal y las operaciones wait/signal de las variables de condición de los monitores?**
   - Cuando un proceso opera **wait** sobre una variable de condición de **monitores**, el proceso se suspende y se coloca la variable en una cola asociada. En cambio, en el **semáforo** no siempre la operación suspende el proceso que la emite, depende del estado del mismo.
   - La operación **signal** sobre una variable de condición de **monitores** libera un procesos suspendido en la cola de dicha variable, pero si no hay ningún proceso suspendido, la operación signal no tiene efecto. En cambio, con el **semáforo** signal siempre afecta el estado del mismo (aunque no haya procesos suspendidos).
   -

5. **¿Por qué generalmente se evitan las esperas activas (busy/waits) cuando se programan aplicaciones concurrentes?**

   Porque se desperdician ciclos de reloj ya que el hilo o proceso está ejecutando un bucle dentro del procesador hasta que cambia alguna condición. Estos ciclos de reloj podrían ser utilizados por otros procesos u otros hilos.

6. **¿Cómo se evitan las esperas activas?**

   Las esperas activas se pueden producir, por ejemplo, cuando se utilizan mecanismos de sincronización de semáforos: se producen cuando un semáforo hace un wait(s) y se desarrolla así: wait(s) { while (s<=0) do s--;}
   Como se puede observar el proceso, si el semáforo es igual a 0, se bloquea activamente realizando un while.
   La manera de evitar esto, es hacer que el proceso se ponga a dormir él mismo.

7. **A pesar de que las esperas activas son indeseables en muchos casos, se utilizan frecuentemente dentro del SO ¿Por qué?**

   Los SO cuando desalojan a un proceso del procesador, deben almacenar todos los registros que esté utilizando este proceso, el PC counter, etc. Esta información es almacenada en el PCB del proceso. A esto se lo conoce como context switching, y esto tiene un overhead. Por lo tanto, en aquellas situaciones en que la espera del proceso va a ser poca, a veces conviene realizar la espera activa antes de realizar el context switching.

Verdadero o falso

A) Se dice que los archivos mapeados de memoria son más rápidos que los archivos accedidos de manera común (open, read, write, close, etc)

Falso, si el archivo mapeado a memoria está bien implementado, la performance de uno con respecto al otro no presenta variaciones. Los archivos mapeados se suelen utilizar porque son más cómodos.

B) La memoria virtual no necesita ningún soporte por parte de HW para ser implementada eficientemente

Falso, la técnica de memoria virtual se implementa mediante el mecanismo de paginación por demanda o segmentación por demanda, los cuales necesitan HW para implementarse eficientemente y correctamente

C) Copy or Write sirve para acceder con facilidad a archivos de gran tamaño

Falsa, COW es una técnica que se utiliza para asignar páginas entre procesos hijos y padres. Para poder acceder con facilidad a archivos de gran tamaño se utiliza archivos mapeados a memoria.

D) Los mecanismos de reubicación sólo son necesarios cuando se utiliza memoria virtual

Falso, los algoritmos de reubicación también se utilizan cuando la memoria primaria posee fragmentación externa y es necesario completarlo. Los mecanismos de reubicación son necesarios en todos los SO multiprogramados, ya sea que su mecanismo de asignación de memoria sea contiguo, por paginación o por segmentación.

E) Las bibliotecas de vinculación dinámica requieren de mecanismos para compartir áreas de memoria entre procesos

Verdadero, una vez que la biblioteca fue cargada en memoria, todos los procesos que la necesiten accederán a la misma región de memoria donde fue alojada la misma. Se usan mecanismos para compartir, no mecanismos de sincronización ya que el código de las bibliotecas debe ser reentrante.

F) El diferenciar entre direcciones lógicas y físicas tiene desventaja

Verdadero. Se produce un overhead (gasto) en el tiempo de acceso a memoria debido a que el mismo se realiza mediante una indirección. Y además dependiendo del esquema de administración de memoria que se utilice, puede haber fragmentación externa o fragmentación interna.