Gestión de procesos:

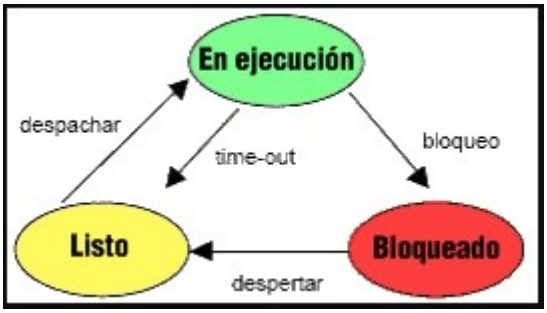
Un proceso en un programa en ejecución. Un proceso simple tiene un hilo de ejecución (o subproceso), en ocasiones, un proceso puede dividirse en varios subprocesos. Un hilo es básicamente una tarea que puede ser ejecutada en paralelo con otra tarea. Por lo que los hilos de ejecución permiten a un programa realizar varias tareas a la vez.

En los sistemas operativos modernos los procesos pueden tener diferentes estados, según el momento de creación, si están en ejecución, si se encuentran a la espera de algún recurso, etc. Pero podemos hacer una simplificación, y un proceso, en un instante dado, puede estar en uno de los tres estados siguientes:

Listo:
: Son los que pueden pasar a estado de ejecución si el planificador del sistema operativo los selecciona, esto es, cuando llegue su turno (según el orden de llegada o prioridad).

En ejecución:
: Son los que se están ejecutando en el procesador en un momento dado.

Bloqueado:
: Los procesos que se encuentran en estado bloqueado están esperando la respuesta de algún otro proceso para poder continuar con su ejecución, por ejemplo una operación de entrada/salida.



El sistema operativo sigue la pista de en qué estado se encuentran los procesos, decide qué procesos pasan a ejecución, cuáles quedan bloqueados, en definitiva, gestiona los cambios de estado de los procesos.

En la planificación del procesador se decide cuánto tiempo de ejecución se le asigna a cada proceso del sistema y en qué momento.
El sistema operativo almacena en una tabla denominad tabla de control de procesos con la información relativa a cada proceso que se esta ejecutando en el procesador. Ésta es:

- Identificación del proceso.
- Identificación del proceso padre.
- Usuario y grupo que lo han lanzado.
- Estado del procesador. El contenido de los registros internos, contador de programa, etc. Es decir, el entorno volátil del proceso.
- Información de control de proceso.
- Información del planificador.
- Segmentos de memoria asignados.
- Recursos asignados.

Una estrategia de planificación debe buscar que los procesos obtengan sus turnos de ejecución de forma apropiada. En general, se buscan cinco objetivos principales:

- Todos los procesos en algún momento obtienen su turno de ejecución o intervalos de tiempo de ejecución hasta su terminación con éxito.
- El sistema debe finalizar el mayor número de procesos por unidad tiempo.
- El usuario no ha de percibir tiempos de espera demasiado largos.
- Evitar el aplazamiento indefinido, los procesos deben terminar en un plazo finito de tiempo. Esto es, el usuario no debe percibir que su programa se ha parado o “colgado”.

La carga de trabajo de un sistema informático a otro puede variar considerablemente, esto depende de las características de los procesos. Nos podemos encontrar:

- Procesos que hacen un uso intensivo de la CPU.
- Procesos que realizan una gran cantidad de operaciones de Entrada/Salida.
- Procesos de menor o mayor duración.

En función de cómo sean la mayoría de los procesos habrá algoritmos de planificación que den un mejor o peor rendimiento al sistema.

Planificación apropiativa y no apropiativa.

La planificación no apropiativa (en inglés, no preemptive) es aquélla en la que, cuando a un proceso le toca su turno de ejecución, ya no puede ser suspendido; es decir, no se le puede arrebatar el uso de la CPU, hasta que el proceso no lo determina no se podrá ejecutar otro proceso. Este esquema tiene sus problemas, puesto que si el proceso contiene ciclos infinitos, el resto de los procesos pueden quedar aplazados indefinidamente. Otro caso puede ser el de los procesos largos que penalizarían a los cortos si entran en primer lugar.

La planificación apropiativa (en inglés preemptive) supone que el sistema operativo puede arrebatar el uso de la CPU a un proceso que esté ejecutándose. En la planificación apropiativa existe un reloj que lanza interrupciones periódicas en las cuales el planificador toma el control y se decide si el mismo proceso seguirá ejecutándose o se le da su turno a otro proceso.

En ambos enfoques de planificación se pueden establecer distintos algoritmos de planificación de ejecución de procesos. Algunos de los algoritmos para decidir el orden de ejecución de los procesos en el sistema son:

- Round Robin (apropiativo)
- El tiempo restante más corto (apropiativo)
- El trabajo más corto primero (no apropiativo)
- El primero en llegar, primero en salir -FIFO- (no apropiativo)

FCFS (*First-Come, First-Served*)

Empezaremos hablando de FCFS o también llamado FIFO (del inglés *First In, First Out*). Este algoritmo es muy sencillo y simple, pero también el que menos rendimiento ofrece, básicamente en este algoritmo el primer proceso que llega se ejecuta y una vez terminado se ejecuta el siguiente.

SJF (*Shortest Job First*).

Este algoritmo siempre prioriza los procesos más cortos primero independientemente de su llegada y en caso de que los procesos sean iguales utilizara el método FIFO anterior, es decir, el orden según entrada. Este sistema tiene el riesgo de poner siempre al final de la cola los procesos más largos por lo que nunca se ejecutarán, esto se conoce como **inanición**.

SRTF (*Short Remaining Time Next*).

Añadiendo la expulsión de procesos al algoritmo SJF obtenemos SRTF, éste será capaz de expulsar un proceso largo en ejecución para ejecutar otros más cortos. El problema que puede surgir es que un proceso largo puede llegar a expulsarse muchas veces y nunca terminar debido a la ejecución de otros mas cortos.

***Round Robin*.**

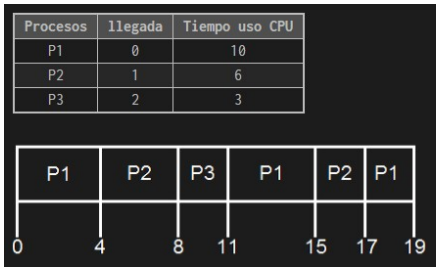
Por último os hablaré de *Round Robin*, este algoritmo de planificación es uno de los más complejos y difíciles de implementar, asigna a cada proceso un tiempo equitativo tratando a todos los procesos por igual y con la misma prioridad.

Este algoritmo es circular, volviendo siempre al primer proceso una vez terminado con el último, para controlar este método a cada proceso se le asigna un intervalo de tiempo llamado *quantum* o cuanto (para definirlo se utiliza esta regla, el 80% de los procesos tienen que durar menos tiempo que el *quantum* definido).

Pueden suceder dos casos con este método (como se aprecia en la imagen inferior):

> El proceso es menor que el quantum: Al terminar antes se planifica un nuevo proceso.
> El proceso es mayor que el quantum: Al terminar el quantum se expulsa el proceso dando paso al siguiente proceso en la lista. Al terminar la iteración se volverá para terminar el primer proceso expulsado

Ejemplo con *quantum* = 4:



Gestión de la memoria:

Actualmente la mayoría de los sistemas operativos son sistemas multitarea, en los que va a haber varios procesos simultáneamente en ejecución. Para que esto sea posible, todos estos procesos deberán estar también simultáneamente en memoria, pues ésta es una condición necesaria para que un proceso pueda ejecutarse. Por tanto, deberá haber mecanismos de gestión para distribuir la memoria principal entre todos estos procesos que quieren ejecutarse.

Intercambio o swapping

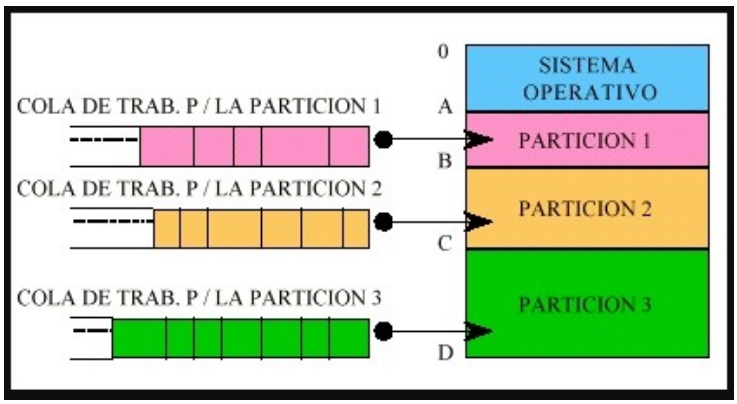
La memoria principal es un recurso limitado, por ello puede ocurrir que haya más procesos esperando a ser cargados en memoria que zonas libres en la misma. En estos casos, el gestor de memoria sacará de la memoria algunos procesos (bloqueados, suspendidos, que estén esperando a que finalice una operación de entrada/salida, etc.) y los llevará a un área de disco (memoria secundaria), conocida como área de intercambio o de swap. A esta operación se la denomina intercambio o swapping. Los procesos permanecerán allí hasta que existan huecos libres en memoria y puedan ser recuperados de disco y reubicados en memoria principal.

Asignación de particiones fijas.

El gestor de memoria necesita reservar un espacio de memoria para el sistema operativo y que el resto de la memoria queda para los procesos de usuarios. Cuando existen varios procesos que requieren ser cargados en memoria el gestor de memoria tiene que organizar el espacio para ubicarlos.
Hay varias alternativas, la primera de ellas es dividir el espacio de memoria en particiones fijas. Estas particiones podrán ser todas del mismo tamaño o tener distintos tamaños. Estas particiones se establecen de forma lógica por el sistema operativo y están predefinidas antes de que lleguen los procesos. El número de particiones se mantiene fijo en el tiempo, así como el tamaño de cada una de las particiones.
La gestión y asignación de particiones a los procesos se puede hacer siguiendo dos tipos de organización:

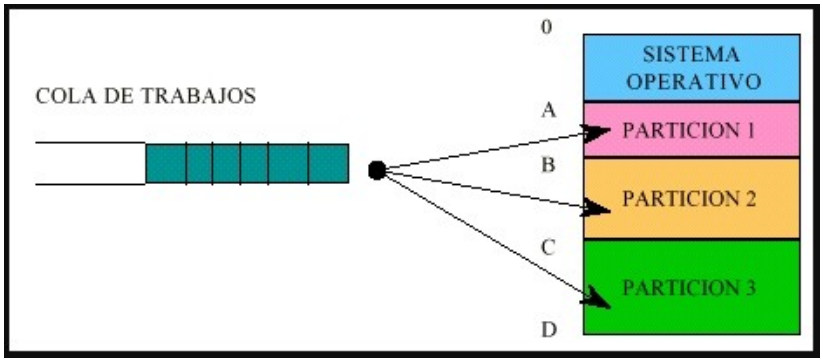
Una cola por partición:

Se tiene una cola por cada partición y se coloca cada trabajo en la cola de la partición más pequeña en que quepa dicho trabajo, a fin de desperdiciar el menor espacio posible.
La planificación de cada cola se hace por separado y, como cada cola tiene su propia partición, no hay competencia entre las colas por la memoria.
La desventaja de este método se hace evidente cuando la cola de una partición grande está vacía y la cola de una partición pequeña está llena.



Una única cola común a todas las particiones.

Se tiene una única cola común para todas las particiones. El sistema operativo decidirá en que partición se ubica cada proceso. En función de la disponibilidad de particiones y las necesidades del proceso en cuestión.

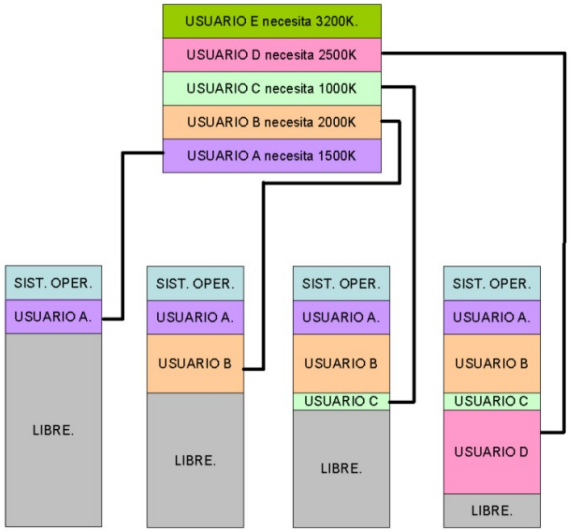


El gestor de memoria establecerá mecanismos para impedir que un proceso pueda acceder a una zona de memoria que está fuera de la memoria correspondiente a la partición en la que se encuentra.
Además de esto, puede surgir el problema de la fragmentación, la cual se produce, cuando en la memoria hay áreas ocupadas intercaladas con áreas libres; es decir, cuando no hay una única área ocupada ni una única área libre.

Asignación de particiones Variables

Con la asignación de particiones fijas se tiene la desventaja de que no se aprovecha, con frecuencia, todo el tamaño de cada partición, ya que el proceso se adapta a los tamaños fijos ya preestablecidos en memoria. En este punto se plantea una segunda alternativa, la asignación de memoria a los procesos mediante particiones variables. La idea es crear las particiones dinámicamente, conforme llegan los procesos y en función de los tamaños de estos. Esta técnica es más realista y aprovecha mejor el espacio de la memoria.

En la asignación de particiones variables, el sistema operativo debe llevar el control de qué partes de la memoria están disponibles y cuales libres.



Fragmentación interna y externa:

La fragmentación interna ocurre cuando la memoria se divide en bloques de tamaño fijo. Siempre que una solicitud de proceso para la memoria, el bloque de tamaño fijo se asigna al proceso. En caso de que la memoria asignada al proceso sea algo mayor que la memoria solicitada, entonces la diferencia entre la memoria asignada y la solicitada es la Fragmentación interna

La fragmentación externa ocurre cuando la memoria se divide en bloques de tamaño variable.
La fragmentación externa se produce cuando hay una cantidad suficiente de espacio en la memoria para satisfacer la solicitud de memoria de un proceso. Pero la solicitud de memoria del proceso no se puede satisfacer ya que la memoria disponible no es contigua.

Memoria Virtual

Hasta este momento los procesos se cargaban enteros en la memoria, pero podría suceder que existan procesos grandes que no quepan en las particiones de la memoria y por tanto, no puedan ser cargados por completo en la memoria.
La memoria virtual da una solución a estos casos, ya que permite dividir los procesos en varias partes y cargar sólo algunas de ellas en memoria. La memoria virtual se basa en el uso de las técnicas de paginación o segmentación.

No todas las partes de un proceso pueden estar cargadas en memoria en un instante determinado. Por ello, cuando un proceso haga referencia a un parte que no se encuentre asignada en memoria provocará un fallo de página o segmento, y el gestor de memoria traerá dicha parte del proceso de disco a memoria.

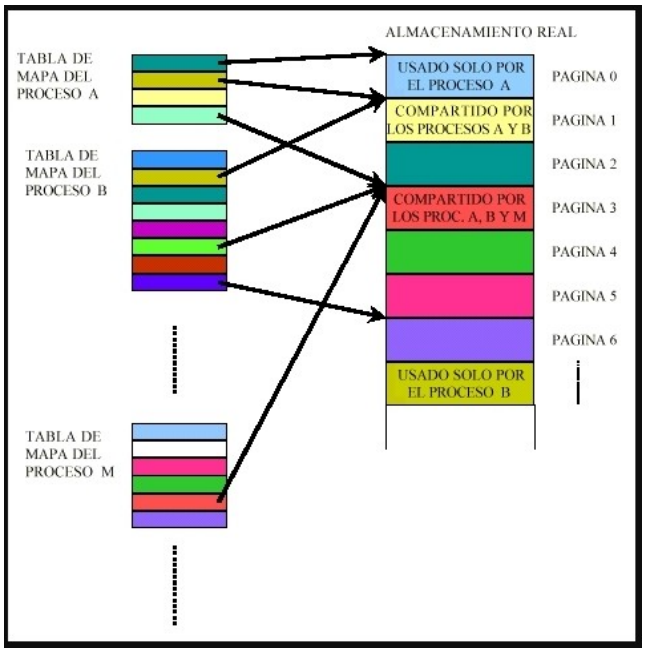
La utilización de las técnicas de paginación o segmentación por parte de la memoria virtual se conocen como:

•Memoria Virtual Paginada: Sigue el funcionamiento de la paginación simple, pero no es necesario cargar todas las páginas de un proceso para que éste pueda ejecutarse. Las páginas que no se encuentren y se necesiten se traerán posteriormente a memoria de manera automática. Reduce la fragmentación

•Memoria Virtual Segmentada: En este caso la operación sería la misma que en la segmentación simple, pero tampoco será necesario cargar todos los segmentos de un proceso. Si se necesitan más segmentos no asignados en memoria se traerán en el momento en que sean referenciados.

Paginación:

La idea es la de dividir la memoria principal en un conjunto de particiones conocidas como “marcos de página” de igual tamaño. Cada proceso se divide a su vez en una serie de partes llamadas “páginas” del mismo tamaño que los marcos. El proceso se carga en memoria situando todas sus páginas en los marcos de página de la memoria, sin embargo, las páginas no tienen porque estar contiguas en memoria. Como ventaja reduce la fragmentación externa de la memoria principal. Sin embargo, puede aparecer cierta fragmentación interna.



En la tabla de páginas de un proceso, se encuentra la ubicación del marco que contiene a cada una de sus páginas. Las direcciones lógicas ahora se forman como un número de página y de un desplazamiento dentro de esa página (conocido comúnmente como *offset*). El número de página es usado como un índice dentro de la tabla de páginas, y una vez obtenida la dirección del marco de memoria, se utiliza el desplazamiento para componer la dirección real o dirección física. Este proceso se realiza en una parte del computador específicamente diseñada para esta tarea, es decir, es un proceso hardware y no software.

Supongamos que se utilizan páginas de 4KB y que un programa quiere acceder a su memoria. Supongamos que el sistema es de 32 bits.
Sabemos que para representar 4KB necesitamos 12 bits.
Por tanto tendremos los primeros 20 bits que representan el número de página y los últimos 12 bits la localización dentro de la página.
si el programa contiene la referencia 2C182 a la que quiere acceder tendremos que convertir la referencia a binario 00000000000000001011000 00110000010

0000000000000001011000 = 88 representa el número de página.
00110000010 representa la localización de memoria a la que queremos acceder.

Cuando se hace una petición de acceso a memoria, la MMU busca en la tabla de páginas del proceso que realizó el pedido por la relación en memoria física. En nuestro ejemplo, la página número 88 del proceso X puede corresponder por ejemplo número 200 en memoria física. La MMU devolverá la dirección del marco en memoria física, con el desplazamiento dentro de esa página, es decir, la dirección real de memoria

Cuando la paginación se utiliza junto con memoria virtual, el sistema operativo mantiene además el conocimiento sobre qué páginas están en memoria principal y cuáles no, usando la tabla de paginación. Si una página buscada está marcada como no disponible (tal vez porque no está presente en la memoria principal, pero sí en el área de intercambio), cuando la CPU intenta referenciar una dirección de memoria en esa página, la MMU responde levantando una excepción (comúnmente llamada fallo de página). Si la página se encuentra en e espacio de intercambio, el sistema operativo invocará una operación llamada *intercambio de página*, para traer a memoria principal la página requerida.

Segmentación:

Cada proceso se divide en una serie de segmentos. La peculiaridad de estos segmentos es que su tamaño no tiene que ser el mismo y puede variar hasta un límite máximo. Un proceso se carga situando todos sus segmentos en particiones dinámicas que no tienen que estar contiguas en memoria. Este sistema reduce la fragmentación interna de la memoria principal.

DMA

El acceso directo a memoria (DMA, del inglés direct memory access) permite a cierto tipo de componentes de una computadora acceder a la memoria del sistema para leer o escribir independientemente de la unidad central de procesamiento (CPU) principal. 1Muchos sistemas hardware utilizan DMA, incluyendo controladores de unidades de disco, tarjetas gráficas y tarjetas de sonido.

Lecturas de disco cuando no se usa DMA.

Primero el controlador lee el bloque (uno o más sectores) de la unidad en serie, bit por bit, hasta que todo el bloque está en el buffer interno del controlador. A continuación, el controlador calcula la suma de verificación para comprobar que no ocurrieron errores de lectura, y luego causa una interrupción.
La CPU entonces, puede leer el bloque del disco del buffer del controlador byte por byte o palabra por palabra, leyéndose en cada iteración un byte o una palabra de un registro del controlador y almacenándose en la memoria.
Naturalmente, un ciclo del CPU programado para leer los bytes del controlador uno por uno desperdicia tiempo de CPU.

Lecturas de disco cuando se utiliza DMA:

La CPU proporciona al controlador la siguiente información, la dirección en disco del bloque: la dirección de memoria donde debe colocarse el bloque y el número de bytes que deben transferirse

Una vez que el controlador ha leído todo el bloque del dispositivo, lo ha colocado en su buffer y ha calculado la suma de verificación, copia el primer byte o palabra en la memoria principal en la dirección especificada por la dirección de memoria de DMA. Luego, el controlador incrementa la dirección de DMA y decrementa la cuenta de DMA en el número de bytes que se acaban de transferir. Este proceso se repite hasta que la cuenta de DMA es cero, y en ese momento el controlador causa una interrupción. Ahora la CPU tiene en memoria los datos del dispositivo y no ha tenido que invertir tiempo en copiarlos