Tema 2: Introducción a los Sistemas Operativos

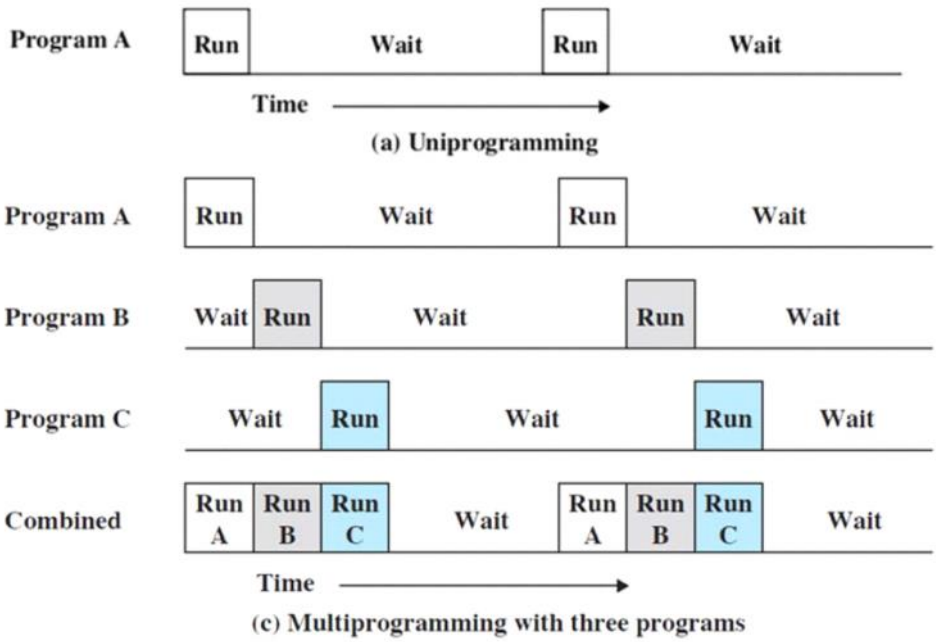
Miguel Ángel Fernández Gutiérrez

2.1 - Componentes de un SO multiprogramado

Concepto de multiprogramación

- **Procesamiento en serie:** el programador interactúa directamente con la máquina, no existe SO. *Problemas:* baja utilización del tiempo de CPU, tiempo de planificación alto.
- **Sistemas por lotes (sistemas *batch*):** agrupa trabajos similares, reduciendo el tiempo de planificación. Trabajo = programa + datos + órdenes de control para el sistema. Falta de interacción entre el usuario y el computador mientras se ejecuta su trabajo. *Principal problema:* CPU ociosa durante las E/S. *Solución:* **spooling**, superpone la E/S de un trabajo al cómputo de otros.

Mientras un programa espera a un dispositivo de E/S, el computador gasta aprox el 96% de su tiempo. Esta ineficiencia puede evitarse. Hay suficiente memoria para contener al SO y a un programa de usuario, si hacemos que haya memoria suficiente para más programas de usuario podemos hacer que mientras que uno necesita esperar por la E/S se asigne el procesador al otro, que probablemente no esté esperando por una operación de E/S. Así con todos los programas que queramos. Este enfoque se llama **multiprogramación** o **multitarea**.



Con la multiprogramación el rendimiento se reduce a más de la mitad, ya que por ejemplo tres procesos que en monoprogramación tardarían 30 min en completarse, usando multiprogramación solo tardarían 15 ya que hay poco conflicto entre los trabajos, todos pueden ejecutar casi en el mínimo tiempo mientras coexisten con los otros en el computador.

Del mismo modo que un sistema en lotes simple, un sistema en lotes multiprogramado también debe basarse en ciertas características hardware del computador entre las que destacan que soporta las interrupciones de E/S y DMA.

En un sistema de tiempo compartido, múltiples usuarios acceden simultáneamente al sistema a través de terminales,
siendo el SO el encargado de entrelazar la ejecución de cada programa de usuario en pequeños intervalos de tiempo o cuantos de computación.

Definiciones

- **SO multiprogramado:** SO que permite que se ejecute más de un proceso simultáneamente y cuyos datos e instrucciones se encuentran en memoria principal.
- **SO monousuario:** proporciona servicios a un único usuario.
- **SO multiusuario:** proporciona servicios a varios usuarios simultáneamente.
- **SO monoprocesador:** SO que gestiona un sistema de computación de un único procesador.
- **SO multiprocesador:** SO que gestiona un sistema de computación de varios procesadores.
- **SO de tiempo compartido:** SO multiprogramado donde se realiza un reparto de tiempo del procesador en pequeños trozos de tal forma que todos los procesos pueden avanzar adecuadamente. Especialmente diseñado para sistemas interactivos.

Cuestiones interesantes

- ¿Un SO multiprogramado es un SO de tiempo compartido? ¿Y al contrario?
- ¿Un SO de tiempo compartido tiene que ser multiusuario? ¿Y monousuario?
- ¿Un SO monoprocesador tiene que ser monousuario? ¿Y multiusuario?
- ¿Un SO multiprocesador tiene que ser monousuario? ¿Y multiusuario?

Concepto de proceso

El concepto de proceso es fundamental en la estructura de los sistemas operativos, este término es un poco más general que el de trabajo. Se han dado muchas definiciones, entre las que se incluyen:

- Un programa en ejecución
- Una **instancia de un programa** ejecutándose en un computador
- La entidad que se puede asignar o ejecutar en un procesador
- Una **unidad de actividad** caracterizada por un **solo flujo (hilo secuencial) de ejecución**, un **estado actual** y un **conjunto de recursos** del sistema asociados.

Se puede considerar que un proceso está formado por los siguientes tres componentes:

- Un programa ejecutable
- Los datos asociados que necesita el programa
- El contexto de ejecución del programa.
- Entidad que consiste en un número de elementos. Los dos elementos esenciales serían el **código de programa** (que puede compartirse con otros procesos que estén ejecutando el mismo programa) y un **conjunto de datos** asociados a dicho código.

Este último es esencial. El **contexto de ejecución** o **estado del proceso** es el conjunto de datos interno por el cual el sistema operativo es capaz de supervisar y controlar el proceso. Esta información interna está separada del proceso, porque el sistema operativo tiene la información a la que el proceso no puede acceder. El contexto incluye toda la información que necesita para gestionar el proceso y que el procesador necesita para ejecutar el proceso adecuadamente. El contexto incluye el contenido de diversos registros del procesador, tales como el contador de programa y los registros de datos. También incluye información de uso del sistema operativo, como la prioridad del proceso y si un proceso está esperando por la finalización de un evento de E/S particular.

Bloque de Control de Proceso (PCB, Process Control Block)

Supongamos que el procesador empieza a ejecutar un proceso. En cualquier instante puntual del tiempo, mientras el proceso está en ejecución, este proceso se puede caracterizar por una serie de elementos:

- **Identificador:** un identificador único asociado a este proceso, para distinguirlo del resto de procesos.

- **Estado:** si el proceso está actualmente corriendo, está en el estado de en ejecución.
- **Prioridad:** nivel de prioridad relativo al resto de procesos.
- **Contador de programa:** la dirección de la siguiente instrucción del programa que se ejecutará.
- **Punteros a memoria:** incluye los punteros al código del programa y los datos asociados a dicho proceso, además de cualquier bloque de memoria compartido con otros procesos.
- **Datos de contexto:** estos son datos que están presentes en los registros del procesador cuando el proceso está ejecutándose.
- **Información de estado de E/S:** incluye las peticiones de E/S pendientes, dispositivos de E/S asignados a dicho proceso, una lista de los ficheros en uso por el mismo, etc.
- **Información de auditoría:** Puede incluir la cantidad de tiempo de procesador y de tiempo de reloj utilizados, así como los límites de tiempo, registros contables, etc.

Toda esta información se suele almacenar en el bloque de control de proceso (**Process Control Block, PCB**), que el SO crea y gestiona. El punto más significativo en relación al PCB es que contiene suficiente información de forma que es posible interrumpir el proceso cuando está ejecutándose y posteriormente restaurar su estado de ejecución como si no hubiera habido interrupción alguna. El PCB es la herramienta clave que permite al sistema operativo dar soporte a múltiples procesos y proporcionar multiprogramación.

Concepto de traza de ejecución

Se puede caracterizar el comportamiento de un determinado proceso listando la secuencia de instrucciones que se ejecutan para dicho proceso, a esta lista se la denomina traza del proceso. Se puede caracterizar el comportamiento de un procesador mostrando cómo las trazas de varios procesos de entrelazan. Se entrelazan como consecuencia de que el SO solo deja que un proceso continúe durante un número determinado de ciclos de instrucción, tras los cuales se interrumpe, lo cual previene que un solo proceso monopolice el uso del tiempo del procesador. De manera adicional, existe un pequeño programa llamado **activador (*dispatcher*)** que intercambia el procesador de un proceso a otro.

- Una **traza de ejecución** es un listado de la secuencia de las instrucciones de un programa que realiza el procesador para un proceso.
- Desde el punto de vista del procesador se entremezclan las trazas de ejecución de los procesos y las trazas del código del SO.

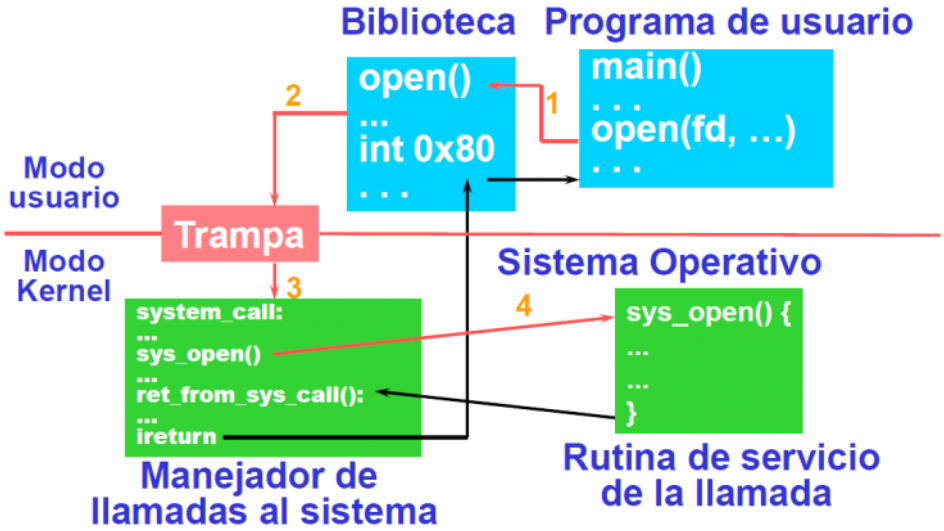
¿En qué situaciones se pueden entremezclar las trazas de los procesos y las trazas del código del SO?

Llamadas al sistema

Las llamadas al sistema son la forma en la que se comunican los programas de usuario con el SO en tiempo de ejecución, son solicitudes al SO de petición de servicio.

Algunos ejemplos de llamada al sistema suelen ser **solicitudes de E/S** en las que un programa se bloquea esperando algún dato por parte de un módulo de E/S, **gestión de procesos** y **de memoria**. Se implementan a través de una **trampa (*trap*)** o **"interrupción software"**.

Pasos realizados durante una llamada al sistema



Un modelo de proceso de dos estados

La responsabilidad principal del sistema operático es controlar la ejecución de procesos; esto incluye determinar el patrón de entrelazado para la ejecución y asignar recursos a los procesos. Cuando el sistema operativo crea un nuevo proceso, crea el PCB asociado a dicho proceso e inserta dicho proceso en el sistema de estado No Ejecutando. De cuando en cuando, el proceso en ejecución se interrumpirá y el activador seleccionará otro proceso a ejecutar.

Existe una sola cola cuyas entradas son punteros al PCB de un proceso en particular, la cola debe consistir en una lista enlazada de bloques de datos en la cual cada bloque representa un proceso. Un proceso que se interrumpe se transfiere a la cola de procesos en espera y si ha finalizado o ha sido abortado se descarta. En cualquier caso, el activador siempre selecciona un proceso de la cola para ejecutar.

Creación y terminación de procesos

- **Creación de un proceso:** el sistema operativo construye las estructuras de datos que se usan para manejar el proceso y reserva el espacio de direcciones en memoria principal para el proceso. En un entorno por lotes, un proceso se crea como respuesta a una solicitud de trabajo. En un entorno interactivo, un proceso se crea cuando un nuevo usuario entra en el sistema, en ambos casos el SO es el responsable de la creación de nuevos procesos. Sin embargo puede ser muy útil permitir a un proceso la creación de otro, ya que puede ser útil en la estructura de la aplicación. Cuando un sistema operativo crea un proceso a petición explícita de otro proceso, se denomina **creación del proceso**. Cuando un proceso lanza a otro, al primero se le denomina **proceso padre** y al proceso creado se le denomina **proceso hijo**. La relación entre ambos procesos necesita comunicación y cooperación entre ellos.
  De forma más específica, las operaciones que debe hacer el SO para crear un proceso son:
  - Asignar un espacio de memoria para albergar la imagen de memoria, en general, este espacio será virtual y estará compuesto de varios segmentos.
  - Seleccionar un PCB libre para la tabla de procesos
  - Rellenar el PCB con la información asignada de identificación del proceso.
  - Cargar en el segmento de texto el código más las rutinas de sistema y en el segmento de datos los datos iniciales contenidos en el archivo objeto.
  - Crear en el segmento de pila la pila inicial del proceso, la pila incluye inicialmente el entorno del proceso y los parámetros que se pasan en la invocación del programa correspondiente.
- **Terminación de procesos:** Un trabajo por lotes debe incluir una instrucción HALT o una llamada a un servicio de sistema operativo que especifica su terminación. En el caso anterior, la instrucción HALT generará una interrupción para indicar al sistema operativo que dicho proceso ha finalizado. Adicionalmente, un número de error o una condición de fallo puede

llevar a la finalización de un proceso. Las condiciones más habituales de finalización de procesos son por estos motivos:

- **Finalización normal:** el proceso ejecuta una llamada al sistema operativo para indicar que ha completado su ejecución.
- **Límite de tiempo excedido:** el proceso ha ejecutado más tiempo del especificado en un límite máximo.
- **Memoria no disponible:** el proceso requiere más memoria de la que el sistema puede proporcionar.
- **Violaciones de frontera:** el proceso trata de acceder a una posición de memoria a la cual no tiene acceso permitido.
- **Error de protección:** el proceso trata de usar un recurso al que no tiene permitido acceder o trata de utilizarlo de una forma no apropiada.
- **Error aritmético:** el proceso trata de realizar una operación de cálculo no permitida.
- **Límite de tiempo:** el proceso ha esperado más tiempo del especificado en un valor máximo para que se cumpla un determinado evento.
- **Fallo de E/S:** se ha producido un error durante la operación de entrada o salida.
- **Instrucción no válida:** el proceso intenta ejecutar una instrucción inexistente.
- **Instrucción privilegiada:** el proceso intenta utilizar una instrucción reservada al SO.
- **Uso inapropiado de datos:** una porción de datos es de tipo erróneo o no se encuentra inicializada.
- **Intervención del operador por el SO:** por alguna razón, el operador o el SO ha finalizado el proceso.
- **Terminación del proceso padre:** cuando un proceso padre termina, el SO puede automáticamente finalizar todos los procesos hijos descendientes de dicho padre.
- **Solicitud del proceso padre:** un proceso padre tiene autoridad para finalizar sus procesos descendientes.

Modelo de los cincos estados

La cola es una **lista de tipo FIFO (first-in-first-out)** y el procesador opera siguiendo una **estrategia cíclica (*round-robin* o turno rotatorio)** sobre todos los procesos disponibles. Sin embargo, esta implementación es inadecuada: algunos procesos que están en el estado de *No Ejecutando* están listos para ejecutar mientras que otros están bloqueados, esperando a que se complete una operación de E/S, por tanto, utilizando una única cola el activador no puede seleccionar únicamente los procesos que lleven más tiempo en la cola, en su lugar, debería recorrer la lista buscando los procesos que no estén bloqueados y que lleven en la cola más tiempo. Una forma natural de manejar esta situación es dividir el estado de *No Ejecutando* en dos estados: *Preparado* y *Bloqueado*. Se han añadido dos estados adicionales también:

- **Ejecutando:** el proceso está actualmente en ejecución.
- **Preparado (o listo):** un proceso que se prepara para ejecutar cuando tenga una oportunidad.
- **Bloqueado:** un proceso que no se puede ejecutar hasta que no se cumpla un evento determinado o se complete una operación de E/S.
- **Nuevo:** un proceso que se acaba de crear y que aún no ha sido admitido en el grupo de procesos ejecutables por el SO. Suele ser un proceso que aún no se ha cargado en memoria principal aunque su PCB si ha sido creado.
- **Finalizado (o saliente):** un proceso que ha sido liberado del grupo de procesos ejecutables por el SO, debido a que ha sido detenido o a que ha sido abortado por alguna razón.

Los estados nuevo y saliente son útiles para construir la gestión de procesos. Un SO puede limitar el número de procesos que puede haber en el sistema por razones de rendimiento o limitaciones de memoria principal. Cuando un proceso se encuentra en estado *Nuevo*, el programa permanece en almacenamiento secundario, normalmente en disco.

La imagen indica qué tipos de eventos llevan a cada transición de estado para cada proceso; las posibles transiciones son:

- **Null → Nuevo:** se crea un nuevo proceso para ejecutar un programa.
- **Nuevo → Preparado:** El SO mueve a un proceso del estado de nuevo al estado listo cuando éste se encuentre preparado para ejecutar un nuevo proceso. La mayoría de sistemas fijan un límite basado en el número de procesos existentes o la cantidad de memoria virtual que se podrá utilizar por parte de los procesos existentes.
- **Preparado → Ejecutando:** cuando llega el momento de seleccionar un nuevo proceso para ejecutar el SO selecciona uno de los procesos que se encuentre en el estado Listo. Esta tarea la lleva a cabo el planificador.
- **Ejecutando → Finalizado:** el proceso actual en ejecución se finaliza por parte del SO tanto si el proceso indica que ha completado su ejecución como si éste aborta.
- **Ejecutando → Preparado:** la razón más habitual para esta transición es que el proceso en ejecución haya alcanzado el máximo tiempo posible de ejecución de forma ininterrumpida; prácticamente todos los sistemas operativos multiprogramados imponen este tipo de restricción de tiempo. Existen otras posibilidades como el caso en el cual el SO asigna diferentes niveles de prioridad a diferentes procesos.
- **Ejecutando → Bloqueado:** un proceso se pone en estado bloqueado si solicita algo por lo que debe esperar. Una solicitud al sistema operativo se realiza habitualmente por medio de una llamada al sistema; una llamada del proceso en ejecución a un procedimiento es parte del código del SO.
- **Bloqueado → Preparado:** un proceso en estado bloqueado se mueve al estado listo cuando sucede el evento por el cual estaba esperando.
- **Preparado → Finalizado:** en algunos sistemas, un padre puede terminar la ejecución de un proceso hijo en cualquier momento. También, si el padre termina, todos los hijos asociados con dicho padre deben finalizarse.
- **Bloqueado → Finalizado:** se aplican los comentarios del caso anterior.

En vez de tener una cola por cada estado, se tiene una cola por cada evento ya que si hubiera una cola por cada estado, cada vez que se produjera un evento el planificador tendría que recorrer toda la lista de bloqueados hasta encontrar al proceso que estaba esperando ese evento, lo cual no es eficiente.

También se tienen colas por nivel de prioridad, el SO puede determinar cuál es el proceso listo de mayor prioridad simplemente seleccionando éstas en orden.

2.2 - Descripción y control de procesos

Descripción de Procesos: PCB

El PCB contiene la información básica del proceso, entre la que cabe destacar la siguiente:

- **Información de identificación:** esta información identifica al usuario y al proceso, como por ejemplo el identificador del proceso, el identificador del proceso padre, información del usuario…
- **Estado del procesador:** contiene los valores iniciales del estado del procesador o su valor en el instante en que fue interrumpido el proceso.
- **Información de control del proceso:** en esta sección se incluye diversa información que permite gestionar al proceso donde destacamos:
    - Información de planificación y estado:
        - Estado del proceso.
        - Evento por el que espera el proceso cuando está bloqueado.
        - Prioridad del proceso.
        - Información de planificación.
    - Descripción de los segmentos de memoria asignados al proceso.
    - Recursos asignados, tales como:
        - Archivos abiertos (tabla de descriptores o manejadores de archivo).
        - Puertos de comunicación asignados.
    - Punteros para estructurar los procesos en colas o anillos.
    - Comunicación entre procesos. El PCB puede contener espacio para almacenar las señales y para algún mensaje enviado al proceso.

Control de procesos: modos de ejecución del procesador

- El modo menos privilegiado a menudo se denomina **modo usuario**, porque los programas de usuario típicamente se ejecutan en este modo, solo tiene acceso a un subconjunto de los registros del procesador, un subconjunto del repertorio de instrucciones máquina y un área de memoria.
- El modo más privilegiado se denomina **modo núcleo (*kernel* o supervisor o sistema)** y se refiere al núcleo del sistema operativo, que es la parte del sistema operativo que engloba las funciones más importantes del sistema.

El motivo por el que se usan dos modos es claro, se necesita proteger al SO y a las tablas clave del sistema. Existe un bit en la palabra de estado de programa (PSW) que indica el modo de ejecución, este bit cambia como respuesta a determinados eventos (para cambiar *a modo kernel*) que suelen ser:

- **Cuando el usuario realiza una llamada a un servicio del SO**, donde se cambia el modo ejecución a modo núcleo hasta la finalización del servicio cuando el modo vuelve a fijarse en usuario.
- **Cuando llega una interrupción**, el procesador limpia la mayor parte de los bits en ***psr*** (registro de estado), incluyendo el campo ***cpl*** (current privilege level) y al final de la rutina de manejo de interrupción, la última instrucción que se ejecuta es ***irt*** (interrupt return) que hace que el procesador restaure el *psr* del programa interrumpido y restaura también el nivel de privilegios de dicho programa.
- **Cuando la aplicación solicita una llamada al sistema** indicando un identificador de llamada de sistema y los argumentos de dicha llamada en unas áreas definidas y posteriormente ejecutando una instrucción especial que puede invocar en modo usuario y que tiene como objeto transferir el control al núcleo.

Seguidamente se ejecuta la rutina del SO correspondiente al evento producido. Finalmente, cuando termina la rutina, el hardware restaura automáticamente el modo de ejecución a *modo usuario*.

Control de procesos: cambio de contexto (cambio de proceso)

Un proceso en estado *Ejecutando* cambia a otro estado y un proceso en estado *Preparado* pasa a estado *Ejecutando*. Un cambio de proceso puede ocurrir en cualquier instrucción en el que el sistema operativo obtiene el control sobre el proceso actualmente en ejecución. Los posibles

momentos en los que el sistema operativo puede tomar dicho control son:

Mecanismo	Causa	Uso
Interrupción	Externa a la ejecución del proceso actualmente en ejecución	Reacción ante un evento externo
Trap	Asociada a la ejecución de la instrucción	Manejo de una condición de error o de excepción
Llamada al sistema	Solicitud explícita	Llamada a una función del sistema

Consideremos las interrupciones del sistema. Podemos distinguir dos tipos: las **interrupciones** y las ***traps***. Las primeras se producen por causa de algún tipo de evento que es externo e independiente al proceso actualmente en ejecución, las otras están asociadas a una condición de error o excepción generada dentro del proceso que está ejecutando.

Dentro de la interrupción ordinaria el control se transfiere inicialmente al manejador de interrupción que realiza determinadas tareas internas y que posteriormente salta a una rutina del SO, encargada de cada uno de los tipos de interrupciones en particular como:

- **Interrupción de reloj:** el SO determina si el proceso ha excedido o no la unidad máxima de tiempo de ejecución, rodaja de tiempo (time slice)
- **Interrupción de E/S:** el SO determina qué acción de E/S ha ocurrido y si la acción constituye un evento que estaban esperando uno o más procesos, el SO los mueve al estado de listos.
- **Fallo de memoria:** el procesador se encuentra con una referencia a una dirección de memoria virtual a una palabra que no se encuentra en memoria principal, por lo que el SO debe traer el bloque que contiene la referencia desde memoria secundaria a memoria principal, tras solicitar esto el proceso pasa a bloqueado y cuando el bloque se ha traído a memoria pasa al estado *Preparado*.

Con un ***trap***, el SO conoce si una condición de error o de excepción es irreversible, si es así, el proceso en ejecución se pone en el estado Saliente y se hace un cambio de proceso, de no ser así, el SO actuará dependiendo de la naturaleza del error y su propio diseño.

Por último, el SO se puede activar por medio de una **llamada al sistema** procedente del programa en ejecución.

Cambio de contexto

La diferencia entre un cambio de modo y un cambio de proceso es que un cambio de modo puede ocurrir sin que se cambie el estado del proceso actualmente en estado *Ejecutando*, en este caso, la salvaguarda del estado y si posterior restauración comportan sólo una ligera sobrecarga. Sin embargo, si el proceso actualmente en estado *Ejecutando* cambia a cualquier otro estado, el SO debe realizar cambios sustanciales en su entorno. Los pasos para un cambio de proceso completo son:

1. Salvar el estado del procesador, incluyendo el PC y otros registros.
2. Actualizar el PCB que está actualmente en el estado *Ejecutando*, esto incluye cambiar el estado de proceso a uno de los otros estados y actualizar otros campos importantes incluyendo la razón por la cual el proceso ha dejado el estado *Ejecutando*, etc.
3. Mover el PCB a la cola apropiada
4. Seleccionar un nuevo proceso a ejecutar.
5. Actualizar el PCB del proceso elegido
6. Actualizar las estructuras de datos de gestión de memoria, esto se puede necesitar, dependiendo de cómo se haga la traducción de direcciones.
7. Restaurar el estado del procesador al que tenía en el momento en el que el proceso seleccionado salió del estado *Ejecutando* por última vez, leyendo los valores anteriores del PC

y registros.

En las transparencias figura así:

1. Salvar los registros del procesador en el PCB del proceso que actualmente está en estado *Ejecutándose*.
2. Actualizar el campo estado del proceso al nuevo estado al que pasa e insertar el PCB en la cola correspondiente.
3. Seleccionar un nuevo proceso del conjunto de los que se encuentran en estado *Preparado* (***Scheduler*** o **Planificador de CPU**).
4. Actualizar el estado del proceso seleccionado a *Ejecutándose* y sacarlo de la cola de preparados.
5. Cargar los registros del procesador con la información de los registros almacenada en el PCB del proceso seleccionado.

Por tanto, el cambio de proceso, que implica un cambio en el estado, requiere un mayor esfuerzo que un cambio de modo.

¿Si se produce una interrupción, una excepción o una llamada al sistema, se produce necesariamente un cambio de modo? ¿y un cambio de contexto?

Control de procesos: Cambio de modo

Si hay una interrupción pendiente, indicada por la presencia de una señal de interrupción, el procesador actúa así:

1. Coloca el contador de programa en la dirección de comienzo de la rutina del programa manejador de la interrupción
2. Cambia de modo usuario a modo núcleo de forma que el código de tratamiento de la interrupción pueda incluir instrucciones privilegiadas.

Después, el procesador pasa a la fase de búsqueda de instrucción y busca la primera instrucción del programa de manejo de interrupción, que dará servicio a la misma. En este punto, el contexto del proceso que se ha interrumpido se guarda en su PCB. Este contexto está constituido por toda la información que se puede ver alterada por la ejecución de la rutina de interrupción y que se necesitará para la continuación del proceso que ha sido interrumpido. Esta información es la información de estado del procesador que incluye el PC, otros registros del procesador y la información de la pila.

Un cambio de modo puede ejecutarse solamente como resultado de una interrupción, una excepción o una llamada al sistema.

Los pasos a seguir en un cambio de modo son:

1. El hardware automáticamente salva como mínimo el PC y PSW y cambia a modo núcleo.
2. Determinar automáticamente la rutina del SO que debe ejecutarse y cargar el PC con su dirección de comienzo
3. Ejecutar la rutina, posiblemente la rutina comience salvando el resto de registros del procesador y termine restaurando en el procesador la información de registros previamente salvada.
4. Volver de la rutina del SO, el hardware automáticamente restaura en el procesador la información del PC y PSW previamente salvada.

2.3 - Hebras (hilos)

Concepto de hebra

Hasta el momento se ha presentado el concepto de proceso como poseedor de dos características:

- **Propiedad de recursos:** un proceso incluye un espacio de direcciones virtuales para el manejo de la imagen del proceso (colección de programa, datos, pila y atributos definidos en el PCB del proceso), de vez en cuando a un proceso se le puede asignar el control o propiedad de recursos como la memoria principal, canales E/S… El SO realiza la función de protección para evitar interferencias no deseadas entre procesos en relación con los recursos.
- **Planificación/ejecución:** la ejecución de un proceso sigue una ruta de ejecución (traza) a través de uno o más programas, esta ejecución debe estar intercalada con ese u otros procesos.

Así, un proceso tiene un estado de ejecución y una prioridad de activación y ésta es la entidad que se planifica y activa por el SO.

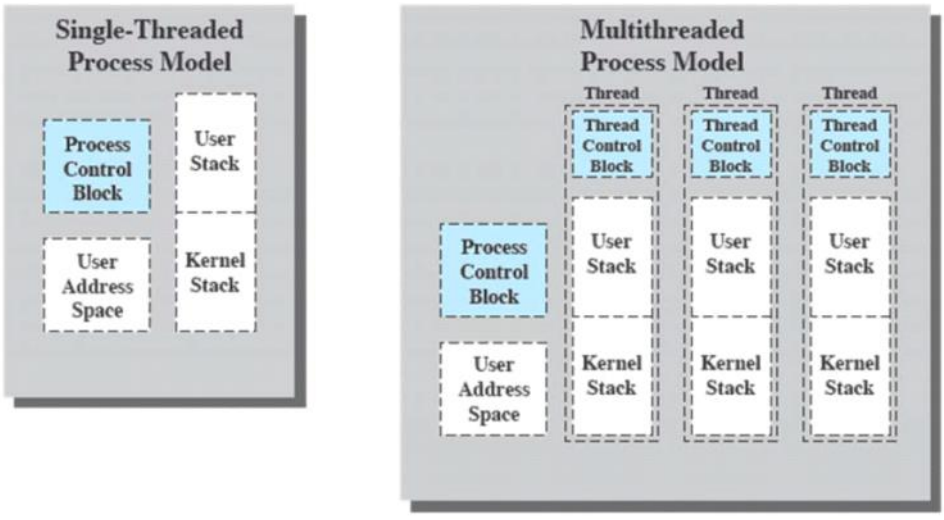
En la mayoría de sistemas operativos estas características son la esencia de un proceso, sin embargo, son independientes y podrían ser tratadas como tales por el SO y así se hace en diversos sistemas operativos. Para distinguir estas dos características, la unidad que se activa se suele denominar **hilo (*thread*)** o **proceso ligero**, mientras que la unidad de propiedad de recursos de suele denominar **proceso** o **tarea**.

*La **tarea** se encarga de soportar todos los recursos necesarios (incluida la memoria), y cada una de las **hebras** permite la ejecución del programa de forma "independiente" del resto de hebras.*

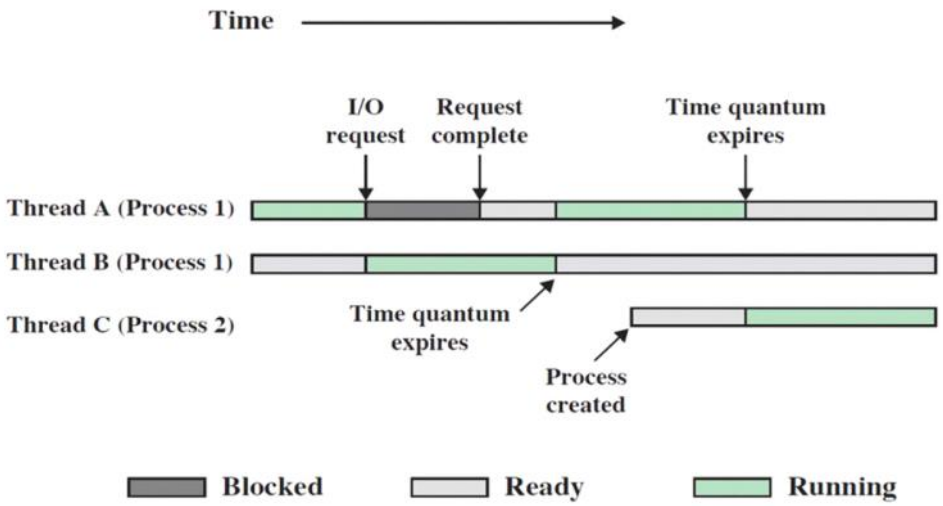
Multihilo se refiere a la capacidad de un SO de dar soporte a múltiples hilos de ejecución en un solo proceso. El enfoque tradicional de un solo hijo de ejecución por proceso se conoce como estrategia monohilo.

En un entorno multihilo, se asocian con procesos los siguientes:
- Un espacio de direcciones virtuales que soporta la imagen del proceso.
- Acceso protegido a procesadores, otros procesos, archivos y recursos de E/S.

Dentro de un proceso puede haber uno o más hilos, cada uno con:
- Un estado de ejecución por hilo
- Un contexto de hilo que se almacena cuando no está en ejecución: una forma de ver a un hilo es como un PC independiente dentro de un proceso
- Una pila de ejecución
- Por cada hilo, espacio de almacenamiento para variables locales
- Acceso a la memoria y recursos de su proceso, compartido con todos los hilos de su mismo proceso.



En la imagen se puede apreciar la diferencia entre hilos y procesos desde un punto de vista de gestión de procesos. En un modelo de proceso monohilo la representación de proceso incluye su PCB y el espacio de direcciones de usuario, además de las pilas de usuario y núcleo para gestionar el comportamiento de las llamadas/retornos en la ejecución de procesos. Mientras el proceso se ejecuta los registros del procesador se controlan por ese proceso y cuando no se ejecuta se almacena el contenido de estos registros. En un entorno multihilo sigue habiendo un único PCB y un espacio de direcciones de usuario asociado al proceso, pero ahora hay varias pilas separadas para cada hilo, así como un PCB para cada hilo que contiene los valores de los registros, la prioridad y otra información relativa al estado del hilo. Así, todos los hilos de un proceso comparten estado y recursos de ese proceso, residen en el mismo espacio de direcciones y tienen acceso a los mismos datos.



Los mayores beneficios de los hilos provienen de las consecuencias del rendimiento:

1. Lleva mucho **menos tiempo crear un nuevo hilo** en un proceso existente que crear un proceso totalmente nuevo.
2. Lleva **menos tiempo finalizar un hilo** que un proceso.
3. Lleva **menos tiempo cambiar entre dos hilos** dentro del mismo proceso
4. Los hilos **mejoran la eficiencia de la comunicación** entre diferentes programas que están ejecutando. En la mayor parte de los SSOO, la comunicación entre procesos independientes requiere la intervención del núcleo para proporcionar protección y los mecanismos necesarios de comunicación. Sin embargo, hilos dentro un mismo proceso al compartir memoria y archivos se pueden comunicar sin necesidad de invocar al núcleo.
5. Permiten aprovechar las técnicas de **programación concurrente** y el **multiprocesamiento simétrico**.

Estados de los hilos

Al igual que con los procesos, los principales estados de los hilos son: *Ejecutando*, *Preparado* y *Bloqueado*. *(Apuntes: cinco estados análogos al modelo de estados para procesos: Ejecutando, Preparado (listo para ejecutarse), Bloqueado, Nuevo, Finalizado).*

Generalmente no tiene sentido aplicar estado de suspensión a un hilo ya que dichos estados son conceptos de nivel de proceso, en particular, si se expulsa un proceso, todos sus hilos se deben expulsar porque comparten el espacio de direcciones del proceso.

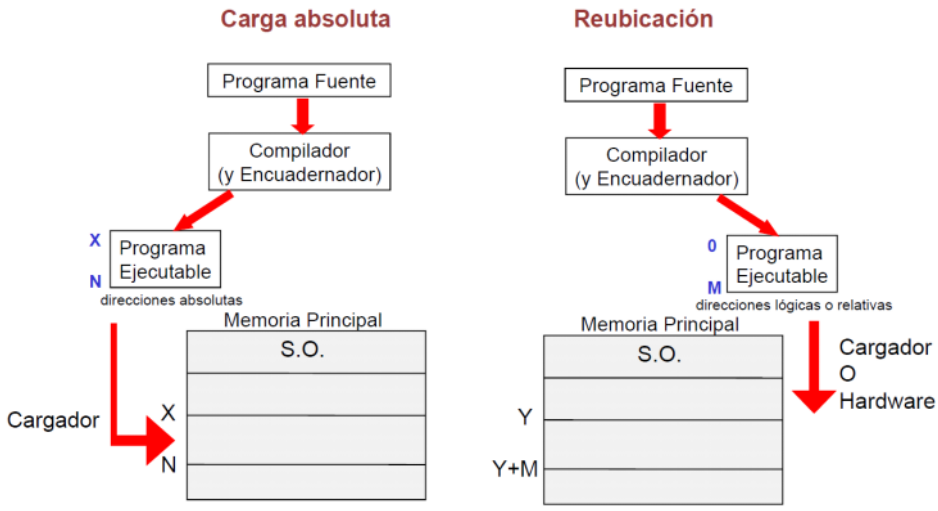
Hay cuatro operaciones básicas relacionadas con los hilos que están asociadas a un cambio de estado del hilo:

- **Creación:** cuando se crea un nuevo proceso, también se crea un hilo de dicho proceso. Este hilo puede crear otro hilo dentro del mismo proceso, proporcionando un puntero a las instrucciones y los argumentos para el nuevo hilo y a este se le proporciona su propio registro de contexto y espacio de pila y se coloca en la cola de listos.
- **Bloqueo:** cuando un hilo necesita esperar por un evento se bloquea, almacenando los registros de usuario, PC y punteros de pila. Cuando un hilo se bloquea, este no bloquea al proceso completo.
- **Desbloqueo:** cuando sucede el evento por el que el hilo está bloqueado, el hilo pasa a la cola de *Preparados*.
- **Finalización:** cuando se completa un hilo, se liberan su registro de contexto y pilas.

2.4 - Gestión básica de memoria

Carga absoluta y reubicación

No es posible saber anticipadamente qué programas residirán en memoria principal en tiempo de ejecución ni dónde se van a colocar. Además sería bueno poder intercambiar procesos para maximizar la utilización del procesador, una vez cargado el programa al disco, es muy limitante llevarlo a la misma región de memoria principal donde se hallaba antes, por el contrario, es necesario reubicar el proceso a un área de memoria diferente. Al no saber dónde se van a colocar los programas debemos permitir que se puedan mover en memoria principal, debido al intercambio o swap. El SO necesita conocer la ubicación de la información de control del proceso y de la pila de ejecución, así como el punto de entrada que utilizará el proceso para iniciar la ejecución. Debido a que el SO es el encargado de gestionar la memoria y responsable de traer el proceso a memoria principal, estas direcciones son fáciles de adquirir. Pero el procesador debe tratar con las referencias de memoria dentro del programa, de alguna forma el hardware del procesador y el software del SO deben poder traducir las referencias de memoria encontradas en el código del programa en direcciones físicas, que reflejan la ubicación actual del programa en la memoria principal.



Carga absoluta

La **carga absoluta** consiste en asignar direcciones físicas (direcciones de memoria principal) al programa en tiempo de compilación, este programa no es reubicable.

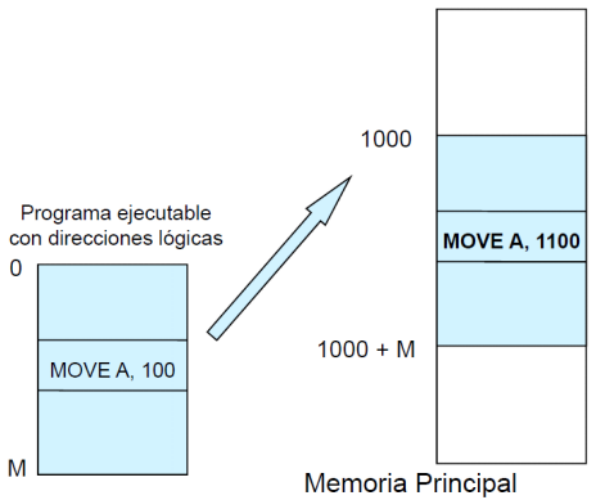
Reubicación: reubicación estática y reubicación dinámica

Reubicación estática

La **reubicación estática** consiste en:

- El compilador genera direcciones lógicas (relativas) de 0 a M.
- La decisión de dónde ubicar el programa en memoria principal se realiza en tiempo de carga.

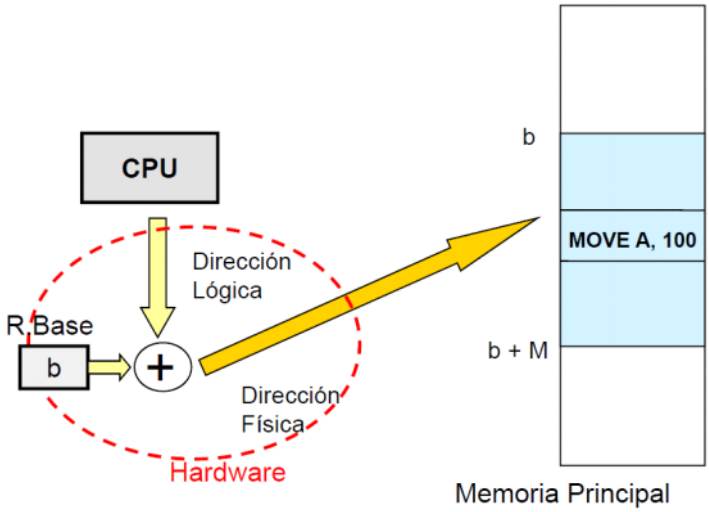
El cargador añade la dirección base de carga a todas las referencias relativas a memoria del programa.



Reubicación dinámica

La **reubicación dinámica** consiste en:

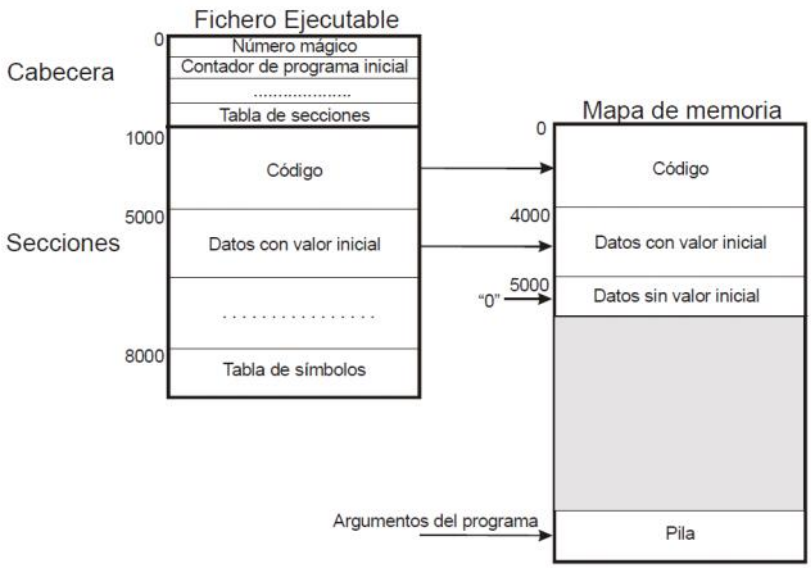
- El compilador genera de direcciones lógicas (relativas) de 0 a M.
- La traducción de direcciones lógicas a físicas se realiza en tiempo de ejecución luego el programa está cargado con referencias relativas.
- Requiere apoyo hardware.



Espacios para las direcciones de memoria

- Espacio de direcciones lógicas: conjunto de direcciones lógicas (o relativas) que utiliza un programa ejecutable.
- Espacio de direcciones físico: conjunto de direcciones físicas (memoria principal) correspondientes a las direcciones lógicas del programa en un instante dado.
- Mapa de memoria de un ordenador: todo el espacio de memoria direccionable por el ordenador, normalmente depende del tamaño del bus de direcciones.
- Mapa de memoria de un proceso: estructura de datos (que reside en memoria) que indica el tamaño total del espacio de direcciones lógico y la correspondencia entre las direcciones lógicas y físicas.

Ejemplo de mapa de memoria de un proceso



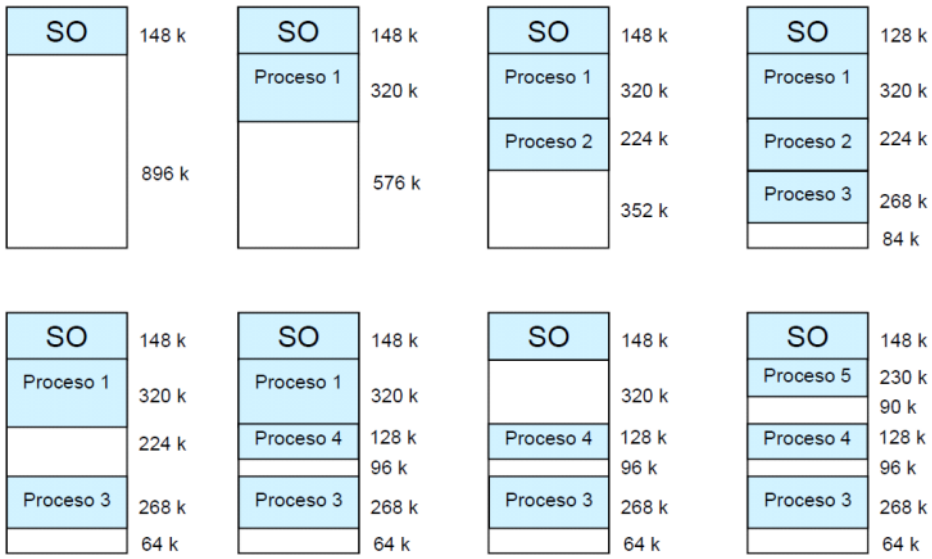
2.5 - Particionamiento de memoria

En la mayoría de esquemas para gestión de memoria se puede asumir que el SO ocupa alguna posición fija de la memoria principal y que el resto de la memoria principal está disponible para múltiples procesos. El esquema más simple es repartir la memoria en regiones con límites fijos.

Particionamiento fijo

En este caso, cualquier proceso cuyo tamaño es menor o igual que el tamaño de partición puede cargarse en cualquier partición disponible. Si todas las particiones están llenas y no hay ningún proceso en estado *Preparado* o *Ejecutando*, el SO puede mandar ***swap*** a un proceso de cualquiera de las particiones y cargar otro proceso de forma que el procesador tenga trabajo que realizar. Existen dos dificultades con el uso de las particiones fijas del mismo tamaño:

- Un programa podría ser demasiado grande para caber en una partición, en este caso, el programador debe diseñar el programa con el uso de ***overlays*** de forma que solo necesite una porción del programa en memoria principal en un momento determinado. Cuando se necesita un módulo que no está presente, el programa de usuario debe cargar dicho módulo en la partición del programa, superponiéndolo (***overlaying***) a cualquier programa o datos que haya allí.
- La utilización de memoria principal es extremadamente ineficiente, cualquier programa, sin importar lo pequeño que sea ocupa una partición entera. Este hecho se conoce como **fragmentación interna**.



Particionamiento dinámico

Para vencer algunas dificultades con particionamiento fijo, se desarrolló el **particionamiento dinámico**. Las particiones ahora son de longitud y número variable. Cuando lleva un proceso a la memoria principal, se le asigna exactamente tanta memoria como requiera y no más.

El problema de este método es que lleva a una situación en la que existen muchos huecos pequeños en memoria, a medida que pasa el tiempo la memoria se fragmenta más y más y la utilización de memoria decrementa. Este fenómeno se conoce como **fragmentación externa**.

Una técnica para eliminar la fragmentación externa es la **compactación**: de vez en cuando, el SO desplaza los procesos en memoria de forma que se encuentren contiguos y de este modo la memoria libre se encontrará unida en bloque, la desventaja de la compactación es que se trata de un procedimiento que consume tiempo y malgasta tiempo de procesador ya que requiere reubicación dinámica.

Paginación

El espacio de direcciones físicas de un proceso puede ser no contiguo. La memoria física se divide en bloques de tamaño fijo, denominados **marcos de página**, de tamaño potencia de dos. El espacio lógico de un proceso se divide conceptualmente en bloques del mismo tamaño, denominados **páginas**. Los marcos de página contendrán páginas de los procesos.

Tanto las particiones de tamaño fijo como variable son ineficientes en el uso de memoria.

Supóngase, sin embargo, que la memoria principal se divide en porciones de tamaño fijo relativamente pequeños y que cada proceso se divide también en porciones pequeñas del mismo tamaño fijo. A dichas porciones del proceso, conocidas como **páginas**, se les asigna porciones disponibles de memoria, conocidas como **marcos** o **marcos de páginas**. El espacio malgastado por fragmentación interna corresponde sólo a una fracción de la última página de un proceso y no hay fragmentación externa.

- **Direcciones lógicas*: número de página, desplazamiento.*** Las genera la CPU.
- **Direcciones físicas: *número de marco, desplazamiento.***

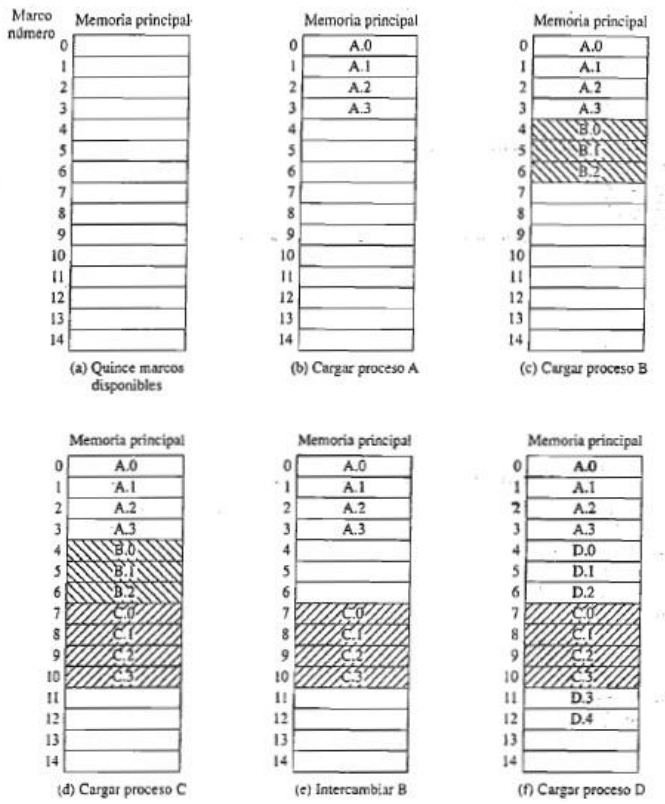
Tabla de páginas

Cuando la CPU genere una dirección lógica será necesario traducirla a la dirección física correspondiente. La **tabla de páginas** mantiene la información necesaria para realizar dicha traducción. Existe una tabla de páginas por proceso.

Además, el SO usa una **tabla de marcos de página**, que contiene información sobre cada marco de página.

Contenido de la tabla de páginas: una entrada por cada página del proceso:
- **Número de marco** en el que está almacenada la página si está en MP.
- **Modo de acceso** autorizado a la página (bits de protección).

Esquema de traducción

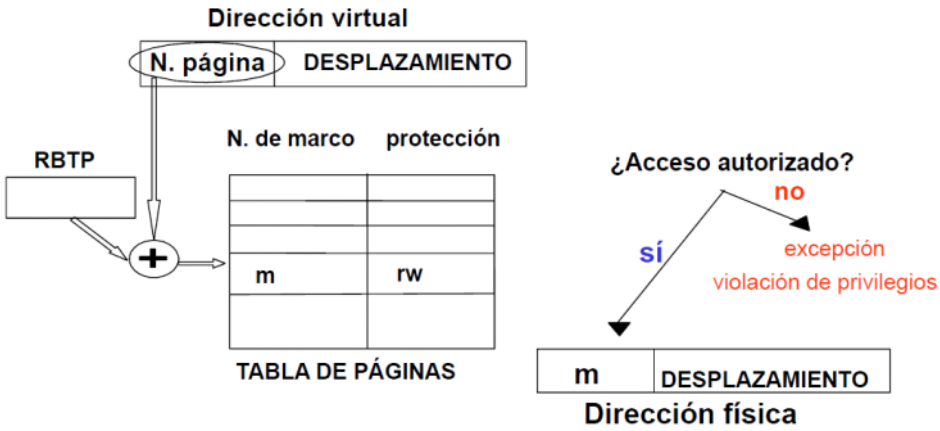


La imagen ilustra el uso de páginas y marcos. En un momento hay marcos en uso y otros libres. El SO mantiene una lista de marcos libres. El proceso A, almacenado en disco, está formado por cuatro páginas. En el momento de cargar este proceso el SO encuentra cuatro marcos libres y carga las cuatro páginas y el proceso A en dichos marcos. Si no hay marcos contiguos suficientes no pasa nada, ya que usando el concepto de dirección lógica, el SO mantiene una **tabla de páginas** por cada proceso que muestra la ubicación del marco por cada página del proceso. Dentro del programa, cada dirección lógica está formada por un número de página y un desplazamiento dentro de la página. La traducción de direcciones lógicas a físicas las realiza el

hardware del procesador.

Presentando una **dirección lógica** (número de página, desplazamiento) el procesador utiliza la tabla de páginas para producir una **dirección física** (número de marco, desplazamiento).

Para hacer este esquema conveniente, el tamaño de página y por tanto el de marco debe ser una potencia de 2, ya que así es fácil demostrar que la dirección relativa, que se define con referencia al origen del programa y la dirección lógica, expresada como número de página y un desplazamiento, es lo mismo. Las consecuencias de usar un tamaño de página que es una potencia de 2 son dobles: por un lado el esquema de direccionamiento lógico es transparente al programador, al ensamblador y al montador y por otro, es relativamente sencillo implementar una función que ejecute el hardware para llevar a cabo dinámicamente la traducción de direcciones en tiempo de ejecución.



*dirección física = m * tamaño página + desplazamiento*

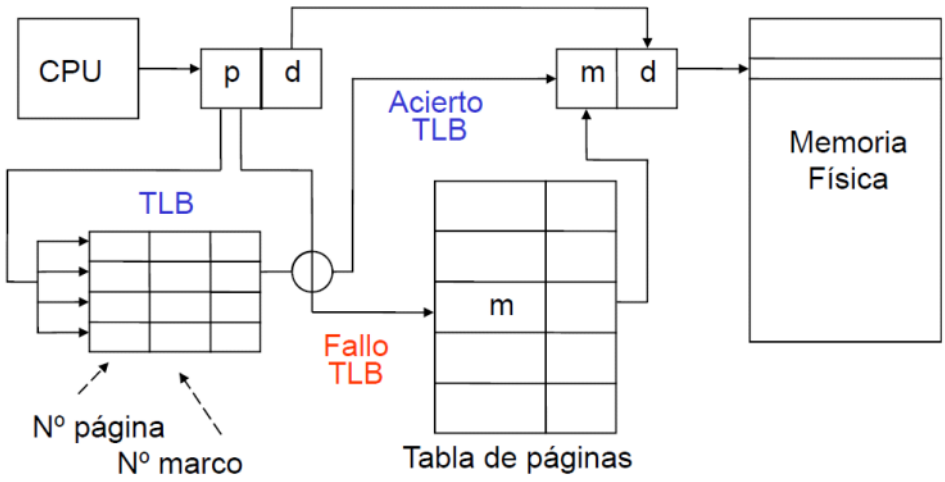
Implementación de la tabla de páginas

La tabla de páginas se mantiene en memoria principal. El **registro base de la tabla de páginas (RBTP)** apunta a la tabla de páginas (suele almacenarse en el PCB del proceso). En este esquema cada acceso a una instrucción o dato, **requiere dos accesos a memoria**, uno a la tabla de páginas y otro a la memoria.

Búfer de Traducción Adelantada (TLB)

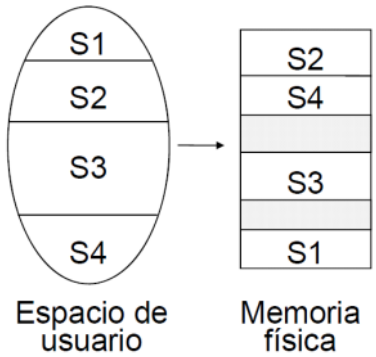
Este problema se resuelve con una caché hardware denominada búfer de traducción adelantada (Translation Look-aside Buffer, TLB) que se implementa como un conjunto de registros asociativos que permiten una búsqueda en paralelo. Así, para traducir una dirección si existe en el registro asociativo, obtenemos el marco; si no, la buscamos en la tabla de páginas y se actualiza el TLB con esta nueva entrada.

Esquema de traducción con TLB



Segmentación

Esquema de organización de memoria que soporta mejor la visión de memoria del usuario: un programa es una colección de unidades lógicas (segmentos), ej. procedimientos, funciones, pila, tabla de símbolos, matrices, etc.



Un programa se puede subdividir usando segmentación, en la cual el programa y sus datos asociados se dividen en un número de segmentos. No se requiere que todos los programas sean de la misma longitud, aunque haya longitud máxima de segmento.

Una dirección lógica utilizando segmentación está compuesta por dos partes: **número de segmento** y **desplazamiento**. Debido al uso de segmentos de distinto tamaño, es similar al particionamiento dinámico. En la ausencia de un esquema de *overlays* o el uso de memoria virtual, se necesitaría que todos los segmentos de un programa se cargaran en memoria para su ejecución. La diferencia, comparada con el particionamiento dinámico, es que con la segmentación un programa podría ocupar más de una partición y estas particiones no tendrían por qué ser contiguas. La segmentación elimina la fragmentación interna pero sufre la externa, aunque gracias a que se divide en piezas pequeñas, la fragmentación externa es menor.

La segmentación es visible y se proporciona como una utilidad para organizar programas y datos. El programador o compilador asigna programas y datos a diferentes segmentos. El inconveniente de este servicio es la limitación de tamaño de segmento máximo. Otra consecuencia de utilizar segmentos de distinto tamaño es que no hay relación simple entre direcciones lógicas y físicas. La segmentación hace uso de una tabla de segmentos y una lista de bloques libres de memoria principal, cada entrada de esta tabla tiene que proporcionar la dirección
inicial de la memoria principal del correspondiente segmento y la longitud del segmento (para asegurar que no se usan direcciones no válidas). Cuando un proceso entra en el estado Ejecutando, la dirección de su tabla de segmentos se carga en un registro especial utilizado por el hardware de gestión de memoria.

Tabla de segmentos

La tabla de segmentos aplica **direcciones bidimensionales** definidas por el usuario en direcciones físicas de una dimensión. Cada entrada de la tabla tiene los siguientes elementos (aparte de **presencia**, **modificación** y **protección**):

- **base:** dirección física donde reside el inicio del segmento en memoria.
- **tamaño:** longitud del segmento.

Implementación de la tabla de segmentos

La **tabla de segmentos** se mantiene en memoria principal. El **Registro Base de la Tabla de Segmentos (RBTS)** apunta a la tabla de segmentos (suele almacenarse en el PCB del proceso). El **Registro Longitud de la Tabla de Segmentos (STLR)** indica el número de segmentos del proceso; el número de segmento *s*, generado en una dirección lógica, es válido si es menor que la longitud de la tabla de segmentos (*s* < STLR) (suele almacenarse en el PCB del proceso).

Esquema de traducción