Tema 11

Soporte del Sistema Operativo

11.1. REQUERIMIENTOS DE LOS SISTEMAS OPERATIVOS.

El sistema operativo es básicamente un programa que controla los recursos del computador, proporciona servicios a los programadores y planifica la ejecución de otros programas. A partir de los µP de 16 bits, las CPU incorporan estructuras de apoyo a los sistemas operativos, por lo que resulta interesante una introducción a dos de las funciones básicas del SO que más inciden en la arquitectura de la CPU: la multiprogramación (o multitarea) y el control de memoria.

11.1.1. MULTIPROGRAMACIÓN.

La multiprogramación es la tarea central de los sistemas operativos modernos. Permite que múltiples programas de usuario o usuarios que se hallan en memoria se alternen entre la utilización de la CPU y los accesos a I/O, de manera que el procesador siempre se mantenga ocupado con un proceso mientras los demás esperan.

PLANIFICACION (SCHEDULING) DE ALTO NIVEL.

Determina qué programas son admitidos por el sistema para ser procesados. El planificador (proyecta) controla pues el grado de multiprogramación (número de procesos en memoria). Una vez admitido, un programa se convierte en un proceso y es añadido a la cola para ser tratado por el distribuidor. El planificador de alto nivel puede limitar el grado de multiprogramación para dar un servicio satisfactorio al conjunto actual de procesos.

PLANIFICACION A CORTO PLAZO (SHORT-TERM SCHEDULING).

Este planificador, conocido también como distribuidor (dispatcher), se encarga de decidir en cada momento cuál de los procesos admitidos por el anterior se ejecutará en siguiente lugar. Esta decisión se basa en el *estado del proceso*.

Estado del proceso.

Básicamente existen cinco posibles estados de un proceso:

1. **Nuevo**: El programa ha sido admitido por el planificador de alto nivel pero no está listo para se ejecutado. El sistema operativo inicializará el proceso, pasándolo al estado siguiente.

2. **Preparado**: El proceso está listo para ser ejecutado, y está esperando acceso al procesador.

3. **En ejecución**: El proceso está siendo ejecutado por el procesador.

4. **Esperando**: Se suspende la ejecución del proceso, en espera de algún recurso del sistema, como I/O.

5. **Parado**: El proceso ha sido terminado y será eliminado por el sistema operativo.

Para cada proceso, el sistema operativo debe mantener una información del estado. Para ello, cada proceso se representa en el SO por un ***bloque de control de proceso***, que contiene generalmente:

- **Identificador**: único para cada proceso actual.
- **Estado**: los tipos vistos anteriormente.
- **Prioridad**: nivel relativo de prioridad.
- **Contador de programa**: La dirección de la siguiente instrucción del programa a ser ejecutada.
- **Punteros de memoria**: Las direcciones de comienzo y final del proceso en memoria.
- **Contexto de datos**: son los datos de los registros del procesador para ese proceso.
- **Información de estado I/O**: incluye dispositivos I/O asignados a ese proceso, lista de ficheros correspondientes al mismo, etc.
- **Información adicional**: Puede incluir la cantidad de tiempo de procesador y tiempo de reloj utilizados, límites temporales, etc.

Cuando el procesador acepta un nuevo proceso, crea un bloque de control de proceso vacío y sitúa el proceso asociado en el estado 'Nuevo'. Después de que el sistema ha rellenado apropiadamente el bloque de control, el proceso se transfiere al estado 'Preparado'.

Técnicas de planificación.

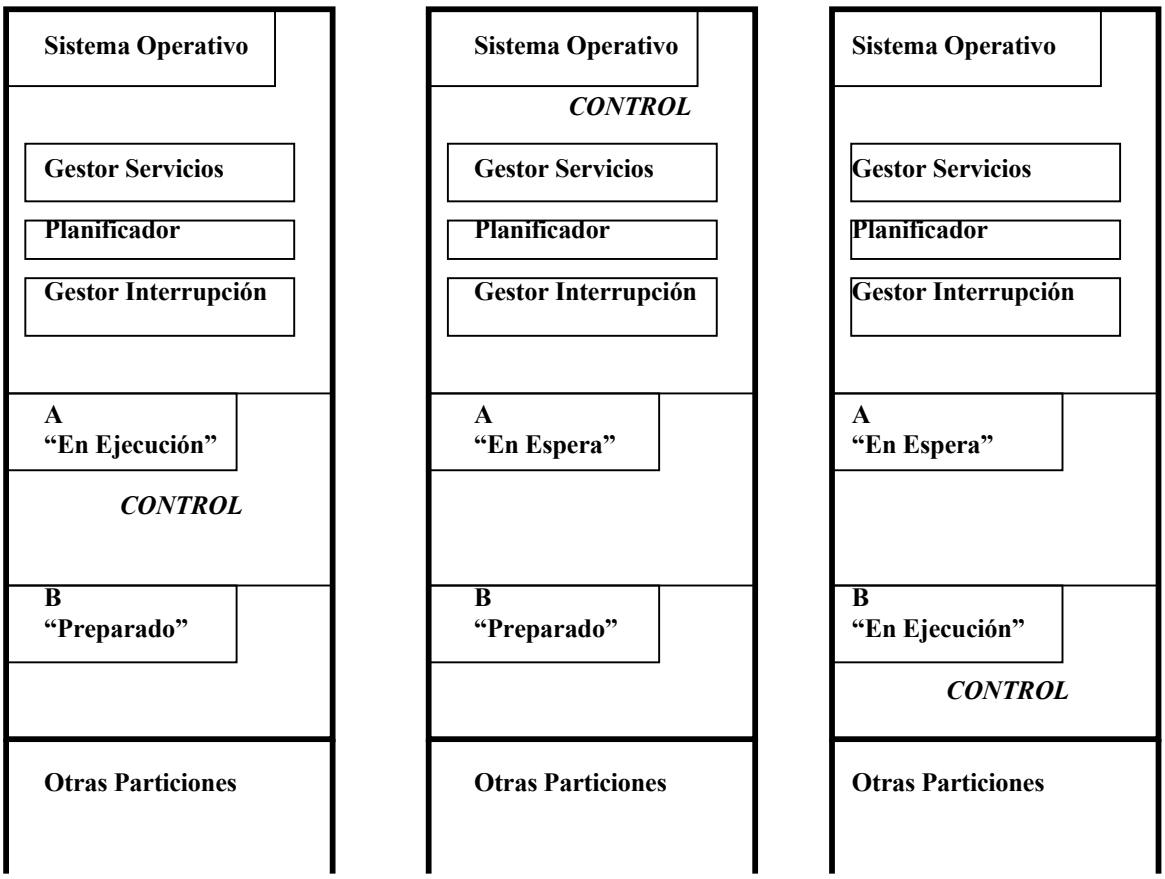


Figura 6.1. Ejemplo de planificación.

La figura muestra una memoria principal con particiones en un momento determinado. El núcleo del SO siempre está residente en memoria. Además existen dos procesos activos, A y B, cada uno en una partición distinta de memoria.

En t=0, el proceso A está siendo ejecutado, y el procesador toma las instrucciones a ejecutar de la partición correspondiente a A. En un momento determinado, el control pasa al sistema operativo. Esto puede producirse por tres razones fundamentalmente:

1. El proceso A realiza una llamada de servicio al SO (por ejemplo, para acceso a I/O). Se suspende la ejecución de A hasta que se satisfaga la llamada.

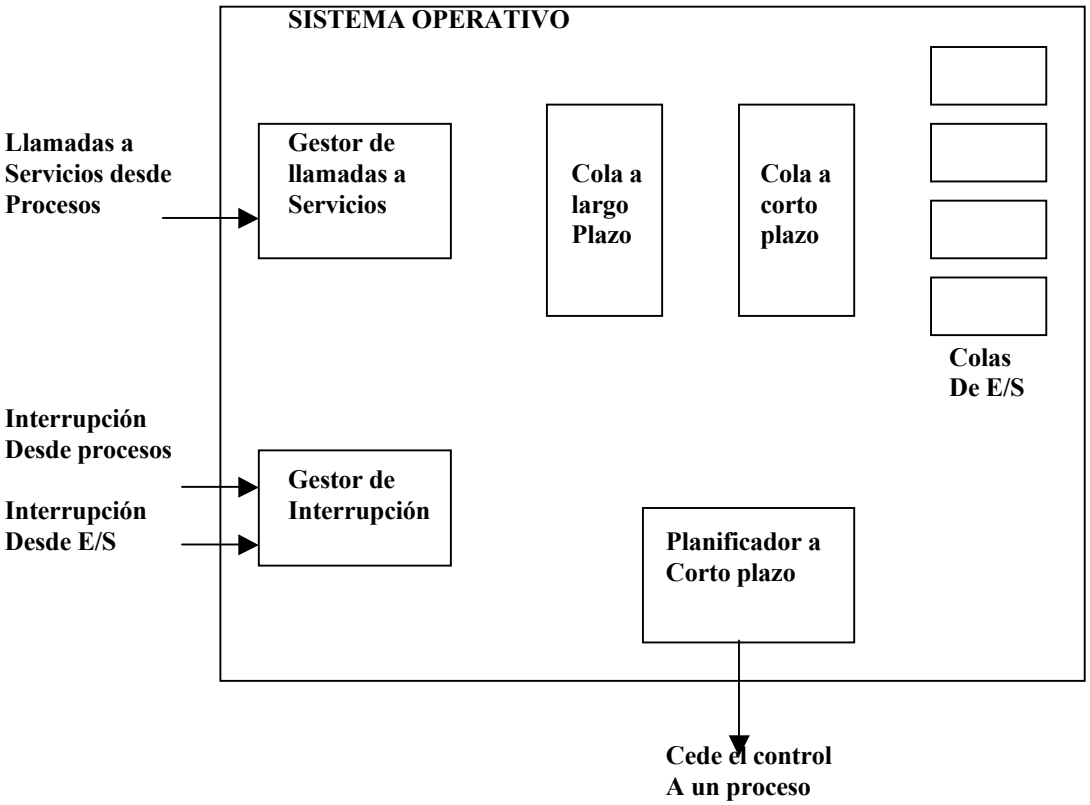
2. El proceso A produce una interrupción. El procesador deja de ejecutar A y transfiere el control al gestor de interrupciones en el SO. Existen varias posibilidades de que esto ocurra. Un ejemplo es un error, que puede estar producido por intentar ejecutar una instrucción privilegiada a la que no tiene acceso A. Otro es sobrepasar el límite de tiempo, que previene la monopolización del procesador por un solo proceso.

3. Otro evento no relacionado con el proceso A que requiere atención del procesador produce una interrupción (por ejemplo, el fin de una operación I/O previa).

En cualquier caso, el resultado es el siguiente. El procesador salva el contexto actual de datos y el PC para A en el bloque de control del proceso A y comienza a ejecutar el SO. Cuando termina la rutina correspondiente, el distribuidor, incluido en el SO, decide qué proceso se realizará en siguiente lugar. El SO ordena al procesador retomar el contexto del proceso B y continuar con su ejecución en el punto en que se había dejado.

Para realizar su tarea, el SO mantiene un cierto número de 'colas'. Cada una de ellas es una lista de espera que contiene los procesos en espera de acceso a algún recurso. La ***cola a largo plazo*** (long-term queue) es la lista de programas que esperan utilizar el sistema. Cuando las condiciones lo permitan, el planificador de alto nivel asignará memoria y creará un proceso para uno de los programas que esperan.

La ***cola a corto plazo*** (short-term queue) contiene todos los procesos en estado 'Preparado'. La selección entre ellos se realiza por algún algoritmo o utilizando prioridades. Finalmente existe una cola por cada dispositivo I/O, donde se almacenan todos los procesos que requieren acceso a ese dispositivo.



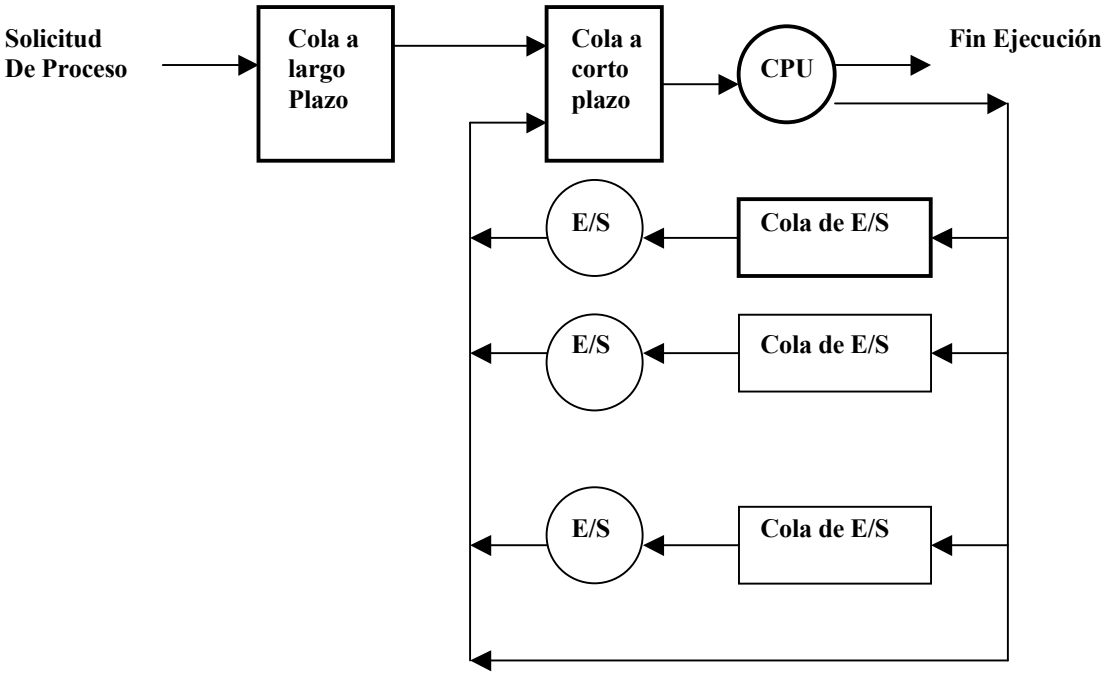


Figura 6.2. Elementos de un SO multiprogramado. Diagrama de colas.

Cada petición de nueva tarea o programa es colocada en la cola a largo plazo. Cuando los recursos van quedando libres, las peticiones se convierten en procesos activos, se pasan al estado 'Preparado' y se almacenan en la cola de corto plazo. El procesador alterna entre la ejecución del SO y los procesos del usuario. Mientras se está ejecutando el SO, se decide qué proceso de la cola de corto plazo será el siguiente a ejecutar.

El SO maneja también las colas de I/O. Cuando se ha completado una operación de I/O, el SO saca el proceso que la solicitó de la cola de I/O y lo coloca en la cola de corto plazo.

CONMUTACIÓN (SWAPPING).

La estructura de colas comentada, que permite implementar la multiprogramación, está almacenada en memoria. Generalmente, la menos utilizada (cola de largo plazo) estará en disco mientras que la de corto plazo puede estar en memoria principal. Esta estructura, aún mejorando las prestaciones del sistema, no evita completamente el que el procesador, debido a su mayor velocidad, tenga que esperar cuando todos los procesos se hallen en colas de I/O. Una solución sería expandir la memoria principal para soportar más procesos, pero resulta cara y la tendencia de cada proceso es a utilizar más memoria. Otra solución es conmutar colas entre memoria principal y disco, lo que implica una cierta gestión de memoria.

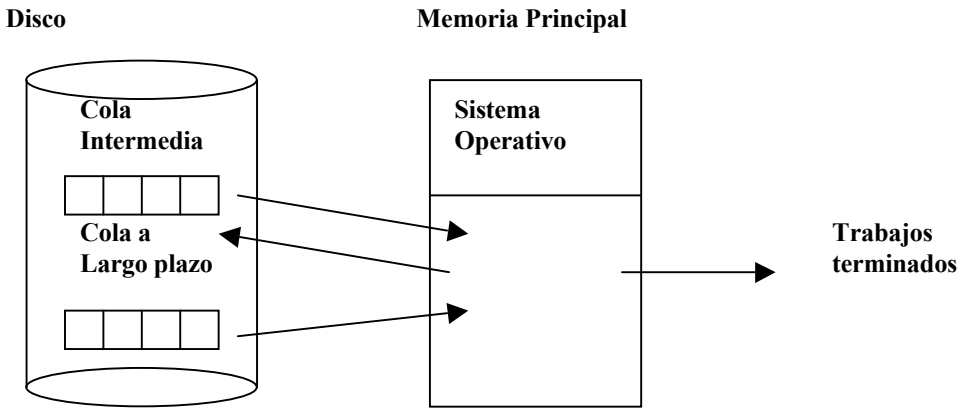


Figura 6.3. El uso de conmutación.

En un sistema simple, desde el punto de vista de gestión de memoria, las transferencias se hacen desde la cola de largo plazo en disco a la de corto plazo en memoria principal. Cuando los procesos se terminan, se sacan de la memoria principal. Si ninguno de los procesos en memoria principal están en el estado de 'Preparado', el procesador debe esperar.

El mecanismo de conmutación utiliza una tercera cola (cola intermedia), generalmente en disco, que contiene aquellos procesos no preparados de la cola a corto plazo y que son temporalmente desplazados de memoria principal, permitiendo así que el SO traiga otro proceso preparado. Este proceso puede tomarse de la misma cola intermedia (algún proceso que se conmutó pero cuyo servicio I/O ya está disponible), o un nuevo proceso de la cola a largo plazo, y llevarse a la cola de corto plazo, donde se ejecuta.

No obstante, la conmutación es una operación de I/O, ya que implica accesos a disco. Pero debido a que éste es generalmente el dispositivo I/O más rápido del sistema, la conmutación mejora las características del mismo.

11.1.2. CONTROL DE MEMORIA.

Con este método, el SO controla el tiempo del procesador y los recursos de I/O. El tercer recurso es la memoria. En un sistema multiprogramado, la memoria correspondiente al usuario debe subdividirse para acomodar múltiples procesos. Esta tarea la realiza el SO y se conoce como control de memoria, y resulta vital ya que una eficiente distribución permitirá disponer más tareas en memoria y necesitar menos accesos a dispositivos I/O, generalmente lentos, y que obligarían al procesador a permanecer en espera frecuentemente.

PARTICION.

El SO ocupa una porción fija de memoria principal. El resto está repartida para uso de los distintos procesos. El esquema más simple para repartir la memoria disponible es utilizar particiones de tamaño fijo. De todos modos, aunque las particiones son de tamaño fijo, no todas son de igual tamaño. Cuando un proceso es cargado en memoria, se coloca en la partición disponible más pequeña que lo pueda contener.

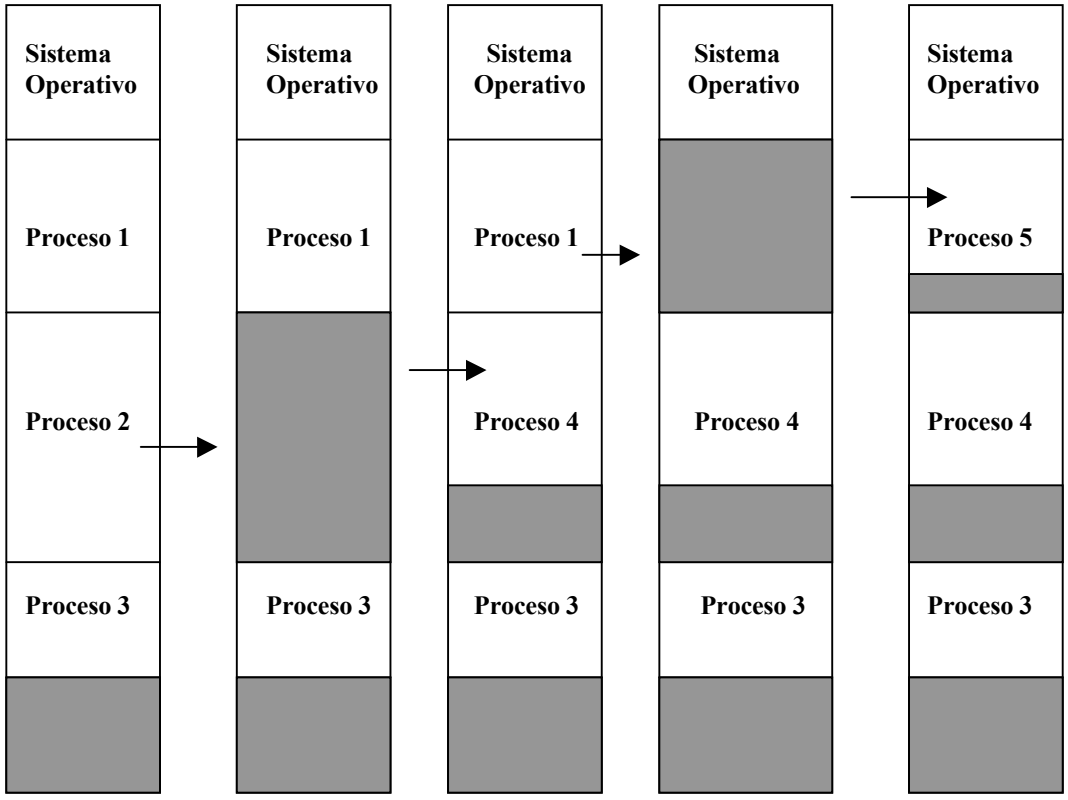


Figura 6.4. Partición de memoria de tamaño variable.

A pesar de estas consideraciones, las particiones de tamaño fijo desaprovechan gran cantidad de memoria. Una aproximación más eficiente es la partición de tamaño variable. De este modo, cuando un proceso es cargado en memoria, ocupa la cantidad exacta que necesita.

Inicialmente la memoria está vacía, excepto el SO. Se cargan los tres primeros procesos, a continuación del final del SO. Esto deja un hueco demasiado pequeño para un 4º proceso. Cuando van terminando los procesos, los huecos que van dejando se ocupan por nuevos procesos. Si la memoria existente entre procesos consecutivos permite la inclusión de un nuevo proceso, se introduce. No obstante, si los huecos son demasiado pequeños, se produce una cierta cantidad de huecos de memoria no utilizados. La utilización de memoria se degrada con el tiempo.

Una posible forma de solucionar esto es la '***compactación***'. Cada cierto tiempo, el SO realiza un desplazamiento de los procesos en memoria para localizar todos los huecos libres consecutivamente. No obstante, es un procedimiento que consume una cantidad de tiempo importante del procesador.

Otro punto importante es el hecho de que los procesos hacen referencias a direcciones tanto de datos como de instrucciones a las que hay que saltar. Los métodos descritos implican localizaciones en memoria cambiantes. Para solucionar esto, se distingue entre ***direcciones lógicas y físicas***. Una dirección lógica está expresada relativamente al comienzo del programa. La dirección física es la dirección real de memoria principal. Cuando el procesador ejecuta un proceso, automáticamente traduce direcciones lógicas a físicas sumando la posición de comienzo actual del proceso, llamada su dirección base, a cada dirección lógica. Este es otro ejemplo de requisitos a cumplir por el hardware de la CPU para soportar el manejo de memoria.

PAGINACION.

Los dos tipos de partición siguen siendo ineficaces en el manejo de memoria. Supongamos ahora que la memoria se reparte en porciones de igual tamaño fijo pero relativamente pequeñas, y que cada proceso es dividido también en porciones de iguales características. Entonces las porciones de programa, llamadas páginas, pueden ser asignadas a porciones disponibles de memoria, llamadas marcos (frames). De este modo, el espacio no utilizado de memoria se reduce a una fracción de la última página.

Como se muestra en la figura, en un momento dado algunas de las frames de memoria están en uso y otras están libres. La lista de las libres la controla el SO. El proceso A, almacenado en disco, ocupa 4 páginas. Cuando llega el momento de cargar este proceso, el SO encuentra 4 frames libres y carga las páginas de A.

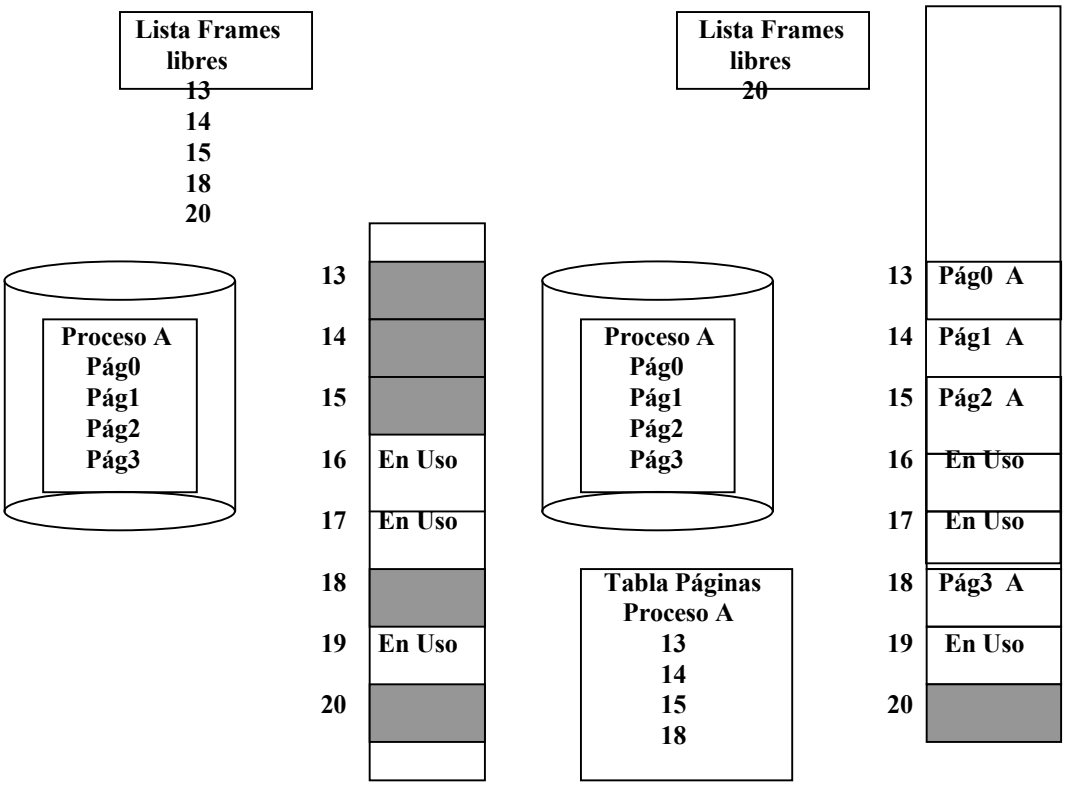


Figura 6.5. Asignación de frames libres.

Supongamos que no exisen 4 frames libres contiguas. para poder distribuir el proceso en frames discontiguas, el SO utiliza el concepto de dirección lógica, pero ya no es suficiente una única dirección base. El SO mantiene una ***tabla de página*** para cada proceso. Esta tabla contiene la localización de frame para cada página del proceso. En el programa, cada dirección lógica consiste de un número de página y una dirección relativa dentro de esa página. La translación de dirección lógica a física en el método de paginación se realiza por hardware de la CPU, por lo que ésta deberá saber cómo acceder a la tabla de página del proceso actual. Al encontrarse con una dirección lógica (número de página, dirección relativa), la CPU usa la tabla de página para producir una dirección física (número de frame, dirección relativa).

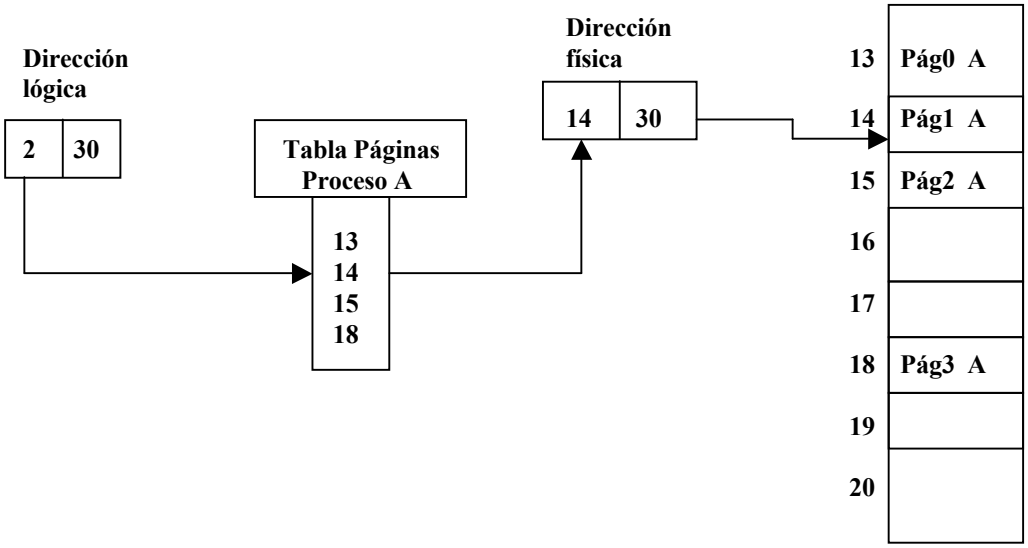


Figura 6.6. Direcciones lógica y física.

Para comprender el mecanismo de memoria virtual, debemos introducir un refinamiento en el método de paginación, conocido como ***paginación solicitada***, por la cual una página de un proceso se crea sólo cuando es necesaria, o solicitada.

Supongamos un proceso largo, consistente en un programa extenso más cierto número de arrays de datos. En un momento dado, tan sólo una pequeña sección de programa (por ejemplo, una subrutina) y quizá sólo uno o dos arrays de datos van a ser usados. Este es el principio de localización. Está claro que no conviene cargar muchas páginas de este proceso cuando sólo unas pocas van a ser utilizadas, por lo que sólo cargaremos esas pocas páginas. Si una operación hace referencia a una página que no está en memoria principal, o a otros datos, se activa un ***'fallo de página'***, que ordena al SO traer esa página a memoria.

Cuando el SO carga la nueva página, debe desechar alguna antigua. Es necesario algún mecanismo de control para determinar qué página desechar, ya que puede tratarse de alguna que vaya a ser accedida a continuación. Esto llevaría a una situación en la que el procesador ocuparía más tiempo manejando la memoria que ejecutando instrucciones ('trashing'). En los años '70 se trabajó en el desarrollo de algoritmos que decidieran, basándose en el histórico reciente, qué páginas tienen menor probabilidad de ser utilizadas a corto plazo.

Con paginación solicitada se puede trabajar con procesos mayores que la memoria principal. Puesto que el proceso se ejecuta realmente en memoria principal, ésta se denomina también memoria real. Pero el programador percibe una memoria de tamaño efectivo mayor (que está localizada realmente en el disco), por lo que se denomina memoria virtual.

Buffer de Traducción Rápida

En principio, toda referencia a memoria virtual puede ocasionar dos accesos a la memoria física: uno para captar el elemento de la tabla de páginas apropiada, y otro para captar el dato deseado, por lo que en principio se duplicaría el tiempo de acceso a memoria. Para resolver este problema, la mayoría de los esquemas de memoria virtual utilizan una caché especial para los elementos de la tabla de páginas, llamada buffer de traducción rápida (TLB: Translation Lookaside Buffer), que contiene los elementos de la tabla de páginas accedidos más recientemente. Debido al principio de localización de referencias, este esquema mejora el rendimiento del sistema.

El mecanismo de memoria virtual interactúa con la caché del sistema. La siguiente figura ilustra este funcionamiento combinado. Una dirección virtual vendrá dada por num_página:desplazamiento. Primero, el sistema de memoria consulta el TLB para comprobar si el descriptor de la tabla de páginas está incluído en él. Si es así, se genera la dirección física. Si no, se accede al elemento correspondiente de la tabla de páginas. Una vez obtenida la dirección real, se consulta la caché para comprobar si el bloque que contiene la palabra está presente. Si es así, se envía a la CPU. Si no, se busca en memoria principal. Deberán actualizarse también, en caso necesario, la TLB y la caché.

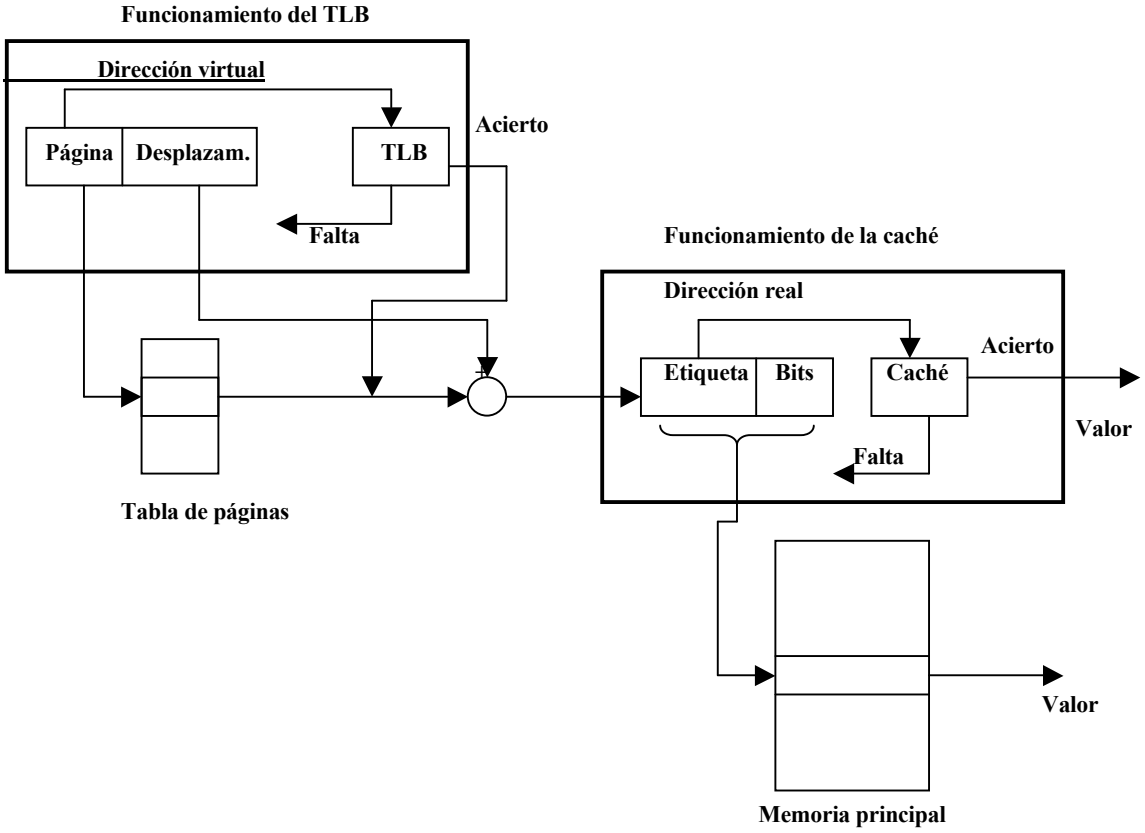


Figura 6.7. TLB y funcionamiento de la caché.

SEGMENTACION.

La segmentación es otra forma de subdividir la memoria direccionable. Mientras la paginación es transparente al programador, la segmentación es generalmente visible, y permite organizar programas y datos, a la vez que asociar privilegios y atributos de protección a los datos e instrucciones.

La segmentación permite al programador ver la memoria organizada en múltiples espacios de direcciones, o ***segmentos***. Los segmentos son de tamaño variable. Pueden existir varios segmentos de programa para varios tipos de programa, al igual que varios segmentos de datos. Cada segmento puede tener asignados ***derechos de acceso*** y uso. Las referencias a memoria constan de número de segmento y offset.

Las ventajas de esta organización son:

1. Simplifica el manejo de estructuras de datos variables. La estructura de datos puede ser asignada a un segmento propio y el SO aumentará o disminuirá el tamaño del segmento dependiendo de las necesidades.

2. Permite modificar y compilar independientemente los programas, sin necesidad de relinkar todo el conjunto de programas de un proceso.

3. Permite compartir programas de utilidades o tablas de datos entre procesos, localizándolos en segmentos que pueden ser direccionados por todos ellos.

4. Permite implementar mecanismos de protección, asignando privilegios de acceso.

Estas ventajas no están disponibles en paginación. Para combinar características, algunos sistemas están equipados con el hardware y el software de SO necesarios para proporcionar ambos.