Tema 2: Introducción a los Sistemas Operativos

Autor: Miguel Ángel Moreno Castro

1. Definición de Sistema Operativo

Un **Sistema Operativo** es un *programa (o conjunto de programas) de control* que tiene por objeto *facilitar* el uso del computador y conseguir que éste se utilice *eficientemente*.

Es un *programa de control*, ya que se encarga de gestionar y asignar recursos hardware que requieren los programas.

El sistema operativo facilita el uso del computador, haciendo transparente al usuario las características hardware concretas de los dispositivos, y también hace que el computador se utilice eficientemente.

El sistema operativo junto con el hardware del computador definen un nivel de **máquina operativa**, ya que este conjunto permite utilizar el computador sin tener que conocer muchos detalles del hardware y ofrece servicios adicionales tales como *memoria virtual*, *dispositivos de E/S virutales*, *multiprogramación*, etc.

El sistema operativo puede considerarse como un programa constituido por una serie de módulos. Estos módulos se lanzan a ejecución por medio de **llamadas al sistema**. Las llamadas al sistema pueden ser realizadas por los usuarios (directamente, con las órdenes del **lenguaje de control**) o por los programas. Son implementadas a través de una trampa (*trap*) o *interrupción software*.

Las llamadas al sistema también se conocen conocen con el nombre de **instrucciones virtuales**, para diferenciarlas de las instrucciones máquina del procesador que son las auténticas instrucciones.

Cuando en un computador introducimos una orden, ésta es captada por el **intérprete de ódenes** (*shell*), que es un programa independiente del sistema operativo. El intérprete se encarga de traducir o descomponer la orden en llamadas al sistema, de forma que se realicen las operaciones asociadas a la ejecución de la orden.

Para comunicarse cómodamente con el computador, el sistema operativo va acompañado de módulos que definen la **interfaz del usuario**. En general una interfaz con ventanas y/o con iconos se denomina **interfaz gráfica de usuario (GUI, *Graphical User Interface*)**.

2. Gestión del Procesador

Existen dos formas básicas de trabajar con un computador: por **lotes** (o cola de trabajos) y en forma **interactiva**. En esta última el procesador está constantemente atendiendo al usuario en una forma que podría denominarse de *diálogo*.

La gestión del procesador por parte del sistema operativo se centra en el concepto de **proceso**.

> Un **proceso** es un programa en el que se ha iniciado su ejecución, y que es tratado por el sistema operativo como un todo. Un programa por sí sólo es un ente pasivo, mientras que un proceso es un ente activo.

2.1. Concepto de Monoprogramación

Los primeros sistemas operativos se denominaban de **monoprogramación** o *procesamiento en serie*, en ellos se ubicaba un programa en memoria principal, además del sistema operativo, y hasta que no finaliza la ejecución de dicho programa no empezaba a ejecutarse otro.

Claramente se observa que con este sistema:

- La memoria casi siempre queda sólo parcialmente ocupada, es decir, se desaprovecha la memoria principal.
- Siempre que hay una operación de E/S, el procesador no se utiliza, y ya que los tiempos de estas operaciones suelen ser mucho mayores que los tiempos del procesador, en consecuencia se desaprovecha el procesador.
- De igual modo, los periféricos están infrautilizados, ya que el programa en ejecución en un instante dado obviamente sólo puede utilizar uno de los recursos hardware.

En consecuencia, la monoprogramación no es eficiente, para conseguir mayor eficiencia en la utilización de los distintos recursos que ofrece un computador se desarrolló la **multiprogramación**.

2.2. Concepto de Multiprogramación

Un **sistema operativo multiprogramación** carga en memoria principal un programa de la cola serie y todos los programas interactivos que se le presten, y el **planificador de trabajos a corto plazo** (o *distribuidor*) del sistema operativo asigna el procesador sucesivamente a los procesos preparados de forma que el procesador se aproveche al máximo y los distintos procesos vayan avanzando en su ejecución sin necesidad de que finalice completamente uno para iniciar la ejecución de otro.

> Se aprovechan los tiempos muertos del procesador que se tienen con la monoprogramación, los tiempos muertos en periféricos y el espacio de memoria principal no ocupado por el proceso o sistema operativo.

MultiprogramaciónSO

Siempre que un proceso se bloquea o concluye, el núcleo del sistema operativo toma el control del procesador para realizar las acciones oportunas. Cuando hay multiprogramación y se detiene la ejecución de un proceso, *Px*, para dar turno a otro, *Py*, (que presumiblemente se había interrumpido previamente), el sistema operativo realiza un cambio de contexto consistente en:

- Actualizar el bloque de control *PCB-Px* del proceso *Px* interrumpido y se salvaguardan los contenidos de los registros del procesador, biestables indicadores, punteros de archivos de discos y punteros de las pilas, etc.
- Restaurar los contenidos de los registros del procesador, biestables indicadores, etc, con los valores del *PCB-Py* del proveso *Py* que va a continuar su ejecución, y cambiar el estado del proceso activo; de esta forma continua sin problema la ejecución del proceso *Py* previamente interrumpido.

Obsérvese que de esta forma el proceso *Px*, cuando el distribuidor le dé nuevamente el turno, podrá renaudar su ejecución justo en el punto exacto en el que se interrumpió.

2.2.1. Asignación del Procesador a los Procesos Teniendo en cuenta cómo se toma la decisión de ir alternando la ejecución de los procesos, la multiprogramación se puede realizar de dos formas básicas: no apropiativa y apropiativa.

- **Multiprogramación no apropiativa (*Nonpreemptive*)**

En este caso, una vez que un proceso está activo, continúa ejecutándose hasta que termina, ó se bloquea por el inicio de una operación de E/S o de cualquier otro servicio solicitado al sistema operativo, ó el propio proceso hace una llamada al sistema operativo para ceder el procesador a otro proceso.

En el momento de interumpirse la ejecución del proceso, el distribuidor, de acuerdo con algún algoritmo de planificación, da el turno a otro de los procesos pendientes que se encuentren en el estado preparado.

> Problema: Un proceso con mucho tiempo de procesador (mucho cálculo) y pocas E/S puede monopolizar el procesador hasta que acabe su ejecución.

Solución: Los sistemas de Multiprogramación Apropiativa resuelven este problema ya que con ellos no se espera a que un proceso pase al estado de bloqueado para que el distribuidor lo detenga y dé el turno a otro proceso que esté preparado.

- **Multiprogramación Apropiativa o Preferente (*Preemptive*)**

En la **multiprogramación apropiativa** el sistema operativo puede interrumpir en cualquier momento el proceso activo apropiándose del procesador con objeto de dar paso a otro proceso que esté preparado.

El sistema operativo detiene la ejecución de un proceso de acuerdo con algún criterio preestablecido, por ejemplo cuando transcurre un determinado tiempo o cuando se presenta un proceso de mayor prioridad. Las decisiones de cuándo se detiene la ejecución de un proceso y de cuál de los procesos preparados pasa a activo se efectúa de acuerdo con un determinado algoritmo de planificación.

1.2.2. Algoritmos de Planificación Si existen varios procesos preparados, un problema que debe resolver el distribuidor es elegir a cuál de ellos darle el turno. El **distribuidor** se encarga de implementar un **algoritmo de planificación**.

Uno de los algoritmos de planificación de mayor interés es el de **turno rotatorio (*round robin*)**. A cada uno de los procesos en memoria se le asigna un intervalo de tiempo fijo, o quantum y se realiza un cambio de contexto de un proceso activo a otro preparado cuando al activo o se le acaba el quantum o se bloquea.

Un proceso suele tener asociada una prioridad. Pueden existir **prioridades estáticas** (dadas en función de la relevancia de los procesos o usuarios), o **prioridades dinámicas** (que asigna y cambia el sistema operativo), o **prioridades mixtas**, mezcla de las dos anteriores.

Otro algoritmo de interés es el de **planificación por prioridad**, según el cual el distribuidor da el turno al proceso preparado con mayor prioridad asignada por el propio planificador. A veces, para que el de mayor prioridad no monopolice el uso del procesador, a cada interrupción del reloj de tiempo real se le baja su prioridad (**planificación con realimentación**).

Para obtener un aprovechamiento equilibrado de todos los recursos del computador y obtener mayor productividad es un buen criterio utilizar el algoritmo **HRRN (*Highest Response Ratio Next*)**, con el que se da prioridad a los procesos que tienen más tiempo de E/S en relación con su tiempo de procesador.

Otros algoritmos de planificación son el **FCFS (*First Come First Served*)**, **SPN (*Shortest Process Next*)**, y **SRT (*Shortest Remaining Time*)**.

2.3. Modos de Procesamiento

Un sistema multiusuario es un sistema de multiprogramación que prevé el uso concurrente de distintos usuarios, con identificación, autentificación y control de los mismos.

- **Procesamiento en serie**

Con los primeros computadores, el programador interaccionaba directamente con el hardware del computador; no existía ningún sistema operativo. Los programas en código máquina se cargaban a través del dispositivo de entrada (por ejemplo, un lector de tarjetas). Si un error provocaba la parada del programa, unas luces indicaban la condición de error. El programador podía entonces examinar los registros del procesador y la memoria principal para determinar la causa de error. Si el programa terminaba de forma normal, la salida aparecía en la impresora.

- **Sistemas en lotes sencillos**

La idea central bajo el esquema de procesamiento en lotes sencillos es el uso de una pieza de software denominada **monitor**. Con este tipo de sistema operativo, el usuario no tiene que acceder directamente

a la máquina. En su lugar, el usuario envía un trabajo a través de una tarjeta o cinta al operador del computador, que crea un sistema por lotes con todos los trabajos enviados y coloca la secuencia de trabajos en el dispositivo de entrada, para que lo utilice el monitor. Cuando un programa finaliza su procesamiento, devuelve el control al monitor, punto en el cual dicho monitor comienza la carga del siguiente programa.

- **Sistemas en lotes multiprogramados**

El procesador se encuentra frecuentemente ocioso, incluso con el secuenciamiento de trabajos automático que proporciona un sistema operativo en lotes simple. El problema consiste en que los dispositivos de E/S son lentos comparados con el procesador. Esta ineficiencia puede evitarse si, cuando un trabajo necesita esperar por la E/S, se asigna el procesador a otro trabajo, que probablemente no esté esperando por una operación de E/S. Más aún, se puede expandir la memoria para que albergue más programas y pueda haber multiplexación entre todos ellos.

- **Sistemas de Tiempo Compartido**

En un sistema de tiempo compartido, múltiples usuarios acceden simultáneamente al sistema a través de terminales, siendo el sistema operativo el encargado de entrelazar la ejecución de cada programa de usuario en pequeños intervalos de tiempo o cuantos de computación. Por tanto, si hay n usuarios activos solicitando un servicio a la vez, cada usuario sólo verá en media *1/n* de la capacidad de computación efectiva, sin contar la sobrecarga introducida por el sistema operativo. Sin embargo, dado el tiempo de reacción relativamente lento de los humanos, el tiempo de respuesta de un sistema diseñado adecuadamente debería ser similar al de un computador dedicado.

2.4. Intercambio Memoria Principal/Disco

El número de trabajos ejecutándose concurrentemente depende (entre otros factores) de la capacidad de la memoria principal.

Para evitar una limitación de usuarios concurrentes existe un procedimiento consistente en transvasar a disco (*roll out*) un proceso de los que están residiento en memoria, para dar cabida a otro que se transvasa de disco a memoria (*roll in*). Este procedimiento, denominado **intercambio memoria principal/disco (*swapping*)** es gestionado por un módulo del sistema operativo, denominado **planificador a medio plazo** o **intercambiador**, y permite al computador realizar concurrentemtente un número de procesos que no esté limitado por la capacidad de la memoria principal. La eficiencia o velocidad de esta técnica depende de la velocidad de transferencia memoria/disco y de la velocidad de funcionamiento de la unidad de disco, necesitándose disponer de DMA y/o procesadores de E/S.

3. Procesos

Un **proceso** es un programa en el que se ha iniciado su ejecución, y que es tratado por el sistema operativo como un todo. Un programa por sí sólo es un ente pasivo, mientras que un proceso es un ente activo.

> Un mismo programa no puede desarrollar más de un proceso. Dentro de los procesos asociados a varios programas podemos establecer nuevos hilos que creen nuevos procesos para el mismo programa.

3.1. Bloque Control de Proceso (*PCB, Process Control Block*)

Contiene datos que el sistema operativo necesita para controlar el proceso:

- **Identificador de Proceso (*PID, Process IDentificator*)**: Un identificador único asociado al proceso, para distinguirlo del resto de procesos.
- **Estado**: En que situación se encuentra el proceso en cada momento (modelo de estado).

- **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 de programa y los datos asociados a dicho proceso, además de cualquier bloque de memoria compartido con otros procesos.
- **Datos del contexto de ejecución**: Contenido de los registros del procesador cuando el proceso se está ejecutando.
- **Información relacionada con recursos que utilice del sistema**: Incluye peticiones de E/S pendientes, dispositivos asignados a dicho proceso, la cantidad de tiempo de procesador/reloj utilizados, etc.

El **estado** completo del proceso en un instante dado se almacena en su **PCB**. Esta estructura permite el desarrollo de técnicas potentes que aseguren la coordinación y la cooperación entre los procesos. Una **traza de ejecución** es un listado de la secuencia de las instrucciones de un programa que realiza el procesador para un proceso.

> Un mismo programa a la hora de ejecutarse puede tener varias trazas de ejecución dependiendo del proceso que quiera ejecutar.

Las **llamadas al sistema** es la forma en la que se comunican los programas de usuario con el sistema operativo en tiempo de ejecución. Son peticiones de servicio que se hace el proceso al sistema operativo (ej: solicitudes de E/S, gestión de procesos/memoria).

Se implementan a través de una trampa (***trap***) o *"interrupción software"*.

> ***Dispatcher***: Módulo del sistema operativo que se encarga de realizar el intercambio de procesos en el procesador.

3.2. Estados de un proceso

Un **proceso**, desde el punto de vista de su ejecución, puede estar en una de diversas situaciones o estados, cuyo número y denominación depende del sistema operativo.

Un **proceso nonato** es un programa que no ha iniciado su ejecución; es decir, en memoria o disco pendiente de que se inicie su ejecución. Puede ser, un programa retenido en una cola esperando a que el **planificador a largo plazo** lo seleccione, o un programa en disco que puede ser seleccionado por medio de un icono para ejecutar.

Un proceso está en **estado preparado** cuando se encuentra en memoria principal, sin operaciones de E/S pendientes, y apto para entrar (o continuar) en ejecución en el instante que el **planificador a corto plazo (*distribuidor*)** le asigne el procesador.

El **estado ejecutándose** corresponde al proceso que en ese momento está siendo atendido por el procesador. Un proceso activo sale del estado de ejecución, en un sistema de tiempo compartido, cuando pasa a **estado bloqueado** o el distribuidor lo interrumpe para atender a otro de mayor prioridad.

Un proceso pasa del estado bloqueado al estado preparado cuando acaba la operación de E/S pendiente. Cuando finaliza la ejecución de un proceso o se detecta un error grave en él, el proceso pasa a **estado finalizado**, liberando el sistema operativo la zona de memoria que ocupaba el proceso.

3.3. Modos de ejecución del procesador

- **Modo Usuario**: El programa (de usuario) que se ejecuta en este modo sólo tiene acceso a un subconjunto de los registros del procesador, a un subconjunto del repertorio de instrucciones máquina y a un área de la memoria
- **Modo Kernel**: El programa (SO) que se ejecuta en este modo tiene acceso a todos los recursos de la máquina, tanto software como hardware.

El modo de ejecución (incluido en PSW) cambia a modo kernel, automáticamente por hardware, cuando se produce una interrupción (se compara la prioridad), una excepción (*overflow*) o una llamada al sistema (leer disco).

Seguidamente se ejecuta la rutina del sistema operativo correspondiente al evento producido, y finalmente, cuando termina la rutina, el hardware restaura automáticamente el modo de ejecución a modo usuario.

3.3.1. Pasos en la operación de cambio de modo Se ejecuta una rutina del SO en el contexto del proceso que se encuentra en estado **ejecutándose**, siempre que el SO pueda ejecutarse, es decir, como resultado de una interrupción, excepción o llamada al sistema.

- El hardware automáticamente salva como mínimo el *PC* y *PSW* y cambia el bit de modo a modo kernel.
- Determina automáticamente la rutina del SO que debe ejecutarse y cargar el *PC* con su dirección de comienzo.
- 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.
- Volver de la rutina del SO al proceso que se estaba ejecutando. El hardware automáticamente restaura en el procesador la información del PC y PSW previamente salvada.

3.4. Concepto de Hebra

El sistema operativo UNIX introdujo el concepto de hebra (*thread*), soportado por los sistemas operativos modernos.

Un proceso puede descomponerse en distintas hebras o tareas diferentes e independientes, que se pueden ejecutar "en paralelo" (*concurrentemente*) entre ellas e incluso con las de otros procesos. Cada hebra tiene asociado su estado, prioridad, privilegios, etc.

El concepto de **proceso** (**tarea**) tiene dos características diferenciadas que permiten al SO:

- Controlar la asignación de los recursos necesarios para la ejecución de programas.
- La ejecución del programa asociado al proceso de forma intercalada con otros programas.

El concepto de **proceso** (**tarea**) y **hebras asociadas** se basa en separar estas dos características:

- La tarea se encarga de soportar todos los recursos necesarios (incluida la memoria).
- Cada una de las hebras permite la ejecución del programa de forma independiente del resto de hebras.

Las ventajas de creación de hebras frente a nuevos procesos son:

- Menor tiempo de creación de una hebra en un proceso ya creado que la creación de un nuevo proceso.
- Menor tiempo de finalización de una hebra que de un proceso.
- Menor tiempo de cambio de contexto (hebra) entre hebras pertenecientes al mismo proceso.
- Facilitan la comunicación entre hebras pertenecientes al mismo proceso.
- Permiten aprovechar las técnicas de programación concurrente y el multiprocesamiento simétrico.

4. Gestión de la Memoria

Un programa máquina es una secuencia de instrucciones en código máquina. Estas instrucciones, en el momento de ejecutarse *"encajan"* en palabras de memoria que pueden numerarse correlativamente de la *0* a la *n-1*. Estas direcciones se denominan **direcciones virtuales** o **lógicas**, *dv*.

El programa anterior, al ser cargado en memoria, ocupará n determinadas posiciones de la misma. Supongamos que las instrucciones del programa se almacenan consecutivamente; si se cargan a partir

de la dirección *dB*, el programa quedará ubicado de la dirección *dB* a la *dB + (n-1)*. A *dB* se le suele denominar **dirección base** y a las direcciones *df* en que realmente se almacena la instrucción de la **dirección virtual** *dv*, se le denomina **dirección física**. Se verifica que:

`df = dB + dv  para todo  0 <= dv < n-1`

A veces el programa se considera por el sistema operativo dividido en segmentos. Un **segmento** es un grupo lógico de información, tal como un programa (**segmento de código**), una pila asociada al programa (**segmento de pila**), o un conjunto de datos asociados al programa (**segmento de datos**). Un programa ejecutable es una colección de segmentos.

En los sistemas operativos de monoprogramación, la memoria principal se puede organizar de diversas maneras. El sistema operativo puede ocupar las primeras posiciones de memoria o las últimas (que pueden ser de tipo ROM), o incluso parte del sistema operativo puede estar en la zona RAM de direcciones bajas, y otra parte de él (los gestores de periféricos) y el cargador inicial en ROM, en las direcciones altas. Esta última situación se corresponde con la de los PC´s compatibles, en los que la ROM se denomina **Sistema Básico de Entrada/Salida (BIOS, *Basic Input Output System*)**.

Las transformaciones entre direcciones virtuales y físicas antiguamente se realizaba por software, pero en la actualidad se suele realizar con ayuda de un registro base (donde se carga previamente la dirección base) que se encuentra o en el propio procesador o en unos circuitos específicos que forman parte de una **Unidad de Gestión de Memoria (MMU, *Memory Management Unit*)**, que se integra en el mismo chip del microprocesador.

> Reubicación: Capacidad de cargar y ejecutar un programa en un lugar arbitrario de la memoria.
>
> Carga absoluta: Consiste en asignar direcciones físicas (direcciones de memoria principal) al programa en tiempo de compilación. El programa no es reubicable.

La asignación de memoria para distintos procesos ejecutándose concurrentemente suele hacerse, dependiendo del sistema operativo, de alguna de las siguientes formas:

- **Particiones**

La memoria se divide en cierto número de **particiones** o zonas, cada una de las cuales contendrá un proceso. Las direcciones base son las direcciones de comienzo de cada partición, y el tamaño de estas es determinado por el operador o por el sistema operativo.

El sistema operativo mantiene una tabla en la que cada fila corresponde a una partición, conteniendo la posición base de la partición, su tamaño (no todas las particiones tienen por qué ser iguales) y el estado de la partición (ocupada o no ocupada). El planificador de trabajos, una vez que una partición está libre, hace que se introduzca el programa de máxima prioridad que haya en la cola de espera y que quepa en dicha partición.

Si el espacio de una partición es m palabras y el programa ocupa n posiciones, se verificará siempre que: `n <= m`

Cada partición tendrá unas posiciones no utilizasas si n < m. Se denomina **fragmentación interna** o **fragmentación de una partición** a: `g = m - n`.

Los programas son introducidos por el sistema operativo inicialmente en posiciones consecutivas de memoria, no existiendo por tanto particiones predefinidas.

El sistema operativo puede gestionar el espacio de memoria usando una **tabla de procesos**, en la que cada línea contiene el número de proceso o identificativo del mismo, el espacio que ocupa, y la dirección base. Existe una tabla complementaria a la anterior con los fragmentos o huecos libres. El planificador de trabajos, periódicamente consulta la tabla, introduciendo en memoria los programas que quepan en los fragmentos.

Obviamente, al iniciarse una sesión de trabajo se carga el primer programa, dejando un fragmento libre donde se pueden incluir otros programas. Al ir acabando de ejecutarse los procesos, el número de fragmentos crecerá (prodrá llegar a haber tantos como procesos hayan pasado por memoria) y el espacio de cada uno de ellos disminuirá, llegando un momento en que el porcentaje de memoria aprovechado es muy reducido.

El problema puede resolverse haciendo uso de la **Compactación**, agrupar todos los fragmentos cuando acaba la ejecución de un proceso, quedando así sólo uno grande. La compactación se efectúa cambiando de sítio o **reubicando** los procesos en ejecución.

- **Reubicación**

La **reubicación** es la capacidad de ubicar un proceso en posiciones distintas de memoria. Puede ser durante la carga o durante la propia ejecución. Si se realiza en tiempo de carga se denomina **estática**; el cargador convierte las direcciones lógicas en direcciones físicas sumándole la dirección base y sitúa el proceso en memoria a partir de esa dirección base. Una vez cargado el proceso en memoria puede cambiar de ubicación durante esa ejecución del proceso puesto que las direcciones que se han escrito en memoria son direcciones físicas. Con reubicación estática sólo cambia la ubicación de una ejecución del proceso a otra.

Si la reubicación se realiza durante la ejecución del proceso se denomina **dinámica**. Para ello es necesario que el cargador no haga la conversión de direcciones lógicas en físicas, por lo que las direcciones que se escriben en memoria son direcciones lógicas. Será durante la ejecución de cada instrucción cuando se haga la conversión de dirección lógica en dirección física.

> Mapa de memoria de un ordenador: Todo el espacio de memoria direccionable por el ordenado. Normalmente depende del tamaño del bus de direcciones.
>
> Mapa de memoria de un proceso: Se almacena en una estructura de datos (que reside en memoria) donde se guarda el tamaño total del espacio de direcciones lógico y la correspondencia entre las direcciones lógicas y las físicas.

4.1. Paginación

Con este procedimiento la memoria principal se estructura en **marcos de página** de longitud fija. Cada marco de página se identifica con un número correlativo (de 0 a FFF, en los ejemplos). Asimismo los procesos de los usuarios se dividen en zonas consecutivas, denominadas **páginas lógicas** o **virtuales**. Para un sistema dado, la capacidad de página y marco de página son coincidentes, de forma que cada página se almacena en un marco.

El fundamento de la paginación reside en que no es necesario que un proceso se almacene en posiciones consecutivas de memoria. Las páginas se almacenan en marcos de página libres independientemente de que estén o no contiguos.

Las **direcciones lógicas**, que son las que genera la CPU, se dividen en *número de página* (bits más significativos) y *desplazamiento* (o posición relativo) dentro de la página (bits menos significativos).

Las **direcciones físicas** se dividen en *número de marco* (marco donde está almacenada la página) y *desplazamiento*.

Cuando la CPU genere una dirección lógica será necesario traducirla a la dirección física correspondiente, la **Tabla de Páginas** mantiene información necesaria para realizar dicha traducción.

Cada proceso tiene asociado una tabla de páginas que, sencillamente, indica el marco de página donde se encuentra almacenada la página correspondiente.

La MMU hace la correspondencia entre dirección virtual y física, y puede contener la tabla de páginas o un registro indicando la posición de memoria donde se encuentra.

En el bloque de control de cada proceso *(PCB)* se guarda la posición donde se encuentra almacenada su correspondietne tabla de páginas para tenerlo en cuenta en los cambios de contexto. EL sistema operativo además mantiene una **tabla de marcos de página**, donde se especifica el proceso y página contenido en cada uno de los marcos y su estado (libre o ocupado).

Implementación de la Tabla de Páginas La tabla de páginas se mantiene en memoria principal, y tienen una entrada por cada página del proceso:

- **Número de marco** en el que está almacenada la página si está en memoria principal.
- **Modo de acceso** autorizado a la página (bits de protección).

El **Registro Base de la Tabla de Páginas *(RBTP)*** apunta a la tabla de páginas (suele almacenarse en el PCB del proceso).

> Problema: Cada acceso a una instrucción o dato requiere **dos accesos a memoria**, uno a la tabla de páginas y otro a memoria.
>
> Solución: Caché hardware de consulta rápida denominada *búfer de traducción adelantada* o TLP *(Translation Look-aside Buffer)*.

El TLB se implementa como un conjunto de registros asociados que permiten una búsqueda en paralelo.

De esta forma, para traducir una dirección:

- Si existe ya en el registro asociado, obtenemos el marco.
- Si no, la buscamos en la tabla de páginas y se actualiza el TLB con esta nueva entrada.

```
dirección física = m * tamaño página + desplazamiento
```

4.2. Segmentación

El programa se considera dividido en sus segmentos (definidos por el programador). La gestión la realiza el sistema operativo, como con las particiones dinámicas, sólo que cada partición no corresponde a un proceso sino a un segmento. El sistema operativo mantiene una tabla-mapa de segmentos, indicando la ubicación en memoria de cada uno de ellos y su tamaño.

Implementación de la 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.

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, s, generado en una dirección lógica, es legal si `s < STLR` (suele almacenarse en el PCB del proceso).

Como el tamaño de segmentos es variable, antes de realizar un acceso a memoria se comprueba que el desplazamiento no supere al tamaño del segmento, para proteger a las zonas de memoria ocupadas por otro segmento.

La segmentación permite que ciertos procesos puedan compartir códico o datos comunes sin necesidad de estar duplicados en memoria principal.

```
dirección física = dirección base + desplazamiento
```