Conceptos de Sistemas Operativos
Modulo 1
Ramiro Cabral

Contents

1 Conceptos Básicos

- **Sistema Operativo:** Software que actúa como intermediario entre el usuario de una computadora y su hardware.

1.1 Objetivos de un SO

- **Comodidad:** Hacer más fácil el uso del hardware.
- **Eficiencia:** Hacer un uso más eficiente de los recursos del sistema.
- **Evolución:** Permitirá la introducción de nuevas funciones al sistema sin interferir con funciones anteriores.

1.2 Perspectiva desde el usuario

- Abstracción con respecto a la arquitectura.
- El SO "oculta" el hardware y presenta a los programas abstracciones más simples de manejar.
- Los programas de aplicación son los clientes del SO.

1.3 Perspectiva desde la administración de recursos

- Administra los recursos de HW de uno o más procesos.
- Provee un conjunto de servicios a los usuarios del sistema.
- Maneja la memoria secundaria y los dispositivos de E/S.
- Ejecución simultánea de procesos.
- Multiplexión en tiempo (CPU) y en espacio (memoria).

1.4 Componentes de un SO

- **Kernel**
- **Shell**
- **Herramientas**

1.4.1 Kernel

- Porción de código que se encuentra en memoria principal y se encarga de la administración de los recursos.
- Implementa servicios esenciales:
  - Manejo de la memoria y la entrada/salida.
  - Manejo de la CPU.
  - Administración de procesos.

1.5 Servicios de un SO

- Administración y planificación del procesador.
- Administración de la memoria.
- Administración del almacenamiento/sistema de archivos.
- Administración de dispositivos.
- Detección de errores y respuestas.
  - Errores de HW internos y externos.
  - Errores de SW.
  - Incapacidad del SO para conceder una solicitud de una aplicación.
- Interacción con el usuario (Shell).
- Telemetría.

1.6 Errores que un SO debe evitar

- Que un proceso se apropie de la CPU.
- Que un proceso intente ejecutar instrucciones de E/S, por ejemplo.
- Que un proceso intente acceder a una dirección de memoria que no le corresponde.

2 Apoyo del Hardware

- Modos de Ejecución: Limitaciones en el conjunto de instrucciones que se pueden ejecutar en cada modo.
- Interrupción de Clock: Se debe evitar que un proceso se apropie de la CPU.
- Protección de la Memoria: Se deben definir límites de memoria a los que puede acceder cada proceso.

2.1 Modos de Ejecución

- Un bit en la CPU indica el modo actual.
- Las instrucciones privilegiadas solo pueden ejecutarse en modo **supervisor/Kernel**.
- En modo **Usuario**, el proceso puede acceder solo a su espacio de direcciones, es decir, a las direcciones "propias".
- El kernel del SO se ejecuta en modo supervisor.
- El resto del SO y los programas de usuario se ejecutan en modo usuario.

2.1.1 Modo Kernel

- **Gestión de procesos:** Creación y terminación, planificación, intercambio, sincronización y soporte para la comunicación entre procesos.
- **Gestión de memoria:** Reserva de espacio de direcciones para los procesos, Swapping, Gestión de Páginas.
- **Gestión E/S:** Gestión de buffers, reserva de canales de E/S y de dispositivos de los procesos.
- **Funciones de soporte:** Gestión de interrupciones, auditoría, monitoreo.
- Cada vez que comienza a ejecutarse un proceso de usuario, el bit de modo se debe poner en modo usuario.
- Cuando hay una trap, el bit de modo se pone en modo Kernel. Esta es la única forma de pasar a modo Kernel.

2.1.2 Modo Usuario

- Debug de procesos, definición de protocolos de comunicación, gestión de aplicaciones.
- Tareas que no requieran accesos privilegiados.
- No se puede interactuar con el hardware.
- Cada proceso trabaja en su propio espacio de direcciones.

2.2 Protección de la E/S

- Las instrucciones de E/S se definen como privilegiadas.
- Deben ejecutarse en Modo Kernel.
- Los procesos de usuario realizan E/S a través de System Calls.

2.3 Protección de la CPU

- Uso de interrupción por clock para evitar que un proceso se apropie de la CPU.
- Las instrucciones que modifican el funcionamiento del reloj son privilegiadas.

2.4 System Calls

- Forma en que los programas de usuario acceden a los servicios del SO.
- Los parámetros asociados a las llamadas pueden pasarse de varias maneras: por registros, bloques, tablas en memoria o la pila.
- Se ejecutan en modo kernel.

3 Procesos

- Programa en ejecución.
- **Programa:**
  - Es estático.
  - No tiene contador de programa.
  - Existe desde que se edita hasta que se borra.
- **Proceso:**
  - Es dinámico.
  - Tiene contador de programa.
  - Su ciclo de vida comprende desde que se solicita ejecutar hasta que termina.

3.1 Atributos de un proceso

- Identificación del proceso y del proceso padre.
- Identificación del usuario que lo disparó.
- Si hay estructura de grupos, grupo que lo disparó.
- En ambientes multiusuario, desde qué terminal y quién lo ejecutó.

3.2 Componentes de un proceso

- Sección de código.
- Sección de Datos.
- Stack(s): Datos temporarios.

3.3 Process Control Block (PCB)

- Estructura de datos asociada al proceso (abstracción).
- Existe una por proceso.
- Es lo primero que se crea cuando se crea un proceso y lo último que se borra cuando termina.
- Contiene la información asociada con cada proceso:
  - PID, PPID, etc.

- Valores de los registros de la CPU.
- Planificación.
- Ubicación en memoria.
- Accounting.
- Entrada/Salida.

3.3.1 Stacks

- Un proceso cuenta con 1 o más stacks.
- Se crean automáticamente y su medida se ajusta en run-time.
- Está formado por stack frames que son pushed (al llamar una rutina) y popped (cuando se retorna de ella).
- El stack frame tiene los parámetros de la rutina y los datos necesarios para recuperar el stack frame anterior.

3.4 Espacio de direcciones de un proceso

- Conjunto de direcciones de memoria que ocupa el proceso.
- No incluye su PCB o tablas asociadas.
- Un proceso en modo usuario solo puede acceder a su espacio de direcciones.
- En modo Kernel, se puede acceder a estructuras internas (PCB del proceso) o a espacio de direcciones de otros procesos.

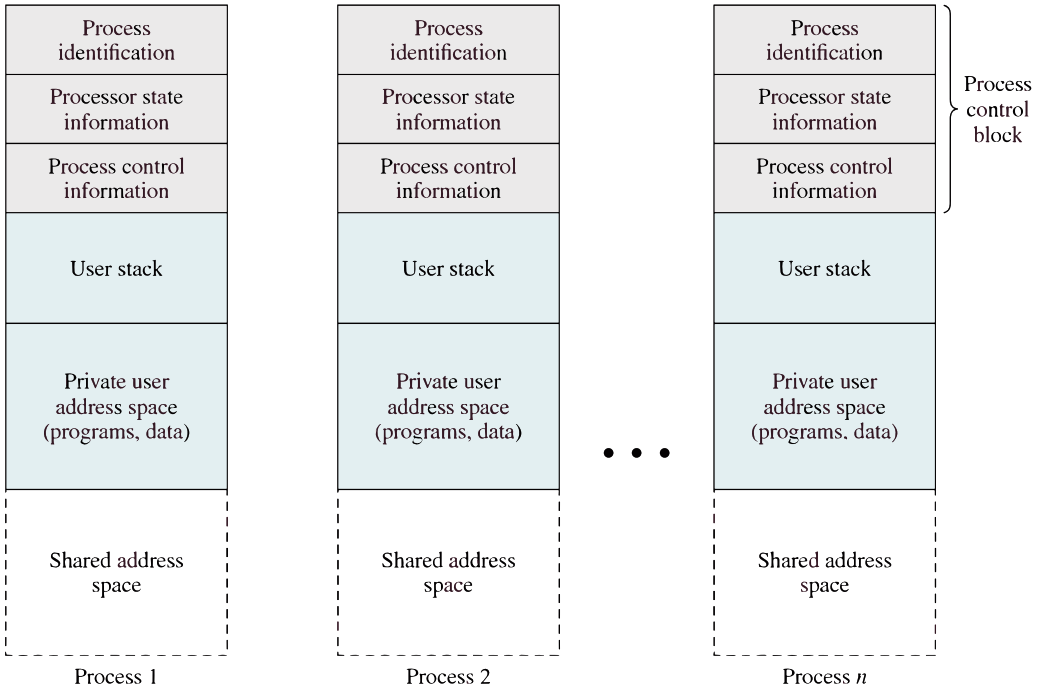

Figure 1: Procesos en Memoria Virtual

3.5 Contexto de un proceso

- Incluye toda la información que el SO necesita para administrar el proceso, y la CPU para ejecutarlo correctamente.
- Son parte del contexto, los registros de CPU, inclusive el contador del programa, prioridad, etc.

3.5.1 Context Switch

- Se produce cuando la CPU cambia de un proceso a otro.
- Se debe resguardar el contexto del proceso saliente, que pasa a espera y retornará después a la CPU.
- Se debe cargar el contexto del nuevo proceso y comenzar desde la instrucción siguiente a la última ejecutada en dicho contexto.
- Es tiempo no productivo de la CPU.
- El tiempo que consume depende del soporte de HW.

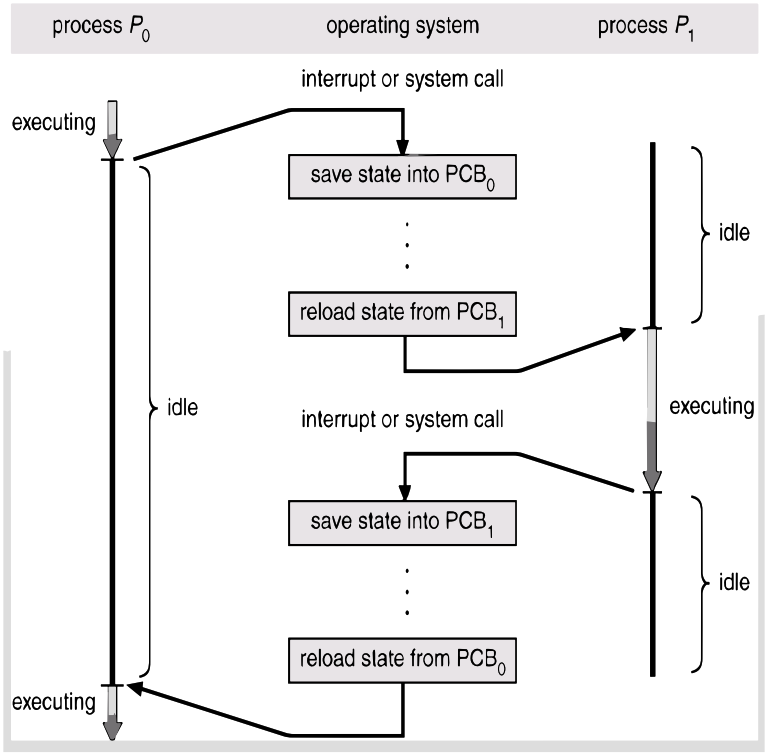

Figure 2: Context Switch

3.6 Ejecución del Kernel

- El kernel es un conjunto de módulos de software.
- Se ejecuta en el procesador como cualquier otro proceso.
- Existen diferentes enfoques de diseño:

3.6.1 El kernel como entidad independiente

- El kernel se ejecuta fuera de todo proceso.
- Cuando un proceso es interrumpido o realiza una System Call, el contexto del proceso se salva y el control se pasa al Kernel del SO.
- El kernel posee su propia región de memoria y su propio Stack.
- Finalizada su actividad, le devuelve el control al proceso.
- El kernel NO es un proceso.
- Se ejecuta como una entidad independiente en modo privilegiado.

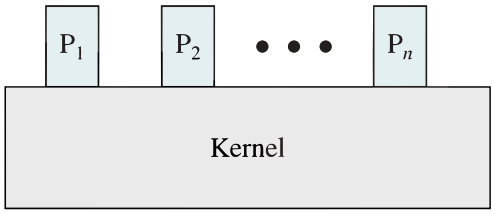

Figure 3: Kernel como entidad independiente

3.6.2 El kernel "dentro" del proceso

- El código del Kernel se encuentra dentro del espacio de direcciones de cada proceso.
- El Kernel se ejecuta en el mismo contexto que algún proceso de usuario.
- El Kernel se puede ver como una colección de rutinas que el proceso utiliza.
- Dentro de un proceso se encuentra el código del programa y el código de los módulos SW del SO (kernel).
- Cada proceso tiene un stack en modo usuario y otro en modo Kernel.
- Cada interrupción es atendida en el contexto del proceso que se encontraba en ejecución (en modo kernel).
- Si el SO determina que el proceso debe seguir ejecutándose luego de atender la interrupción, cambia a modo usuario y devuelve el control.

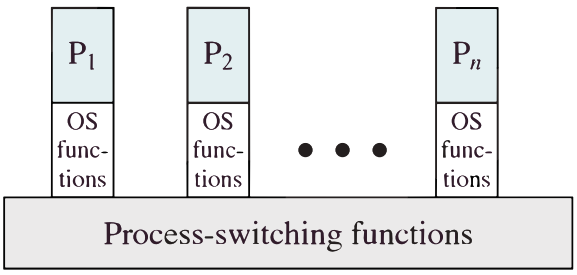

Figure 4: Kernel dentro del proceso

3.7 Estados de un Proceso

En su estado de vida, un proceso pasa por diferentes estados:

- **Nuevo (new)**.
- **Listo (ready)**.
- **Ejecución (running)**.
- **En espera/bloqueado (waiting/blocked)**.
- **Terminado (terminated)**.

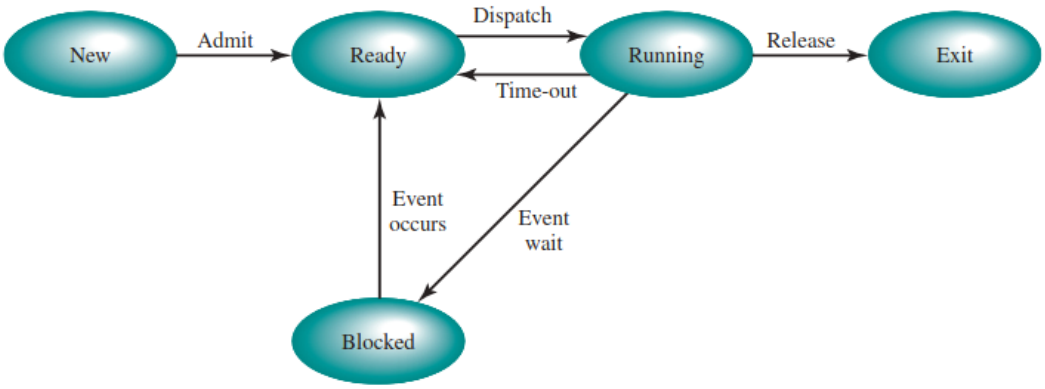

Figure 5: Estados de un proceso

3.8 Colas en la planificación de procesos

- Para realizar la planificación, el SO utiliza la PCB de cada proceso como una abstracción del mismo.
- Las PCB se enlazan en colas siguiendo un orden determinado.

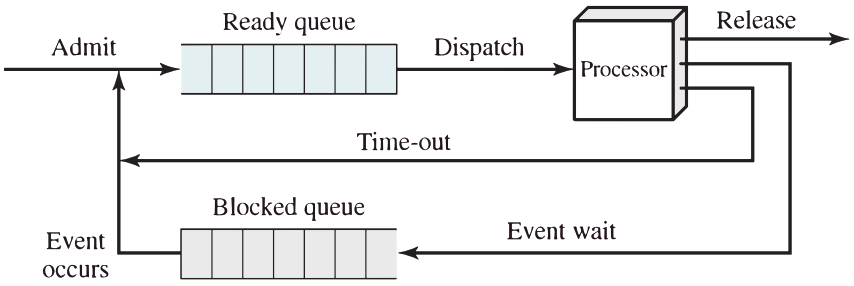

Figure 6: Colas de Planificación

3.8.1 Módulos de la planificación

- Son módulos de SW del Kernel que realizan distintas tareas asociadas a la planificación.
- Se ejecutan ante determinados eventos:
  - Creación/Terminación de procesos.
  - Eventos de sincronización.
  - Finalización de lapso de tiempo.
  - Etc.
- Existen 3 schedulers:
  - **Long Term Scheduler**
  - **Short Term Scheduler**
  - **Medium Term Scheduler**
- **Dispatcher:** realiza el cambio de contexto, cambio de modo de ejecución y despacha el proceso elegido por el Short Term.
- **Loader** carga en memoria el proceso elegido por el long term.

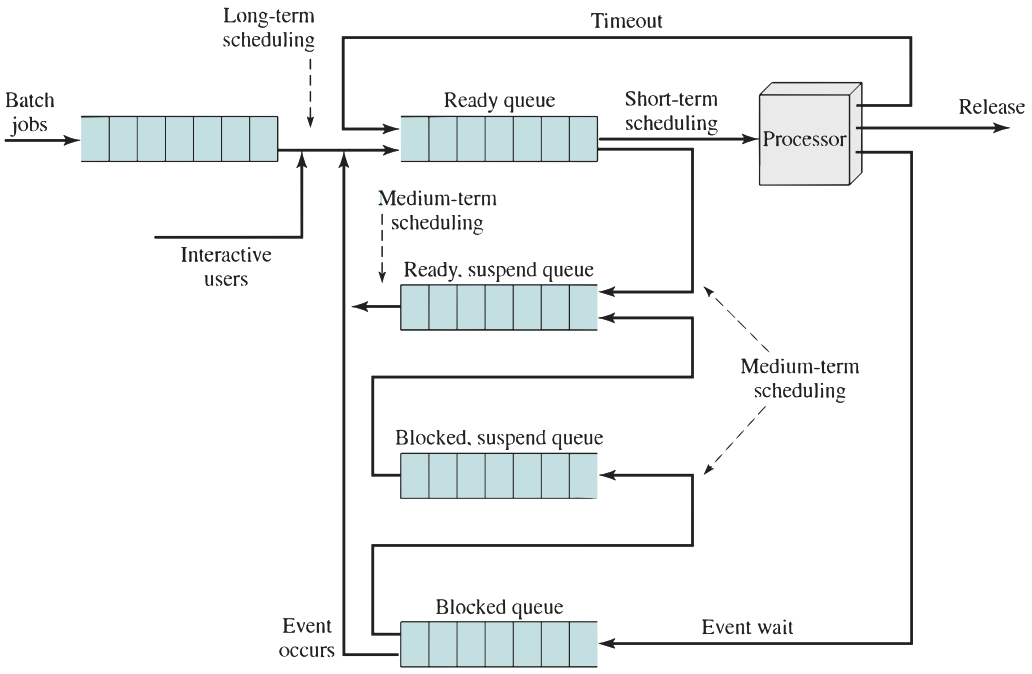

Figure 7: Funcionamiento de los Schedulers

3.8.2 Schedulers

- **Long Term Scheduler**
  - Controla el grado de multiprogramación.
  - Puede no existir este scheduler y absorber esta tarea el de short term.
- **Medium Term Scheduler**
  - Si es necesario, reduce el grado de multiprogramación.
  - Saca temporalmente de memoria los procesos que sean necesarios para mantener el equilibrio del sistema.
- **Short Term Scheduler**
  - Decide a cuál de los procesos en la cola de listos se elige para que use la CPU.
  - Términos asociados: apropiativo, no apropiativo, algoritmo de scheduling.

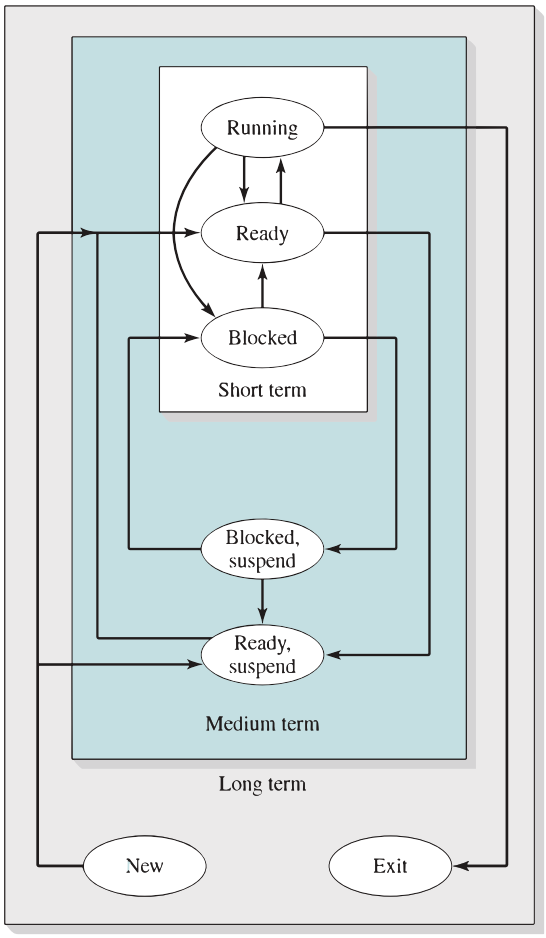

Figure 8: Schedulers

3.9 Estados de los procesos

- **Nuevo (new):**
  - Un usuario "dispara" el proceso. Un proceso es creado por otro proceso: su proceso padre.
  - En este estado, se crean las estructuras asociadas, y el proceso queda en la cola de procesos, normalmente en espera de ser cargado en memoria.
- **Listo (ready):**
  - Luego de que el Long Term Scheduler elige al proceso para cargarlo en memoria, el proceso queda en estado listo.
  - El proceso solo necesita que se le asigne CPU.

  - Está en la cola de procesos listos (ready queue).

- **Ejecución (running):**
  - El Long Term Scheduler lo eligió para asignarle CPU.
  - Tendrá la CPU hasta que se termine el período de tiempo asignado, termine o hasta que necesite realizar alguna operación de E/S.

- **En espera/bloqueado (waiting/blocked):**
  - El proceso necesita que se cumpla el evento esperado para continuar.
  - El evento puede ser la terminación de una E/S solicitada, o la llegada de una señal por parte de otro proceso.
  - Sigue en memoria, pero no tiene la CPU.
  - Sigue en memoria, pero no tiene la CPU.

3.10 Comportamiento de los procesos

- **CPU-bound:** mayor parte del tiempo utilizando la CPU.
- **I/O bound:** mayor parte del tiempo esperando por I/O.

3.11 Algoritmos de Planificación

- **Planificación:** necesidad de determinar cuál de todos los procesos que están listos para ejecutarse, será el próximo en ejecutarse.
- **Algoritmos Preemptive:** existen situaciones que hacen que el proceso en ejecución sea expulsado de la CPU.
- **Algoritmos No Preemptive:** los procesos se ejecutan hasta que el mismo (por su propia cuenta) abandone la CPU.

3.12 Algoritmos según el tipo de proceso

3.12.1 Procesos Batch

- No existen usuarios que esperen una respuesta en la terminal.
- Se pueden utilizar algoritmos no apropiativos.
- Metas propias de este tipo de algoritmos:
  - Rendimiento: Maximizar el número de trabajos por hora.

  - Tiempo de Retorno: Minimizar los tiempos entre el comienzo y la finalización.
  - El tiempo de espera se puede ver afectado.
  - Uso de la CPU: Mantener la CPU ocupada la mayor cantidad de tiempo posible.

3.12.2 Procesos Interactivos

- No solo interacción con los usuarios.
- Son necesarios algoritmos apropiativos para evitar que un proceso acapare la CPU.
- Metas propias de este tipo de algoritmos:
  - Tiempo de respuesta: Responder a peticiones con rapidez.
  - Proporcionalidad: Cumplir con las expectativas de los usuarios. Por ejemplo, al poner STOP al reproductor de música, debe dejar de ser reproducida en un tiempo corto.

3.13 Creación de procesos

- Un proceso es creado por otro proceso.
- Un proceso padre tiene uno o más procesos hijos.
- Se forma un árbol de procesos.
- Actividades en la creación:
  - Crear la PCB.
  - Asignar PID único.
  - Asignarle memoria para regiones.
  - Crear estructuras de datos asociadas.

3.13.1 Relación entre procesos padre e hijo

- El padre puede continuar ejecutándose concurrentemente con su hijo.
- El padre puede esperar a que el/los procesos hijos terminen para continuar la ejecución.

4 Memoria

4.1 Rol del Sistema Operativo

El SO debe:

- Llevar un registro de las partes de memoria que se están utilizando y de aquellas que no.
- Asignar espacio en memoria principal a los procesos cuando estos lo necesitan.
- Liberar espacio de memoria asignada a procesos que han terminado.
- Lograr que el programador se abstraiga de la asignación de los programas.
- Brindar seguridad entre los procesos para que unos no accedan a secciones privadas de otros.
- Brindar la posibilidad de acceso compartido a determinadas secciones de la memoria.
- Garantizar el rendimiento.

4.2 Requisitos

- **Reubicación**
  - El programador no debe ocuparse de conocer dónde será colocado en la memoria RAM.
  - Mientras un proceso se ejecuta, puede ser sacado y traído a la memoria (swap) y, posiblemente, colocarse en diferentes direcciones.
  - Las referencias a la memoria se deben "traducir" según la ubicación actual del proceso.
- **Protección**
  - Los procesos NO deben referenciar/acceder a direcciones de memoria de otros procesos (salvo que tengan permiso).
  - El chequeo se debe realizar durante la ejecución.
- **Compartición**
  - Permitir que varios procesos accedan a la misma porción de memoria.
  - Permitir un mejor uso de la memoria principal, evitando copias innecesarias (repetidas) de instrucciones.

4.3 Direcciones

4.3.1 Espacio de direcciones

- Rango de direcciones (a memoria) posibles que un proceso puede utilizar para direccionar sus instrucciones y datos.
- Es independiente de la ubicación "real" del proceso en la memoria RAM.

4.4 Tipos de direcciones

- **Lógicas/Virtuales**
  - Referencia a una localidad de memoria.
  - Representa una dirección en el "Espacio de Direcciones del Proceso".

  Físicas
  - Referencia una localidad en la memoria principal.

Es necesario algún tipo de conversión de direcciones lógicas a físicas y viceversa.

4.4.1 Conversión de Direcciones

Una forma simple de realizar la conversión es utilizando registros auxiliares.

- **Registro Base:** dirección de comienzo del Espacio de Direcciones del proceso en la memoria principal.
- **Registro Límite** dirección final del proceso o medida del proceso.
- Ambos valores se fijan cuando el espacio de direcciones del proceso es cargado a memoria.
- Varían entre procesos.

Si la conversión se realiza en tiempo de ejecución, las direcciones lógicas se denominan **direcciones virtuales**, y son diferentes a las físicas. En este caso, el mapeo entre ambos tipos de direcciones se realiza por hardware, mediante la **Memory Management Unit (MMU)**.

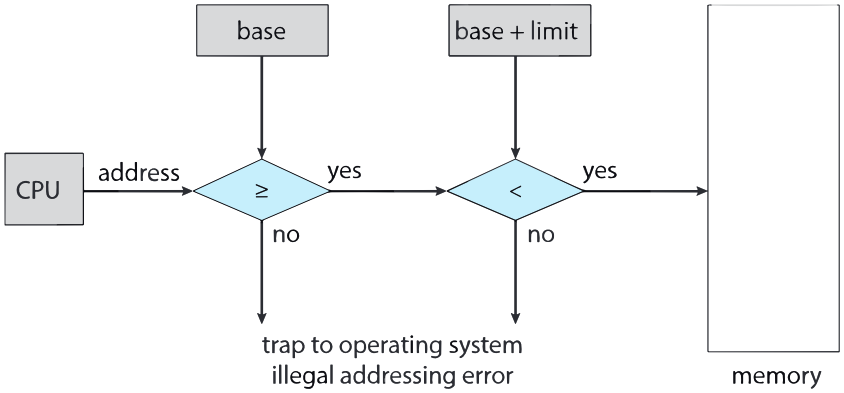

Figure 9: Protección de direcciones mediante registros base y límite.

4.5 Memory Management Unit (MMU)

- Es un dispositivo de hardware que mapea direcciones virtuales a físicas.
- Es parte de la CPU.
- Reprogramarla es una operación privilegiada.
- El valor en el "registro de realocación" es sumado a cada dirección generada por el proceso de usuario al momento de acceder a la memoria.
- Los procesos solo utilizan direcciones virtuales.

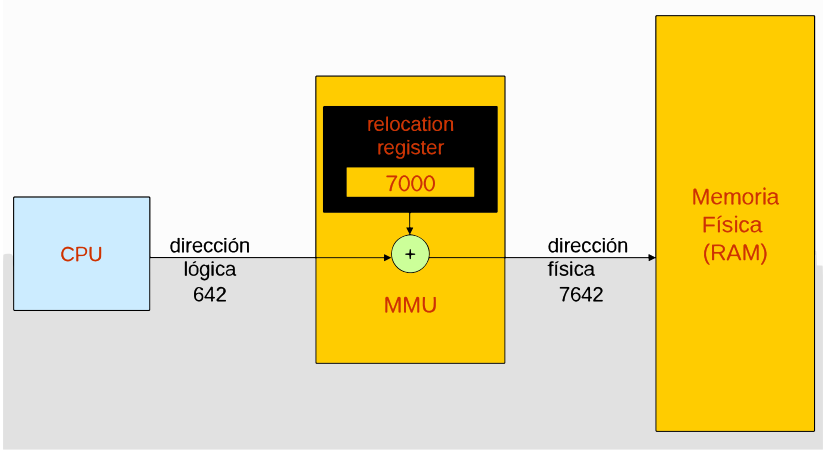

Figure 10: Funcionamiento de la MMU

4.6 Mecanismos de asignación de memoria

- **Particiones Fijas**
  - La memoria se divide en particiones o regiones de tamaño fijo (del mismo tamaño o no).
  - Alojan un proceso cada una.
  - Cada proceso se coloca de acuerdo a algún criterio (worst-fit, best-fit, etc).
- **Particiones Dinámicas**
  - Las particiones varían en tamaño y número.
  - Alojan un proceso cada una.
  - Cada partición se genera en forma dinámica, del tamaño exacto que necesita el proceso.

4.7 Fragmentación

La **fragmentación** se produce cuando una alocalidad de memoria no puede ser utilizada por no encontrarse en forma continua. Existen dos tipos:

- **Fragmentación Interna**
  - Se produce en el esquema de particiones fijas.
  - Es la porción de la partición que queda sin utilizar.
- **Fragmentación Externa**
  - Se produce en el esquema de particiones dinámicas.
  - Son huecos que van quedando en la memoria a medida que los procesos finalizan.
  - Al no encontrarse en forma contigua, puede darse el caso de que tengamos memoria libre para alojar un proceso, pero que no la podamos utilizar.
  - Puede solucionarse utilizando la **compactación**.

4.8 Paginación

- La memoria física es dividida lógicamente en pequeños trozos de igual tamaño llamados **Marcos**.
- La memoria lógica (espacio de direcciones) es dividida en trozos de igual tamaño que los marcos, llamados **Páginas**.

- El SO debe mantener una tabla de páginas por cada proceso, donde cada entrada contiene (entre otras), el Marco en el que se coloca cada página.
- La dirección lógica se interpreta como un número de página y un desplazamiento dentro de la misma.

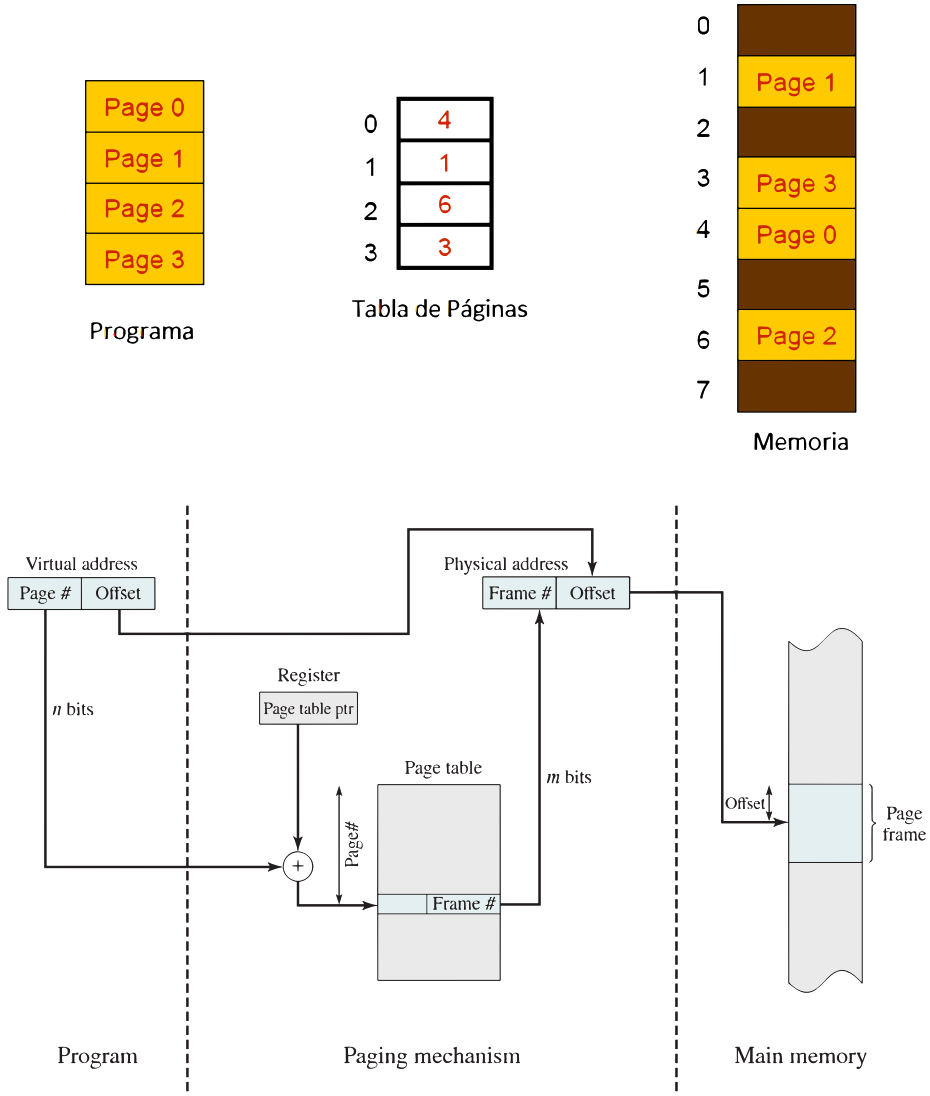

Figure 11: Ejemplo de Paginación

4.9 Segmentación

- Esquema que se asemeja a la "visión del usuario". El programa se divide en partes/secciones.
- Un programa es una colección de segmentos. Un segmento es una unidad lógica, como Programa Principal, Procedimientos y Funciones, variables locales/globales, etc.
- Puede causar fragmentación.
- Todos los segmentos de un programa pueden no tener el mismo tamaño.
- Las direcciones lógicas consisten en 2 partes:
  - Selector de Segmento.
  - Desplazamiento dentro del segmento.
- **Tabla de Segmentos:** permite mapear la dirección lógica a física. Cada entrada contiene:
  - Base: Dirección física del comienzo del segmento.
  - Límite: Tamaño del segmento.
- Segment-Table base register (STBR): contiene la dirección de la tabla de segmentos.
- Segment-Table length register (STLR): contiene la cantidad de segmentos de un programa.

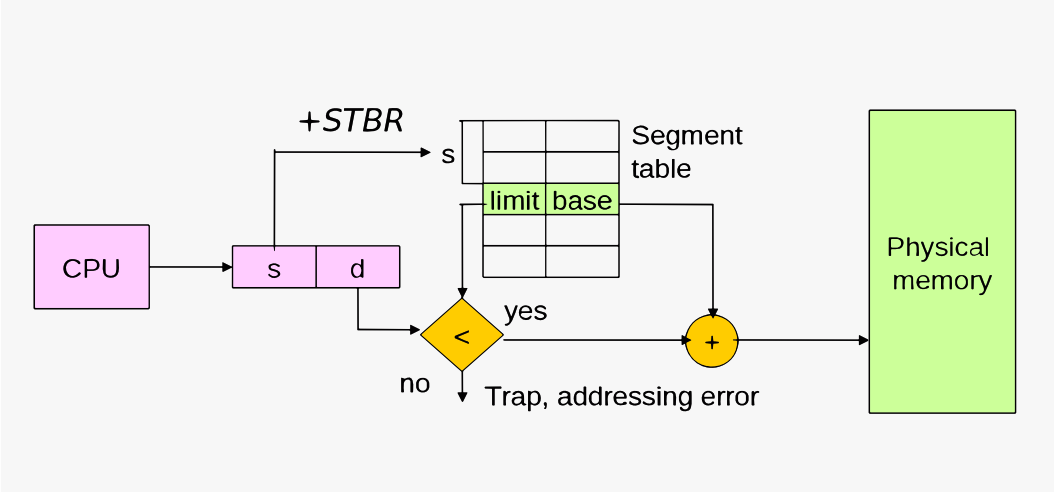

Figure 12: Traducción de direcciones en un sistema con Segmentación

4.9.1 Segmentación Paginada

- La paginación:
  - Transparente al programador.
  - Elimina Fragmentación Externa.
- La segmentación:
  - Es visible al programador.
  - Facilita modularidad, estructuras de datos grandes y da mejor soporte a la compartición y protección.
- **Segmentación Paginada:** cada segmento es dividido en páginas de tamaño fijo.

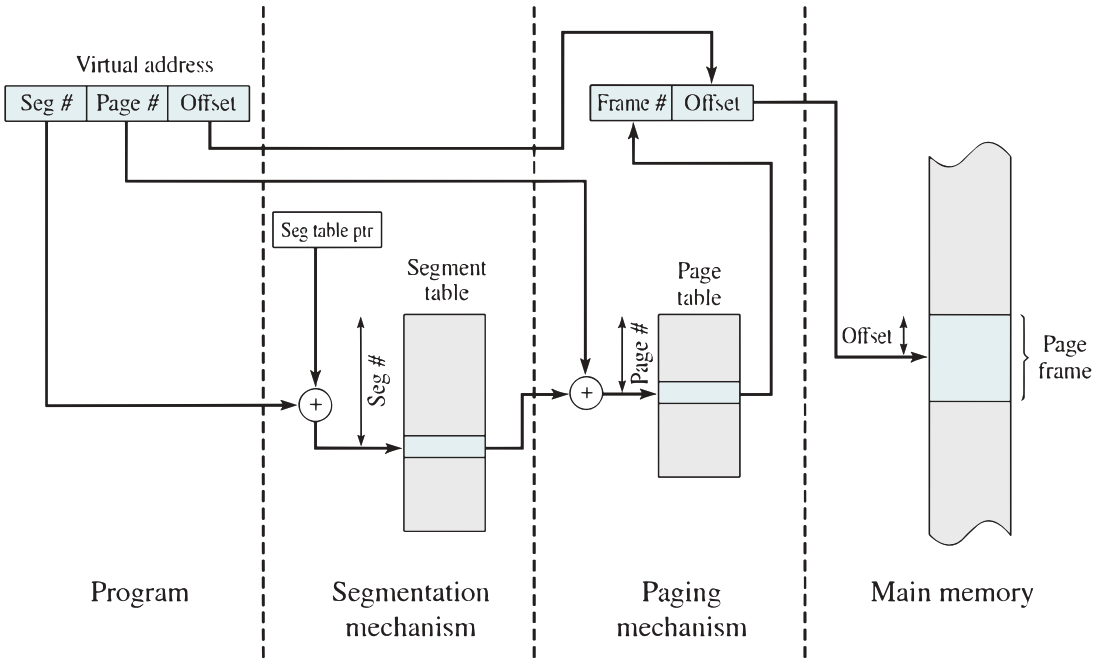

Figure 13: Traducción de direcciones en un sistema con Segmentacion Paginada