Tema 2. Introducción a los Sistemas Operativos

Contenidos

2.1 Componentes de un Sistema Operativo (SO) multiprogramado.
- 2.1.1 Sistemas multiprogramados y de tiempo compartido.
- 2.1.2 Concepto de proceso.
- 2.1.3 Modelo de cinco estados de los procesos.

2.2 Descripción y control de procesos.
- 2.2.1 Bloque de control de proceso (PCB).
- 2.2.2 Control de procesos.

2.3 Hebras (hilos).

2.4 Gestión básica de memoria.
- 2.4.1 Paginación
- 2.4.2 Segmentación

Objetivos

- Conocer los elementos necesarios para implementar la multiprogramación en un sistema operativo (SO).
- Conocer el concepto de proceso y el modelo de cinco estados de los procesos.
- Conocer el uso que realiza el SO del apoyo hardware e integrarlo en el modelo de cinco estados.
- Conocer el concepto de hebra (hilo), su modelo de cinco estados y su utilidad.
- Conocer la gestión básica de memoria que realiza el SO.

Bibliografía básica

[Stal05] W. Stallings, **Sistemas Operativos, Aspectos Internos y Principios de Diseño (5ª Edición)**. Pearson Education, 2005

[Carr07] J. Carretero, F. García, P. de Miguel, F. Pérez, **Sistemas Operativos (2ª Edición)**, McGraw-Hill, 2007

2.1 Componentes de un SO multiprogramado

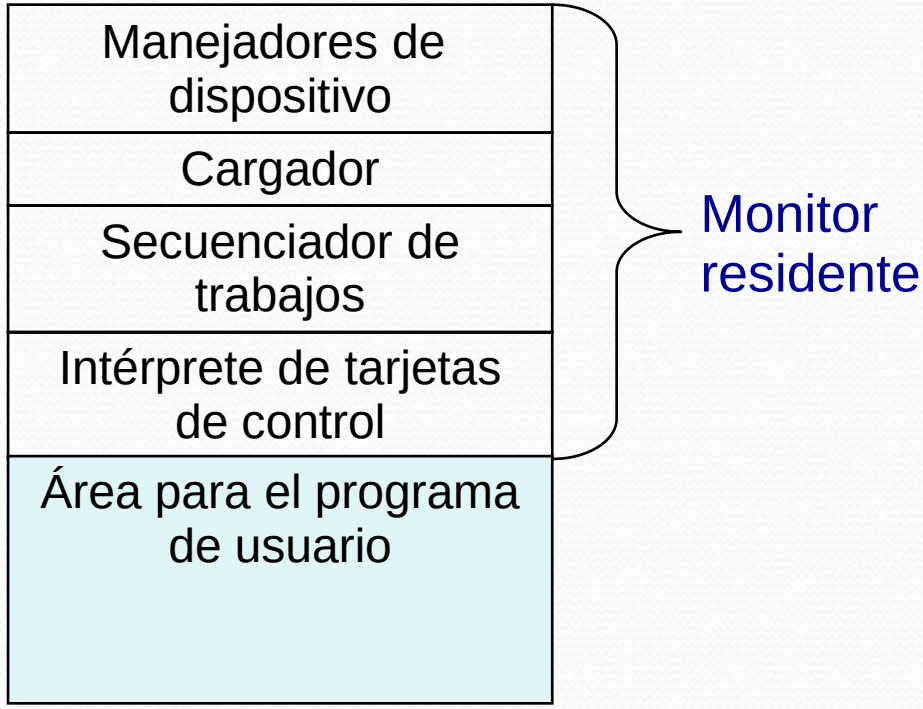
2.1.1 Concepto de multiprogramación [Stall05] (pp. 58-67)

Procesamiento en Serie:

- El programador interactúa directamente con la máquina, no existe S.O.
- Problemas: Baja utilización del tiempo de CPU, Tiempo de planificación alto.

Sistemas por lotes (Sistemas Batch):

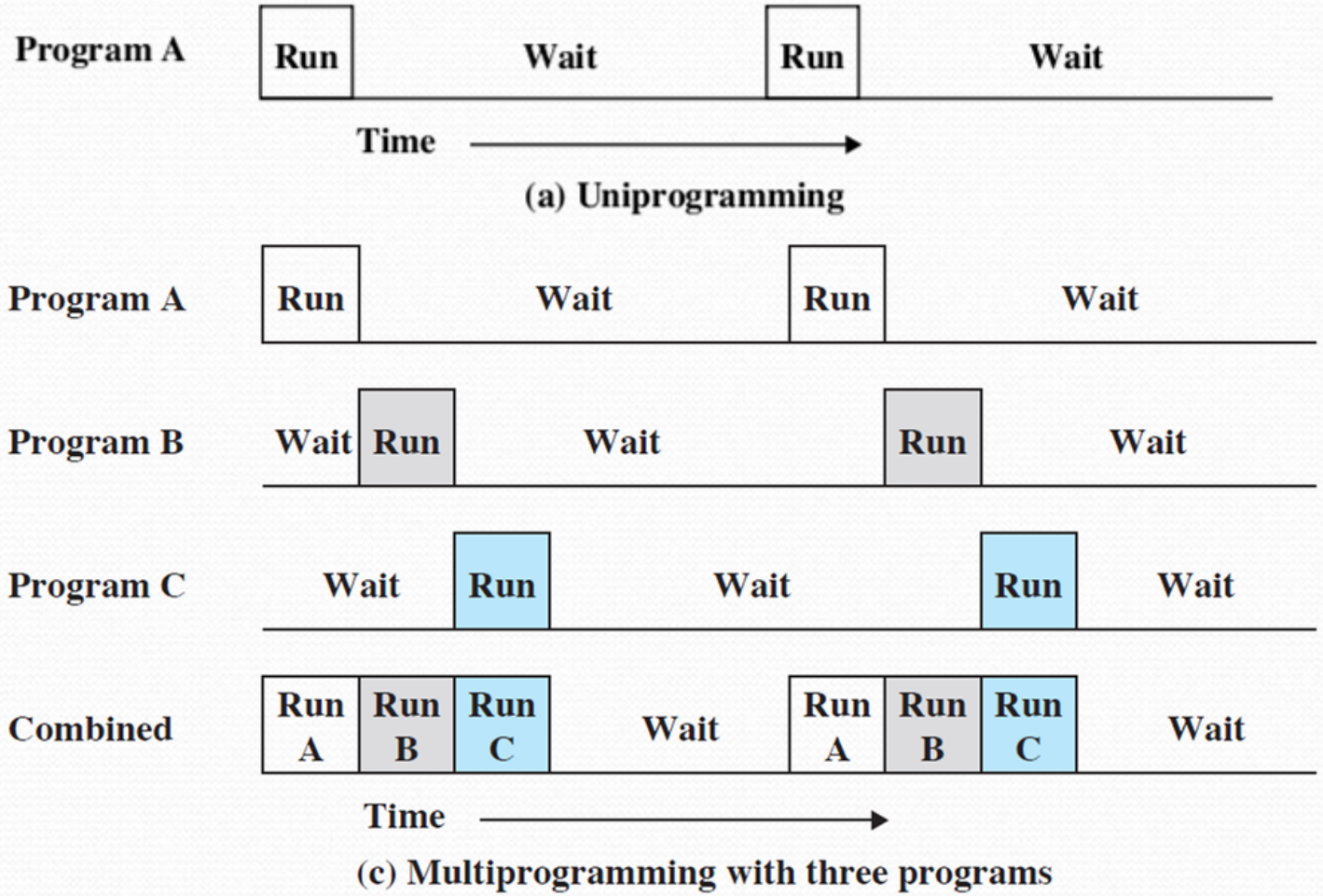
- Agrupa trabajos similares: reduce el tiempo de planificación.
- Trabajo = (programa + datos + ordenes de control para el sistema)
- Falta de interacción entre el usuario y el computador mientras se ejecuta su trabajo.
- Principal problema: CPU ociosa durante las E/S. *Solución:* Spooling: Superpone la E/S de un trabajo al cómputo de otros.



- Control inicial en el Monitor
- Transferencia del control al trabajo
- Al finalizar el trabajo, el control vuelve al monitor

2.1 Componentes de un SO multiprogramado

2.1.1 Concepto de multiprogramación



2.1 Componentes de un SO multiprogramado

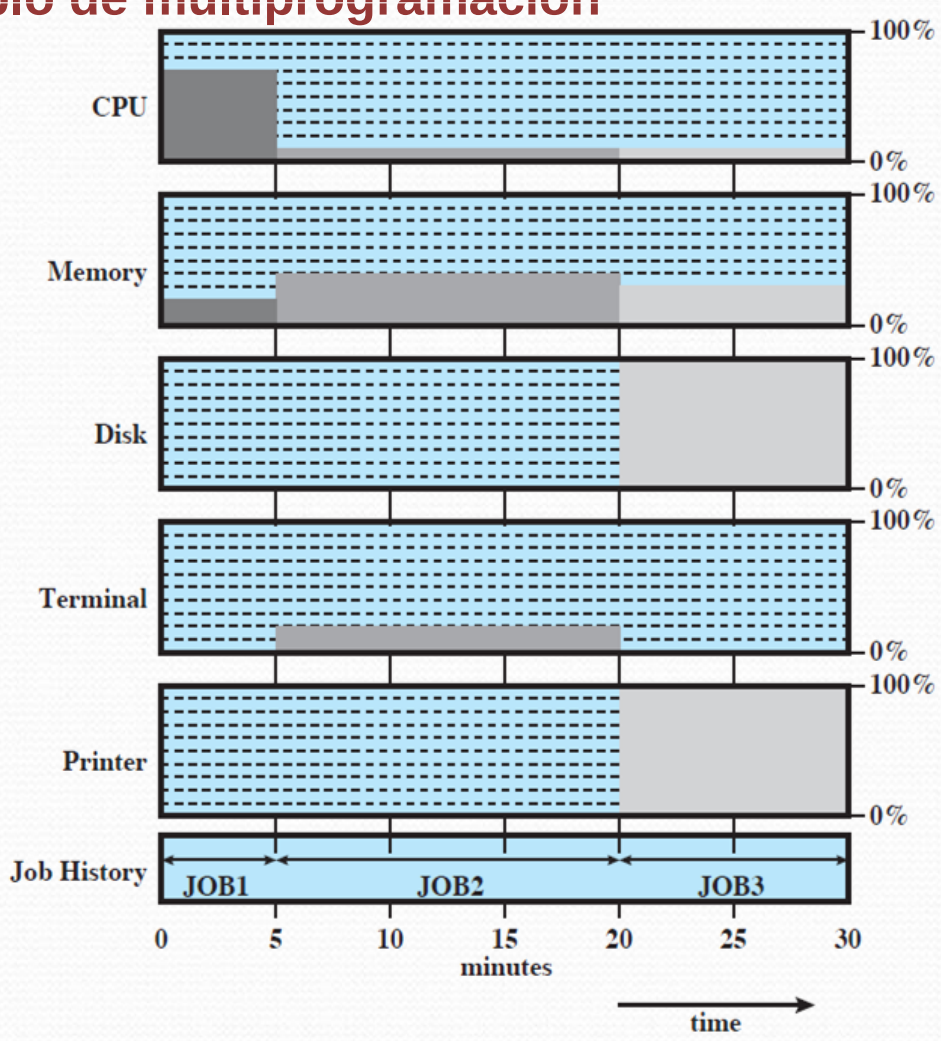
2.1.1 Concepto de multiprogramación

	Trabajo 1	Trabajo 2	Trabajo 3
Tipo de trabajo	Computación pesada	Gran cantidad de E/S	Gran cantidad de E/S
Duración	5 minutos	15 minutos	10 minutos
Memoria requerida	50 MB	100 MB	75 MB
¿Necesita disco?	NO	NO	SI
¿Necesita terminal?	NO	SI	NO
¿Necesita impresora?	NO	NO	SI

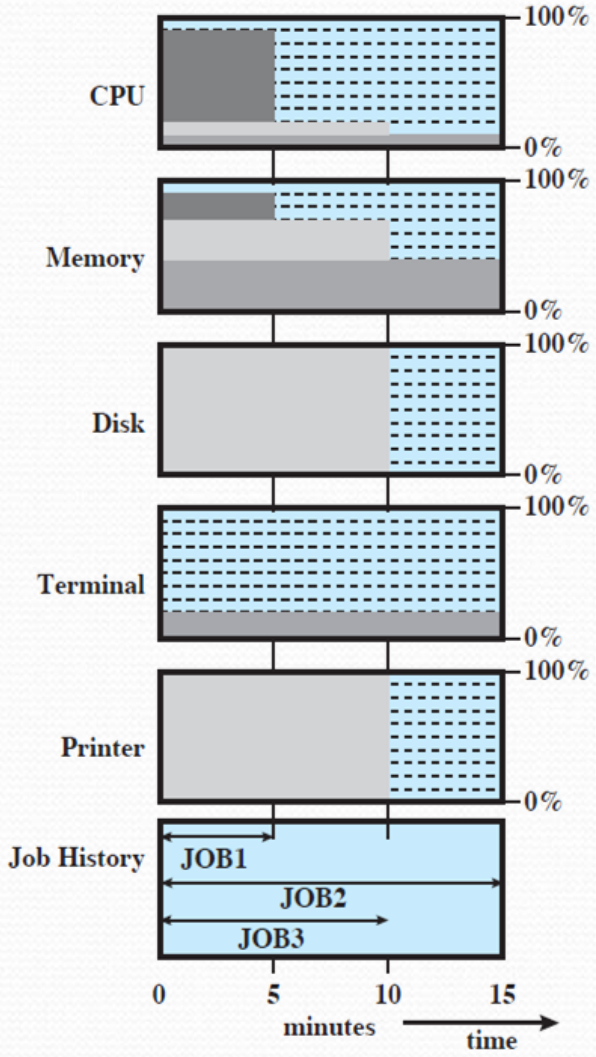
- Trabajo1 utiliza mucho la CPU, Trabajo2 y Trabajo3 utilizan mucho los periféricos de E/S.

2.1 Componentes de un SO multiprogramado

Ejemplo de multiprogramación



(a) Uniprogramming



(b) Multiprogramming

Ejemplo de multiprogramación

	Monoprogramación	Multiprogramación
Uso del procesador	20%	40%
Uso de memoria	33%	67%
Uso de disco	33%	67%
Uso de impresora	33%	67%
Tiempo transcurrido	30 minutos	15 minutos
Productividad	6 trabajos/hora	12 trabajos/hora
Tiempo de respuesta medio	18 minutos	10 minutos

2.1 Componentes de un SO multiprogramado

Definiciones

S.O. multiprogramado: S.O. que permite que se ejecute más de un proceso simutáneamente y cuyos datos e instrucciones se encuentran en memoria principal.

S.O. monousuario: Proporciona servicios a un único usuario.

S.O. multiusuario: Proporciona servicios a varios usuarios simultáneamente.

S.O. monoprocesador: S.O. que gestiona un sistema de computación de un único procesador.

S.O. multiprocesador: S.O. que gestiona un sistema de computación de varios procesadores.

S.O. de tiempo compartido: S.O. 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.

2.1 Componentes de un SO multiprogramado

Algunas cuestiones interesantes …

- ¿Un S.O. multiprogramado es un S.O. de tiempo compartido? ¿y al contrario?
- ¿Un S.O. de tiempo compartido tiene que ser multiusuario? ¿y monousuario?
- ¿Un S.O. monoprocesador tiene que ser monousuario? ¿y multiusuario?
- ¿Un S.O. multiprocesador tiene que ser monousuario? ¿y multiusuario?

2.1.2 Concepto de proceso [Stal05] (pp. 68-71)

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

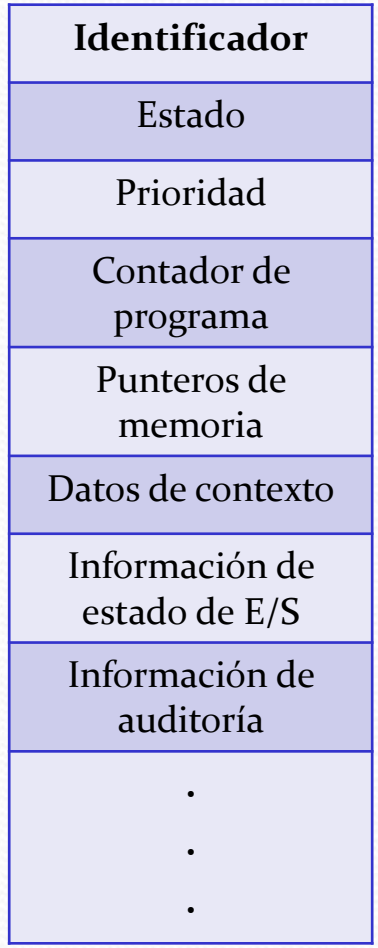
- Un **proceso** está formado por:
  - ❑ Un **programa ejecutable**.
  - ❑ **Datos** que necesita el **SO** para **ejecutar** el **programa**.

2.1 Componentes de un SO multiprogramado

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

[Stal05] (pp. 108-114)

- **Identificador de proceso**, (PID, del inglés Process IDentificator).
- **Contexto de ejecución**: Contenido de los registros del procesador.
- **Memoria** donde reside el programa y sus datos.
- **Información** relacionada con recursos del sistema.
- **Estado:** En que situación se encuentra el proceso en cada momento (modelo de estados).
- **Otra información.**

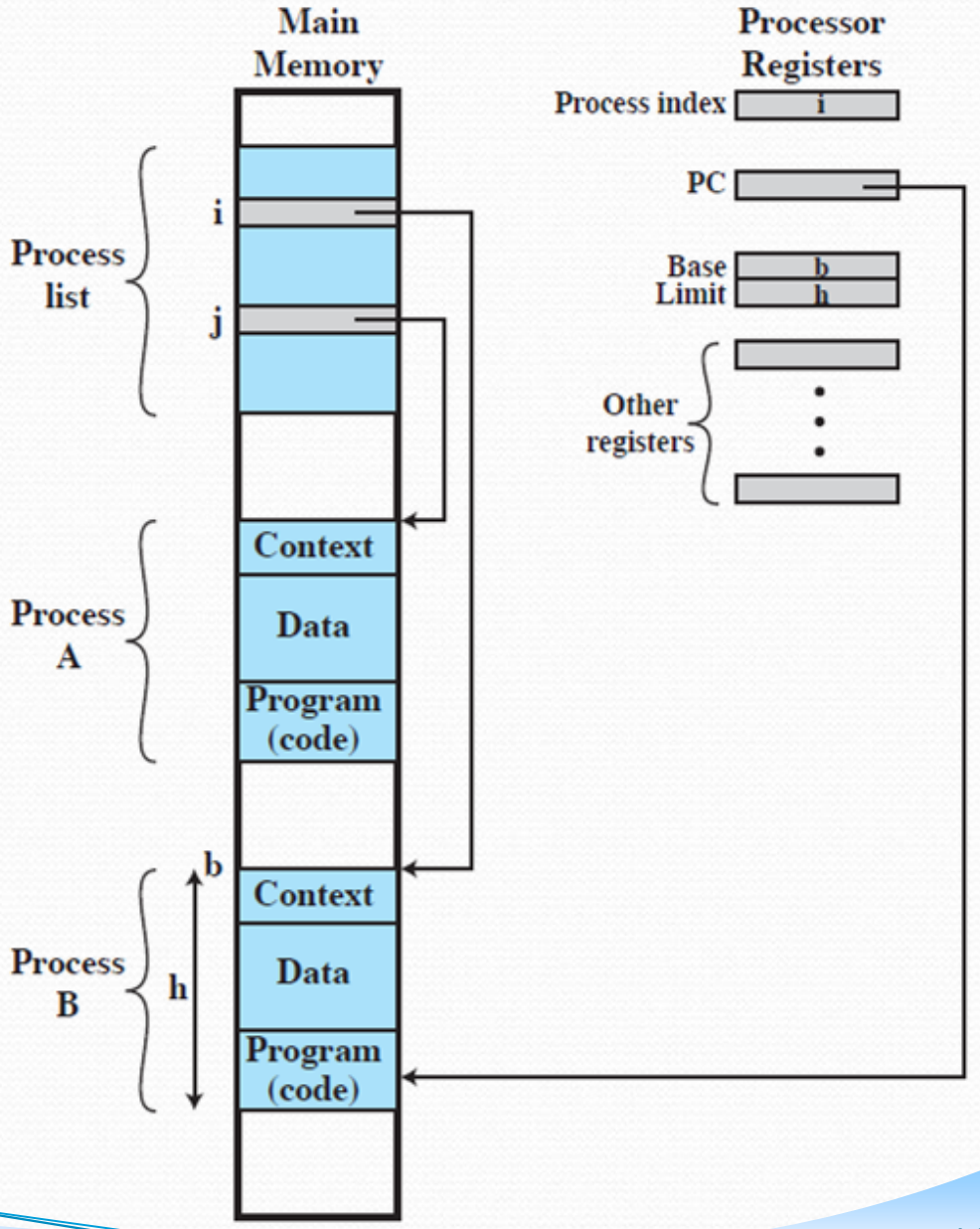


2.1 Componentes de un SO multiprogramado

Implementación de Procesos Típica

[Stal05] (Fig. 2.8, p. 70)

- Esta implementación permite ver al **proceso** como una **estructura de datos**.
- El **estado** completo del proceso en un instante dado se almacena en su **PCB** (*Context* en la figura).
- Esta estructura permite el desarrollo de técnicas potentes que aseguren la **coordinación** y la **cooperación** entre los **procesos**.



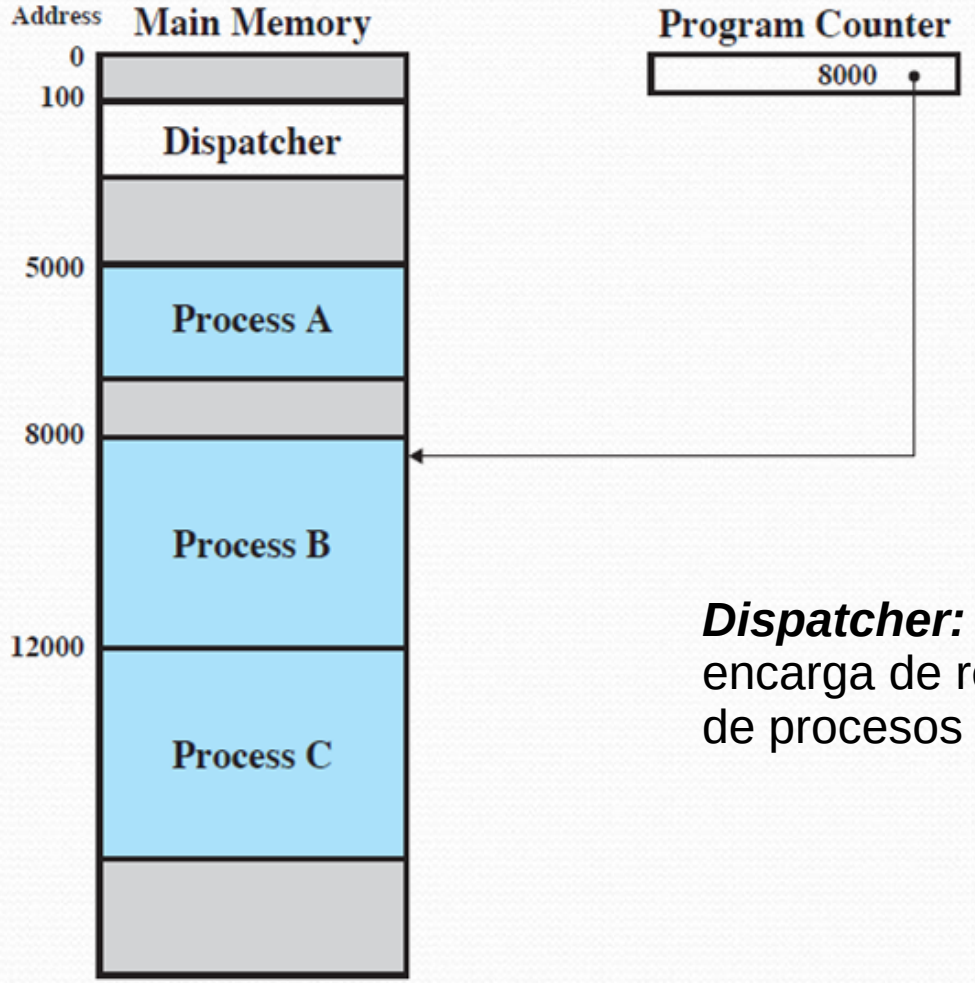
Concepto de traza de ejecución

- 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?

2.1 Componentes de un SO multiprogramado

Ejemplo de traza de ejecución



Dispatcher: Módulo del SO que se encarga de realizar el intercambio de procesos en el procesador.

2.1 Componentes de un SO multiprogramado

Ejemplo de traza de ejecución

(a) Trace of Process A	(b) Trace of Process B	(c) Trace of Process C
5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

5000 = Starting address of program of Process A
8000 = Starting address of program of Process B
12000 = Starting address of program of Process C

100 = Starting address of dispatcher program

Shaded areas indicate execution of dispatcher process;
first and third columns count instruction cycles;
second and fourth columns show address of instruction being executed

1	5000	27	12004
2	5001	28	12005
3	5002	----------	Timeout
4	5003	29	100
5	5004	30	101
6	5005	31	102
----------	Timeout	32	103
7	100	33	104
8	101	34	105
9	102	35	5006
10	103	36	5007
11	104	37	5008
12	105	38	5009
13	8000	39	5010
14	8001	40	5011
15	8002	----------	Timeout
16	8003	41	100
----------	I/O Request	42	101
17	100	43	102
18	101	44	103
19	102	45	104
20	103	46	105
21	104	47	12006
22	105	48	12007
23	12000	49	12008
24	12001	50	12009
25	12002	51	12010
26	12003	52	12011
		----------	Timeout

Llamadas al Sistema [Carr07] (pp. 114-115)

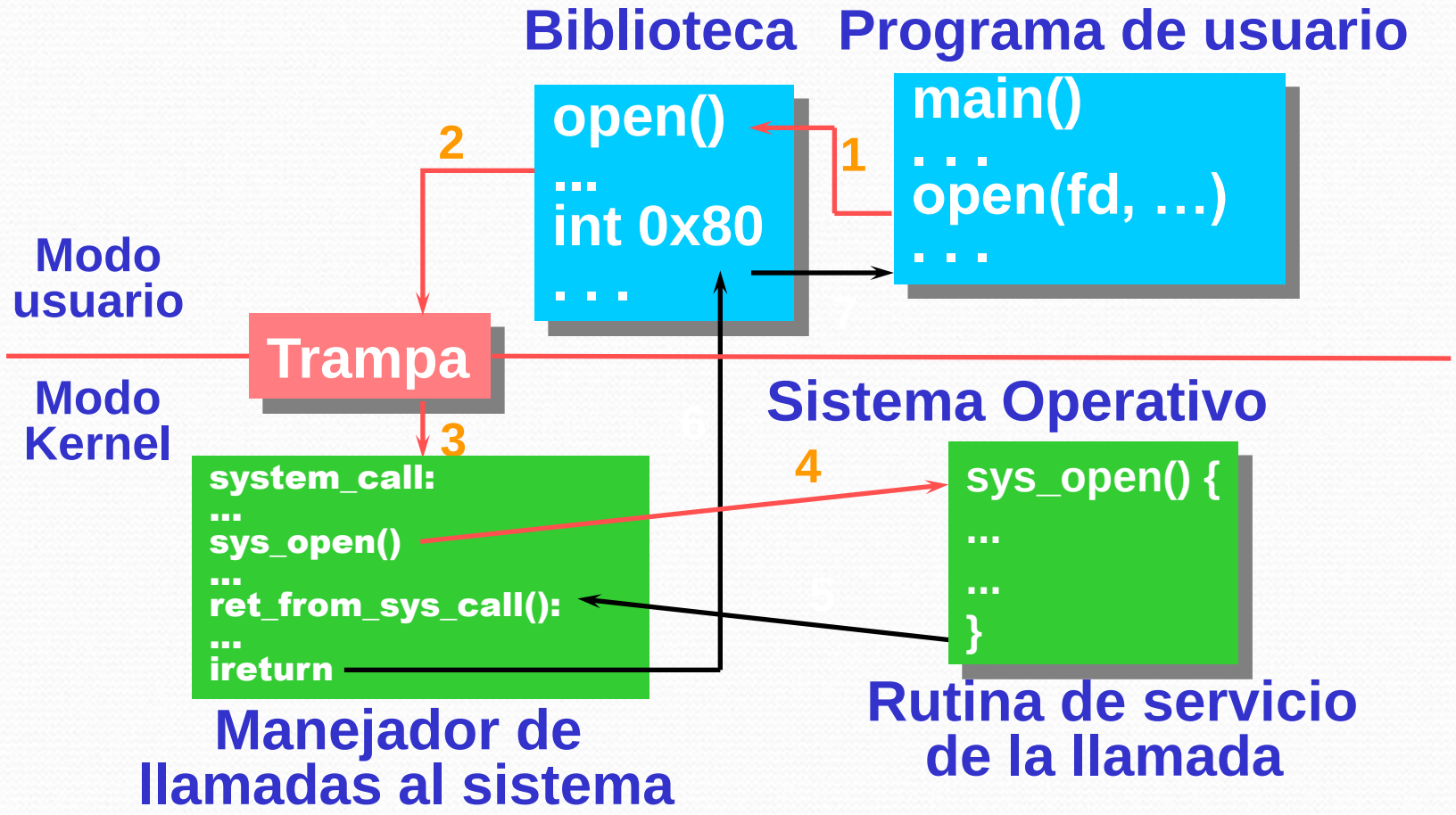
- Es la forma en la que se comunican los programas de usuario con el SO en tiempo de ejecución.
- Son peticiones de servicio que se hace el proceso al SO.

- Ejemplos de llamadas al sistema:
  - Solicitudes de E/S.
  - Gestión de procesos.
  - Gestión de memoria.

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

2.1 Componentes de un SO multiprogramado

Pasos realizados durante una llamada al Sistema

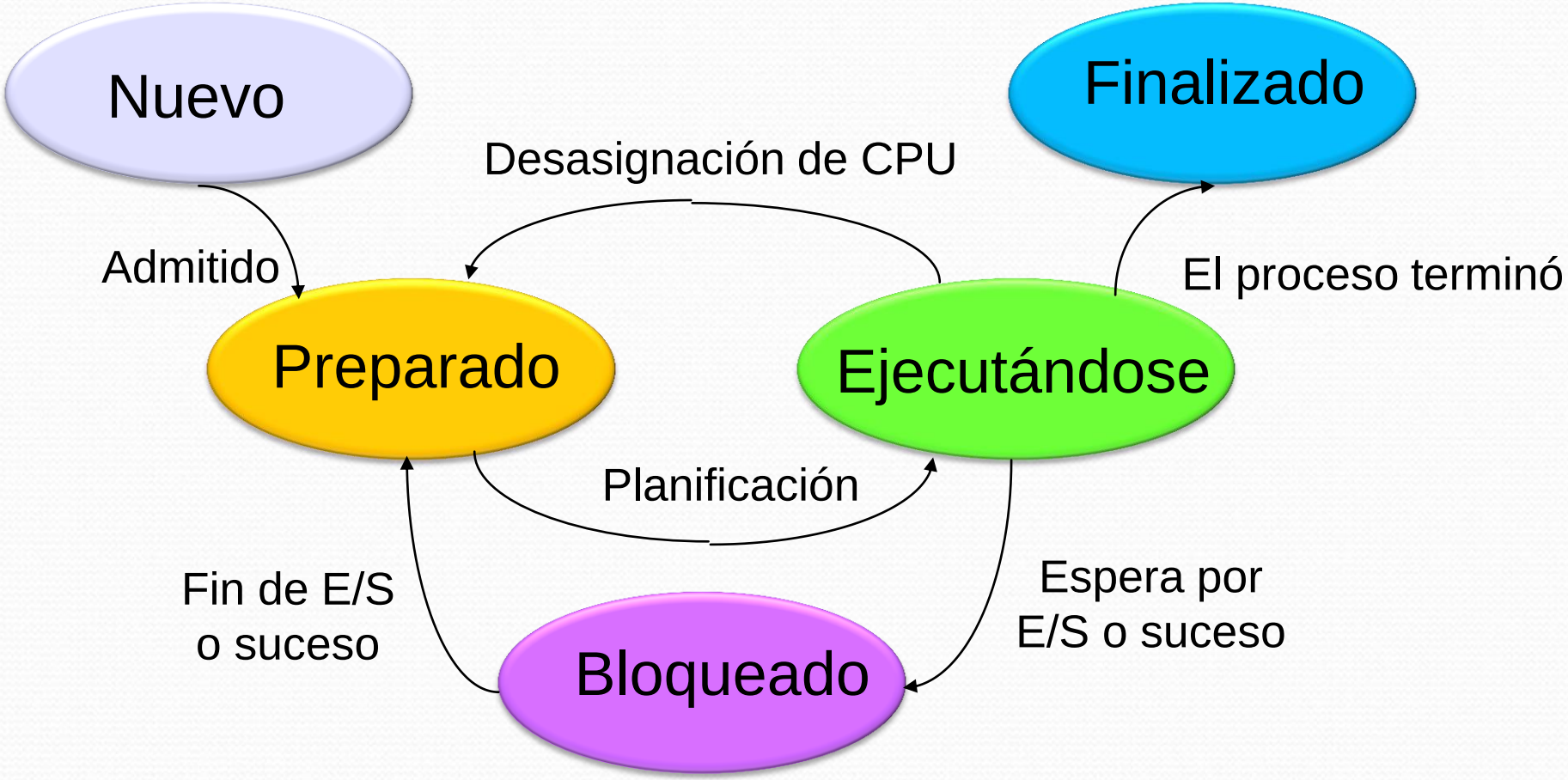


2.1.3 Modelo de cinco estados de los procesos [Stal05] (pp. 114-120)

- El modelo de cinco estados trata de representar las actividades que el SO lleva a cabo sobre los procesos:
  - Creación.
  - Terminación.
  - Multiprogramación.

- Para ello hace uso de cinco estados:
  - **Nuevo**
  - **Preparado** (listo para ejecutarse)
  - **Ejecutándose**
  - **Bloqueado**
  - **Finalizado**

2.1.3 Modelo de cinco estados de los procesos



Transiciones entre estados

- **Nuevo → Preparado**. El PCB está creado y el programa está disponible en memoria.
- **Ejecutándose → Finalizado**. El proceso finaliza normalmente o es abortado por el SO a causa de un error no recuperable.
- **Preparado→ Ejecutándose**. El SO (planificador CPU) selecciona un proceso para que se ejecute en el procesador.
- **Ejecutándose → Bloqueado**. El proceso solicita algo al SO por lo que debe esperar.
- **Ejecutándose → Preparado**. Un proceso ha alcanzado el máximo tiempo de ejecución ininterrumpida.
- **Bloqueado → Preparado**. Se produce el evento por el cual el SO bloqueó al proceso.

Preparado (o Bloqueado) → Finalizado. Terminación de un proceso por parte de otro (en la mayoría de los SO modernos, no se permite).

2.2.1 Descripción de procesos: PCB [Carr07] (p. 87)

- **Identificadores**: Del proceso, del padre del proceso, del usuario, …
- **Contexto de ejecución**: valores de los registros **PC**, **PSW**, **SP**, …
- **Información** para **control** del **proceso**:
  - Información de estado y planificación.
  - Descripción de las regiones de memoria asignadas**.**
  - Recursos asignados.
  - Enlaces a colas de procesos.
  - Comunicación entre procesos.

Creación de un proceso: Inicialización de PCB [Stal05] (p. 137)

- Asignar **identificador** único al **proceso.**
- Asignar un nuevo **PCB.**
- Asignar **memoria** para el programa asociado.
- **Inicializar PCB**:
  - **PC**: Dirección inicial de comienzo del programa.
  - **SP**: Dirección de la pila de sistema.
  - **Memoria** donde reside el programa.
  - El resto de **campos** se inicializan a valores por omisión.

2.2.2 Control de procesos: modos de ejecución del procesador [Carr07] (p. 4)

- **Modo usuario**. El programa (de usuario) que se ejecuta en este modo sólo se tiene acceso a:
  - Un subconjunto de los registros del procesador.
  - Un subconjunto del repertorio de instrucciones máquina.
  - Un área de la memoria.

- **Modo núcleo (kernel o supervisor o sistema).** El programa (SO) que se ejecuta en este modo tiene acceso a todos los recursos de la máquina, tanto software como hardware.

¿Cómo utiliza el SO el modo de ejecución?

- El modo de ejecución (incluido en PSW) cambia a modo kernel, automáticamente por hardware, cuando se produce:
  - Una interrupción.
  - Una excepción.
  - Una llamada al sistema.

- 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: operación de cambio de modo [Stal05] (pp. 139-140)

- Se ejecuta una **rutina del SO** en el contexto del proceso que se encuentra en estado "**Ejecutándose**".

- ¿Cuándo puede realizarse? Siempre que el SO pueda ejecutarse, luego solamente como resultado de:
  - Una interrupción.
  - Una excepción.
  - Una llamada al sistema.

Pasos en la operación de cambio de modo

1) El **hardware** automáticamente **salva** como mínimo el **PC** y **PSW** y cambia el bit de modo a **modo kernel**.
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** al proceso que se estaba ejecutando. El **hardware** automáticamente **restaura** en el procesador la información del **PC** y **PSW** previamente **salvada**.

Control de procesos: operación de cambio de contexto (cambio de proceso) [Stal05] (pp. 138-139)

- Un proceso en estado "**Ejecutándose**" cambia a otro estado y un proceso en estado "**Preparado**" pasa a estado "**Ejecutándose**".

- ¿Cuándo puede realizarse? Cuando el **SO** pueda **ejecutarse** y decida llevarlo a cabo. Luego solamente como resultado de:
  - Una interrupción.
  - Una excepción.
  - Una llamada al sistema.

Pasos en una operación de cambio de contexto (Dispatcher)

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) **S**eleccionar 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.

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

2.3 Concepto de Hebra [Stal05] (pp. 158-161)

• 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**.

Concepto de Hebra

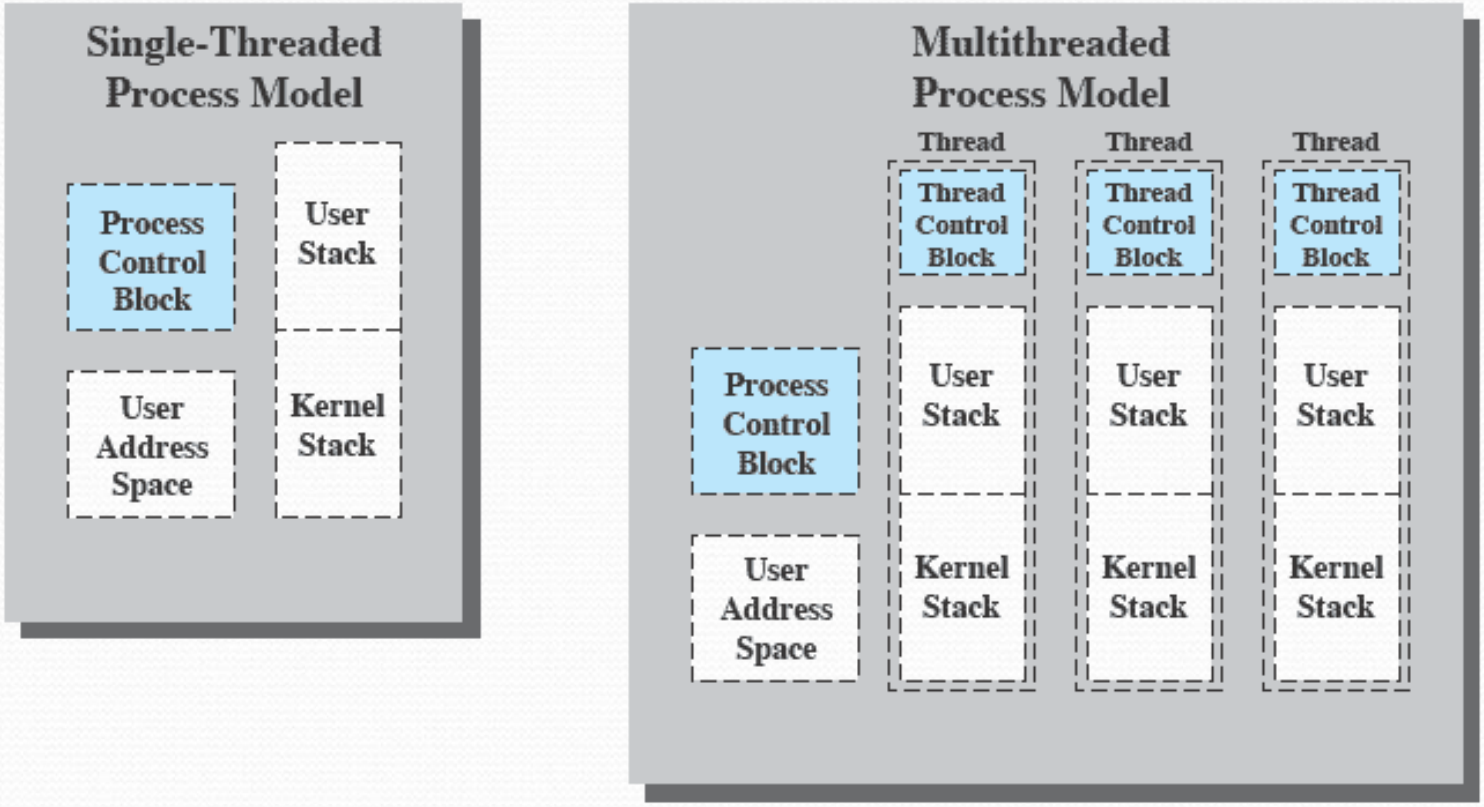
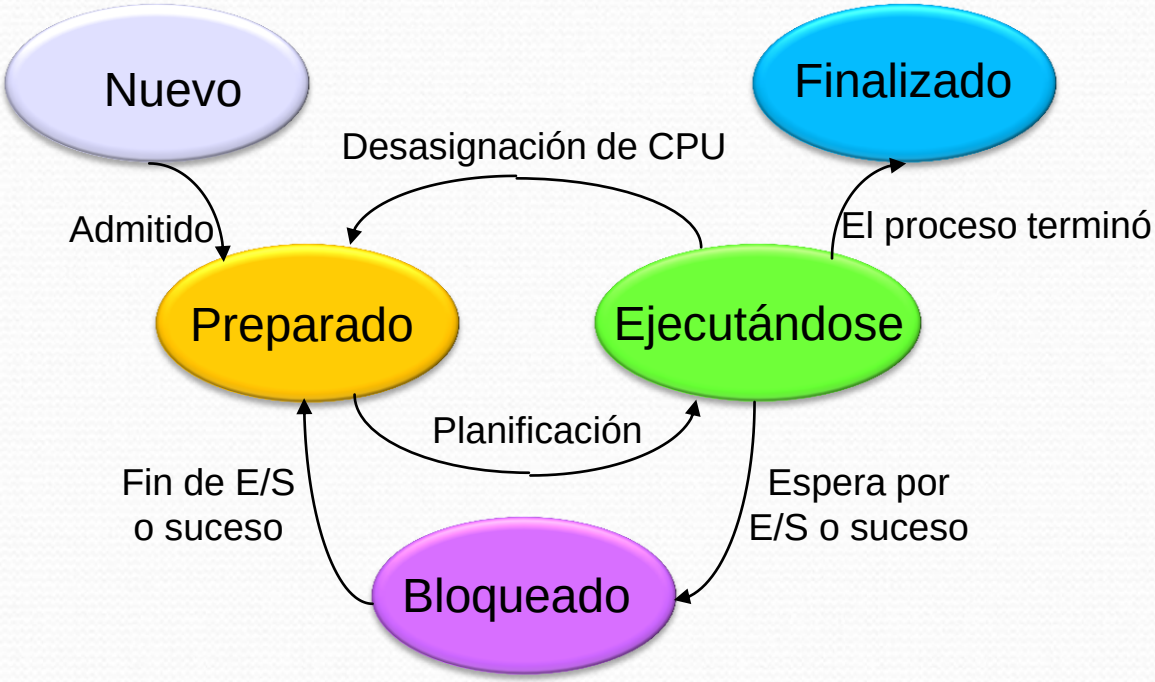


Figure 4.2 Single Threaded and Multithreaded Process Models

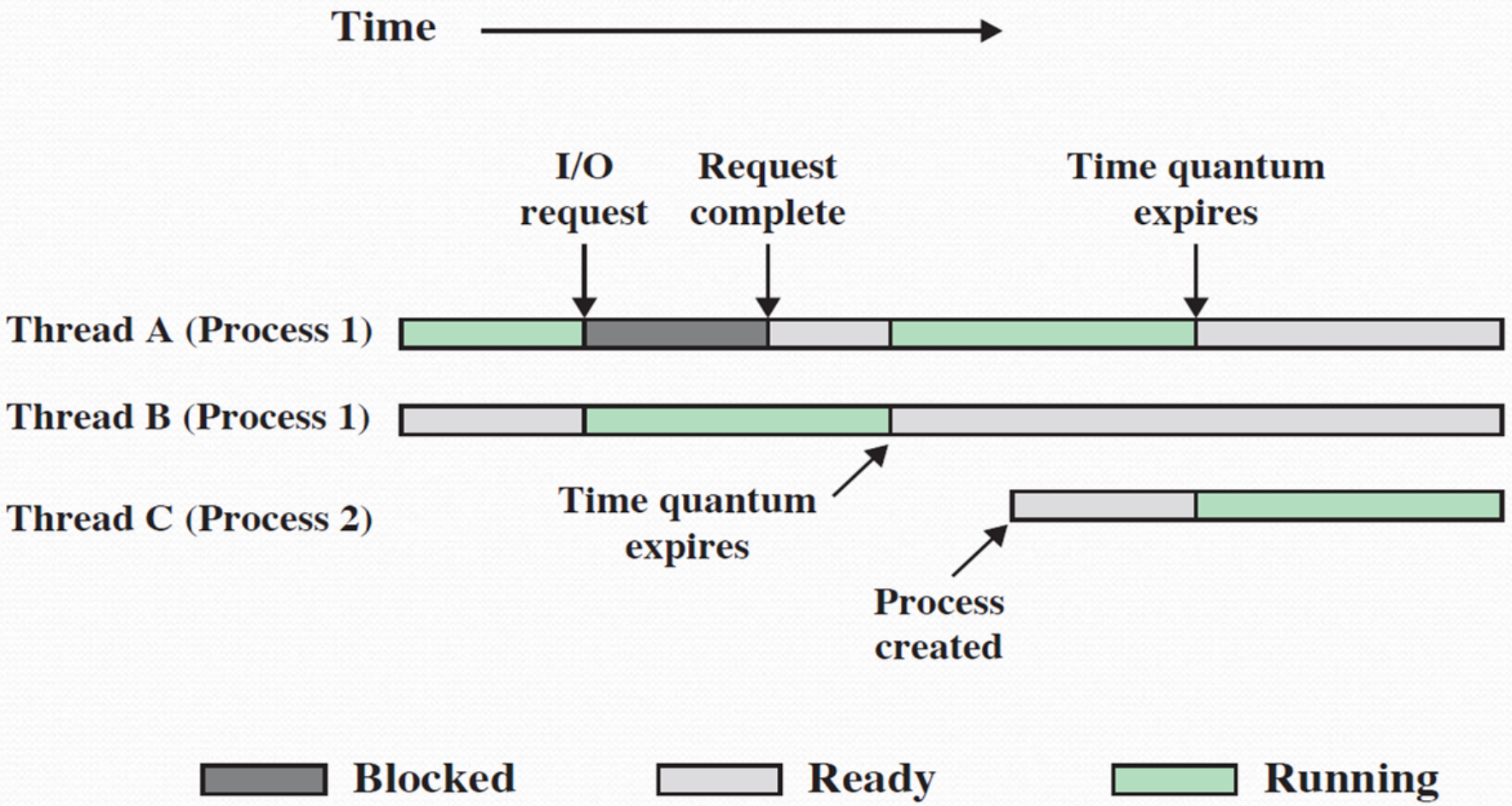
Modelo de cinco estados para hebras

• Las hebras debido a su característica de ejecución de programas presentan cinco estados análogos al modelo de estados para procesos:

- **Ejecutándose**
- **Preparado** (listo para ejecutarse)
- **Bloqueado**
- **Nuevo**
- **Finalizado**

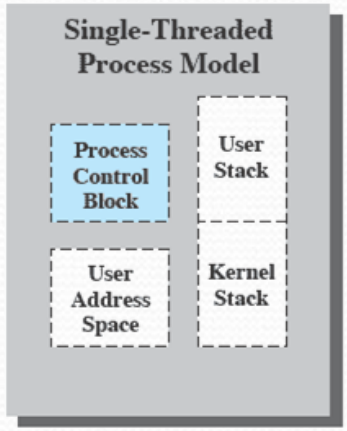


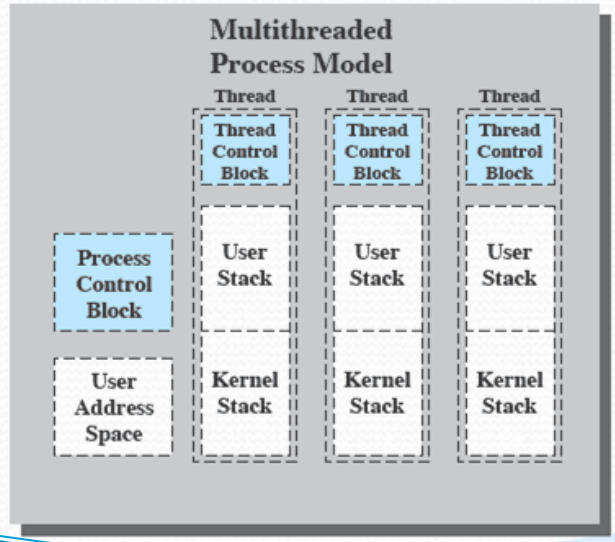
Ventajas de las hebras [Stal05] (p. 163)



Ventajas de las hebras [Stal05] (pp. 160-165)

- **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**.



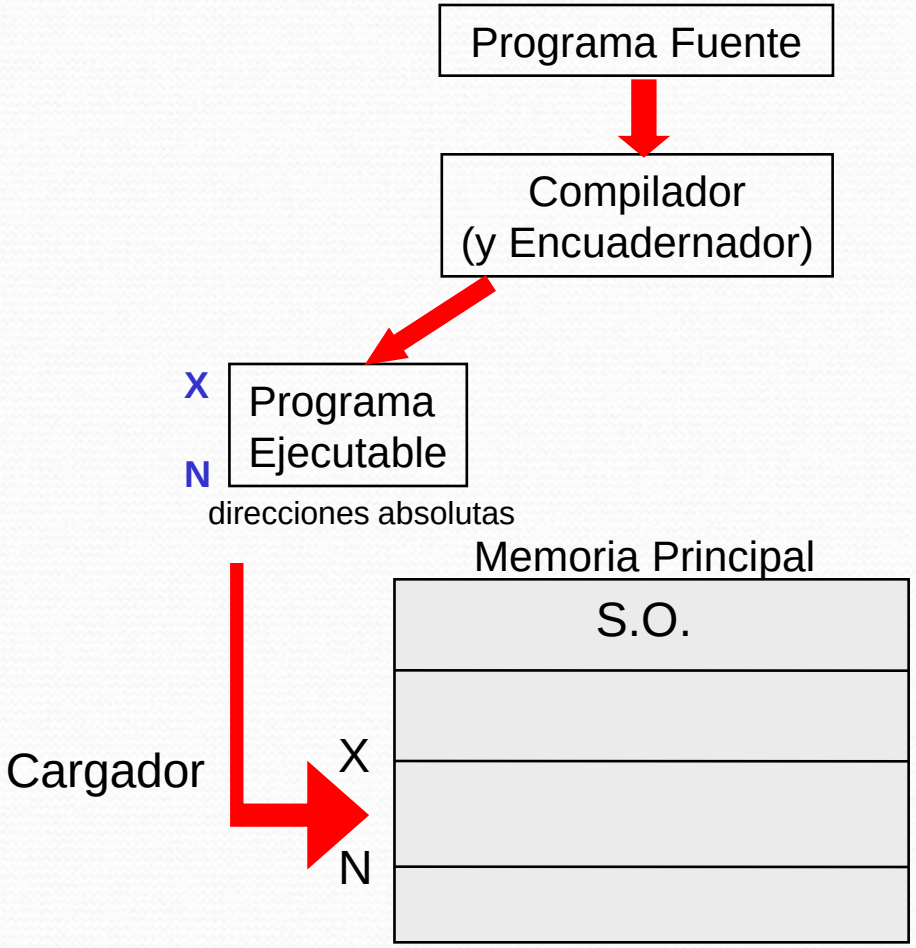


2.4 Carga absoluta y reubicación [Stall05] (pp. 308-309, 331-337)

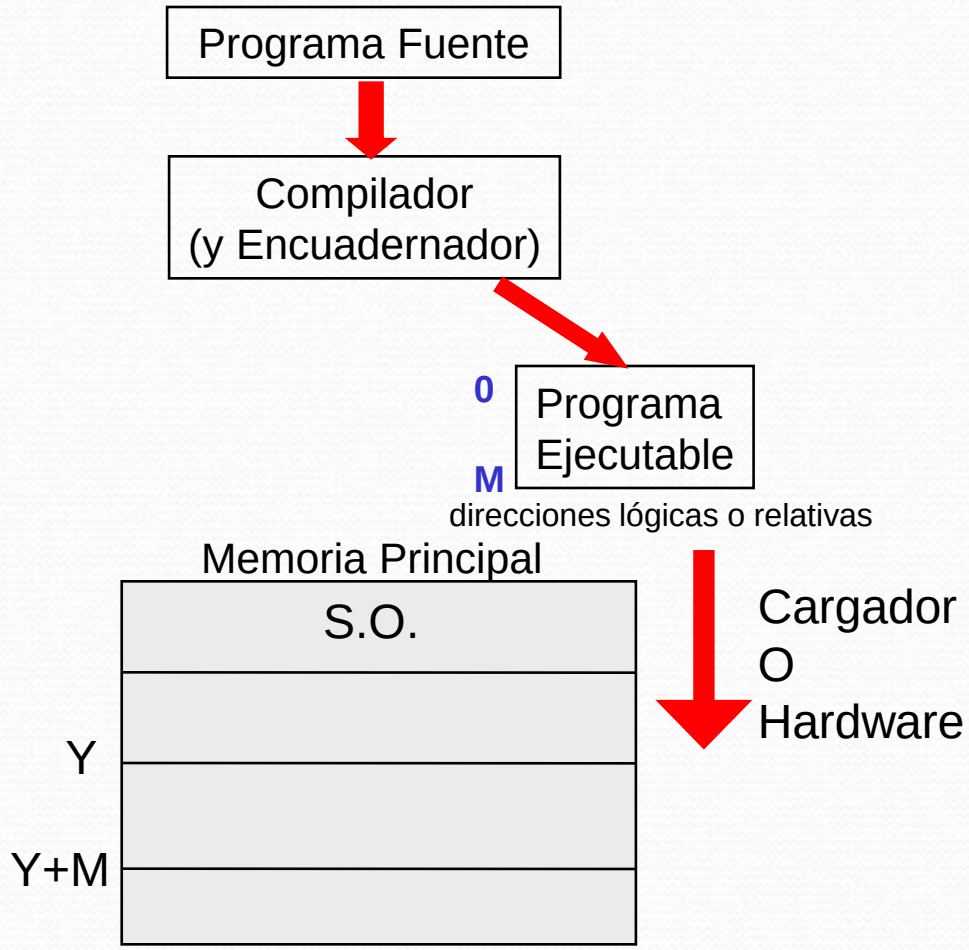
- **Carga absoluta.** Asignar direcciones físicas (direcciones de memoria principal) al programa en tiempo de compilación. El programa no es reubicable.
- **Reubicación**. Capacidad de cargar y ejecutar un programa en un lugar arbitrario de la memoria.

2.4 Gestión básica de memoria

Carga absoluta

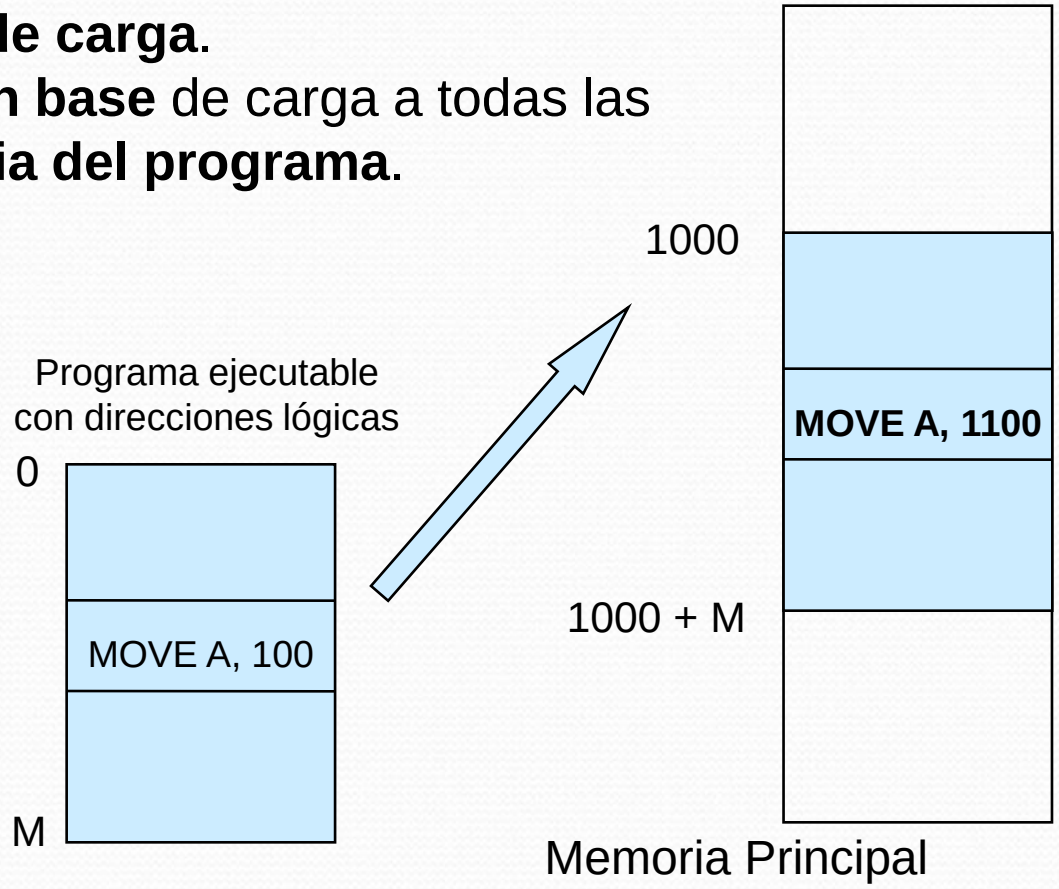


Reubicación



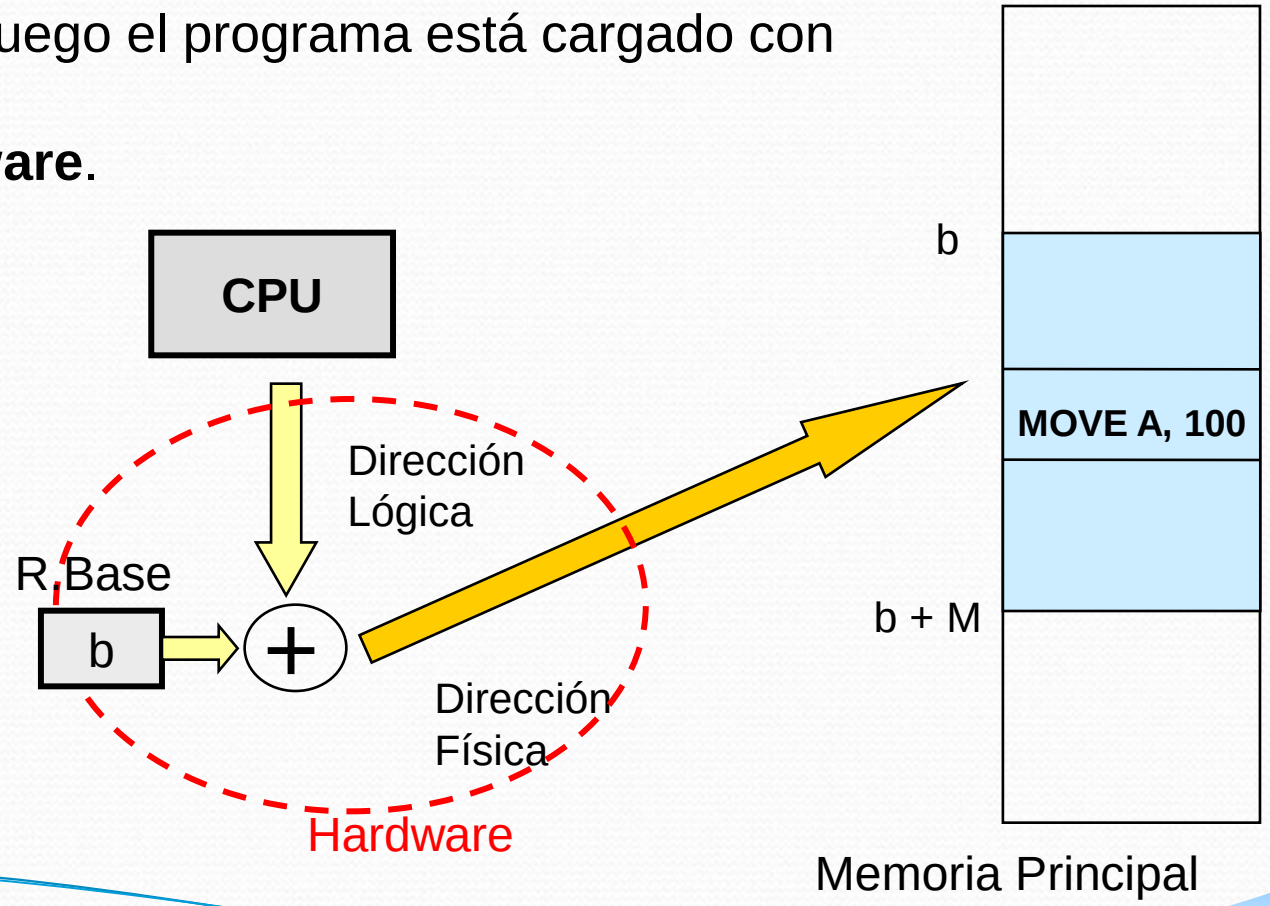
Reubicación estática

- 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

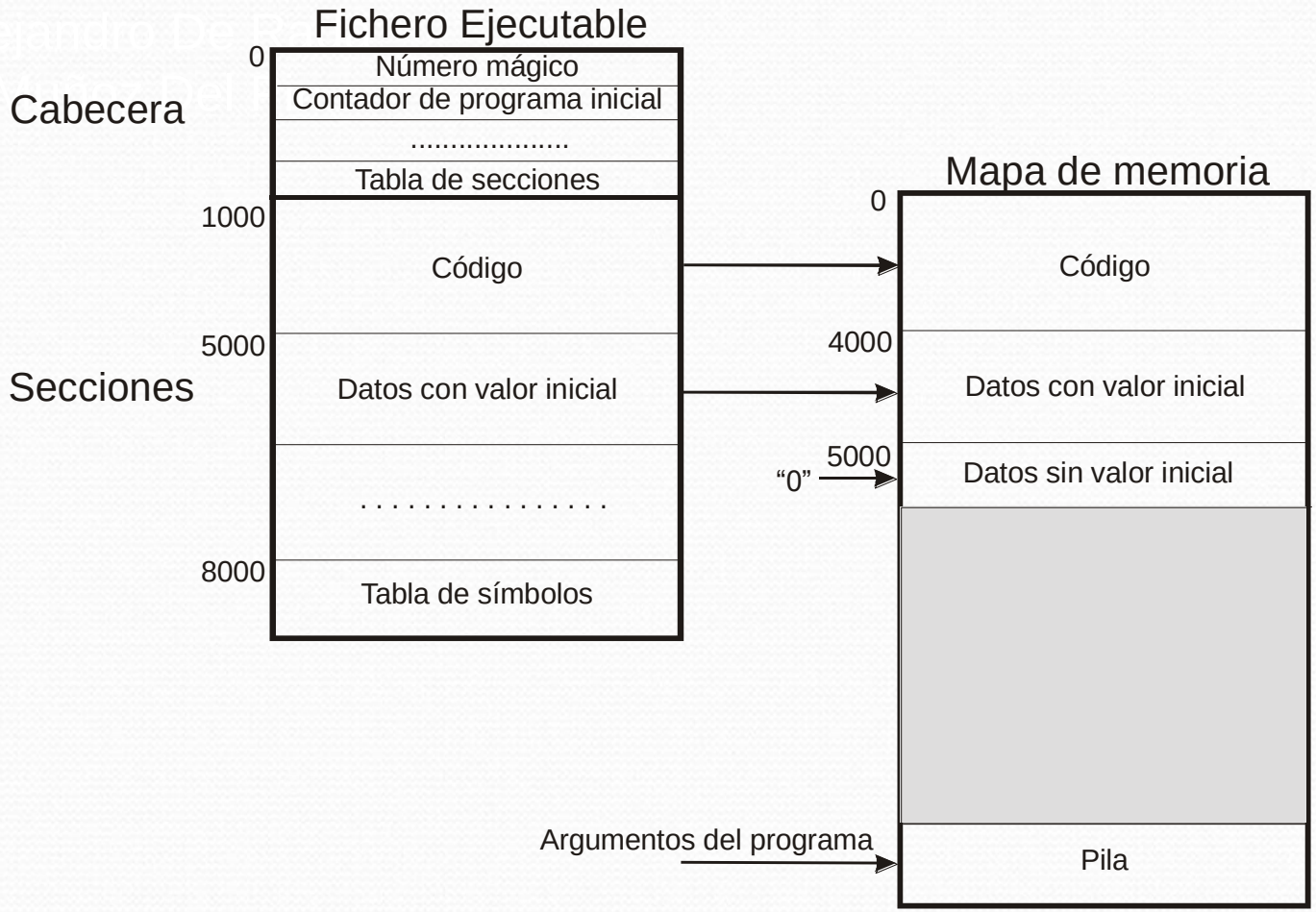
- El **compilador** genera **direcciones lógicas** (relativas) de 0 a M.
- La **traducción** de **direcciones lógicas** a **físicas** se realiza en **tiempo de ejecución** luego el programa está cargado con referencias relativas.
- Requiere apoyo **hardware**.



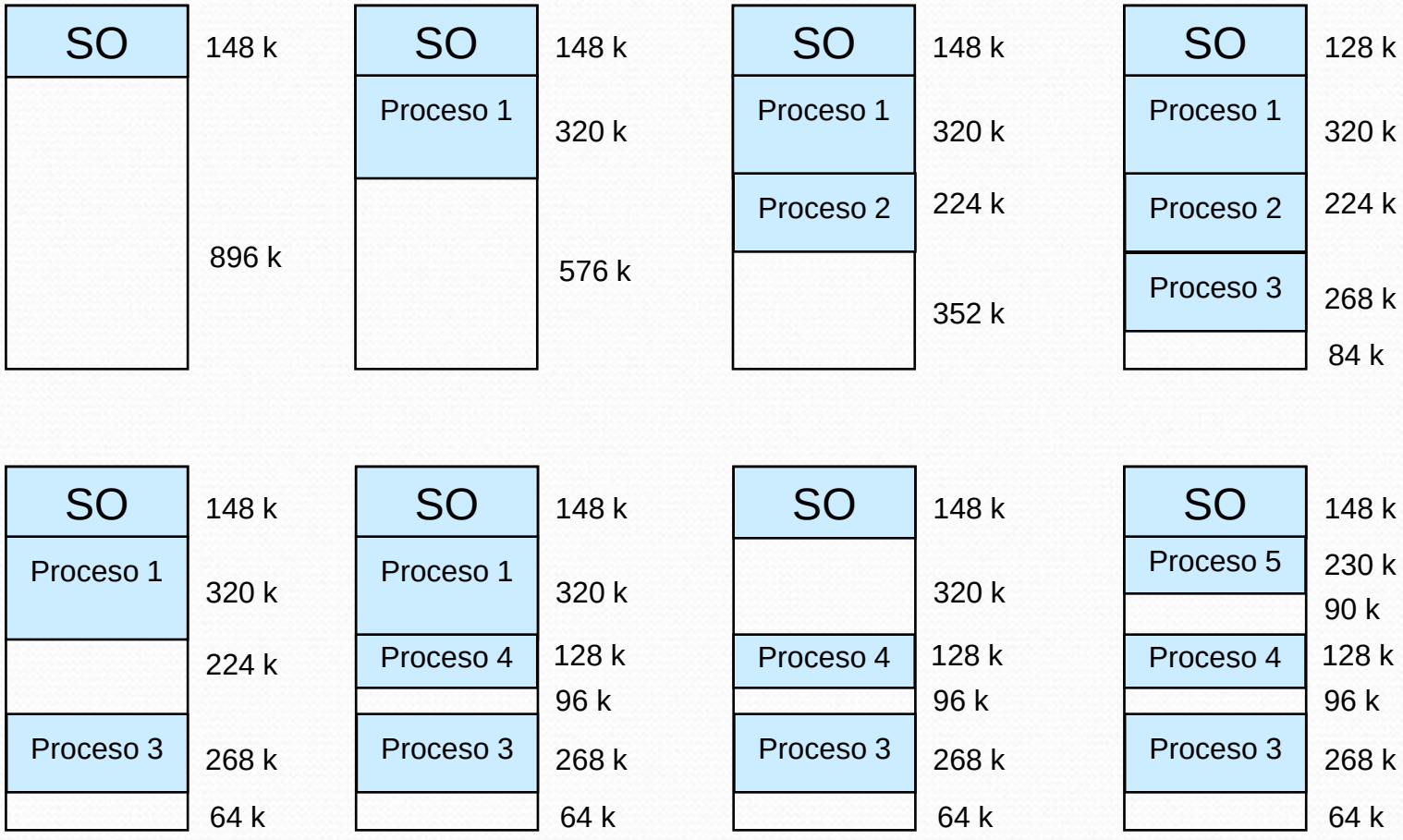
Espacios para las direcciones de memoria

- **Espacio de direcciones lógico.** 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**. 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.

Ejemplo de mapa de memoria de un proceso [Carr07] (p. 277)



Problema de la fragmentación de memoria [Stal05] (pp. 314-316)



Solución a la fragmentación de memoria [Stall05] (pp. 321-327)

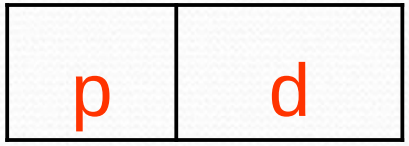
- Trocear el espacio lógico en unidades más pequeñas: **páginas** (elementos de longitud fija), o **segmentos** (elementos de longitud variable).
- Los trozos no tienen por qué ubicarse consecutivamente en el espacio físico.
- Los esquemas de organización del espacio lógico de direcciones y de traducción de una dirección del espacio lógico al espacio físico que comentaremos son:
  - **Paginación**
  - **Segmentación**

2.4.1 Paginación

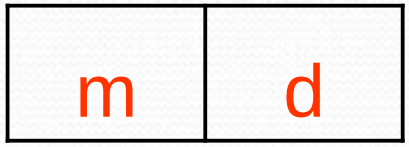
- El espacio de direcciones físicas de un proceso puede ser no contiguo.
- La memoria física se divide en bloques de tamaño fijo, denominados *marcos de página*. El tamaño es potencia de dos, de 512 B a 8 KB.
- El espacio lógico de un proceso se divide conceptualmente en bloques del mismo tamaño, denominados *páginas.*
- Los marcos de página contendrán páginas de los procesos.

2.4.1 Paginación

Las **direcciones lógicas**, que son las que genera la CPU, se dividen en ***número de página*** (p) y ***desplazamiento*** dentro de la página (d).



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



2.4.1 Paginación

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. *Existe una tabla de páginas por proceso.*

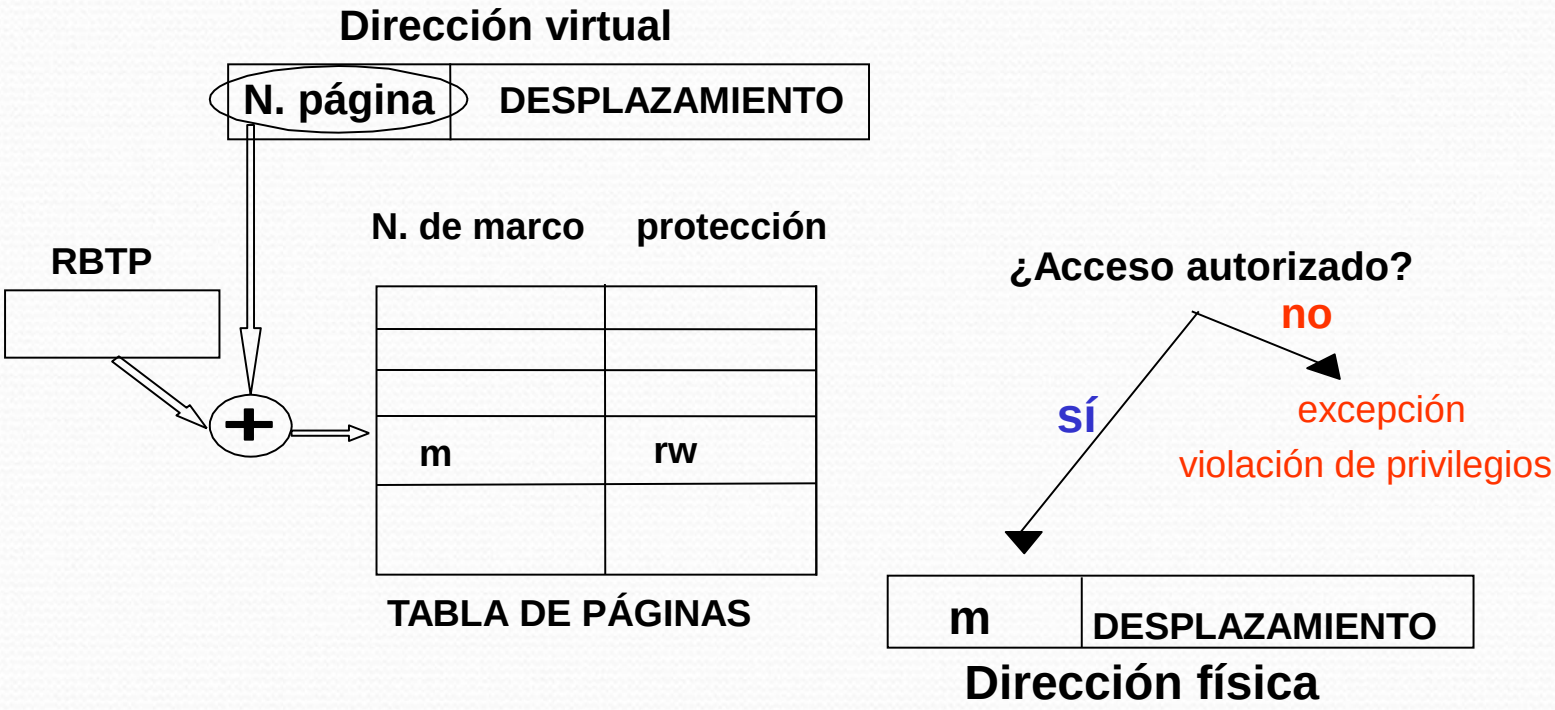
Tabla de marcos de página, usada por el S.O. y contiene información sobre cada marco de página.

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



Dirección física = m * tamaño_página + Desplazamiento

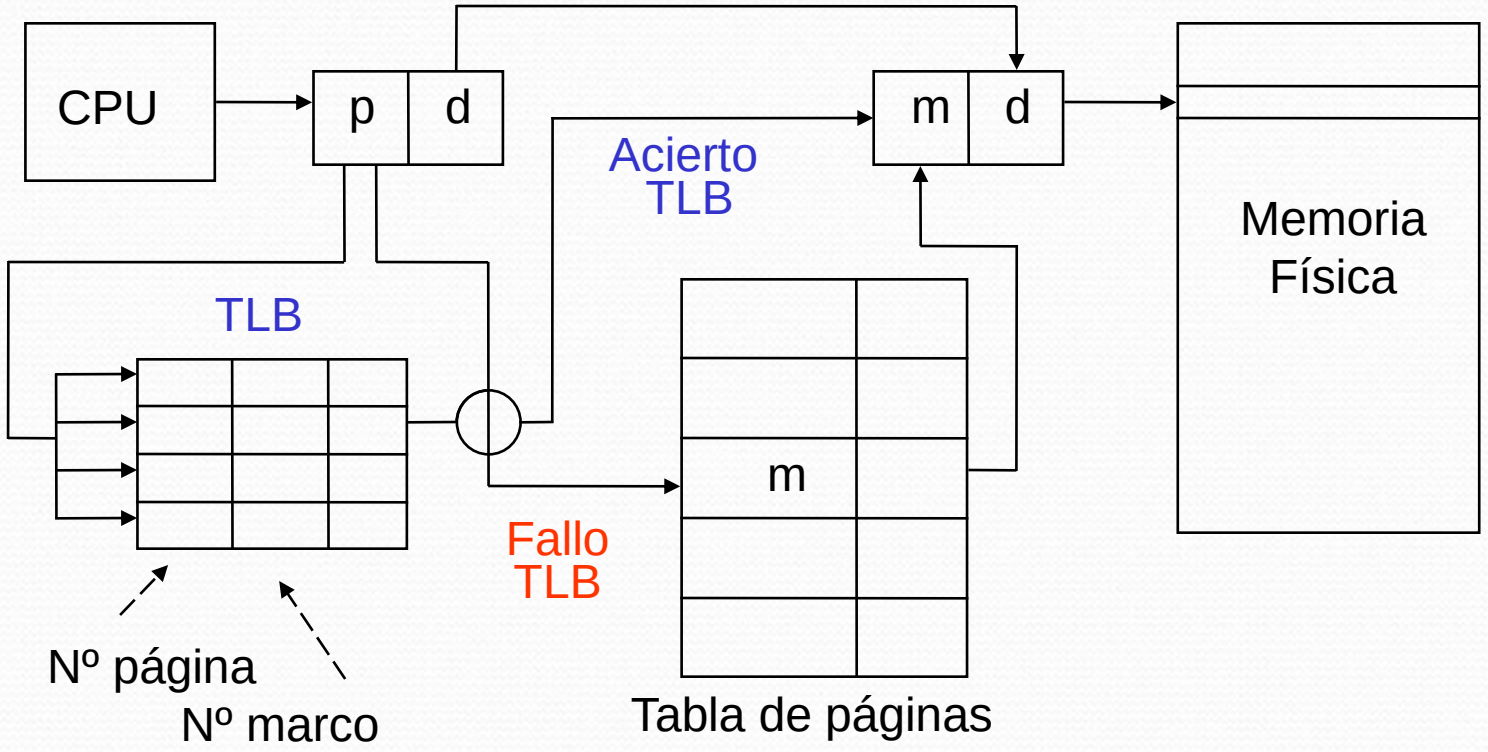
Implementación de la Tabla de Páginas

- 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 este esquema cada acceso a una instrucción o dato requiere **dos accesos a memoria**, uno a la tabla de páginas y otro a memoria.

Búfer de Traducción Adelantada (TLB) [Stal05] (pp. 349-351)

- El problema de los dos accesos a memoria se resuelve con una caché hardware de consulta rápida denominada *búfer de traducción adelantada* o *TLB* (*Translation Look-aside Buffer*).
- El TLB se implementa como un conjunto de *registros asociativos* que permiten una búsqueda en paralelo.
- De esta forma, para traducir una dirección:
  1. Si existe ya en el registro asociativo, obtenemos el marco.
  2. Si no, la buscamos en la tabla de páginas y se actualiza el TLB con esta nueva entrada.

Esquema de traducción con TLB



2.4.2 Segmentación

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

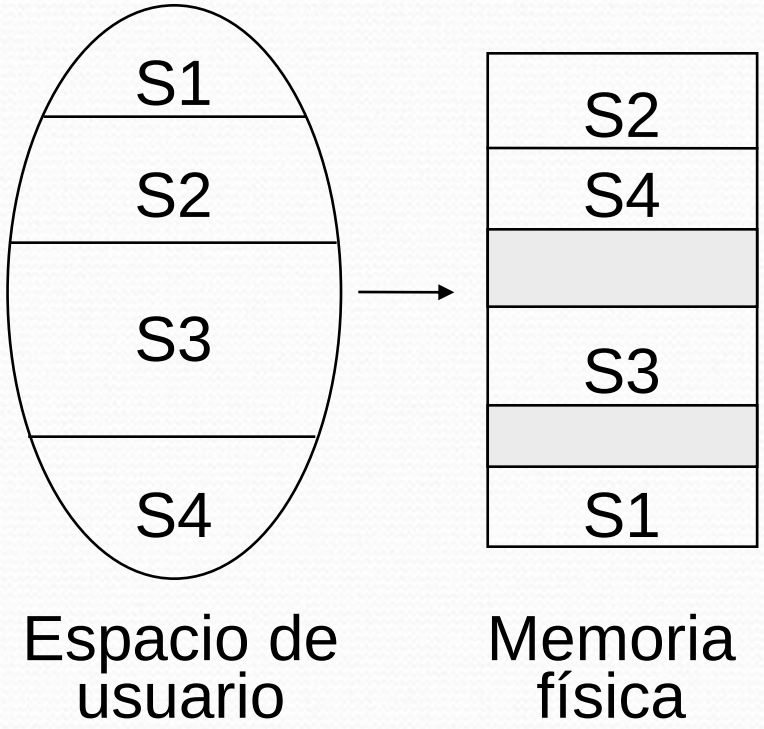


Tabla de Segmentos

Una dirección lógica es una tupla:

<número_de_segmento, desplazamiento>

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 nº de segmento *s*, generado en una dirección lógica, es legal si $s < STLR$ (suele almacenarse en el PCB del proceso).

Esquema de traducción