Resumen Tema 2 FS

1. La evolución de los sistemas operativos

Procesamiento Serie

El programador interaccionaba directamente con el hardware, no había ningún sistema operativo.

El programa se cargaba mediante un dispositivo de entrada. Si ocurría un error el programador debía examinar la memoria y el procesador para determinar la causa del error y si el programa terminaba de forma normal, la salida aparecía en la impresora.

Presentaban dos problemas:

- **Planificación:** un usuario podía solicitar un bloque de tiempo de por ejemplo una hora y terminar en 45 minutos lo que implicaba malgastar tiempo de procesamiento. Al igual, podía tener problemas si no terminaba en el tiempo asignado y era forzado a terminar antes.
- **Tiempo de configuración:** los pasos para la ejecución de un único programa (denominado **trabajo**) podían suponer montar y desmontar varias cintas y si ocurría un error el usuario tenía que empezar desde el principio. Se utilizaba una gran cantidad de tiempo en la configuración del programa que se iba a ejecutar.

Sistemas en lotes sencillos

La idea de procesamiento en lotes sencillos es el uso de una pieza de software denominada **monitor**.

De esta forma el usuario no tiene que acceder directamente a la máquina. En su lugar solo *envía los trabajos al operador del computador*, el cual *crea un sistema por lotes con todos los trabajos y coloca la secuencia de dichos trabajos en el dispositivo de entrada*, para que lo utilice el monitor.

El **monitor** es un programa que realiza una función de planificación creando una cola donde se sitúa un lote de trabajos. Además, el monitor mejora el tiempo de configuración de los trabajos. Cada uno de los trabajos incluye un conjunto de instrucciones en **lenguaje de control de trabajos** (JCL).

Sistemas en lotes multiprogramados

El procesador frecuentemente se encuentra parado. El problema consiste en que los dispositivos de E/S son lentos comparados con el procesador. En el siguiente se muestra un ejemplo de monoprogramación (existe un único programa) y multiprogramación.

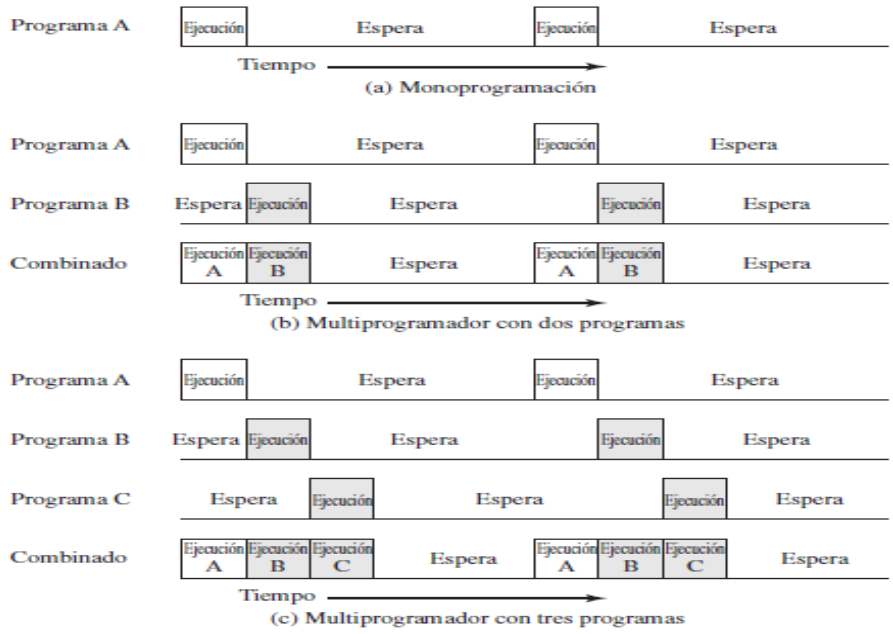


Figura 2.5. Ejemplo de multiprogramación.

El concepto de **multiprogramación** o **multitarea** se entiende por cuándo se pueden ejecutar varios programas o procesos de forma simultánea. Prácticamente todos los sistemas operativos actuales son **multiprogramacion**. Se diseñó para alcanzar la **máxima eficiencia**.

Se puede expandir la memoria para que albergue tres, cuatro o más programas y pueda haber multiplexación entre todos ellos.

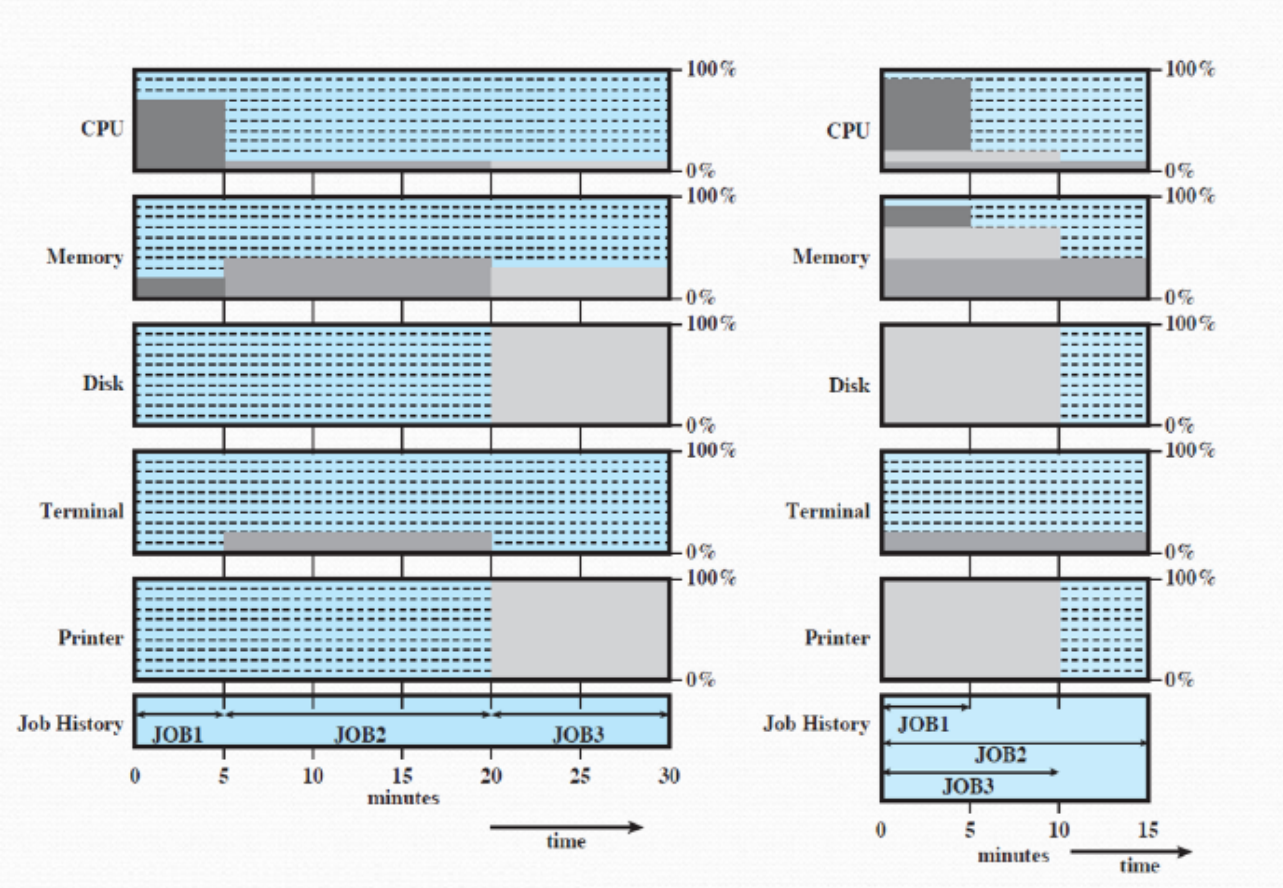
Para que quede más claro el concepto a continuación dejaré este ejemplo con la diferencia entre monoprogramación y multiprogramación.

Tabla 2.1. Atributos de ejecución de ejemplos de programas.

	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 M	100 M	75 M
¿Necesita disco?	No	No	Sí
¿Necesita terminal?	No	Sí	No
¿Necesita impresora?	No	No	Sí

Tabla 2.2. Efectos de la utilización de recursos sobre la multiprogramación.

	Monoprogramación	Multiprogramación
Uso de 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



Un **sistema en lotes multiprogramado**, debe basarse en ciertas características hardware. La más útil para la multiprogramación es el hardware que soporta las interrupciones de **E/S y DMA (acceso directo a memoria)**. De esta forma el procesador puede solicitar un mandato de E/S para un trabajo y continuar con la ejecución de otro trabajo mientras.

Los **sistemas operativos multiprogramados** son bastantes más sofisticados que los **monoprogramados**. Para poder tener varios trabajos listos para ejecutar, estos deben de guardarse en memoria principal requiriendo de alguna forma de **gestión de memoria**.

Sistemas de tiempo compartido

El concepto de **tiempo compartido** consiste en que múltiples usuarios pueden acceder simultáneamente al sistema a través de terminales. Al igual que la multiprogramación permite al procesador gestionar múltiples trabajos en lotes, la multiprogramación también se puede utilizar **para gestionar múltiples trabajos interactivos** de distintos usuarios.

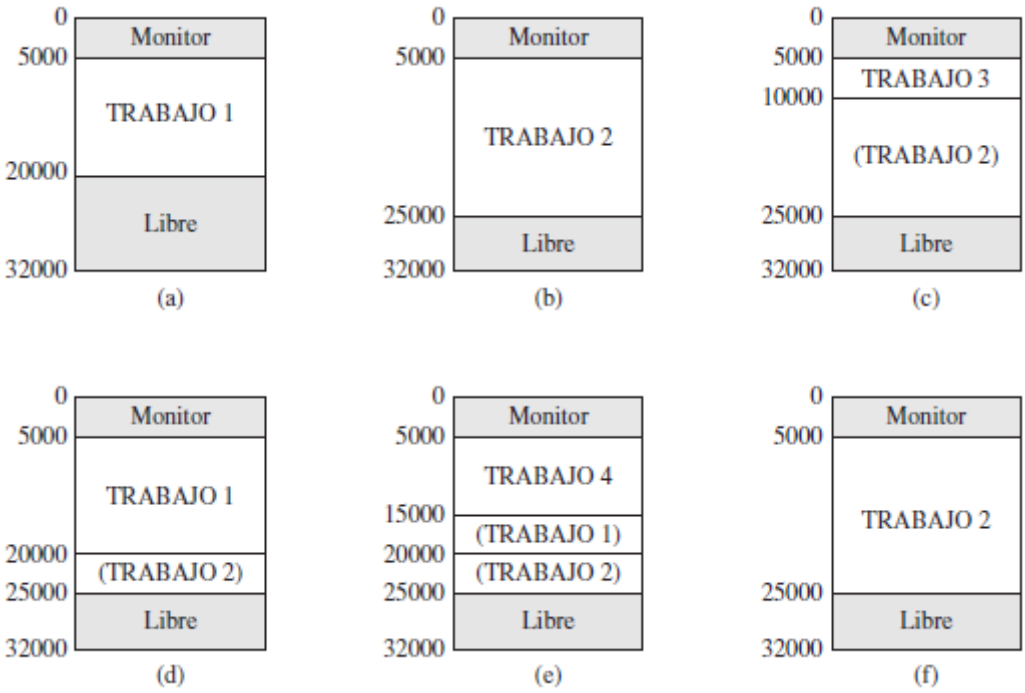
Ambos tipos de procesamiento, en lotes y tiempo compartido, utilizan multiprogramación. Las diferencias más importantes entre ellos son:

	Multiprogramación en lotes	**Tiempo compartido**
Objetivo principal	Maximizar el uso del procesador	Minimizar el tiempo de respuesta
Fuente de directivas al sistema operativo	Mandatos del lenguaje de control de trabajos proporcionados por el trabajo	Mandatos introducidos al terminal

Uno de los primeros sistemas operativos de tiempo compartido desarrollados fue el **CTSS**. Para preservar el estado del programa de usuario antiguo, los **programas de usuario y los datos se escriben en el disco** antes de que se lean los nuevos programas. Para

minimizar el tráfico de disco, solo se escriben en el disco cuando el programa entrante va a sobrescribir al antiguo.

- TRABAJO1: 15.000
- TRABAJO2: 20.000
- TRABAJO3: 5000
- TRABAJO4: 10.000



Esta técnica es primitiva comparada con otras técnicas más modernas de tiempo compartido actuales, pero funcionaba.

2. Procesos

Concepto de proceso

Se han dado muchas definiciones del término **proceso**

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

- **Identificadores**: Del proceso, del padre del proceso, del usuario, …
- **Contexto** de registros del procesador: **PC**, **PSW**, **SP**, …
- **Información** para **control** del **proceso**:
  - **Estado** del proceso. (modelo de estados)
  - Parámetros de **planificación**
  - **Evento** que mantiene al proceso bloqueado
  - Cómo acceder a la **memoria** que aloja el programa asociado al proceso
  - **Recursos** utilizados por el proceso

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

- ✓ Un programa ejecutable.
- ✓ Los datos asociados que necesita el SO para ejecutar el programa.
- ✓ **El contexto de ejecución del programa.**

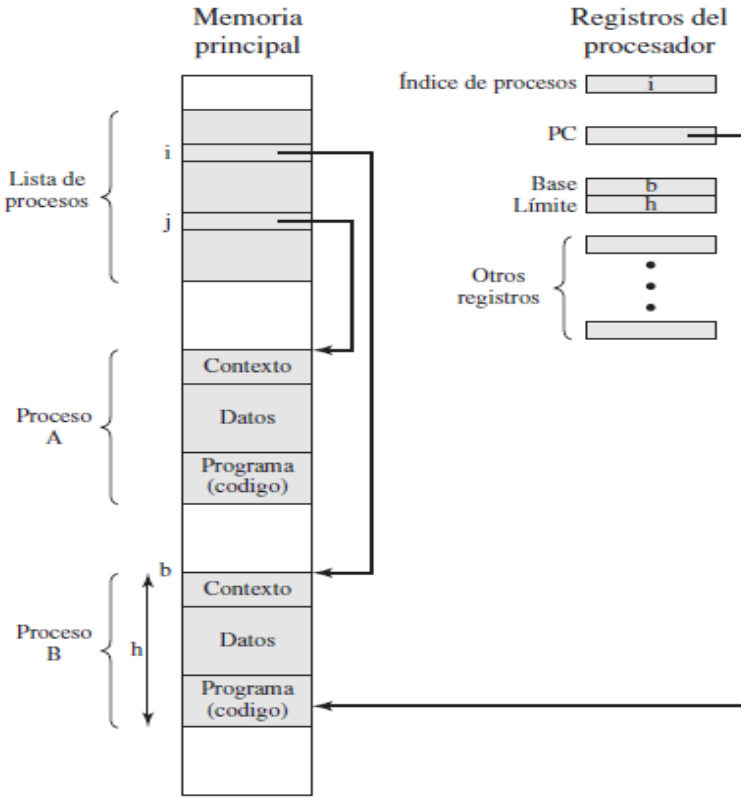
Este último elemento es esencial. El **contexto de ejecución**, o **estado del proceso** es el conjunto de datos por el cual el SO es capaz de supervisar y gestionar el proceso para la correcta ejecución de este.

Esta información podría estar contenida en su totalidad en la **entrada** del proceso (la cual se encuentra en **la lista de procesos** y contiene un **puntero a la ubicación** del bloque de memoria que contiene dicho proceso; cada proceso tiene su propia entrada) o bien la entrada contendrá **solo una parte** del contexto del proceso y el resto del contexto podría estar almacenado en el **bloque del propio proceso** o en **otra región** de memoria.

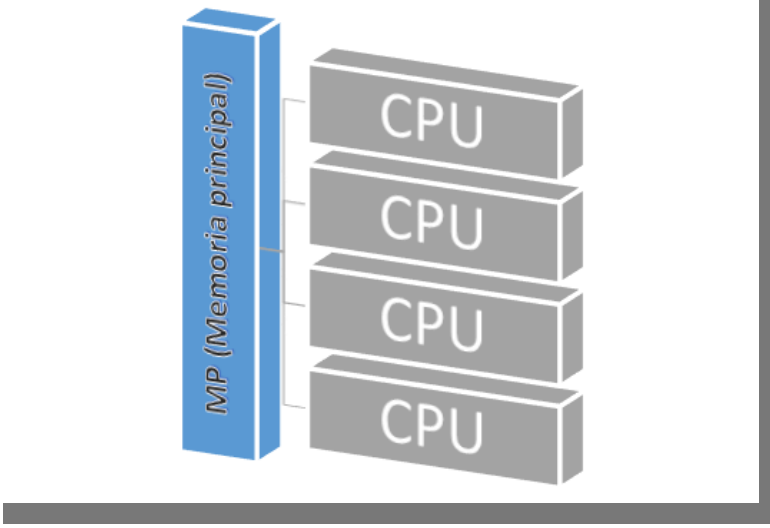
Cada proceso se incluye en una **lista de procesos** (mencionada anteriormente) que construye y mantiene al SO.

El **registro índice** del proceso contiene el **índice del proceso** que el procesador está controlando actualmente en la lista de procesos. El contador de programa **(PC)** apunta a la siguiente instrucción del programa que se va a ejecutar.

Los **registros base y límite** definen la región de memoria ocupada por el proceso: el **registro base (b)** contiene la dirección inicial del bloque del proceso y el **registro límite (h)** contiene el tamaño de dicho bloque (bytes o palabras).



- ✓ **PSW:** Program Status Word (Palabra de Estado del Programa)
- ✓ **PCB:** Bloque de control de proceso
- ✓ **SMP:** Symetric Multi-Processing
  - Podemos tener más de un procesador en el equipo, pero solo tenemos una memoria principal repartida para todos ellos.

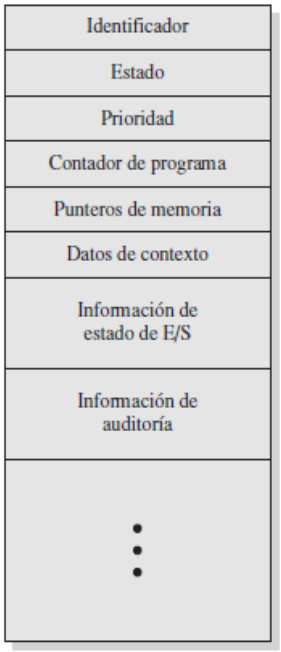


Cuando un proceso es **interrumpido**, su **estado** pasa a “esperando ejecutarse” y previamente los contenidos de todos los registros de dicho proceso son guardados en el **contexto** para una posterior continuación de la ejecución de este.

Esta estructura permite el desarrollo de técnicas potentes que aseguren la **coordinación y la cooperación** entre los procesos.

Bloque de control de proceso (PCB o BCP, Process Control Block)

Un proceso se puede caracterizar por una serie de elementos:



- **Identificador (PID)**. Un identificador único asociado al proceso, para distinguirlo del resto.
- **Estado**. En qué situación se encuentra el proceso en cada momento (modelo de estados).
- **Prioridad**. Nivel de prioridad con respecto al resto de procesos.
- **Contador de programa**. La dirección de la siguiente instrucción del programa que se ejecutará.
- **Punteros a memoria**. Punteros con la ubicación del código del programa además de cualquier bloque de memoria compartido con otros procesos.
- **Datos de contexto**. Datos presentes en los registros del procesador cuando el proceso está corriendo.
- **Información de estado de E/S.** Incluye peticiones de E/S pendientes, dispositivos de E/S asignados a dicho proceso, etc.
- **Información de auditoria.** Puede incluir la cantidad de tiempo de procesador y de tiempo de reloj utilizados.

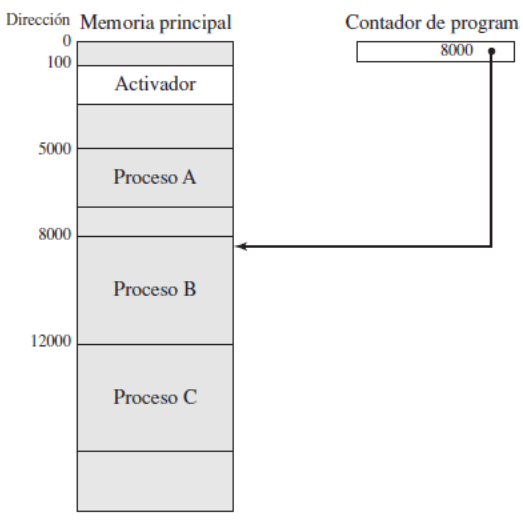
La información de la lista anterior se almacena en una estructura de datos llamada **bloque de control de proceso** que el SO crea y gestiona. El BCP es la **herramienta clave** que permite al sistema operativo dar soporte a múltiples procesos y proporcionar multiprogramación.

El BCP tiene **suficiente información** como para poder **interrumpir** el proceso que está actualmente corriendo, dejando el SO libre para poner otro proceso en estado de ejecución y posteriormente **restaurar su estado** de ejecución como si no hubiera habido interrupción alguna.

Concepto de traza de ejecución

Se puede caracterizar el comportamiento de un determinado proceso, listando la **secuencia de instrucciones** que se ejecuta para dicho proceso. A esta lista se la denomina **traza** del proceso. Se puede caracterizar el comportamiento de un procesador mostrando como las trazas de varios procesos se entrelazan.

Existe un pequeño programa lla
intercambiar el procesador de u



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
(a) Traza del Proceso A	(b) Traza del Proceso B	(c) Traza del Proceso C

5000 = Dirección de comienzo del programa del Proceso A.
8000 = Dirección de comienzo del programa del Proceso B.
12000 = Dirección de comienzo del programa del Proceso C.

Figura 3.2.

La Figura 3.4 muestra las trazas entrelazadas resultante de los 52 primeros ciclos de ejecución. En este ejemplo, el SO sólo deja que [illegible]clos de instrucción, después de los cuales se interrumpe.

EXPLICACION FIGURA 3.4

Las primeras 6 instrucciones del proceso A se ejecutan seguidas de una **alarma de temporización (time-out)** y de la ejecución del **activador**, que ejecuta seis instrucciones antes de devolver el control al proceso B.

Después de que se ejecuten 4 instrucciones, el proceso B solicita una acción de E/S, para la cual debe esperar. Por tanto, el procesador deja de ejecutar el proceso B y pasa a ejecutar el proceso C, por medio del activador.

Después de otra alarma de temporización, el procesador vuelve al proceso A.

Cuando este proceso llega a su temporización, el proceso B aún estará esperando que se complete su operación de E/S, por lo que el activador pasa de nuevo al proceso C.

Figura 3.3. Traza de los procesos de la Figura 3.2.

1	5000		27	12004	
2	5001		28	12005	
3	5002				Temporización
4	5003		29	100	
5	5004		30	101	
6	5005		31	102	
		Temporización	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				Temporización
16	8003		41	100	
		Petición de E/S	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	
					Temporización

100 = Dirección de comienzo del programa activador.
Las zonas sombreadas indican la ejecución del proceso de activación;
la primera y la tercera columna cuentan ciclos de instrucciones;
la segunda y la cuarta columna las direcciones de las instrucciones que se ejecutan

Figura 3.4. Traza combinada de los procesos de la Figura 3.2.

Creación de un proceso

Cuando se va a añadir un nuevo proceso, el SO construye las estructuras de datos se usan para manejar el proceso y se reserva el espacio en memoria para dicho proceso. Cuando un SO crea un nuevo proceso a petición explícita de otro, dicha acción se denomina **creación del proceso**.

Cuando un proceso lanza otro, el primero se denomina **proceso padre** y proceso creado se denomina **proceso hijo**.

- Asignar **identificador** único al **proceso**
- Asignar un nuevo **PCB**
- Asignar **memoria** para el programa asociado
- **Inicializar PCB**:
  - **PC**: Dirección inicial del 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

Terminación de un proceso

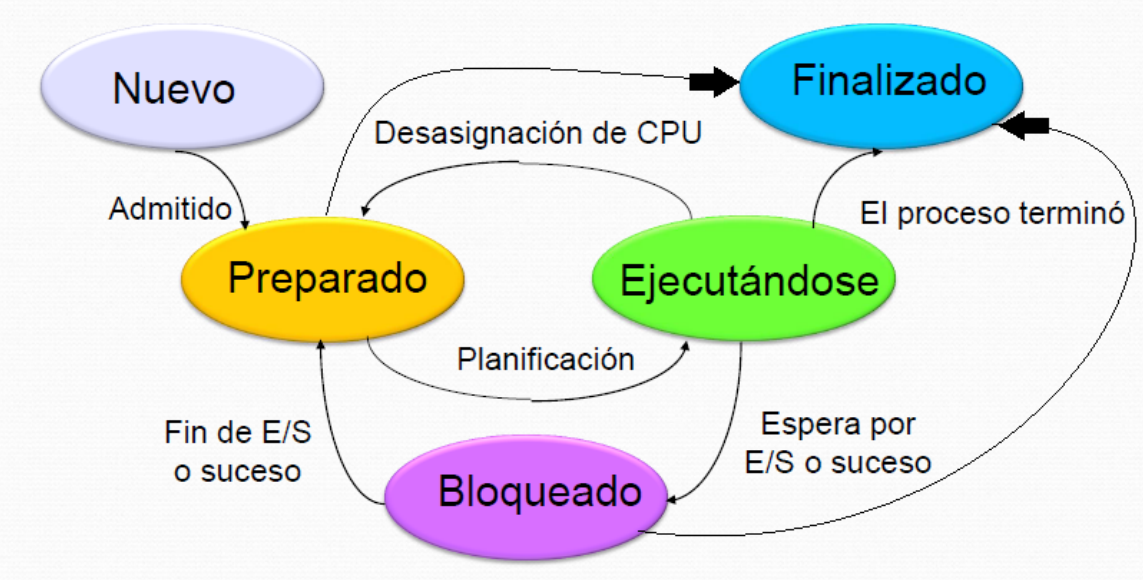
Viene de venir prevenido por una instrucción HALT, una llamada a un servicio del SO específica para la terminación del proceso, la finalización normal, algún error, la finalización de un proceso padre, etc.

Todas estas acciones tienen como resultado final la solicitud de un servicio al SO para terminar con el proceso solicitante.

Modelo de cinco estados de los procesos

Anteriormente vimos el modelo de dos estados en el cual el proceso solo se podía encontrar como **“Ejecutando”** o **“No ejecutando”**. En este modelo, dividiremos el estado “No ejecutando” en dos: **“Listo”** y **“Bloqueado”**, además de dos estados más adicionales para su correcta gestión:

- **Ejecutando.** El proceso se encuentra actualmente en ejecución.
- **Preparado/Listo.** Un proceso que está preparado para su ejecución cuando esta sea posible.
- **Bloqueado.** Un proceso que se encuentra detenido y no se puede ejecutar hasta que se cumpla un evento determinado o se complete una operación E/S.
- **Nuevo.** Un proceso que se acaba de ser definido pero aún no se encuentra en el grupo de procesos ejecutables. Típicamente, se trata de un nuevo proceso que no ha sido cargado en memoria principal, aunque su PCB si ha sido creado.
  Cuando un proceso se encuentra en estado Nuevo, el programa permanece en almacenamiento secundario (normalmente en disco).
- **Finalizado/Saliente.** Un proceso que ha sido liberado del grupo de procesos ejecutables por el SO, debido a que ha sido detenido o abortado por algún motivo.



Las posibles transiciones de un estado del proceso a otro son las siguientes:

- **Nuevo Preparado.** El PCB está creado y el programa está disponible en memoria. La mayoría de los SO fija un límite basado en el número de procesos existentes para evitar que se degrade el rendimiento del sistema.
- **Preparado Ejecutándose.** El planificador de la CPU selecciona un proceso que se encuentra preparado para proceder a su ejecución.
- **Ejecutándose Finalizado.** El proceso finaliza normalmente o es abortado por el SO a causa de un error no recuperable.

- **Ejecutándose Preparado.** Un proceso ha alcanzado el máximo tiempo de ejecución ininterrumpida.
- **Ejecutándose Bloqueado.** El proceso solicita algo al SO por lo que debe de permanecer a la espera.
- **Bloqueado Preparado.** Un proceso de estado Bloqueado se mueve al estado Listo cuando sucede el evento por el cual estaba esperando.
- **Preparado Finalizado.** Se produce cuando un proceso padre finaliza y esto conlleva que finalicen todos su procesos hijos también.
- **Bloqueado Finalizado.** Igual que el anterior.

Control de procesos: Modos de ejecución

- **Modo usuario.** Es el modo menos privilegiado. En este modo el programa (de usuario) solo tiene acceso a:
  - Un subconjunto de los registros del procesador.
  - Un subconjunto del repertorio de instrucciones de la máquina.
  - Un área específica de memoria.
- **Modo núcleo (kernel)**. Es el modo más privilegiado. El programa que se ejecuta en este modo tiene acceso a todos los recursos de la máquina.

Llamadas al sistema

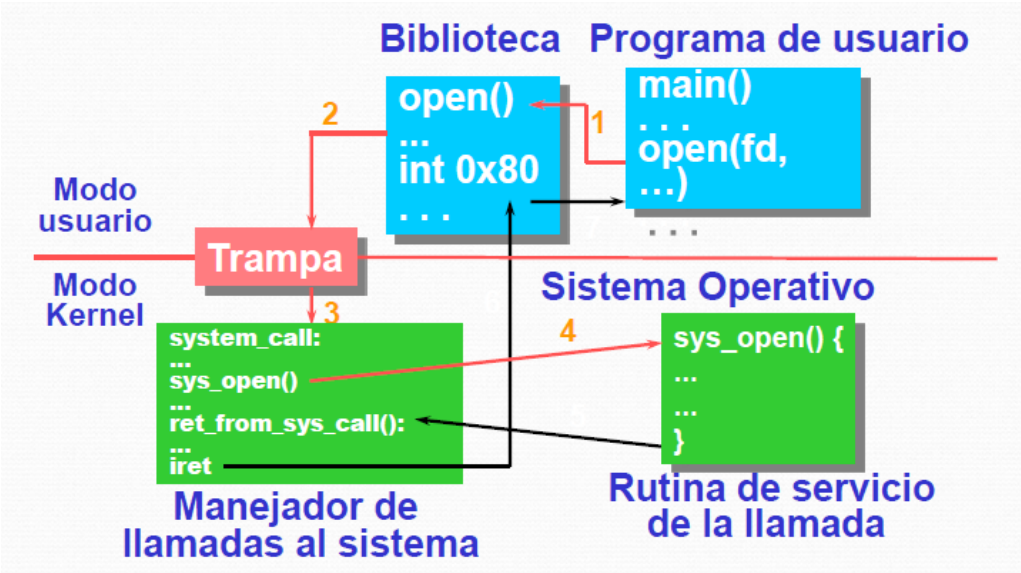
Mediante esta forma se comunican los programas de usuario con el SO en tiempo de ejecución. Son peticiones al SO de petición de servicio.

La llamada al sistema es la forma en la cual un proceso requiere un servicio de núcleo especifico.

Algunos ejemplos son:

- Solicitudes de E/S.
- Gestión de procesos.
- Gestión de memoria.

Se implementan a través de una **“trampa”** o **“interrupción software”**.



Cambio de proceso

¿Cuándo puede realizarse un cambio de proceso? Puede ocurrir en cualquier instante en el que el SO tome el control sobre el proceso actualmente en ejecución. Los posibles momentos en los que esto puede ocurrir son:

- **Interrupcion**
- **Excepción o trap**
- **Llamada al sistema**

Pasos para llevar a cabo dicho cambio:

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

Cambio de modo

¿Cuándo puede realizarse un cambio de modo? Siempre que el SO pueda ejecutarse y solamente como resultado de:

- **Una interrupcion**
- **Una excepción o trap**
- **Una llamada al sistema**

Pasos para llevar a cabo dicho cambio:

1. El **hardware** automáticamente **salva** como mínimo el **PC** y **PSW** y cambia 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**. El **hardware** automáticamente **restaura** en el procesador la información del **PC** y **PSW** previamente **salvada**.

3. Hebras (hilos)

Concepto de hebra (hilo)

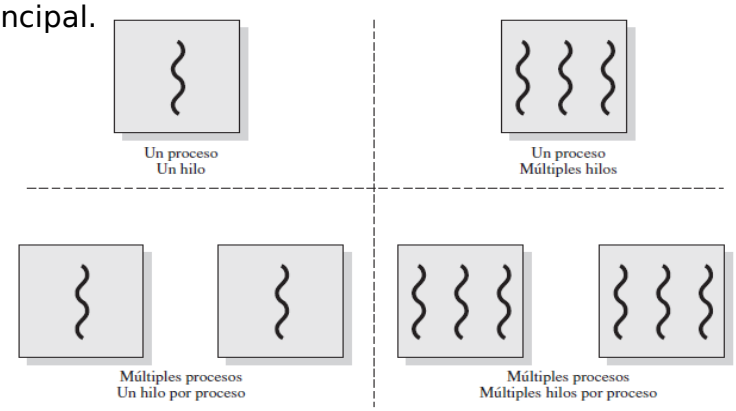
Un hilo es simplemente **una tarea que puede ser ejecutada al mismo tiempo con otra tarea**. Es decir, un proceso con un hilo (monohilo) puede realizar una tarea, pero si deseásemos realizar esa misma tarea otra vez de forma simultánea deberíamos de crear otro proceso. Esto resulta muy costoso, por lo que es más eficiente crear otro hilo perteneciente al mismo proceso (multihilo) que se encargue de realizar dicha tarea.

- 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**.

Ejemplo

Si ejecutamos el **procesador de textos Word** con un solo documento abierto, el programa Word convertido en proceso estará ejecutándose en un único espacio de memoria, tendrá acceso a determinados archivos (galerías de imágenes, corrector ortográfico, etc.) y tendrá acceso al hardware (impresora, disquetera), etc. En definitiva, este proceso, de momento, tiene un **hilo.**

Si en esta situación, sin cerrar el programa, abrimos un nuevo documento, Word no se vuelve a cargar como proceso. Simplemente el programa Word, convertido en proceso, tendrá a su disposición dos hilos o hebras diferentes, de tal forma que el proceso sigue siendo el mismo (el original). Word se está ejecutando una sola vez y el resto de documentos de texto que abramos en esta misma sesión de trabajo, no serán procesos propiamente dichos, sino hilos o hebras del proceso principal.



Estados de los hilos (hebras)

Hay cuatro operaciones básicas relacionadas con los hilos:

- **Creación.** Cuando se crea un nuevo proceso, también se crea un hilo de dicho proceso. Se le proporciona su propio contexto y espacio de pila y se coloca en la cola de Preparados.
- **Bloqueo.** Cuando un hilo necesita esperar por un evento se bloquea.
- **Desbloqueo.** Cuando sucede el evento por el que el hilo estaba bloqueado, el hilo pasa a la cola de Preparados.
- **Finalizado.** Cuando se completa un hilo, se liberan su contexto y su espacio de pila.

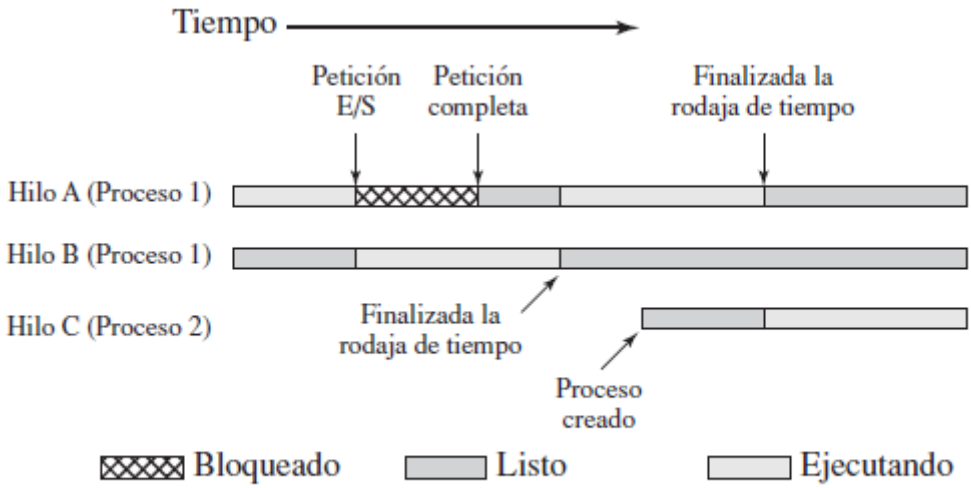


Figura 4.4. Ejemplo multihilo en un uniprocesador.

Ventajas de las hebras (hilos)

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

- **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.
- Lleva **menos tiempo cambiar entre dos hilos** dentro del mismo proceso que cambiar de proceso.
- Los hilos **facilitan la comunicación entre diferentes programas** que se están ejecutando.
- Permiten aprovechar las técnicas de **programación concurrente y el multiprocesamiento simétrico**.

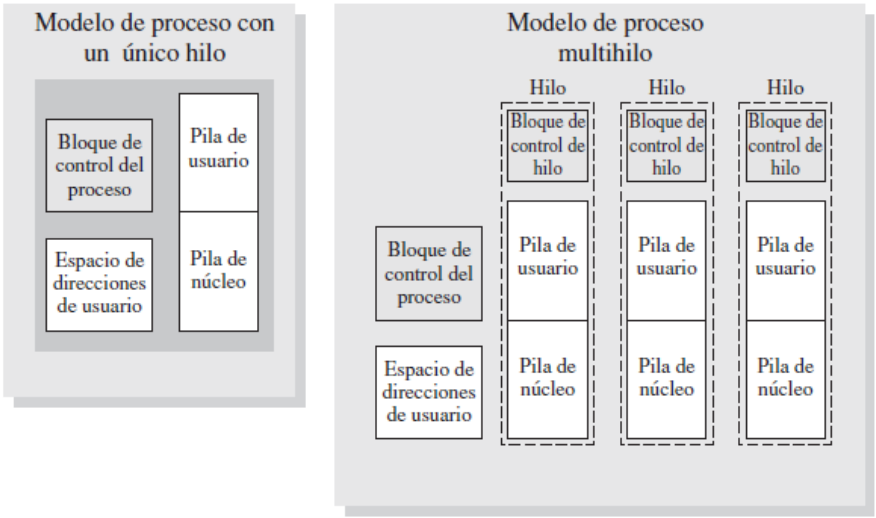
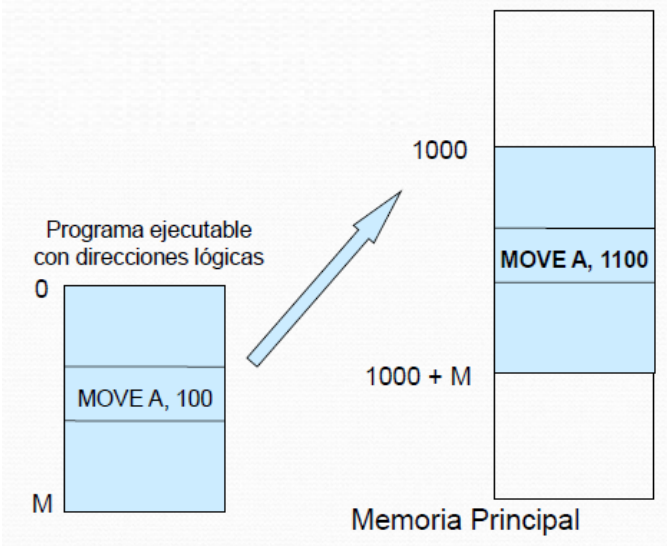


Figura 4.2. Modelos de proceso con un único hilo y multihilo.

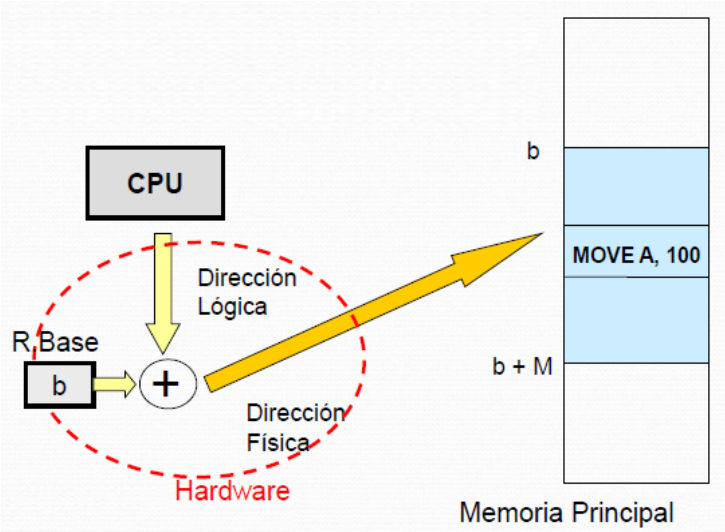
4. Gestió

- **Carga ab** cargarse siempre en la misma ubicación de la memoria principal. Esto conlleva que el programa no es reubicable.
- **Reubicación**. Capacidad de cargar y ejecutar un programa en un lugar arbitrario de la memoria. Se debe permitir que los programas se puedan mover en la memoria principal, ya que **sería bastante limitante** tener que colocarlos en la misma región de memoria donde se hallaba anteriormente.
  - **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.

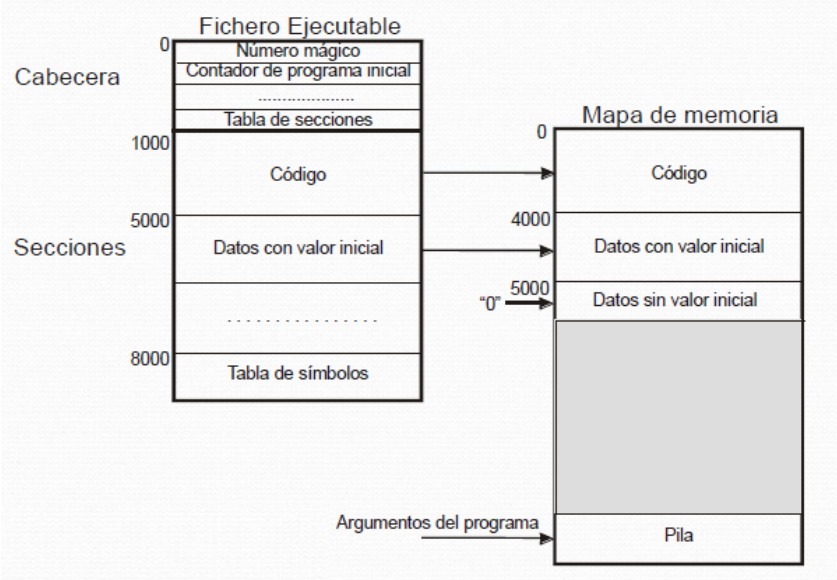


- **Reubicación dinámica**.
  - El compilador genera direcciones lógicas (relativas) de 0 a M.
  - La traducción de direcciones lógicas a físicas se realiza en tiempo de ejecución luego el programa está cargado con referencias relativas.
  - Requiere apoyo hardware.



Espacio 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**. Estructura de datos (que reside en memoria) que indica el tamaño total del espacio de direcciones lógico y la correspondencia entre las direcciones lógicas y las físicas.



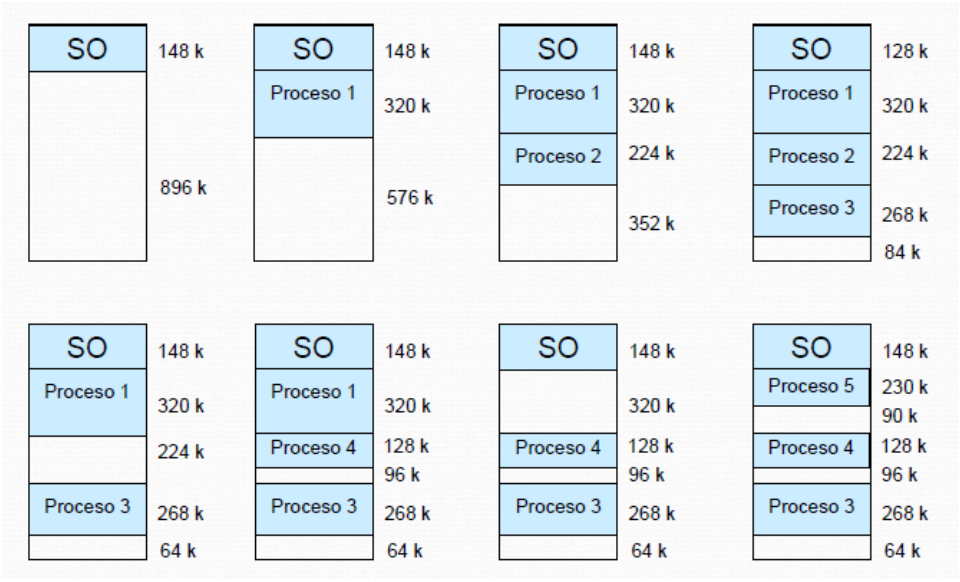
Ejemplo de mapa de memoria de un proceso

Fragmentación

La fragmentación es la **memoria** que queda **desperdiciada** al usar los métodos de gestión de memoria.

Solución a la fragmentación de memoria

- **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.



Ejemplo de fragmentación de memoria