Tema 2: Procesos e hilos

Sistemas Operativos

Javier Pérez Rodríguez
javier.perez[at]uco.es

Departamento de Informática y Análisis Numérico
Universidad de Córdoba

23 de octubre de 2024

Programa vs Proceso

- Un **programa** es una unidad inactiva almacenada en un disco en forma de archivo.
- Un **proceso** es una actividad cargada en memoria principal formada por un único hilo secuencial de ejecución, un estado y un conjunto de datos y recursos del sistema asociados (contexto de un proceso).

Contexto de un Proceso

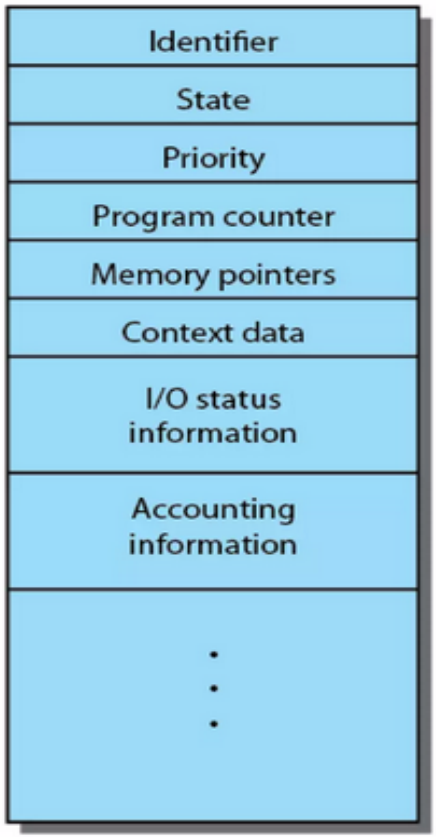


Figura: Process Control Block

- El contexto de un proceso es un conjunto de datos que permite controlarlo y supervisarlo.
- Este contexto se almacena en una estructura de datos llamada Bloque de Control de Proceso (en inglés, PCB).

Información almacenada en el PCB

- Identificación del proceso.
- Identificación del proceso padre.
- Estado del proceso: R, B, T, W, Z.
- Información de planificación: nivel de prioridad del proceso.
- Descripción de los segmentos de memoria (RAM) que tiene asignados.
- Punteros a memoria: punteros al código del programa y datos asociados.
- Datos de contexto en relación a los registros de la CPU: registros del procesador, banderas de estado, señales, datos temporales en registro...

Información almacenada en el PCB

- Información de estado de E/S: peticiones pendientes, dispositivos, ficheros usados...
- Comunicación entre procesos: IPC.
  - Memoria compartida.
  - Señales.
  - Semáforos.
  - Paso de mensajes: pipelines o colas
  - Paso de mensajes en red: sockets, MPI.
- Información de auditoría: ciclos utilizados, cuánto le queda por ejecutar...

Función del PCB

- El PCB contiene suficiente información para que un proceso pueda ser interrumpido mientras se ejecuta en el procesador y, posteriormente, su estado de ejecución sea restaurado en el mismo estado.
- El PCB es la entidad que permite que el sistema operativo soporte la multiprogramación, es decir, la ejecución simultánea de varios procesos.
- Un PCB se crea cuando un proceso es aceptado en el planificador a largo plazo.
- Se define como imagen de un proceso al conjunto de datos, código, pila (*stack*), montículo (*heap*) y PCB asociados a un proceso.

Monoprocesador y Mononúcleo

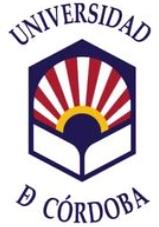
- En un computador monoprocesador y mononúcleo, solo se puede ejecutar un programa a la vez.
- La conmutación entre procesos da lugar a la multiprogramación.
- El componente encargado de la conmutación entre procesos es el activador o **dispatcher**.

Situaciones que activan el dispatcher

El **dispatcher** entra en acción cuando ocurre alguna de las siguientes situaciones:

- Interrupciones (de tiempo o de cualquier otro tipo).
- Solicitud de operaciones bloqueantes (invocación de `wait()`).
- Solicitud de operaciones de Entrada/Salida (E/S).

El planificador a corto plazo es el que le indica al *dispatcher* qué proceso debe introducir en la CPU. Es decir, en realidad es un proceso que usa el planificador a corto plazo.
Gracias al PCB, el *dispatcher* puede introducir un proceso nuevo u otro que continúe por donde iba.



Ejemplo de ejecución del dispatcher

- Suponga tres procesos: A, B, C. Los tres residen en memoria principal y representan a tres programas. De manera adicional, existe un pequeño programa activador (*dispatcher*) que forma parte del núcleo del sistema y que intercambia procesos en el procesador.
- En este ejemplo, se asume que el sistema operativo sólo deja que un proceso continúe durante 6 ciclos de instrucción, después de los cuales se interrumpe por alarma de reloj (temporizador). Esto previene que un solo proceso monopolice el uso del tiempo del procesador y provoque inanición en los demás.

Ejemplo de ejecución del dispatcher

Tenga en cuenta los siguientes supuestos:

- La dirección de memoria de comienzo de A es 5000, la de B es 8000 y la de C 12000.
- Los procesos A y C ejecutan 12 instrucciones, y B ejecuta 4 instrucciones.
- El activador es un pequeño programa con 6 instrucciones con dirección de memoria de comienzo en 100.
- El sistema operativo sólo deja que un proceso continúe durante 6 ciclos de instrucción, después de los cuales se interrumpe por alarma de reloj (temporizador). Esto previene que un solo proceso monopolice el uso del tiempo del procesador y provoque inanición en los demás.

Por simplicidad se omite:

- la rutina ISR que trata la alarma de reloj.
- el planificador a corto plazo.

1er ciclo:

- A (5000 - 5005) RELOJ
- Planificador
- Dispatcher (100 - 105)

2º ciclo:

- B (8000 - 8003) TERMINADO
- Planificador
- Dispatcher (100 - 105)

3º ciclo:

- C (12000 - 12005) RELOJ
- Planificador
- Dispatcher (100 - 105)

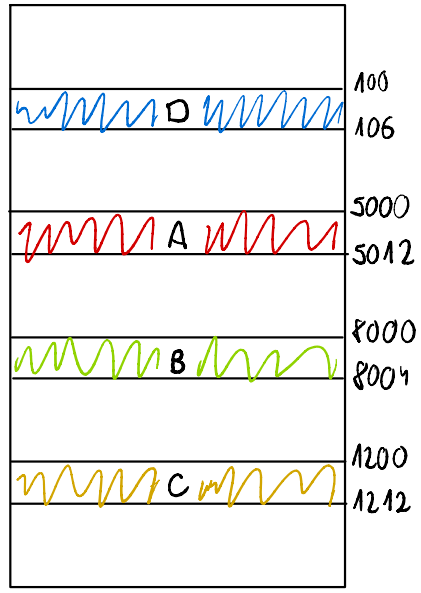
4º ciclo:

- A (5006 - 5012) TERMINADO
- Planificador
- Dispatcher (100 - 105)

5º ciclo:

- C (12006 - 12012) TERMINADO

Memoria:



D: 100 - 105

A: 5000 - 5011

B: 8000 - 8003

C: 12000 - 12012

Arranque del sistema

Etapa 1: Ejecución del programa de arranque

- Se ejecuta el programa de arranque almacenado en ROM o EEPROM, conocido como **firmware** o **BIOS**.
- La **BIOS** identifica y diagnostica los dispositivos del sistema, y detecta el disco duro para leer el cargador de arranque.
- La CPU ejecuta la BIOS, sabiendo su dirección de arranque en la ROM.
- Actualmente, se utiliza la **UEFI, Unified Extensible Firmware Interface**, que es más rápida que la BIOS tradicional.

Arranque del sistema

Etapa 2: Cargador de arranque de segunda etapa

- La BIOS ejecuta el gestor de arranque de segunda etapa, que suele estar en el primer sector del dispositivo de arranque, llamado **Master Boot Record** (MBR).
- Un disco con una partición marcada como de arranque se denomina **disco de arranque**.
- La CPU busca en el MBR y lo carga en memoria principal.

Arranque del sistema

Etapa 3: Ejecución del gestor de arranque

- El gestor de arranque explora el sistema de archivos para localizar el **kernel** del sistema operativo.
- Carga el kernel en memoria principal.

Arranque del sistema

Etapa 4: Ejecución del Kernel

- El sistema operativo establece el espacio de usuario.
- En GNU/LINUX se ejecuta el proceso `init()`, que lanza varios servicios y scripts, tales como:
  - Montaje de otras particiones o discos.
  - Inicialización del servicio de red.
  - Inicio del entorno gráfico.
  - Control de logins y sesiones de usuario.
  - Inicialización de Bluetooth.
  - Inicialización del servicio de impresión.
- Hoy en día, muchas distribuciones han migrado a *systemd*, un sistema de inicio más moderno y complejo. *Systemd* sigue utilizando un proceso con PID 1, pero gestiona los servicios de una forma diferente, utilizando unidades declarativas en lugar de scripts de shell tradicionales.

Creación de procesos I

Los procesos pueden crearse de varias maneras:

1. Arranque del sistema: Los demonios y servicios se ejecutan en el espacio de usuario y además pueden acceder al núcleo para realizar las tareas necesarias.
2. Mediante invocación interna desde un proceso de un ejecutable.

Creación de procesos II

3. Mediante una llamada al sistema, como por ejemplo `fork()`.
4. En sistemas interactivos, la creación de procesos puede realizarse mediante un doble clic del ratón o a través de la invocación desde un terminal.

Terminación de procesos

Un proceso puede terminar por varias razones:

- **Salidas voluntarias**:
  - **Salida normal**: El proceso ha concluido su trabajo (por ejemplo, `exit()` en POSIX).
  - **Salida controlada por error**: Se produce cuando el proceso detecta un error predecible, como intentar compilar un archivo con un nombre incorrecto.
- **Salidas involuntarias**:
  - **Error fatal**: Puede ser debido a una instrucción ilegal, referencia a una zona de memoria no válida o una división por 0.
  - **Eliminado por otro proceso**: Otro proceso puede ejecutar una llamada al sistema para eliminarlo, como la función `kill()`, o si se cierra el terminal desde el que fueron lanzados.

Modelo de estados para procesos

Una de las principales tareas del SO es controlar la ejecución de los procesos y asignarle recursos. Para ello se debe valer de un modelo de comportamiento para los procesos.

Modelo para cinco estados

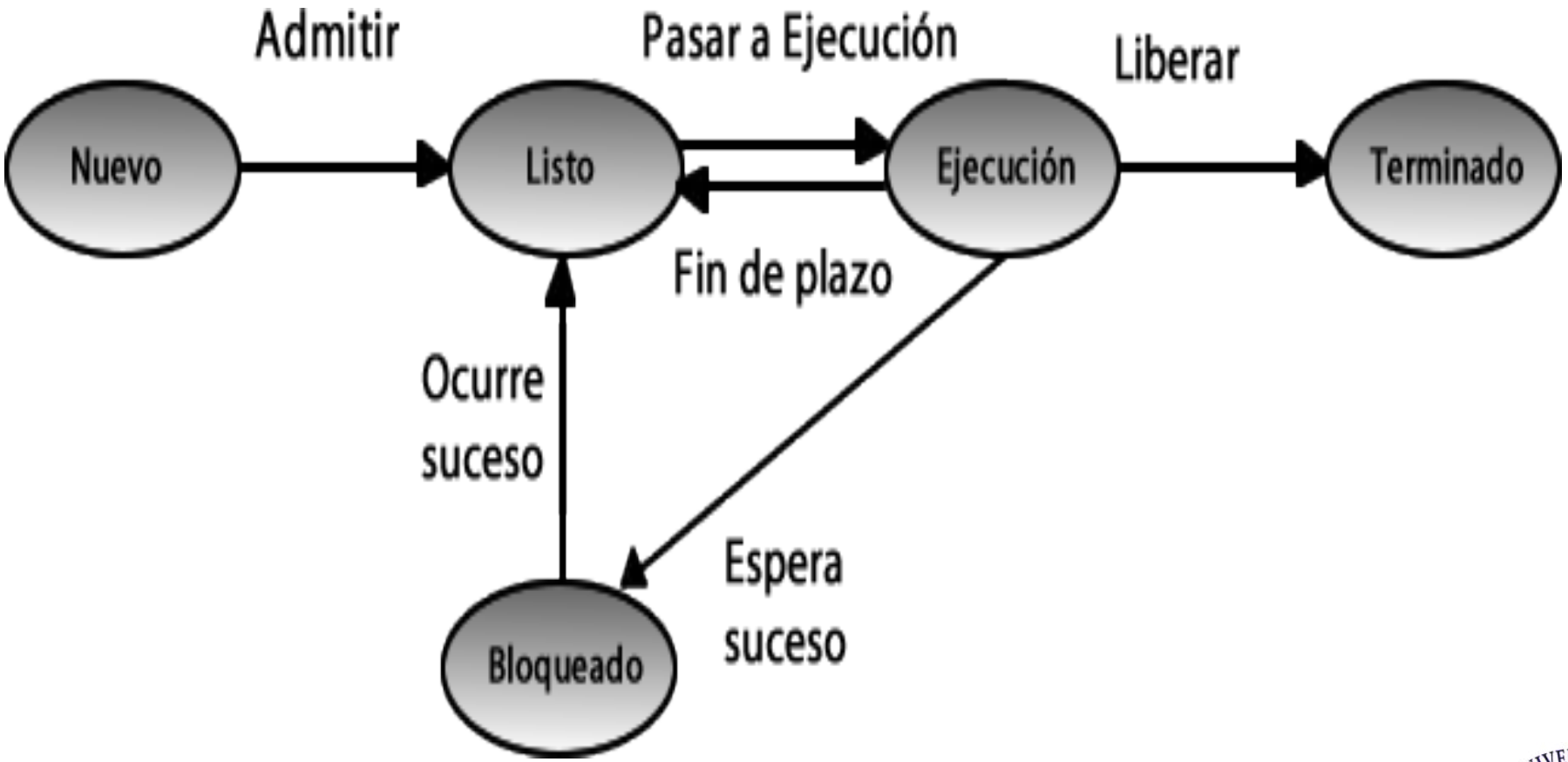


Figura: Modelo de procesos de cinco estados

Estado: Nuevo

Procesos recien creados pero que aún no han sido admitidos para ser ejecutables. No han sido cargados en memoria principal, aunque su BCP si ha sido creado. Mediante alguna de las formas de creación de procesos, el SO le asocia un ID a dicho proceso y se construyen todas aquellas tablas que se necesiten para gestionarlo.

Estado: Ejecutando

Proceso que se está actualmente en ejecución. Sólo un proceso puede estar en este estado en un instante determinado.

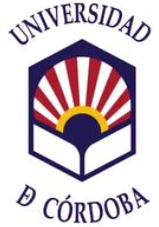
Estado: Listo

Proceso que está preparado para ejecutar cuando tenga oportunidad y lo decida el planificador.

Estado: Bloqueado

Proceso que no puede ejecutar hasta que se cumpla un evento determinado:

- una operación E/S que tenía pendiente.
- la comunicación de otro proceso al que espera mediante una llamada bloqueante para poder continuar (por ejemplo, *wait()*).

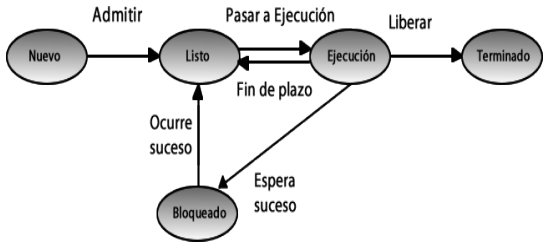


Estado: Terminado

Proceso que ha sido liberado del grupo de procesos ejecutables por el SO debido a que ha terminado por alguna razón voluntaria o involuntaria.

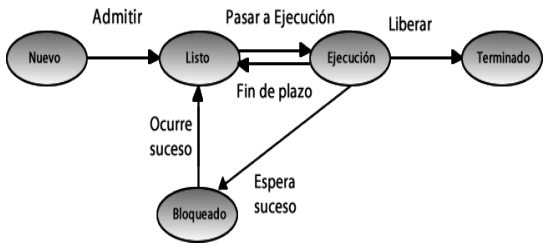
Transiciones

- Null → Nuevo: Mediante alguna acción vista en la diapositiva 15.
- Nuevo → Listo: Cuando haya suficientes recursos disponibles.
- Listo → Ejecutando: Según política de planificación.
- Ejecutando → Terminado: Mediante alguna acción vista en la diapositiva 17.



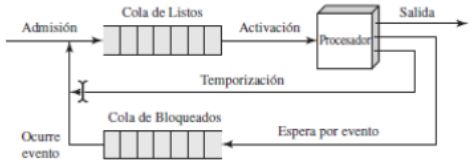
Transiciones

- Ejecutando → Listo: Lo habitual, por política de planificación (interrupción por alarma de reloj), aunque podría ser para atender una interrupción de E/S o por política de prioridades.
- Ejecutando → Bloqueado: Si el proceso solicita algo por lo cual debe esperar o no está disponible (operación E/S, *wait()*, comunicación de procesos).
- Bloqueado → Listo: Cuando sucede el evento por el cual estaba esperando.
- Listo → Terminado y Bloqueado → Terminado: *kill()*.



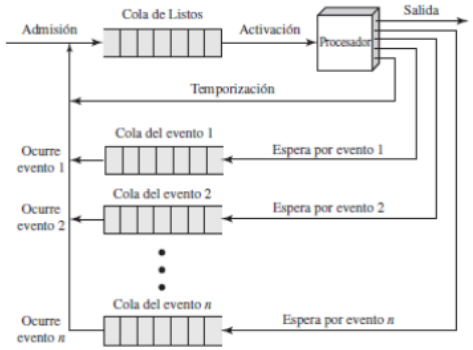
Esquema de colas por estados

- Cada proceso admitido por el SO se coloca en la cola de *Listos*.
- Cuando llega el momento de que el SO seleccione un proceso a ejecutar lo escoge de esta lista. En ausencia de un esquema de prioridad, esta cola puede ser una lista de tipo FIFO.
- Cuando el proceso en ejecución termina de utilizar el procesador o bien finaliza o bien se coloca en la cola de *Listos* o de *Bloqueados*, dependiendo de las circunstancias.
- Cuando sucede un evento, cualquier proceso en la cola de *Bloqueados* que únicamente esté esperando a dicho evento, se mueve a la cola de *Listos*.



Esquema de colas por estados

- Esta última transición significa que, cuando sucede un evento, el SO debe recorrer la cola entera de *Bloqueados*, buscando aquellos procesos que estén esperando por dicho evento.
- En los SO con muchos procesos, esto puede significar cientos o incluso miles de procesos en esta lista, por lo que sería mucho más eficiente tener una cola por cada evento.
- De esta forma cuando sucede un evento, la lista entera de procesos de la cola correspondiente se movería al estado de Listo.



Modelo para siete estados

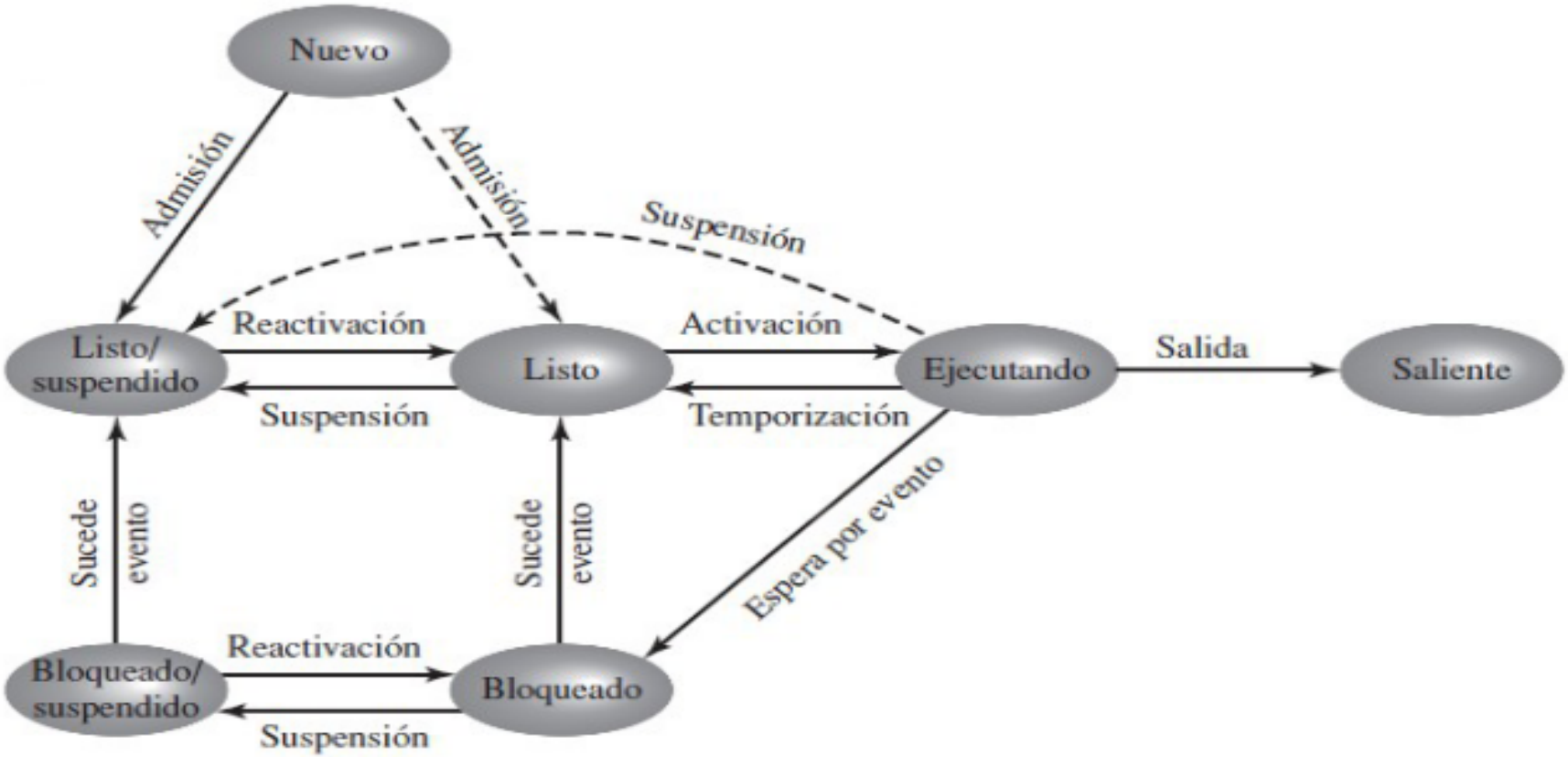


Figura: Modelo de procesos de siete estados

Modelo para siete estados

En un sistema con memoria virtual, los procesos deben estar en memoria principal (RAM) para ser ejecutados. Sin embargo, cuando la RAM está llena o se necesita liberar espacio para procesos prioritarios, algunos procesos pueden ser enviados a la memoria secundaria (*swap*) con excepción de su BCP (Bloque de Control de Proceso).
Se generan dos nuevos estados para los procesos suspendidos:

- Listo/Suspendido: El proceso está en memoria secundaria y no puede ejecutarse hasta que no se cargue nuevamente en RAM.
- Bloqueado/Suspendido: El proceso está en memoria secundaria esperando a que ocurra un evento para continuar.

Modelo para siete estados

Cuando el SO ha realizado una operación de *swap*, tiene dos opciones para seleccionar un nuevo proceso para traerlo a estado *Listo*:

1. Puede admitir un *Nuevo* proceso que se haya creado por parte del planificador.
2. Puede traer un proceso que anteriormente se encontrase en estado de *Suspendido*.

Parece que sería preferible traer un proceso que anteriormente estuviese *Suspendido*, para proporcionar dicho servicio en lugar de incrementar la carga total del sistema, pero eso dependerá de la política de planificación y de los recursos existentes.

Hilos

- Hasta este momento se ha presentado el concepto de un proceso como:
  - Propietario de recursos. Un proceso incluye un espacio de direcciones virtuales para el manejo de la imagen del proceso. El sistema operativo realiza la función de protección para evitar interferencias no deseadas entre procesos en relación con los recursos.
  - Ejecución. Un proceso sigue una ruta de ejecución (traza) de un programa. Los procesos pueden ser intercalados en su ejecución. El SO planifica y activa la ejecución de procesos en base a prioridad y estado.
- Sin embargo, estas dos características son independientes y podrían ser tratadas como tales por el SO.

Hilos

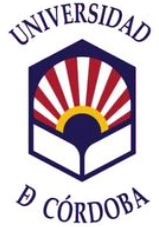
Un **hilo** (proceso ligero o hebra) es una unidad básica de utilización de la CPU. La mayoría de los SO permiten que un proceso (pesado) tenga múltiples hilos.

- Cada hilo perteneciente a un proceso posee lo siguiente de manera individual:
  - Un estado de ejecución por hilo.
  - Un contexto propio. Un hilo se puede ver como un contador de programa independiente dentro de un proceso.
  - Una pila de ejecución por hilo, con sus correspondientes registros de activación: dirección de retorno, parámetros, valor devuelto, variables locales, etc...

Hilos

Información compartida con el proceso al que pertenece y con los restantes hilos de dicho proceso:

- Zona de código, ya que los hilos se crean desde el mismo código del proceso padre.
- Zona de datos estáticos. Deben ser protegidos adecuadamente.
- Montón. La reserva de memoria que haga un proceso padre o un hilo es visible para el resto de hilos.
- Otros recursos asociados al proceso: Señales y manejadores de señales, directorio de trabajo actual, variables de entorno del proceso.



Hilos

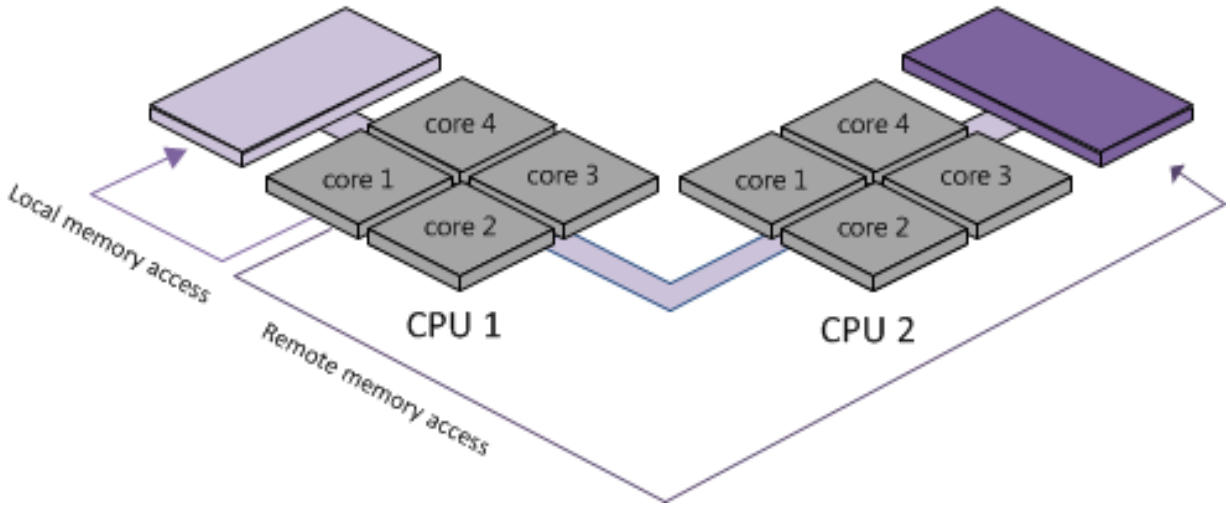


Figura: CPUs con múltiples cores.

Hilos

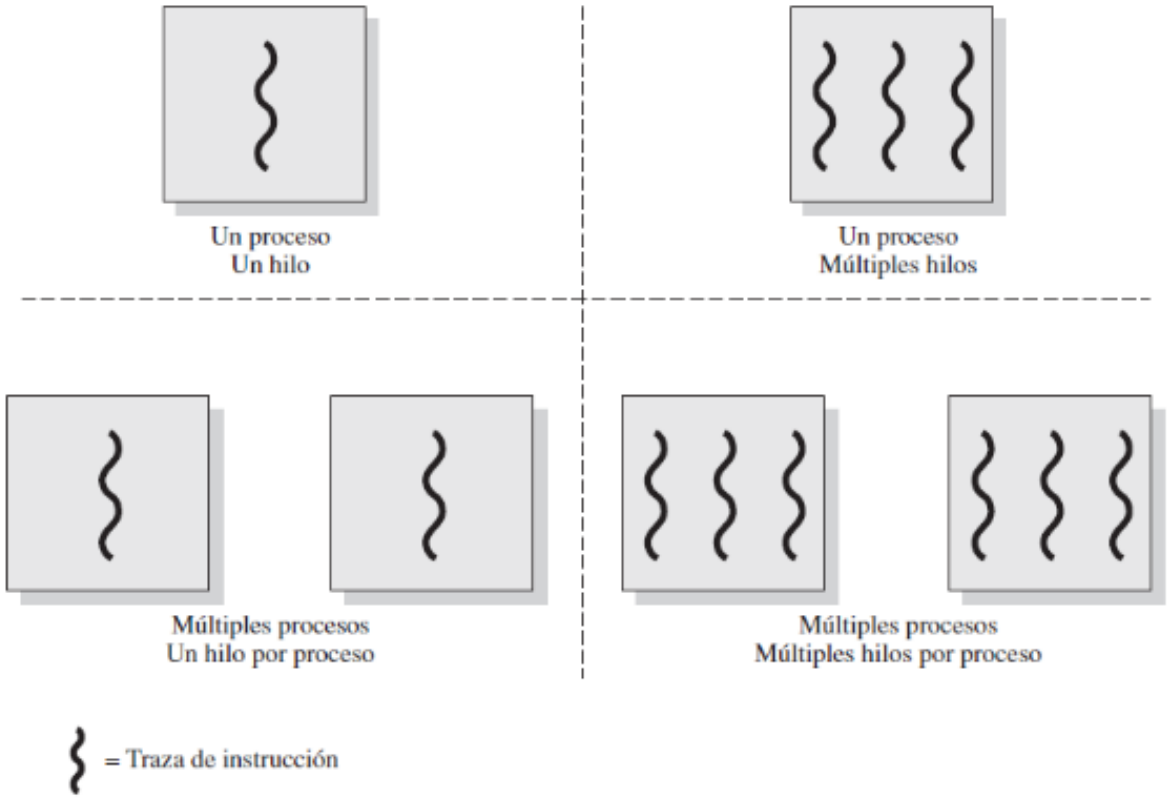


Figura: Distintos entornos de ejecución.

Ejemplos del uso de hilos

- Servidor web: un hilo para cada petición.
- Procesador de textos: trabajos en primer y segundo plano.

Planificación y activación de hilos

- En un SO que soporta hilos, la planificación y activación se realizan a nivel de hilo.
- Los principales estados de los hilos son:
  - Ejecutando
  - Listo
  - Bloqueado

Gestión de hilos y procesos

- Existen acciones que afectan a todos los hilos de un proceso.
- El SO debe gestionar estas acciones a nivel de proceso.
- Suspender un proceso implica liberar el espacio de direcciones en memoria principal para hacer espacio para otro proceso.

Suspensión y finalización de procesos

- Todos los hilos de un proceso comparten el mismo espacio de direcciones.
- Al suspender un proceso, todos los hilos se suspenden al mismo tiempo.
- La finalización de un proceso, voluntaria o involuntaria, termina todos los hilos asociados al proceso.
- Esto ocurre porque todos los hilos están en el mismo espacio de direcciones que es liberado.

Motivación y ventajas de los hilos

- Implementar una aplicación con un conjunto de hilos es mucho más eficiente que hacerlo con procesos independientes.
- Las hebras permiten una mejor utilización de los recursos del SO y una mayor rapidez en la ejecución y comunicación.

La programación multihilo presenta una serie de ventajas debido a:

1. Utilización sobre arquitecturas multiprocesador.
2. Compartición de recursos del proceso.
3. Tiempo de ejecución.
4. Comunicación más eficiente.

Utilización en arquitecturas multiprocesador

- Los hilos pueden ejecutarse en paralelo en diferentes procesadores o núcleos.
- El kernel del SO mantiene información del proceso y de los hilos individuales.
- La planificación realizada por el SO se hace a nivel de hilo.
- El uso de múltiples hilos en una aplicación permite que un proceso continúe ejecutándose incluso aunque parte de él esté bloqueado o realizando una operación muy larga.
- Ejemplo: Un navegador web puede permitir interacción con el usuario mientras carga una imagen en otro hilo.

Compartir de recursos

- Los hilos comparten parte de la memoria y recursos del proceso al que pertenecen.
- No es necesaria la duplicación de memoria ni de recursos para cada hilo.

Ventajas en tiempo de ejecución

En general, se consume mucho más tiempo en crear y gestionar los procesos que las hilos:

- Gestionar procesos es mucho más lento que hacerlo con hilos. Crear un proceso es treinta veces lento que crear una hebra
- Crear o terminar un hilo es más rápido que un proceso, ya que comparten recursos del proceso padre.
- El cambio de contexto entre hilos es más rápido que entre procesos. El cambio de contexto es aproximadamente cinco veces más lento a nivel de proceso que a nivel de hilo.

Eficiencia en comunicación

- La comunicación entre hilos es más rápida que entre procesos.
- Los hilos pueden compartir memoria sin necesidad de invocar al núcleo del SO.
- En los procesos, suele intervenir el núcleo a través de llamadas como IPC o memoria compartida.