Tema 2

Procesos

3º I.T.I.

Gestión

Contenido

Tema 2. Procesos

2.1 Concepto de proceso

2.2 Estados de un proceso

- Transiciones de estado

2.3 Descripción de procesos

2.4 Control de procesos

2.5 Operaciones con procesos

2.6 Threads

2.7 Ejemplos:

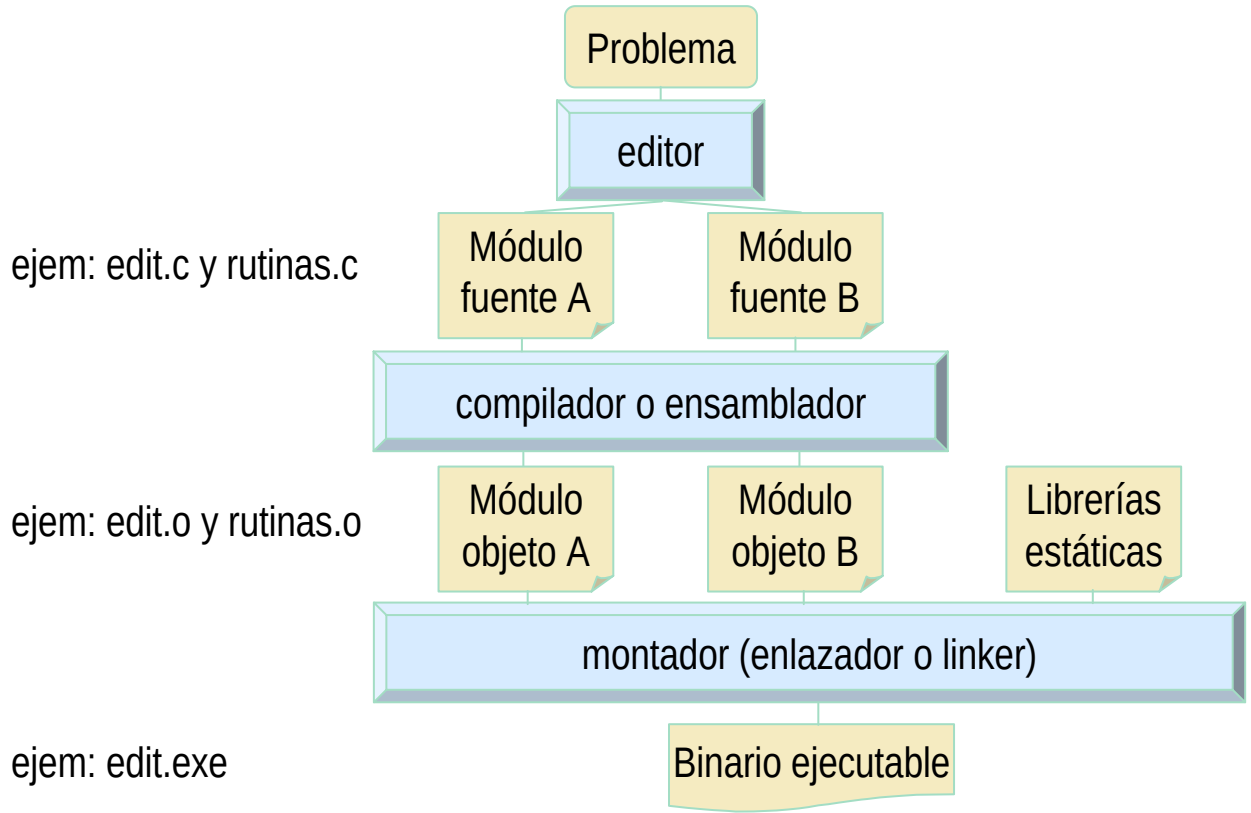
- Descripción de procesos en UNIX, Linux y Windows XP

2.1 Concepto de proceso

2.1 Concepto de proceso

¿qué es un programa?

- Secuencia de instr. que implementan un algoritmo (un problema a resolver)
- Generación de un programa ejecutable:



2.1 Concepto de proceso

ejecución de un programa

- Ejecución de un programa
  - Una instancia del programa: “C:> edit <enter>”
    - Un programa del SO (el cargador) busca espacio en Mem
    - Carga el código y los datos. Reserva espacio para la pila

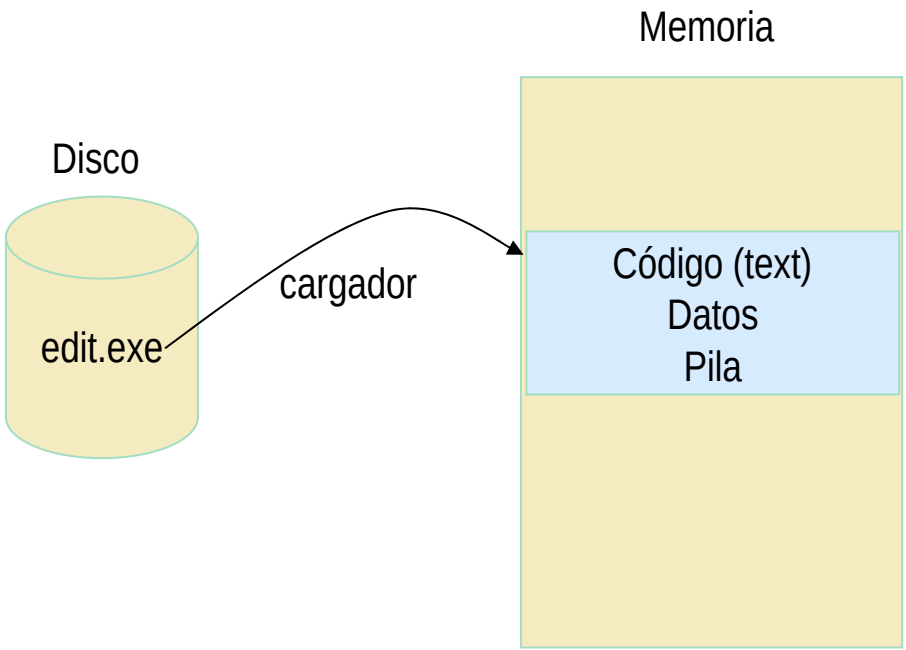
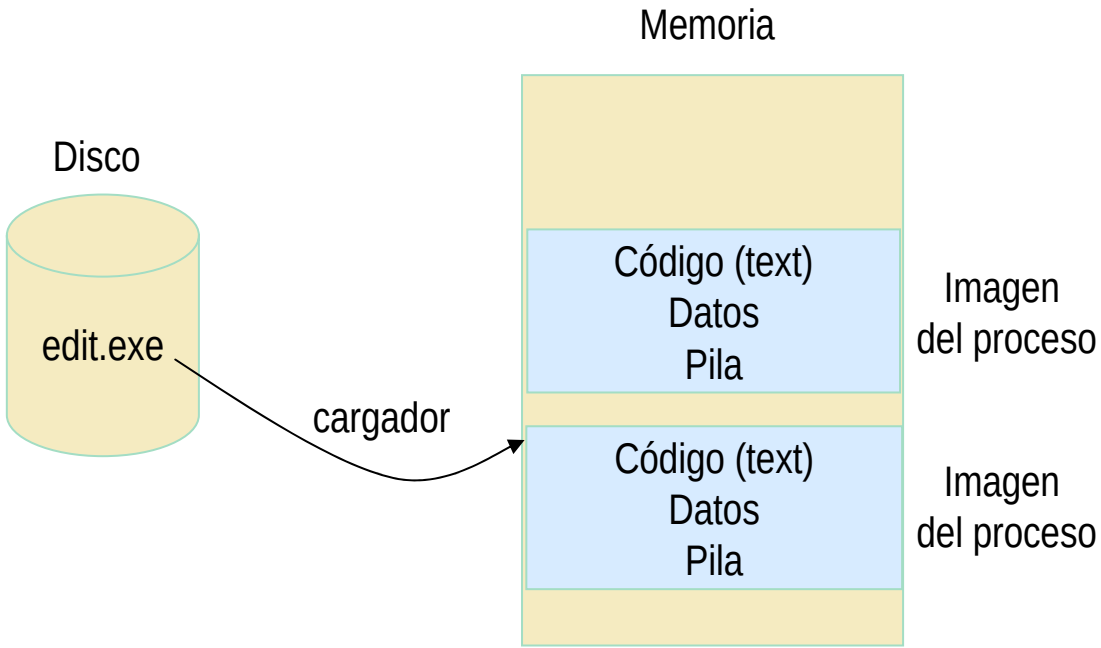


Imagen en memoria del proceso: Información que mantiene el proceso en el mapa de memoria

2.1 Concepto de proceso

ejecución de un programa

- Ejecución de un programa
  - Otra instancia del programa: “C:> edit <enter>”
    - Necesitas una región de datos y pila diferentes
    - Los SO antiguos también replican el código
      - Pero no es necesario si éste no se modifica durante la ejecución



2.1 Concepto de proceso

¿qué es un proceso?

- Definiciones:
  - Programa en ejecución.
  - Instanciación de un programa
- Suelen ser intercambiables los términos: proceso (process), tarea (task) y trabajo (job).
- Otras definiciones:
  - Elemento SW susceptible de ser planificado
  - Actividad asincrónica
  - “Espíritu animado del procedimiento” ☺

Operating Systems. H.M. Deitel. Addison Wesley.1990.

2.1 Concepto de proceso

¿por qué utilizar procesos?

- El mejor modo de resolver un problema complejo es dividirlo en subproblemas más pequeños.
- En un SO, y otros programas complejos, existen actividades que ocurren a la vez: 10 solicitudes de disco, 2 mensajes de red, 7 pulsaciones de teclas, 20 aplicaciones, ...
- Para programar tal sistema:

  (a) un programa que lo gestiona todo: ! La producción del bucle está limitada por la función más lenta ¡ Esto NO es aceptable en un SO real (imagina: entorno de ventanas que no atiende al ratón mientras dibuja una ventana).

  (b) aislar cada actividad en un proceso.

2.1 Concepto de proceso

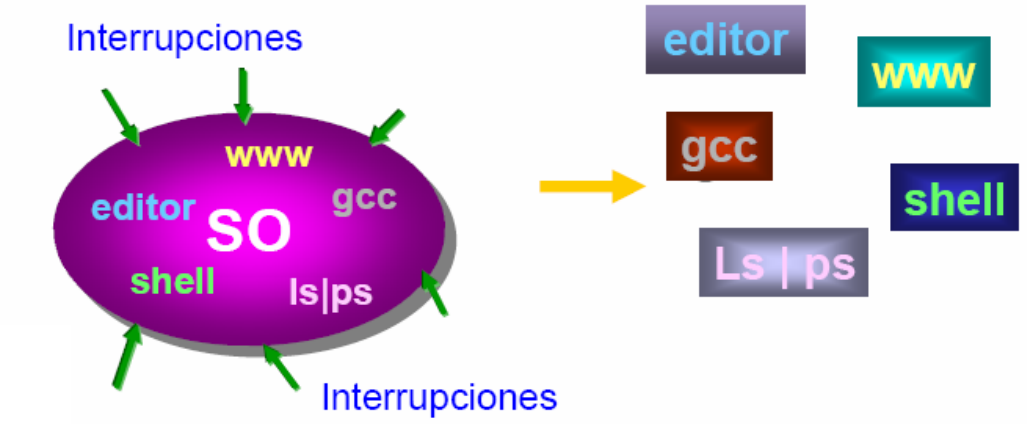
¿por qué utilizar procesos?

- Simplicidad: en un sistema existen muchas operaciones independientes (gcc, lpr, edit,..) que podemos envasar cada una en un proceso, acotando así sobre lo que tenemos que razonar.
- Velocidad: si un proceso se bloquea (esperando por disco, teclado, red,...) cambiamos a otro, como si tuviésemos más de una CPU. De esta forma, los SOs operativos modernos permiten el seudoparalelismo en la ejecución de varios procesos: parecen ejecutarse simultáneamente.
- Seguridad: limitamos los efectos de un error.

2.1 Concepto de proceso

¿por qué utilizar procesos?: simplicidad

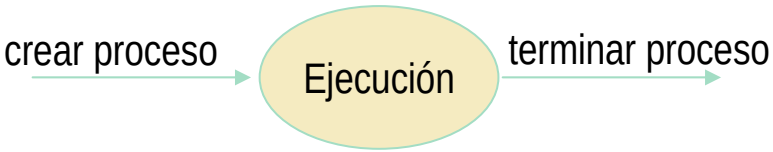
- Aislando cada actividad en un proceso: tratamos con un proceso cada vez, y los procesos sólo tratan con el SO.



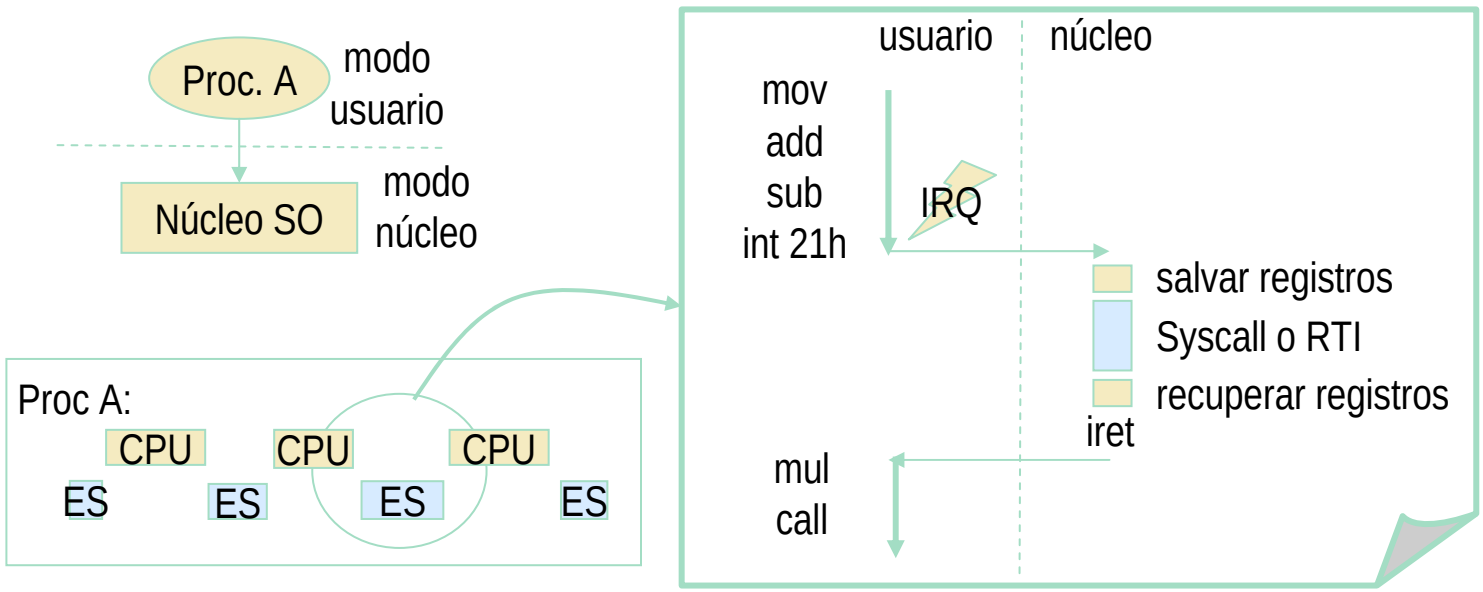
2.1 Concepto de proceso

¿por qué utilizar procesos?: velocidad

- Monotarea:
  - Sólo cuando acaba un proceso puede ejecutarse otro



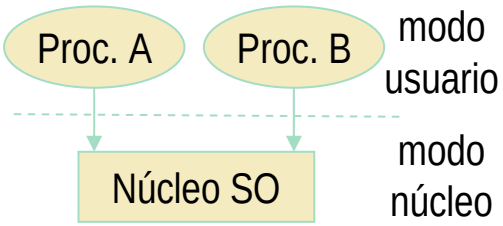
  - El proceso tiene ráfagas de CPU y de E/S
    - Las operaciones de E/S las "sirve" el SO mediante syscalls
    - Durante la ejecución del proc. pueden llegar IRQs que atiende el SO

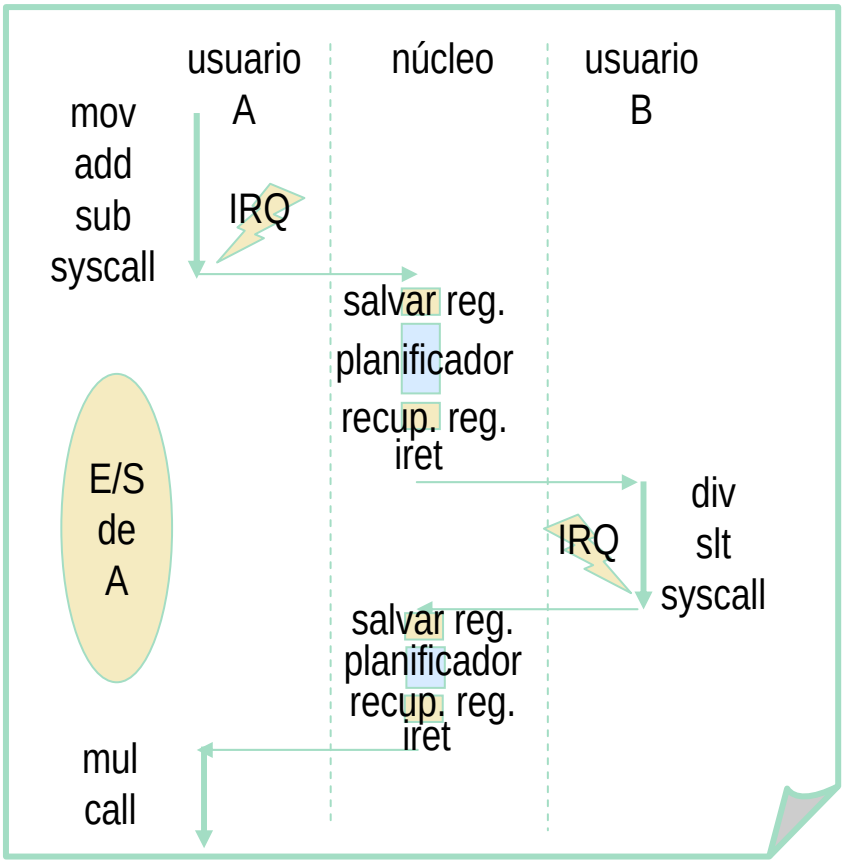


2.1 Concepto de proceso

¿por qué utilizar procesos?: velocidad

- Multiprogramación
  - La memoria almacena varios procesos
    - Grado de multiprogramación: número de procesos activos
  - Paralelismo entre CPU y E/S (gracias al DMA)





- Cambio de contexto
  - Iniciado por
    - Syscall (Int SW)
      - » E/S
      - » exit()
    - IRQ (Int HW)
      - » E/S, reloj

2.1 Concepto de proceso

¿por qué utilizar procesos?: seguridad

- Vamos a hacer creer a los programas que están solos en la máquina.
- Cada proceso se ejecuta en SU propio espacio de direcciones –> no puede acceder directamente a espacios de direcciones de otros procesos.
- Su ejecución esta confinada a su espacio y también sus errores.
- Coste: compartir información se complica

2.1 Concepto de proceso

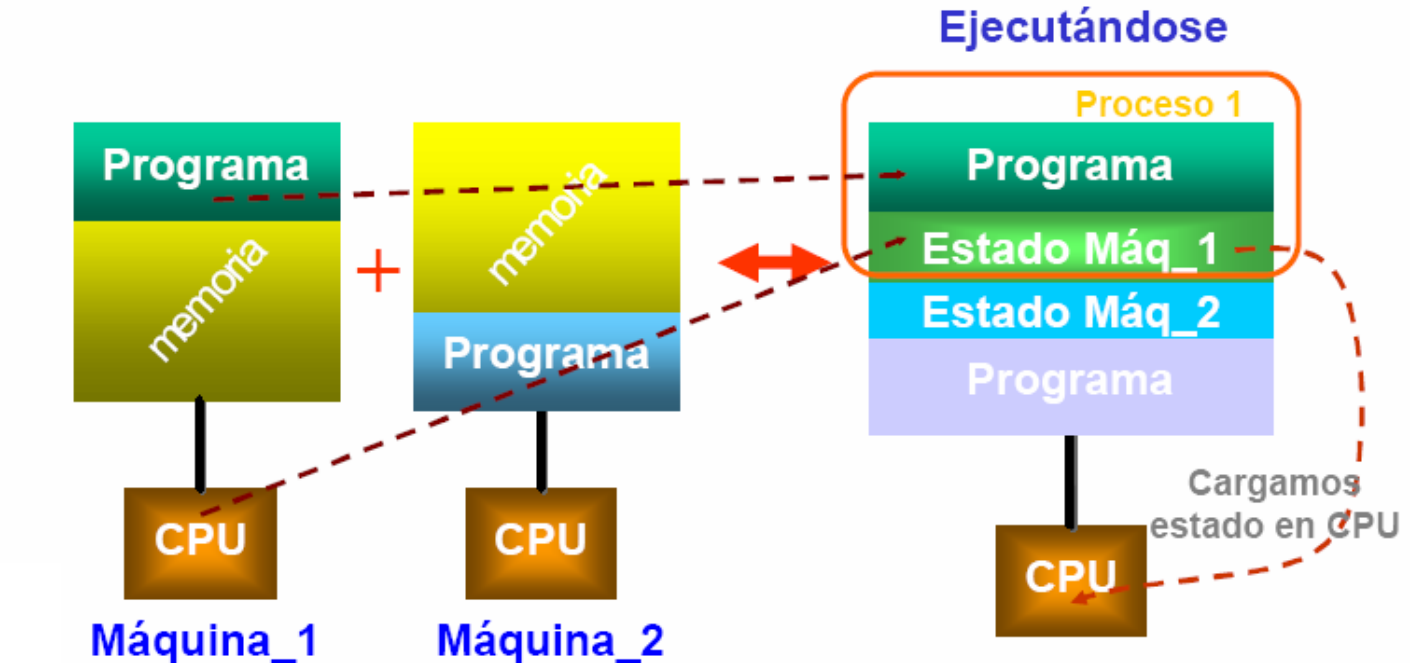
proceso = CPU virtual

- **Proceso** = “programa en ejecución” = un flujo secuencial de ejecución en su propio espacio de direcciones. Puede ser trazado.
- Un proceso es un recurso virtual – una abstracción que desacopla la CPU física del recurso de computo presentado al programa - creamos la ilusión de tener más de una CPU.

2.1 Concepto de proceso

proceso = CPU virtual

- Multiplexamos la CPU en el tiempo, hacemos creer a cada proceso que es único en la máq.



¿Qué hay en un proceso?:

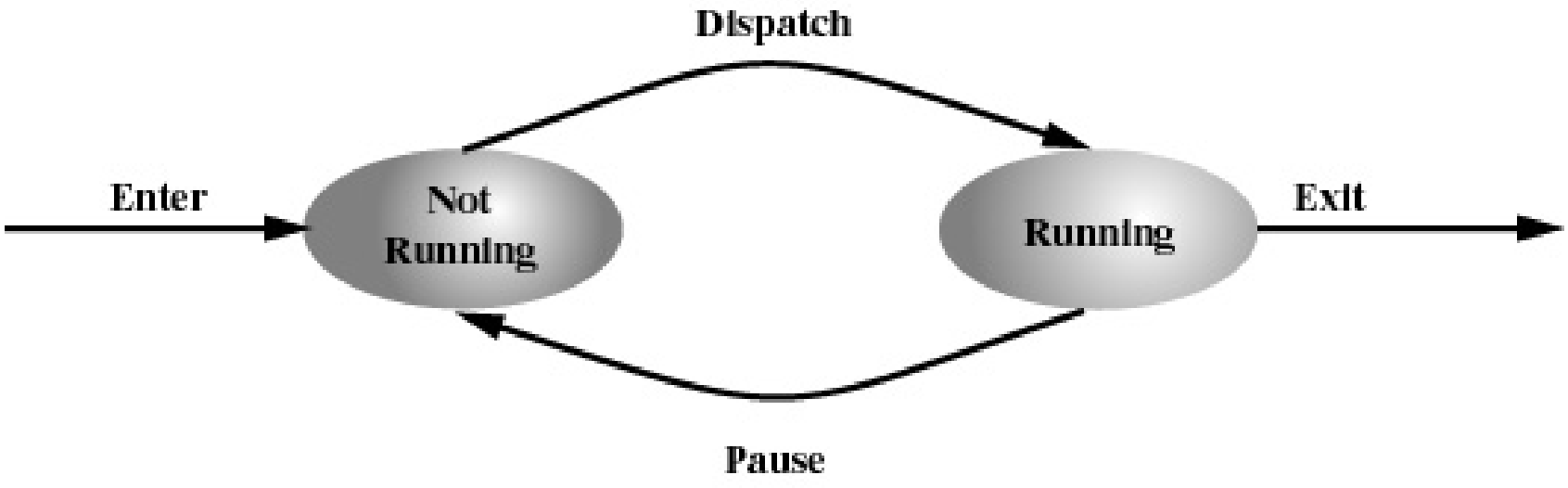
- el código y datos del programa.
- una pila de ejecución.
- el PC indicando la próxima instrucción.
- los valores actuales del conjunto de registros de la CPU.
- un conjunto de recursos del sistema (memoria, archivos abiertos, etc.).

2.2 Estados de un proceso

2.2 Estados de un proceso

modelo de procesos de dos estados

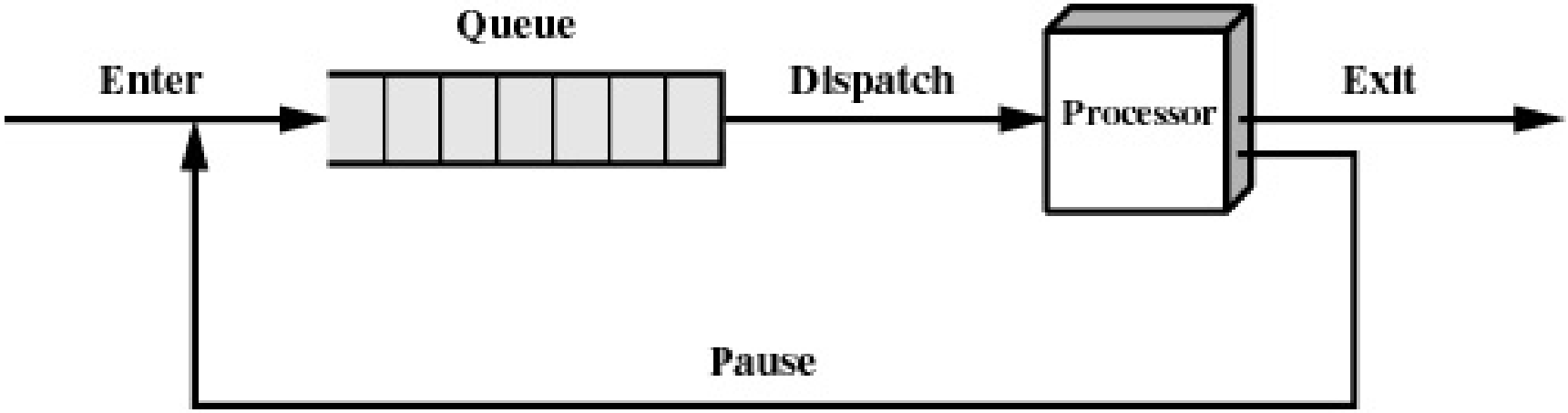
- El proceso puede estar en uno de dos estados
  - Running (ejecutándose)
  - Not-running



(a) State transition diagram

2.2 Estados de un proceso

estados de un proceso



(b) Queuing diagram

- La cola es una lista enlazada de bloques de datos, cada uno de los cuales representa un proceso.
- Un nuevo proceso entra en la CPU si el que se está ejecutando es interrumpido o termina. Si es interrumpido entra en una cola.

2.2 Estados de un proceso

creación de un proceso

- Un proceso se crea como consecuencia de:
  - Envío de un trabajo por lotes (batch)
  - Conexión de usuario (log-in)
  - Creados para proporcionar servicios como imprimir
  - Procesos que crean otros procesos (modularidad, paralelismo)
- El S.O. crea el proceso de forma transparente al usuario (programa):
  - construye las estructuras de datos necesarias para manejarlo
  - le asigna un espacio de direcciones en memoria

2.2 Estados de un proceso

terminación de un proceso

- Finalización normal (llamada al sistema, exit())
- Se excede el tiempo límite
- No hay memoria disponible
- Violación de límites de memoria
- Error de protección
  - ejemplo: escribir en un fichero de sólo lectura
- Error aritmético (i.e. división por cero)
- Tiempo de espera sobrepasado
  - un proceso espera un evento más tiempo del máximo especificado
- Fallo de entrada/salida (i.e. no existe el fichero, operación inválida)
- Instrucción inválida
  - Ocurre cuando se intenta ejecutar datos (salto incorrecto)
- Intento de ejecutar instrucciones privilegiadas en modo usuario
- Uso incorrecto de datos (tipo incorrecto o no inicializado)
- Intervención del Sistema Operativo
  - cuando se detecta un interbloqueo (deadlock)
- El padre termina, así que los hijos mueren (toda la descendencia)
- Por petición del proceso padre (tiene autoridad para terminar a su descendencia)

2.2 Estados de un proceso

modelo de 5 estados

- El modelo de 2 estados no puede funcionar puesto que habrá procesos en espera de E/S
- Los procesos not-running pueden estar:
  - Listo para ejecutar
  - Bloqueado: esperando una entrada/salida
- El planificador (dispatcher) no puede seleccionar forzosamente el proceso que más tiempo lleve en la cola porque puede estar bloqueado

2.2 Estados de un proceso

un modelo de 5 estados

- Running (en ejecución): tantos procesos como procesadores.
- Ready (listo): preparado para ejecutar.
- Blocked (bloqueado): en espera de E/S.
- New (nuevo): todavía no está cargado en memoria (i.e. demasiados procesos activos).
- Exit (salida): terminado por alguna razón (i.e. contabilidad, core)

2.2 Estados de un proceso

un modelo de 5 estados

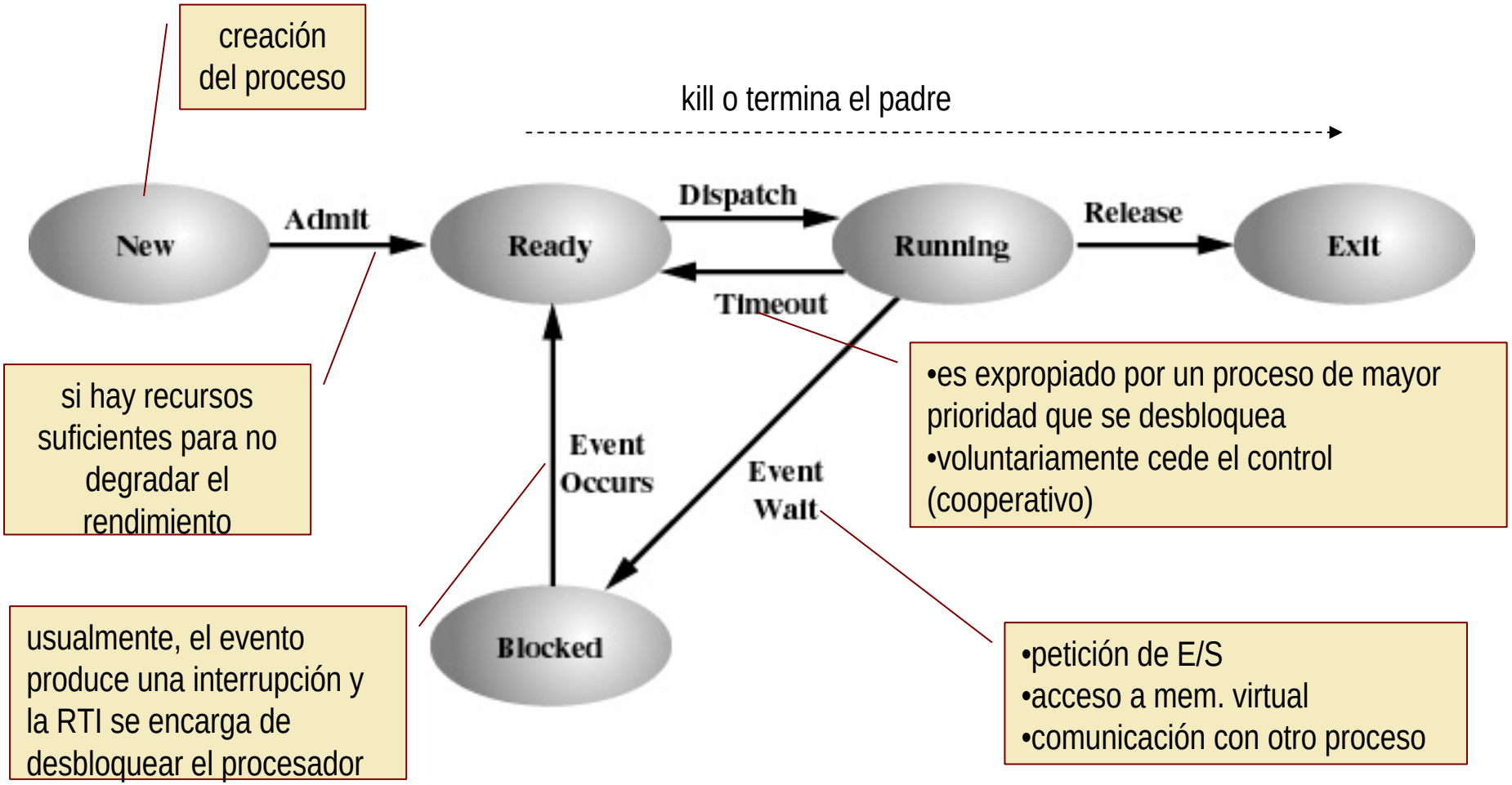


Figure 3.5 Five-State Process Model

2.2 Estados de un proceso

ejemplo 1: trazas de los procesos

(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

Figure 3.2 Traces of Processes of Figure 3.1

2.2 Estados de un proceso

ejemplo 1: trazas combinadas+snapshot

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

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

Figure 3.3 Combined Trace of Processes of Figure 3.1

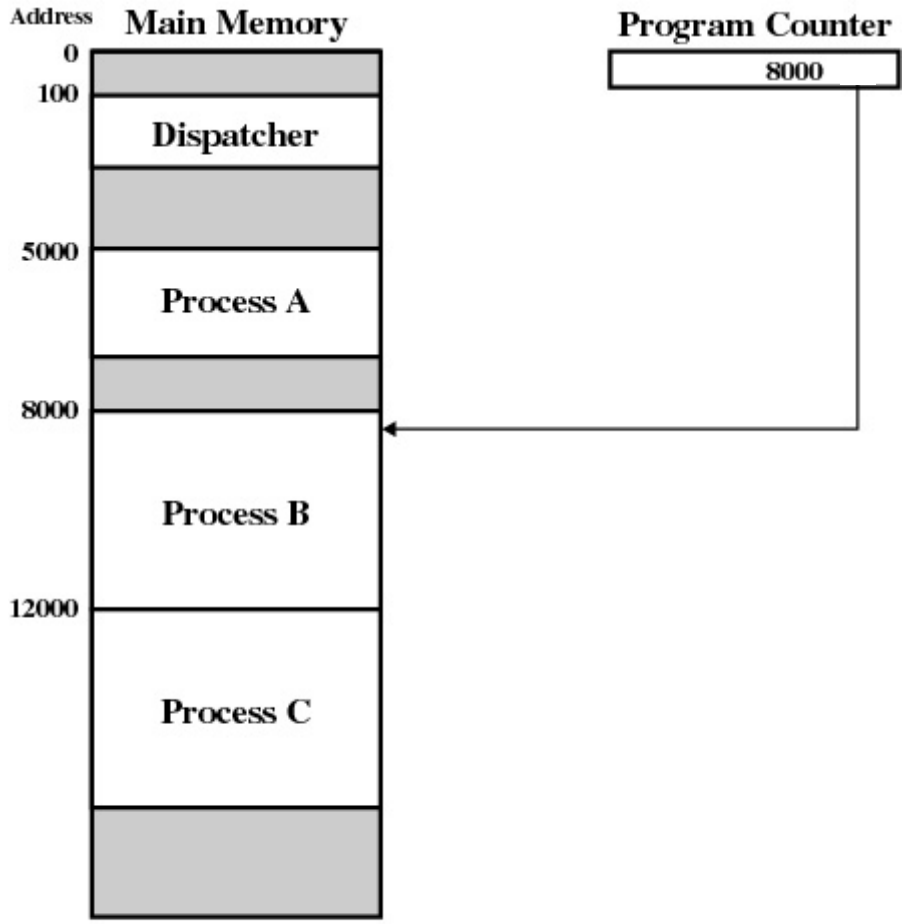


Figure 3.1 Snapshot of Example Execution (Figure 3.3) at Instruction Cycle 13

2.2 Estados de un proceso

ejemplo 1: estados de los procesos

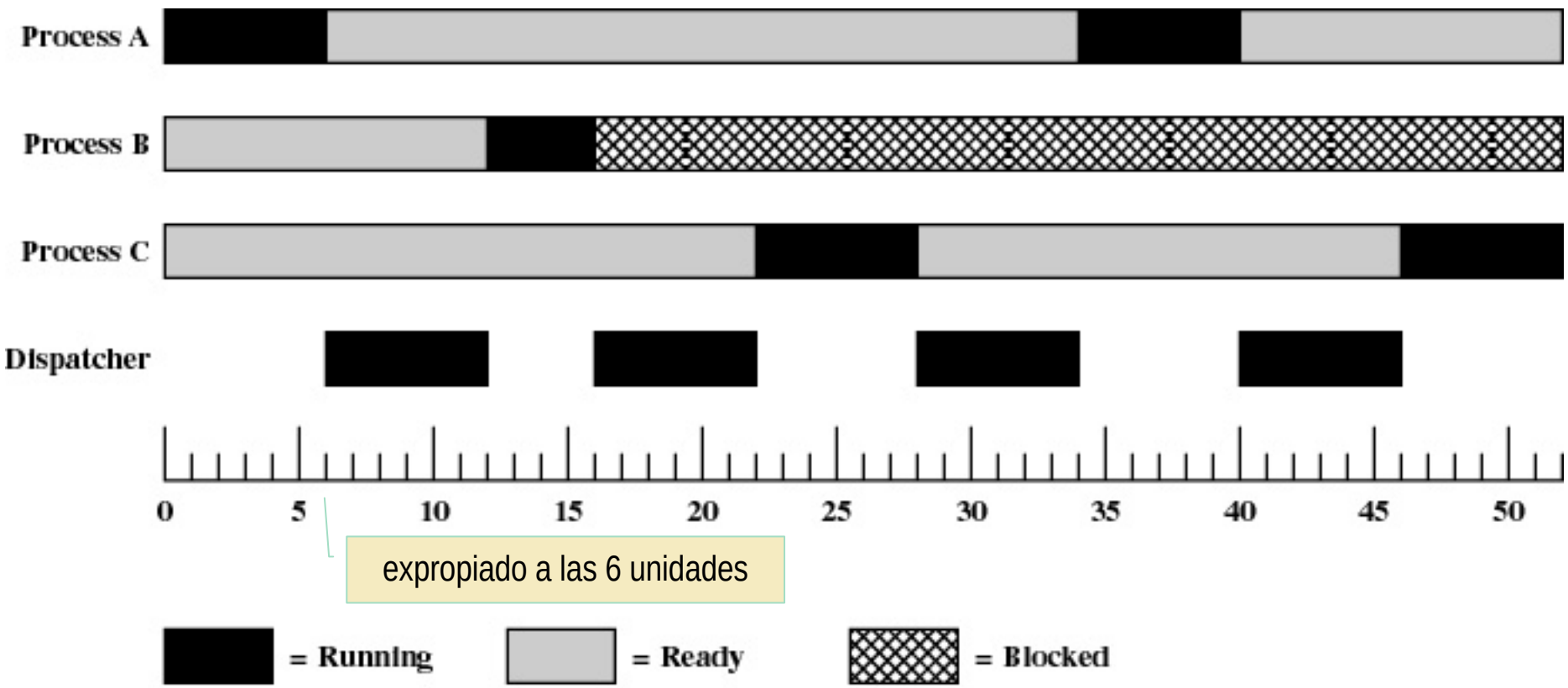
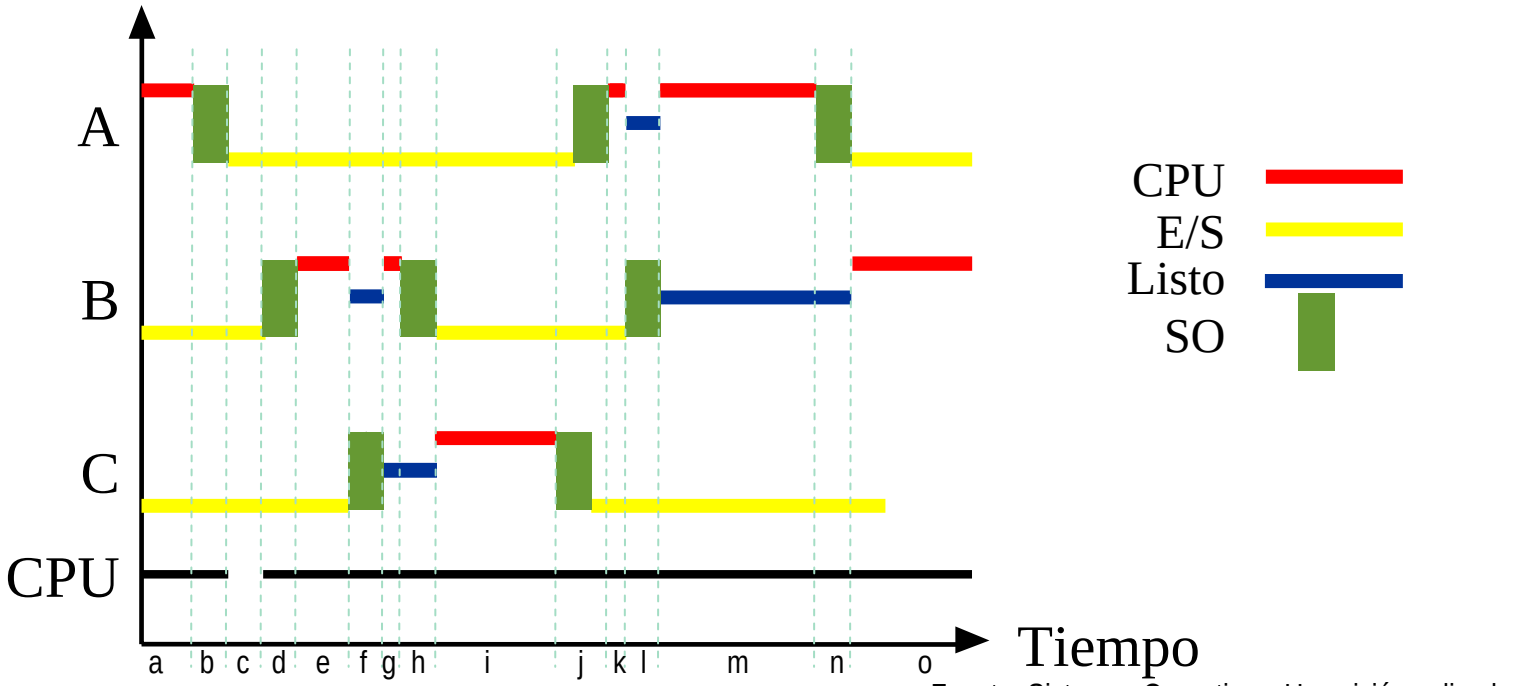


Figure 3.6 Process States for Trace of Figure 3.3

2.2 Estados de un proceso

ejemplo 2



Fuente: Sistemas Operativos. Una visión aplicada.

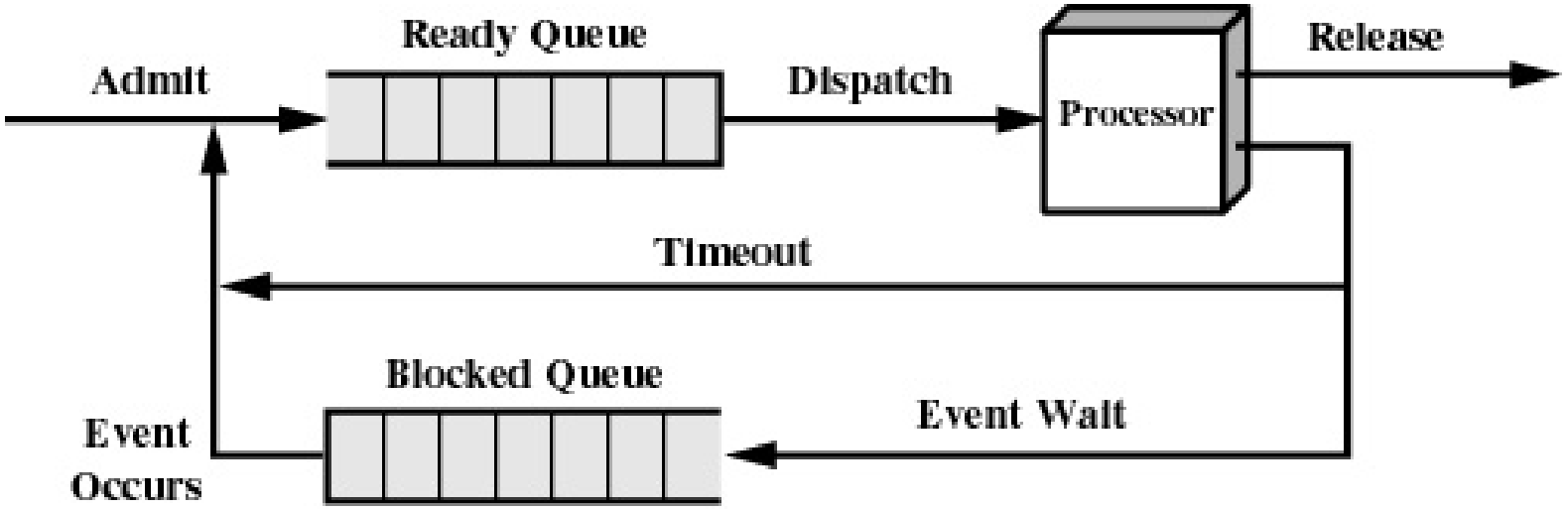
a: A en CPU, B y C bloqueados
b: A llama al SO para E/S
c: Todos bloqueados (CPU idle)
d: Acaba E/S de B (despierta y dispatch)
e: B en ejecución
f: Acaba E/S de C (despierta), B ready
g: Sigue B en CPU y C ready
h: B hace syscall. SO despacha C y bloquea B

i: C en CPU, A y B bloqueados
j: C llama al SO para E/S y A se despierta
k: A en ejecución
l: Una int E/S llama al SO para despertar B
m: A continua en Run y B espera ready
n: A se bloquea
o: B pasa a ejecución (run)

CPU ocupada siempre salvo en c

2.2 Estados de un proceso

usando dos colas

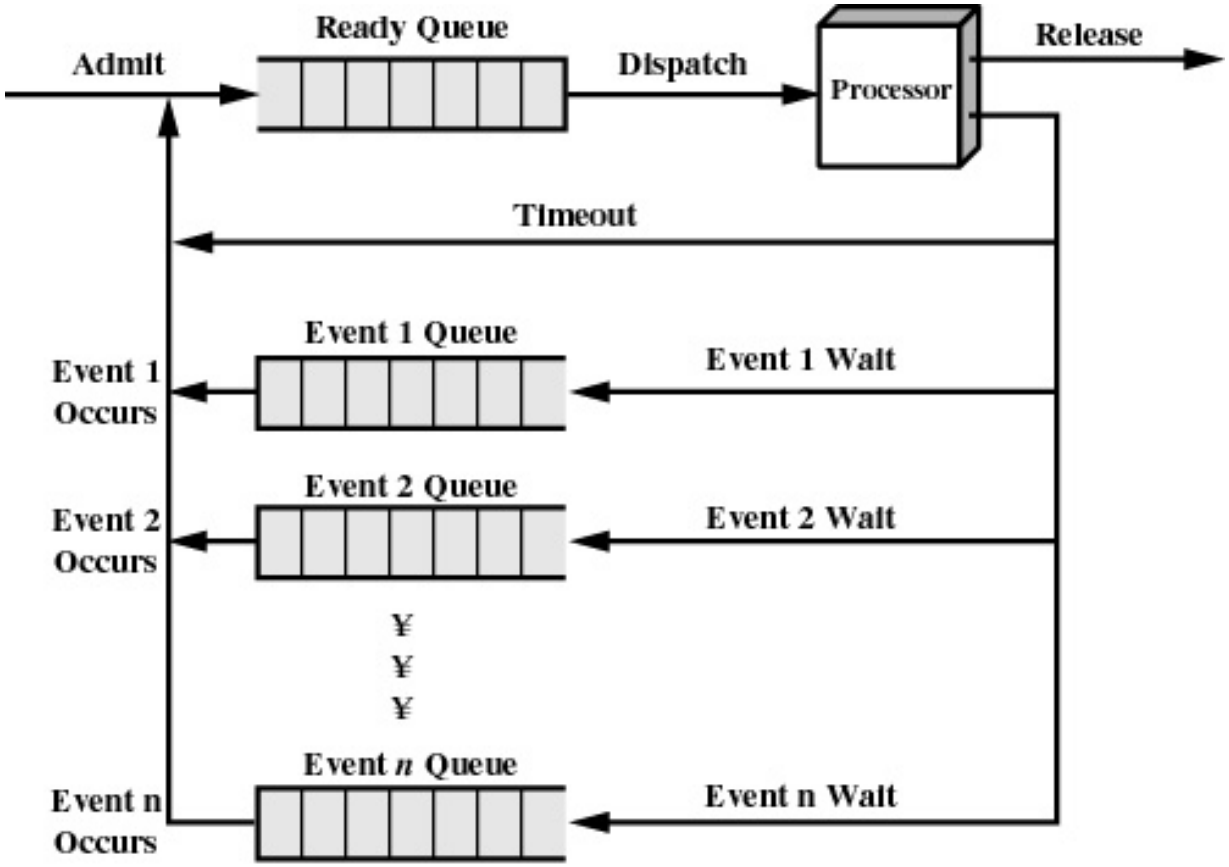


(a) Single blocked queue

- Cuando se produce un evento hay que buscar en la cola qué proceso bloqueado espera ese evento. Puede haber muchos (100~1000) y que sea una operación lenta

2.2 Estados de un proceso

más de dos colas



(b) Multiple blocked queues

- También puede haber varias colas de listos con diferente prioridad

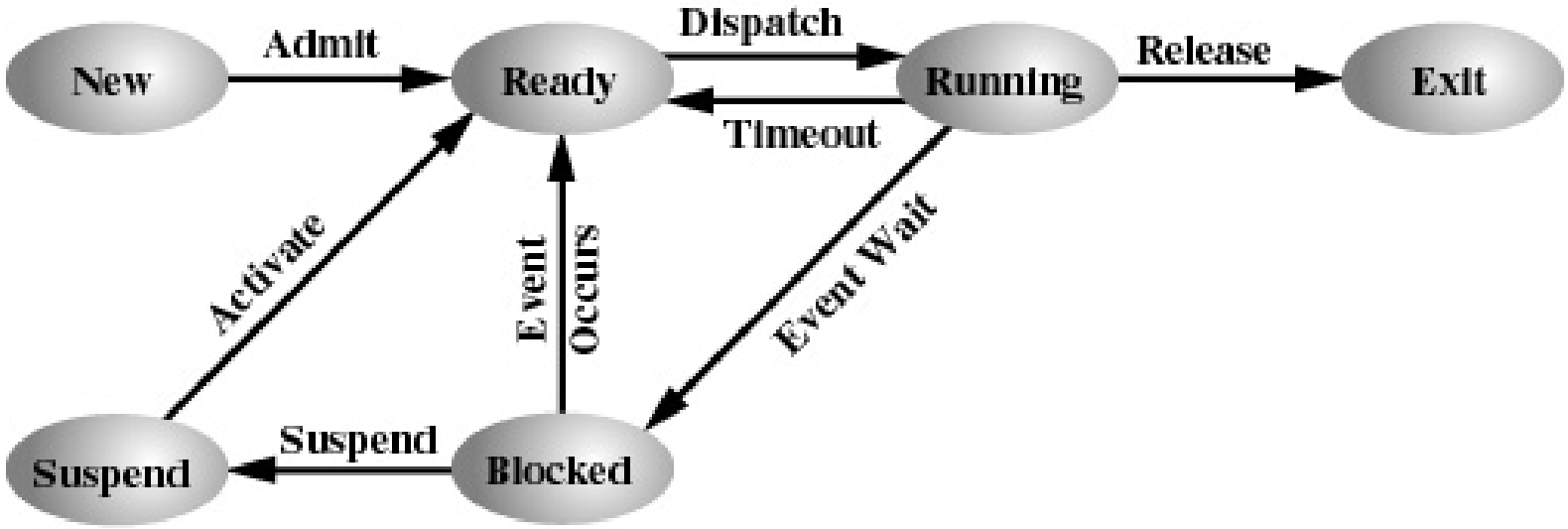
2.2 Estados de un proceso

procesos suspendidos

- El procesador es más rápido que la E/S, así que todos los procesos pueden estar esperando E/S
- ¿Hace falta más memoria para tener más procesos listos?: NO -> Sacar estos procesos al disco para liberar más memoria (swap)
- De estado bloqueado pasa a suspendido cuando el proceso es 'movido' al disco (swap)
- Dos nuevos estados
  - Bloqueado, suspendido
  - Listo (ready), suspendido

2.2 Estados de un proceso

diagrama de estados con suspendido



(a) With One Suspend State

Cuando todos los procesos están bloqueados, se **suspende** uno, liberando memoria para: (a) admitir un proceso new (b) recuperar un proceso suspendido que ya esté libre (opción preferida)

2.2 Estados de un proceso

diagrama con dos estados de suspendido

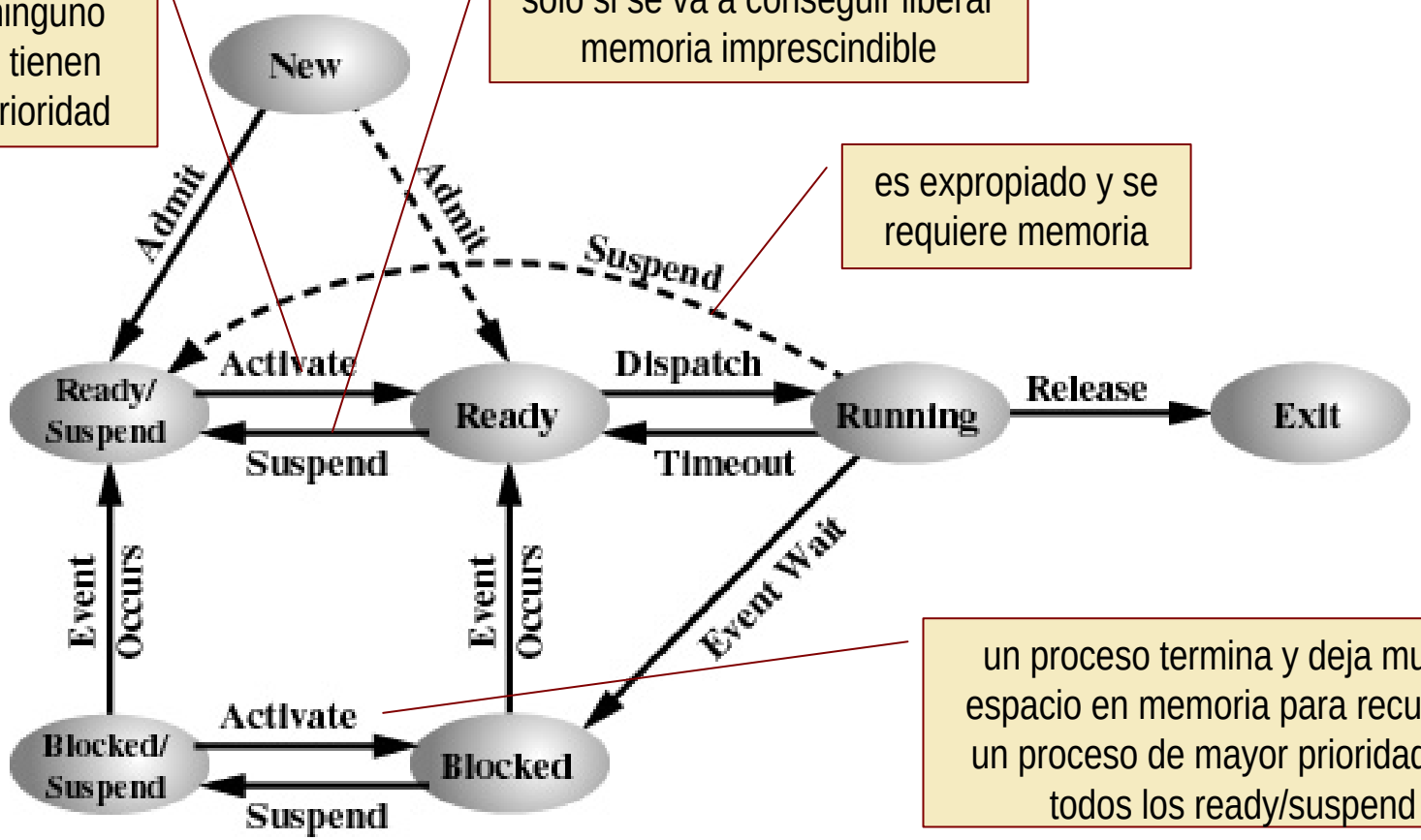
Para evitar recuperar un suspendido que todavía esté bloqueado:

no hay ninguno ready, o tienen menos prioridad

sólo si se va a conseguir liberar memoria imprescindible

es expropiado y se requiere memoria

un proceso termina y deja mucho espacio en memoria para recuperar un proceso de mayor prioridad que todos los ready/suspend



(b) With Two Suspend States

2.2 Estados de un proceso

estados de un proceso

- Ventajas de la multitarea
  - Aprovecha el tiempo de espera de un proceso mientras termina una operación de E/S
  - Aumenta el uso de CPU
  - Permite una implementación eficiente de sistemas multiusuario
  - Facilita la programación
    - Divides un programa en procesos que colaboran entre sí
    - Modularidad en la programación
- Limitación del grado de multiprogr.
  - Si hay muchos procesos ready aumenta el tiempo de respuesta
  - El tamaño limitado de la memoria también restringe el número de procesos cargados

2.3 Descripción de procesos

2.3 Descripción de procesos

- El SO es la entidad que gestiona el uso de los recursos del sistema por parte de los procesos.
- ¿qué información necesita el S.O. para controlar los procesos y gestionar los recursos?.

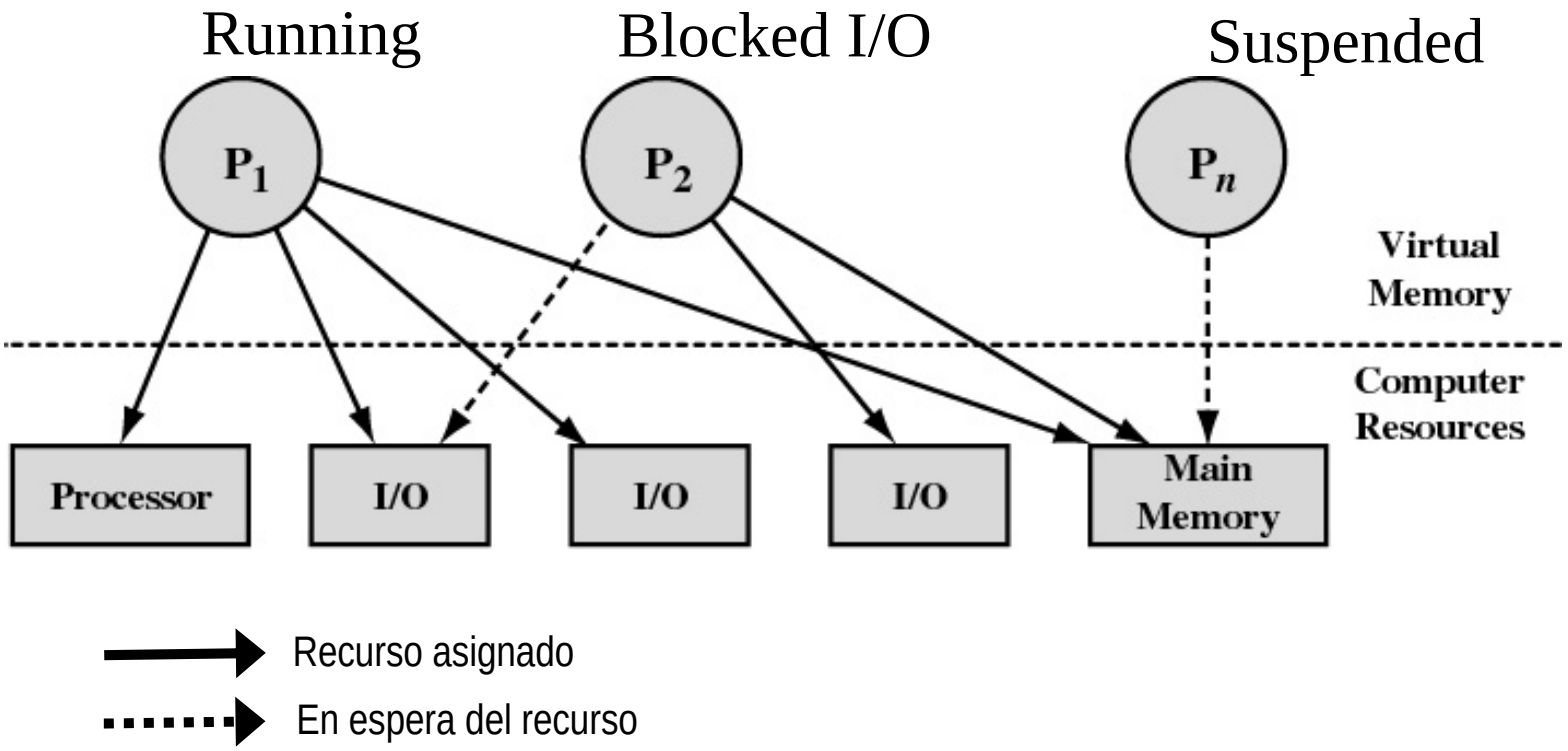


Figure 3.9 Processes and Resources (resource allocation at one snapshot in time)

2.3 Descripción de procesos

estructuras de control del SO

- Información sobre el estado actual de cada proceso y recurso
- Las tablas se construyen para cada entidad que gestiona el sistema operativo:
  - Memoria: (memoria principal (real) y virtual)
  - E/S
  - Ficheros
  - Procesos
- Las tablas tienen referencias cruzadas (i.e. memoria, E/S y ficheros se gestionan por parte de los ficheros: las tablas de procesos los referenciarán)
- Inicialización de las tablas: el SO conoce el entorno (mem. disponible, dispositivos E/S...y demás configuración). Introducido por: instalador del SO o SW autoconfiguración)

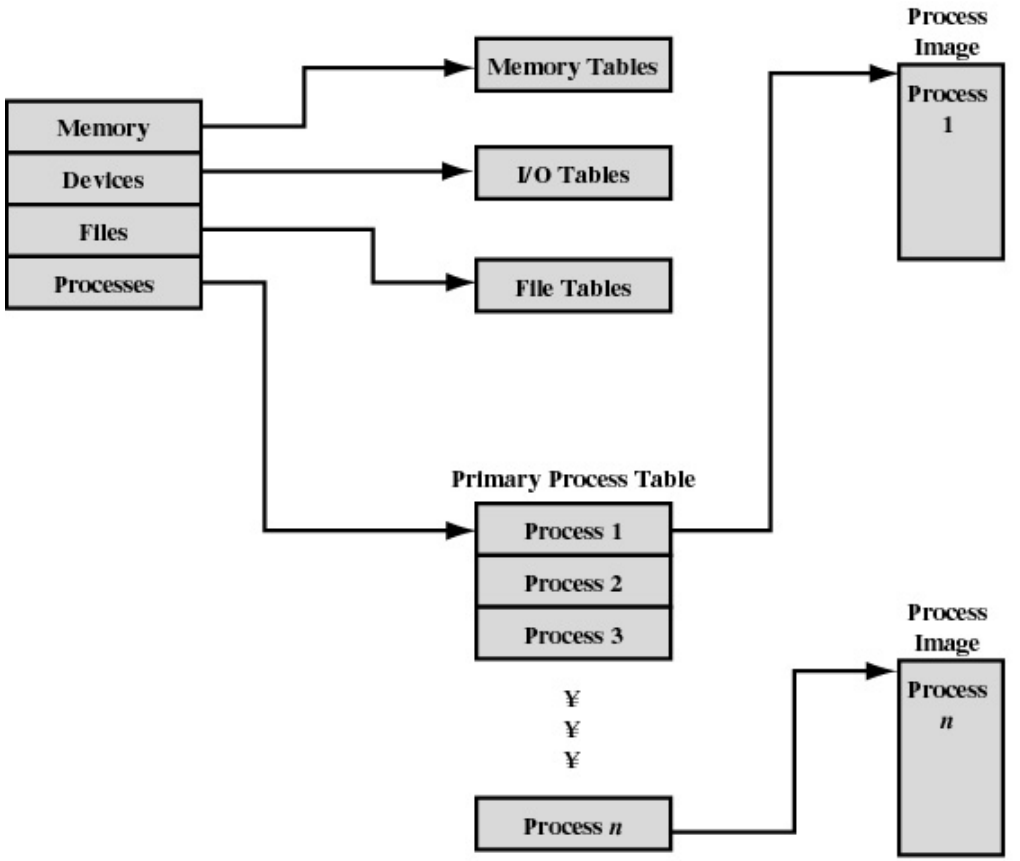


Figure 3.10 General Structure of Operating System Control Tables

2.3 Descripción de procesos

estructuras de control: tablas de memoria

- Asignación de memoria principal a procesos
- Asignación de memoria secundaria a procesos
- Atributos de protección del acceso a regiones de memoria compartida
- Información necesaria para gestionar la memoria virtual

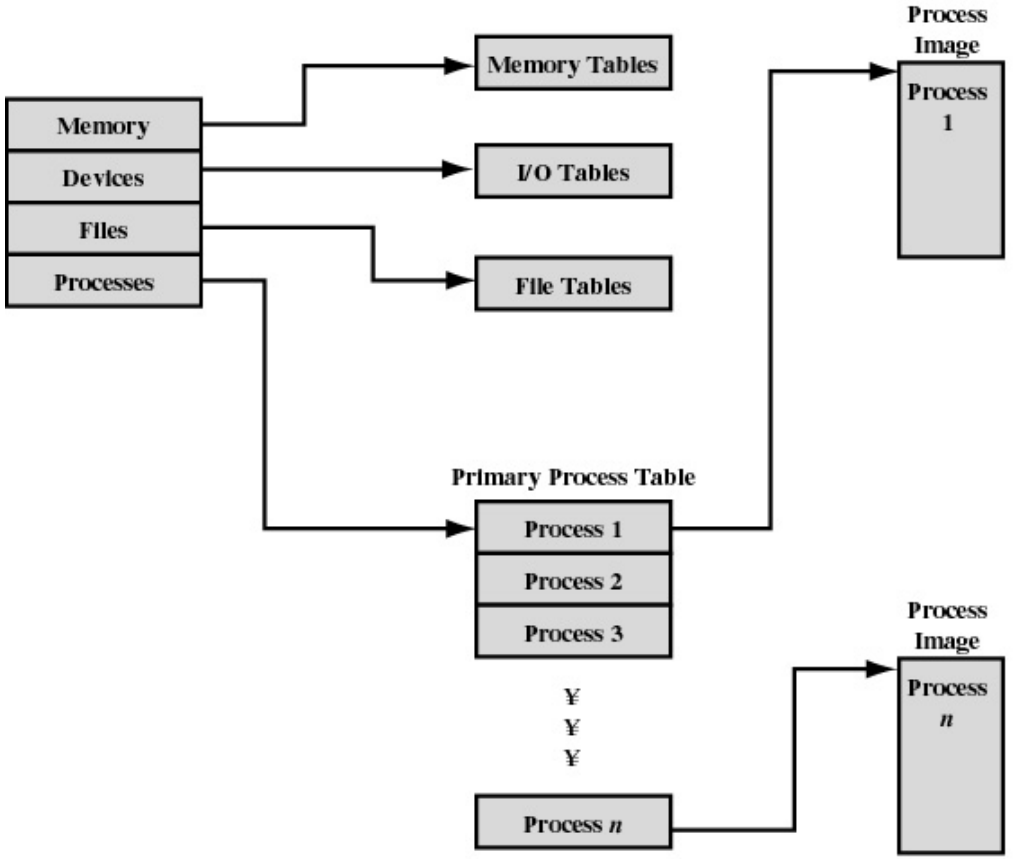


Figure 3.10 General Structure of Operating System Control Tables

2.3 Descripción de procesos

estructuras de control: tablas de E/S

- Dispositivos de E/S que están disponibles o asignados
- Estado de las operaciones de E/S
- Ubicación en memoria principal del espacio usado como fuente o destino en una transferencia de E/S

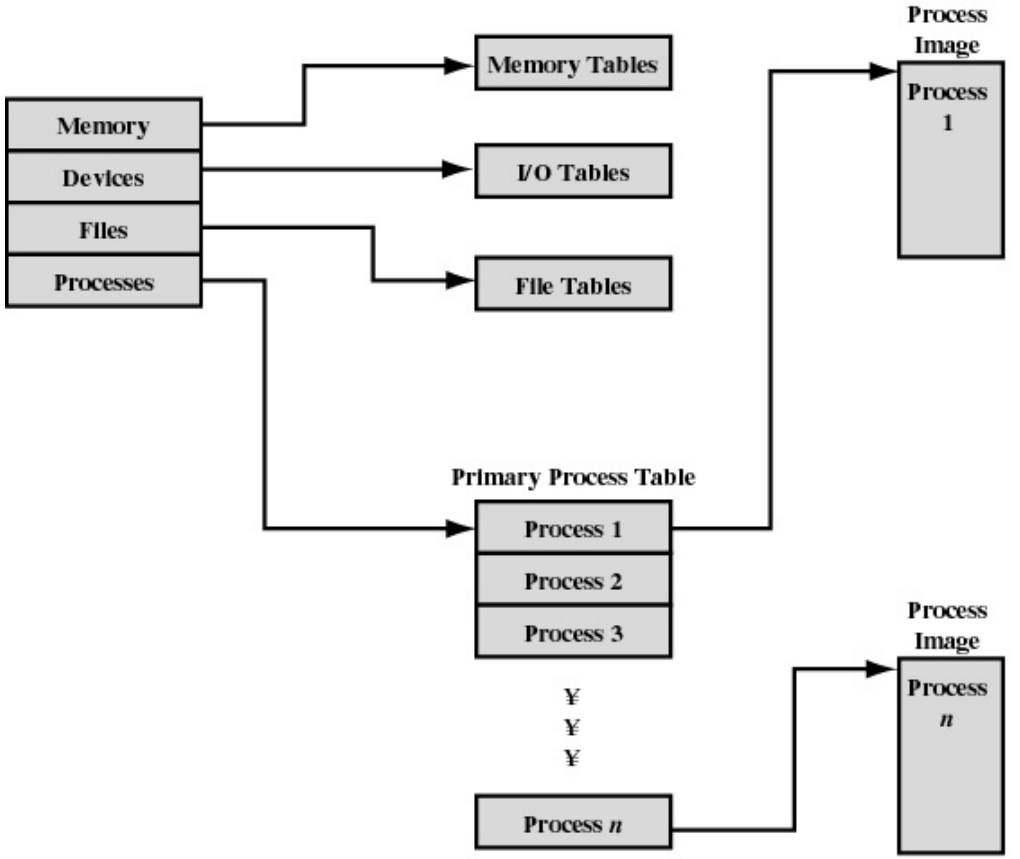


Figure 3.10 General Structure of Operating System Control Tables

2.3 Descripción de procesos

estructuras de control: tablas de ficheros

- Existencia de Ficheros
- Ubicación en memoria secundaria
- Estado actual y otros atributos
- A veces la mayoría de esta información es mantenida por el sistema de gestión de ficheros y el SO tiene poco conocimiento sobre los ficheros.

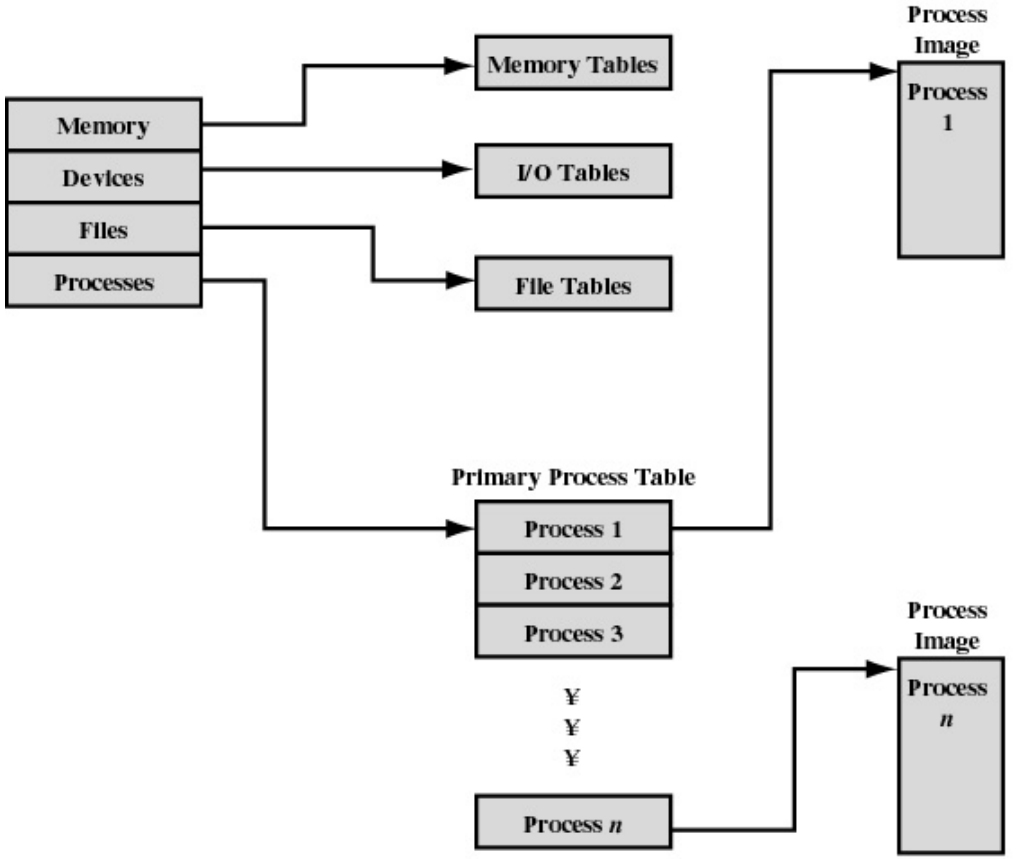
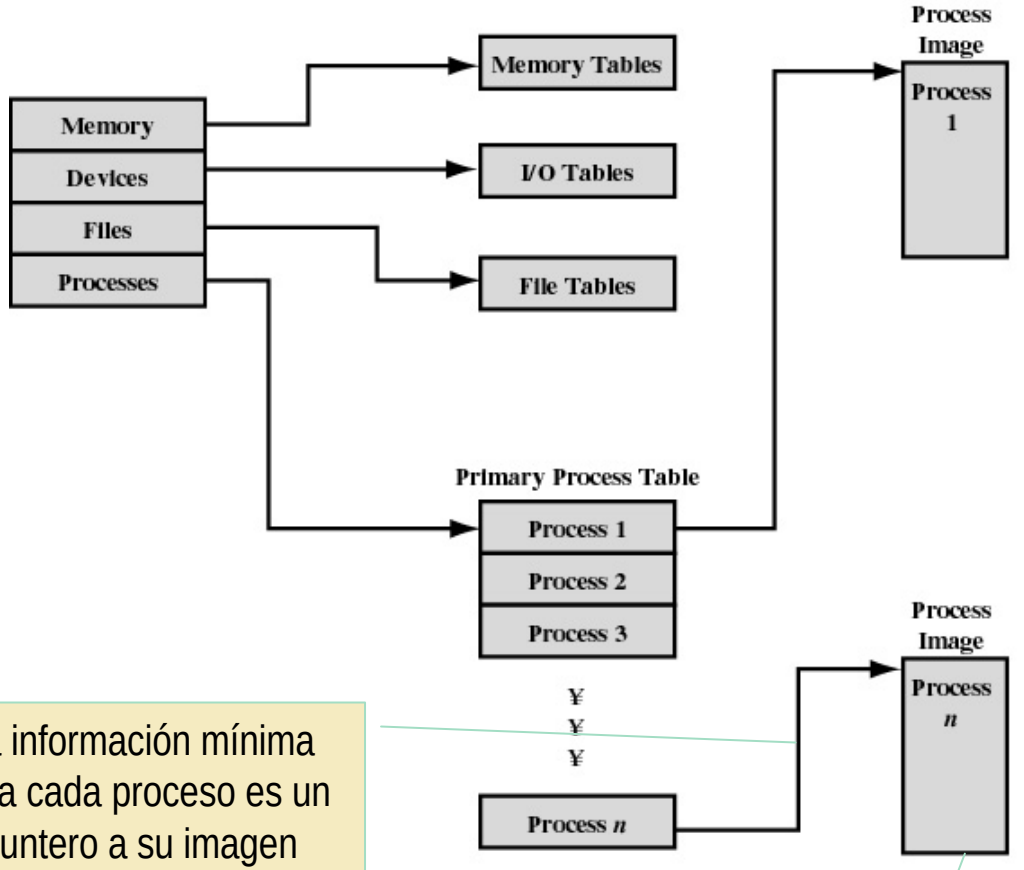


Figure 3.10 General Structure of Operating System Control Tables

2.3 Descripción de procesos

estructuras de control: tablas de procesos

- Dónde está ubicado un proceso (imagen del proceso)
- Atributos necesarios para su gestión (PCB)
  - Identificador del proceso (ID)
  - Estado del proceso
  - ...



La información mínima para cada proceso es un puntero a su imagen

Figure 3.10 General Structure of Operating System Control Tables

La imagen no tiene por qué estar en bloques contiguos de memoria virtual.
Si ocupa varios bloques, están registrados en la tabla de memoria o en la de procesos

2.3 Descripción de procesos

imagen del proceso

- Imagen del proceso: colección del programa, datos, pila, y atributos
- El proceso necesita almacenar en memoria:
  - programa a ejecutar
  - datos para variables locales y globales
  - constantes definidas
  - pila

  y además, una colección de atributos utilizados por el SO para controlar el proceso: bloque de control del proceso (BCP)
- Ubicación de la imagen (depende del esquema de gestión de memoria utilizado):
  - bloque de memoria contiguo en memoria secundaria:
    - para gestionar el proceso una pequeña porción de su imagen debe estar en mem. ppal.
    - para ejecutar el proceso la imagen completa debe cargarse en mem. ppal. o al menos virtual
  - bloque de memoria física no contiguo: procesos parcialmente residentes en la mem. ppal.

2.3 Descripción de procesos

bloque de control de procesos

- También llamado Bloque de Control de la Tarea, Descriptor del Proceso o Descriptor de la Tarea
- Es la estructura más importante del SO:
  - un error en su manipulación (PCB dañado) puede destruir el proceso
  - su rediseño obliga a rescribir mucho código del SO
- Un SO multiprogramado actual requiere una gran cantidad de información para manejar cada proceso:
  - Identificación del proceso
    - Identificadores

      Los identificadores numéricos que deben ser almacenados en el PCB incluyen:
      - El identificador de ese proceso
      - El identificador del proceso que creó este proceso (proceso padre)
      - Identificador del usuario

2.3 Descripción de procesos

bloque de control de procesos

- Información del Estado del Procesador
  - Registros visibles por el usuario
    - Un registro visible por el usuario puede ser referenciado a través de instrucciones del lenguaje máquina que el procesador ejecuta. Normalmente hay de 8 a 32 de esos registros, aunque algunas implementaciones RISC tiene más de 100.
  - Registros de Control y Estado

    Hay una variedad de registros del procesador que se emplean para controlar el funcionamiento del procesador. Estos incluyen:
    - *Contador del Programa:* Contiene la dirección de la siguiente instrucción a buscar
    - *Códigos de Condición:* El resultado de la operación aritmética o lógica más reciente (p.ej., signo, cero, acarreo, desbordamiento)
    - *Información de Estado:* Incluye el modo de ejecución, 'flag' de habilitación/inhibición de las interrupciones

2.3 Descripción de procesos

información del estado del procesador

ejemplo: el registro EFLAGS de las máquinas Pentium II

ID = Identification flag
VIP = Virtual interrupt pending
VIF = Virtual interrupt flag
AC = Alignment check
VM = Virtual 8086 mode
RF = Resume flag
NT = Nested task flag
IOPL = I/O privilege level
OF = Overflow flag

DF = Direction flag
IF = Interrupt enable flag
TF = Trap flag
SF = Sign flag
ZF = Zero flag
AF = Auxiliary carry flag
PF = Parity flag
CF = Carry flag

Figure 3.11 Pentium II EFLAGS Register

- Información del Estado del Procesador
  - Punteros a la Pila
    - Cada proceso tiene una o más pilas LIFO asociadas. La pila se usa para almacenar los parámetros y las direcciones de retorno de las llamada a procedimientos y a las funciones del sistema. El puntero de pila apunta a la cima de ésta.

2.3 Descripción de procesos

bloque de control de procesos

- Información de Control del Proceso
  - Información de Planificación y Estado

    Es la información que necesita el sistema operativo para realizar su función de planificación. Información típica:

    – *Estado del Proceso:* define la disponibilidad del proceso para que se planifique su ejecución (p.e., running, ready, waiting, halted).

    – *Prioridad:* Se pueden usar uno o más campos para describir la prioridad de planificación del proceso. En algunos sistemas se requieren varios valores (p.e., por defecto, actual, mayor permitido)

    – *Información relacionada con la planificación:* Dependerá del algoritmo de planificación. Por ejemplo, el tiempo que el proceso ha estado esperando y el tiempo que estuvo ejecutandose por última vez.

    – *Evento:* Identificador del evento que el proceso está esperando para que se pueda reanudar

2.3 Descripción de procesos

bloque de control de procesos

- Información de Control del Proceso
  - Estructuración de los datos
    - Un proceso puede estar enlazado a otro proceso en una cola, anillo, u otra estructura. Por ejemplo, todos los procesos en espera de un nivel de prioridad particular estarán *linkados* a una cola. Un proceso puede exhibir una relación padre-hijo (creador-creado) con otro proceso. El PCB contiene punteros a otros procesos para dar soporte a estas estructuras.
  - Comunicación Inter-procesos
    - Algunos flags, señales y mensajes pueden estar asociados con la comunicación entre procesos independientes. Alguna o toda esta información puede estar mantenida en el PCB.
  - Privilegios del Proceso
    - Los procesos tienen unos privilegios concedidos en términos de memoria de la que pueden disponer y tipos de instrucciones que puede ejecutar. Además, los privilegios se pueden aplicar al uso de ciertas utilizades y servicios del sistema.

2.3 Descripción de procesos

bloque de control de procesos

- Información de Control del Proceso
  - Gestión de Memoria
    - Esta sección puede incluir punteros a segmento y/o tablas de página que describen la memoria virtual asignada a este proceso.
  - Utilización y Propiedad de Recursos
    - Los recursos controlados por el proceso deben estar indicados, tales como ficheros abiertos. Un historial de utilización del procesador u otros recursos también se pueden incluir; esta información la puede necesitar el planificador.

2.3 Descripción de procesos

imágenes de procesos

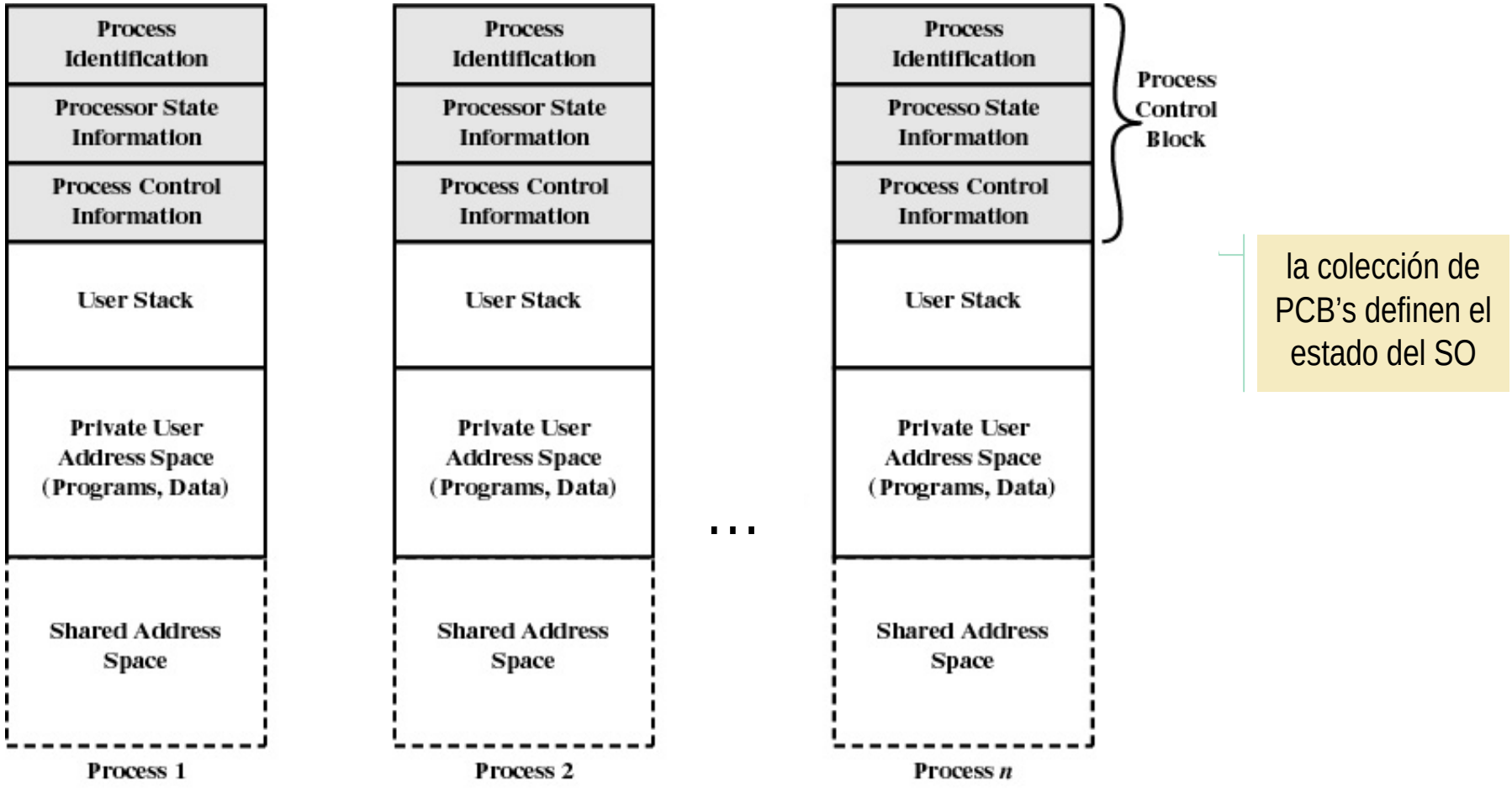
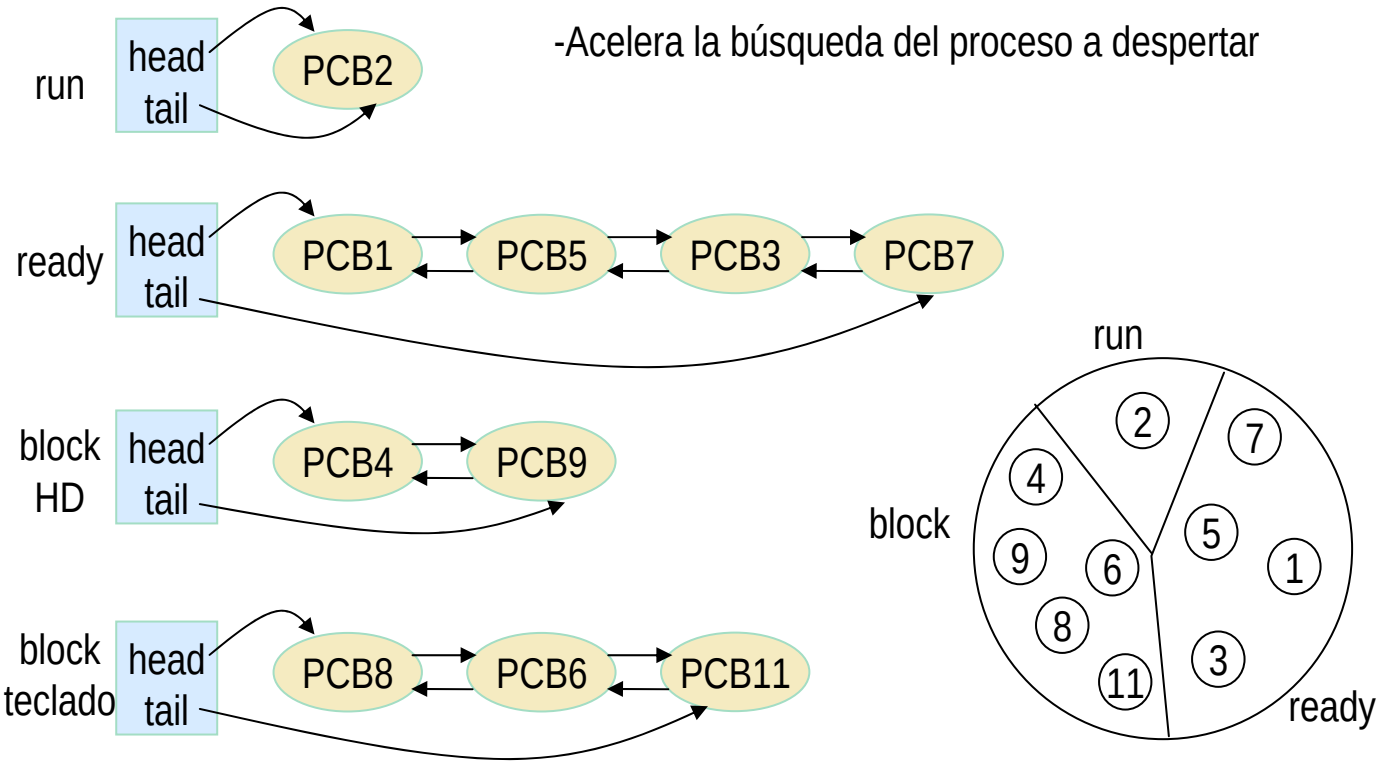


Figure 3.12 User Processes in Virtual Memory

2.3 PCB

estructuración de los datos: listas de PCBs

- Organización de los procesos
  - Listas de procesos de un tipo determinado
    - » Run: tantos procesos como procesadores
    - » Ready: procesos listos. Lista ordenada por el planificador
    - » Block: varias listas desordenadas
      -Acelera la búsqueda del proceso a despertar



2.4 Control de procesos

2.4 Control de procesos

modos de ejecución

- Modo usuario
  - modo con menos privilegios
  - los programas de usuario normalmente se ejecutan en este modo
- Modo del sistema, modo de control o modo kernel
  - modo con más privilegios (acceder a ciertas zonas de memoria, modificar la palabra de estado, instrucciones de E/S...)
  - kernel del sistema operativo: es necesario proteger al SO y sus tablas principales, como los PCB asociados a los procesos de usuario.

2.4 Control de procesos

creación del proceso

- Asignar un identificador único (se crea una nueva entrada en la tabla de procesos primaria)
- Destinar espacio para el proceso (imagen)
- Inicializar el PCB (ID, PC, SP...)
- Actualizar los *links* adecuados
  - Ej.: añadir el nuevo proceso a la lista usada para la planificación
- Crear o expandir otras estructuras de datos
  - Ej: mantener un fichero de contabilidad

2.4 Control de procesos

cuándo conmutar (switch) de proceso

1. Interrupción del reloj (timer)
   - el proceso se ha ejecutado durante el máximo periodo de tiempo permitido (time slice)
2. Interrupción E/S
3. Fallo de memoria
   - la dirección de memoria está en la memoria virtual y hay que traerla a la memoria principal
4. Excepción
   - ocurre un error
   - puede causar que un proceso se mueva al estado de Exit
5. Llamada al Supervisor (al sistema)
   - tal como abrir un fichero

2.4 Control de procesos

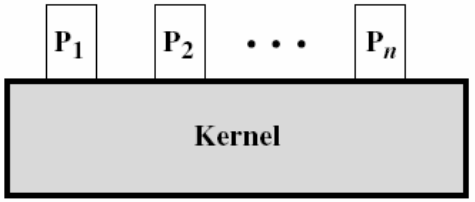
cambio de estado del proceso

- Puede haber un cambio de modo sin que se produzca cambio del estado del proceso en ejecución => sobrecarga menor (sólo salvaguardar estado y recuperarlo después)
- El cambio de contexto supone la realización de cambios sustanciales en el entorno del SO:
  1. Guardar el contexto del procesador incluyendo el contador de programa y otros registros
  2. Actualizar el PCB del proceso que actualmente se está ejecutando
  3. Mover el PCB a la cola apropiada: listo, bloqueado,...
  4. Seleccionar otro proceso para ejecución
  5. Actualizar el PCB del proceso seleccionado
  6. Actualizar la estructura de datos del gestor del memoria
  7. Restaurar el contexto del proceso seleccionado

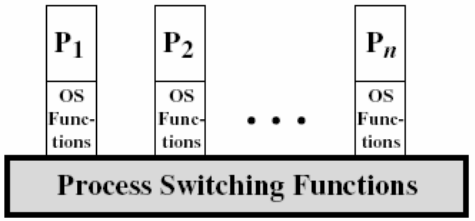
2.4 Control de procesos

ejecución del sistema operativo

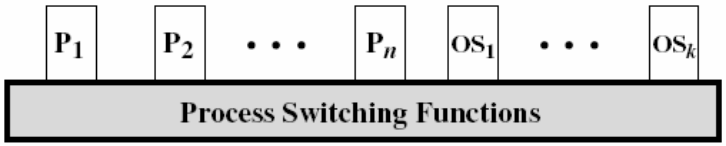
- Non-process Kernel
  - ejecuta el kernel fuera de cualquier proceso
  - el código del sistema operativo se ejecuta como una entidad separada que opera en modo privilegiado
- Ejecución en procesos de usuario
  - software del sistema operativo en el contexto de un proceso de usuario
  - el proceso se ejecuta en modo privilegiado cuando ejecuta código del sistema operativo
- Sistema Operativo basado en procesos
  - las funciones principales del kernel son procesos individuales
  - es útil en entornos multiprocesador o multicomputador



(a) Separate kernel



(b) OS functions execute within user processes



(c) OS functions execute as separate processes

Figure 3.14 Relationship Between Operating

2.4 Control de procesos

ejecución del sistema operativo

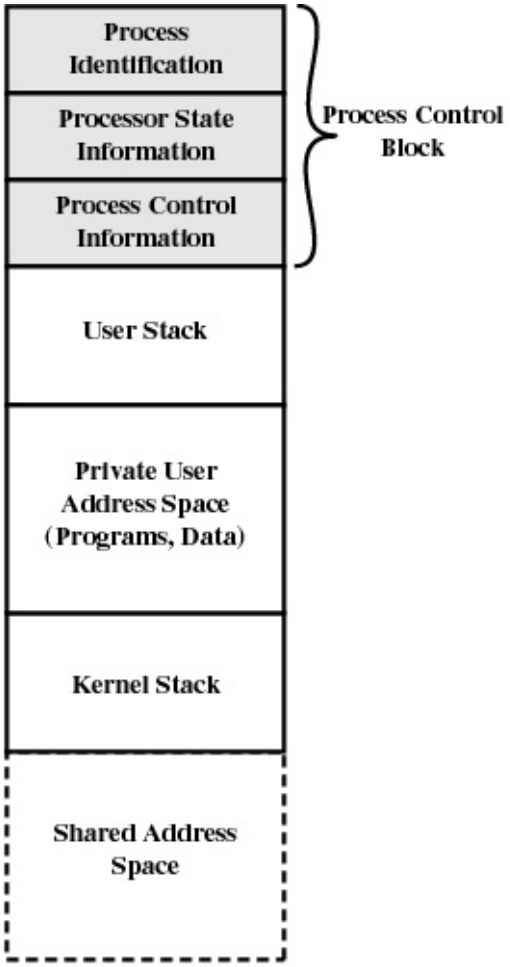


Figure 3.15 Process Image: Operating System Executes Within User Space

2.5 Operaciones con proc.

2.5 Operaciones con proc.

creación de procesos

- Un proceso puede crear otros procesos, y
  - Formar un árbol de procesos (UNIX): relación de parentesco entre procesos.
  - No mantener una jerarquía (Win 2000).
- ¿Compartir recursos entre el creador y el creado?
  - Comparten todos los recursos, o un subconjunto.
  - Creador y creado no comparten recursos.

2.5 Operaciones con proc.

creación de procesos

- Respecto a la ejecución:
  - Creador/creado se ejecutan concurrentemente.
  - Creador espera a que el creado termine.
- Sus espacios de direcciones son:
  - Clonados: se copia el espacio del creador para el creado. Comparten el mismo código, datos, ... ej. Unix.
  - Nuevos: el proceso creado inicia un programa diferente al del creador. ej. Windows 2000.

2.5 Operaciones con proc.

terminación de procesos

- Un proceso finaliza cuando ejecuta la declaración de terminación (explícita o implícitamente).
- Los pasos a seguir:
  1. Envío de datos del proceso creado a creador. P.ej. Código de finalización.
  2. El SO desasigna los recursos que tiene.
- El proceso puede finalizar la ejecución de otro si:
  - Ha sobrepasado los recursos asignados.
  - La tarea asignada al proceso ya no es necesaria.
  - Va a finalizar y el SO no permite que los procesos creados por él puedan continuar: terminación en cascada.

2.5 Operaciones con proc.

APIs de UNIX y Win32

Operación	Unix	Win32
Crear	`fork()` `exec()`	`CreateProcess()`
Terminar	`_exit()`	`ExitProcess()`
Obtener código finalización	`wait` `waitpid`	`GetExitCodeProcess`

Obtener tiempos	`times`
	`wait3`
	`wait4`
		`GetProcessTimes`

Identificador	`getpid`	`GetCurrentProcessId`
Terminar otro proceso	`kill`	`TerminateProcess`

2.5 Operaciones con proc.

operaciones con procesos

- Operaciones con procesos
  - Identificación de procesos
    - getpid, GetCurrentProcessId
  - Entorno de un proceso
    - getenv, GetEnvironmentStrings
    - Path, directorio de trabajo, tipo de terminal, dir. personal
  - Creación de procesos
    - fork, CreateProcess
  - Cambio de programa de un proceso: exec
  - Esperar la terminación de un proceso
    - wait, WaitForSingleObject, WaitForMultipleObjetcs
  - Finalizar la ejecución de un proceso
    - exit, TerminateProcess
  - Enviar señales a un proceso: kill
  - Comunicar procesos

2.5 Operaciones con proc.

servicios POSIX

- Servicios POSIX

pid_t fork(void)

- Crea un proceso hijo. Devuelve 0 al proceso hijo y el pid del hijo al proceso padre.

int execlp(const char *file, const char *arg, ...)

- Permite a un proceso ejecutar un programa (código) distinto. Cambia la imagen de memoria del proceso. El pid no cambia.

pid_t wait(int *status)

- Permite a un proceso padre esperar hasta que termine un proceso hijo. Devuelve el identificador del proceso hijo y el estado de terminación del mismo.

void exit(int status)

- Finaliza la ejecución de un proceso indicando el estado de terminación del mismo.

pid_t getpid(void)

- Devuelve el identificador del proceso.

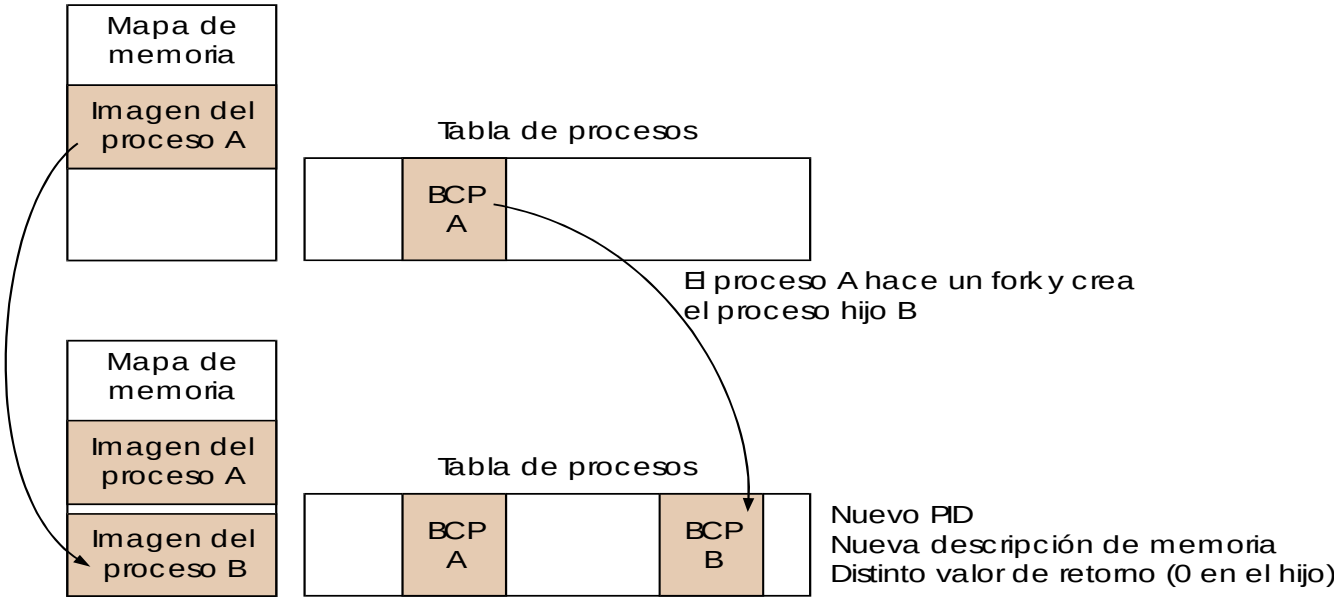
pid_t getppid(void)

- Devuelve el identificador del proceso padre.

2.5 Operaciones con proc.

fork

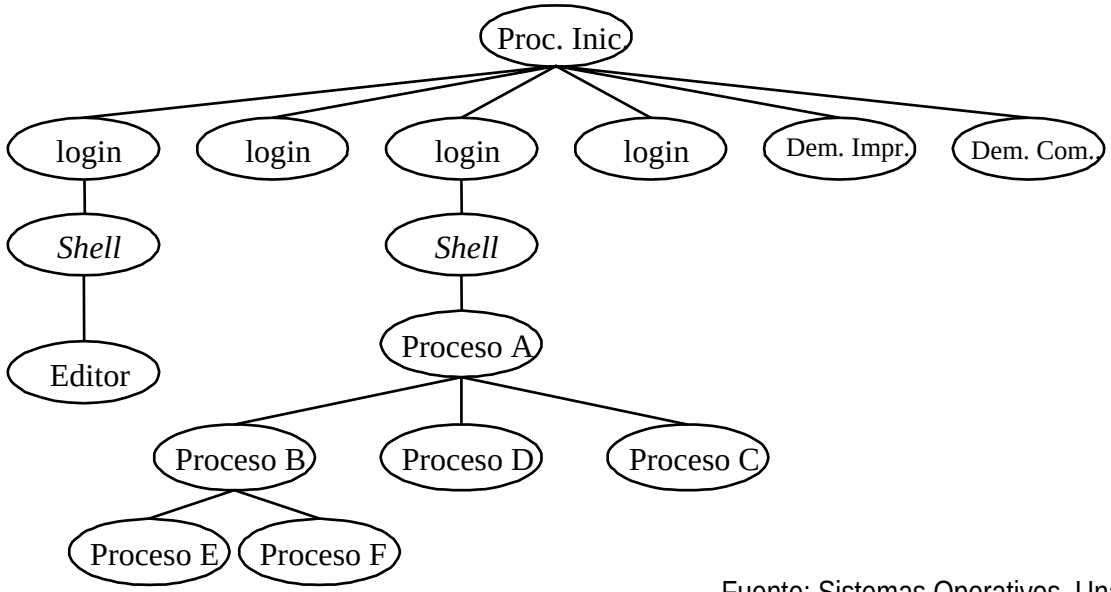
- `pid_t fork(void);`
  - Devuelve:
    - El identificador de proceso hijo al proceso padre y 0 al hijo
    - -1 el caso de error
  - Descripción:
    - Crea un proceso hijo que ejecuta el mismo progr. que el padre
    - Hereda los ficheros abiertos (se copian los descriptores).



2.5 Operaciones con proc.

árbol de procesos

- Unix mantiene las relaciones padre-hijo
  - El primer proceso es Init (padre del resto)
  - Init lanza procesos demonios de login
  - El login se convierte en shell
  - Desde el shell el usuario lanza procesos



Fuente: Sistemas Operativos. Una visión aplicada.

2.5 Operaciones con proc.

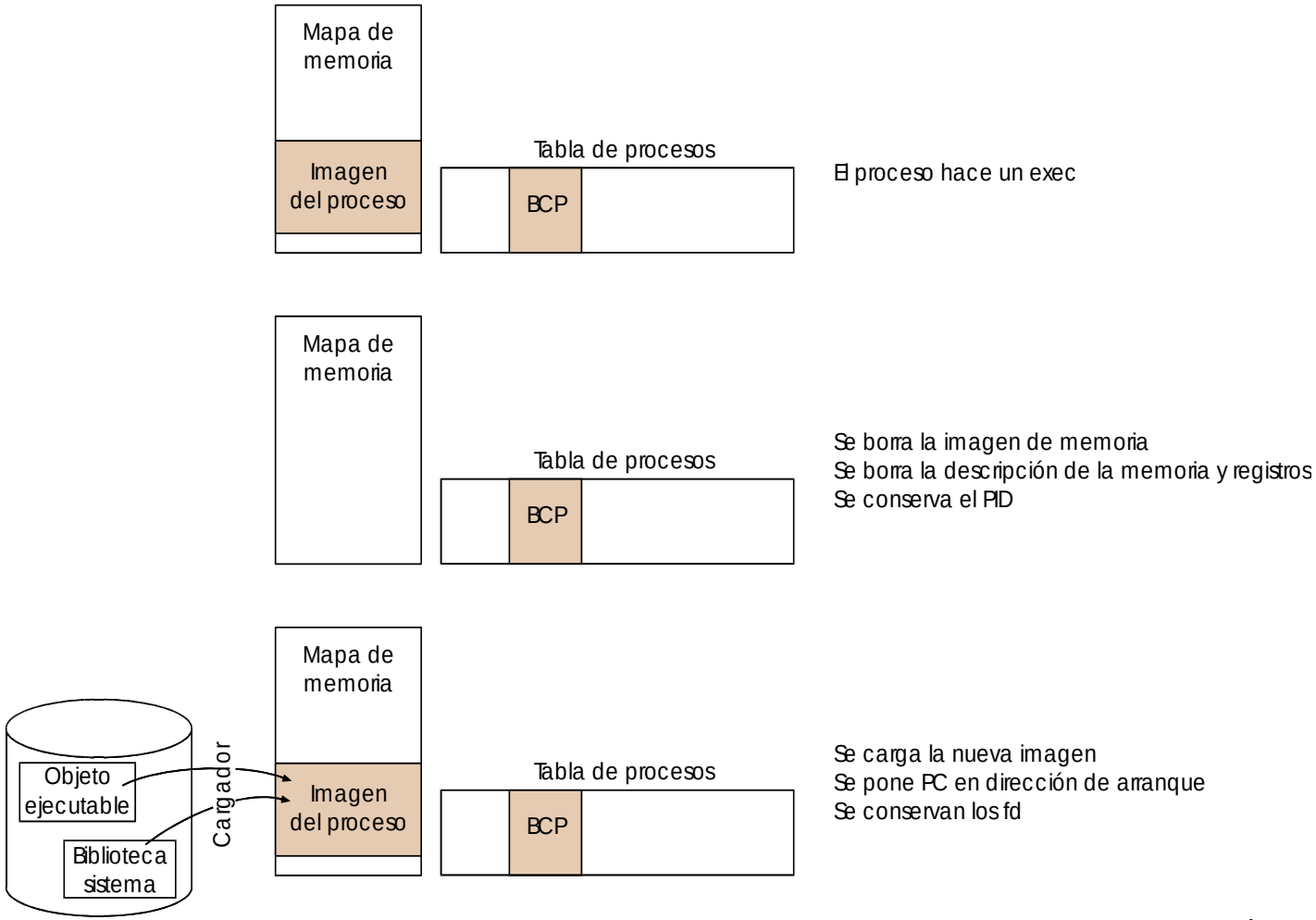
exec y exit

- `int execlp(const char *file, const char *arg, ..)`
  - Argumentos:
    - `path, file`: nombre del archivo ejecutable
    - `arg`: argumentos
  - Descripción:
    - Devuelve -1 en caso de error, en caso contrario no retorna.
    - Cambia la imagen de memoria del proceso.
    - El mismo proceso ejecuta otro programa.
    - Los ficheros abiertos permanecen abiertos
- `int exit(int status);`
  - Argumentos: Código de retorno al proceso padre
  - Descripción:
    - Finaliza la ejecución del proceso.
    - Se cierran todos los descriptores de ficheros abiertos.
    - Se liberan todos los recursos del proceso

2.5 Operaciones con proc.

exec ilustrado

- exec() cambia el código de un proceso

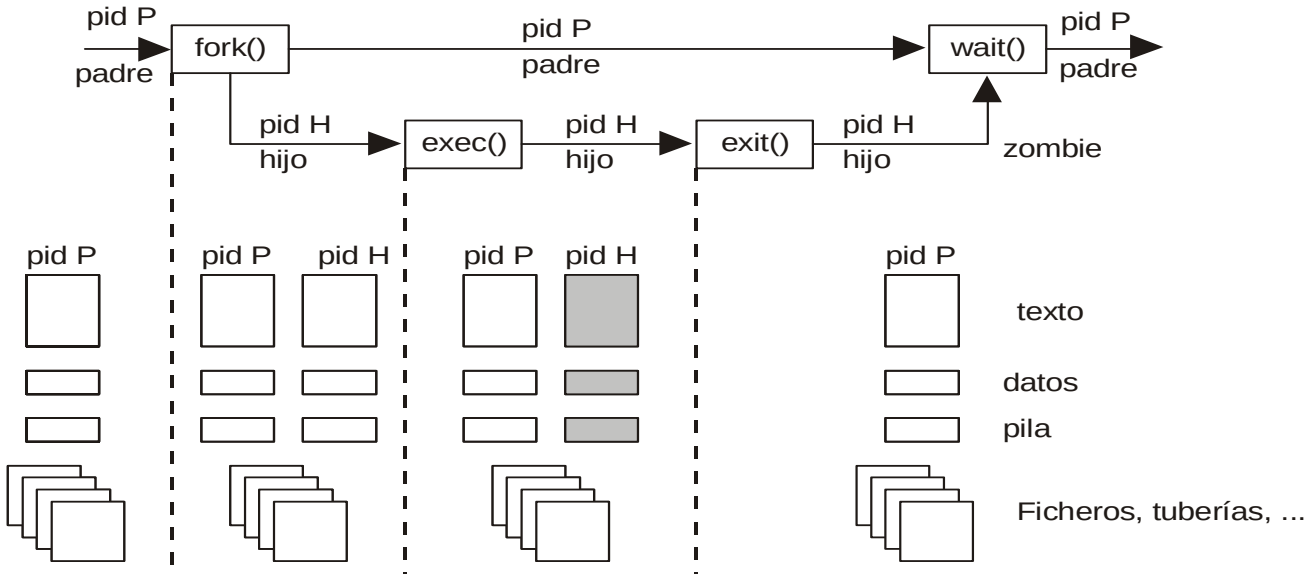


Fuente: Sistemas Operativos. Una visión aplicada.

2.5 Operaciones con proc.

wait

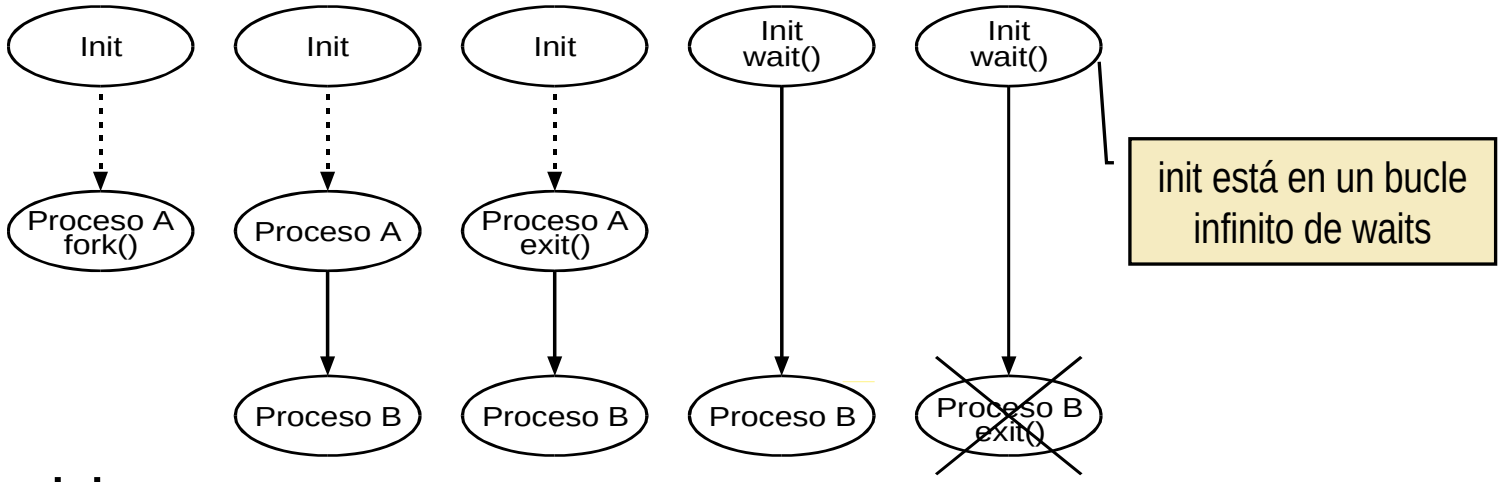
- `pid_t wait(int *status);`
  - Argumentos:
    - Devuelve el código de terminación del proceso hijo.
  - Descripción:
    - Devuelve el identificador del proceso hijo o -1 en caso de error.
    - Permite a un proceso padre esperar hasta que termine un proceso hijo. Devuelve el identificador del proceso hijo y el estado de terminación del mismo.



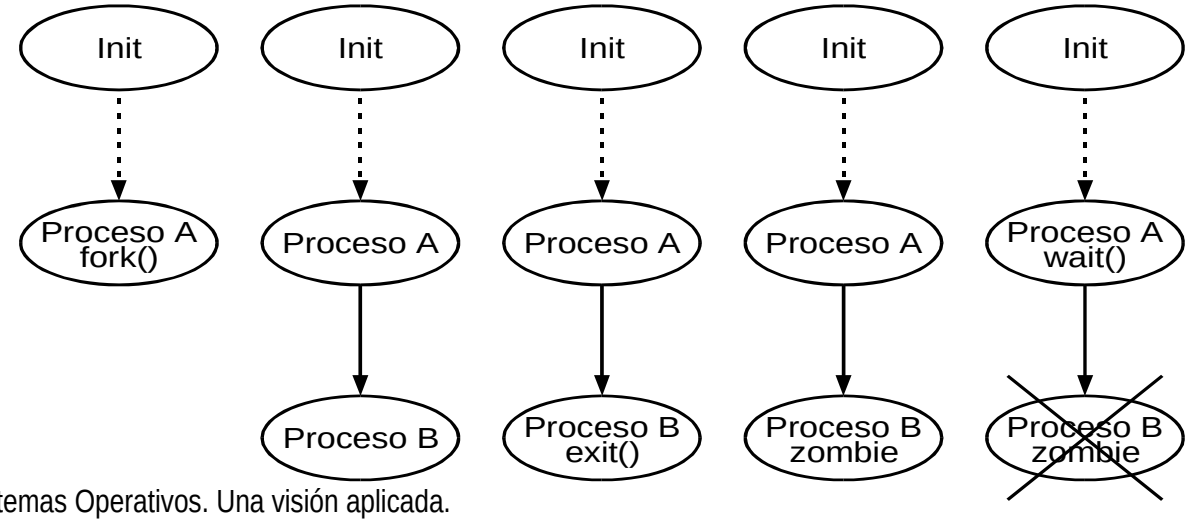
2.5 Operaciones con proc.

estado zombie

- Si el padre muere: init “adopta” a los hijos



- Zombie: desde que el hijo hace exit hasta que el padre hace wait



Fuente: Sistemas Operativos. Una visión aplicada.

2.5 Operaciones con proc.

ejemplo de creación de proceso

- Un proceso que lanza un “ls –l”

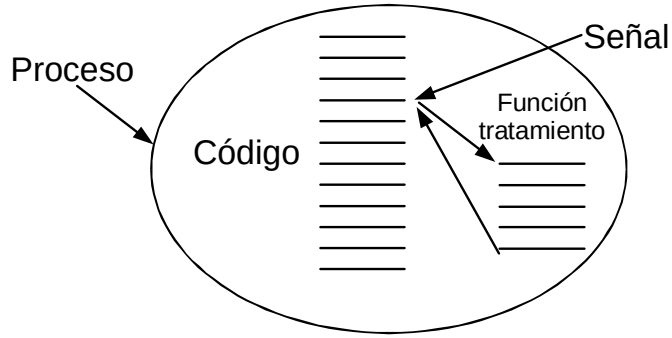
```
#include <sys/types.h>
#include <stdio.h>
/* programa que ejecuta el mandato ls -l */
main() {
   pid_t pidfork;
   int status;
   pidfork = fork();
   if (pidfork == 0)   { /* proceso hijo */
      execlp("ls","ls","-l",NULL);
      exit(-1);
   }
   else       /* proceso padre */
     while (pidfork != wait(&status));
   exit(0);
}
```

devuelve 0 (existo) o cualquier otro valor (fallo)

2.5 Operaciones con proc.

gestión de señales en POSIX

- Las señales son interrupciones al proceso
  - Se envían
    - de proceso a proceso con “kill”
    - del SO a un proceso



Fuente: Sistemas Operativos. Una visión aplicada.

  - Tipos de señales (“`kill -l`” da la lista)
    - SIGKILL -9- (mata un proceso: no enmascarable)
    - SIGTERM -15- (termina un proceso: enmascarable)
    - SIGSTOP -19- (suspende un proceso)
    - SIGCONT (reanuda un proceso)
  - El proceso debe estar preparado para recibir la señal (especifica un proced.) o la enmascara
    - Si no está preparado: muere o la ignora

2.5 Operaciones con proc.

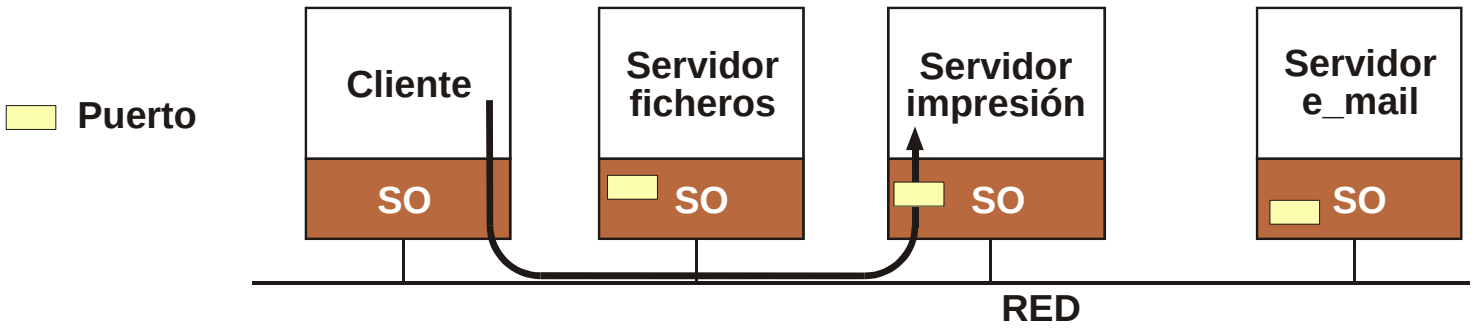
servicios para gestión de señales

- **int kill(pid_t pid, int sig)**
  - Envía al proceso "pid" la señal “sig”
- **int sigaction(int sig, struct sigaction *act, struct sigaction *oact)**
  - Permite especificar la acción a realizar como tratamiento de la señal “sig”
- **int pause(void)**
  - Bloquea al proceso hasta la recepción de una señal.
- **unsigned int alarm(unsigned int seconds)**
  - Genera la recepción de la señal SIGALRM pasados “seconds” segundos.
- **sigprocmask(int how, const sigset_t *set, sigset_t *oset)**
  - Se utiliza para examinar o modificar la máscara de señales de un proceso.

2.5 Operaciones con proc.

procesos demonios

- Es un proceso que ejecuta:
  - En background (su padre no le espera)
  - No asociado a un terminal o proceso login
  - Que espera que ocurra un evento (petición de un cliente)
  - O que debe realizar una tarea de forma periódica
- Características
  - Se arrancan al iniciar el sistema
  - No mueren
  - Están normalmente en espera de evento
  - No hacen el trabajo, lanzan otros procesos o threads



Fuente: Sistemas Operativos. Una visión aplicada.

2.5 Operaciones con proc.

servicios Win32

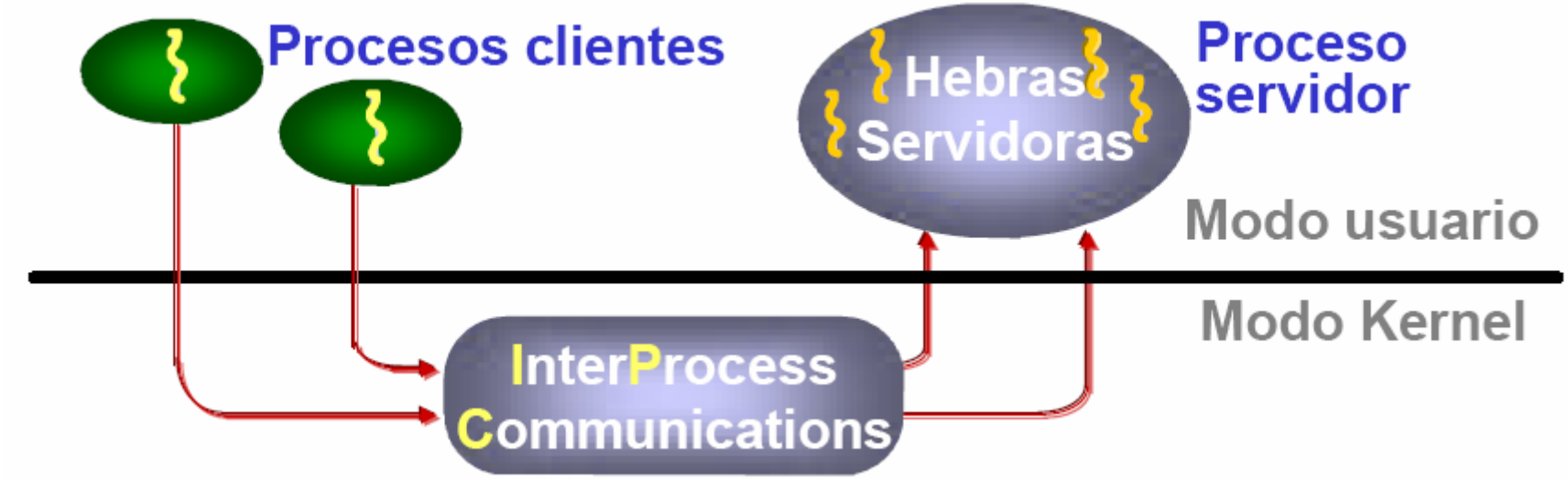
- Servicios Win32:
  - Crear un proceso
    - `BOOL` **`CreateProcess(....);`**
  - Terminar la ejecución de un proceso
    - `VOID` **`ExitProcess`**`(UINT nExitCode);`
  - Obtener el código de term. de un proceso
    - `BOOL` **`GetExitCodeProcess`**`(HANDLE hProcess, LPDWORD lpdwExitCode);`
  - Finalizar la ejecución de otro proceso
    - `BOOL` **`TerminateProcess`**`(HANDLE hProcess, UINT uExitCode);`
  - Esperar por la finalización de un proceso
    - `DWORD` **`WaitForSingleObject`**`(HANDLE hObject, DWORD dwTimeOut);`
    - `DWORD` **`WaitForMultipleObjects`**`(DWORD cObjects, LPHANDLE lphObjects, BOOL fWaitAll, DWORD dwTimeOut);`

2.6 Threads

2.6 Threads

limitaciones de los procesos

- Hay aplicaciones por naturaleza paralelas que necesitan un espacio común de direcciones.
  - Servidores de bases de datos
  - Monitores de procesamiento de transacciones
  - Procesamiento de protocolos de red
  - Etc.

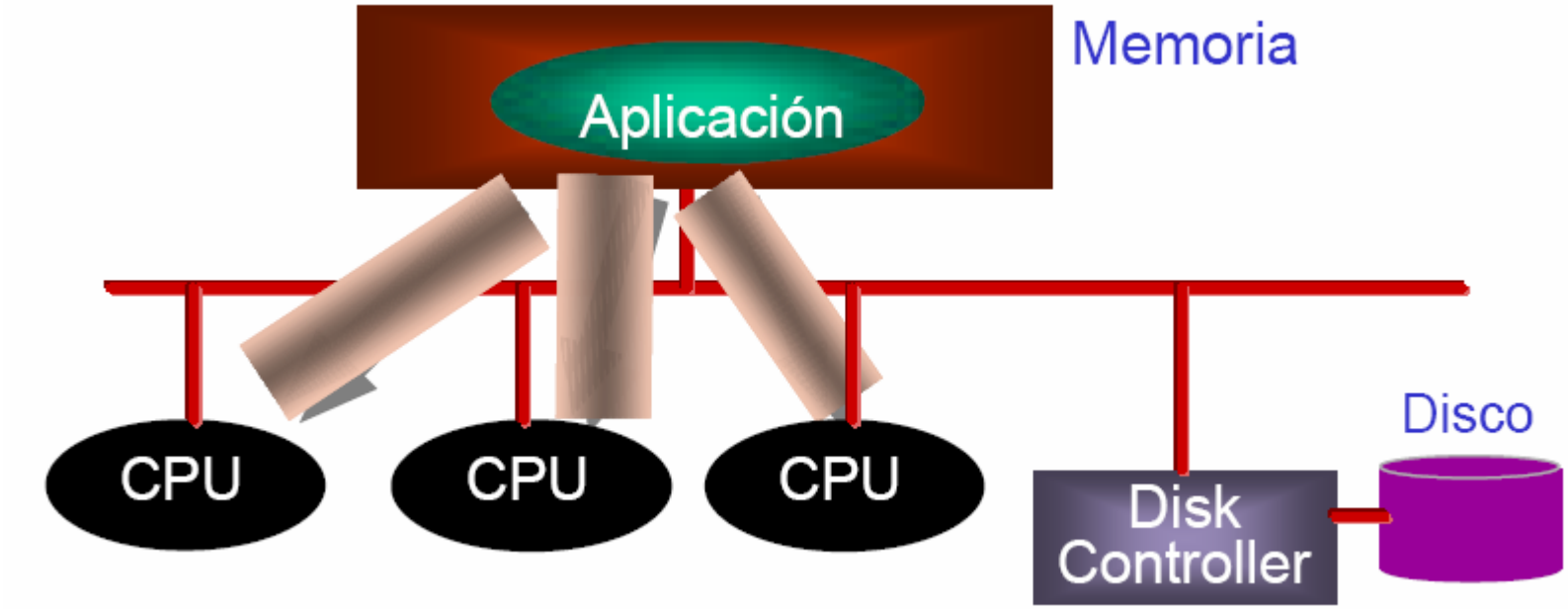


Fuente: Sistemas Operativos. Una visión aplicada.

2.6 Threads

limitaciones de los procesos

- Los procesos no pueden sacar partido de las arquitecturas multi-procesador
  - Un proceso sólo puede usar un procesador a la vez



Fuente: Sistemas Operativos. Una visión aplicada.

2.6 Threads

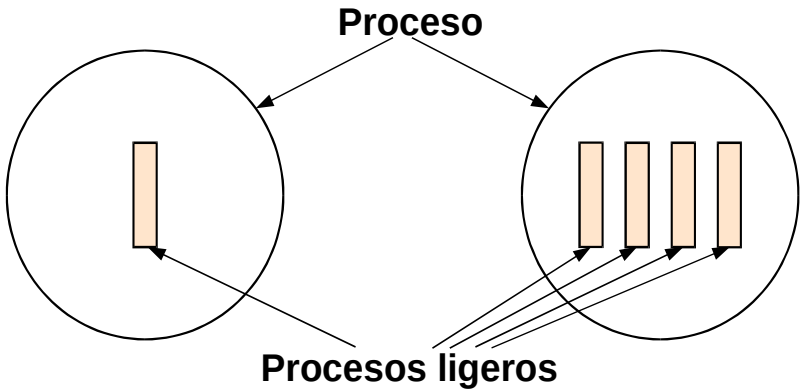
concepto de proceso revisado

- Aspectos separables que confluyen en un proceso:
  - Unidad planificable (flujo de control)
    - Secuencia de instrucciones a ejecutar; determinadas por el PC, la pila y los registros.
  - Unidad propietaria de los recursos
    - Direcciones de memoria a las que se puede acceder y los recursos asignados (archivos, E/S, ...)
- Queremos introducir un modelo de ejecución diferente del modelo de proceso visto pero que sea compatible

Fuente: Sistemas Operativos. Una visión aplicada.

2.6 Threads

- Idea: Permitir más de un flujo de control dentro del mismo espacio de direcciones
  - Dentro del mismo programa (datos y código) y los mismos recursos del SO, permitimos varias ejecuciones.
- Separar las 2 unidades de un proceso
  - Proceso: sólo la unidad propietaria de recursos
  - Thread (hilo): sólo la unidad planificable
- Posibilidad: varios threads en un proceso

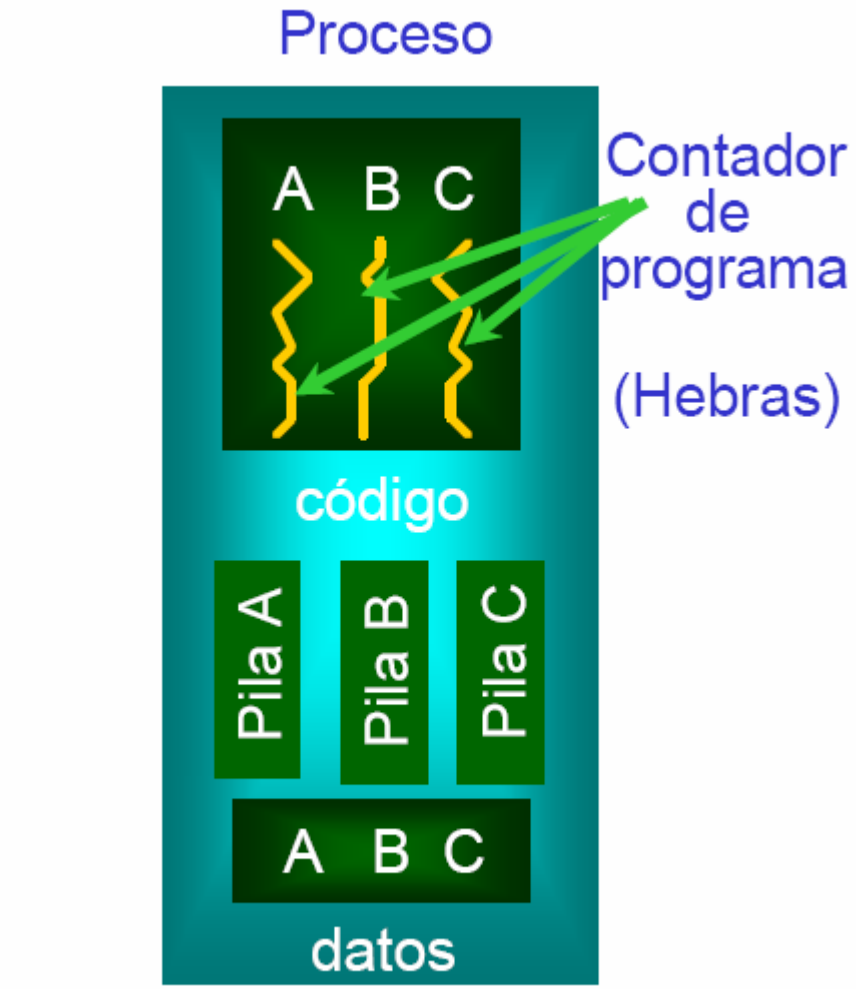


Fuente: Sistemas Operativos. Una visión aplicada.

2.6 Threads

definición

- Hebra: Unidad de asignación de la CPU (de planificación)
- Tiene su propio contexto hardware
  - Su valor del contador de programa
  - Los valores de los registros
  - Su pila de ejecución



" Programa multihebrado"

Fuente: Sistemas Operativos. Una visión aplicada.

2.6 Threads

información de los threads

- Información propia del thread
  - Estado de ejecución del thread (run, ready, block)
  - Contexto del thread (guardado cuando no se está ejecutando): contador de programa, registros.
  - Pila
  - Almacenamiento estático per-thread para variables locales
- Todos los threads comparten el acceso a la memoria y recursos de su proceso:
  - Espacio de memoria
  - Variables globales
  - Archivos abiertos
  - Procesos hijos
  - Señales

2.6 Threads

información de los threads

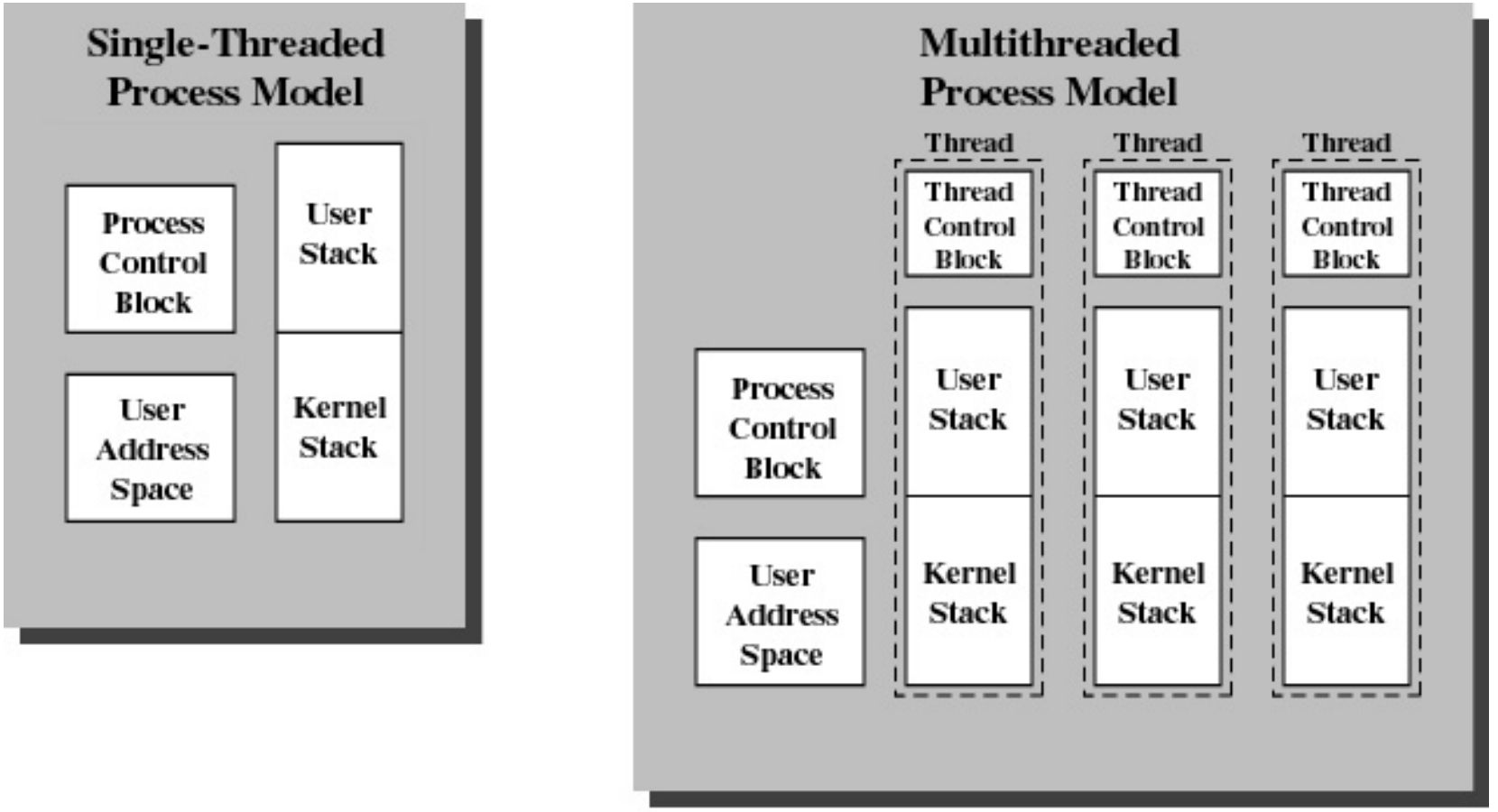


Figure 4.2 Single Threaded and Multithreaded Process Models

2.6 Threads

multithreading

- El sistema operativo permite múltiples threads de ejecución por cada proceso
- MS-DOS sólo tiene un thread
- UNIX permite múltiples procesos de usuario, pero sólo un thread por proceso
- Windows XP, Solaris y Linux soportan múltiples threads.

2.6 Threads

multithreading

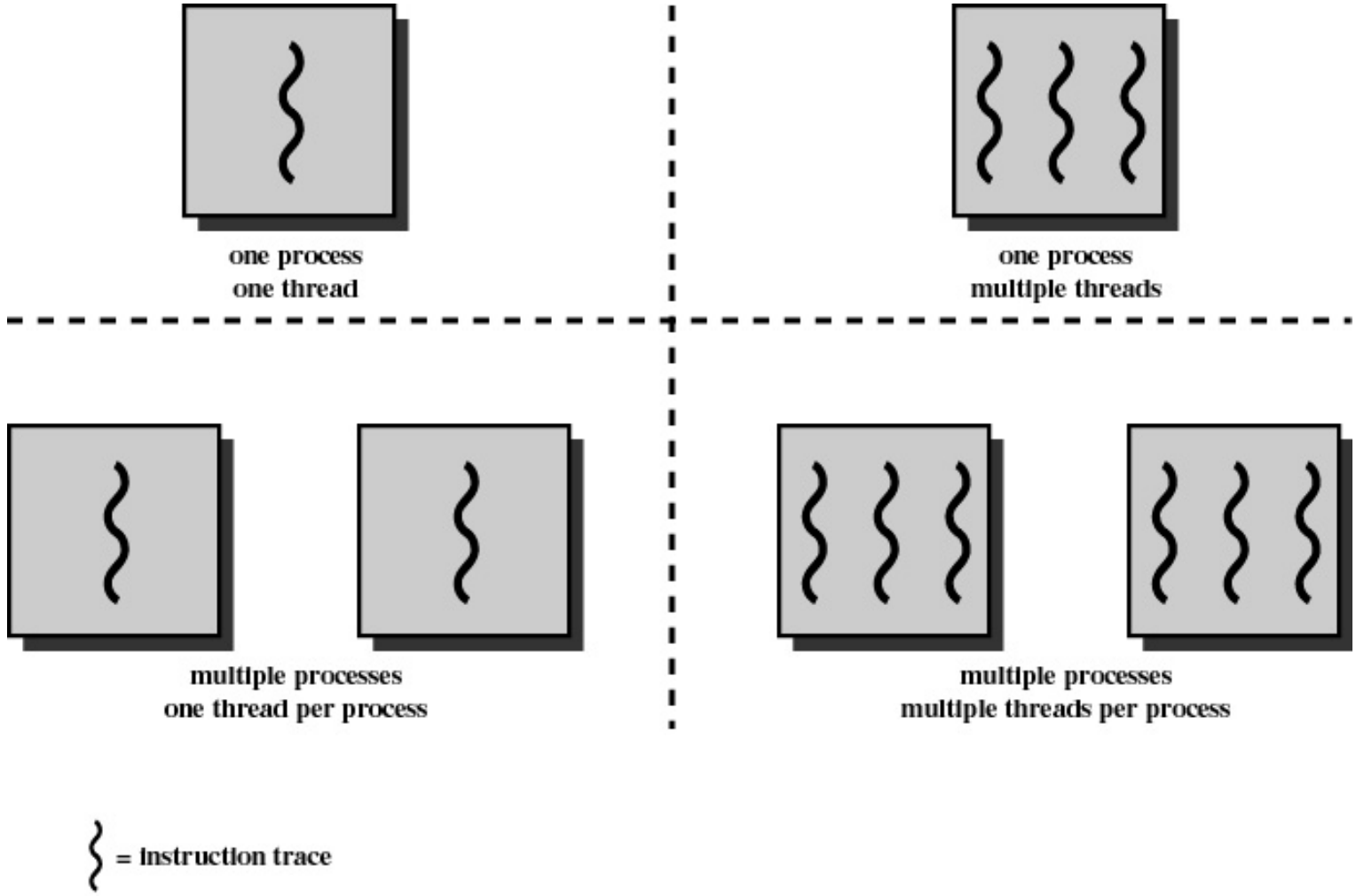


Figure 4.1 Threads and Processes [ANDE97]

2.6 Threads

tipos de threads

- Tipos de threads:
  - Threads a nivel de usuario
  - Threads a nivel de kernel
  - Enfoques combinados
- Threads a nivel de Usuario
  - Toda la gestión de los threads se realiza desde la aplicación (a través de biblioteca de usuario que actúa como un 'kernel en miniatura')
  - Ventajas:
    - Alto rendimiento al no consumir recursos kernel (no hacen llamadas al sistema).
    - Conmutación entre threads muy rápida: el switch no necesita el paso a modo kernel (evita el 'doble switch').
    - Planificación específica a la aplicación (no depende del SO).
    - Funciona en cualquier SO mientras esté disponible la librería
  - Desventajas:
    - Si un thread se bloquea, bloquea al proceso completo
    - No puede sacar provecho de un multiprocesador

2.6 Threads

tipos de threads

- Threads a nivel de Kernel
  - Ejemplos: Windows XP, Linux y OS/2
  - El kernel mantiene información de contexto de los procesos y threads
  - La planificación se realiza en base a los threads
  - Ventajas:
    - un thread en espera de E/S no bloquea al resto de threads del mismo proceso
    - puede aprovecharse de los multiprocesadores
  - Desventajas:
    - el switch es más lento que que el caso de threads a nivel de usuario
    - la planificación la fija el SO.

2.6 Threads

tipos de threads

- Enfoques combinados
  - Implementan threads de kernel y de usuario
  - La creación de threads se hace en el espacio de usuario
  - La mayor parte de la planificación y sincronización de threads se hace en el espacio de usuario
  - Ejemplo: Solaris 2.x

2.6 Threads

tipos de threads

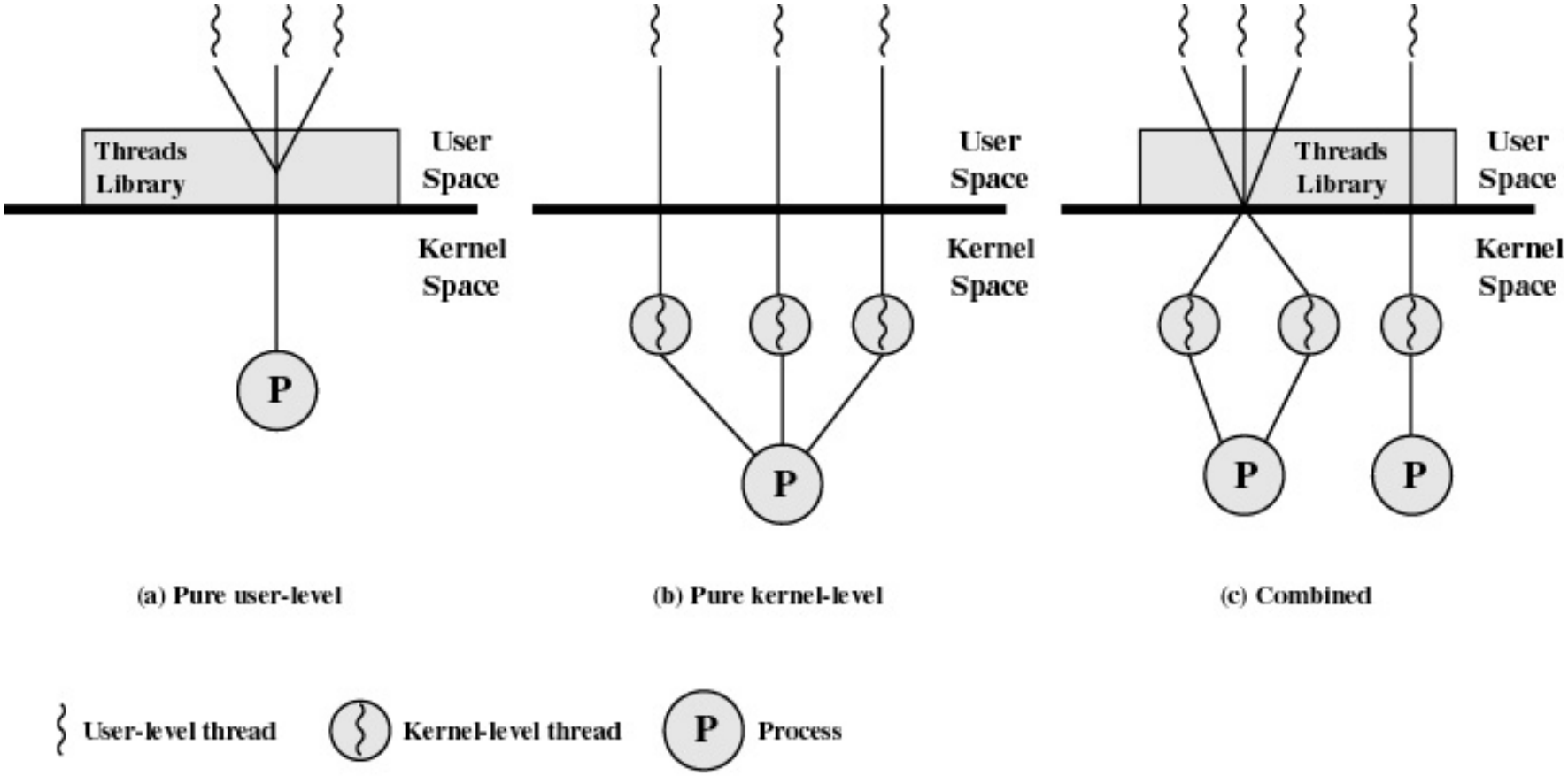


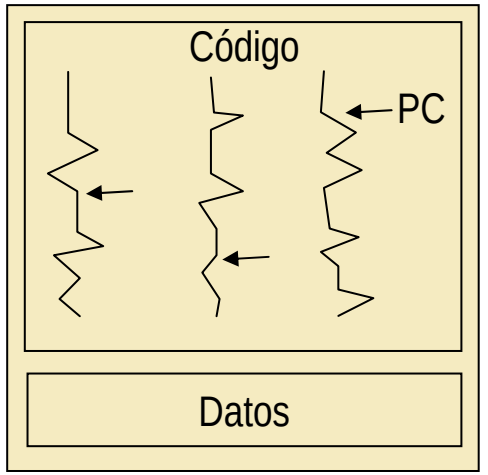
Figure 4.6 User-Level and Kernel-Level Threads

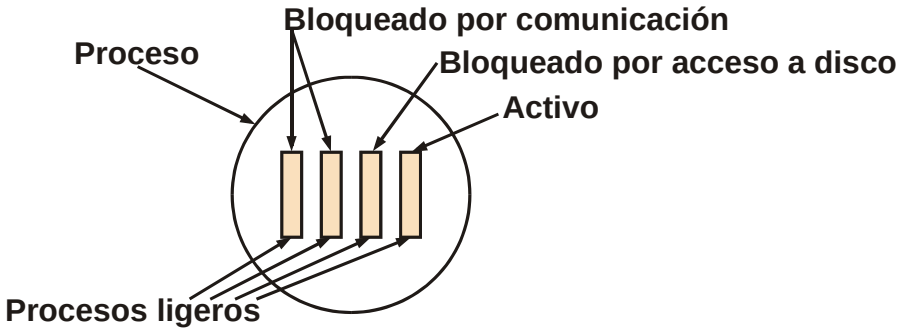
2.6 Threads

estados de los threads

- Estados del thread: run, ready, block
  - Threads kernel: varios procesadores -> varios en run
- Estado del proceso: Depende de los threads que contiene

Threads a nivel de usuario	Threads a nivel de kernel
Run: Si tiene algún thread en ejecución y ninguno bloqueado	Run: si tiene algún thread en ejecución
Ready: Todos los threads están ready (no hay ningún thread en run ni bloqueado)	Ready: si no hay ningún thread en run y alguno/s ready
Block: si hay algún thread bloqueado	Block: si todos los thread están blocked





Fuente: Sistemas Operativos. Una visión aplicada.

2.6 Threads

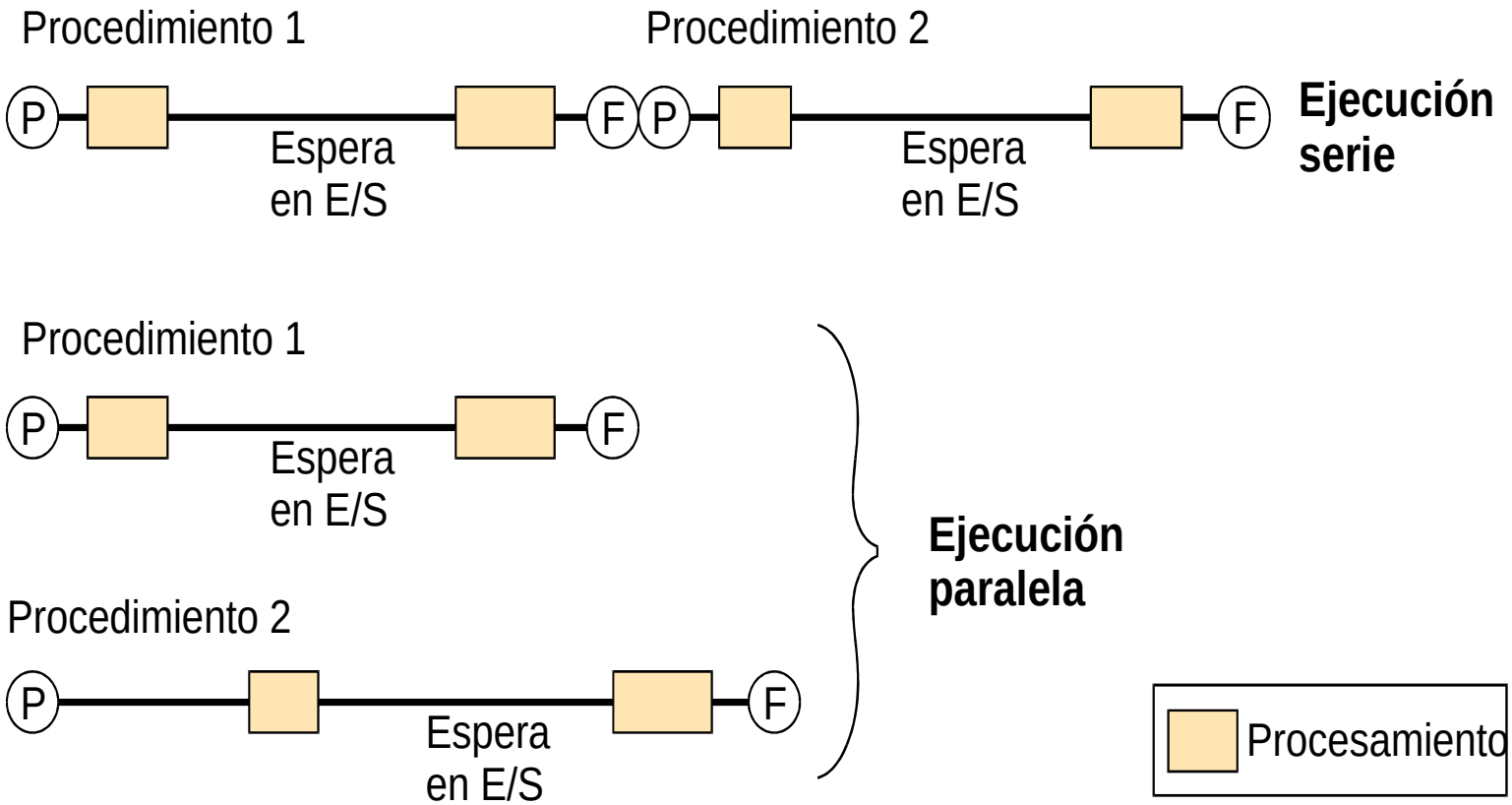
ventajas de los threads

- Ventajas del uso de threads
  - Conmutación de threads más rápida
  - Crear y terminar threads es más rápido
  - Baja el n° de conmutación de procesos
  - Simplifica la programación
    - Cuando un algoritmo tiene varias líneas de ejecución
    - Ejemplo: un simulador de vuelo
  - Mejora la interactividad
    - Mientras un thread atiende al usuario otro procesa
  - Comunicación eficiente entre threads
    - Ya que comparten memoria
  - Eficiencia en modelos cliente-servidor
  - Soporte SMP (symmetric multiprocessing)

2.6 Threads

paralelismo con threads

- Paralelismo

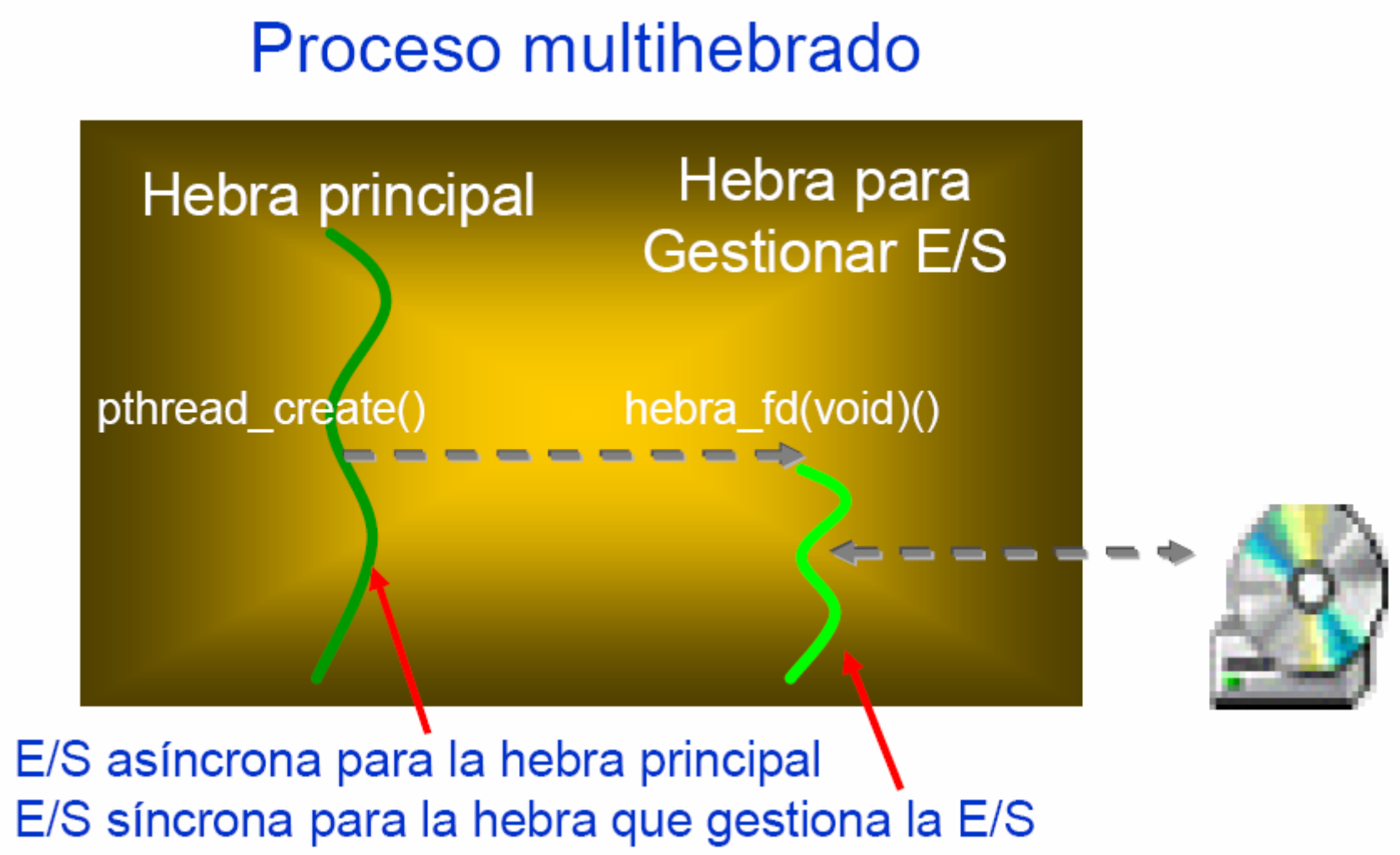


Fuente: Sistemas Operativos. Una visión aplicada.

2.6 Threads

E/S asíncrona

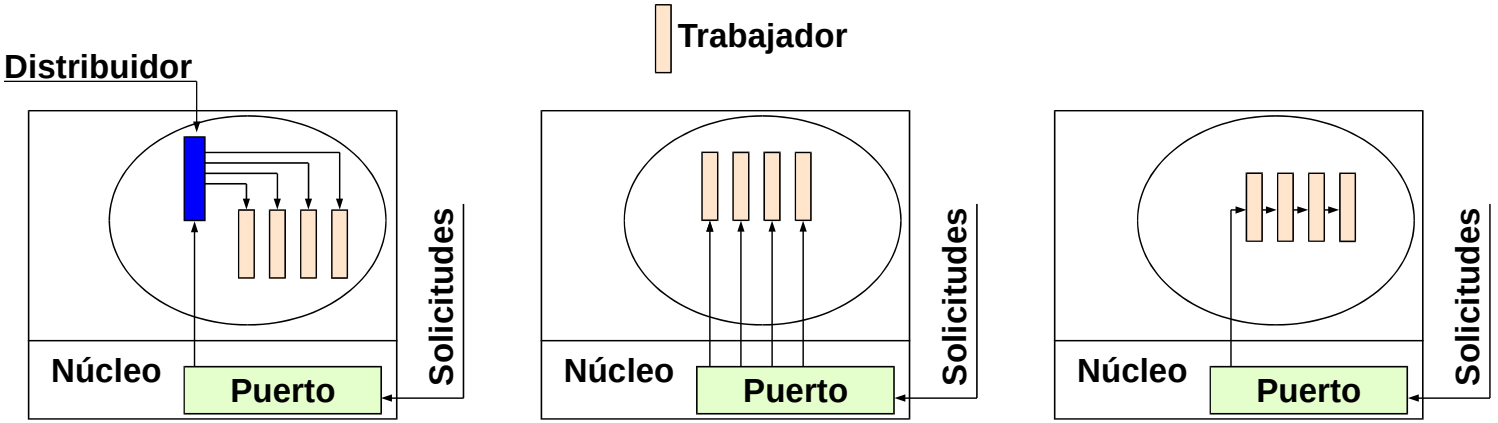
- E/S asíncrona



2.6 Threads

servidores con threads

- Diseño de servidores con threads
  - Thread distribuidor
    - Cuando llega una petición crea un hijo para hacer la tarea
  - Todos los threads iguales
    - Cuando llega una solicitud la atiende un thread libre
  - Segmentación
    - Cada thread hace una parte del trabajo (pipeline)



Fuente: Sistemas Operativos. Una visión aplicada.

2.6 Threads

estándares de threads

- POSIX (Pthreads)
  - ISO/IEEE estándar
  - API común
  - Casi todos los UNIX tiene una biblioteca de threads
- Win32
  - Muy diferente a POSIX
  - Existen bibliotecas comerciales de POSIX
- Solaris
  - Anterior a POSIX
- Hebras Java

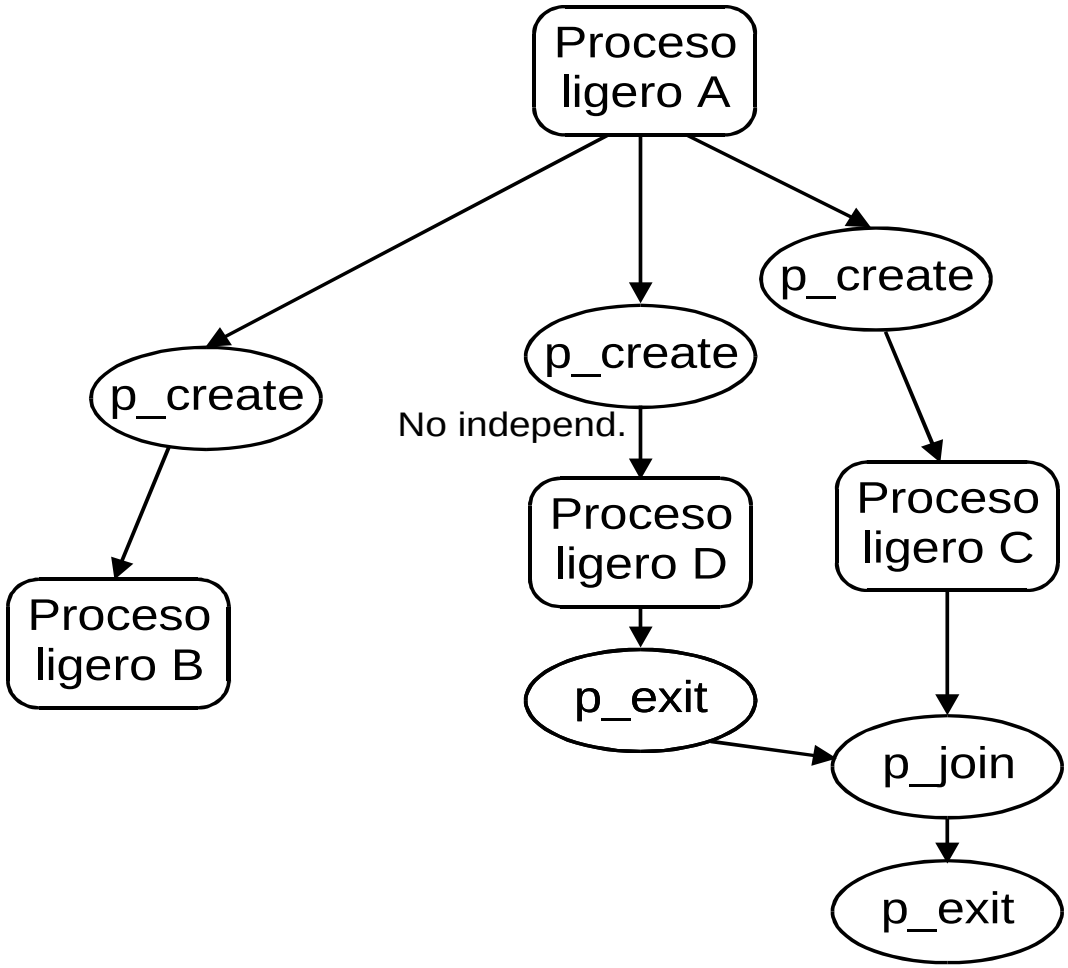
2.6 Threads

servicios POSIX para threads

- **int pthread_create** (pthread_t *thread, const pthread_attr_t *attr, void *(*func)(void *), void *arg)
  - Crea un proceso ligero que ejecuta "func" con argumento "arg" y atributos "attr".
  - Los atributos permiten especificar: tamaño de la pila, prioridad, política de planificación, etc.
  - Existen diversas llamadas para modificar los atributos.
- **int pthread_join** (pthread_t thid, void **value)
  - Suspende la ejecución de un proceso ligero hasta que termina el proceso ligero con identificador "thid".
  - Devuelve el estado de terminación del proceso ligero.
- **int pthread_exit** (void *value)
  - Permite a un proceso ligero finalizar su ejecución, indicando el estado de terminación del mismo.
- **pthread_t pthread_self** (void)
  - Devuelve el identificador del thread que ejecuta la llamada.

2.6 Threads

jerarquía de threads



Fuente: Sistemas Operativos. Una visión aplicada.

2.6 Threads

servicios Win32 para threads

- Crear un proceso ligero
  - BOOL **CreateThread** (
    LPSECURITY_ATTRIBUTES lpsa,
    DWORD cbStack,
    LPTHREAD_START_ROUTINE lpStartAddr;
    LPVOID lpvThreadParam,
    DWORD fdwCreate,
    LPDWORD lpIdThread);
- Terminar un proceso ligero
  - VOID **ExitThread**(DWORD dwExitCode);

2.7 Ejemplos

2.7 Ejemplos

estados de los procesos UNIX

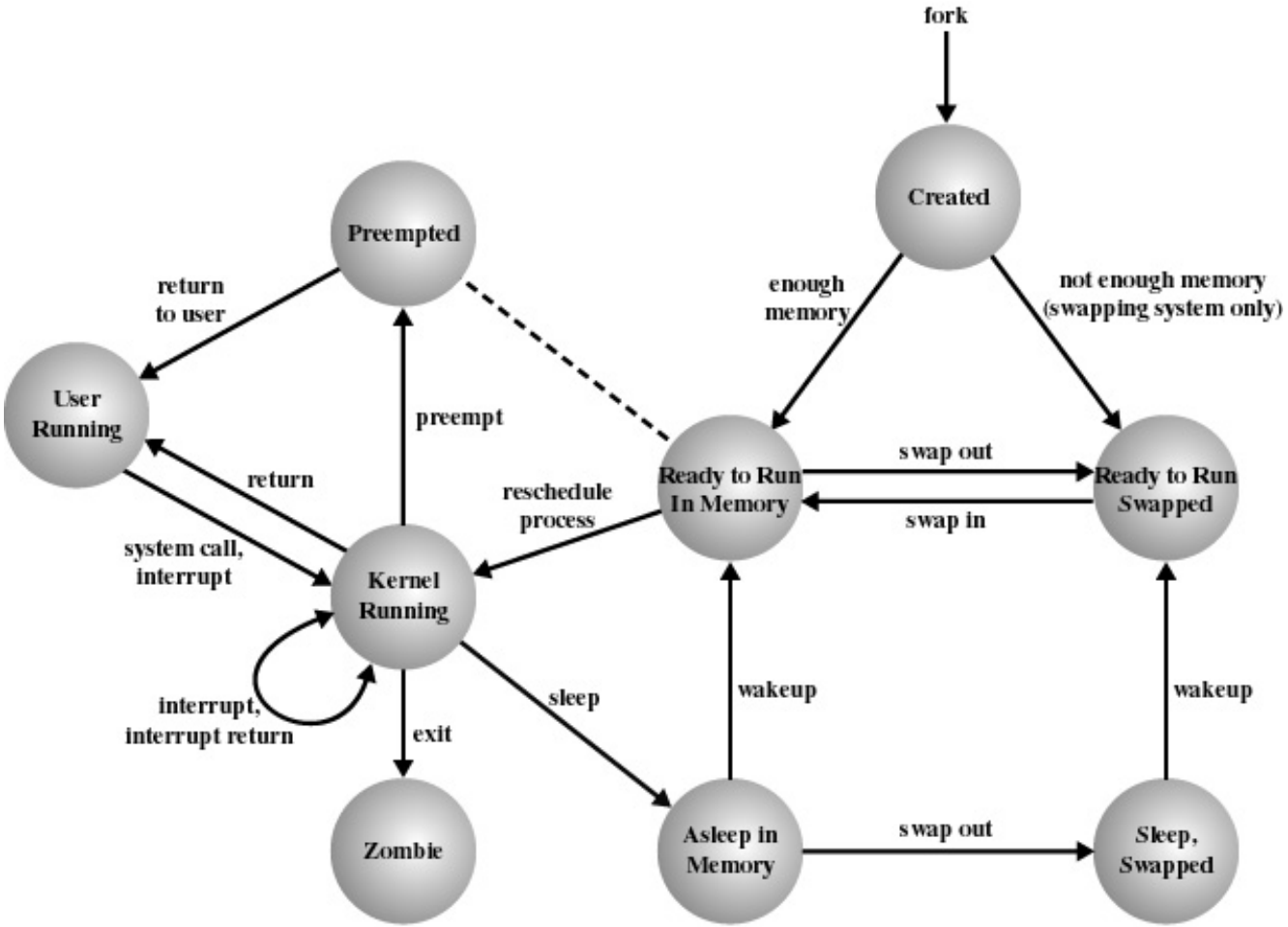


Figure 3.16 UNIX Process State Transition Diagram

Fuente: W. Stallings. Sistemas Operativos

2.7 Ejemplos

estados de los procesos UNIX

User Running	Executing in user mode.
Kernel Running	Executing in kernel mode.
Ready to Run, in Memory	Ready to run as soon as the kernel schedules it.
Asleep in Memory	Unable to execute until an event occurs; process is in main memory (a blocked state).
Ready to Run, Swapped	Process is ready to run, but the swapper must swap the process into main memory before the kernel can schedule it to execute.
Sleeping, Swapped	The process is awaiting an event and has been swapped to secondary storage (a blocked state).
Preempted	Process is returning from kernel to user mode, but the kernel preempts it and does a process switch to schedule another process.
Created	Process is newly created and not yet ready to run.
Zombie	Process no longer exists, but it leaves a record for its parent process to collect.

2.7 Ejemplos

estados de los procesos Linux

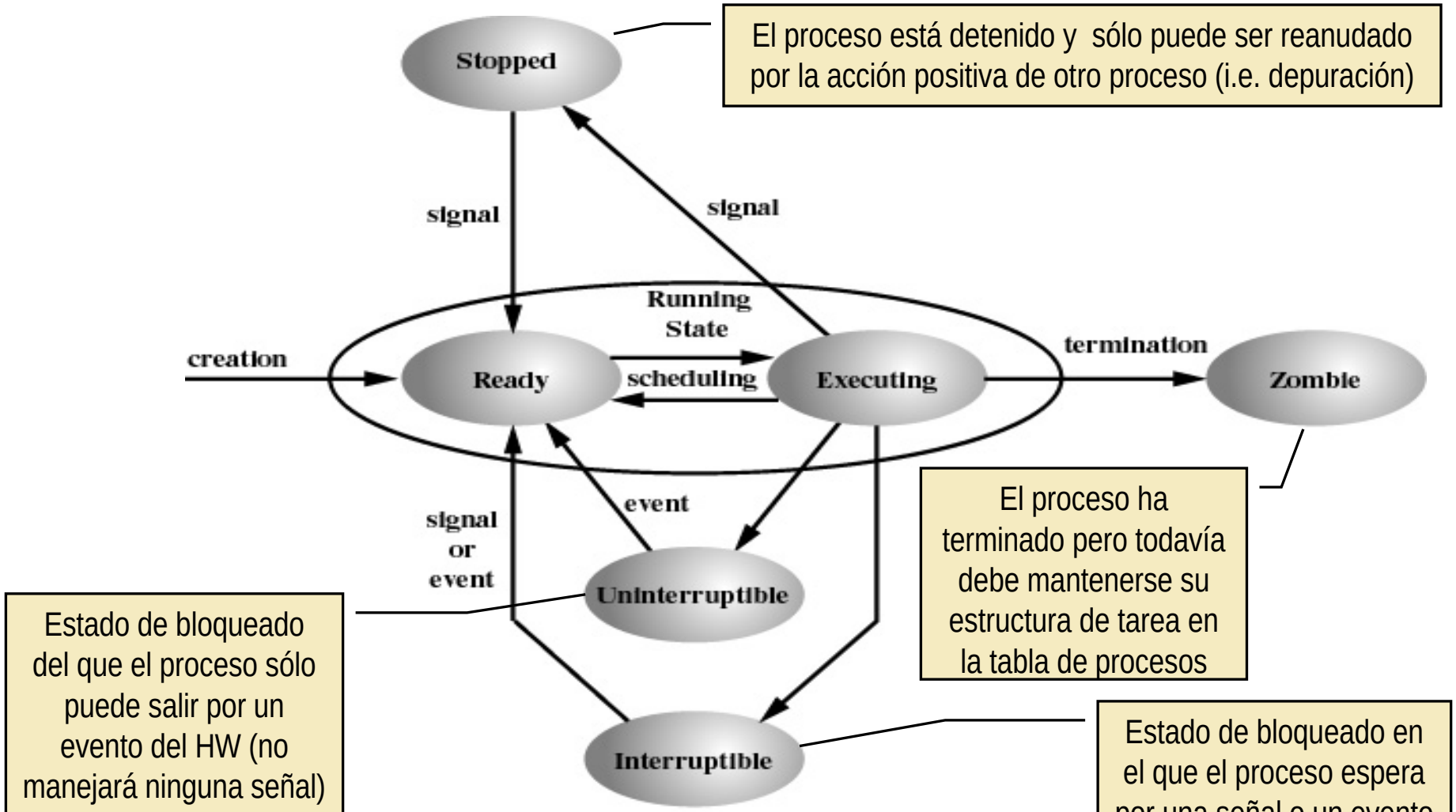


Figure 4.18 Linux Process/Thread Model

Fuente: W. Stallings. Sistemas Operativos

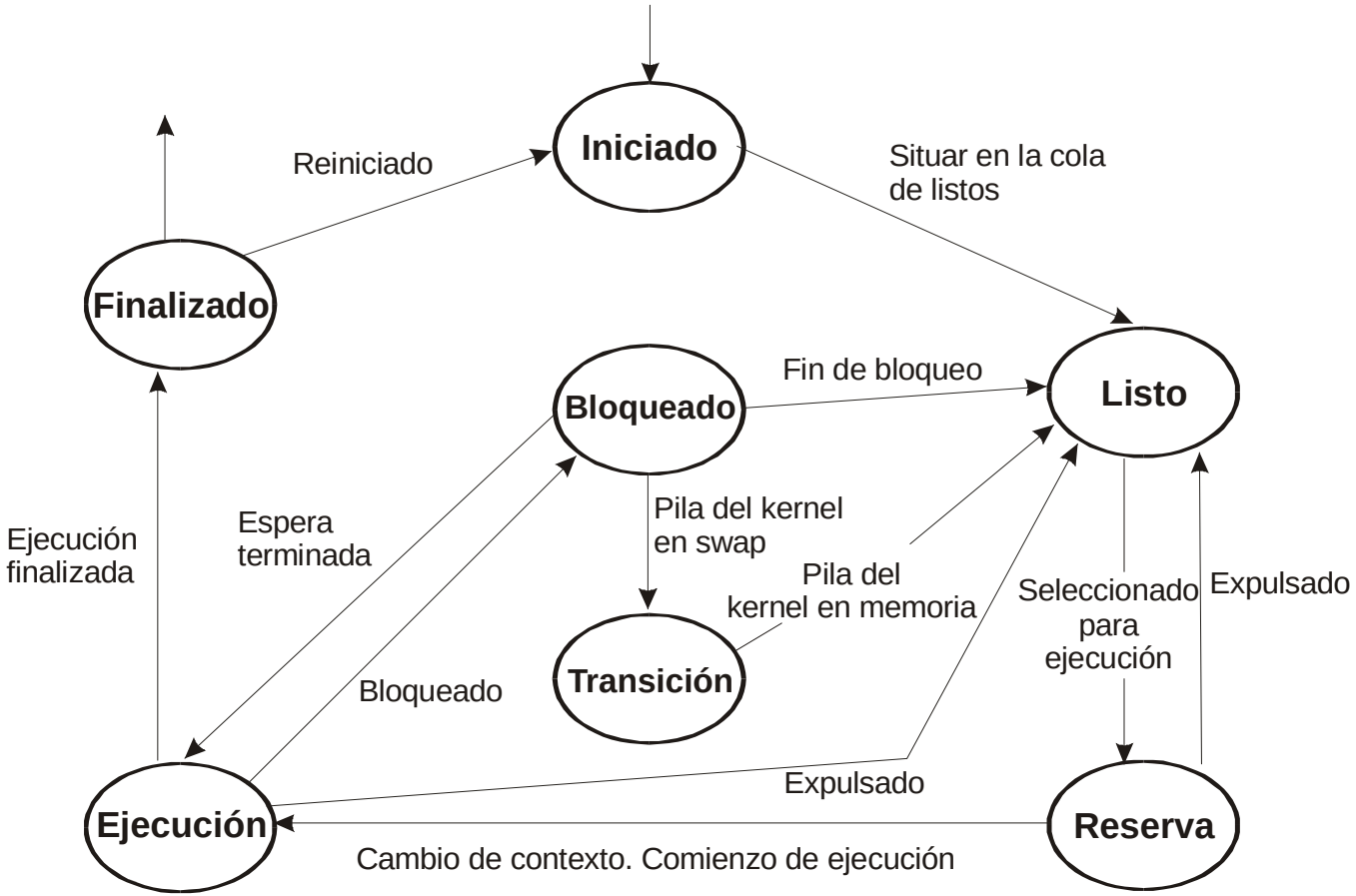
2.7 Ejemplos

PCB en Linux

- Estructura de datos task_struct
  - Estado: Run, Ready, Block, Stoped, Zombie
  - Estado del procesador: registros
  - Información de planificación: el proceso puede ser normal o de tiempo real, y tener una prioridad. Además hay un contador que lleva la cuenta de la cantidad de tiempo que el proceso ha estado ejecutando.
  - Identificadores: de proceso, usuario y grupo (asignación de privilegios de acceso a recursos a un grupo de procesos)
  - IPC: Inter-Process Communication, mecanismo encontrado en UNIX SVR4
  - Vínculos con padre, hijos y hermanos
  - Tiempos y temporizadores: tiempo de creación del proceso, cantidad de tiempo de CPU consumida hasta el momento, temporizadores asociados.
  - Sistema de archivos: punteros a los archivos abiertos, al directorio actual y al raíz para el proceso.
  - Espacio de direcciones: define el espacio de direcciones virtual asignado al proceso.

2.7 Ejemplos

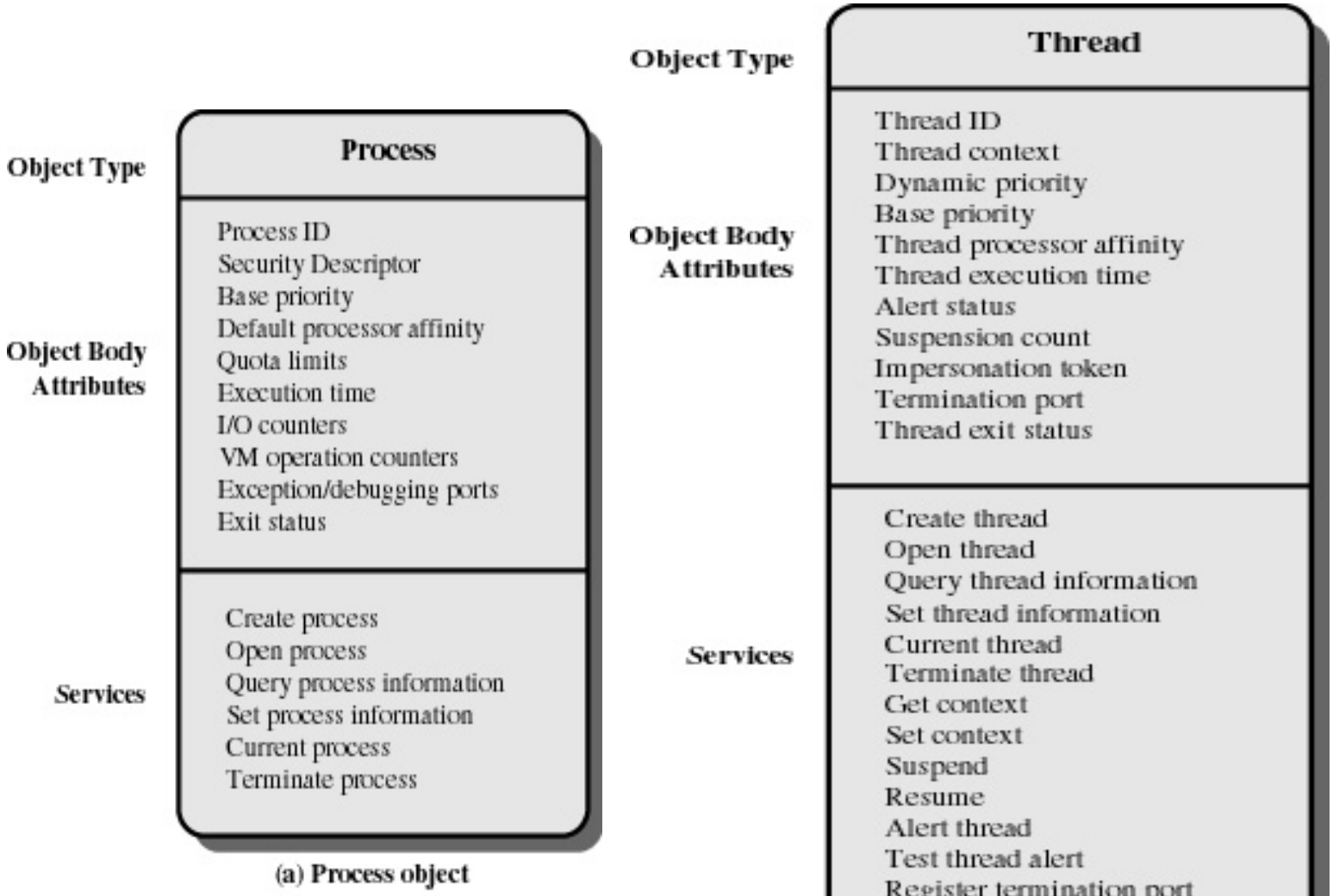
estados de los procesos Windows XP



Fuente: Sistemas Operativos. Una visión aplicada.

2.7 Ejemplos

PCB en Windows XP



(a) Process object

(b) Thread object

Fuente: W. Stallings. Sistemas Operativos