

Sistemas Operativos

Grado en Ingeniería Informática
2018-2019

Procesos e hilos

Basado en:

Sistemas Operativos
J. Carretero [et al.]

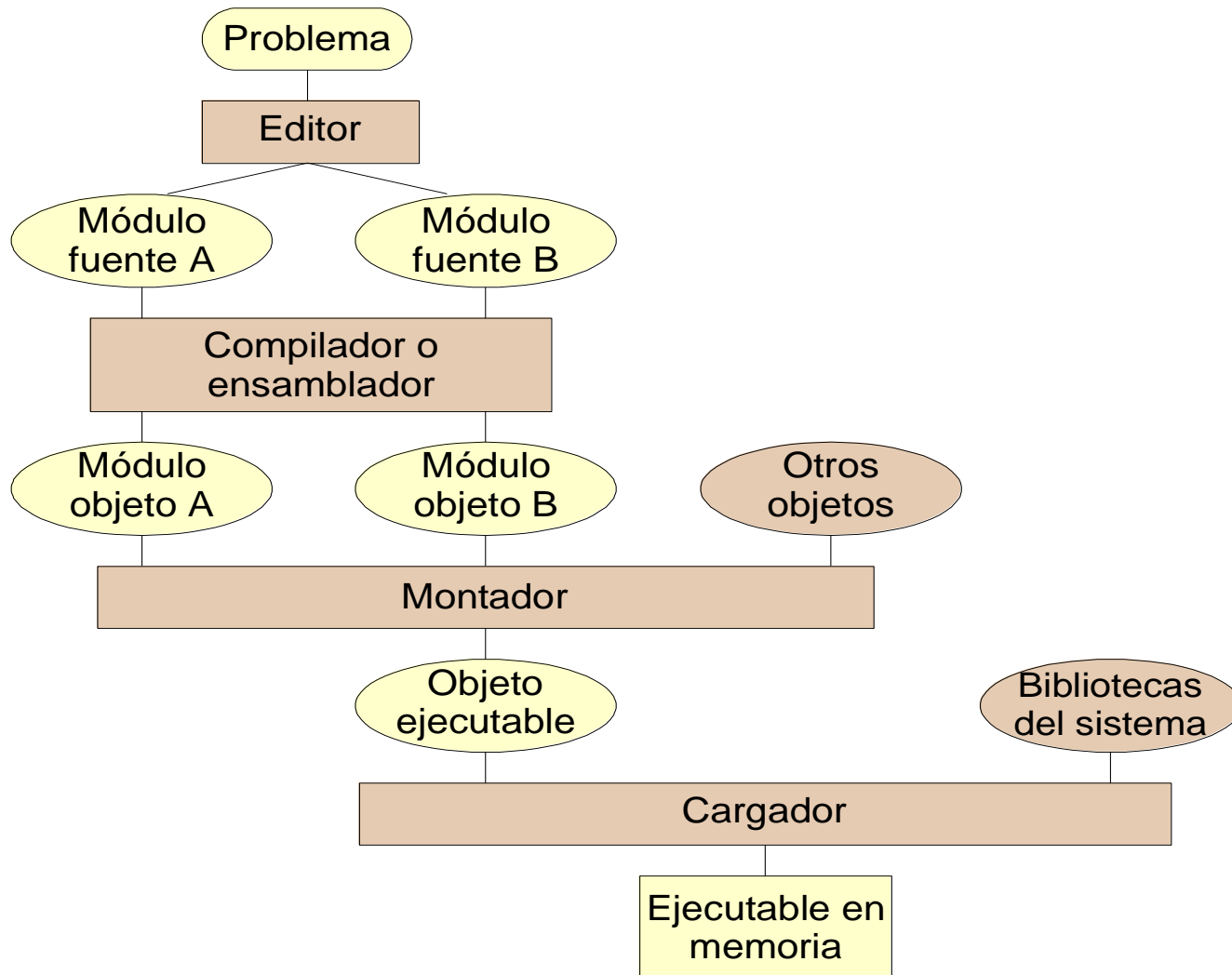
Contenido

- Procesos
- Multitarea
- Información del proceso
- Formación y estados de un proceso
- Señales
- Hilos o *threads*

Concepto de proceso

- Programa: fichero ejecutable en un dispositivo de almacenamiento permanente
- Proceso:
 - Programa en ejecución
 - Unidad de procesamiento gestionada por el SO
 - En realidad, la unidad *mínima* de procesamiento es el hilo (*thread*) → Un proceso puede constar de uno o varios hilos
- Información del proceso:
 - Imagen de memoria: *core image*
 - Estado del procesador: registros del modelo de programación
 - Bloque de control del proceso **BCP**

Preparación del código de un proceso



Recuerda: comandos útiles

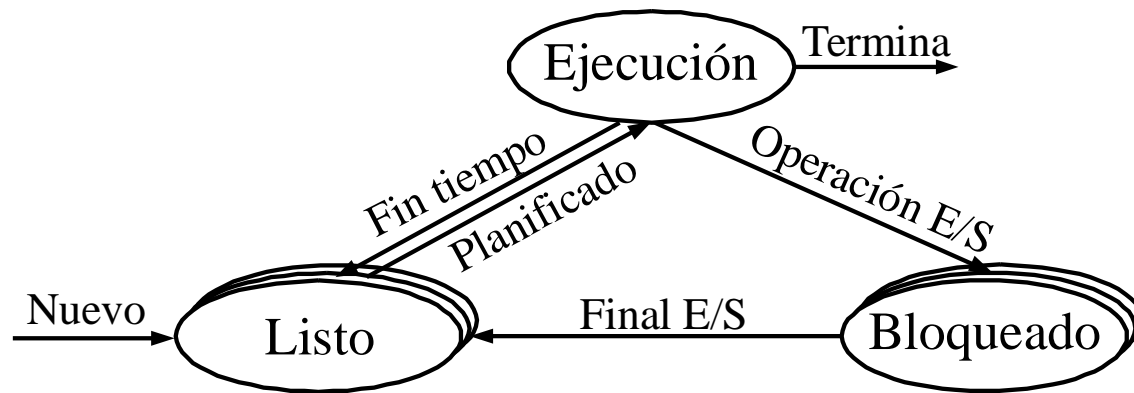
- gcc
 - Compilador C de GNU
 - Realiza todas las etapas
 - > `gcc -save-temps hello.c -o hello.o`
- ldd
 - Permite ver las librerías con las que hemos enlazado
- nm / objdump
 - Permiten visualizar partes de un ejecutable
 - Opciones típicas de objdump: -S, -t, -f, -h

Curiosidades Linux: /proc

- Directorio /proc contiene un directorio por cada proceso en ejecución
- Permite consultar información sobre el proceso:
 - Línea de comando
 - Mapa de memoria
 - Tabla de páginas
 - ...

Estados básicos de un proceso

- En ejecución (uno por procesador/core)
- Bloqueado (en espera de completar E/S o por motivos de sincronización)
- Listo para ejecutar



- Planificador: Componente del SO que decide qué proceso se ejecuta en cada procesador y en qué instante
- Proceso nulo o *idle* (uno por cada procesador)

Entorno del proceso

- Tabla *NOMBRE-VALOR* que se pasa al proceso en su creación
- Se establece:
 - Por defecto (heredados del proceso padre)
 - Mediante comandos del *shell* (*export NOMBRE=valor*)
 - Mediante funciones de la biblioteca estandar de “C” (*putenv*, *getenv*)
- Ejemplo:
PATH=/usr/bin:/home/pepe/bin
TERM=vt100
HOME=/home/pepe
PWD=/home/pepe/libros/primero
TIMEZONE=EDT

Jerarquía de procesos (UNIX)

■ Familia de procesos

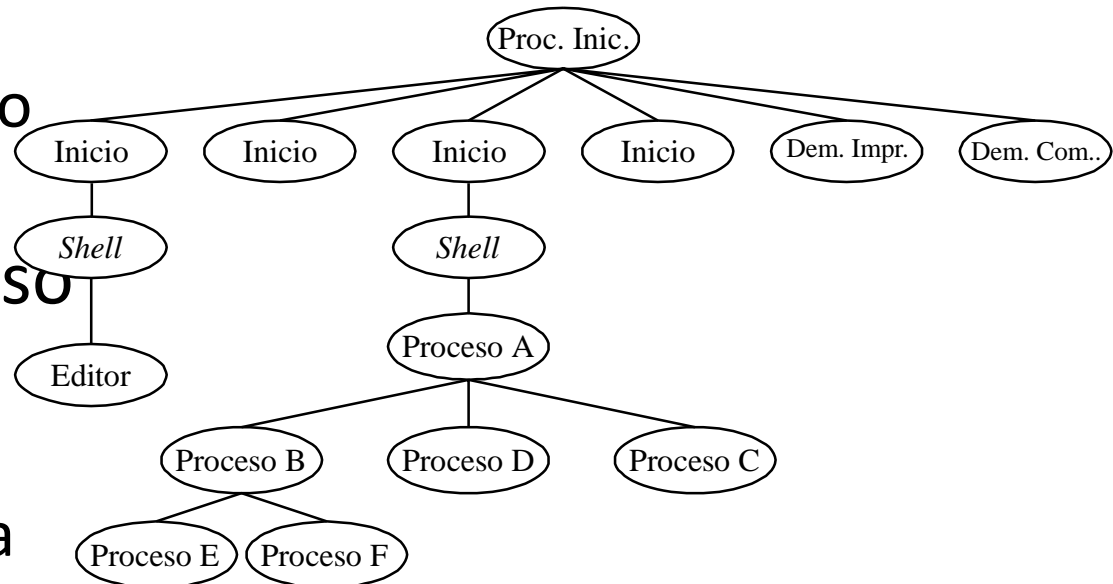
- Proceso hijo
- Proceso padre
- Proceso hermano
- Proceso abuelo

■ Vida de un proceso

- Crea
- Ejecuta
- Muere o termina

■ Ejecución del proceso

- Batch
- Interactivo



Consulta procesos en ejecución

- `ps`
 - Permite ver la información de todos los procesos en ejecución
 - *man ps* para consultar las múltiples opciones
- `top`
 - Muestra los procesos en ejecución, refrescando la información periódicamente
 - Permite interaccionar con los procesos (enviar señales)

Usuario

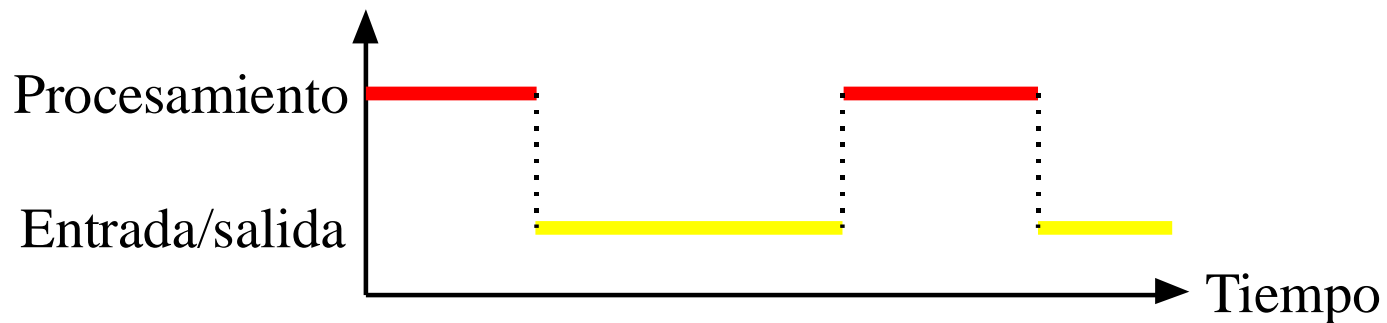
- Usuario: Persona autorizada a utilizar un sistema
 - Se identifica en la autenticación mediante:
 - Código de cuenta
 - Clave (*password*)
 - Internamente el SO le asigna el “uid” (*user identification*)
- Superusuario (*root*)
 - Tiene todos los derechos
 - Administra el sistema
- Grupo de usuarios
 - Los usuarios se organizan en grupos
 - Alumnos
 - Profesores
 - Todo usuario ha de pertenecer al menos a un grupo

Contenido

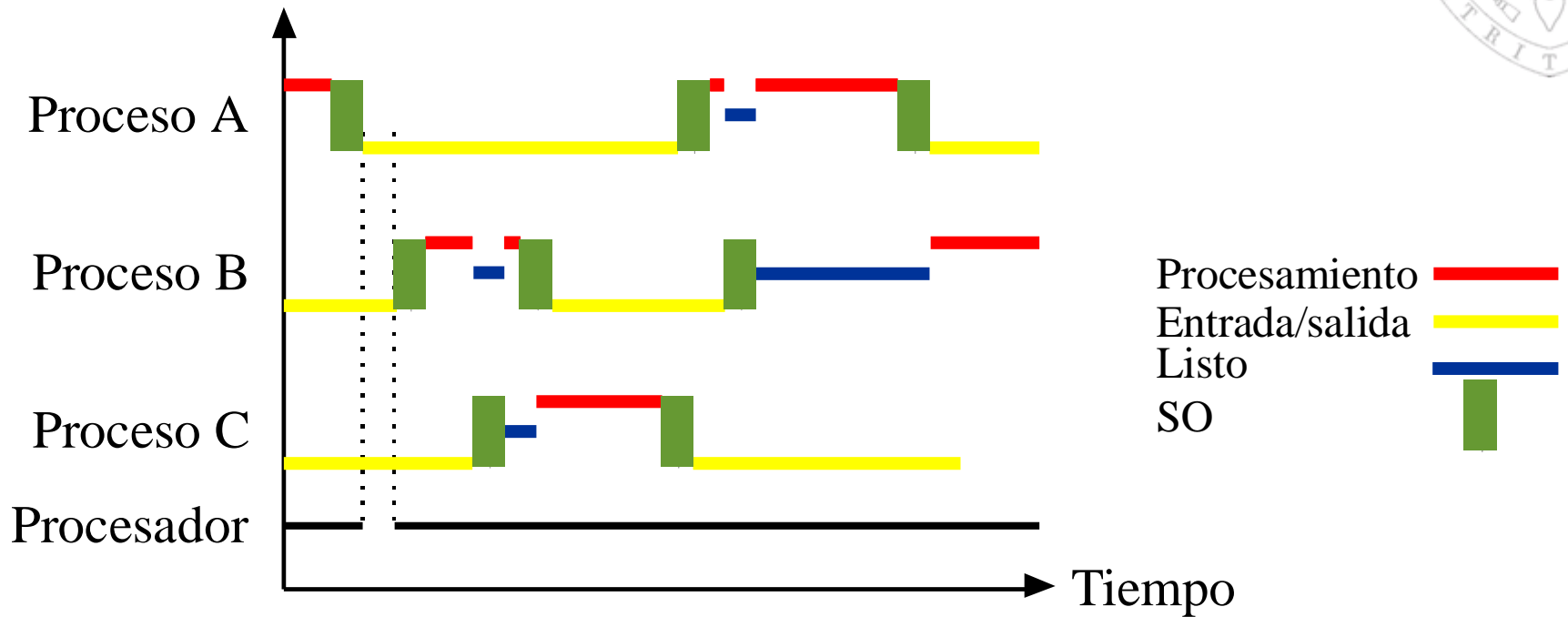
- Procesos
- Multitarea
- Información del proceso
- Formación y estados de un proceso
- Señales
- Hilos o *threads*

Base de la multitarea

- Paralelismo real entre E/S y CPU (DMA)
- Alternancia en los procesos de fases de E/S y de procesamiento
- La memoria almacena varios procesos



Ejecución en un sistema multitarea



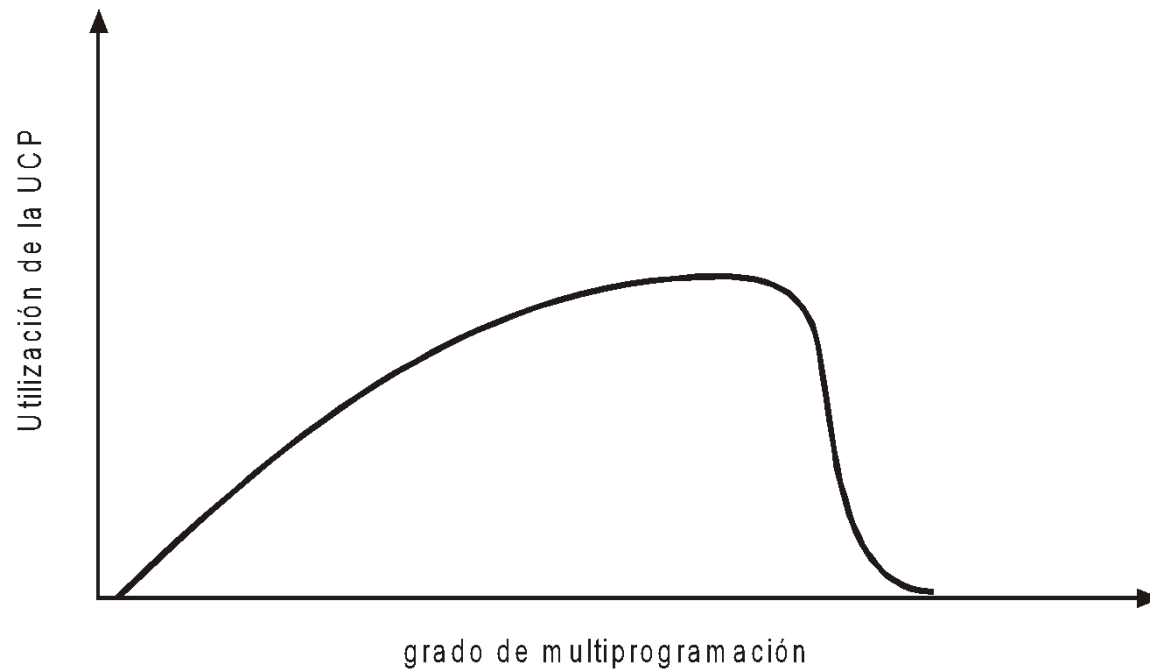
- Proceso nulo o *idle*

Ventajas de la multitarea

- Facilita la programación, dividiendo los programas en procesos (modularidad)
- Permite el servicio interactivo simultáneo de varios usuarios de forma eficiente
- Aprovecha los tiempos que los procesos pasan esperando a que se completen sus operaciones de E/S
- Aumenta el uso de la CPU

Grado de multiprogramación

- Grado de multiprogramación: nº de procesos activos
- Para sistemas con memoria virtual:



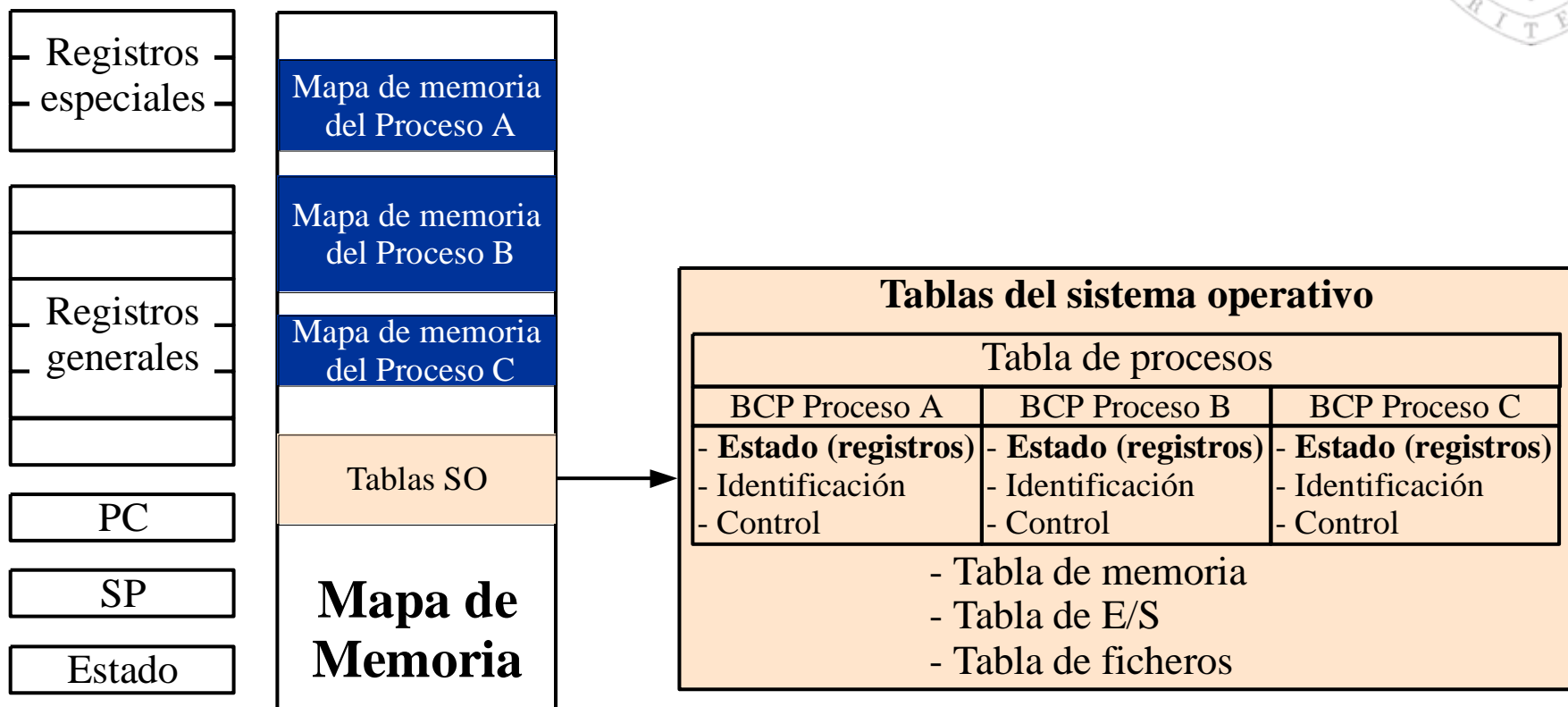
Contenido

- Procesos
- Multitarea
- Información del proceso
- Formación y estados de un proceso
- Señales
- Hilos o *threads*

Información de un proceso

- **Estado del procesador:** contenido de los registros del modelo de programación
- **Imagen de memoria:** contenido de los segmentos de memoria en los que reside el código y los datos del proceso
- **Bloque de control del proceso (BCP) o Descriptor de proceso**
 - Estado actual del proceso
 - Estado del procesador
 - Actualizado cuando proceso no se está ejecutando en la CPU
 - Identificadores *pid*, *uid*, etc.
 - Prioridad
 - Segmentos de memoria (espacio de direcciones)
 - Ficheros abiertos
 - Temporizadores
 - Señales
 - Semáforos
 - Puertos

Información de un proceso (II)



Estado del procesador

- Está formado por el contenido de todos sus registros:
 - Registros generales
 - Contador de programa
 - Puntero de pila
 - Registro de estado
 - Registros especiales
- Cuando un proceso está ejecutando su estado reside en los registros del computador.
- Cuando un proceso no ejecuta su estado reside en el BCP.

Imagen de memoria

- La imagen de memoria está formada por el conjunto de regiones de memoria que un proceso está autorizado a utilizar
- Si un proceso genera una dirección que esta fuera del espacio de direcciones el HW genera una excepción que el SO captura
- La imagen de memoria, dependiendo del computador, puede estar referida a memoria virtual o memoria física

Información del BCP

- Información de identificación:
 - PID del proceso, PID del padre (PPID)
 - ID de usuario y grupo reales (uid/gid reales)
 - ID de usuario y grupo efectivos (uid/gid efectivos)
- Estado del procesador
- Información de control del proceso:
 - Información de planificación y estado
 - Descripción de los segmentos de memoria del proceso
 - Recursos asignados (ficheros abiertos, ...)
 - Recursos de comunicación entre procesos
 - Punteros para estructurar los procesos en listas o colas

Información del BCP II

- Información fuera del BCP
 - Conveniente por implementación (la consideramos del BCP)
 - Para compartirla
- La tabla de páginas se pone fuera
 - Describe la imagen de memoria del proceso
 - Tamaño variable
 - El BCP contiene el puntero a la tabla de páginas
 - La compartición de memoria requiere que sea externa al BCP
- Punteros de posición de los ficheros
 - Si se añaden a la tabla de ficheros abiertos (en el BCP) no se pueden compartir
 - Si se asocian al nodo-i se comparte siempre
 - Se ponen en una estructura común a los procesos y se asigna uno nuevo en cada servicio OPEN

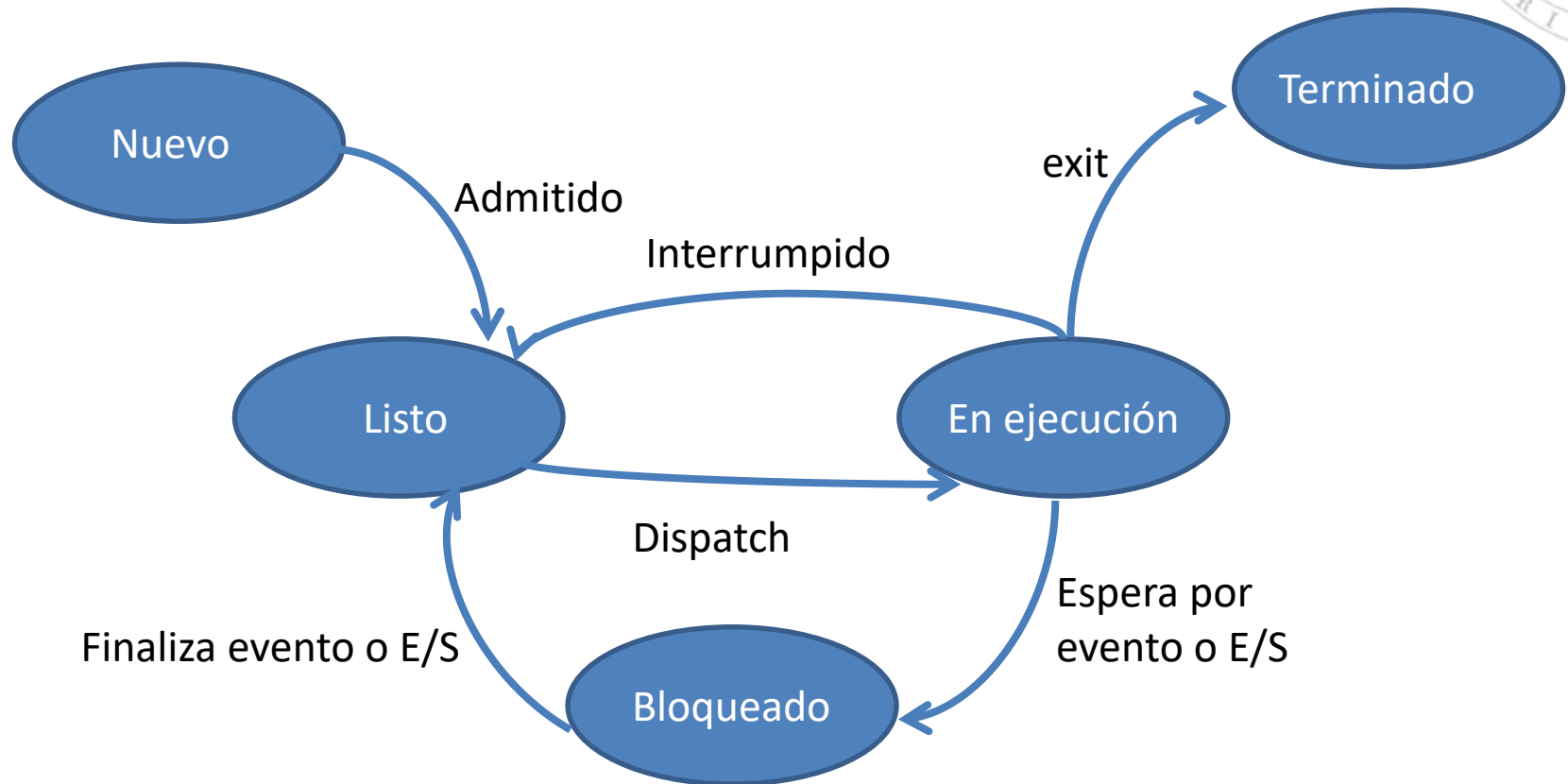
Tablas del sistema operativo

- **Tabla de procesos** (tabla de BCP)
- **Tabla de memoria:** información sobre el uso de la memoria.
- **Tabla de E/S:** guarda información asociada a los periféricos y a las operaciones de E/S
- **Tablas de fichero:** información sobre los ficheros abiertos.
- La información asociada a cada proceso en el BCP
- La decisión de incluir o no una información en el BCP se toma según dos criterios:
 - Eficiencia
 - Compartir información

Contenido

- Procesos
- Multitarea
- Información del proceso
- Formación y estados de un proceso
- Señales
- Hilos o *threads*

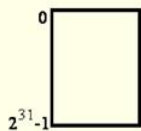
Estados del proceso



Cambio de modo del procesador

- Cuando se produce una interrupción o excepción mientras un proceso se ejecuta en modo usuario el procesador cambia de modo de ejecución (modo kernel)
- Al producirse la interrupción:
 - Se pasa a ejecutar la rutina de tratamiento de interrupción (RTI), bajo control del SO
 - Se salva el valor de los registros (estado del procesador) en la pila del kernel
- Al finalizar la RTI, si el proceso actual sigue “en ejecución”:
 - Restaura los registros del procesador
 - Termina con una instrucción RETI (retorno de interrupción)
 - Restituye el registro de estado (bit de nivel de ejecución)
 - Restituye el contador de programa (para el nuevo proceso).

Modo usuario y modo kernel

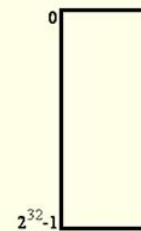


Mapa de memoria

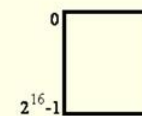


Juego de Instrucciones

Modo usuario



Mapa de memoria



Mapa de E/S



Juego de Instrucciones

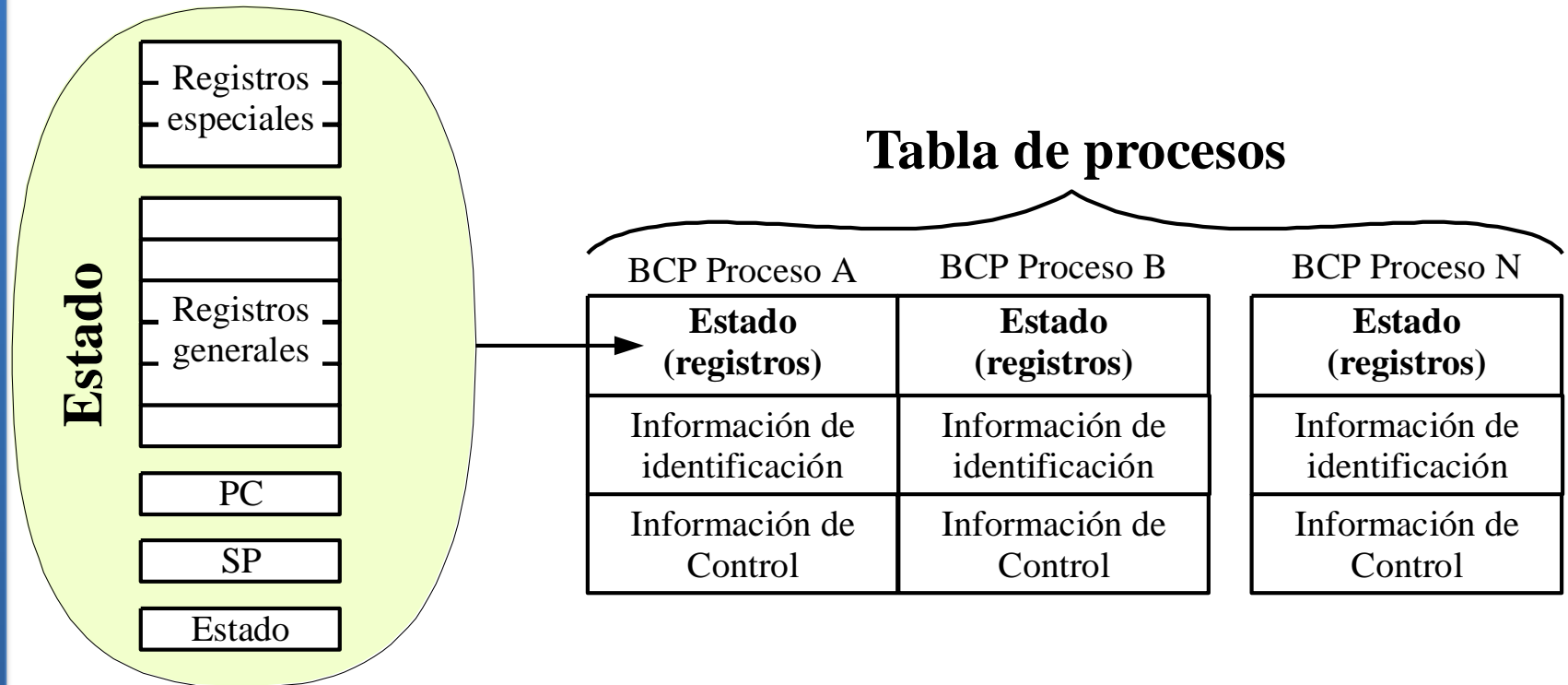
Modo núcleo o modo kernel

Cambio de contexto

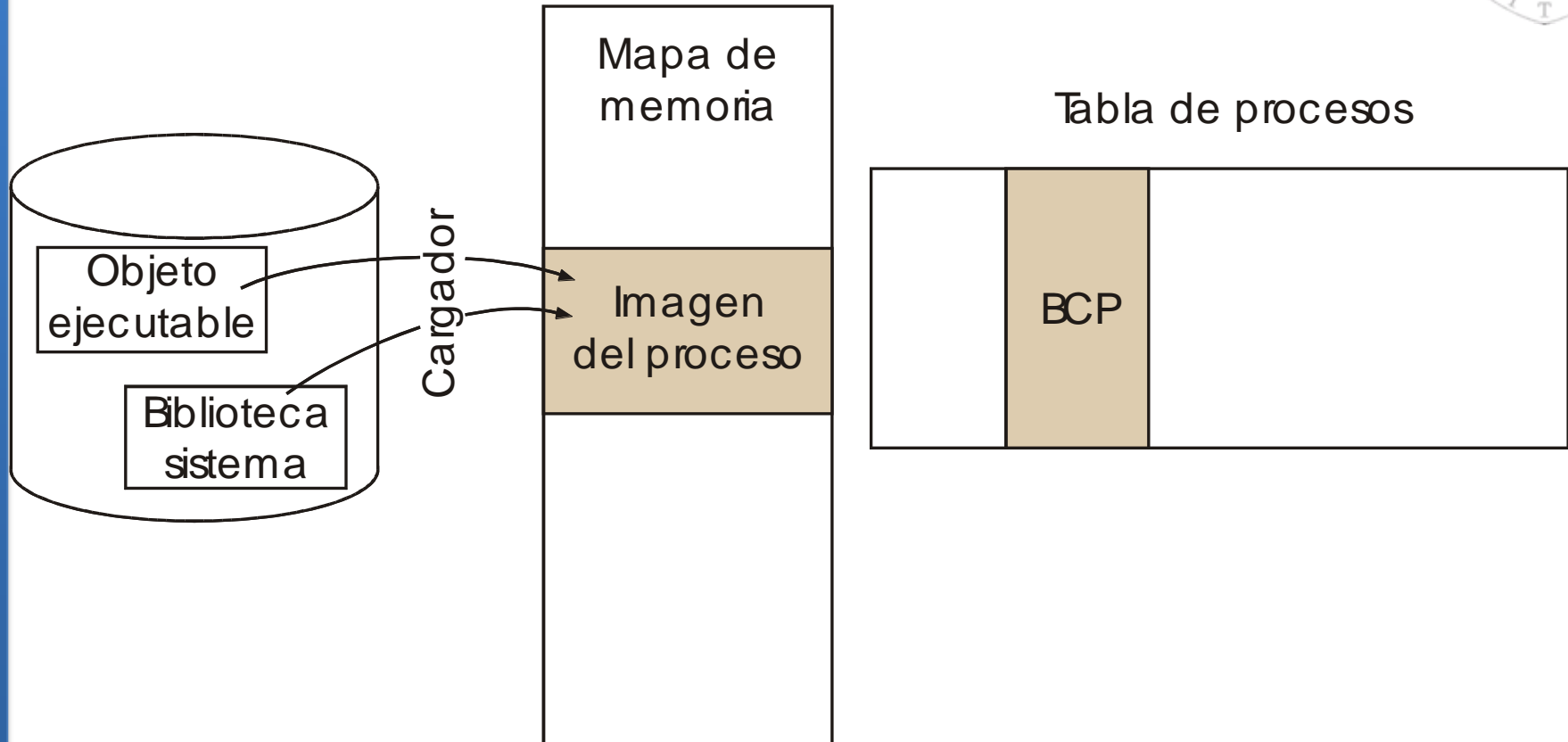
- El **cambio de contexto** es el conjunto de acciones que realiza el SO para cambiar el proceso que está actualmente en ejecución en una CPU
- Acciones (simplificación):
 1. Salvar el contexto del proceso saliente (registros del modelo de programación) en su BCP
 2. Cambiar el estado del proceso saliente (En ejecución -> Otro Estado)
 3. Intercambio de los espacios de direcciones
 - Segmentos o regiones de memoria que puede usar un proceso
 - En x86. GDT (Global descriptor Table)
 - En algunas arquitecturas → Invalidación de entradas de la TLB
 4. Cambiar el estado del proceso entrante, (Listo -> En ejecución)
Restaurar su contexto (BCP -> registros) y volver a modo usuario
- Puede llegar a ser una operación bastante costosa
- *El cambio de modo de ejecución del procesador no siempre desencadena un cambio de contexto*

Cambio de contexto

- Se salva el estado:



Formación de un proceso

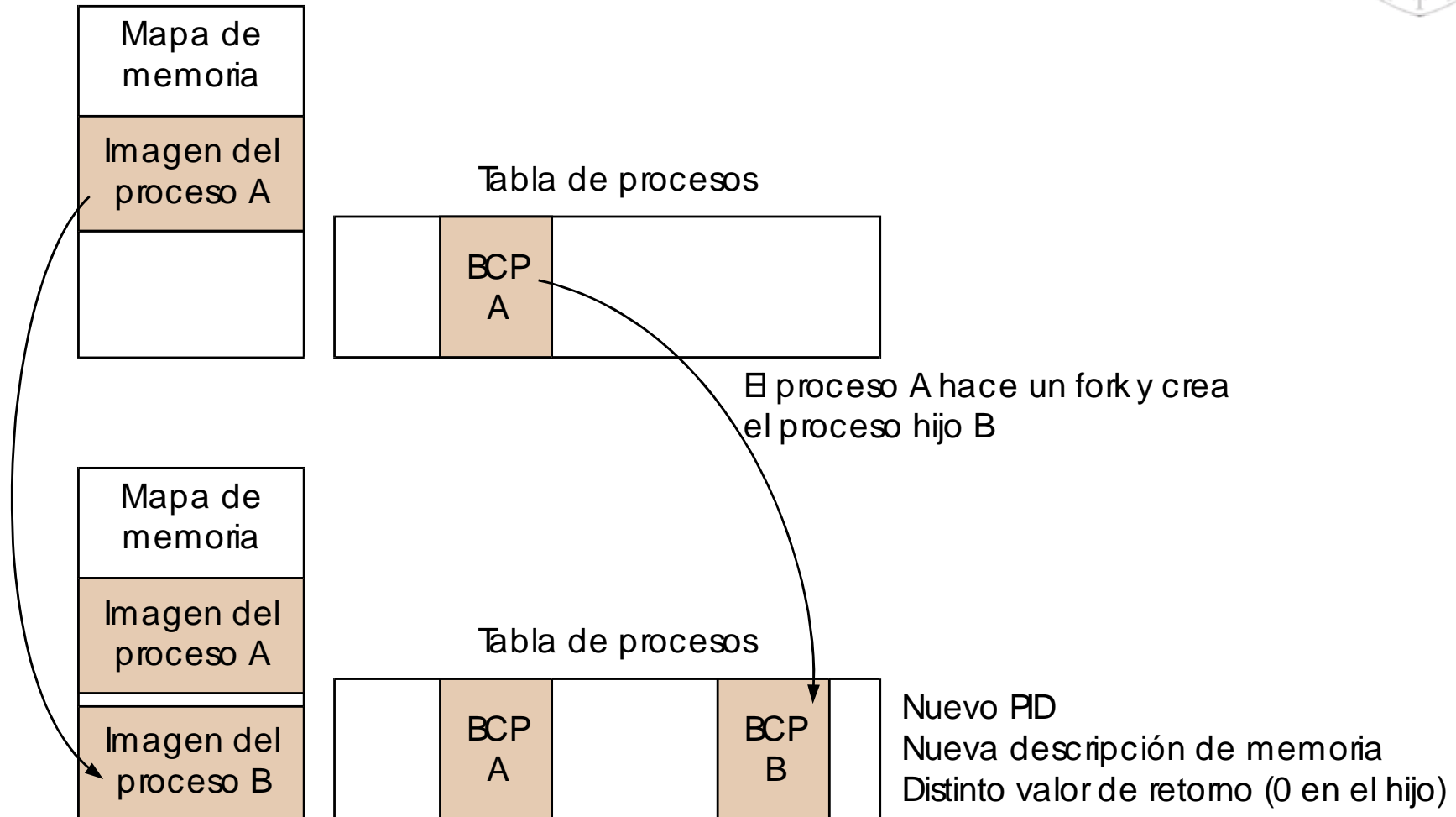


Proceso vs. Ejecutable

- Si tenemos nuestro fichero binario ejecutable *X*, lo ejecutamos y sin esperar a que termine lo volvemos a ejecutar....
 - ¿Tendré uno o dos procesos?
 - Si tengo dos, ¿comparten todas las zonas de memoria?
 - Si uno abre un fichero, ¿el otro ya lo tiene abierto?

Servicios POSIX: fork

- Crea un proceso clonando al padre



fork. Crea un proceso



■ Servicio:

`pid_t fork(void);`

■ Devuelve:

- El identificador de proceso hijo al proceso padre y 0 al hijo
- -1 el caso de error

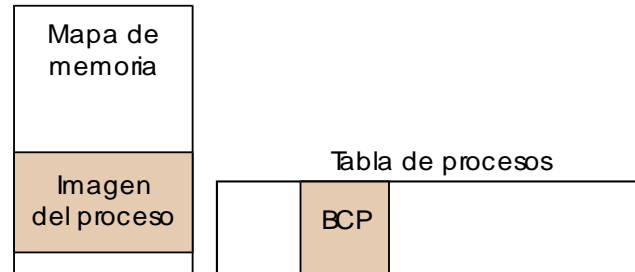
■ Descripción:

- Crea un proceso hijo que ejecuta el mismo programa que el padre
- Hereda los ficheros abiertos (se copian los descriptores).
- Las alarmas pendientes se desactivan.
- El proceso hijo sólo tiene un hilo.

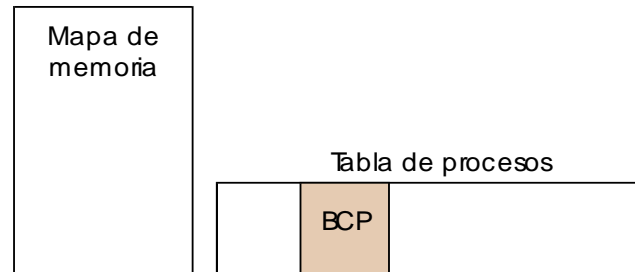
```
void main()
{
    pid_t pid;
    ...
    pid = fork();
    if (pid == 0) {
        /* proceso hijo */
        ...
    }
    else if (pid > 0) {
        /* proceso padre */
        ...
    }
    else {
        /* error al crear */
        ...
    }
    ...
}
```

Servicios POSIX: exec

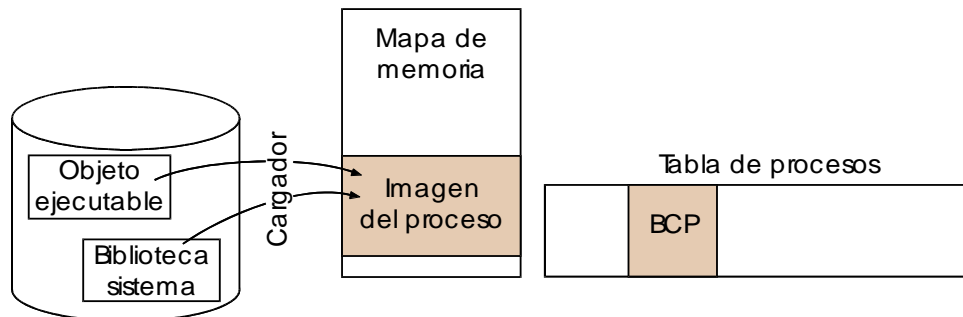
■ Cambia el programa de un proceso



El proceso hace un exec



Se borra la imagen de memoria
Se borra la descripción de la memoria y registros
Se conserva el PID



Se carga la nueva imagen
Se pone PC en dirección de arranque
Se conservan los fd

exec. Cambio del programa de un proceso

■ Servicios:

```
int execl (const char *path, const char *arg, ...)  
int execelp(const char *file, const char *arg, ...)  
int execvp(const char *file, char *const argv[])
```

■ 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
- Las señales con la acción por defecto seguirán por defecto, las señales con manejador tomarán la acción por defecto.

exit. Terminación de un proceso

- Servicios:

```
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

wait. Espera la terminación de un proceso hijo

■ Servicios:

```
#include <sys/types.h>
pid_t wait(int *status);
```

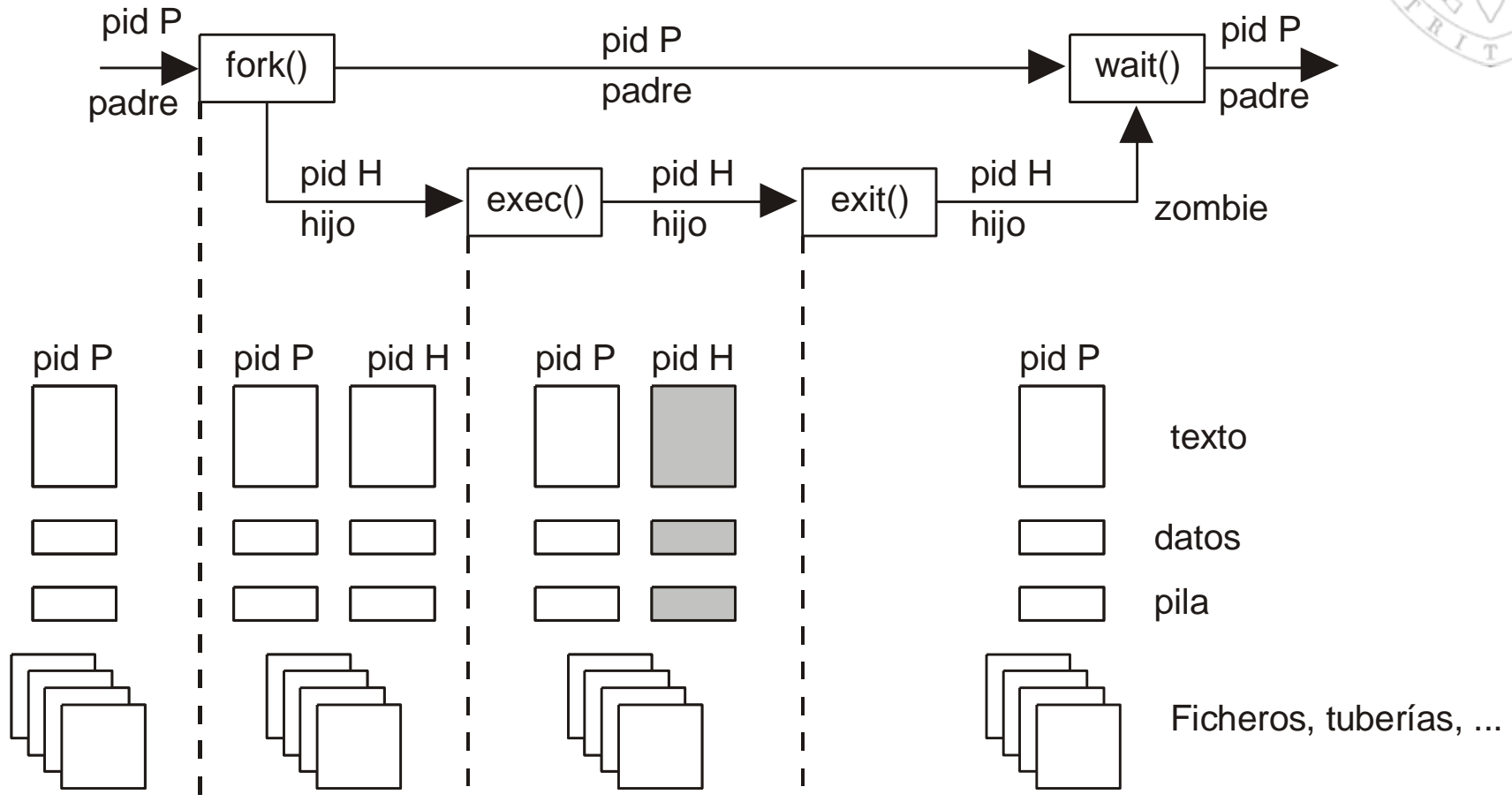
■ Argumentos:

- Devuelve el código de terminación del proceso hijo.

■ Descripción:

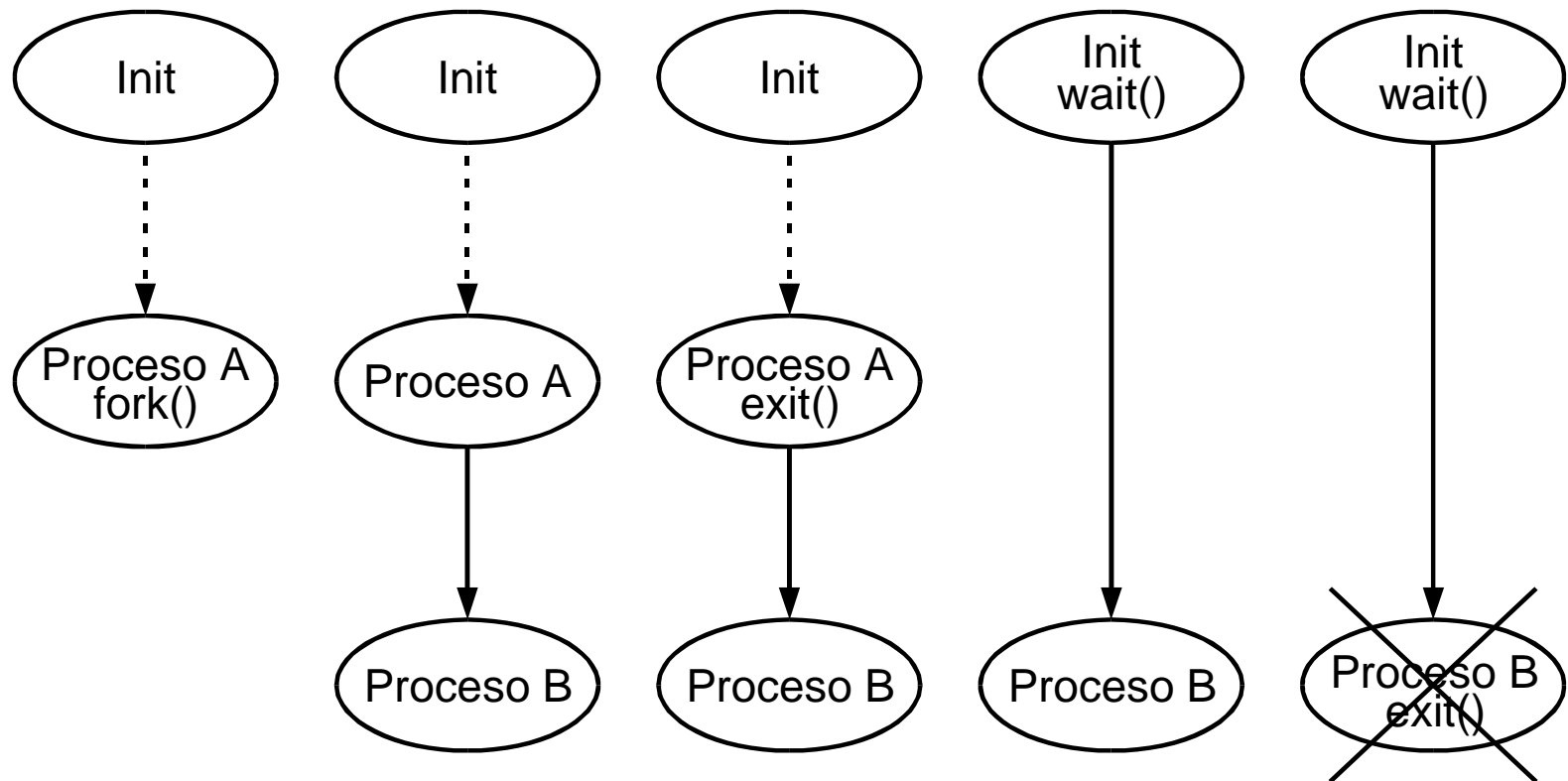
- Devuelve el identificador del proceso hijo o -1 en caso de error.
- Permite a un proceso padre esperar hasta que termine un proceso hijo. Devuelve el identificador del proceso hijo y el estado de terminación del mismo.

Uso normal de los servicios



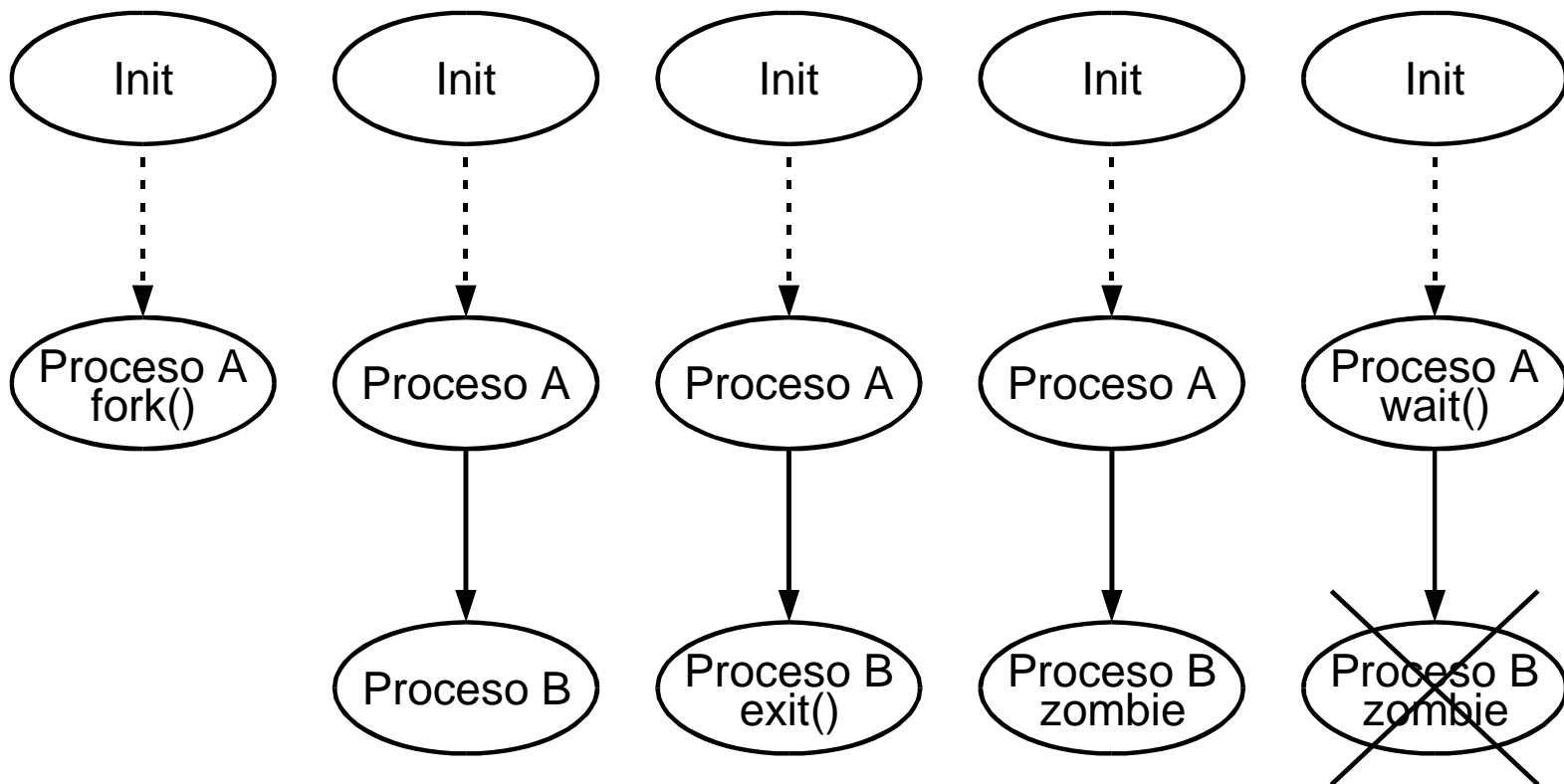
Evolución de procesos I

- El padre muere: *INIT* acepta los hijos



Evolución de procesos II

- *Zombie*: el hijo muere y el padre no hace *wait*



Programa de ejemplo

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

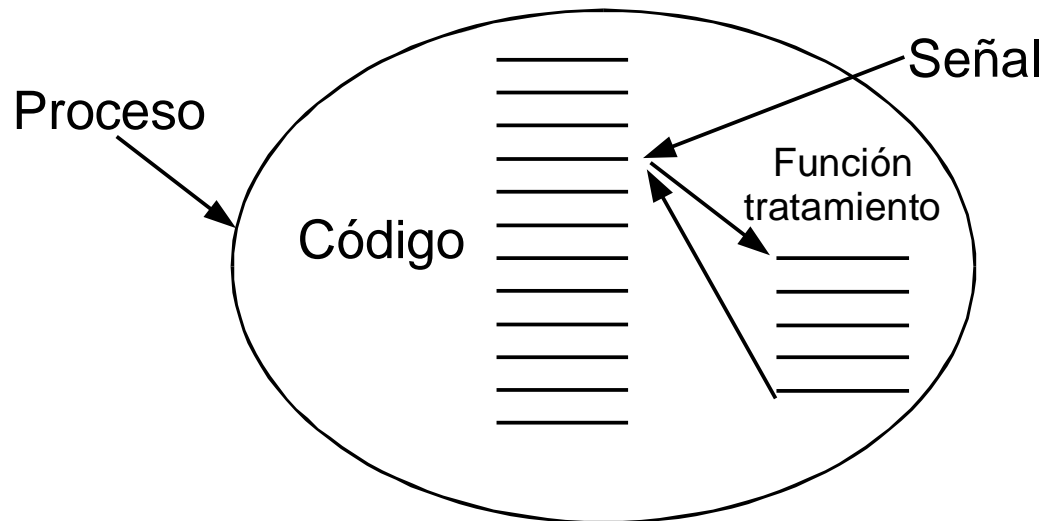
Contenido

- Procesos
- Multitarea
- Información del proceso
- Formación y estados de un proceso
- Señales
- Hilos o *threads*

Señales



- Las señales son interrupciones al proceso
- Envío o generación:
 - Proceso \rightarrow Proceso (con mismo *uid*) con *kill*
 - SO \rightarrow Proceso



Señales II

- Hay muchos tipos de señales, según su origen
 - SIGILL instrucción ilegal
 - SIGALRM vence el temporizador
 - SIGKILL mata al proceso
- El SO las transmite al proceso
 - El proceso debe estar preparado para recibirla
 - Especificando un manejador de señal con *sigaction*
 - Enmascarando la señal con *sigprogmask*
 - Si no está preparado → acción por defecto
 - El proceso, en general, muere
 - Hay algunas señales que se ignoran o tienen otro efecto
- El servicio *pause* para el proceso hasta que recibe una señal

Señales: servicios POSIX

- **int kill(pid_t pid, int sig)**
 - Envía al proceso *pid* la señal *sig*
- **int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);**
 - Permite especificar la acción a realizar *act* como tratamiento de la señal *signum*. Permite almacenar la acción previa en *oldact*
- **int pause(void)**
 - Bloquea al proceso hasta la recepción de una señal.
- Ejemplo:

\$ kill -SIGINT pid

Envía al proceso con identificador *pid* la señal de interrupción (Ctrl C).

Si el proceso tiene un manejador para esa señal ejecutará el código del manejador.

En caso contrario, el proceso muere.

Contenido

- Procesos
- Multitarea
- Información del proceso
- Formación y estados de un proceso
- Señales
- Hilos o *threads*

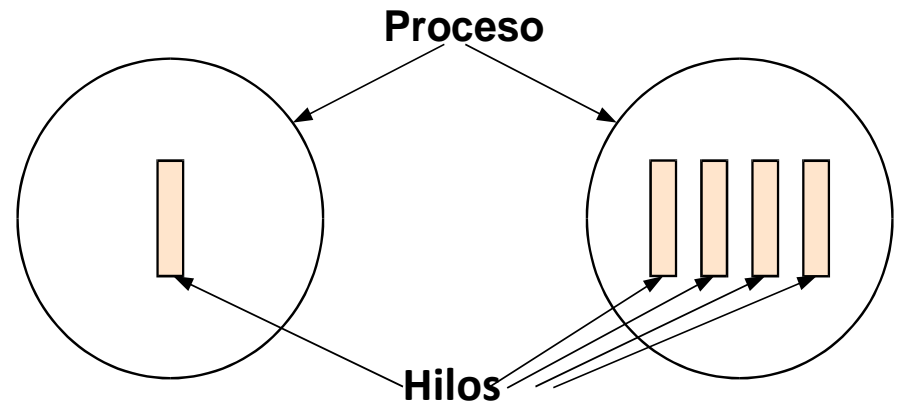
Hilos o threads

■ Por Hilo

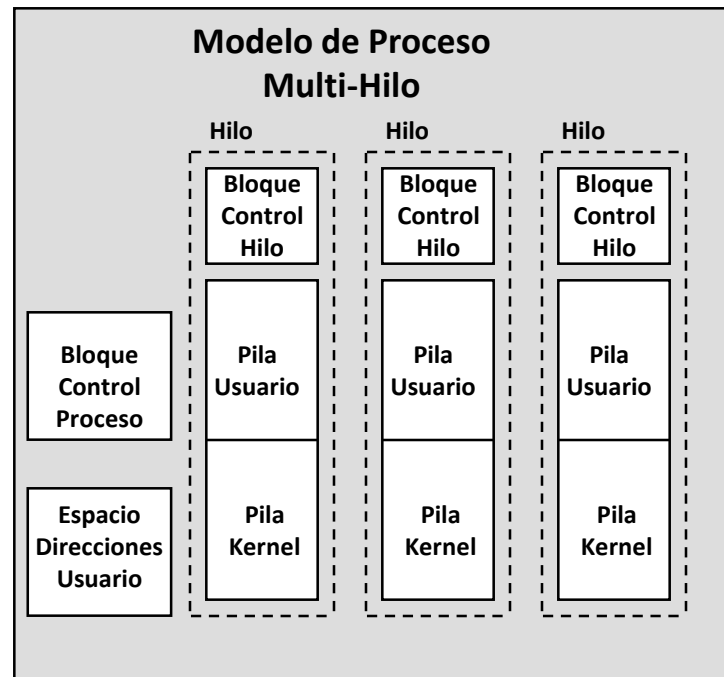
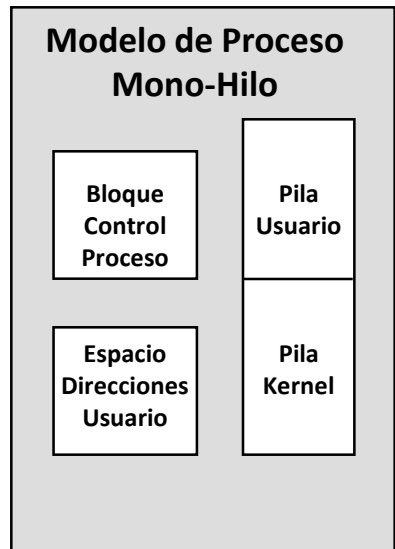
- Contador de programa, Registros
- Pila
- Estado (ejecutando, listo o bloqueado)
- Bloque de control de *thread*

■ Por proceso

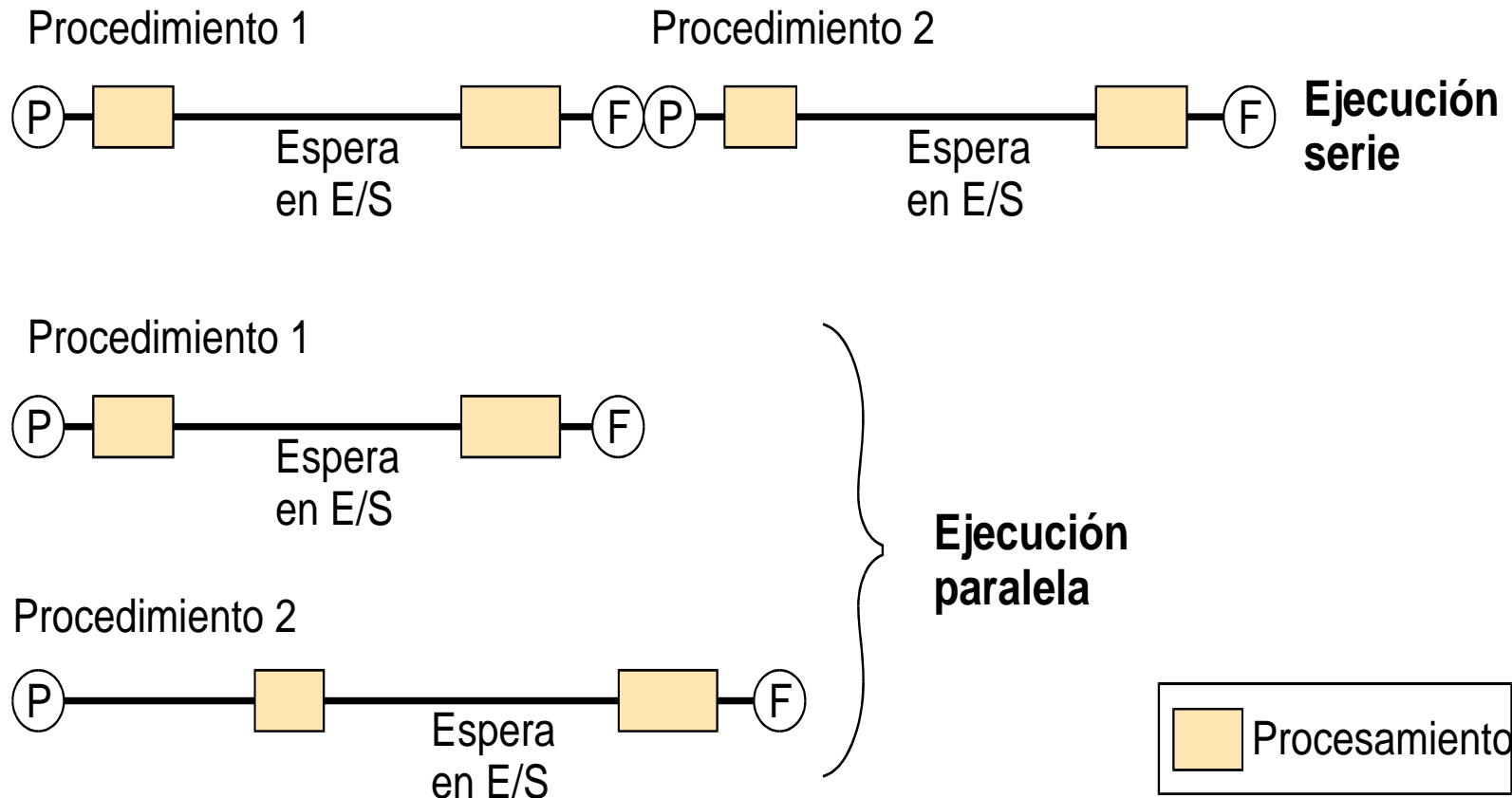
- Espacio de direcciones de memoria
- Variables globales
- Ficheros abiertos
- Procesos hijos
- Temporizadores
- Señales y semáforos



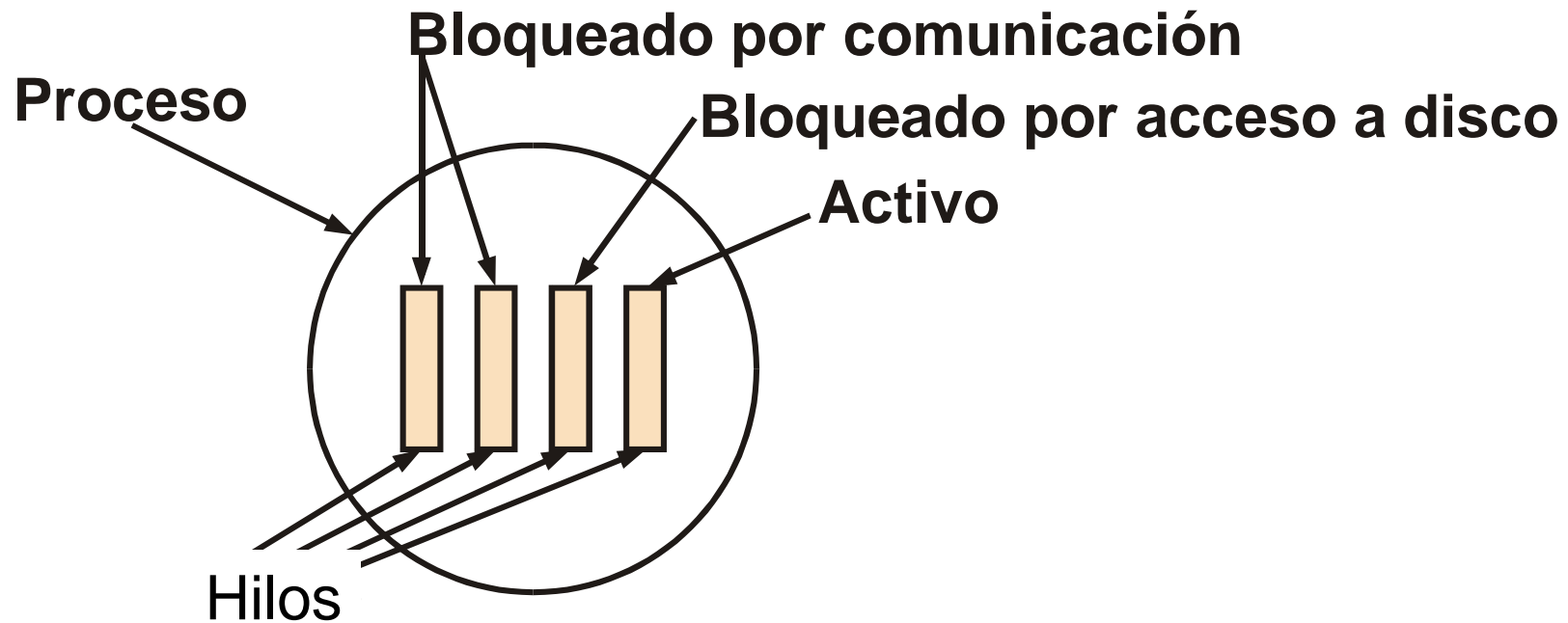
Mono-hilo vs Multi-hilo



Paralelización utilizando hilos



Estados de un hilo



Ventajas threads vs. procesos

- Tiempo de procesador para operaciones relacionadas con creación, destrucción, planificación y sincronización:
 - 10 hilos vs 100 proceso.
- El cambio de contexto entre hilos (de kernel) de un mismo proceso es menos costoso → No es necesario cambiar el espacio de direcciones “activo” de usuario
- Permiten compartir memoria entre ellos de forma fácil y eficiente
 - ¡¡Todos tienen el mismo espacio de direcciones!!

Diseño con hilos

- Permite separación de tareas
- Paralelismo
 - Aumenta la velocidad de ejecución del trabajo
- Programación concurrente (memoria compartida)
 - Variables o estructuras de datos compartidas
 - Funciones reentrantes
 - Imaginar otra llamada al mismo código
 - Mecanismos de sincronización entre hilos (mutex, semáforos,...)
 - Variables globales
 - Simplicidad vs exclusión en el acceso

Alternativas al diseño multihilo

- Proceso con un solo hilo
 - No hay paralelismo
 - Llamadas al sistema bloqueantes
 - Paralelismo gestionado por el programador
 - Llamadas al sistema no bloqueantes
- Múltiples procesos convencionales cooperando
 - Permite paralelismo
 - No comparten variables
 - Mayor sobrecarga de ejecución

POSIX para la gestión de hilos

- **int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*func)(void *), void *arg)**
 - Crea un hilo 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 hilo hasta que termina el hilo con identificador "thid".
 - Devuelve el estado de terminación del hilo.
- **int pthread_exit(void *value)**
 - Permite a un hilo finalizar su ejecución, indicando el estado de terminación del mismo.
- **pthread_t pthread_self(void)**
 - Devuelve el identificador del thread que ejecuta la llamada.

POSIX para la gestión hilos (II)

- **int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate)**
 - Establece el estado de terminación de un hilo.
 - Si *detachstate* = PTHREAD_CREATE_DETACHED el hilo liberará sus recursos cuando finalice su ejecución.
 - Si *detachstate* = PTHREAD_CREATE_JOINABLE no se liberarán los recursos, es necesario utilizar pthread_join().

Programa de ejemplo

```
#include <stdio.h>
#include <pthread.h>
#define MAX_THREADS 10
void* func(void* arg) {
    printf("Thread %d \n", pthread_self());
    pthread_exit(0);
}
int main(void) {
    int j;
    pthread_attr_t attr;
    pthread_t thid[MAX_THREADS];
    pthread_attr_init(&attr);
    for(j = 0; j < MAX_THREADS; j++)
        pthread_create(&thid[j], &attr, func, NULL);
    for(j = 0; j < MAX_THREADS; j++)
        pthread_join(thid[j], NULL);
    return 0;
}
```