UT 1. Programación multiproceso

1. Conceptos básicos.
2. Programación concurrente.
3. Funcionamiento básico del sistema operativo.
4. Procesos.
5. Gestión de procesos.

1. Conceptos básicos

Para poder empezar a entender cómo se ejecutan varios programas a la vez, es imprescindible adquirir ciertos conceptos.

- **Programa**: se puede considerar un programa a toda la información (tanto código como datos) almacenada en disco de una aplicación que resuelve una necesidad concreta para los usuarios.
- **Proceso**: cuando un programa se ejecuta, podremos decir de manera muy simplificada que es un proceso. En definitiva, puede definirse "proceso" como un programa en ejecución. Este concepto no se refiere únicamente al código y a los datos, sino que incluye todo lo necesario para su ejecución. Esto incluye tres cosas:
    - Un contador del programa: algo que indique por dónde se está ejecutando.
    - Una imagen de memoria: es el espacio de memoria que el proceso está utilizando.
    - Estado del procesador: se define como el valor de los registros del procesador sobre los cuales se está ejecutando.

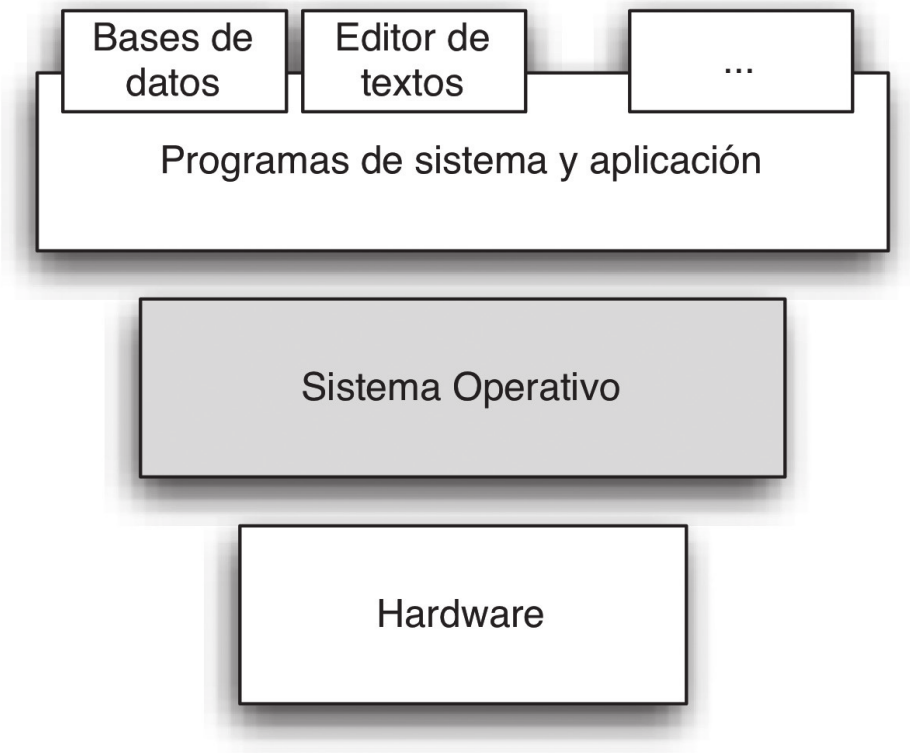
cuestión

¿Dónde se almacena la información mientras el programa se está ejecutando en la CPU?

Es importante destacar que los procesos **son entidades independientes, aunque ejecuten el mismo programa.** De tal forma, pueden coexistir dos procesos que ejecuten el mismo programa, pero con diferentes datos (es decir, con distintas imágenes de memoria) y en distintos momentos de su ejecución (con diferentes contadores de programa).

- **Ejecutable**: un fichero ejecutable contiene la información necesaria para crear un proceso a partir de los datos almacenados de un programa. Es decir, llamaremos "ejecutable" al fichero que permite poner el programa en ejecución como proceso.
- **Sistema operativo**: programa que hace de intermediario entre el usuario y las

aplicaciones que utiliza y el hardware del ordenador.



- **Demonio**: proceso no interactivo que está ejecutándose continuamente en segundo plano, es decir, es un proceso controlado por el sistema sin ninguna intermediación del usuario. Suelen proporcionar un servicio básico para el resto de procesos.

2. Programación Concurrente.

La **computación concurrente** permite la posibilidad de tener en ejecución al mismo tiempo múltiples tareas interactivas. Es decir, permite realizar varias cosas al mismo tiempo, como escuchar música, visualizar la pantalla del ordenador, imprimir documentos, etc. Pensad en todo el tiempo que perderíamos si todas esas tareas se tuvieran que realizar una tras otra. Dichas tareas se pueden ejecutar en:

- Un **único procesador (multiprogramación)**. En este caso, aunque para el usuario parezca que varios procesos se ejecutan al mismo tiempo, si solamente existe un único procesador, solamente un proceso puede estar en un momento determinado en ejecución. Para poder ir cambiando entre los diferentes procesos, el sistema operativo se encarga de cambiar el proceso en ejecución después de un período corto de tiempo (del orden de milisegundos). Esto permite que en un segundo se ejecuten múltiples procesos, creando en el usuario la percepción de que múltiples programas se están ejecutando al mismo tiempo.
  Este concepto se denomina **programación concurrente**. La programación concurrente no mejora el tiempo de ejecución global de los programas ya que se ejecutan intercambiando unos por otros en el procesador. Sin embargo, permite que varios programas parezca que se ejecuten al mismo tiempo.

- **Varios núcleos** en un mismo procesador (**multitarea**). La existencia de varios núcleos o cores en un ordenador es cada vez mayor, apareciendo en Dual Cores, Quad Cores, en muchos de los modelos i3, i5 e i7, etc. Cada núcleo podría estar ejecutando una instrucción diferente al mismo tiempo. El sistema operativo, al igual que para un único procesador, se debe encargar de planificar los trabajos que se ejecutan en cada núcleo y cambiar unos por otros para generar multitarea. En este caso todos los cores comparten la misma memoria por lo que es posible utilizarlos de forma coordinada mediante lo que se conoce por programación paralela.
  La programación paralela permite mejorar el rendimiento de un programa si este se ejecuta de forma paralela en diferentes núcleos ya que permite que se ejecuten varias instrucciones a la vez. Cada ejecución en cada core será una tarea del mismo programa pudiendo cooperar entre sí. El concepto de “tarea” (o “hilo de ejecución”) se explicará en profundidad más adelante. Y, por supuesto, se puede utilizar conjuntamente con la programación concurrente, permitiendo al mismo tiempo multiprogramación. **(programación concurrente + multitarea).**

- **Varios ordenadores distribuidos en red**. Cada uno de los ordenadores tendrá sus propios procesadores y su propia memoria. La gestión de los mismos forma parte de lo que se denomina **programación distribuida.**
  La programación distribuida posibilita la utilización de un gran número de dispositivos (ordenadores) de forma paralela, lo que permite alcanzar elevadas mejoras en el rendimiento de la ejecución de programas distribuidos.

Sin embargo, como cada ordenador posee su propia memoria, imposibilita que los procesos pueden comunicarse compartiendo memoria, teniendo que utilizar otros esquemas de comunicación más complejos y costosos a través de la red que los interconecte.

Vídeo Tipos programación (http://www.youtube.com/watch?v=Svpgi5Hf26g)

3. Funcionamiento básico del sistema operativo.

La parte central que realiza la funcionalidad básica del sistema operativo se denomina **kernel.** Es una parte software pequeña del sistema operativo, si la comparamos con lo necesario para implementar su interfaz (y más hoy en día que es muy visual). A todo lo demás del sistema se le denomina programas del sistema. El kernel es el responsable de gestionar los recursos del ordenador, permitiendo su uso a través de llamadas al sistema.

En general, el **kernel del sistema funciona en base a interrupciones.** Una interrupción es una s**uspensión temporal de la ejecución de un proceso**, para pasar a ejecutar una rutina que trate dicha interrupción. Esta rutina será dependiente del sistema operativo. Es importante destacar que mientras se está atendiendo una interrupción, se deshabilita la llegada de nuevas interrupciones. Cuando finaliza la rutina, se reanuda la ejecución del proceso en el mismo lugar donde se quedó cuando fue interrumpido.

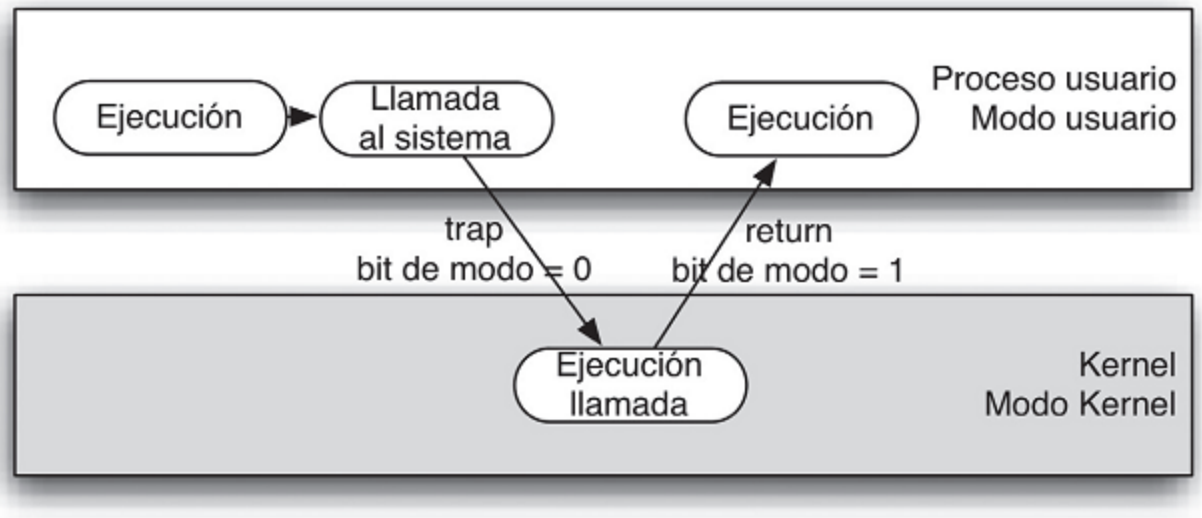
Es decir, el sistema operativo no es un proceso demonio propiamente dicho que proporcione funcionalidad al resto de procesos, sino que él solo se ejecuta respondiendo a interrupciones. Cuando salta una interrupción se transfiere el control a la rutina de tratamiento de la interrupción. Así, las rutinas de tratamiento de interrupción pueden ser vistas como el código propiamente dicho del kernel.

Las **llamadas al sistema** son la interfaz que proporciona el kernel para que los programas de usuario puedan hacer uso de forma segura de determinadas partes del sistema.

Los errores de un programa podrían afectar a otros programas o al propio sistema operativo, por lo que para asegurar su ejecución de la forma correcta, el sistema implementa una interfaz de llamadas para evitar que ciertas instrucciones peligrosas sean ejecutadas directamente por programas de usuario.
El **modo dual** es una característica del hardware que permite al sistema operativo protegerse. El procesador tiene dos modos de funcionamiento indicados mediante un bit:

- Modo usuario (1). Utilizado para la ejecución de programas de usuario.
- Modo kernel(0), también llamado “modo supervisor” o “modo privilegiado”. Las

instrucciones del procesador más delicadas solo se pueden ejecutar si el procesador está en modo kernel.



Cuestiones.

1. Explica con tus propias palabras el funcionamiento de la programación concurrente en un mismo PC.
2. ¿Se puede aplicar multiprogramación en un sistema con varios ordenadores en red ?
3. Explica con tus propias palabras el funcionamiento de la multitarea. ¿Por qué no se introdujo la multitarea en los primeros sistemas informáticos?
4. Explica con tus propias palabras la programación distribuida.
5. ¿Qué es una llamada al sistema?
6. ¿Qué aporta el modo dual?

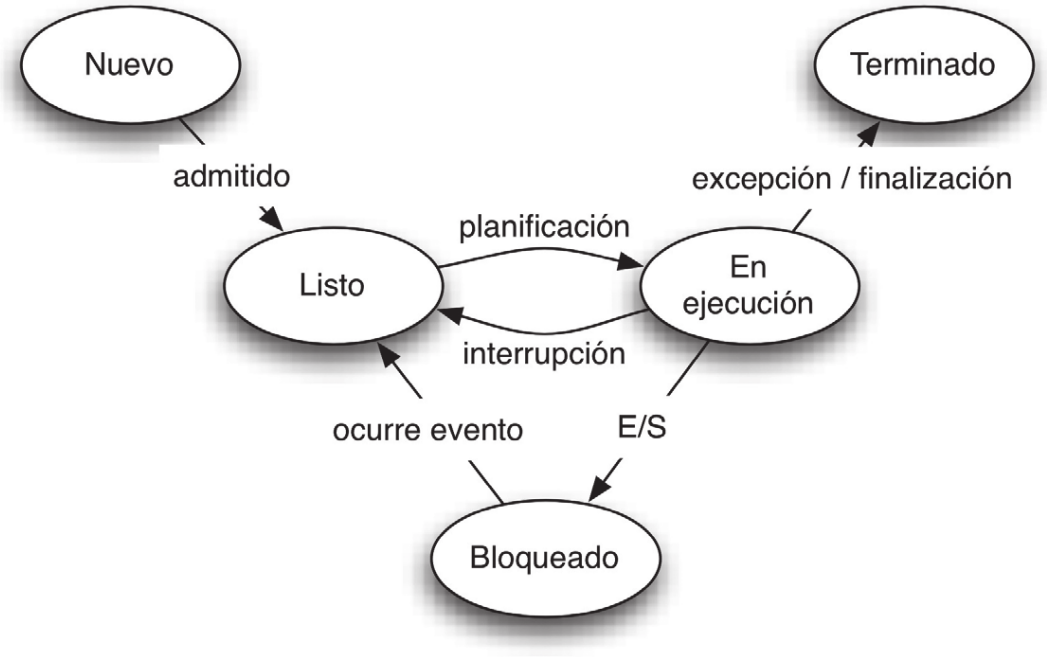
4. Procesos.

El sistema operativo es el encargado de poner en ejecución y gestionar los procesos. Para su correcto funcionamiento, a lo largo de su ciclo de vida, los procesos pueden cambiar de estado. Es decir, a medida que se ejecuta un proceso, dicho proceso pasará por varios estados. El cambio de estado también se producirá por la intervención del sistema operativo.

4.1 Estados de un proceso.

Los estados de un proceso son:

- **Nuevo.** El proceso está siendo creado a partir del fichero ejecutable.
- **Listo**: el proceso no se encuentra en ejecución aunque está preparado para hacerlo. El sistema operativo no le ha asignado todavía un procesador para ejecutarse. El planificador del sistema operativo es el responsable de seleccionar que proceso está en ejecución, por lo que es el que indica cuando el proceso pasa a ejecución.
- **En ejecución:** el proceso se está ejecutando. El sistema operativo utiliza el mecanismo de interrupciones para controlar su ejecución. Si el proceso necesita un recurso, incluyendo la realización de operaciones de entrada salida (E/S), llamará a la llamada del sistema correspondiente. Si un proceso en ejecución se ejecuta durante el tiempo máximo permitido por la política del sistema, salta un temporizador que lanza una interrupción. En este último caso, si el sistema es de tiempo compartido, lo para y lo pasa al estado Listo, seleccionando otro proceso para que continúe su ejecución.
- **Bloqueado:** el proceso está bloqueado esperando que ocurra algún suceso (esperando por una operación de E/S, bloqueado para sincronizarse con otros procesos, etc.). Cuando ocurre el evento que lo desbloquea, el proceso no pasa directamente a ejecución sino que tiene que ser planificado de nuevo por el sistema.
- **Terminado:** el proceso ha finalizado su ejecución y libera su imagen de memoria. Para terminar un proceso, el mismo debe llamar al sistema para indicárselo o puede ser el propio sistema el que finalice el proceso mediante una excepción (una interrupción especial).



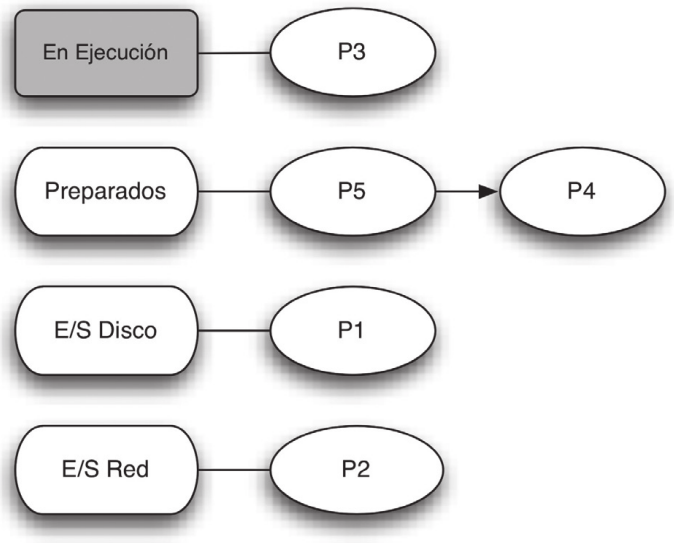
4.2 Colas de procesos.

Uno de los objetivos del sistema operativo es la multiprogramación, es decir, admitir varios procesos en memoria para maximizar el uso del procesador. Esto funciona ya que los procesos se irán intercambiando el uso del procesador para su ejecución de forma concurrente. Para ello, el sistema operativo organiza los procesos en varias colas, migrándolos de unas colas a otras:

- Una **cola de procesos** que contiene todos los procesos del sistema.
- Una **cola de procesos preparados** que contienen todos los procesos listos esperando para ejecutarse.
- **Varias colas de dispositivo** que contiene los procesos que están a la espera de alguna operación de E/S.

Cuestiones.

7. Supongamos que varios procesos (1, 2, 3, 4 y 5) se están ejecutando concurrentemente en un ordenador. El proceso 1 se está ejecutando cuando realiza una petición de E/S al disco. En ese momento, el proceso 2 pasa a ejecución, haciendo otra operación de E/S, en este caso por la tarjeta de red. El proceso 3 pasa a ejecutarse. En ese momento, las colas de procesos serán las siguientes:



En ese momento supongamos que el proceso 3 realiza una petición de E/S teniendo que esperar por el disco duro. Dibuja el estado de las colas correspondientes e identifica cuál sería el proceso que se ejecutará utilizando la filosofía FIFO para sacar los procesos de las colas.

8. ¿Qué es una pila? Explica su funcionamiento.
9. ¿Qué diferencias encontramos entre una pila y una cola?

4.3 Planificación de procesos

Para gestionar las colas de procesos, es necesario un planificador de procesos. El planificador es el encargado de seleccionar los movimientos de procesos entre las diferentes colas. Existen dos tipos de planificación:

- **A corto plazo**: selecciona qué proceso de la cola de procesos preparados pasará a ejecución. Se invoca muy frecuentemente.
    - Planificación sin desalojo. Únicamente se cambia el proceso en ejecución si dicho proceso se bloquea o termina.
    - Planificación apropiativa. Se ejecuta el proceso con mayor prioridad.

    - Tiempo compartido: cada cierto tiempo (llamado cuanto) se desaloja el proceso que estaba en ejecución y se selecciona otro proceso para ejecutarse. En este caso, todas las prioridades de los hilos se consideran iguales.
- **A largo plazo:** selecciona qué procesos nuevos deben pasar a la cola de procesos reparados. Se invoca con poca frecuencia, por lo que puede tomarse más tiempo en tomar la decisión. Controla el grado de multiprogramación (número de procesos en memoria)

4.4 Cambios de contexto.

Cuando el procesador pasa a ejecutar otro proceso, lo cual ocurre muy frecuentemente, el sistema operativo debe guardar el contexto del proceso actual y restaurar el contexto del proceso que el planificador a corto plazo ha elegido ejecutar. La salvaguarda de la información del proceso en ejecución se produce cuando hay una interrupción.

Se conoce como contexto a:

- Estado del proceso.
- Estado del procesador: valores de los diferentes registros del procesador.
- Información de gestión de memoria: espacio de memoria reservada para el proceso.

El cambio de contexto es tiempo perdido, ya que el procesador no hace trabajo útil durante ese tiempo. Únicamente es tiempo necesario para permitir la multiprogramación y su duración depende de la arquitectura en concreto del procesador

5. Gestión de procesos.

5.1 Árbol de procesos

El sistema operativo es el encargado de crear y gestionar los nuevos procesos siguiendo las directrices del usuario.Así, cuando un usuario quiere abrir un programa, el sistema operativo es el responsable de crear y poner en ejecución el proceso correspondiente que lo ejecutará. Aunque el responsable del proceso de creación es el sistema operativo, ya que es el único que puede acceder a los recursos del ordenador, el nuevo proceso se crea siempre por petición de otro proceso. La puesta en ejecución de un nuevo proceso se produce debido a que hay un proceso en concreto que está pidiendo su creación en su nombre o en nombre del usuario.

En este sentido, cualquier proceso en ejecución siempre depende del proceso que lo creó, estableciéndose un vínculo entre ambos. A su vez, el nuevo proceso puede crear nuevos procesos, formándose lo que se denomina un árbol de procesos. Cuando se arranca el ordenador, y se carga en memoria el kernel del sistema a partir de su imagen en disco, se crea el proceso inicial del sistema. A partir de este proceso, se crea el resto de procesos de forma jerárquica, estableciendo padres, hijos, abuelos, etc.

Para identificar a los procesos, los sistemas operativos suelen utilizar un **identificador de proceso** (process identifier [PID]) unívoco para cada proceso. La utilización del PID es básica a la hora de gestionar procesos, ya que es la forma que tiene el sistema de referirse a los procesos que gestiona.

Cuestiones.

10. En Linux mediante las siguiente orden podemos obtener el árbol de procesos de nuestro sistema. > pstree -pl.
    ¿Averigua cuál es el proceso del que cuelgan el resto?
11. En Windows Utiliza el Administrador de tareas para obtener los procesos del sistema.

5.2. Operaciones básicas con procesos.

Siguiendo el vínculo entre procesos establecido en el árbol de procesos, el proceso creador se denomina padre y el proceso creado se denomina hijo. A su vez, los hijos pueden crear nuevos hijos. A la operación de creación de un nuevo proceso la denominaremos create.

Cuando se crea un nuevo proceso tenemos que saber que padre e hijo se ejecutan concurrentemente. Ambos procesos comparten la CPU y se irán intercambiando siguiendo la política de planificación del sistema operativo para proporcionar multiprogramación. Si el proceso padre necesita esperar hasta que el hijo termine su ejecución para poder continuar la suya con los resultados obtenidos por el hijo, puede hacerlo mediante la operación wait.

Como se ha visto al inicio del capítulo, los procesos son independientes y tienen su propio espacio de memoria asignado, llamado **imagen de memoria.** Padres e hijos son procesos y, aunque tengan un vínculo especial, mantienen esta restricción. Ambos usan espacios de memoria independientes. En general, parece que el hijo ejecuta un programa diferente al padre, pero en algunos sistemas operativos esto no tiene porque ser así. Por ejemplo, mientras que en sistemas tipo Windows existe una función createProcess() que crea un nuevo proceso a partir de un programa distinto al que está en ejecución, en sistemas tipo UNIX, la operación a utilizar es fork(), que crea un proceso hijo con un duplicado del espacio de direcciones del padre, es decir, un duplicado del programa que se ejecuta desde la misma posición. Sin embargo, en

ambos casos, los padres e hijos (aunque sean un duplicado en el momento de la creación en sistemas tipos UNIX) son independientes y las modificaciones que uno haga en su espacio de memoria, como escritura de variables, no afectarán al otro.

Como padre e hijo tienen espacios de memoria independientes, pueden compartir recursos para intercambiarse información. Estos recursos pueden ir desde ficheros abiertos hasta zonas de memoria compartida. La **memoria compartida** es una región de memoria a la que pueden acceder varios procesos cooperativos para compartir información. Los procesos se comunican escribiendo y leyendo datos en dicha región. El sistema operativo solamente interviene a la hora de crear y establecer los permisos de qué procesos pueden acceder a dicha zona. Los procesos son los responsables del formato de los datos compartidos y de su ubicación

Reflexiona
¿Qué problemas podemos encontrar con el acceso a memoria compartida y multiprogramación?

Al terminar la ejecución de un proceso, es necesario avisar al sistema operativo de su terminación para que de esta forma el sistema libere si es posible los recursos que tenga asignados. En general, es el propio proceso el que le indica al sistema operativo mediante una operación denominada exit que quiere terminar, pudiendo aprovechar para mandar información respecto a su finalización al proceso padre en ese momento.

El proceso hijo depende tanto del sistema operativo como del proceso padre que lo creó. Así, el padre puede terminar la ejecución de un proceso hijo cuando crea conveniente. Entre estos motivos podría darse que el hijo excediera el uso de algunos recursos o que la funcionalidad asignada al hijo ya no sea necesaria por algún motivo.

Para ello puede utilizar la operación destroy. Esta relación de dependencia entre padre e hijo, lleva a casos como que si el padre termina, en algunos sistemas operativos no se permita que sus hijos continúen la ejecución, produciéndose lo que se denomina “terminación en cascada”.