PROGRAMACION CONCURRENTE

I.2 Recursos para la concurrencia.

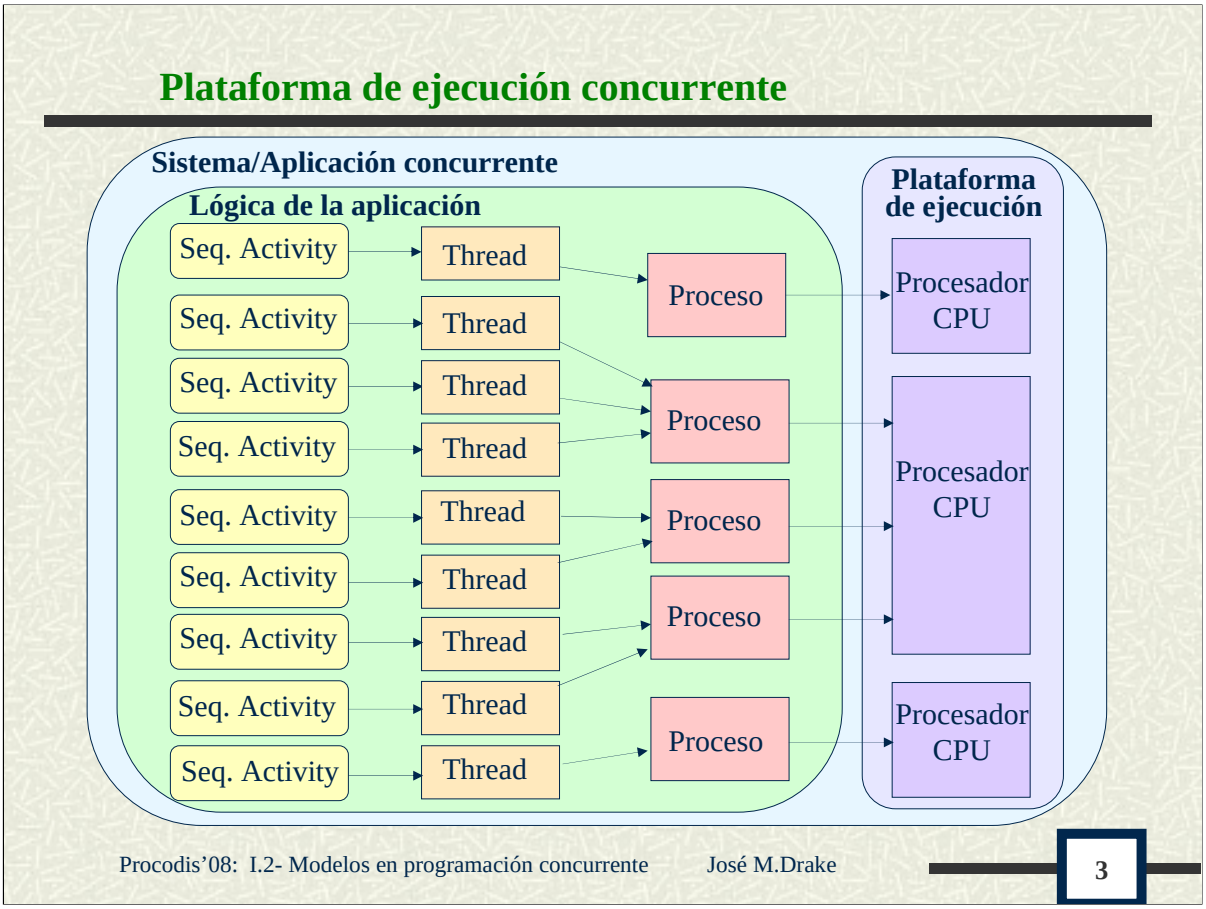
J.M. Drake

Notas:

Procesos Concurrentes

- Concepto de proceso.
- Estructura de los procesos.
- Estados de un proceso.
- Gestión de los procesos.

Notas:



Notas:

La concurrencia no es un término fácil de definir. Informalmente, un programa concurrente es un programa que ejecuta mas de una actividad simultáneamente. Si embargo, en muchos casos esta simultaneidad es una ilusión. En algunos sistemas concurrentes, las diferentes actividades son realizadas en diferentes CPU y la concurrencia es real (concurrencia física). En otros sistemas, solo hay una CPU y las diferentes actividades se ejecutan secuencialmente siguiendo una estrategia de entrelazamiento basada en repartir el tiempo de ejecución de la CPU (concurrencia virtual).

Un sistema informático puede ser un único procesador, una máquina multiprocesadora o un grupo de máquinas administradas como una sola unidad y que comparten un mecanismo de gestión común.

Un proceso es una abstracción de los sistemas operativos que hace posible que un sistema informático pueda ejecutar simultáneamente diferentes actividades secuenciales. Un proceso es una abstracción lógica (no física). Las ligaduras de los procesos a los procesadores pueden cambiar dinámicamente. A cada proceso se le garantiza un cierto grado de independencia, de ausencia de interferencias y de seguridad. No se permite que un proceso pueda acceder a la memoria local de otro, deben comunicarse entre ellos a través de utilidades de comunicación proporcionadas por el entorno. La gestión de los procesos es pesada y requiere una gran capacidad de procesamiento para crear, gestionar, comunicar y cancelar los proceso.

Los thread son abstracciones del sistema operativo que hace posible ejecutar de forma concurrente una actividad secuencial. Los threads son ligeros y sólo disponen de los recursos necesarios para mantener independientemente su hilo de control que ejecuta la actividad secuencial asignada. Los threads de un mismo proceso comparten la memoria local y la utilizan para comunicarse entre ellas. La ejecución de la actividad que se asocia a un threads se puede planificar de forma independiente, utilizando para ello diferentes protocolos. Los threads se comunican entre sí de forma no segura a través de la memoria local que comparte, y de forma segura a través de los mecanismos sincronizados que le ofrecen los sistemas operativos.

Programas concurrentes y procesos.

- Un programa concurrente se concibe como un **conjuntos de thread o hilos de control**
- Cada thread ejecuta una única actividad secuencial.
- Cada thread se ejecuta en su propio **procesador virtual** ejecutando independientemente acciones de acuerdo con la actividad que tiene asociada.
- Los threads concurrentes intercambian entre sí mensajes con **información** y mensajes de **sincronismo**.
- Existen tres formas de **implementar** un programa concurrente:
  - Mediante un lenguaje concurrente (Ada, Java, etc.).
  - A través de un sistema operativo (UNIX, POSIX, Windows, etc).
  - Apoyándose en un software de comunicaciones (DIS-ADA, MMS, CORBA).

Notas:

Un programa concurrente se suele [Dijkstra, 98] concebir como un conjunto de **procesos**. Cada uno de los procesos se considera que es un programa secuencial, con una única línea de flujo de control (thread).

Se considera que cada proceso se ejecuta en su propio **procesador virtual**, y progresa de forma independiente a los otros procesos, ejecutando la secuencia de acciones que corresponden a su propia línea de control de flujo.

En un **programa concurrente** se tienen que definir cada uno de los procesos que lo componen, así como las acciones de intercambio de información, y de sincronismo que deben tener lugar para que la ejecución concurrente de los procesos progrese constructivamente en su conjunto.

Un programa concurrente se formula utilizando bien un **lenguaje concurrente** (como Ada), o definiendo independientemente los procesos que constituyen el programa mediante un lenguaje secuencial (como el lenguaje C), y haciendo uso de la capacidad de multiprocesado que proporciona un sistema operativo multiprocesador (como UNIX) o un software de comunicación (como MMS).

Programa secuencial y concurrente

- **Programa secuencial**

  P;
  Q;
  R;

- Se interpreta como:

  $\forall$ Ejecución • $P \rightarrow Q \rightarrow R$

- Para todas las ejecuciones válidas P debe terminar antes de empezar Q y Q debe terminar antes de empezar R.

- **Programa concurrente**

  cobegin
  P; Q; R;
  coend;

- Se interpreta como:

  $\exists$ Ejecución • {not(P→Q) and not(P→R) and not(Q→P) and not(Q→R) and not(R→P) and not(R→Q)}

- Existe al menos una ejecución válida en la que los tres procesos se estén ejecutando simultáneamente.

Notas:

Un programa secuencial se puede considerar como un conjunto de procesos que se ejecutan secuencialmente. Así, un programa secuencial constituido por los tres procesos P, Q y R, tal como el siguiente:

P; Q; R;

equivale formalmente a establecer que

$\forall$ Ejecución • $P \rightarrow Q \rightarrow R$

Expresión que debe leerse como: Para todas las ejecuciones válidas del programa, P debe concluir antes que comience Q, y Q debe finalizar antes de que comience R.

Un programa concurrente se puede considerar como un conjunto de procesos que se pueden ejecutar bien secuencialmente en cualquier orden o concurrentemente con su ejecución solapada en el tiempo. Por ejemplo, un programa concurrente compuesto por tres procesos P, Q y R que se ejecutan concurrentemente se puede expresar como,

cobegin P; Q; R; **coend;**

lo cual significa formalmente que

$\exists$ Ejecución • {not(P→Q) and not(P→R) and not(Q→P) and not(Q→R) and not(R→P) and not(R→Q)}

que debe leerse como, los procesos P, Q y R son concurrentes si existe al menos una ejecución válida en la que durante algún intervalo de tiempo, los tres procesos se estén ejecutando simultáneamente.

Aspectos de los procesos

Cada entorno de programación concurrente necesita establecer algunas características sobre sus procesos:

- Naturaleza de los procesos.
- Jerarquía de los procesos.
- Mecanismos de inicialización.
- Mecanismos de finalización.
- Formas de declaración.

Notas:

Naturaleza de los procesos.

- Los procesos pueden ser:
  - **Procesos estáticos**: son instanciados al comienzo del programa. El programador los declara explícitamente.
  - **Procesos dinámicos**: Son creados durante la ejecución del programa, en función de los datos.
- En función de los procesos que pueden existir en un sistema, se clasifican:
  - **Sistema cerrado**: Se tiene un conocimiento en tiempo diseño (estático) sobre los procesos que constituyen el sistema y su actividad.
  - **Sistema abierto**: El número de procesos que constituyen el sistema varia en el tiempo de forma impredecible (en fase de diseño), bien porque crea dinámicamente procesos, porque carga clases desconocidas potencialmente activas, invoca procedimientos externos en otros subsistemas, etc.
- Crear, instanciar y destruir un proceso son tareas complejas y costosas en tiempo de procesador. Los procesos dinámicos no se pueden utilizar en sistemas de tiempo real.
- Mantener un proceso tiene un costo en recursos del sistema.

Notas:

Los procesos que constituyen un programa concurrentes pueden ser:

•**Procesos estáticos:** Son aquellos que son creados e instanciados al comienzo del programa. El número de procesos estáticos de un programa se mantiene fijo durante la ejecución del programa y todos ellos han sido explícitamente declarados por el programador.

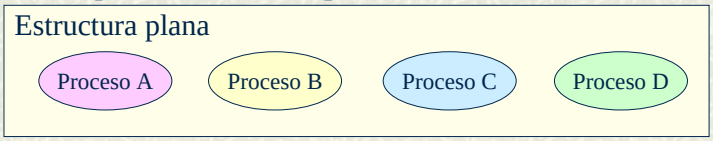
•**Procesos dinámicos:** Los procesos dinámicos que son creados durante la ejecución del programa, en función de los datos que están siendo manejados en una ejecución.

La creación, instalación y dotación de recursos de un proceso es una tarea compleja y muy costosa en tiempo. Esto conduce a que en los programas de tiempo real es muy poco frecuente que se utilicen procesos de generación dinámica.

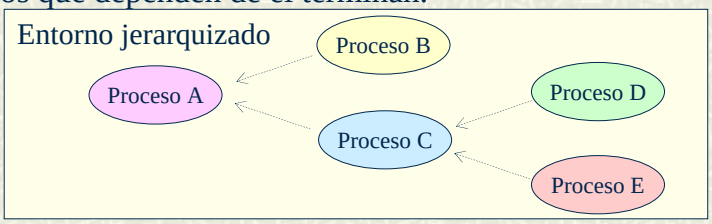
Jerarquía de los procesos.

Los procesos de un programa se pueden organizar en:

- **Estructura plana**: Todos los procesos son equivalentes. Evolucionan y terminan independientemente (procesos UNIX).



- **Estructura jerarquizada**: Entre los procesos existe unas relaciones de dependencia con estructura de árbol. Un proceso no termina hasta que los procesos que dependen de él terminan.



Notas:

Los procesos que constituyen un programa concurrente pueden organizarse de acuerdo con dos estructuras básicas:

•**Estructura plana:** En este casos todos los procesos, que en cada momento de la ejecución del programa concurrente han sido creados, son equivalentes, pudiendo cada uno de ellos evolucionar y terminar de forma independiente.

•**Estructura jerarquizada:** En estos casos, entre los procesos que constituyen el programa existe una dependencia jerárquica con estructura de árbol. Cada proceso se considera hijo del proceso en el que ha sido creado. En estos casos, un proceso debe permanecer activo, sin concluir, hasta que todos sus descendientes hayan terminado.

Modo de inicialización de los procesos.

- Inicializar un proceso es transferirle tras su creación la información que requiere para caracterizarlo individualmente.
- Existen diferentes estrategias:
  - En UNIX cada proceso es inicialmente una réplica del padre.
  - En Ada o Java en la creación se establecen unos parámetros de inicialización.
  - Tras la creación el hijo comunica con el padre y recibe la información de inicialización.

Notas:

La tarea de inicialización de un proceso, es la actividad por la que en la fase de creación del proceso, se les pasa a las variables de cada proceso unos valores iniciales. Tener la capacidad de inicialización de los procesos que se crean es muy útil, en especial cuando se generan dinámicamente o siguiendo estructuras tipo array, ya que en estos casos y a través de la inicialización, se puede particularizar cada proceso respecto de los demás, que son clónicos con él.

En ciertos ámbitos (por ejemplo dentro de UNIX), los procesos heredan de forma completa el entorno que su proceso padre tiene en el instante de creación, lo que a todos los efectos es una forma de inicialización.

Cuando no se dispone de la capacidad de inicialización de los procesos, se necesita que tras ser estos creados, establezcan explícitamente un proceso de comunicación con otro proceso existente (habitualmente su padre) a fin de recibir de él la información de inicialización.

Modos de finalización de los procesos.

- Existen múltiples formas de terminar un proceso:
  - La línea de flujo de control alcanza el final (con éxito).
  - Se produce una excepción no atendida (con fallo).
  - Un proceso ejecuta una sentencia “self terminate” (con fallo).
  - El proceso es abortado por otro proceso (con fallo).
  - Finalización coordinada “terminate” (con éxito).
  - No terminan nunca.

Notas:

La terminación de un proceso puede ser inducida por un amplio conjunto de causas:

•Conclusión de la ejecución del proceso de acuerdo con su línea de control de flujo. (Conclusión con SUCCESS).

•Conclusión por ocurrencia de una situación de error irrecuperable. (Conclusión con FAILURE).

•Conclusión del proceso por ejecución de una sentencia tipo "self terminate" (Conclusión con FAILURE internamente provocado).

•Conclusión del proceso por ejecución en otro proceso de la sentencia tipo "aborted". (Conclusión con FAILURE externamente provocado).

•Cuando un proceso en conjunción con su padre y hermanos alcanzan un estado terminación conjunta "terminate". (Conclusión con SUCCESS coordinada).

•Nunca, como consecuencia de que la línea de control de flujo no alcanza nunca una situación de finalización.

No todas estas causas de terminación están habilitadas en cada lenguaje de programación concurrente, o sistema operativo.

Declaración de procesos concurrentes.

- Declaración por sentencias estructuradas.

```
cobegin
    proceso_1, proceso_2, ..., proceso_n
coend;
```

- Declaración como componentes de bloque:

```
procedure Padre;
    process proceso_1;  begin ....end;
    process proceso_2; begin ....end;
begin .... end;
```

- Bifurcación (fork).

```
.....
{if  (fork() = = 0 )
        {.... }     /*(Se jecuta sólo en el proceso hijo}
else
        {.... }     /*(Se ejecuta sólo en el proceso padre y pid_hijo contiene el pid del nuevo
                     proceso hijo}
}.....
```

- Invocación no estructurada

Notas:

Existen diferentes formas de declarar un conjunto de procesos concurrentes, algunas de ellas son:

•**Declaración mediante sentencia estructurada:** En este caso se dispone de una sentencia estructurada, tal como **cobegin ... coend** (o **parbegin .. parend**) que declara dentro de un programa la ejecución de un conjunto de procesos concurrentes. En estos casos el proceso padre queda suspendido, hasta que todos los procesos que se han declarado concluyen.

•**Declaración como componentes del programa o bloque**: Cada bloque que tenga contenga declarados uno o mas procesos en su parte declarativa, se ejecuta en concurrencia con ellos. En este caso, los procesos hijos inician su ejecución en sincronismo con el "begin" que inicia el programa o bloque en que están declarados, padre e hijos se ejecutan concurrentemente, y así mismo el padre no puede concluir hasta que todos los hijos hayan terminado de algún modo.

•**Bifurcación (fork):** En este caso la ejecución de una sentencia fork en el proceso padre induce la creación de un proceso hijo clónico que continua ejecutándose concurrentemente con el proceso padre, ambos a partir de la sentencia siguiente a la sentencia fork. Esta es la forma habitual de crear procesos en un entorno UNIX.

•**Ejecución del comando no estructurado (exec):** En este caso el proceso padre realiza una llamada al sistema operativo, para que este cree un nuevo proceso, ejecutando sobre él, los programas que se pasan como argumento. Estos programas arrancan desde su inicio, y se ejecutan en concurrencia y con igual nivel jerárquico que el proceso padre que continua por la sentencia posterior a la sentencia de llamada al sistema.

Estados de un proceso.

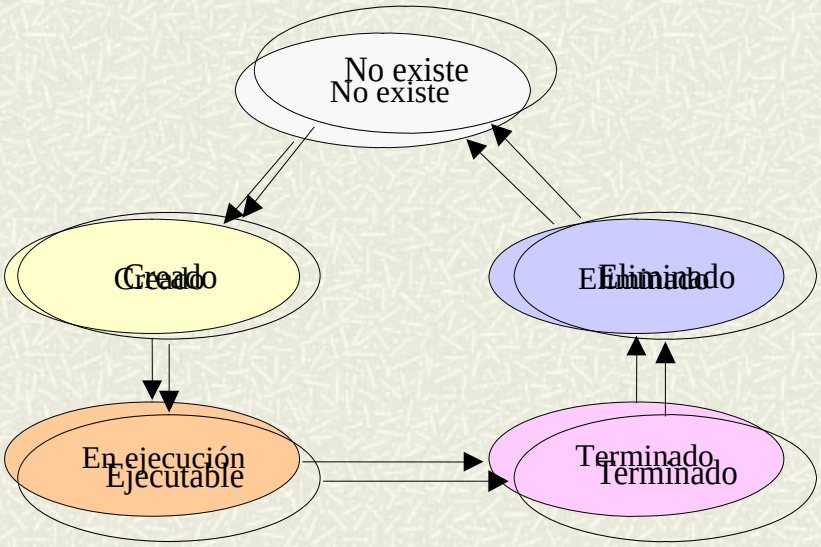


Diagrama básico de estados de un proceso.

Diagrama básico de estados de un proceso.

Notas:

Un proceso tiene un ciclo de vida que se puede describir mediante un diagrama de estados. En la siguiente figura se muestra un diagrama inicial que muestra los estados básicos de un proceso.

- Un proceso pasa a estar **creado** cuando se instancia un componente de programa en el que está declarado.
- El proceso pasa a estado de **ejecutable** cuando se ejecuta la sentencia (cobegin .. coend) del proceso padre en el que se requiere su ejecución.
- El proceso pasa al estado t**erminado,** cuando por causa de concluir su línea de control de flujo, o por algún tipo de error finaliza su operación.
- Cuando la sentencia "cobegin ... coend" a la que pertenece un proceso, concluye como consecuencia de que todos los procesos en ella incluidos han finalizado, el proceso queda **eliminado** y todos los recursos que le habían sido asignados son retornados al sistema.
- Por último, el proceso deja de existir cuando concluye el componente software en el que fue definido.

- El diagrama de estados que puede seguir un proceso, es mucho más compleja que esta. En especial en un sistema real, el estado "ejecutable" se descompone en múltiples nuevos estados.

Estados de un proceso en un entorno multiproceso.

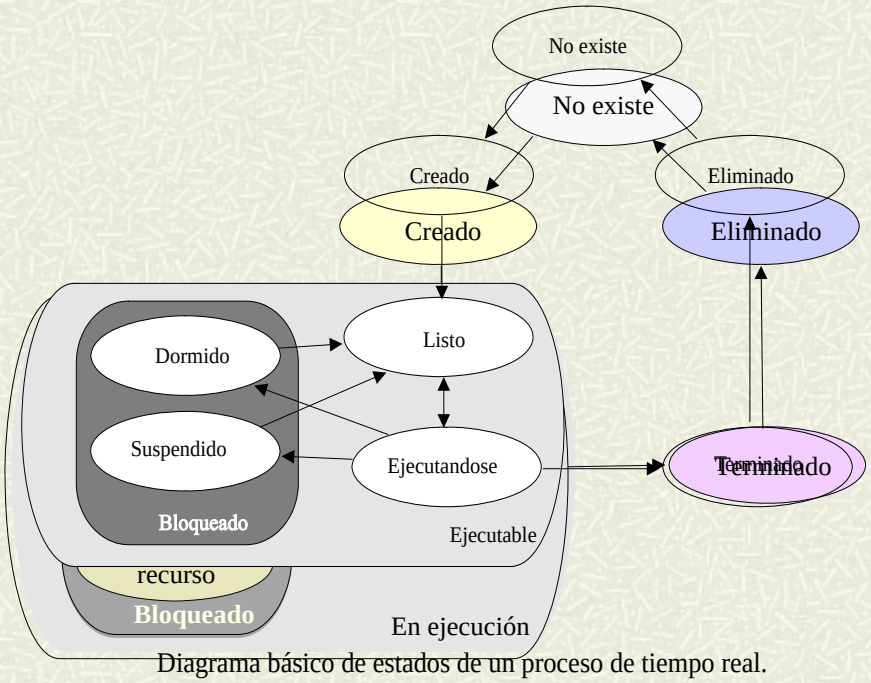


Diagrama básico de estados de un proceso de tiempo real.

Notas:

Para introducirnos en un mayor de detalle sobre los estados que sigue un proceso, debemos pasar a un nivel más bajo de descripción y considerar que un cierto número de procesos se encuentran en estado ejecutable en un mismo procesador físico. En este caso, de entre todos los procesos ejecutables en el procesador solo uno de ellos podrá estar haciendo uso de la CPU y se encontrará en un nuevo estado que hemos denominado **ejecutándose.** Los restantes procesos se encontrarán en estado **suspendido** (listo para su ejecución). El tránsito repetido para cada proceso entre los estados ejecutándose y listo para ejecución es controlado por el **planificador de procesos** (scheduler) del sistema operativo multiproceso del procesador físico.

•Un proceso "suspendido" para ejecución pasará al estado "ejecutándose", cuando tras encontrarse libre la CPU del procesador, es seleccionado por el planificador, de acuerdo con su criterio de gestión (FIFO, priorizada, etc.).

•Un programa que se encuentra en el estado "ejecutándose" en la CPU, pasa a estado "suspendido" por :

- •Al finalizar el tiempo de ejecución que tenía asignado el proceso.
- •Cuando en el mismo procesador pasa a estado de "suspendido" un proceso de mayor prioridad.
- •Cuando el proceso queda a la espera de un eventos externo.

•Estado **dormido** que corresponde al estado que alcanza un proceso cuando ejecuta una sentencia de suspensión temporal tal como "sleep" o "wait".

•Estado **en espera de evento** que corresponde al estado que alcanza un proceso como consecuencia de que se encuentra a la espera de un evento.

•Estado **en espera de un recurso** corresponde a un proceso que se encuentra bloqueado en un mutex en espera de recibir el acceso a un recurso.

Gestión de los procesos.

- Los procesos de un programa concurrente son gestionados por el sistema de soporte de tiempo real (RTSS) del sistema operativo
- El RTSS debe disponer de un bloque de control de procesos (PCB):
  - Identificador del proceso (PID)
  - Status del proceso.
  - Entorno del proceso.
  - Enlaces.

Notas:

La gestión de un programa concurrente es una tarea mucho más compleja que la gestión de un proceso secuencial. Los múltiples procesos que constituyen un programa concurrente, deben ser creados, gestionados y terminados en cada procesador disponible del sistema. Todas estas actividades son realizadas por una componente del sistema operativo que se denomina **Sistema de Soporte de Tiempo Real (RTSS).**

El RTSS debe disponer de información relativa a cada uno de los procesos creados en el sistema. Esta información se ubica dentro de una estructura de datos que se suele denominar **bloque de control de proceso (PCB)**. En particular, un proceso debe ser capaz de ser identificado de forma unívoca por el RTSS. A cada proceso se le ha asignados un conjunto de recursos, y así mismo, los procesos pueden estar asociados en múltiples configuraciones, según su estado funcional. Así mismo, a lo largo de la existencia de un proceso, es asignado muchas veces al correspondiente procesador. Cada vez que el sistema es retirado del procesador y de nuevo es asignado a él, se requiere que el proceso continúe su ejecución por el mismo punto en que estaba, y sin que nada haya cambiado (desde el punto de vista interno del proceso) por la doble transición. Esto significa que en cada transición, el estado de los registros de la CPU de cada proceso, deben ser conservados durante la fase de no ejecución.

La información que debe estar incluida en el PCB de un proceso, es al menos:

•**Identificador del Proceso (ID)**: Estructura de datos que identifica de forma única el proceso.

•**Status del proceso** : Describe el estado en que se encuentra el proceso.

•**Entorno del proceso**: Información que requiere el proceso para continuar su ejecución.

•**Enlaces**: Permite la asociación entre procesos, y su adscripción a las colas.

Estrategia de implementación del RTSS

Existen dos estrategias de incorporar el RTSS:

- El compilador incorpora el RTSS al código ejecutable del programa.
- El compilador introduce en el código sentencias de acceso al S.O. que es el que gestiona los procesos.

La primera estrategia es mas simple. La segunda es mas eficiente y segura.

Notas:

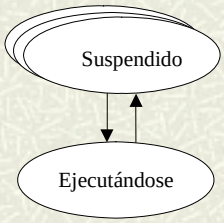
En la organización de un sistema operativo, o de un lenguaje de programación concurrente, existen dos estrategias de incorporación del RTSS:

•El compilador del lenguaje concurrente genera un RTSS al que incorpora todas las capacidades de gestión de lo procesos, y que se empaqueta (linked) con los códigos correspondientes a todos los procesos que constituyen el programa. Todos juntos constituyen un programa global que es ejecutado por el sistema operativo como si fuese una aplicación simple.

•La gestión de los procesos es soportada directamente por el sistema operativo, y en la compilación de cada proceso, el compilador incorpora al código las llamadas al sistema operativo que se necesitan para realizar las operaciones de interacción entre procesos y de uso de los recursos que exige el programa concurrente.

Aunque el primero es el más simple de realizar, y es el que corresponde a muchos de los compiladores de lenguajes de programación concurrentes, el segundo es mucho más eficiente, y es el único que puede llevar la programación concurrente hasta su posibilidades más avanzadas.

Planificación de los procesos



- La política de planificación de procesos (Schedulling) es el criterio para elegir el proceso que se ejecuta de entre los “Suspendidos” estos es listos para la ejecución.
- El comportamiento funcional de un programa no debe depender de la política de planificación.
- La política de planificación si influye en la temporización del programa.
- La sincronización entre procesos no debe basarse en la política de planificación.

Notas:

Durante la ejecución de un programa concurrente hay muchos procesos que se encuentran en el estado "listos" para la ejecución. Todos estos procesos compiten entre sí por los procesadores disponibles. El orden en que los procesos en espera se asignan a la CPU, es controlada por el RTSS, y la política que sigue para elegir el proceso que debe ejecutarse es denominada **política de planificación de procesos (scheduling)**.

Un principio fundamental de la programación concurrente es:

> **"El comportamiento funcional de un programa concurrente no debe depender de la política de planificación de los procesos que impone el RTSS.“**

Es importante resaltar, que en este principio se refiere al comportamiento funcional del programa. La respuesta temporal de los procesos (y en consecuencia , la correctitud funcional de un sistema de tiempo real) si que es fuertemente afectada por la política de planificación.

No respetar este principio supone una dependencia inaceptable de la plataforma en que se ejecuta el programa concurrente.

Un corolario de este principio, es que nunca debe tenerse en cuenta la política de planificación a efecto de establecer relaciones de sincronización entre procesos.

Características de la política de planificación

- Capacidad de desalojar el proceso en ejecución:
  - Política desalojante (preemptive).
  - Política no desalojante (non-preemptive)
- Criterio de selección del proceso a ejecutar:
  - Política FIFO.
  - Política LIFO.
  - Política Round-Robin
  - Prioridad estática
  - Prioridad dinámica.

Notas:

Un primer aspecto de una política de planificación es su capacidad o no de desalojo del proceso en ejecución. Desde este punto de vista caben dos posibilidades:

•**Política desalojante (pre-emptive)**: El proceso que se encuentra en ejecución puede ser desalojado por el RTSS si por el criterio de priorización que se utiliza, hay otro proceso ejecutable de mayor prioridad que él.

•**Política no desalojante (no pre-emptive):** En proceso que se encuentra en ejecución no puede ser desalojado de la CPU hasta que termina, se duerme o se suspende en espera de un evento.

Un segundo aspecto de la política de planificación, es el criterio por el que en un punto de planificación se selecciona el proceso que debe ser elegido para ser ejecutado. Políticas utilizadas son:

•**Orden FIFO:** Se elige el proceso mas antiguo de entre los que están en la lista de espera.

•**Orden LIFO: :** Se elige el proceso mas reciente de entre los que están en la lista de espera.

•**Round Robin:** con esta política el RTSS asigna a cada proceso una intervalo de tiempo, y se sigue para la selección del siguiente proceso una política FIFO. Cuando un proceso es desalojado por concluir el tiempo asignado, pasa a ocupar el último puesto de la cola.

•**Prioridad estática:** a cada proceso se le asigna una prioridad absoluta. El RTSS selecciona en cada punto de planificación aquel proceso con mayor prioridad de la lista de espera.

•**Prioridad dinámica:** a cada proceso se le asigna una prioridad base, y un coeficiente de incremento de la prioridad con el tiempo de espera. La prioridad de un proceso se considera variable, y en cada instante toma un valor función de la prioridad base y del tiempo transcurrido desde la ultima asignación al procesador.

Planificación de una aplicación de tiempo real.

- La planificación de una aplicación de tiempo real es el aspecto crítico del diseño de estos sistemas:
  - ¿Existe una estrategia de planificación que permita realizar todas las tareas previstas cumpliendo cada una de ellas los requerimientos temporales establecidos?
  - ¿cuál es la correcta asignación de prioridades que debe asignarse a los procesos para que los requerimientos temporales se satisfagan?
- Curso **"Sistemas de Tiempo Real"**

Notas: