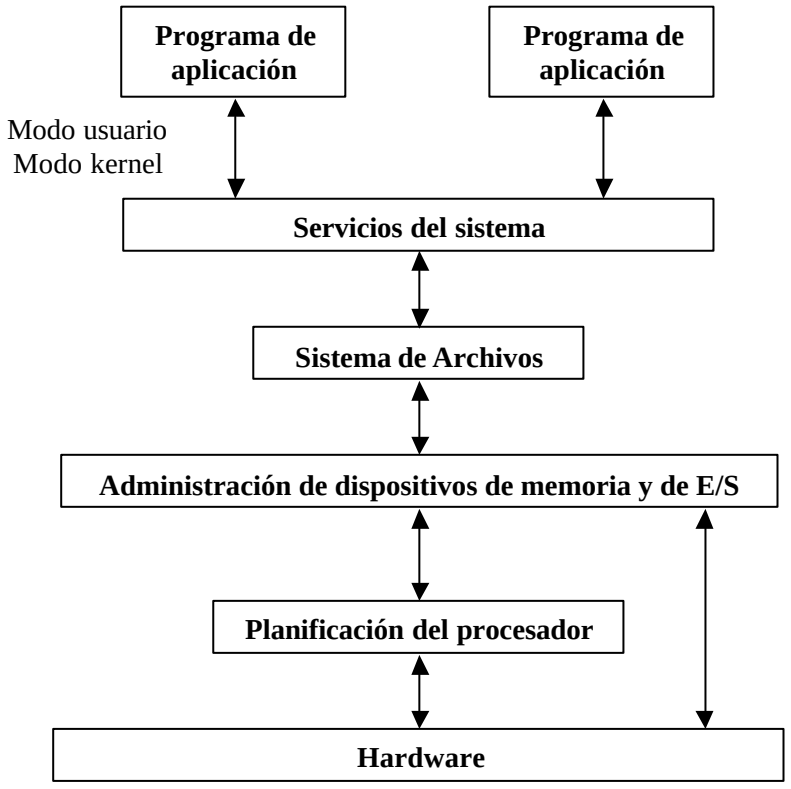
TEMA 3: EL NÚCLEO DE UN SISTEMA OPERATIVO

1. Introducción. Funciones del núcleo de un S.O.

Los Sistemas Operativos proporcionan un número potencialmente grande de servicios accesibles al usuario. Uno de los problemas principales que se encuentran los diseñadores de sistemas operativos es cómo manejar esta complejidad de funciones a muchos niveles de detalle mientras proporcionan un producto fiable y fácil de mantener.

Con la experiencia que se tiene sobre el tema, se está tendiendo a la implantación de sistemas operativos en forma de niveles jerárquicos de abstracción. De esta forma, se obtendrán una serie de módulos que ocultarán los detalles de la estructura de datos y los algoritmos de procesamiento que utilicen. Externamente se conocerá cada módulo por realizar unas funciones específicas que sólo pueden usar las capas superiores, pero los detalles de cómo lo hacen no estarán disponibles.

Como resultado, los módulos tienden a ser pequeños y fáciles de comprender e implementar. Con todo esto, el diseño jerárquico ofrece una mayor facilidad en la depuración, modificación y verificación de los sistemas operativos.



El núcleo del sistema operativo es el nivel más interno del sistema operativo. Actúa de interfaz entre el hardware básico y el resto del sistema operativo. Su finalidad es constituir un entorno adecuado en el que se puedan desarrollar los distintos procesos.

Las funciones básicas del núcleo de un sistema operativo son:

- Manipulación de las interrupciones
- Creación/Destrucción de procesos
- Cambios de estados de procesos
- Planificación de los procesadores
- Suspensión/Reanudación de procesos
- Sincronización de procesos
- Comunicación entre procesos
- Manipulación de bloques de control de procesos
- Soporte de las actividades de entrada/salida
- Soporte de la asignación/liberación del almacenamiento
- Soporte del sistema de archivos
- Soporte de un mecanismo de llamada/regreso al procedimiento
- Soporte de ciertas funciones contables del sistema

Características del núcleo:

- Se encuentra residente en Memoria Principal
- Tienden a ser ininterrumpibles
- Se ejecutan con el máximo privilegio

El núcleo es la parte más dependiente del hardware. Normalmente se escribe en ensamblador, pero existen otros lenguajes para diseño de sistemas operativos: BCPL (Basic Compiled Programming Language), BLISS (Basic Language for Implementing Systems), C, Pascal Concurrente, PL/2, ...

2. Requisitos básicos del hardware

Veamos el hardware básico necesario para soportar un modelo de sistema operativo sencillo.

Se requieren las siguientes posibilidades en el hardware:

- Un mecanismo de interrupción. Salvará como mínimo el PC (Contador de Programa) y transferirá el control a una posición de memoria.

- Un mecanismo de protección que debe implementarse en el hardware de direccionamiento de la memoria. Esto se realiza para proteger la memoria de un proceso del acceso no autorizado por parte de otros procesos.

- Repertorio de instrucciones reservadas. Con el fin de que varios procesos concurrentes no se interfieran entre ellos. Estas instrucciones llevan a cabo funciones tales como:
    - Autorizar e inhibir las interrupciones
    - Conmutar un procesador entre distintos procesos
    - Acceder a los registros usados por el hardware de protección de memoria
    - Llevar a cabo la E/S
    - Manejo y control del reloj en tiempo real
    - Parar el procesador central

Para establecer cuándo se pueden usar estas instrucciones reservadas, existen dos modos de funcionamiento del computador:

- Modo núcleo o supervisor (kernel): Ejecuta todas las instrucciones (tanto reservadas como de usuario)

- Modo usuario: Sólo ejecuta las instrucciones de usuario

Cambio de modos:

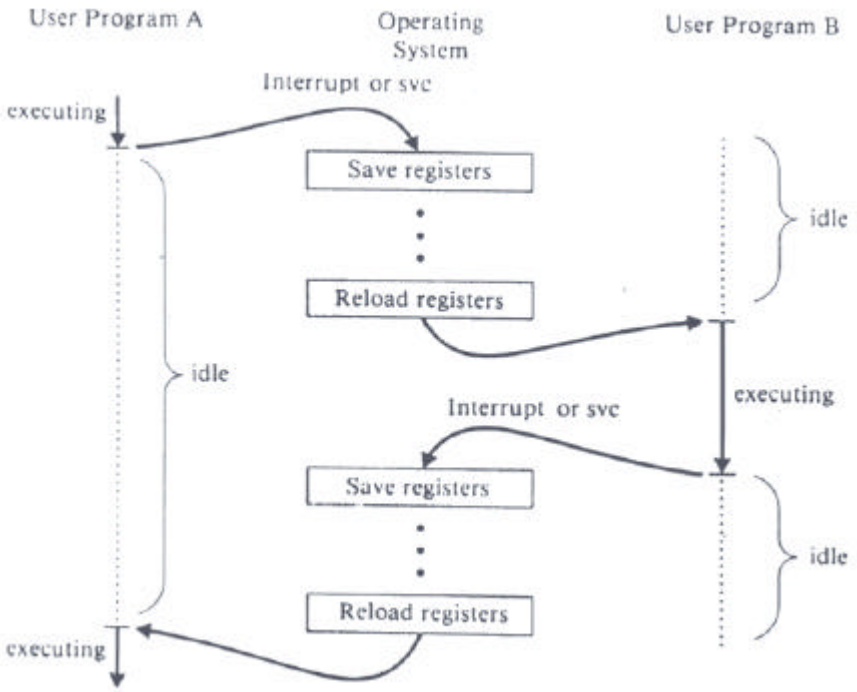
- Modo usuario a modo supervisor:
    - El proceso usuario solicita una llamada al supervisor para usar alguna función del sistema operativo

- Ocurre una interrupción
- Error en un proceso de usuario
- Se intenta ejecutar una instrucción reservada en modo usuario (caso especial de error)

- Modo supervisor a modo usuario:
  - Instrucción reservada

3. Gestión de interrupciones

Una interrupción es una respuesta a un suceso asíncrono o excepcional que automáticamente reserva el estado actual de la CPU para permitir su continuación posterior y provoca una transferencia automática a una rutina específica.



Básicamente, existen seis clases de interrupciones:

1. **Interrupciones SVC (SuperVisor Call, llamadas al supervisor).** Son iniciadas por un proceso en ejecución para solicitar una operación de E/S, obtener memoria, …

2. **Interrupciones de E/S.** Son iniciadas por el hardware de E/S. Indican a la CPU el cambio de estado de un canal o dispositivo. Se producen cuando finaliza una operación de E/S o cuando un dispositivo para a estado de listo.

3. **Interrupciones externas.** Son causadas por diversos sucesos, incluyendo la expiración del cuantum de reloj, la pulsación de la tecla de interrupción del teclado, …

4. **Interrupciones de reinicio.** Ocurren cuando se pulsa el botón de reinicio de la consola (Reset), …

5. **Interrupciones de verificación del programa.** Son causadas por varios tipos de errores experimentados al ejecutar un proceso, como una operación de división por cero, formato de datos erróneo, ejecución de una operación inválida, intento de acceder a una posición de memoria fuera de los límites (violación de memoria), …

6. **Interrupciones de verificación de la máquina.** Son ocasionadas por el mal funcionamiento de la máquina.

La gestión de interrupciones la realiza el manipulador (controlador) de interrupciones (FLIH, First Level Interrupt Handler).

El manipulador de interrupciones es la parte del sistema operativo responsable de proporcionar la respuesta adecuada a las señales procedentes tanto del exterior como del interior del sistema (interrupciones externas e internas).

Misión del FLIH:

1. Determinar el origen de las interrupciones.

2. Iniciar el servicio de las mismas.

Funcionamiento del FLIH:

1. Inhibir las interrupciones (si no lo hace el hardware).

2. Guardar la información no salvada por el hardware. Dos posibles formas:

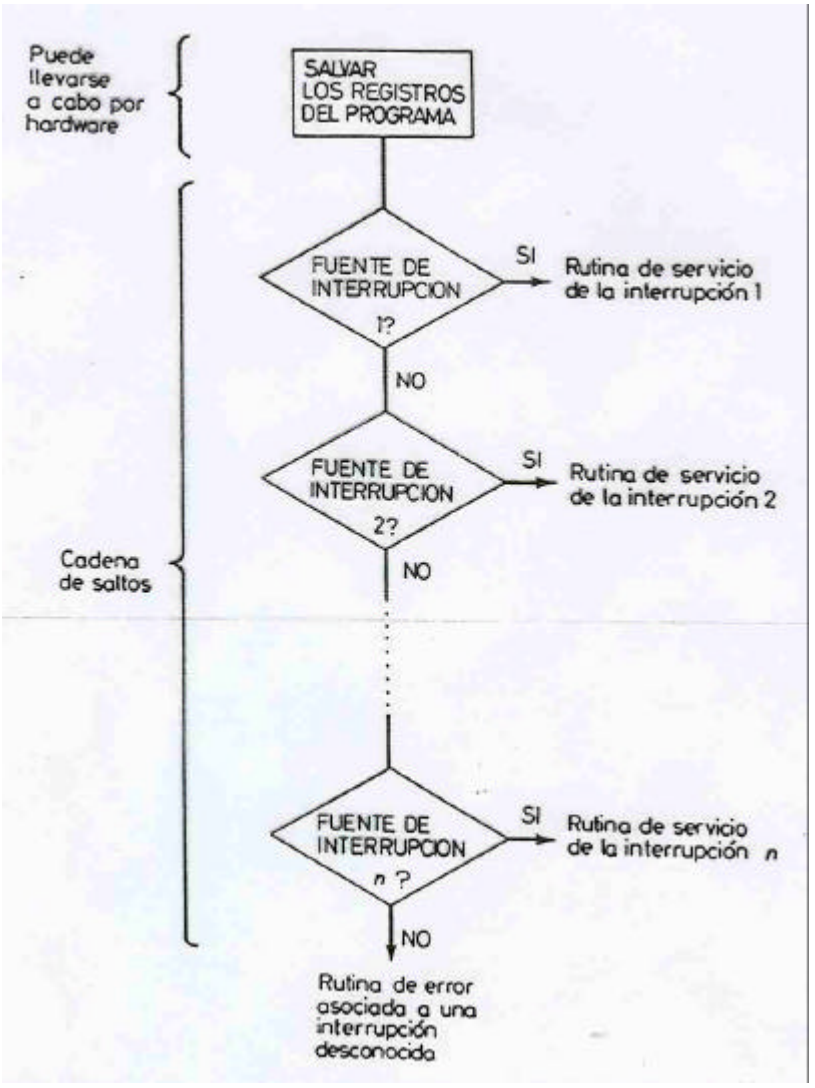
    - Salvar los registros que se usen en el tratamiento de la interrupción.

    - Usar un conjunto de registros suplementarios sólo usados en modo supervisor para procesar el servicio de la interrupción con lo que no se necesitan salvar los registros afectados del programa.

3. Identificar la causa de la interrupción (depende del hardware). Podemos distinguir varios casos:

   - Con todas las interrupciones se transfiere el control a la misma posición de memoria (caso más sencillo).

   - El hardware de interrupción es capaz de distinguir las diferentes fuentes de interrupción y transferir el control del programa a una posición de memoria distinta para cada una de ellas, con lo que se reduce el tiempo de identificación de la interrupción.



   - Agrupar las interrupciones por tipos. El hardware es capaz de distinguir las interrupciones según su tipo y transferir el control a posiciones de memoria distintas.

Consideraciones en el tratamiento de las interrupciones:

- Inhibición de las interrupciones: Si durante el procesamiento de una interrupción ocurre otra, ésta queda pendiente hasta que el mecanismo de interrupción se reactive. Es decir, cuando se inhiben las interrupciones, la CPU no puede atender a otras interrupciones hasta que se reactive el mecanismo de interrupción.

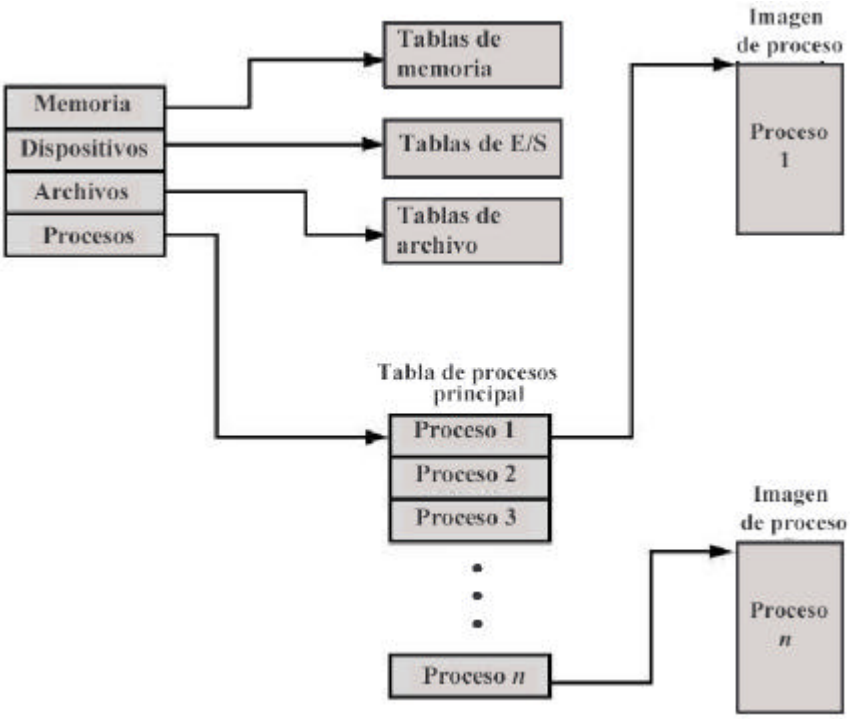
- Prioridades entre las distintas fuentes de interrupción: Una rutina de interrupción puede ser interrumpida por otra fuente de interrupción de más alta prioridad (caso en que un periférico requiera una respuesta mucho más rápida que otro).

- Una interrupción puede alterar el estado de los procesos.

4. Gestión de procesos. Planificación

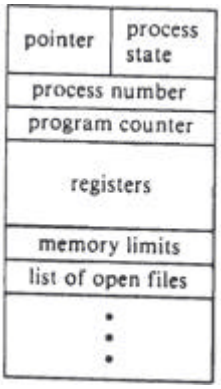
4.1. Representación de los procesos

Los programas del núcleo actúan sobre estructuras de datos que constituyen la representación física de todos los procesos del sistema (tablas de control). En las tablas de control están representadas todas las estructuras de datos del sistema (memoria, E/S, ficheros y procesos).

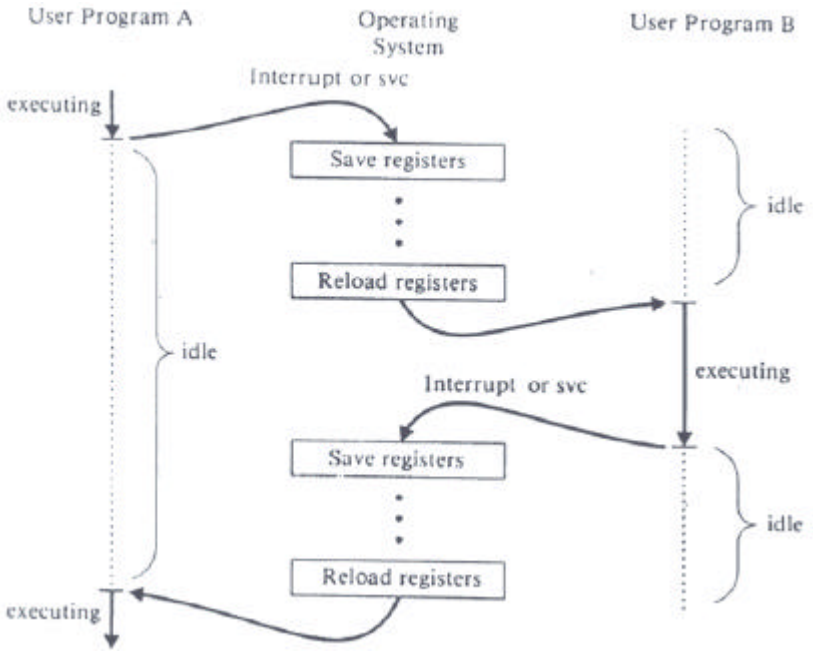


Para poder mantener un entorno adecuado en el que puedan existir los procesos, se debe de trabajar sobre algún tipo de estructura de datos. Cada proceso se representa mediante un descriptor de proceso o bloque de control de proceso (PCB) que estará formado por toda aquella información que puede variar de un proceso a otro. Es decir, contendrá:

- El estado del proceso: preparado, en ejecución, bloqueado, ...
- Entorno volátil: Formado por toda la información que hay que guardar cuando se pierde el control del procesador, como puede ser el Contador de Programa (CP), los registros acumuladores, el registro de estado, ...
- Otra información: Identificador del proceso (índice o número), nombre, fecha de creación del proceso, ...

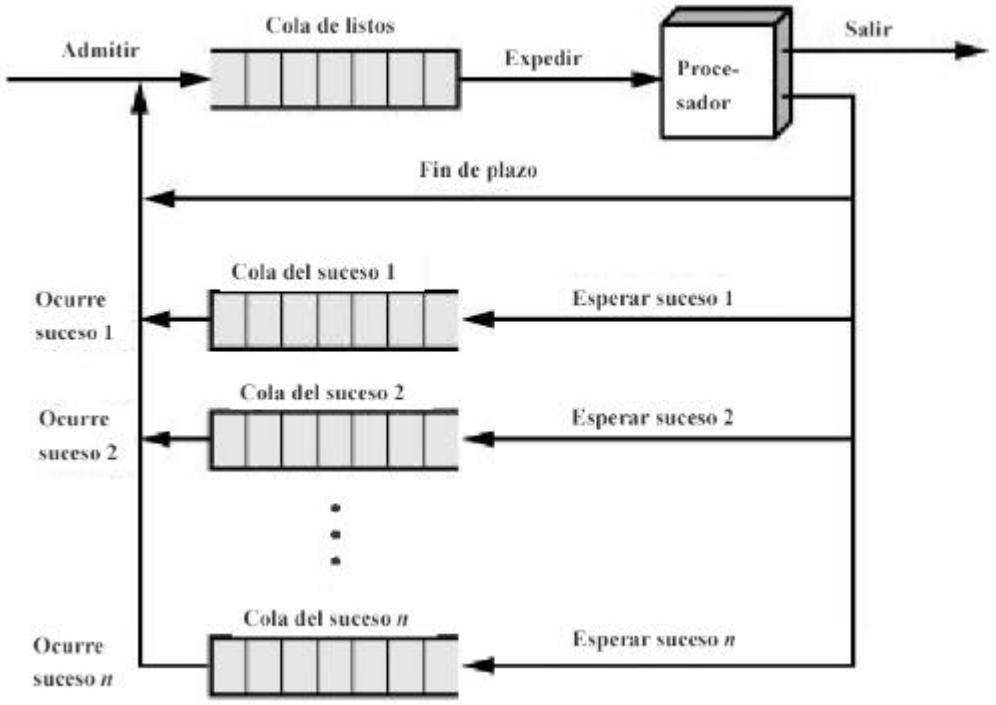


Cada vez que un proceso debe liberar la CPU, se debe guardar la información de estado (entorno volátil) para poder continuar correctamente más adelante. El "cambio de contexto" consiste en retirar el control de la CPU al proceso actual y asignárselo a otro proceso.



Las formas de agrupar los descriptores de procesos pueden ser mediante una o varias listas encadenadas:

- Cola única que contiene a todos los procesos.
- Múltiples colas correspondientes a los distintos tipos de procesos (por estado, por dispositivo, ...).



El encargado de asignar los procesadores a los distintos procesos será el despachador (dispatcher).

4.2. Funcionamiento del dispatcher

1. Decidir si se cambia el proceso actual. Se plantea la siguiente pregunta: ¿Es el proceso en curso el más apropiado para seguir ejecutándose en el procesador?
2. Decidir qué proceso en estado preparado es el siguiente al que se le asigna el procesador. Existen dos formas:
    - Que el propio dispatcher incorpore el algoritmo que determina el proceso a ejecutarse.
    - Que se elija directamente el primer proceso de la cola de procesos preparados. En este caso, existirá otro módulo que se encargará de mantener la cola de procesos preparados en un determinado orden. Este módulo se ejecutará antes de que se realice un cambio de contexto.

3. Salvar la información el proceso que se estaba ejecutando (entorno volátil) en el descriptor del proceso (si no se realizó durante el tratamiento de la interrupción).

4. Cargar el entorno volátil del proceso elegido.

5. Pasar el control de la CPU a dicho proceso elegido.

Intervención del dispatcher

El dispatcher actuará cuando:

1. Un proceso no quiera seguir ejecutándose (por ejemplo, cuando finaliza o cuando realiza una operación wait sin éxito)

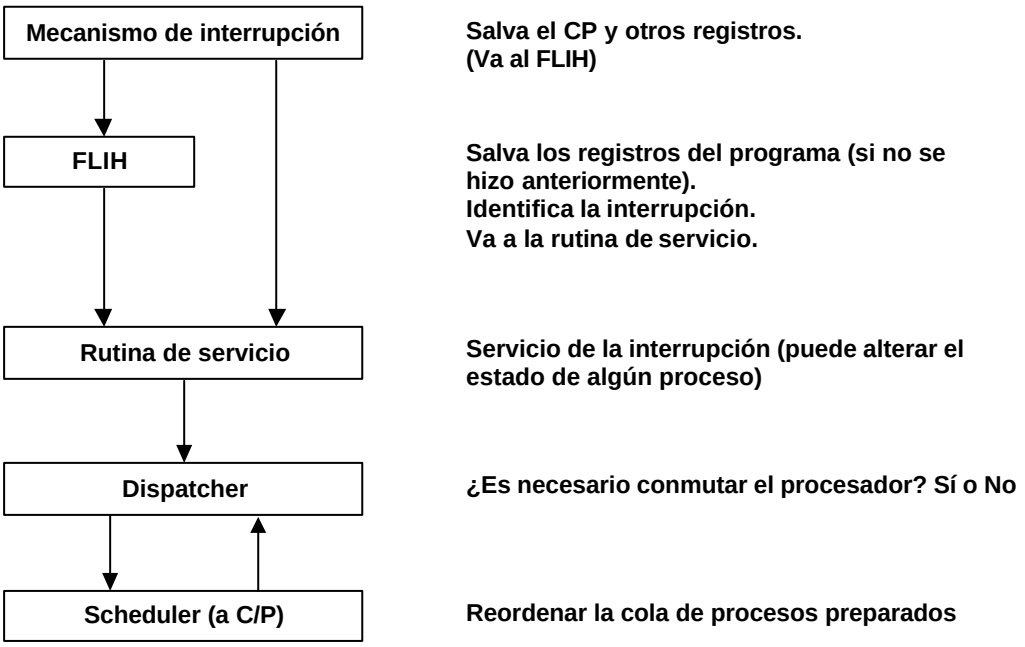
2. Un proceso inicia una operación y el S.O. determina que no puede seguir ejecutándose (por ejemplo, cuando un proceso inicia una operación de E/S).

3. Un proceso agota su cuantum de tiempo.

4. Un suceso cambia el estado de un proceso bloqueado, pasándolo a preparado. En este caso, el despachador entrará en funcionamiento para asegurarse de que se ejecutará el de mayor prioridad.

Todos estos sucesos causan modificaciones que afectan a los parámetros globales como la prioridad de los procesos o el número de procesos en el sistema.

Relación entre el FLIH y el despachador:



4.3. Planificación de procesos

La planificación estudia el problema de cuándo asignar los procesadores y a qué procesos asignárselos. La planificación la realiza el planificador o scheduler.

La idea es planificar todos los procesos que se encuentran en la cola de procesos preparados (procesos ejecutables).

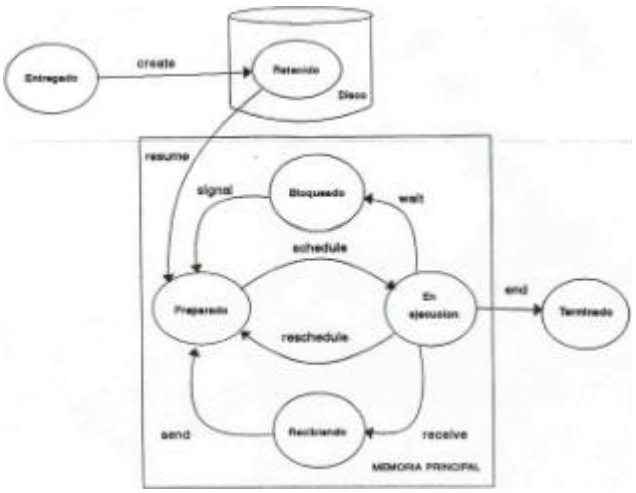
4.3.1. Niveles de planificación

Existen dos schedulers de CPU principales: el scheduler a corto plazo y el scheduler a largo plazo.

- **Scheduler a largo plazo:** determina qué trabajos se admiten en el sistema para su procesamiento. Inicialmente, los procesos son encolados en un dispositivo de almacenamiento masivo (normalmente, un disco) para que posteriormente el scheduler a corto plazo los pueda seleccionar y consigan iniciar su ejecución.
- **Scheduler a corto plazo (scheduler de la CPU):** determina qué proceso debe ser ejecutado en cada instante de tiempo o, dicho de otra forma, el scheduler a corto plazo determinará el proceso que seleccionará el dispatcher para asignarle el control de la CPU.

La diferencia principal entre estos schedulers es la frecuencia de su ejecución. Mientras que el scheduler a corto plazo se ejecuta muy continuamente (cuando se realiza un cambio de contexto, que ocurre con frecuencia del orden de milisegundos), el scheduler a largo plazo se ejecuta menos frecuentemente (pueden pasar minutos entre las llegadas de trabajos al sistema). El scheduler a largo plazo controla el grado de multiprogramación (número de procesos en memoria).

Con la utilización de un scheduler a largo plazo, aparece un nuevo estado en los procesos:



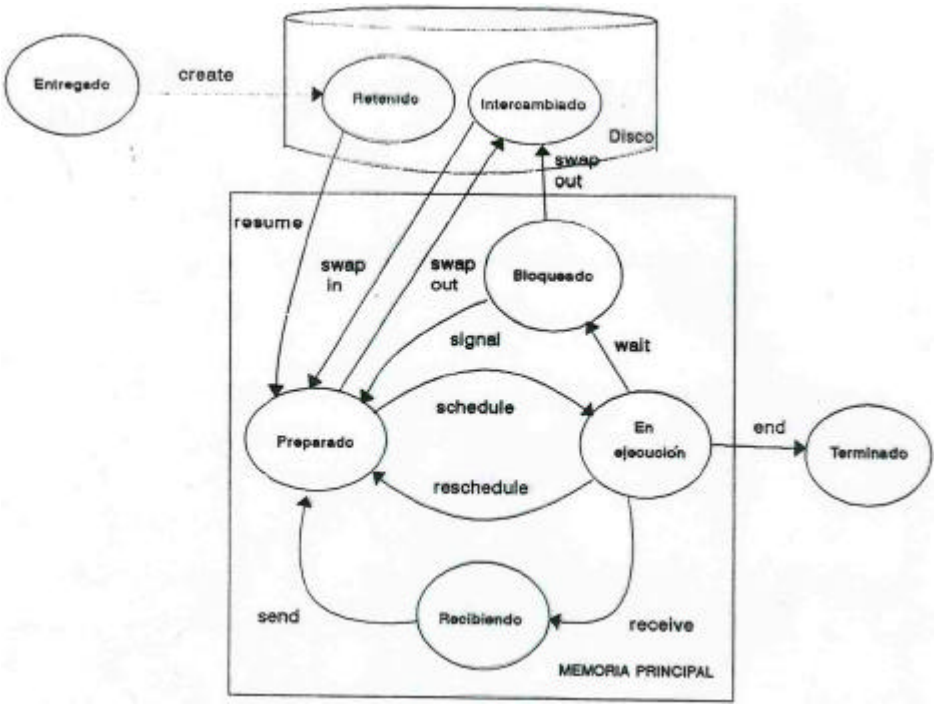
Estado **Retenido**: Cuando un programa desea formar parte de la lista de procesos preparados para entrar en ejecución.

Transición **resume**: Cuando el scheduler a largo plazo determina que el trabajo ha sido admitido en el sistema para su procesamiento.

En algunos sistemas, el scheduler a largo plazo puede estar ausente o ser mínimo. Esto puede ocurrir en sistemas en los que un nuevo proceso es directamente puesto en memoria por el scheduler a corto plazo. La estabilidad de estos sistemas depende de la limitación física (por ejemplo, el número de terminales disponibles) o del número de usuarios.

En otros sistemas, especialmente con memoria virtual o tiempo compartido, puede introducirse un estado intermedio de planificación. El **scheduler a medio plazo** decide qué procesos deben abandonar momentáneamente la memoria principal y, transcurrido un tiempo, decidirá cuáles podrán volver a ella para continuar su ejecución. A este esquema se le conoce como intercambio o swapping. Puede ser conveniente para reducir el grado de multiprogramación o para disponer de más memoria principal.

El scheduler a medio plazo incorpora un nuevo estado en los procesos:



Estado **Intercambiado**: Cuando no hay espacio en memoria central para poder seguir en ella.

Transición **swap out**: Cuando se pretende intercambiar un proceso al área de swap (área de intercambio).

Transición **swap in**: Cuando el proceso retorna del área de swap (área de intercambio).

4.3.2. Consideraciones para la planificación

Para evaluar el comportamiento de los distintos algoritmos, se definen los siguientes parámetros:

- **Tiempo de retorno:** Tiempo que transcurre entre la entrega del programa para su ejecución y la obtención de los resultados. Es similar al tiempo de ejecución. Se utiliza en sistemas batch.

- **Tiempo de respuesta:** Tiempo que transcurre desde que el sistema tiene el programa hasta que comienzan a salir los resultados. Estará formado por un tiempo de espera y un tiempo de ejecución. Se usa en sistemas interactivos.

- **Rendimiento o productividad:** Número de trabajos procesados por unidad de tiempo.

Las complicaciones que aparecen para el scheduler son:

- El tiempo disponible de CPU es finito, por tanto, si a un usuario (proceso) se le asigna más tiempo, será a costa de otro.

- Objetivos contrapuestos, ya que minimizar el tiempo de respuesta para usuarios interactivos implica no ejecutar trabajos batch (excepto cuando no haya usuarios interactivos).

- Los procesos son únicos e impredecibles, es decir, cuando se inicia un proceso, nunca se sabe cuánto tiempo pasará hasta que el proceso se bloquee por E/S, semáforo u otro motivo (no se puede predecir cuánto tiempo necesitará cada proceso).

Los algoritmos de planificación de procesos deben tener en cuenta lo siguiente:

- **Justicia.** Asegurar que cada proceso reciba su parte justa de CPU.

- **Eficiencia.** Mantener ocupada la CPU el 100% del tiempo.

- **Minimizar el tiempo de respuesta para usuarios interactivos.**

- **Minimizar el tiempo de retorno para procesos batch.**

- **Maximizar la productividad.**

Los criterios más importantes que se utilizan para clasificar los algoritmos de planificación de procesos son si usan una estrategia de planificación apropiativa o no apropiativa, y el sistema de prioridad.

4.3.2.1. Planificación apropiativa y planificación no apropiativa

Una estrategia de planificación es no apropiativa si, una vez que se le ha asignado la CPU a un proceso, ya no se le puede arrebatar a no ser que el proceso decida terminar o solicite una operación de E/S. A las planificaciones no apropiativas no les afectan las interrupciones del sistema.

Una estrategia de planificación es apropiativa si al proceso se le puede arrebatar la CPU. Esto puede ocurrir cuando se produce una interrupción, cuando se acaba el cuantum o cuando el sistema determina que la CPU puede ser usada mejor por otro proceso.

La planificación apropiativa es útil en sistemas en los cuales los procesos de alta prioridad requieren una atención rápida, como ocurre con los sistemas en tiempo real. También en los sistemas interactivos de tiempo compartido, la planificación apropiativa es importante para garantizar tiempos de respuesta aceptables. La apropiación tiene un precio, ya que el cambio de contexto implica un gasto extra. Por ello, para que esta técnica sea efectiva, se deben mantener muchos procesos en almacenamiento principal de manera que siempre haya procesos preparados para cuando la CPU esté disponible.

En los sistemas no apropiativos, los trabajos largos retrasan a los cortos, pero el tratamiento para todos los procesos es más justo. Los tiempos de respuesta son más predecibles porque los trabajos nuevos de alta prioridad no pueden desplazar a los trabajos en espera.

4.3.2.2. Sistema de prioridad

A cada proceso se le asigna una prioridad, de forma que se le asigne la CPU a los procesos en estado “preparado” que tengan la prioridad más alta.

Para evitar que los procesos de alta prioridad se ejecuten indefinidamente, el “scheduler” puede decrementar la prioridad del proceso actualmente en ejecución (a esto se le conoce como envejecimiento o aging).

Las prioridades pueden ser asignadas automáticamente por el sistema o bien asignarse externamente. Pueden ser estáticas o dinámicas. Pueden asignarse de forma racional o de manera arbitraria, ... Las prioridades pueden o no estar implícitas en el algoritmo de planificación.

Las prioridades estáticas no cambian. No responden a cambios en el ambiente, que podrían hacer necesario un ajuste de prioridades.

Las prioridades dinámicas responden a los cambios. Pueden variar según las condiciones del sistema, procesos, ...

Un algoritmo muy sencillo que utiliza una prioridad dinámica es el siguiente:

Se asigna una prioridad 1/f siendo f la fracción del último cuantum que el proceso utilizó:

Ej.: Si utilizó 2 ms de 100 ms ? Prioridad = 1/2/100 = 50
Si utilizó 50 ms de 100 ms ? Prioridad = 1/50/100 = 2
Si utilizó todo el cuantum ? Prioridad = 1/100/100 = 1

Se considera que la prioridad más alta es la de menor valor.

4.3.3. Algoritmos de planificación de procesos

Estos algoritmos son válidos para los tres schedulers.

Planificación por orden de llegada: FCFS (First Come First Served)

Es la disciplina de planificación más simple. La carga de trabajo se procesa por orden de llegada. Ofrece un bajo rendimiento y un índice de productividad bajo al no considerar el estado del sistema y las necesidades de recursos de los procesos, y un pobre tiempo de respuesta a sucesos del sistema debido a la falta de prioridad de los procesos y a no ser una planificación apropiativa.

Planificación del trabajo más corto primero: SJF (Shortest Job First)

Es una planificación no apropiativa según la cual se ejecuta primero el proceso en espera que tenga menor tiempo estimado de ejecución hasta terminar. SJF reduce el tiempo de espera promedio del FCFS.

SJF favorece a los trabajos cortos frente a los largos.

El problema que tiene es que se exige conocer con exactitud el tiempo que tardará en ejecutarse un proceso y esa información no suele estar disponible; por ello, se basará en los tiempos estimados de otras ejecuciones (esto se puede obtener en sistemas de producción). Por otra parte, SJF requiere que todos los trabajos se encuentren disponibles simultáneamente.

Esta planificación, al igual que FCFS, no es apropiativa y, por tanto, no resulta útil en ambientes de tiempo compartido en los que se deben garantizar tiempos de respuesta razonables.

En el caso de 4 trabajos con tiempos de ejecución a, b, c y d, el primer trabajo finaliza en el tiempo a, el segundo en tiempo a+b, el tercero en tiempo a+b+c y el cuarto en tiempo a+b+c+d, con lo que

$$t_{medio-respuesta} = \frac{4a + 3b + 2c + d}{4}$$

"a" contribuye más al promedio que los otros tiempos, por lo que debe ser el trabajo más corto, "b" el siguiente, ...

Planificación por fracciones de tiempo: RR (Round-Robin)

En los entornos interactivos (como los sistemas de tiempo compartido) el objetivo principal es dar buenos tiempos de respuesta y, en general, compartir equitativamente los recursos del sistema entre todos los usuarios. Por ello, sólo las planificaciones apropiativas se pueden considerar en tales entornos y una de las más populares es la división en el tiempo en fracciones o Round Robin.

Básicamente, el tiempo de CPU se divide en fracciones o cuantums que se asignan a los procesos. Ningún proceso puede ejecutarse durante más de una fracción de tiempo habiendo procesos esperando.

Puede ocurrir que:

- Un proceso necesite más de un cuantum: El proceso pasará a la cola de procesos preparados.

- Un proceso ceda el control de la CPU (por ejemplo, operación de E/S): Se planifica otro proceso para que se ejecute.

Cada proceso recibirá 1/N (siendo N el número de procesos preparados) del tiempo de la CPU.

Los procesos cortos tendrán buenos tiempos de respuesta (ya que se pueden ejecutar en un único cuantum de tiempo).

¿Qué longitud debe tener el cuantum?

Supongamos que la conmutación de un proceso a otro toma 5 ms (ocupado en salvar y cargar registros, mapas de memoria, ...).

Supongamos que $t_{cuantum}$ = 20 ms.

Por tanto, cada proceso tarda en total 25 ms (20 ms por el proceso y 5 ms por la conmutación de la CPU). Esto significa que el 20% del tiempo de CPU lo gasta el S.O.

Para mejorar la eficacia, podemos aumentar el tamaño del cuantum a 500 ms, con lo que el S.O. gasta el 1%.

Problema: Si 10 usuarios pulsan <return> al mismo tiempo, entonces habrá 10 procesos en la lista de procesos ejecutables. Inmediatamente, uno de los procesos comenzará, el segundo tendrá que esperar aproximadamente ½ sg, ... y el último tendrá que esperar 5 sg antes de tener una oportunidad, siempre y cuando todos los procesos agoten su cuantum. Es decir, el tiempo de respuesta de un comando corto es grande.

Conclusión:

- Cuantum pequeño: Se conmutan muchos procesos y, por tanto, menos eficiencia de CPU.
- Cuantum grande: Respuesta pobre a demandas de usuarios interactivos.

Solución de compromiso: $t_{cuantum}$ = unos 100 ms.

NOTA: Se supone que todos los procesos son de la misma importancia

Planificación por colas de niveles múltiples: MLQ (Multi Level Queues)

La idea de esta planificación es clasificar la carga de trabajo de acuerdo con sus características y mantener colas separadas de procesos atendidos por distintos planificadores.

Un ejemplo de división de la carga de trabajo podría ser: procesos del sistema, programas interactivos y trabajos por lotes.

Un proceso se adscribe a una cola específica en virtud de sus atributos, que pueden ser suministrados por el usuario o por el sistema.

La planificación que se puede usar entre las colas son por prioridad absoluta o de división del tiempo con alguna variante que refleje la prioridad relativa de los procesos colocados en las colas específicas. En el caso de la planificación por prioridad absoluta, a los procesos procedentes de la cola que tiene la prioridad más alta se les da servicio hasta que la cola se quede vacía.

Una posible planificación para las colas de procesos individuales es: para colas de procesos interactivos usar planificación por fracción de tiempo y para colas de procesos por lotes (batch) una planificación FCFS.

Planificación por colas de niveles múltiples con realimentación: MLFQ (Multi Level Feedback Queues)

Es una variante de la planificación MLQ. En lugar de hacer que se asignen clases fijas de procesos a colas específicas, la idea es hacer depender el paso de un proceso por el sistema según el tiempo de ejecución. Así, por ejemplo, cada proceso puede comenzar en la cola de nivel superior. Si el proceso se completa en una fracción de tiempo dada, podrá salir del sistema. Los procesos que necesiten más de una fracción de tiempo para completarse, pasarán a una cola de procesos con prioridad inferior.

La idea es dar tratamiento preferente a los procesos cortos y hacer que los que consumen recursos o los procesos largos se "hundan" lentamente en colas de nivel inferior para mantener elevada la utilización de la CPU.

Una posible variante de la organización sería:

Cola más alta	Se asigna 1 cuantum de tiempo
...	Se asignan 2 cuantums de tiempo
...	Se asignan 4 cuantums de tiempo
...	...

Este algoritmo de planificación es válido con vistas a los procesos que se van incorporando.

Planificación primero con el tiempo de ejecución más corto: SRTF (Shortest Run-Time First)

Es una versión más dinámica y apropiativa que el SJF. Como el SJF, este algoritmo requiere estimación del tiempo de procesamiento requerido para cada proceso. Sin embargo, estas estimaciones sirven para determinar sólo las prioridades iniciales. Cada vez que un proceso necesita parar o es interrumpido, el tiempo de procesamiento transcurrido hasta ese instante es restado del estimado. Cuando el proceso vuelve a estar preparado otra vez, adquiere una prioridad más alta, reflejando la realidad de que le queda menos trabajo.

Planificación de procesos en Unix de SUN

El sistema Unix de SUN realiza la planificación de procesos asignando prioridades base (la más alta es la -20 y la más baja la +20) y efectuando ajustes de prioridad. Estos ajustes se calculan en respuesta a cambios en las condiciones del sistema. Los ajustes se suman a las prioridades base para calcular las prioridades actuales. El proceso con la prioridad actual mayor se despacha primero.

El ajuste de prioridades favorece en gran medida a los procesos que recientemente han utilizado poco tiempo de CPU.

5. Implementación de primitivas de comunicación y sincronización. Equivalencias entre primitivas

Se debe implementar en el núcleo de todo S.O. al menos un mecanismo de sincronización y comunicación.

Los semáforos se incluyen en el núcleo por las siguientes razones:

- Deben de estar al alcance de todos los procesos.
- WAIT puede provocar que un proceso quede bloqueado, lo que hará que se llame al despachador para reasignar el procesador.
- Para una rutina de interrupción, una forma de activar un proceso (hacerlo ejecutable) es ejecutar un SIGNAL sobre un semáforo sobre el cual un proceso haya llevado a cabo una operación WAIT sin éxito.

Desarrollemos nuestra implementación en base a los siguientes conceptos:

- Bloqueo y desbloqueo
- Entrada y salida de procesos en la cola del semáforo
- Asignación de los procesadores
- Indivisibilidad

Bloqueo y desbloqueo

WAIT: Bloquea el proceso cuando el valor del semáforo es 0.

SIGNAL: Libera un proceso bloqueado (en caso de que lo haya).

Consideraciones en la implementación de los semáforos:

- Con cada semáforo asociamos una cola de procesos.
- Cuando se efectúa una operación WAIT sin éxito, el proceso se añade a la cola del semáforo y se hace "no ejecutable" (estado bloqueado).
- La operación SIGNAL sobre un semáforo, saca un proceso de la cola asociada al semáforo y lo hace "ejecutable" (estado listo).
- Un semáforo se implementará en base a dos elementos:
    - Entero
    - Puntero asociado a la cola

Implementación de las operaciones:

WAIT(S):	SI (S<>0) ENTONCES	S=S-1
	DE LO CONTRARIO	añadir el proceso a la cola del semáforo y hacerlo "no ejecutable"

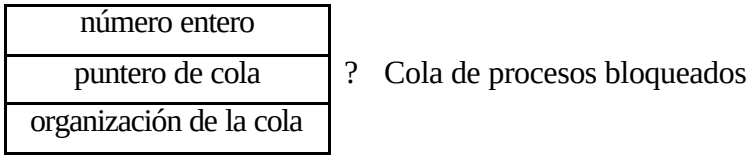
SIGNAL(S):	SI (la cola está vacía) ENTONCES	S=S+1
	DE LO CONTRARIO	sacar un proceso de la cola del semáforo y hacerlo "ejecutable"

Si se bloquea un proceso, cuando se desbloquee con un SIGNAL se ejecuta directamente la siguiente instrucción al WAIT que bloqueó el proceso. Con este criterio, no tenemos que incrementar el valor del semáforo en la operación SIGNAL cuando se desbloquee el proceso.

Entrada y salida de procesos en la cola del semáforo

- Organización de la cola: Cada semáforo puede tener una organización diferente. Ej.: FIFO, prioridades, …

- Estructura del semáforo:



Asignación de los procesadores

Las operaciones WAIT y SIGNAL pueden alterar el estado de un procesador por lo que se tendrá que llamar al despachador para decidir qué proceso debe ejecutarse. Pueden ocurrir dos casos:

- Que se efectúe sin éxito una operación WAIT (se bloquea el proceso) y, por tanto, el procesador conmutará de proceso. Si se efectúa una operación SIGNAL sobre un semáforo en el que hay algún proceso bloqueado, también habría que llamar al dispatcher.

- Que tenga éxito la operación WAIT o que la operación SIGNAL se realice sobre un semáforo en el que no haya ningún proceso bloqueado y, por tanto, es bastante probable que se siga ejecutando el mismo proceso. Este caso se podria implementar de dos formas:

    - La vuelta desde dichas operaciones al mismo proceso se realiza directamente (más eficiente, pero más compleja).

    - La vuelta se realiza llamando al dispatcher (menos eficiente, pero menos costosa). Es conveniente.

Indivisibilidad

Solo un proceso puede ejecutar una operación WAIT o una operación SIGNAL sobre el mismo semáforo en un momento determinado. Por tanto, se deben implementar como secciones críticas.

En una configuración de:

- Un único procesador:
    - Bloqueo: Inhibe el mecanismo de interrupción. El proceso que ejecuta un WAIT o un SIGNAL no puede perder el control del procesador central.
    - Desbloqueo: Activación del mecanismo de interrupción.
- Varios procesadores con una memoria común:
    - Se utiliza la instrucción TSL (comprobar y actualizar de forma indivisible). Durante la ejecución de esta instrucción se inhibe el acceso por parte de otros procesadores a la posición de memoria utilizada.

El bloqueo y el desbloqueo no se pueden usar como sustitutivas de WAIT y SIGNAL. Veamos esto más detenidamente en la siguiente tabla:

	WAIT y SIGNAL	Bloqueo y desbloqueo
Propósito:	sincronización entre procesos	exclusión mutua de los procesos respecto de las rutinas WAIT y SIGNAL
Nivel de implementación:	software	hardware
Mecanismo de retardo:	con una cola	espera ocupada o inhibición de las interrupciones
Tiempo típico de retardo:	algunos segundos	algunos microsegundos