UNIDAD 2

2. Administración de procesos y del procesador.

Competencia a desarrollar.	**Actividades de Aprendizaje.**
Específica(s): Comprende las técnicas de administración de procesos para crear procesos empleando los mecanismos que presenta el sistema operativo para la comunicación y sincronización. **Genéricas:** • Capacidad de análisis y síntesis. • Capacidad de investigación. • Habilidad para buscar, procesar y analizar información procedente de fuentes diversas	• Elaborar un diagrama las transiciones de estado de los procesos para reconocer las características que los distinguen. • Representar mediante ejemplos de la vida real el concepto de proceso, programa y procesador, y trasladarlo al contexto de las computadoras. • Diferenciar los conceptos de: algoritmo, programa, proceso, tarea o job, sesión y lote, valorando la utilidad de cada uno de ellos mediante un glosario. • Definir las diferencias fundamentales y específicas de proceso, thread y multi-thread • Investigar los mecanismos empleados para la sincronización y comunicación entre procesos, así como diferenciar los Threads y Procesos. • Definir el concepto de interbloqueo (deadlock) y analizar su detección, prevención y recuperación.

2.1 Concepto de proceso.

Un proceso es básicamente como un programa en ejecución. Consta del programa ejecutable, los datos y la pila del programa, su contador de programa, apuntador de pila y otros registros, y la otra información que se necesita para ejecutar el programa.

La manera sencilla de tener una noción intuitiva de lo que es un proceso consiste en pensar en los sistemas con tiempo compartido. En forma periódica el sistema operativo decide suspender la ejecución de un proceso y dar inicio a la ejecución de otro, por ejemplo, porque el primero haya tomado ya más de su parte del tiempo del CPU, en terrenos del segundo.

Cuando un proceso se suspende temporalmente como éste, debe reiniciarse después exactamente en el mismo estado en que se encontraba cuando se detuvo. Esto significa que toda la información relativa al proceso debe guardarse en forma explícita en algún lugar durante la suspensión.

En muchos sistemas operativos, toda la información referente a cada proceso, diferente del contenido de su espacio de direcciones, se almacena en una tabla de sistema operativo, llamada Bloque de control de Procesos, ver ***figura #3***, la cual es un arreglo o lista enlazada de estructuras, una para cada proceso en existencia corriente.

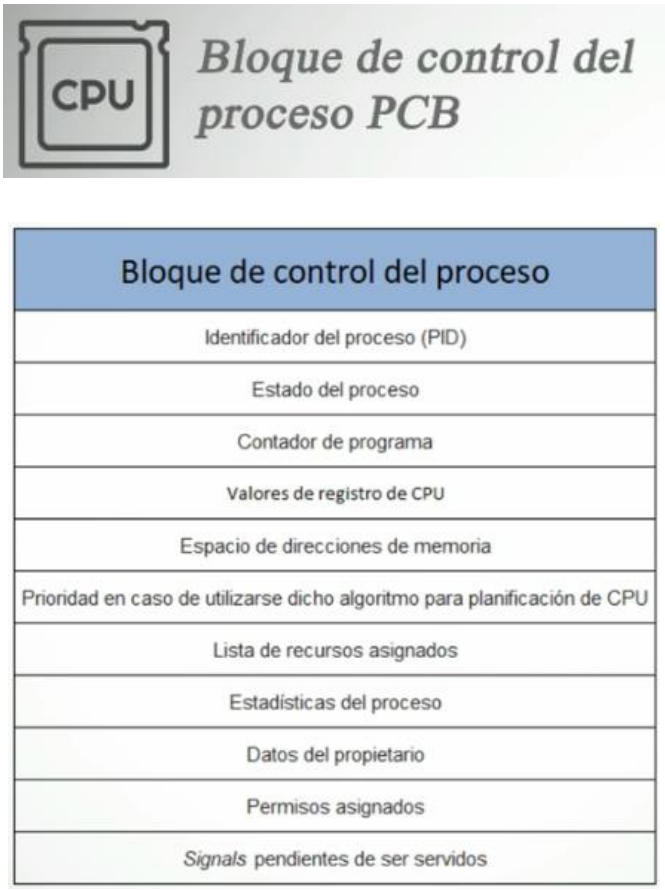

Figura #3. Bloque de Control de Procesos.

Si un proceso puede crear uno o más procesos diferentes (conocidos como proceso hijo) y estos procesos a la vez originan procesos hijos, se llega rápidamente a la estructura del árbol de procesos, observe ***figura # 4.***

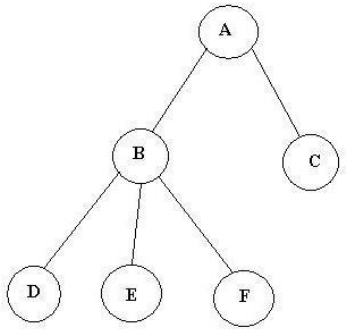

Figura # 4. Estructura de árbol. Proceso Padre - Hijo.

El proceso A creó dos procesos derivados, B y C. El proceso B creó tres derivados, D, E y F.

Se dispone de otras llamadas al sistema para solicitar más memoria (o para liberar memoria no utilizada), esperar a que termine un proceso hijo y cubrir su programa con uno diferente.

En un sistema de multiprogramación, el (CPU) también cambia de un programa a otro, ejecutando cada uno en decenas o cientos de milisegundos. En tanto que, en rigor, en cualquier instante de tiempo, el CPU está ejecutando sólo un programa, en el curso de un segundo puede trabajar en varios programas, con la ilusión de paralelismo.

Proceso: Informalmente se define como la actividad que resulta cuando un proceso ejercita un programa, y formalmente consiste en un vector formado por lo siguiente, ***figura # 5:***

P = < C, Co, E, S, A >

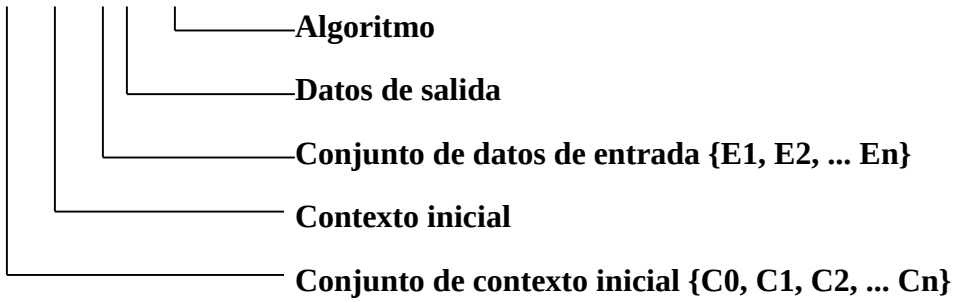

Figura # 5. Definición formal de proceso.

Un proceso puede tomar diferentes estados, puede estar ***corriendo***, puede estar ***libre*** o puede estar ***bloqueado***. Si consideramos que todo proceso está constituido de una serie finita de actividades elementales una ***región crítica*** de un proceso se define como el conjunto de actividades elementales cuya ejecución exige el monopolio de recursos compartidos.

Formalmente se define como el conjunto de partes de los contextos internos compartidos, observe la ***figura # 6.***

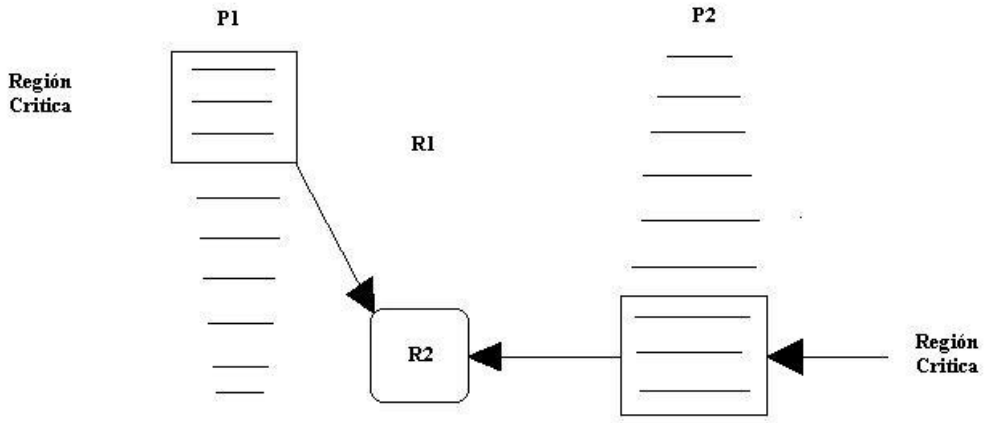

Figura # 6. Regiones críticas.

Los problemas que deben resolverse en un contexto de procesos concurrentes (como regiones críticas) son los siguientes: 1

- Exclusión mutua.

- Sincronización.

- Dead lock (Abraso mortal ó Interbloqueo)

2.2 Estados y transiciones de los procesos.

Los estados de los procesos son internos del sistema operativo y transparente al usuario. Para éste, su proceso estará siempre en ejecución independientemente del estado en que se encuentre internamente el sistema.

Los procesos se pueden encontrar en tres estados, observe ***figura # 7***.

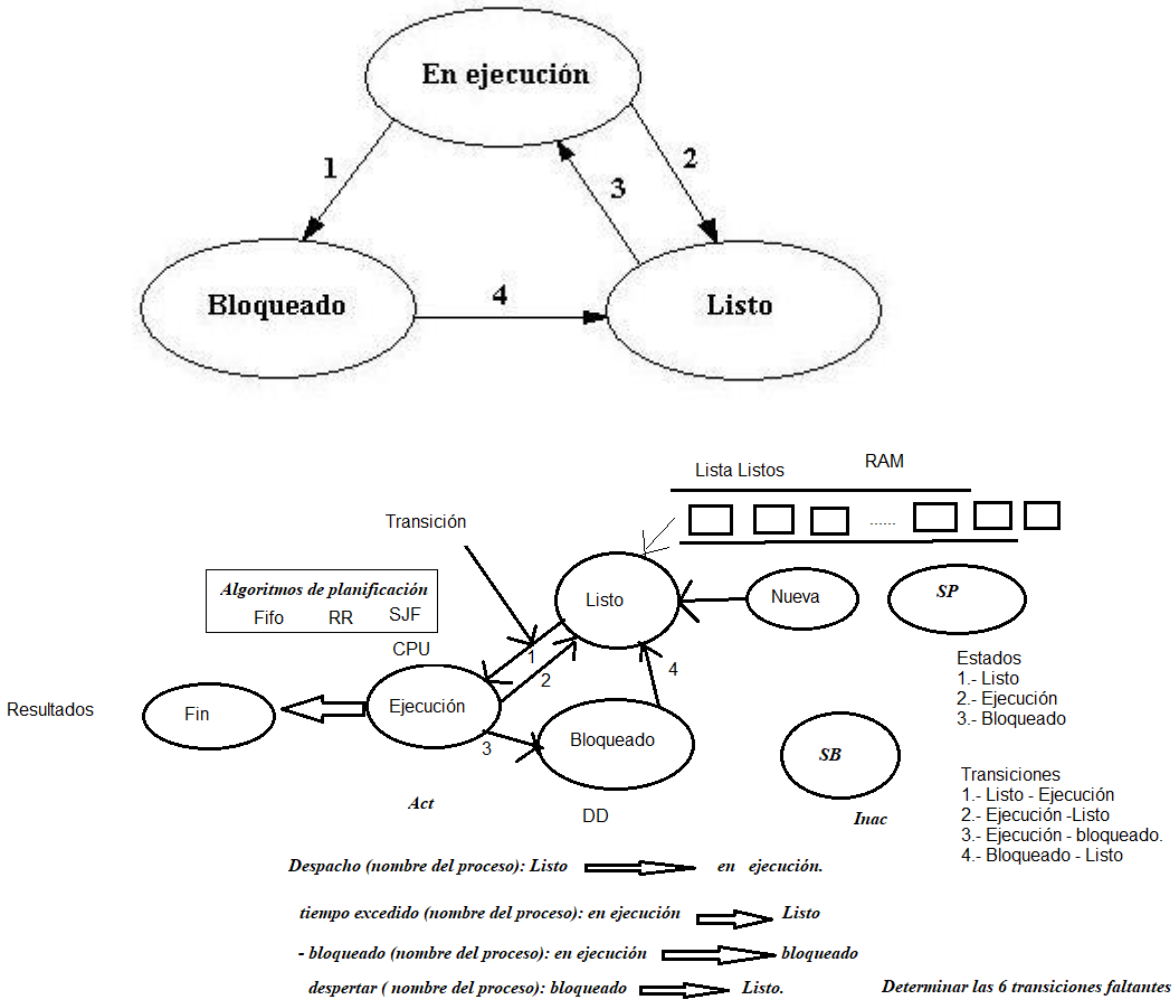

Figura # 7. Estados de los procesos.

Un proceso puede encontrarse en estado de ejecución, bloqueado o listo (que también se llama ejecutable).

De estos estados de los procesos se derivan las siguientes transiciones y estados:

Transición: El paso de un estado a otro.

Transiciones:

1. El proceso se bloquea en la entrada.
2. El planificador elige otro proceso.
3. El planificador elige este proceso.
4. La entrada se vuelve disponible.

Estados:

1. **Ejecución** (que en realidad hace uso del CPU en ese instante).
2. **Bloqueado** (incapaz de correr hasta que suceda algún evento externo.
3. **Listo** (ejecutable; se detiene temporalmente para permitir que se ejecute otro proceso).

En estos tres estados son posibles cuatro transiciones:

1. Ocurre cuando un proceso descubre que no puede continuar. En algún sistema el proceso debe ejecutar una llamada al sistema, BLOCK, para entrar en estado bloqueado.

2 y 3. Son ocasionadas por el planificador del proceso, que es parte del sistema operativo sin que el proceso llegue a saber de ella.

2. Ocurre cuando el planificador decide que el proceso en ejecución ya ha corrido el tiempo suficiente y es tiempo de permitir que otro proceso tome tiempo de CPU.

3. Ocurre cuando todos los procesos han utilizado su parte del tiempo y es hora de que el primer proceso vuelva a correr.

4. Ocurre cuando aparece el evento externo que estaba esperando un proceso (como el arribo de alguna entrada). Si ningún otro proceso corre en ese instante, la transición 3 se activará de inmediato y el proceso iniciará su ejecución, de lo contrario tendrá que esperar, en estado listo.

Transiciones de estado.

Todo proceso a lo largo de su existencia puede cambiar de estado varias veces. Cada uno de estos cambios se denomina transición de estado. Transiciones de estado de proceso, ver **figura # 8.**

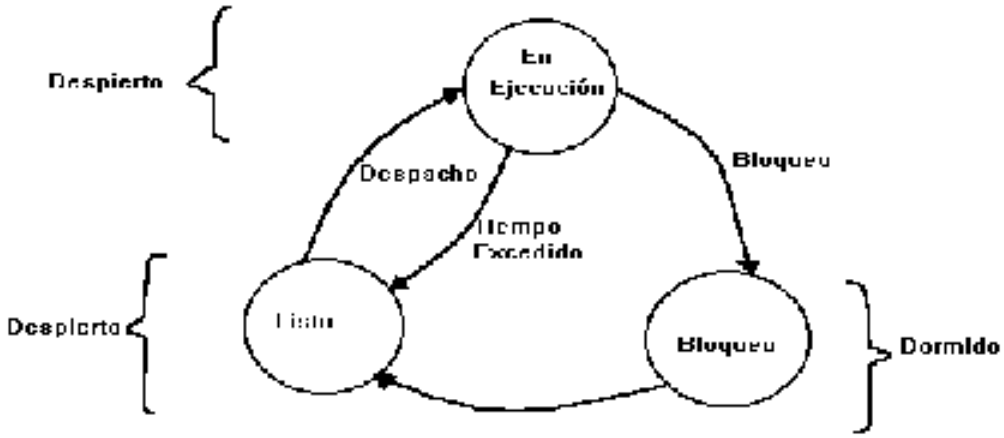

Figura # 8. Transiciones de estado.

La asignación del CPU al primer proceso de la lista de listos es llamada despacho, y es ejecutado por la entidad del sistema llamada despachador. Indicamos esta transición de la manera siguiente:

Despacho (nombre del proceso): Listo ⟶ en ejecución.

Mientras el proceso tenga CPU, se dice que está en ejecución. Para prevenir que cualquier proceso monopolice el sistema, ya sea de manera accidental o maliciosamente el sistema operativo ajusta un reloj de interrupción del hardware para permitir al usuario ejecutar su proceso durante un intervalo de tiempo específico o cuánto. Si el proceso no abandona voluntariamente el CPU, antes de que expire el intervalo, el reloj genera una interrupción, haciendo que el sistema operativo recupere el control. El sistema operativo hace que el proceso que anteriormente se hallaba en estado de ejecución pase al de listo, y hace que el primer proceso de la lista de listos pase al estado de ejecución.

Estas transiciones de estado se indican como:

- ***tiempo excedido (nombre del proceso): en ejecución ⟶ Listo***

- ***bloqueado (nombre del proceso): en ejecución ⟶ bloqueado***

El proceso cambia del estado bloqueado al estado listo:

- ***despertar (nombre del proceso): bloqueado ⟶ Listo.***

<u>Con esto tenemos definidas 4 transacciones de estado</u>.

- ***despacho (nombre del proceso): Listo ⟶ en ejecución***

- ***tiempo excedido (nombre del proceso): en ejecución ⟶ Listo***

- ***bloqueado (nombre del proceso): en ejecución ⟶ bloqueado***

- ***despertar (nombre del proceso): bloqueado ⟶ Listo.***

Los estados de los procesos se pueden dividir en dos tipos: **activos** e **inactivos.**

* **Estados activos.**

Son aquellos que compiten por el procesador o están en condiciones de hacerlo. Se dividen en, observe la **figura # 9.**

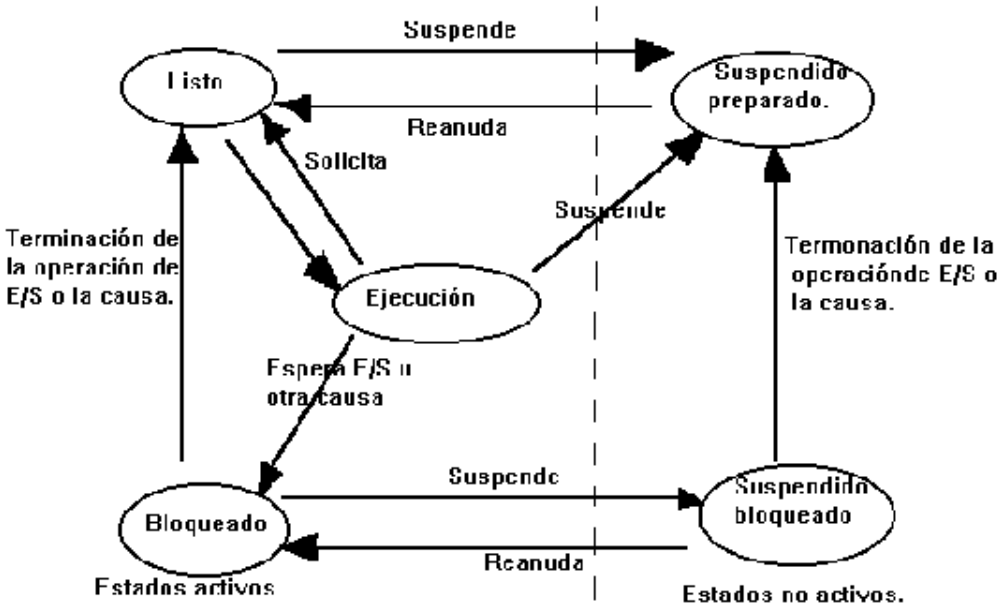

Figura # 9. Estados de un proceso y sus transiciones.

. Ejecución. Estado en el que se encuentra un proceso cuando tiene el control del procesador. En un sistema monoprocesador este estado sólo lo puede tener proceso.

. Listo. Aquellos procesos que están dispuestos para ser ejecutados, pero no están en ejecución por alguna causa (interrupción, haber entrado, en la cola estando otro proceso en ejecución, etc.

. Bloqueados. Son los procesos que no pueden ejecutarse de momento por necesitar algún recurso no disponible (generalmente recursos de E/S).

*** Estados inactivos.**

Son aquellos que no pueden competir por el procesador, pero que puedan volver a hacerlo por medio de ciertas operaciones. En estos estados se mantiene el bloque de control de proceso suspendido hasta que vuelva a ser activado.

Son de dos tipos:

Suspendido bloqueado.

Es el proceso que fue suspendido en espera de un evento, sin que hayan desaparecido las causas de su bloqueo.

Suspendido preparado.

Es el proceso que ha sido suspendido, pero no tiene causa para estar bloqueado.

Suspensión y Reanudación.

Un proceso suspendido no puede proseguir sino hasta que lo reanuda otro proceso. Reanudar (o activar) un proceso implica reiniciarlo a partir del punto en el que se suspendió.

Las operaciones de suspensión y reanudación son importantes por diversas razones:

* Si un sistema está funcionando mal y es probable que falle, se puede suspender los procesos activos para reanudarlos cuando se haya corregido el problema.

* Un usuario que desconfíe de los resultados parciales de un proceso puede suspenderlo (en vez de abortarlo) hasta que verifique si el proceso funciona correctamente o no.

* Algunos procesos se puede suspender como respuesta a las fluctuaciones a corto plazo de la carga del sistema y reanudarse cuando las cargas regresen a niveles normales.

*La **figura # 10**. Muestra los procesos con suspensión y reanudación.*

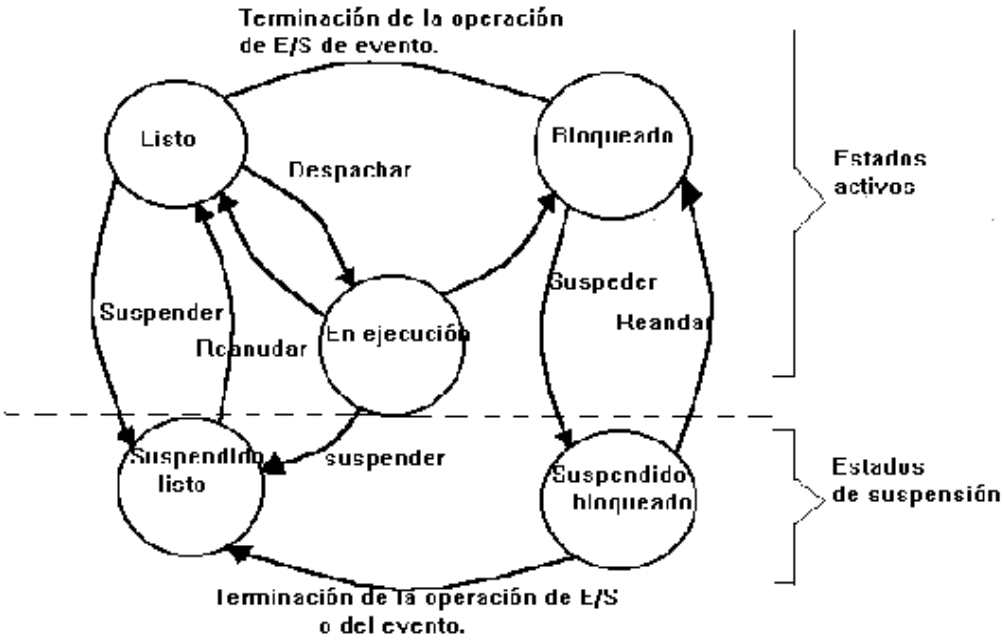

Figura # 10. Transiciones de estados de los procesos con suspensión y reanudación.

La **figura # 10**, muestra el diagrama de transiciones de estado de los procesos, modificado para incluir las operaciones de suspensión y reanudación. Se han añadido dos nuevos estados, denominados suspendido-listo y suspendido bloqueado; no hay necesidad de un estado suspendido-ejecutado. Sobre la línea discontinua se encuentran los estados activos, y debajo de ella los estados suspendidos.

Una suspensión puede ser iniciada por el propio proceso o por otro. En un sistema con un solo procesador el proceso en ejecución puede suspenderse a sí mismo; ningún otro proceso podría estar en ejecución al mismo tiempo para realizar la suspensión (aunque otro proceso sí podría solicitar la suspensión cuando se ejecute). En un sistema de múltiples procesadores, un proceso en ejecución puede suspender a otro que se esté ejecutando en ese mismo momento en un procesador diferente.

Solamente otro proceso puede suspender un proceso listo. La transición correspondiente es:

1) Suspender (nombre_del_proceso): Listo ⟶ Suspendido-Listo.

Un proceso puede hacer que otro proceso que se encuentre en el estado suspendido-listo pase al estado listo. La transición correspondiente es:

2) Reanudar (nombre_del_proceso): Suspendido-Listo ⟶ Listo.

Un proceso puede suspender a otro proceso que esté bloqueado. La transición correspondiente es:

3) Suspende (nombre_del_proceso): Bloqueado ⟶ Suspendido-Bloqueado.

Un proceso puede reanudar otro proceso que esté suspendido-bloqueado. La transición correspondiente es:

4) Reanudar(nombre_del_proceso): Suspendido-Bloqueado ⟶ Bloqueado.

Como la suspensión es por lo general una actividad de alta prioridad, se debe realizar de inmediato. Cuando se presenta finalmente el término de la operación (si es que termina), el proceso suspendido-bloqueado realiza la siguiente transición.

5) Completar (nombre del _proceso): suspendido-bloqueado ⟶ suspendido listo.

6) Suspender (nombre_del_proceso): Ejecución ⟶ Suspendido-Listo.

En conclusión, los sistemas que administran procesos deben ser capaces de realizar ciertas operaciones sobre procesos y con ellos. Tales operaciones incluyen:

- Crear un proceso.
- Destruir un proceso.
- Suspender un proceso.
- Reanudar un proceso.
- Cambiar la prioridad de un proceso.
- Bloquear un proceso.
- Despertar un proceso.

- Despachar un proceso.
- Permitir que un proceso se comunique con otro (esto se denomina comunicación entre procesos).

Crear un proceso implica operaciones como:

- Dar un nombre a un proceso.
- Insertarlo en la lista de procesos conocidos del sistema (o tabla de procesos)
- Determinar la prioridad inicial de proceso.
- Crear el bloque de control de proceso.
- Asignar los recursos iniciales al proceso.

Un proceso puede crear un nuevo proceso. Si lo hace el proceso creador se denomina proceso padre, y el proceso creado, proceso hijo. Sólo se necesita un padre para crear un hijo. Tal creación origina una estructura jerárquica de procesos. No se puede destruir un proceso cuando este ha creado otros procesos.

Destruir un proceso implica eliminarlo del sistema. Se le remueve de la tabla o listas del sistema, sus recursos se devuelven al sistema y su bloque de control de proceso se borra (es decir, el espacio de memoria ocupado por su PCB se devuelve al espacio de memoria disponible.

2.3 Procesos ligeros: Hilos o hebras.

Un “hilo” es una unidad básica de utilización del CPU y tiene poco estado no compartido. Un grupo de hilos semejantes comparten código, espacio de direcciones y recursos del sistema operativo. El entorno en el cual se ejecuta un hilo se llama tarea. Un proceso tradicional (pesado) equivale a una tarea con un hilo. *Una tarea no hace nada sino contiene hilos y un hilo debe encontrarse exactamente en una tarea*. Un hilo posee por lo menos su propio estado de registro y por lo general su propia pila. Como el compartimiento

es frecuente el costo de conmutación del CPU entre hilos del mismo grupo y el de creación de hilos se reduce en comparación con el cambio de contexto entre procesos pesados. De esta manera, una solución razonable para el problema de cómo un servidor puede manejar eficientemente varias solicitudes es bloquear un hilo y pasar a otro.

Un hilo tiene un identificador (ID), una pila, una prioridad de ejecución y una dirección de inicio de la ejecución. Los hilos de un proceso comparten todo el espacio de direcciones de ese proceso; pueden modificar variables globales, acceder a descriptores de archivos abiertos o interferirse mutuamente de otras maneras.

Se dice que un hilo es dinámico si se puede crear en cualquier instante durante la ejecución de un proceso y si no es necesario especificar por adelantado el número de hilos.

2.4 Concurrencia y secuencialidad.

Los procesos del sistema pueden ejecutarse concurrentemente, puede haber múltiples tareas en el CPU con varios procesos. Existen varias razones para permitir la ejecución concurrente:

* ***Compartir recursos físicos***.

Ya que los recursos del hardware de la computadora son limitados, nos podemos ver obligados a compartirlos en un entorno multiusuario.

*** Compartir recursos lógicos**.

Puesto que varios usuarios pueden interesarse en el mismo elemento de información (por ejemplo, un archivo compartido), debemos proporcionar un entorno que permita el acceso concurrente a estos tipos de recursos.

**** Acelerar los cálculos.***

Si queremos que una tarea se ejecute con mayor rapidez, debemos dividirla en subtareas, cada una de las cuales se ejecutara, en paralelo con las demás.

*** Modularidad.**

Podremos construir el sistema en forma modular, dividiendo las funciones del sistema en procesos separados.

**** Comodidad***.

Un usuario puede tener que ejecutar varias tareas a la vez, por ejemplo, puede editar, imprimir y compilar en paralelo.

La ejecución concurrente que requiere la cooperación entre procesos necesita un mecanismo para la sincronización y comunicación de procesos, exclusión mutua y sincronización.

2.4.1 Exclusión mutua de secciones críticas.

Exclusión Mutua.

Consiste en garantizar que durante la ejecución de una región crítica los recursos compartidos solo se asignen a uno y solo a uno de los procesos.

Si un recurso compartido es una variable, la exclusión mutua asegura que a lo más un proceso a la vez ha accedido a ella durante la actualización crítica que conduce a valores temporalmente inestables. Consecuentemente los otros procesos ven solamente valores estables de las variables compartidas. Con los dispositivos compartidos la necesidad para la exclusión mutua puede ser incluso más obvia cuando uno considera el problema que puede provocar sus usos.

Por ejemplo, supongamos que en el sistema existe un archivo formado por registros compuestos por 5 campos, ***figura # 11:***

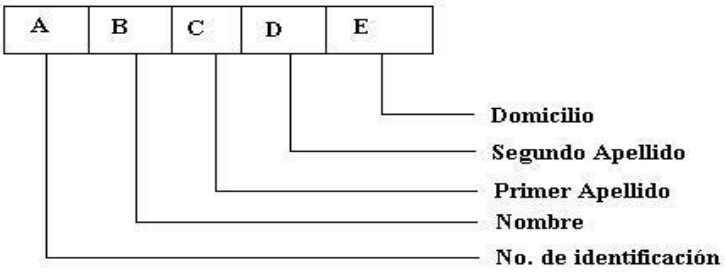

Figura # 11. **Registro de 5 campos.**

Para que un registro sea válido debe estar totalmente actualizado, es decir, si se modifica el valor del campo A, el resto de los campos deben ser coherentes con el nuevo valor de dicho campo, ya que de otro modo el registro sería inconsistente.

Si en el momento en que un proceso escribe o modifica un registro y existe otro proceso que realiza la lectura de ese mismo registro, y el primero de ellos sólo hubiese tenido tiempo de modificar el campo A, la información obtenida por el segundo proceso seria inconsistente. Para evitar esto se deben sincronizar los procesos de manera que mientras uno escribe, otro pueda leer.

Esto no ocurre en aquellos casos en que dos procesos tratan de leer en el mismo archivo, aun coincidiendo en el mismo registro.

Generalizando. Supongamos que los dos procesos que coinciden en el mismo registró, uno está escribiendo y el otro leyendo, llamados ESCRIBIR y LEER, se encuentran en un sistema monoprocesador multiprogramado y son los únicos presentes en ese momento, ver ***figura # 12.***

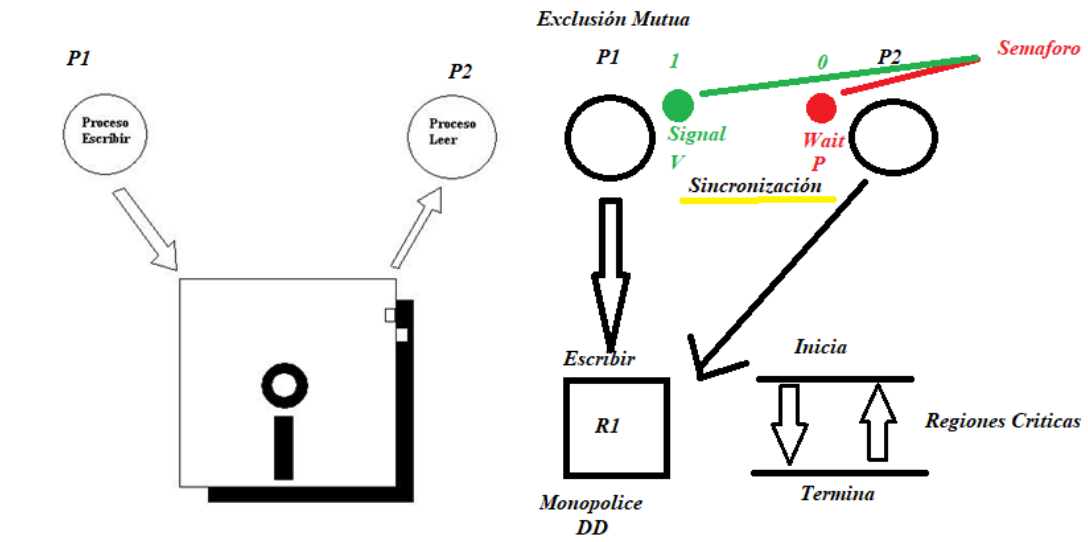

Figura #12. **Concurrencia.**

En el momento de un cambio de proceso del uno al otro se pueden producir las siguientes situaciones:

. **Sin sincronización entre procesos**. Puede darse el caso de que ESCRIBIR esté actualizando un registro y se quede a medías, sorprendiéndole el cambio de proceso, por tanto, terminará de escribirlo cuando vuelva a hacer uso del procesador. Con el cambio le tocará el turno al proceso LEER, que accederá a dicho registro pudiendo leerlo completamente. Es evidente que los datos leídos serán inconsistentes.

. **Con sincronización entre procesos**. Supongamos algún mecanismo que prohíba la lectura (bloqueo de registros) a cualquier proceso, mientras el proceso ESCRIBIR esté realizando alguna operación. En este caso, LEER, al hacer uso del procesador que se encuentra bloqueado, quedaría en espera de que el registro quede totalmente escrito y se proceda a su desbloqueo, LEER pasaría a estado bloqueado, ESCRIBIR terminaría su trabajo sobre el registro y en el siguiente cambio LEER procedería a hacer el suyo.

Esta sincronización por la cual una actividad impide que otras puedan tener acceso a un dato mientras se encuentra realizando una operación sobre el mismo es lo que se conoce como exclusión mutua.

La zona de código de un proceso que no puede ser interrumpida por otro, por los motivos expuestos anteriormente se le llama Región Crítica.

2.4.2 Sincronización de procesos en Secciones Criticas.

En procesos concurrentes, la ejecución de un proceso se desarrolla en forma asíncrona respecto a los otros. Sin embargo, cuando dos, o más procesos necesitan entrar en región crítica, es necesario que exista una ***sincronización*** entre ellos a fin de garantizar que al menos uno y solo un proceso entrará en región crítica.

Si una actividad desea impedir que otra acceda a ciertos datos compartidos, mientras no se cumpla una determinada condición, debemos sincronizar las actividades con dicha condición. Por tanto, la sincronización es un elemento necesario para asegurar la exclusión mutua.

2.4.2.1 Mecanismo de semáforos.

Un **semáforo** es una variable entera que constituye el método clásico para restringir o permitir el acceso a recursos compartidos. Fueron inventados por Edsger Dijkstra y se usaron por primera vez en el sistema operativo THEOS. Sólo puede accederse mediante dos operaciones ***atómicas*** comunes: espera (wait) y señal (signal). (Originalmente, a estas instrucciones se las denominó P (para espera) y V (para señal).

Operaciones:

Los semáforos sólo pueden ser manipulados usando las siguientes operaciones:

```
Inicia(Semáforo s, Entero v)
{
  s = v;
}
P(Semáforo s)
{
  while (s == 0);  /* espera hasta que s > 0 */
  s = s-1;
}
V(Semáforo s)
{
  s = s+1;
}
```

Los nombres de estas funciones, **V** y **P**, tienen su origen en el idioma holandés. "Verhogen" significa incrementar y "Proberen" probar, aunque Dijkstra usó la palabra inventada *prolaag* [1], que es una combinación de ***probeer te verlagen*** (intentar decrementar). El valor del semáforo es el número de unidades del recurso que están disponibles (si sólo hay un recurso, se utiliza un "semáforo binario" con los valores 0 y 1).

Inicia se utiliza para inicializar el semáforo antes de que se hagan peticiones sobre él, y toma por argumento a un entero. La operación *P* cuando no hay un recurso disponible, detiene la ejecución quedando en espera activa (o durmiendo) hasta que el valor del semáforo sea positivo, en cuyo caso lo reclama inmediatamente decrementándolo. *V* es la operación inversa: hace disponible un recurso después de que el proceso ha terminado de usarlo. Las operaciones *P* y *V* han de ser indivisibles (o atómicas), lo que quiere decir que cada una de las operaciones no debe ser interrumpida en medio de su ejecución.

La operación *V* es denominada a veces *subir* el semáforo (*up*) y la operación *P* se conoce también como *bajar* el semáforo (*down*), y también son llamadas *signal* y *wait* o *soltar* y *tomar*.

Para evitar la espera activa, un semáforo puede tener asociada una cola de procesos (normalmente una cola FIFO). Si un proceso efectúa una operación *P* en un semáforo que tiene valor cero, el proceso es detenido y añadido a la cola del semáforo. Cuando otro proceso incrementa el semáforo mediante la operación *V* y hay procesos en la cola asociada, se extrae uno de ellos (el primero que entró en una cola FIFO) y se reanuda su ejecución.

Usos

Los semáforos se emplean para permitir el acceso a diferentes partes de programas (llamados **secciones críticas**) donde se manipulan variables o recursos que deben ser accedidos de forma especial. Según el valor con que son inicializados se permiten a más o menos procesos utilizar el recurso de forma simultánea.

Un tipo simple de semáforo es el **binario**, que puede tomar solamente los valores 0 y 1. Se inicializan en 1 y son usados cuando sólo un proceso puede acceder a un recurso a la vez. Son esencialmente lo mismo que los mutex. Cuando el recurso está disponible, un proceso accede y decrementa el valor del semáforo con la operación P. El valor queda entonces en 0, lo que hace que si otro proceso intenta decrementarlo tenga que esperar. Cuando el proceso que decrementó el semáforo realiza una operación *V*, algún proceso que estaba esperando puede despertar y seguir ejecutando.

Para hacer que dos procesos se ejecuten en una secuencia predeterminada puede usarse un semáforo inicializado en 0. El proceso que debe ejecutar primero en la secuencia realiza la operación *V* sobre el semáforo antes del código que debe ser ejecutado después del otro proceso. Éste ejecuta la operación *P*. Si el segundo proceso en la secuencia es programado para ejecutar antes que el otro, al hacer *P* dormirá hasta que el primer proceso de la secuencia pase por su operación *V*. Este modo de uso se denomina señalación (*signaling*), y se usa para que un proceso o hilo de ejecución le haga saber a otro que algo ha sucedido.

Ejemplo de uso:

Los semáforos pueden ser usados para diferentes propósitos, entre ellos:

- Implementar cierres de exclusión mutua o locks
- Barreras
- Permitir a un máximo de N threads acceder a un recurso, inicializando el semáforo en N
- Notificación. Inicializando el semáforo en 0 puede usarse para comunicación entre threads sobre la disponibilidad de un recurso

En el siguiente ejemplo se crean y ejecutan n procesos que intentarán entrar en su sección crítica cada vez que puedan, y lo lograrán siempre de a uno por vez, gracias al uso del semáforo **S** inicializado en **1**. El mismo tiene la misma función que un lock.

```
const int n    /* número de procesos */
Inicia (s,1)   /* Inicializa un semáforo con nombre s con valor 1 */

void P (int i)
{
   while (cierto)
   {
      P(s)
      /* Sección crítica */
      V(s)
      /* resto del proceso */
   }
}

void main()
{
Comenzar-procesos(P(1), P(2), ... ,P(n));
}
```

2.4.2.2 Mecanismo de monitores.

Los **monitores** son estructuras de datos utilizadas en lenguajes de programación para sincronizar dos o más procesos o hilos de ejecución que usan recursos compartidos.

En el estudio y uso de los semáforos se puede ver que las llamadas a las funciones necesarias para utilizarlos quedan repartidas en el código del programa, haciendo difícil corregir errores y asegurar el buen funcionamiento de los algoritmos. Para evitar estos inconvenientes se desarrollaron los monitores. El concepto de monitor fue definido por primera vez por Charles Antony Richard Hoare en un artículo del año 1974. La estructura de los monitores se ha implementado en varios lenguajes de programación, incluido Pascal concurrente, Modula-2, Modula-3 y Java, y como biblioteca de programas.

Un monitor tiene cuatro componentes: inicialización, datos privados, procedimientos del monitor y cola de entrada.

- Inicialización: contiene el código a ser ejecutado cuando el monitor es creado
- Datos privados: contiene los procedimientos privados, que sólo pueden ser usados desde *dentro* del monitor y no son visibles desde fuera
- Procedimientos del monitor: son los procedimientos que pueden ser llamados desde *fuera* del monitor.
- Cola de entrada: contiene a los threads que han llamado a algún procedimiento del monitor, pero no han podido adquirir permiso para ejecutarlos aún.

Un monitor es un conjunto de procedimientos y estructuras de datos que puede ser compartidos por varios procesos al mismo tiempo, pero solo pueden ser usado por uno solo a la vez.

Conceptualmente, un monitor se puede ver como una región crítica con puertos de entrada principales y puertos secundarios (las variables de condición del monitor).

Propiedades:

§ El monitor está formado por dos partes: declaraciones y cuerpo.

- Los datos locales a un monitor, solo pueden ser acezados y modificados desde el mismo monitor. Cualquier acceso externo es negado.
- Un solo proceso puede entrar a la vez a un monitor para ejecutar un procedimiento del mismo y cualquier llamada se mantendrá en espera.
- Las variables locales al monitor, son inaccesibles desde afuera. Esto garantiza la exclusión mutua entre las mismas.

Sintaxis.

<nombre del monitor>: monitor;

Begin

<declaraciones de datos locales del monitor>

PROCEDURE <nombre de procedimiento> (parámetros)

Begin

<cuerpo del procedimiento>

End;

Declaraciones de otros procedimientos....

Inicialización de las variables.

End.

Los monitores permiten la sincronización en el uso de los recursos, a través de dos operaciones:

Wait:

Al ser ejecutada por el monitor hace que el proceso que ejecutó la llamada, sea colocado en estado de espera y abandone el monitor.

Signal:

Al ser ejecutada desde el monitor, permite que uno de los procesos que este en espera (por una operación wait previa) pueda ser reasumido en el punto previo donde fue detenido.

Exclusión mutua en un monitor

Los monitores están pensados para ser usados en entornos multiproceso o multihilo, y por lo tanto muchos procesos o threads pueden llamar a la vez a un procedimiento del monitor. Los monitores garantizan que, en cualquier momento, a lo sumo un thread puede estar ejecutando *dentro* de un monitor. Ejecutar dentro de un monitor significa que sólo un thread estará en estado de ejecución mientras dura la llamada a un procedimiento del monitor. El problema de que dos threads ejecuten un mismo procedimiento dentro del monitor es que se pueden dar condiciones de carrera, perjudicando el resultado de los cálculos. Para evitar esto y garantizar la integridad de los datos privados, el monitor hace cumplir la exclusión mutua **implícitamente**, de modo que sólo un procedimiento esté siendo ejecutado a la vez. De esta forma, si un thread llama a un procedimiento mientras otro thread está dentro del monitor, se bloqueará y esperará en la cola de entrada hasta que el monitor quede nuevamente libre. Aunque se la llama cola de entrada, no debería suponerse ninguna política de encolado.

Para que resulten útiles en un entorno de concurrencia, los monitores deben incluir algún tipo de forma de sincronización. Por ejemplo, supóngase un thread que está dentro del monitor y necesita que se cumpla una condición para poder continuar la ejecución. En ese caso, se debe contar con un mecanismo de bloqueo del thread, a la vez que se debe liberar el monitor para ser usado por otro hilo. Más tarde, cuando la condición permita al thread bloqueado continuar ejecutando, debe poder ingresar en el monitor en el mismo lugar donde fue suspendido. Para esto los monitores poseen **variables de condición** que son accesibles sólo desde adentro. Existen dos funciones para operar con las variables de condición:

- cond_wait(c): suspende la ejecución del proceso que la llama con la condición *c*. El monitor se convierte en el dueño del lock y queda disponible para que otro proceso pueda entrar
- cond_signal(c): reanuda la ejecución de algún proceso suspendido con *cond_wait* bajo la misma condición. Si hay varios procesos con esas características elige uno. Si no hay ninguno, no hace nada.

Nótese que, al contrario que los semáforos, la llamada a *cond_signal(c)* se pierde si no hay tareas esperando en la variable de condición *c*.

Las variables de condición indican eventos, y no poseen ningún valor. Si un thread tiene que esperar que ocurra un evento, se dice espera por (o en) la variable de condición correspondiente. Si otro thread provoca un evento, simplemente utiliza la función *cond_signal* con esa condición como parámetro. De este modo, cada variable de condición tiene una cola asociada para los threads que están esperando que ocurra el evento correspondiente. Las colas se ubican en el sector de datos privados visto anteriormente.

La política de inserción de procesos en las colas de las variables condición es la FIFO, ya que asegura que ningún proceso caiga en la espera indefinida, cosa que sí ocurre con la política LIFO (puede que los procesos de la base de la pila nunca sean despertados) o con una política en la que se desbloquea a un proceso aleatorio.

2.4.3 Interbloqueo (DeadLock).

En sistemas operativos, el **bloqueo mutuo** (también conocido como interbloqueo, traba mortal, *deadlock*, abrazo mortal) es el bloqueo permanente de un conjunto de procesos o hilos de ejecución en un sistema concurrente que compiten por recursos del sistema o bien se comunican entre ellos. A diferencia de otros problemas de concurrencia de procesos, no existe una solución general para los interbloqueos.

Todos los interbloqueos surgen de necesidades que no pueden ser satisfechas, por parte de dos o más procesos. En la vida real, un ejemplo puede ser el de cuatro vehículos que se encuentran en una intersección en el mismo momento. Cada uno necesita que otro se mueva para poder continuar su camino, y ninguno puede continuar. Los recursos compartidos en este caso son los cuatro cuadrantes. El vehículo que se dirige de oeste a este, por ejemplo, necesita de los cuadrantes suroeste y sureste. ***Ver figura # 13.***

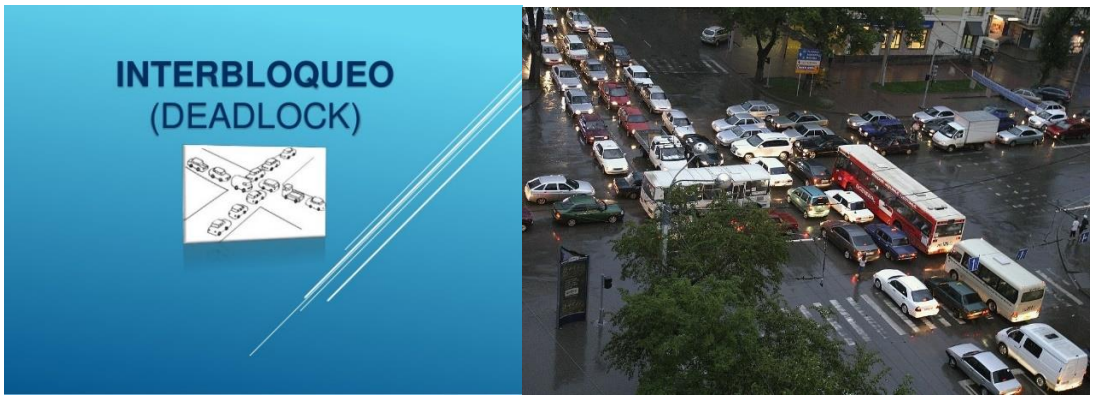

Figura # 13. Ejemplo de un Interbloqueo.

En el siguiente ejemplo, dos procesos compiten por dos recursos que necesitan para funcionar, que sólo pueden ser utilizados por un proceso a la vez. El primer proceso obtiene el permiso de utilizar uno de los recursos (adquiere el *lock* sobre ese recurso). El segundo proceso toma el *lock* del otro recurso, y luego intenta utilizar el recurso ya utilizado por el primer proceso, por lo tanto, queda en espera. Cuando el primer proceso a su vez intenta utilizar el otro recurso, se produce un interbloqueo, donde los dos procesos esperan la liberación del recurso que utiliza el otro proceso, ver ***Figura #14.***

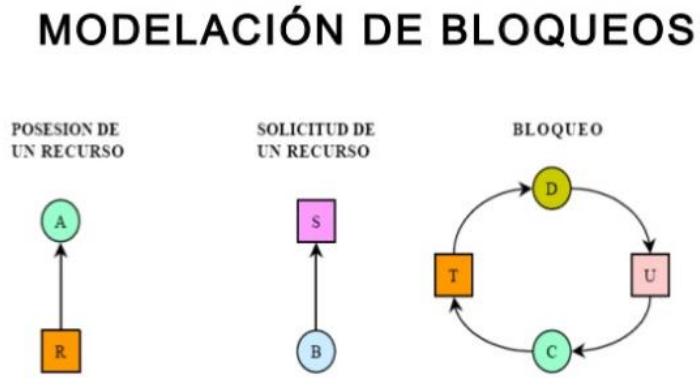

Figura #14. Modelado del Interbloqueo.

Representación del Interbloqueo mediante grafos.

El Bloqueo mutuo también puede ser representado usando grafos dirigidos, donde el proceso es representado por un círculo y el recurso, por un cuadrado. Cuando un proceso solicita un recurso, una flecha es dirigida del círculo al cuadrado. Cuando un recurso es asignado a un proceso, una flecha es dirigida del cuadrado al círculo, ver ***figura #15.***

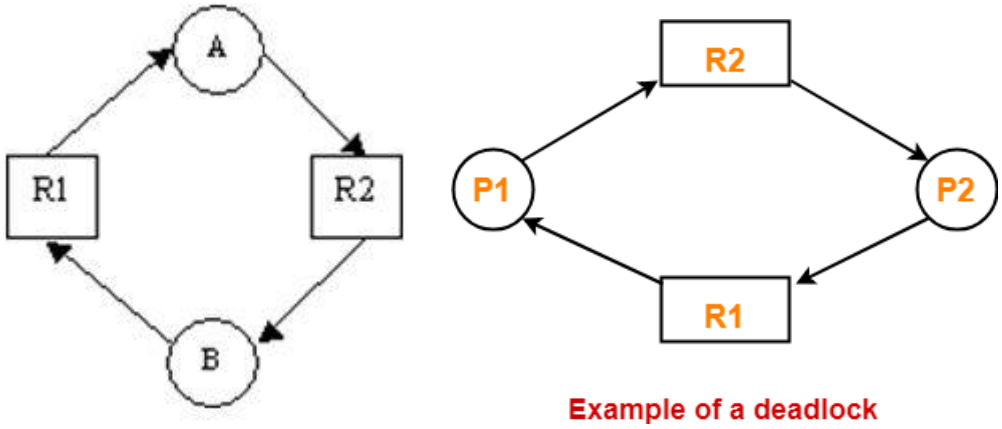

Figura # 15. Ejemplo de representación de Bloqueo Mutuo en grafos de procesos y recursos con dos procesos A y B, y dos recursos R1 y R2.

En la ***figura # 15*** del ejemplo, se pueden ver dos procesos diferentes (**A** y **B**), cada uno con un recurso diferente asignado (**R1** y **R2**). En este ejemplo clásico de bloqueo mutuo, es fácilmente visible la condición de **espera circular** en la que los procesos se encuentran, donde cada uno solicita un recurso que está asignado a otro proceso.

Condiciones necesarias para que ocurra un Interbloqueo.

También conocidas como *condiciones de Coffman* por su primera descripción en 1971 en un artículo escrito por E.G.Coffman.

Estas condiciones deben cumplirse simultáneamente y no son totalmente independientes una de otra.

Sean los procesos *Po, P1,.. Pn* y los recursos *Ro, R1,..., Rm*:

- ***Condición de exclusión mutua***: Existencia al menos de un recurso compartido por los procesos, al cual sólo puede acceder uno simultáneamente.

- ***Condición de Posesión y espera***: Al menos un proceso *Pi* ha adquirido un recurso *Ri*, y lo mantiene mientras espera al menos un recurso *Rj* que ya ha sido asignado a otro proceso.

- ***Condición de no expropiación***: Los recursos no pueden ser apropiados por los procesos, es decir, los recursos sólo podrán ser liberados voluntariamente por sus propietarios.

- ***Condición de espera circular***: Dado el conjunto de procesos *P0...Pn*, *P0* está esperando un recurso adquirido por *P1*, que está esperando un recurso adquirido por *P2*, que...,que está esperando un recurso adquirido por *Pn*, que está esperando un recurso adquirido por *P0*. Esta condición implica la condición de retención y espera.

2.4.3.1 Prevención.

La estrategia básica de la prevención del interbloqueo consiste, a grandes rasgos, en diseñar su sistema de manera que esté excluida, a priori, la posibilidad de interbloqueo. Ver ***Figura #16.***

Figura #16. Métodos para prevenir los Interbloqueos.

Los métodos para prevenir el interbloqueo son de dos tipos:

- Los métodos indirectos que consisten en impedir la aparición de alguna de las tres condiciones necesarias para que se de el interbloqeo.
- Los métodos directos que consisten en evitar la aparición del círculo vicioso de espera.

Exclusión mutua. Si ningún recurso se puede asignar de forma exclusiva, no se producirá interbloqueo. Sin embargo, existen recursos para los que no es posible negar la condición de exclusión mutua. No obstante, es posible eliminar esta condición en algunos procesos. Por ejemplo, una impresora es un recurso no compatible pues si se permite que dos procesos escriban en la impresora al mismo tiempo, la salida resulta caótica. Pero con el spooling de salida varios procesos pueden generar salida al mismo tiempo. Puesto que el spooler nunca solicita otros recursos, se elimina el bloqueo originado por la impresora.

El inconveniente es que no todos los recursos pueden usarse de esta forma (por ejemplo, la tabla de procesos no se presenta al spooling y, además, la implementación de esta técnica puede introducir nuevos motivos de interbloqueo, ya que el spooling emplea una zona de disco finita)

Retención y espera La condición de retención y espera puede prevenirse exigiendo que todos los procesos soliciten todos los recursos que necesiten a un mismo tiempo y bloqueando el proceso hasta que todos los recursos puedan concederse simultáneamente. Esta solución resulta ineficiente por dos factores:

- En primer lugar, un proceso puede estar suspendido durante mucho tiempo, esperando que concedan todas sus solicitudes de recursos, cuando de hecho podría haber avanzado con solo algunos de los recursos.

- Y, en segundo lugar, los recursos asignados a un proceso pueden permanecer sin usarse durante periodos considerables, tiempo durante el cual se priva del acceso a otros procesos.

No apropiación La condición de no apropiación puede prevenirse de varias formas. Primero, si a un proceso que retiene ciertos recursos se le deniega una nueva solicitud, dicho proceso deberá liberar sus recursos anteriores y solicitarlos de nuevo, cuando sea necesario, junto con el recurso adicional. Por otra parte, si un proceso solicita un recurso que actualmente está retenido por otro proceso, el sistema operativo debe expulsar al segundo proceso y exigirle que libere sus recursos. Este último esquema evitará el interbloqueo sólo si no hay dos procesos que posean la misma prioridad.

Esta técnica es práctica sólo cuando se aplica a recursos cuyo estado puede salvarse y restaurarse más tarde de una forma fácil, como es el caso de un procesador.

Circulo vicioso de espera La condición del circulo vicioso de espera puede prevenirse definiendo una ordenación lineal de los tipos de recursos. Si a un proceso se le han asignado recursos de tipo R, entonces sólo podrá realizar peticiones posteriores sobre los recursos de los tipos siguientes a R en la ordenación.

Para comprobar el funcionamiento de esta estrategia, se asocia un índice a cada tipo de recurso. En tal caso, el recurso Ri antecede a Rj en la ordenación si i<j. Entonces, supóngase que dos procesos A y B, están interbloqueados, porque A ha adquirido Ri y solicitado Rj, mientras que B ha adquirido Rj y solicitado Ri. Esta condición es imposible porque implica que i<j y j<i.

Como en la retención y espera, la prevención del circulo vicioso de espera puede ser ineficiente, retardando procesos y denegando accesos a recursos innecesariamente.

2.4.3.2 Detección.

Las estrategias de prevención de interbloqueo son muy conservadoras; resuelven el problema limitando el acceso a recursos e imponiendo restricciones sobre los procesos. En cambio, las estrategias de detección de interbloqueo, no limitan el acceso a recursos ni restringen las acciones del proceso. Con la detección del interbloqueo, se concederán los

recursos que los procesos necesiten siempre que sea posible. Periódicamente, el S. O. ejecuta un algoritmo que permite detectar la condición de círculo vicioso de espera.

La detección del interbloqueo es el proceso de determinar si realmente existe un interbloqueo e identificar los procesos y recursos implicados en él. Una posibilidad detectar un interbloqueo es monitorear cada cierto tiempo el estado de los recursos. Cada vez que se solicita o se devuelve un recurso, se actualiza el estado de los recursos y se hace una verificación para observar si existe algún ciclo.

Este método está basado en suponer que un interbloqueo no se presente y que los recursos del sistema que han sido asignados, se liberarán en el momento que otro proceso lo requiera.

Algoritmo de detección del interbloqueo Una comprobación para interbloqueo puede hacerse con igual o menor frecuencia que cada solicitud de recursos, dependiendo de qué tan probable es que ocurra un interbloqueo. Comprobar cada solicitud de recursos tiene dos ventajas: Conduce a la detección temprana y el algoritmo es simple, de manera relativa porque se basa en cambios crecientes al estado del sistema. Además, las comprobaciones frecuentes consumen un tiempo considerable de procesador.

Los algoritmos de detección de bloqueos implican cierta sobrecarga en tiempo de ejecución:

surge el siguiente interrogante:

¿compensa la sobrecarga implícita en los algoritmos de detección de bloqueos, el ahorro potencial de localizarlos y romperlos?

El empleo de algoritmos de detección de interbloqueo implica cierto gasto extra durante la ejecución. Así pues, se presenta de nuevo la cuestión de costeabilidad, tan habitual en los sistemas operativos. Los algoritmos de detección de interbloqueo determinan por lo general si existe una espera circular.

2.4.3.3 Recuperación.

Cuando se ha detectado que existe un interbloqueo, podemos actuar de varias formas. Una posibilidad es informar al operador que ha ocurrido un interbloqueo y dejar que el operador se ocupe de él manualmente. La otra posibilidad es dejar que el sistema se recupere automáticamente del interbloqueo. Dentro de esta recuperación automática tenemos dos opciones para romper el interbloqueo: Una consiste en abortar uno o más procesos hasta romper la espera circular, y la segunda es apropiar algunos recursos de uno o más de los procesos bloqueados.

La recuperación después de un interbloqueo se complica porque puede no estar claro que el sistema se haya bloqueado. Las mayorías de los Sistemas Operativos no tienen los medios suficientes para suspender un proceso, eliminarlo del sistema y reanudarlo más tarde.

Actualmente, la recuperación se suele realizar eliminando un proceso y quitándole sus recursos. El proceso eliminado se pierde, pero gracias a esto ahora es posible terminar. Algunas veces es necesario, eliminar varios procesos hasta que se hayan liberado los recursos necesarios para que terminen los procesos restantes.

Los procesos pueden eliminarse de acuerdo con algún orden de prioridad, aunque es posible que no existan prioridades entre los procesos bloqueados, de modo que el operador necesita tomar una decisión arbitraria para decidir que procesos se eliminarán.

Recuperación Manual

Está forma de recuperación consiste en avisarle al administrador o al operador del sistema que se ha presentado un interbloqueo, y será el administrador el que solucione dicho problema de la manera más conveniente posible, de modo que su decisión no afecte demasiado a al usuario del proceso en conflicto, y sobre todo que no afecte a los demás usuarios del sistema.

Abortar los Procesos

Para eliminar interbloqueos abortando un proceso, tenemos dos métodos; en ambos, el sistema recupera todos los recursos asignados a los procesos terminados.

1) Abortar todos los procesos interbloqueados. Esta es una de las soluciones más comunes, adoptada por Sistemas Operativos. Este método romperá definitivamente el ciclo de interbloqueo, pero con un costo muy elevado, ya que estos procesos efectuaron cálculos durante mucho tiempo y habrá que descartar los resultados de estos cálculos parciales, para quizá tener que volver a calcularlos más tarde.

2) Abortar un proceso en cada ocasión hasta eliminar el ciclo de interbloqueo. El orden en que se seleccionan los procesos para abortarlos debe basarse en algún criterio de costo mínimo. Después de cada aborto, debe solicitarse de nuevo el algoritmo de detección, para ver si todavía existe el interbloqueo. Este método cae en mucho tiempo de procesamiento adicional.

Quizá no sea fácil abortar un proceso. Si éste se encuentra actualizando un archivo, cortarlo a la mitad de la operación puede ocasionar que el archivo quede en un mal estado.

Si se utiliza el método de terminación parcial, entonces, dado un conjunto de procesos bloqueados, debemos determinar cuál proceso o procesos debe terminarse para intentar romper el interbloqueo. Se trata sobre todo de una cuestión económica, debemos abortar los procesos que nos representen el menor costo posible. Existen muchos factores que determinan el proceso que se seleccionará, siendo los principales los siguientes:

La prioridad del proceso. Se elimina el proceso de menor prioridad. Tiempo de procesador usado. Se abortará aquel proceso que haya utilizado menos tiempo el procesador, ya que se pierde menos trabajo y será más fácil recuperarlo más tarde. Tipos de recursos utilizados. Si los recursos son muy necesarios y escasos será preferible liberarlos cuanto antes. Cuántos recursos más necesita el proceso. Es conveniente eliminar a aquellos procesos que necesitan un gran número de recursos. Facilidad de suspensión / reanudación. Se eliminarán aquellos procesos cuyo trabajo perdido sea más fácil de recuperar.

Apropiación de Recursos

Para eliminar interbloqueos utilizando la apropiación de recursos, vamos quitando sucesivamente recursos de los procesos y los asignamos a otros hasta romper el ciclo de interbloqueo.

Si se utiliza la apropiación de recursos para tratar los interbloqueos, hay que considerar tres aspectos:

- Selección de la víctima

- Retroceso

- Bloqueo indefinido

La detección y recuperación es la estrategia que a menudo se utiliza en grandes computadoras, especialmente sistemas por lote en los que la eliminación de un proceso y después su reiniciación suele aceptarse.

2.5 Niveles, objetivos y criterios de planificación.

La asignación de los procesadores físico a los procesos es lo que les permite completar su trabajo. Esta asignación es uno de los problemas complejos que maneja el sistema operativo. Este problema se le conoce como ***planificación del procesador*** y consiste en decidir cuándo y a cuál proceso se debe asignar un procesador. Además, se debe considerar la admisión de nuevo procesos al sistema, suspender y/o reanudar procesos para ajustar la carga del sistema.

Niveles de planificación.

Se consideran tres niveles de planificación importantes, los que se detallan a continuación en la **figura # 17.**

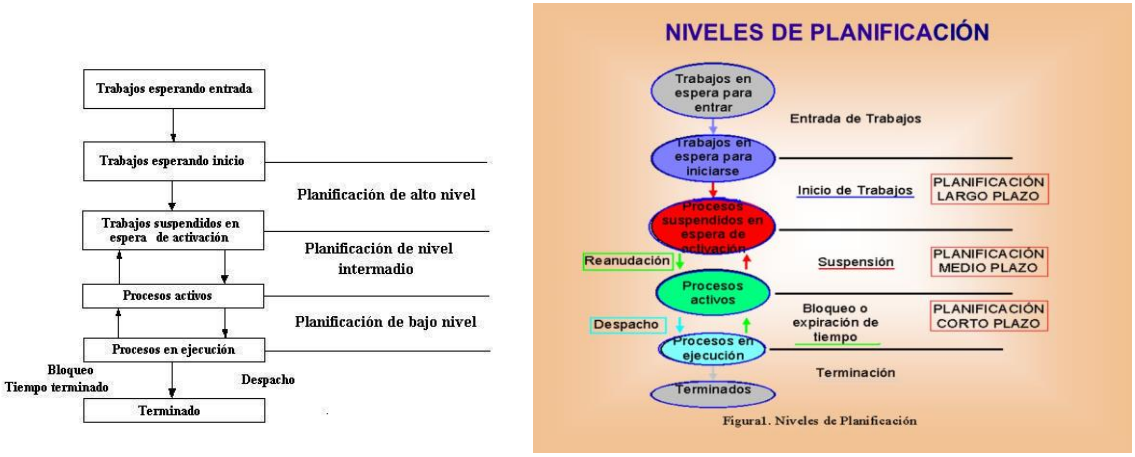

Figura # 17. Niveles de planificación.

* **Planificación de alto nivel**. Algunas veces llamada planificación de trabajos, determina a que trabajos se les va a permitir por los recursos del sistema, Algunas veces se le llama planificación de admisión porque determina a que procesos se les permite entrar al sistema.

* **Planificación de nivel intermedio**. Esta determina a que proceso se le permite competir por el CPU. Responde a las fluctuaciones a corto plazo en la carga del sistema. Suspende y/o activa temporalmente procesos para mantener una operación uniforme en el sistema y ayuda a realizar algunas funciones para optimizar el rendimiento del sistema.

* **Planificación de bajo nivel**. Determinar a qué proceso deberá asignarse el CPU (despachar). Esta operación se realiza muchas veces por segundo, por lo que el despachador debe estar de forma permanente en la memoria primaria.

Objetivo de la planificación

Se deben considerar muchos objetivos en el diseño de una disciplina de planificación. En general, debe:

. **Ser justa**. No deben existir procesos postergados indefinidamente.

. **Maximizar la capacidad de ejecución**. Maximizar el número de procesos por unidad de tiempo.

. **Maximizar el número de usuarios interactivos.**

. **Ser predecible**. Un proceso debe ser ejecutado aproximadamente en el mismo tiempo independientemente de la carga del sistema.

. **Minimizar la sobre carga.**

. **Equilibrar el uso de recursos.** Debe darse prioridad a los procesos que usan los recursos menos solicitados.

. **Lograr el equilibrio entre tiempos de respuesta y utilización de recursos.**

. Evitar la postergación indefinida.

. Asegurar las prioridades.

. Dar preferencia a los procesos que mantienen los recursos clave.

. Mejor tratamiento a los procesos con alto rendimiento.

. Degradarse en forma suave con cargas pesadas.

Muchos de estos objetivos están en conflicto unos con otros, esto hace que la planificación sea un problema complejo.

Criterios de planificación.

Para lograr los objetivos de planificación, un mecanismo de planificación debe considerar lo siguiente. (Deitel)

. Las limitaciones de I/O de un proceso.

. La limitación de CPU de un proceso.

. Si el proceso es por lotes o es interactivo.

. La urgencia de una respuesta rápida.

. La prioridad del proceso.

. La frecuencia, con la que un proceso genera fallos de página.

. Con que frecuencia un proceso ha sido apropiado por un proceso de alta prioridad.

. Cuánto tiempo de ejecución real a recibido un proceso.

Planificación Apropiativa y NO Apropiativa.

Una disciplina de planificación es NO Apropiativa, si, una vez que le ha otorgado el CPU a un proceso, ya no se le puede retirar. Una disciplina de planificación es apropiativa si se puede retirar el CPU en cualquier momento.

La planificación Apropiativa es útil en sistemas en los cuales los procesos de alta prioridad requieren atención rápida (sistemas de tiempo real). La apropiatividad tiene un costo, el intercambio de contexto implica sobrecarga. En los sistemas NO apropiativos, los trabajos largos hacen esperar a los trabajos cortos, pero el tratamiento que reciben los procesos es más justo.

Prioridades.

Las prioridades pueden ser asignadas automáticamente por el sistema o pueden ser asignadas externamente. Las prioridades pueden ser ganadas o adquiridas, pueden ser estáticas o dinámicas. Puede ser asignadas racional o arbitrariamente (no importa cuál proceso es más importante).

- Prioridades Estáticas y Dinámicas. Las prioridades estáticas no cambian nunca. Los mecanismos para implementar este esquema son fáciles de implementar y tienen una sobrecarga relativamente baja. Pero, no pueden responder a cambios en el ambiente.

- Los mecanismos de prioridad dinámica responden al cambio. Los esquemas de prioridades dinámicas son más complejos de implementar y tienen una sobrecarga mayor que los sistemas estáticos; pero, la sobrecarga se justifica ampliamente por el incremento de sensibilidad del sistema.

- Prioridades adquiridas. Un sistema operativo debe proporcionar un servicio razonable y competente a una gran comunidad de usuarios, pero debe estar preparado para aquellas situaciones en la que un miembro de la comunidad de usuarios necesita un tratamiento especial. Un usuario que necesite que su trabajo se ejecute con rapidez puede estar dispuesto a pagar un precio, adquirir prioridad, por un servicio de nivel alto. Este cargo extra se debe a que probablemente se retirarán recursos a otros usuarios que también

2.6 Técnicas de administración del planificador.

El objetivo de la multiprogramación es que en todo momento se ejecute un proceso para maximizar la utilización del CPU. En un sistema monoprocesador nunca habrá más de un proceso en ejecución. Si hay más procesos tendrán que esperar a que el CPU esté libre y pueda volver a planificarse.

El concepto de multiprogramación es muy sencillo: un proceso se ejecuta hasta que tenga que esperar, generalmente a que termine una solicitud de E/S. En un sistema de

cómputo sencillo, el CPU permanecerá inactivo; todo este tiempo de espera se desperdicia sin efectuar una actividad útil. Con la multiprogramación tratamos de emplear productivamente este tiempo. Varios procesos se conservan en la memoria a la vez, y cuando uno de ellos tiene que esperar, el sistema operativo le quita el CPU al proceso y se lo da a otro; este modelo continúa, Cada vez que un proceso tiene que esperar, otro pueda utilizar el CPU.

Los beneficios de la multiprogramación son un aumento de la utilización del CPU y una mayor productividad.

Tipos de Planeación.

La planificación es una función fundamental del sistema operativo. Casi todos los recursos de una computadora se planifican antes de usarse. Por supuesto, el CPU es una de los principales

Recursos de la computadora, de modo que su planificación es parte modular del diseño de los sistemas operativos.

- Ciclo de ráfaga del CPU y de E/S.

El éxito de la planificación del CPU depende de la siguiente prioridad observada de los procesos: la ejecución de un proceso consiste en un ciclo de ejecución del CPU y de E/S, y los procesos se alternan entre estos dos estados. La ejecución del proceso se inicia con una ráfaga de CPU; a ésta le siguen una ráfaga de E/S, otra ráfaga de CPU, una más de E/S, etc. Finalmente, la última ráfaga de CPU terminará con una solicitud al sistema para que concluya la ejecución, en vez de otra ráfaga de E/S.

Las duraciones de estas ráfagas de CPU se han medido, y, aunque varían considerablemente de un proceso a otro y entre computadoras, tienden a presentar una curva de frecuencias similar a la que se muestra en la ***figura # 18***. Generalmente la curva se caracteriza como exponencial o hiperhexponencial. Hay un gran número de ráfagas de CPU de corta duración y un pequeño número de larga duración.

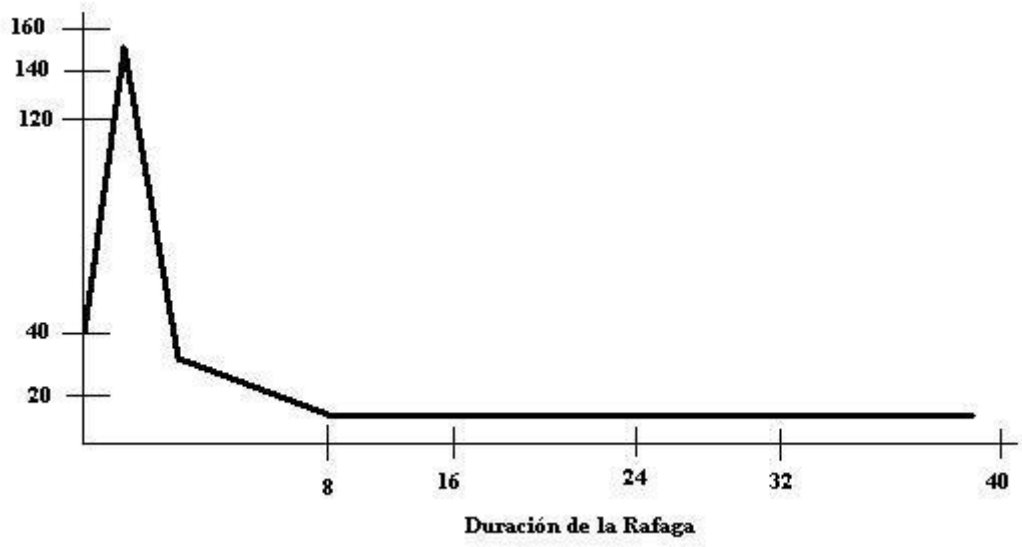

Figura # 18. Histograma de tiempos de ráfaga del CPU.

- Planificador del CPU.

Siempre que el CPU queda inactivo, el sistema operativo debe seleccionar para su ejecución uno de sus procesos de la cola de procesos listos. El proceso de selección es revisado por el planificador a corto plazo. (o planificador del CPU). El planificador selecciona uno de los procesos en memoria que están listos para ejecución y le asigna el CPU.

- Estructura de planificación.

Las decisiones de planificación del CPU pueden efectuarse en una de las cuatro circunstancias siguientes:

1. Cuando un proceso cambia del estado de ejecución a estado de espera (por ejemplo, solicitud de E/S petición de esperar la terminación de uno de los procesos hijo).

2. Cuando un proceso cambia del estado de ejecución al estado listo (por ejemplo, cuando ocurre una interrupción)

3. Cuando un proceso cambia del estado de espera al estado listo (por ejemplo, al completarse la E/S).

4. Cuando termina un proceso.

Planificación de plazo fijo.

En la planificación de plazo fijo se programan ciertos trabajos para terminarse en un tiempo específico o plazo fijo. Estas tareas pueden tener un gran valor si se entregan a tiempo, y carecer de él si se entregan después del plazo. Esta planificación es compleja por varios motivos:

El usuario debe informar por adelantado de las necesidades precisas de recursos del proceso. Semejante información rara vez está disponible.

El sistema debe ejecutar el proceso en un plazo fijo sin degradar demasiado el **servicio** a los otros usuarios y debe planificar cuidadosamente sus necesidades de recursos dentro del plazo. Esto puede ser difícil por la llegada de nuevos procesos que impongan demandas imprevistas al sistema.

Si hay muchas tareas a plazo fijo activas al mismo tiempo, la planificación puede ser tan compleja que se necesiten métodos de optimización avanzados para cumplir los plazos.

La administración intensiva de recursos requerida por la planificación de plazo fijo puede producir un **gasto extra** substancial.

2.6.1 FIFO

2.6.2 SJF

2.6.3 RR

2.6.5 HRN

2.6.6 Queves multi-level.

2.6.7 Multi-level feedback queves.