Gestión de procesos

Sistemas Operativos

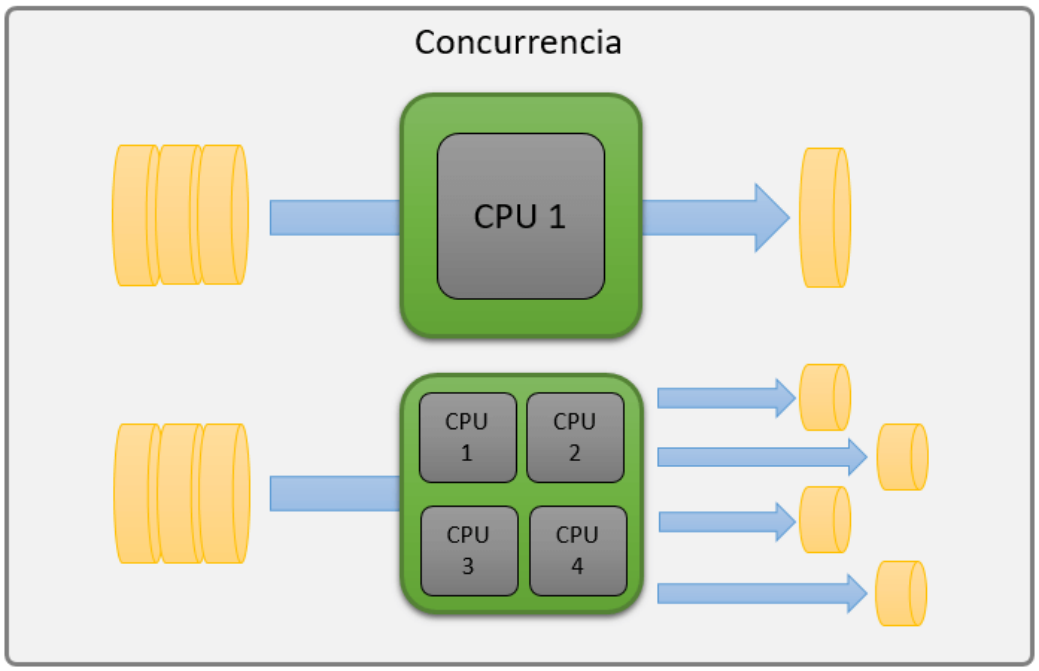
Elías Martín

El concepto fundamental de todo sistema operativo es el de proceso, pero, ¿Qué es un proceso? Es un programa en ejecución, normalmente, de un programa surgen varios procesos y estos son ejecutados de manera concurrente, permitiendo el uso del programa por más usuarios simultáneamente. La mayoría de ordenadores actualmente son sistema con multiprogramación, que quiere decir, que permiten ejecutar varios programas a la vez.

Aunque la ejecución no es simultánea, sino que, en cada instante, la CPU ejecuta un sólo programa, pero da la sensación de paralelismo, es decir, de ejecutar varios procesos al mismo tiempo.

Ejemplo de concurrencia: Imagina a una persona la cual trabaja en múltiples tareas al mismo tiempo, y que rápidamente cambia de una tarea a otra. Por ejemplo, imaginemos a una persona la cual se encuentra programando, realizando cálculos en excel y contestando correos electrónicos, todo esto al mismo tiempo. Dedica un par de segundos a cada tarea, y rápidamente, cambia de tarea. Concluimos que la persona trabaja de forma concurrente. Las tareas que realiza no necesariamente deben seguir un orden, quizás, después de contestar un correo regresa con los cálculos en excel, le dedica un par de segundo, regresa a responder otro correo y finaliza con la codificación del programa, u, otro escenario pudiera ser que después finalizar ciertos cálculos, la persona continúa codeando un par de segundos para después responder un par de correos y regresar con los cálculos.

Si llevamos esto al procesador, antes tenemos que dejar claro cómo funciona. Un procesador puede procesar al mismo tiempo el mismo número de procesos que el número de CORES que tiene, de esta forma, si un procesador tiene un CORE (núcleo), entonces solo podrá ejecutar un proceso a la vez, por otro parte, si tenemos 4 CORES, entonces podremos ejecutar hasta 4 procesos al mismo tiempo.



Al conjunto de elementos que son necesarios para la ejecución de un proceso lo denominamos **imagen del proceso,** como puede ser la pila dónde se gestionan las llamadas a procedimientos y funciones, o una serie de atributos del proceso llamados **bloques de control del proceso o BCP**. Estos deben permanecer en memoria principal para que el SO pueda planificar la ejecución de los procesos.

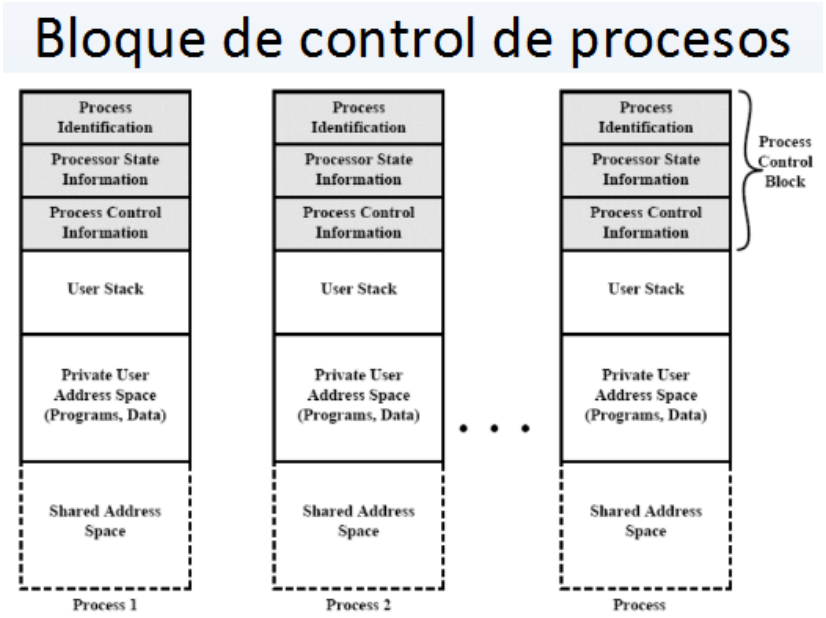
La concurrencia implica que los procesos entren y salgan de la CPU sin haber terminado de ejecutarse, para evitar perder la información, esta es guardada en el BCP de cada proceso. **La estructura del BCP se divide en tres partes:**

1. Un identificador del proceso **(PID)**, con información del proceso que lo creó y el usuario que lo generó.

2. Información sobre el estado del procesador, entre las que destacamos los registros de control y de estado que controlan el funcionamiento, los punteros a pila (cada proceso tiene una o más pilas LIFO del sistema asociadas, utilizadas para almacenar parámetros y

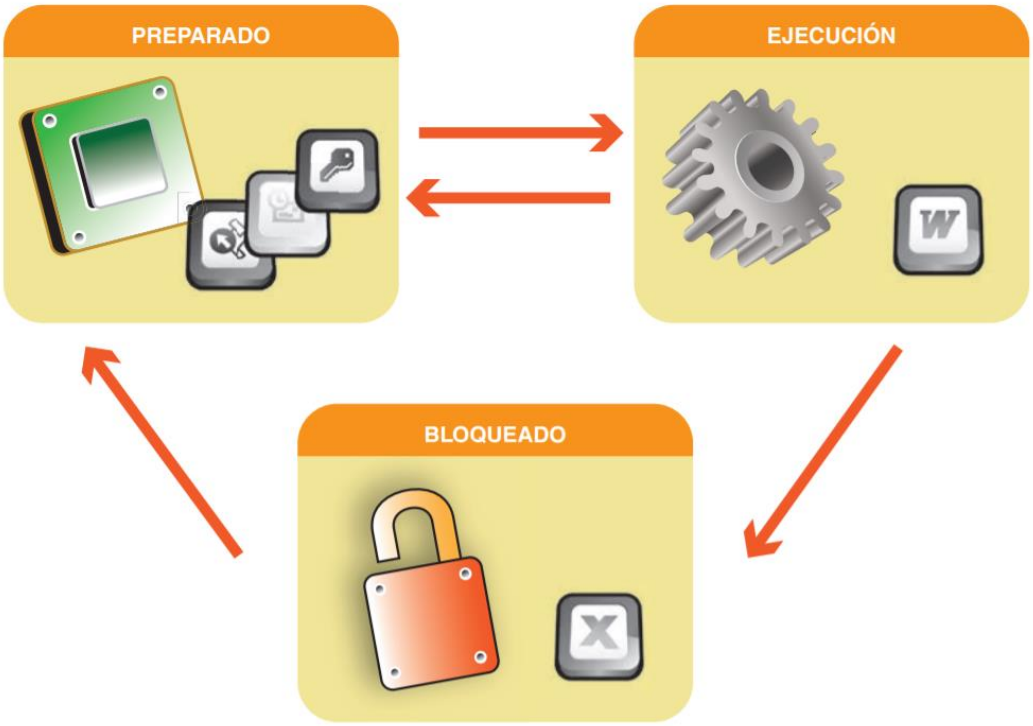
direcciones de retorno de los procedimientos y llamadas al sistema) y el contador de programa para ejecutar el proceso desde el punto donde se paró.

3. Información de control del proceso, necesaria por el SO para llevar a cabo la planificación de los procesos. Incluye información sobre el estado del proceso (ejecutándose, listo, esperando, bloqueado…), su prioridad, el suceso que está esperando que ocurra para reanudarse y otro tipo de información como el tiempo que lleva esperando o el tiempo que lleva sin ejecutarse desde la última vez.



El BCP se genera cuando se crea el proceso, y se gestiona y almacena en **una tabla de bloques de procesos (TBCP)**. Cuando un proceso entra en ejecución, los datos almacenados en su BCP se vuelcan sobre los registros de la CPU, y se realizará el proceso inverso cuando la ejecución del proceso acabe y salga. Si queremos que el rendimiento del sistema sea óptimo, estos cambios deben hacerse lo más rápidos posible, ya que la CPU estará inactiva mientras se produzcan.

Estados de los procesos: Básicamente los procesos pueden encontrarse en tres estados diferentes, el primer estado es en espera o bloqueado, dónde no tienen la posibilidad de entrar a ejecución, por ejemplo, porque dos procesos usen el mismo fichero de datos. En estado preparado o listo, con posibilidad de entrar en ejecución, es decir, está esperando turno para poder utilizar su intervalo de tiempo y poner en funcionamiento sus instrucciones accediendo a los recursos del sistema. Y, por último, en ejecución o activo, que indica que el procesador está ejecutando instrucciones del programa que lo compone y tiene concedido el tiempo de uso de la CPU en un instante concreto.

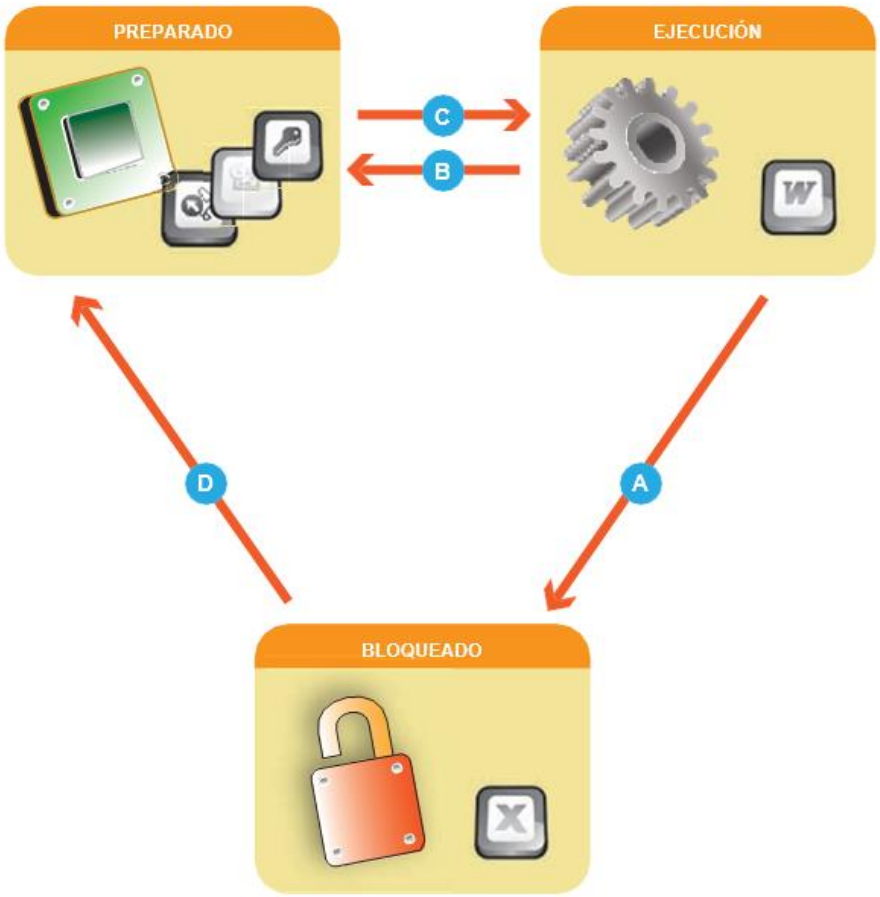


Una vez que un programa se ha lanzado y se ha convertido en proceso, puede cambiar de estado por diferentes situaciones hasta su finalización. En primer lugar, no podrá ejecutarse directamente, antes se coloca en una cola de procesos con el estado preparado, si dispone de los recursos que necesita para su ejecución, si no es así, estará bloqueado hasta entonces.

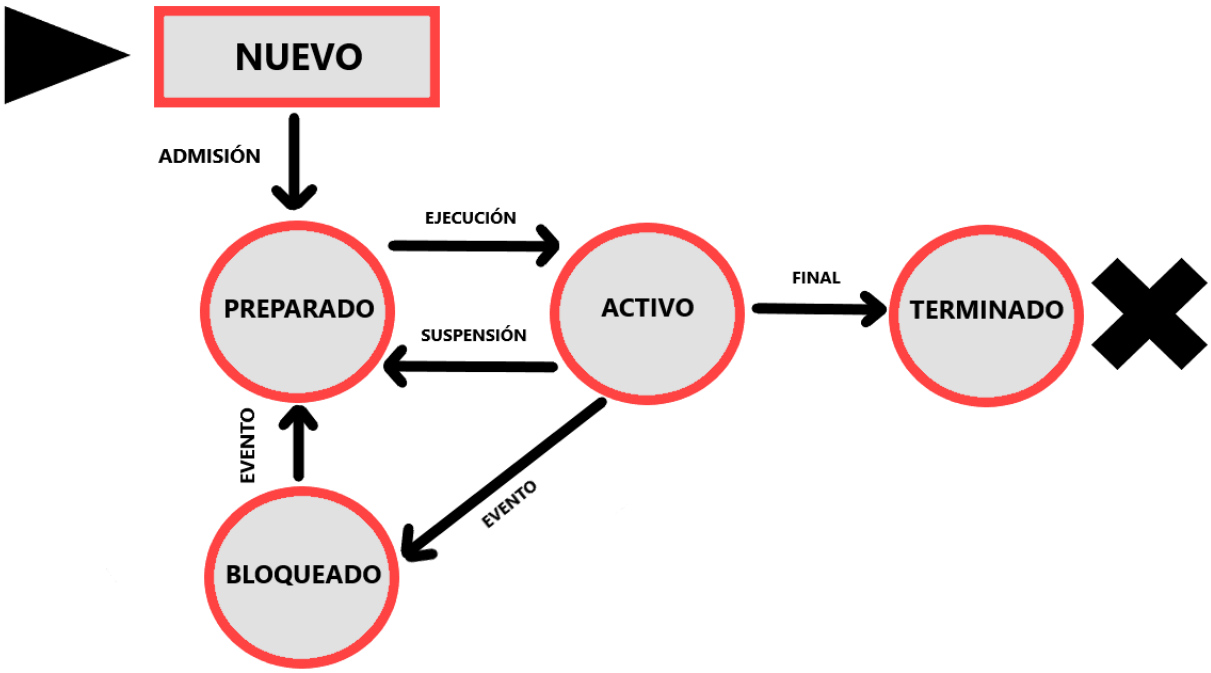
Cuando la CPU le asigna su tiempo, el proceso pasa de preparado a ejecución. Los estados preparados y en ejecución se alternarán en caso de que se esté ejecutando más de un proceso en el sistema, por ejemplo, cuando el SO crea que ya han estado el tiempo suficiente ejecutándose.

Los cambios de estado en los que se puede encontrar un proceso se denominan transiciones. En la imagen siguiente se recogen las transiciones o cambios de estado que pueden experimentar los procesos.

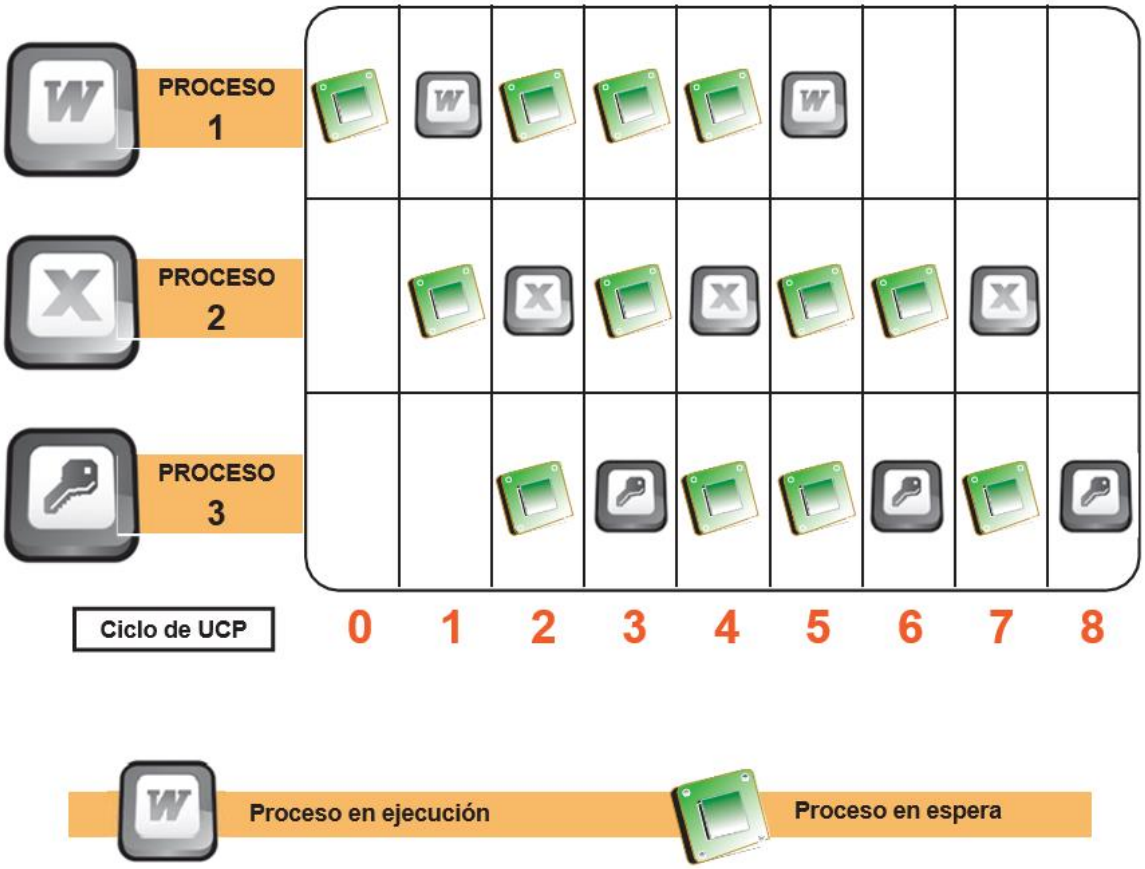
- Transición A. Ocurre cuando el programa que está en ejecución necesita algún elemento, señal, dato, etc., para continuar ejecutándose.
- Transición B. Ocurre cuando un programa o proceso ha utilizado el tiempo asignado por la UCP (procesador) para su ejecución y tiene que dejar paso al siguiente proceso.
- Transición C. Ocurre cuando el proceso que está preparado pasa al proceso de ejecución, es decir, cuando al proceso le llega una nueva disposición de tiempo de la UCP para poder ejecutarse.
- Transición D. Ocurre cuando el proceso pasa de estar bloqueado a estar preparado, es decir, cuando el proceso recibe una orden o señal que estaba esperando para pasar al estado de preparado y, posteriormente, tras la transición, a estado de ejecución.



Lo que acabamos de leer es llamado **modelo de tres estados**, en cambio, si añadimos dos más, pasamos al de cinco estados, estos son el estado nuevo, asignado a los procesos que acaban de crearse. Un proceso nuevo, pese a estar listo, si el sistema tiene un límite de procesos listos, no cambiará su estado de nuevo a listo. El otro estado es terminado, para procesos que pueden haber tenido un error y que no deben volver a ejecutarse pero que no son eliminados aún por si otro programa necesita extraer alguna información de sus tablas. Por último, para evitar que el sistema pierda eficiencia al tener muchos procesos bloqueados en memoria principal, se crea el estado suspendido, que es un proceso bloqueado pero que se llevará a disco en lugar de tenerlo en memoria principal.



Una vez visto lo que es el modelo de procesos, finalizando este apartado explicaremos la forma de implementación, la cual se basa en que el SO utilizará la tabla de procesos, que se actualiza cuando los procesos pasan del estado de ejecución a listo, como, por ejemplo, cuando el procesador está ejecutando un proceso y llega una interrupción, en ese momento se necesita salvar el proceso que se estaba ejecutando para que el que solicitó la interrupción, se pase al estado listo para ejecutarse. Lo que ocurra después dependerá del algoritmo de planificación utilizado, que veremos más adelante.



Comunicación entre procesos

Antes de pasar a ver la planificación del procesador, en este nuevo apartado trataremos una parte muy importante, la forma de comunicarse los procesos. Para ello es conveniente usar algún método estructurado y evitar el uso de interrupciones.

Cuando los procesos que trabajan juntos acceden a espacios compartidos en los que pueden leer o almacenar datos se pueden producir **condiciones de competencia**. Conceptualmente, la solución de los problemas de acceso a recursos compartidos es prohibiendo que más de un proceso lea o escriba simultáneamente datos compartidos, es decir, la solución es establecer una exclusión mutua. Por ejemplo, mientras editas un archivo y intenta cambiar el nombre del mismo, no te dejará hasta que el proceso que está editando el archivo termine.

Sección critica: Por otro lado, denominamos sección crítica a aquella parte del programa en la cual se accede al recurso compartido. De este modo, si podemos evitar que dos procesos entren simultáneamente en una sección crítica evitaremos las condiciones de competencia.

Exclusión mutua: Los métodos de exclusión mutua pretenden evitar que dos o más procesos se solapen en sus secciones críticas. A continuación, explicaremos algunos métodos.

- **Desactivación de interrupciones:** El proceso que entra en la región crítica desactiva todas las interrupciones, de este modo ningún otro proceso podrá entrar también en su sección crítica. Este método es inadmisible por permitir que un proceso de usuario pueda desactivar totalmente las interrupciones y además por ser inútil si se tienen más de una CPU.
- **Algoritmo de Peterson:** Antes de acceder a su sección crítica, cada proceso llama a una función denominada entrarSecciónCrítica pasando como parámetro su número de proceso. Si el otro proceso está en esa sección crítica, esperará hasta que la abandone para poder entrar, en caso contrario, entrará inmediatamente. Al final de la sección crítica cada proceso llamará a una función denominada salirSecciónCrítica.

  Vistos los inconvenientes que se presentan en las anteriores soluciones, es difícil generalizarlas para problemas de sincronización más complejos, a continuación, veremos el soporte que ofrecen los SO a la concurrencia de procesos.

- **Semáforos:** En primer lugar, tenemos los semáforos. Son estructuras que nos permiten asegurar la exclusión mutua y sincronizar los procesos, evitando que realicen continuamente comprobaciones a la CPU para poder entrar. Los semáforos constan básicamente de dos operaciones, up y down, que operan sobre un tipo especial de

variable semáforo, que sólo puede tomar valores enteros. Las dos operaciones tienen como función incrementar o decrementar el valor del semáforo. Las condiciones que se establecen para implementar un semáforo son que todos los procesos pueden acceder a la variable semáforo, que sólo pueda ser modificada con sus operaciones y que una vez inicializada una operación con un semáforo, ningún otro proceso pueda acceder a él.

Por lo tanto, un semáforo deberá ser inicializado con un valor entero, si tomamos los semáforos binarios, estos sólo toman los valores 1, que indica que no hay procesos ejecutándose en la sección crítica en ese momento, y 0, que quiere decir que sí hay alguno. Los semáforos binarios, por lo tanto, deberán ser inicializados con 1. En el momento en que un proceso quiera entrar a la sección crítica, deberá comprobar el valor del semáforo, en caso de ser 1, deberá realizar la operación down para decrementar el valor del semáforo antes de entrar. Y, en caso de ser 0, quedará bloqueado hasta que termine el proceso que se esté ejecutando. Cuando un proceso termina, realizará la operación up para incrementar de nuevo el valor del semáforo para que otro pueda entrar, generalmente el siguiente que llegó.

El principal inconveniente de los semáforos está en que las operaciones down y up deben distribuirse por todo el programa y no es fácil controlar el efecto global de estas operaciones sobre los semáforos a los que afectan. En los semáforos, tanto la exclusión mutua como la sincronización son responsabilidades del programador. Es debido a esto, por lo que se desarrollaron los monitores.

- **Monitores:** Como hemos dicho, los semáforos son una herramienta básica, pero potente y flexible, para hacer cumplir la exclusión mutua y coordinar procesos. Sin embargo, puede resultar muy difícil construir un programa correcto por medio de semáforos. A continuación, veremos los monitores, estructuras de un lenguaje de programación que ofrecen una funcionalidad equivalente a la de los semáforos y que son más fáciles de controlar.

  Un monitor es una colección de procedimientos, variables y estructuras de datos que se agrupan en un tipo especial de módulo. Los procesos pueden invocar los procedimientos de un monitor en el momento en que deseen, de esta forma, entran en el monitor, pero nunca tendrán acceso a las estructuras de datos internas del monitor desde procedimientos declarados fuera de él.

Los monitores poseen una propiedad especial que los hace útiles para lograr la exclusión mutua: sólo un proceso puede estar activo en un monitor en un momento dado. El resto deberá quedar a la espera formando una cola dónde los procesos que llegan antes entran primeros, con excepciones de prioridades. Al ser estructuras del lenguaje de programación, el compilador es el encargado de la exclusión mutua entre los procedimientos del monitor, por lo tanto, el programador sólo tendrá que transformar las secciones críticas en estos procedimientos.

Por último, tenemos que tener en cuenta el **interbloqueo**, el cual consiste en que dos o más procesos adquieren algún recurso necesario para su operación a la vez que esperan a que se liberen otros recursos que retienen otros procesos, llegándose a una situación que hace imposible que ninguno de ellos pueda continuar.

Para ello, se deben cumplir de forma simultánea las cuatro condiciones siguientes:

1. Exclusión mutua: Los procesos utilizan de forma exclusiva los recursos que han adquirido. Si otro proceso pide el recurso debe esperar a que este sea liberado.
2. Retención y espera: Los procesos retienen los recursos que han adquirido mientras esperan a adquirir otros recursos que está siendo retenidos por otros procesos.
3. No expropiación: Los recursos no pueden quitarse a los procesos que los tienen. Su liberación se produce voluntariamente una vez que los procesos han finalizado su tarea.
4. Espera circular: Existe una cadena circular de procesos en la que cada proceso retiene al menos un recurso que es solicitado por el siguiente proceso de la cadena.

Deberemos prevenir las causas del interbloqueo, comprobando periódicamente si se ha producido o evitando alguna de las causas anteriores.

Planificación del procesador

Es ahora cuando hemos de hablar de la planificación. Con esta técnica conseguimos indicar al ordenador los procesos que deben ejecutarse y los estados que estos deben adoptar. Gracias a los algoritmos de planificación podemos decidir qué proceso ha de ejecutarse en cada momento y por qué. En estos cambios de proceso, el sistema operativo tiene que saber qué ficheros están abiertos en cada proceso, qué periféricos se están utilizando, etc.

Cuando se están ejecutando varias tareas a la vez (procesos), es necesario compartir el tiempo de trabajo de la CPU. El tiempo compartido consiste en dividir el tiempo de ejecución del procesador en minúsculos intervalos de tiempo (quantum) e ir asignando cada uno de esos intervalos de ejecución a cada proceso que está en ejecución.

Para planificar se definen tres tipos de planificación, de admisión, dónde su objetivo es determinar que trabajos deben admitirse en el sistema. Planificación intermedia, que determina que procesos pueden competir por el uso de la CPU, y la planificación a bajo nivel, que determina que procesos en estado listo, se les asignará la CPU. En este apartado, nos vamos a centrar en este último tipo de planificación.

Planificación a bajo nivel: En primer lugar, un buen algoritmo deberá asignar el procesador de forma equitativa a los procesos, mantener al procesador ocupado en todo momento, maximizar el número de procesos por unidad de tiempo y minimizar el tiempo de respuesta de los procesos.

Los algoritmos que veremos más adelante en detalle serán:

- Algoritmo FIFO (First In First Out) o FCFS (First Come First Serve)
- Algoritmo Round Robind o de rueda
- Prioridad
- SJF (Shortest Join First)
- SRT (Shortest Remaining Time - El que tiene menor tiempo restante)

Procesos e hilos (hebras)

En los SO clásicos, cada proceso tiene un espacio de direcciones y un único hilo de control. En los SO multitarea modernos, es posible tener muchos hilos dentro de un mismo proceso. Podemos definir los hilos como distintas líneas de ejecución dentro de un mismo proceso, que pueden ser ejecutados al mismo tiempo, en otras palabras, podemos tratar a los hilos como subprocesos.

Un proceso clásico será aquel que solo posea una hebra. Pongamos un ejemplo. Si ejecutamos el procesador de textos Word, con un solo documento abierto, el programa Word convertido en proceso estará ejecutándose en un único espacio de memoria, tendrá acceso a determinados archivos (galerías de imágenes, corrector ortográfico, etc.), tendrá acceso al hardware (impresora, disquetera), etc. En definitiva, este proceso, de momento, solamente tiene una hebra.

Un proceso clásico será aquel que solo posea una hebra. Pongamos un ejemplo. Si ejecutamos el procesador de textos Word, con un solo documento abierto, el programa Word convertido en proceso estará ejecutándose en un único espacio de memoria, ten- drá acceso a determinados archivos (galerías de imágenes, corrector ortográfico, etc.), tendrá acceso al hardware (impresora, disquetera), etc. En definitiva, este proceso, de momento, solamente tiene una hebra.

Word se está ejecutando una sola vez y el resto de documentos de texto que abramos en esta misma sesión de trabajo no serán procesos propiamente dichos. Serán hilos o hebras del proceso principal, que es el propio procesador de textos.

Cada hilo se ejecuta secuencialmente con los otros hilos del proceso, alternándose la CPU de la misma forma que lo hacen los procesos, destacamos, además, que en sistemas multiprocesador se pueden ejecutar realmente en paralelo. La creación y uso de nuevos hilos frente a nuevos procesos, ofrece muchas ventajas, entre ellas destacamos las siguientes:

- El SO tardará mucho menos en crear, terminar o modificar hilos que lo que tarda con los procesos, debido a que la mayoría de información del BCP es compartida por todos los hilos del proceso.
- En segundo lugar, al pertenecer a un mismo proceso, los hilos permiten la comunicación entre ellos de forma inmediata, ya que se localizan en la misma posición de memoria.
- Y por último destacar que, como un proceso puede generar varios hilos, estos podrían ejecutarse de forma paralela en un sistema multiproceso.

Su implementación puede hacerse de dos maneras, a nivel de usuario, donde la gestión de los hilos es realizada por una aplicación, de esta forma el núcleo no participa, y no aprovecha los sistemas multiprocesador, ya que el núcleo asigna un proceso a un solo procesador y sólo se puede ejecutar en él un hilo de proceso a la vez. O a nivel de kernel o núcleo, el cual sabe de la existencia de los hilos y debe gestionarlos completamente. Para cada proceso, el núcleo tiene una tabla con una entrada por cada hilo y su información, ahora las llamadas que gestionan los hilos se implantan como llamadas al sistema. La principal ventaja es que el núcleo se encarga directamente de la gestión, evitando todos los problemas que surgen del anterior método a nivel de usuario.