Sistemas Operativos

Alumno: Santiago Vietto

Docente: Gustavo Funes

Institución: UCC

Año: 2022

Introducción a los sistemas informáticos

Sistemas operativos

_ Un sistema operativo explota los recursos del hardware de uno o más procesadores para ofrecer un conjunto de servicios a los usuarios del sistema. También gestiona la memoria secundaria y los dispositivos de entrada/salida (E/S) para sus usuarios.

Elementos básicos

_ Los elementos básicos del sistema operativo son:

Procesador: controla el funcionamiento del computador y realiza sus funciones de procesamiento de datos. Cuando sólo hay un procesador, se lo denomina usualmente como unidad central de proceso (Central Processing Unit, CPU).

Memoria principal: almacena datos y programas. Esta memoria es habitualmente volátil; es decir, que cuando se apaga el computador, se pierde su contenido, a diferencia del contenido de la memoria del disco que se mantiene incluso cuando se apaga el computador. A esta memoria se le conoce también como memoria real o memoria primaria.

Módulos de E/S: estos transfieren los datos entre el computador y su entorno externo. El entorno externo está formado por diversos dispositivos, incluyendo dispositivos de memoria secundaria (por ejemplo, discos), equipos de comunicaciones y terminales.

Interconexión de sistemas: permiten la comunicación entre procesadores, memoria principal y los módulos de E/S.

Componentes de más alto nivel

_ Una de las funciones del procesador es el intercambio de datos con la memoria. Para este fin, se utilizan normalmente dos registros internos (al procesador):

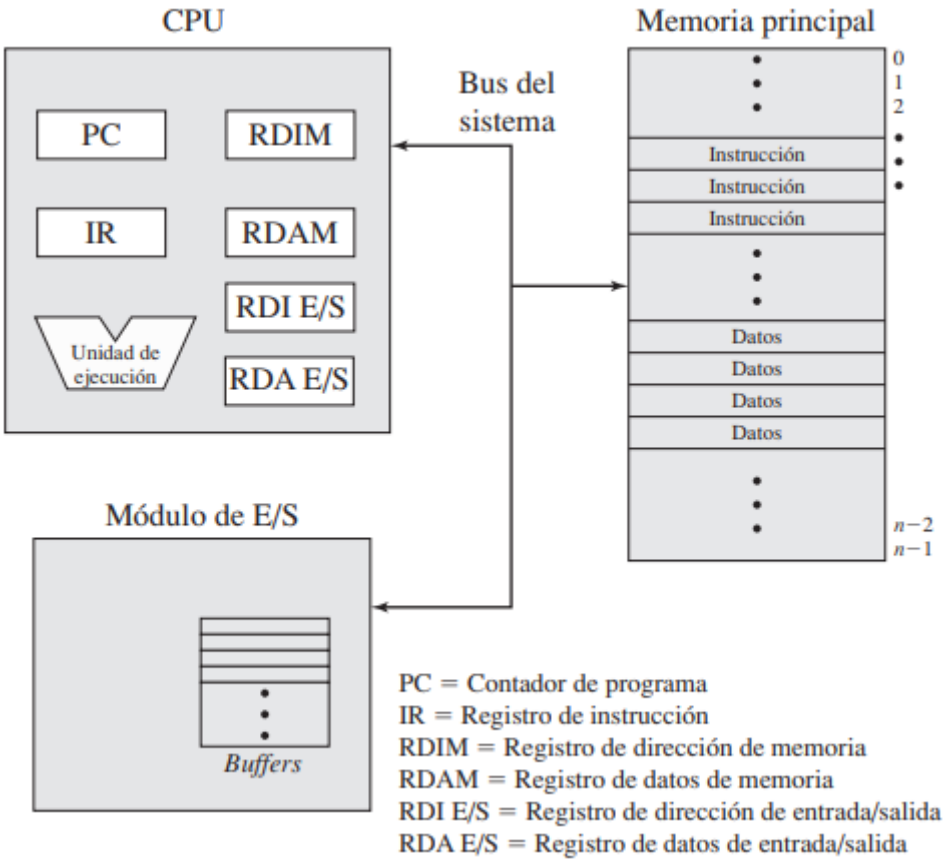
- Registro de dirección de memoria (RDIM): especifica la dirección de memoria de la siguiente lectura o escritura.
- Registro de datos de memoria (RDAM): contiene los datos que se van a escribir en la memoria o que recibe los datos leídos de la memoria.

_ De manera similar tenemos los siguientes registros:

- Registro de dirección de E/S (RDIE/S): especifica un determinado dispositivo de E/S.
- Registro de datos de E/S (RDAE/S): permite el intercambio de datos entre un módulo de E/S y el procesador.

_ Analizando un poco el siguiente esquema, un módulo de memoria consta de un conjunto de posiciones definidas mediante direcciones numeradas secuencialmente. Cada posición contiene un patrón de bits que se puede interpretar como una

instrucción o como datos. Un módulo de E/S transfiere datos desde los dispositivos externos hacia el procesador y la memoria, y viceversa. Contiene buffers (es decir, zonas de almacenamiento internas) que mantienen temporalmente los datos hasta que se puedan enviar.



Registros del procesador

_ Un procesador incluye un conjunto de registros que proporcionan un tipo de memoria que es más rápida y de menor capacidad que la memoria principal. Los registros del procesador sirven para dos funciones:

Registros visibles de usuario

_ Estos permiten al programador en lenguaje máquina o en ensamblador minimizar las referencias a memoria principal optimizando el uso de registros. Para lenguajes de alto nivel, un compilador que realice optimización intentará tomar decisiones inteligentes sobre qué variables se asignan a registros y cuáles a posiciones de memoria principal. A un registro visible para el usuario se puede acceder por medio del lenguaje de máquina ejecutado por el procesador que está generalmente disponible para todos los programas, incluyendo tanto programas de aplicación como programas de sistema. Los tipos de registros que están normalmente disponibles son:

Registro de datos: son de propósito general y pueden usarse con cualquier instrucción de máquina que realice operaciones sobre datos. Por ejemplo, puede haber registros dedicados a operaciones de coma flotante y otros a operaciones con enteros.

Registros de dirección: contienen direcciones de memoria principal de datos e instrucciones, o una parte de la dirección que se utiliza en el cálculo de la dirección efectiva o completa. Estos registros pueden ser en sí mismos de propósito general, o pueden estar dedicados a una forma, o modo, particular de direccionamiento de memoria. Se incluyen algunos ejemplos:

- Registro índice: se basa en el direccionamiento indexado, que implica sumar un índice a un valor de base para obtener una dirección efectiva.
- Puntero de segmento: con direccionamiento segmentado, la memoria se divide en segmentos, que son bloques de palabras de longitud variable. Una referencia de memoria consta de una referencia a un determinado segmento y un desplazamiento dentro del segmento.
- Puntero de pila: si hay direccionamiento de pila visible para el usuario, hay un registro dedicado que apunta a la parte superior de la pila. Esto permite el uso de instrucciones que no contienen campo de dirección, tales como las que permiten apilar (push) y extraer (pop).

Registros de control y de estado

_ Se emplean varios registros del procesador para controlar el funcionamiento del mismo. En la mayoría de las máquinas, muchos de ellos no son visibles para el usuario. A algunos de ellos se puede acceder mediante instrucciones de máquina ejecutadas en lo que se denomina modo de control o de sistema operativo. Los siguientes son esenciales para la ejecución de instrucciones:

Contador de programa (Program Counter, PC): contiene la dirección de la próxima instrucción de la memoria que será leída.

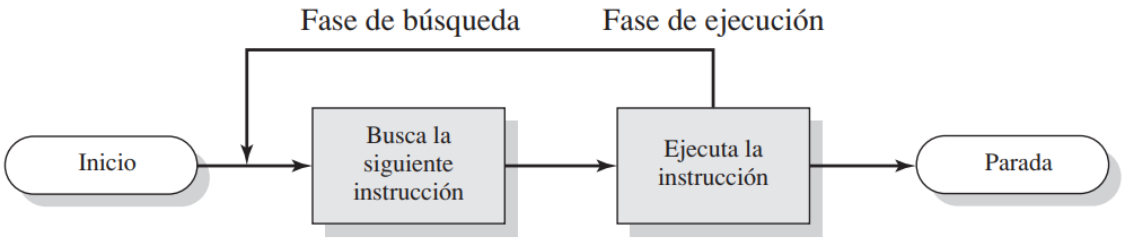
Registro de instrucción (Instruction Register, IR): contiene la última instrucción leída.

Palabra de estado del programa (Program Status Word, PSW): contiene información de estado, además contiene códigos de condición, además de otra información de estado como un bit para habilitar/inhabilitar las interrupciones y un bit de modo usuario/supervisor.

Códigos de condición o flags: son bits cuyo valor lo asigna normalmente el hardware de procesador teniendo en cuenta el resultado de determinadas operaciones. Por ejemplo, una operación aritmética puede producir un resultado positivo, negativo, cero o desbordamiento. Estos se pueden acceder a través de un programa o que se lean mediante una referencia implícita, pero no pueden ser alterados por una referencia explícita debido a que están destinados a la realimentación del resultado de la ejecución de una instrucción.

Ejecución de instrucciones

Ciclo de instrucción: se denomina asi al procesamiento requerido por una única instrucción.



Búsqueda y ejecución de una instrucción

_ Al principio de cada ciclo de instrucción, el procesador lee una instrucción de la memoria. Luego el contador del programa (PC) almacena la dirección o lleva la cuenta de la siguiente instrucción que se va a leer. Después, a menos que se le indique otra cosa, el procesador siempre incrementa el PC después de cada instrucción ejecutada, de manera que se leerá la siguiente instrucción en orden secuencial.

Registro de instrucción

_ La instrucción leída se carga dentro de un registro del procesador conocido como registro de instrucción (IR). El procesador interpreta la instrucción y lleva a cabo la acción requerida. En general, estas acciones se dividen en cuatro categorías:

Procesador-memoria: se pueden transferir datos desde el procesador a la memoria o viceversa.

Procesador-E/S: se pueden enviar o recibir datos desde un dispositivo periférico, transfiriéndolos entre el procesador y un módulo de E/S.

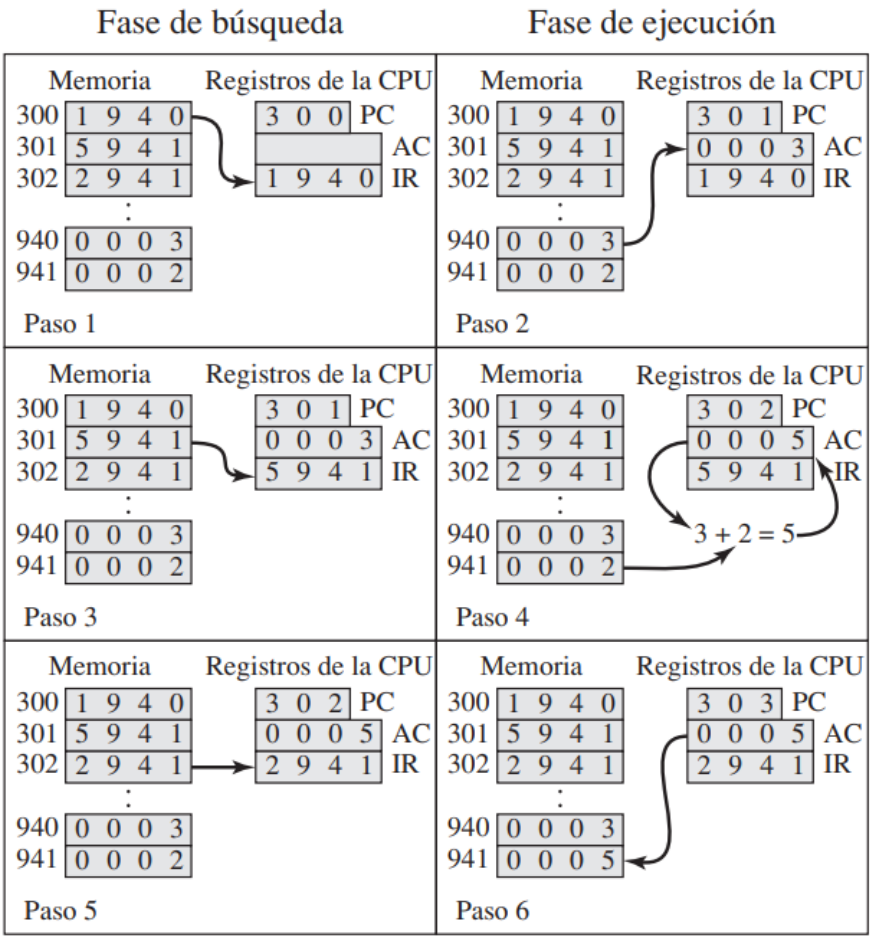
Procesamiento de datos: el procesador puede realizar algunas operaciones aritméticas o lógicas sobre los datos.

Control: una instrucción puede especificar que se va a alterar la secuencia de ejecución.

_ A continuación, tenemos un ejemplo de ejecución de un programa (contenido de la memoria y los registros en hexadecimal):

1. El PC contiene el valor 300, la dirección de la primera instrucción. Esta instrucción (el valor 1940 en hexadecimal) se carga dentro del registro de instrucción IR y se incrementa el PC. Nótese que este proceso involucra el uso del registro de dirección de memoria (RDIM) y el registro de datos de memoria (RDAM). Para simplificar, no se muestran estos registros intermedios.

2. Los primeros 4 bits (primer dígito hexadecimal) en el IR indican que en el AC se va a cargar un valor leído de la memoria. Los restantes 12 bits (tres dígitos hexadecimales) especifican la dirección de memoria, que es 940.
3. Se lee la siguiente instrucción (5941) de la posición 301 y se incrementa el PC.
4. El contenido previo del AC y el contenido de la posición 941 se suman y el resultado se almacena en el AC.
5. Se lee la siguiente instrucción (2941) de la posición 302 y se incrementa el PC.
6. Se almacena el contenido del AC en la posición 941.



Sistema de E/S

Acceso directo a memoria (Direct Memory Access, DMA): en algunos casos, es deseable permitir que los intercambios de E/S se produzcan directamente con la memoria para liberar al procesador de la tarea de E/S. En tales casos, el procesador otorga a un módulo de E/S la autorización para leer o escribir en la memoria, de manera que la transferencia entre memoria y E/S puede llevarse a cabo sin implicar al procesador.

Interrupciones

_ Básicamente, las interrupciones constituyen una manera de mejorar la utilización del procesador. Se trata de una interrupción de la ejecución normal del procesador, en donde mejora la eficiencia del procesamiento, permite al procesador ejecutar otras instrucciones mientras una operación de E/S está en proceso. Cuando hay una interrupción de un proceso debida a un factor externo al mismo, se lleva a cabo de tal modo que el procesador pueda reanudarse.

Clases de interrupciones

De programa: generadas por alguna condición que se produce como resultado de la ejecución de una instrucción, tales como un desbordamiento aritmético, una división por cero, un intento de ejecutar una instrucción de máquina ilegal, y las referencias fuera del espacio de la memoria permitido para un usuario.

Por temporizador (reloj): generadas por un temporizador del procesador. Permite al sistema operativo realizar ciertas funciones de forma regular.

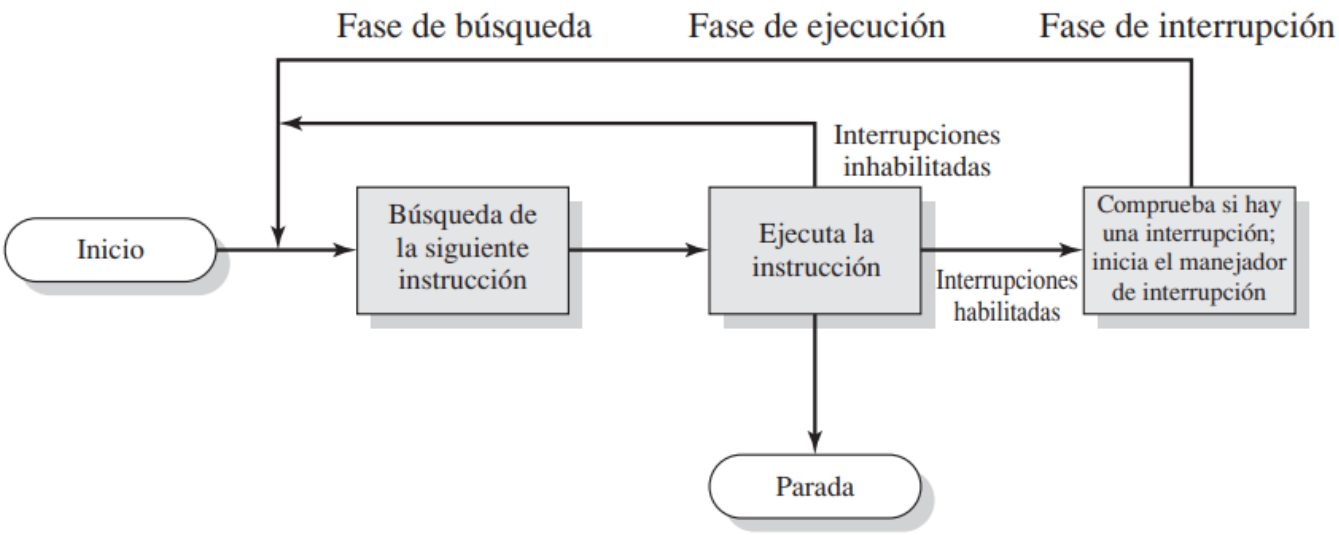
De E/S: generadas por un controlador de E/S para señalar la conclusión normal de una operación o para indicar diversas condiciones de error.

Por fallo del hardware: generada por un fallo, como un fallo en el suministro de energía o un error de paridad en la memoria.

Tratamiento y ciclo de la interrupción

_ Para tratar las interrupciones, se añade una fase de interrupción al ciclo de instrucción como se ve en el gráfico. En la fase de interrupción, el procesador comprueba si se ha producido cualquier interrupción, y lo sabrá por la presencia de una señal de interrupción. Si no hay interrupciones pendientes, el procesador continúa con la fase de búsqueda y lee la siguiente instrucción del programa actual. Si está pendiente una interrupción, el procesador suspende la ejecución del programa actual y ejecuta la rutina del tratamiento de interrupción. La rutina del manejador de interrupción es generalmente parte del sistema operativo. Normalmente, esta rutina determina la naturaleza de la interrupción y realiza las acciones que se requieran.

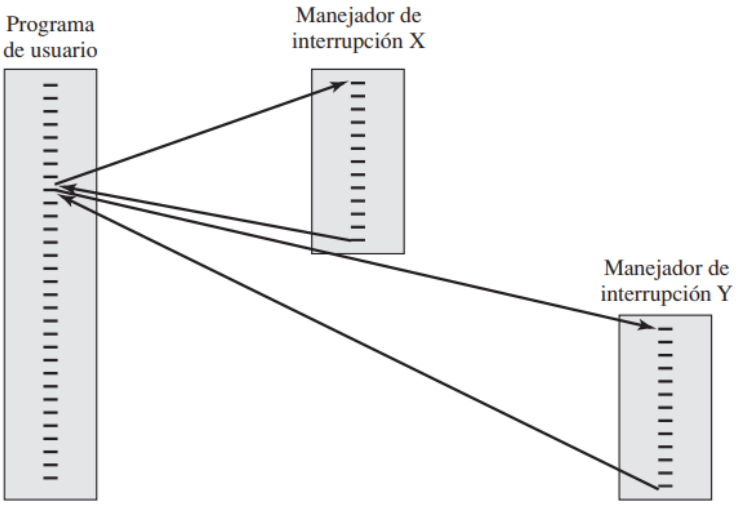
_ Vemos el ciclo de instrucción con interrupciones:



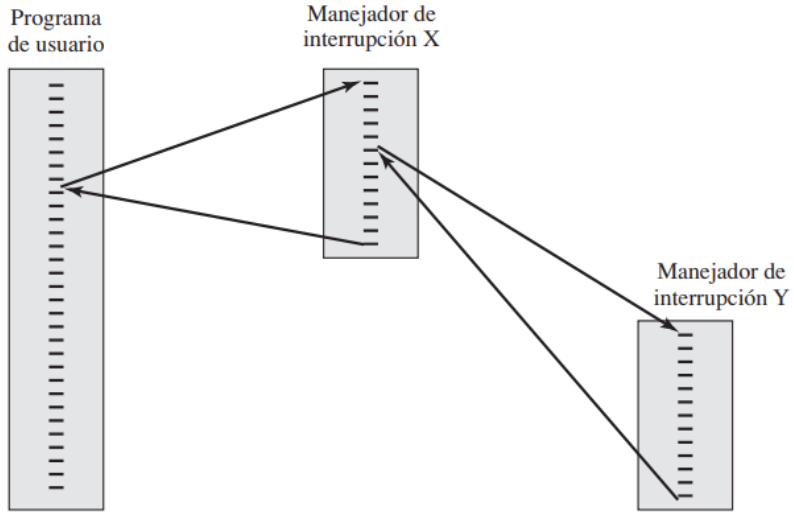
Múltiples interrupciones

_ Se pueden considerar dos alternativas a la hora de tratar con múltiples interrupciones:

Procesamiento secuencial de interrupciones: consiste en inhabilitar las interrupciones mientras que se está procesando una interrupción. Una interrupción inhabilitada significa simplemente que el procesador ignorará cualquier nueva señal de petición de interrupción. Si se produce una interrupción durante este tiempo, generalmente permanecerá pendiente de ser procesada, de manera que el procesador sólo la comprobará después de que se rehabiliten las interrupciones. Por tanto, cuando se ejecuta un programa de usuario y se produce una interrupción, se inhabilitan las interrupciones inmediatamente. Después de que se completa la rutina de manejo de la interrupción, se rehabilitan las interrupciones antes de reanudar el programa de usuario, y el procesador comprueba si se han producido interrupciones adicionales. La desventaja de la estrategia anterior es que no tiene en cuenta la prioridad relativa o el grado de urgencia de las interrupciones.



Procesamiento anidado de interrupciones: consiste en definir prioridades para las interrupciones y permitir que una interrupción de más prioridad cause que se interrumpa la ejecución de un manejador de una interrupción de menor prioridad. Por ejemplo, cuando llega una interrupción desde la línea de comunicaciones, se necesita atender ésta rápidamente para hacer sitio a nuevas entradas.



Multiprogramación

_ Cuando el procesador trata con varios programas, la secuencia en la que se ejecutan los programas dependerá de su prioridad relativa, así como también de si están esperando la finalización de una operación de E/S. Cuando se interrumpe un programa y se transfiere el control a un manejador de interrupción, una vez que se ha completado la rutina del manejador de interrupción, puede ocurrir que no se le devuelva inmediatamente el control al programa de usuario que estaba en ejecución en ese momento. En su lugar, el control puede pasar a algún otro programa pendiente de ejecutar que tenga una prioridad mayor. Posteriormente, se reanudará el programa de usuario interrumpido previamente, en el momento en que tenga la mayor prioridad.

Jerarquía de la memoria

_ Las restricciones de diseño en la memoria de un computador se pueden resumir en la capacidad, la velocidad y su coste. Si se dispone de una determinada capacidad, probablemente se desarrollarán aplicaciones que la usarán. La cuestión acerca de la velocidad, para alcanzar un rendimiento máximo, la memoria debe ser capaz de mantener el ritmo del procesador. Es decir, según el procesador va ejecutando instrucciones, no debería haber pausas esperando que estén disponibles las instrucciones o los operandos. Y por último, el coste de la memoria debe ser razonable en relación con los otros componentes.

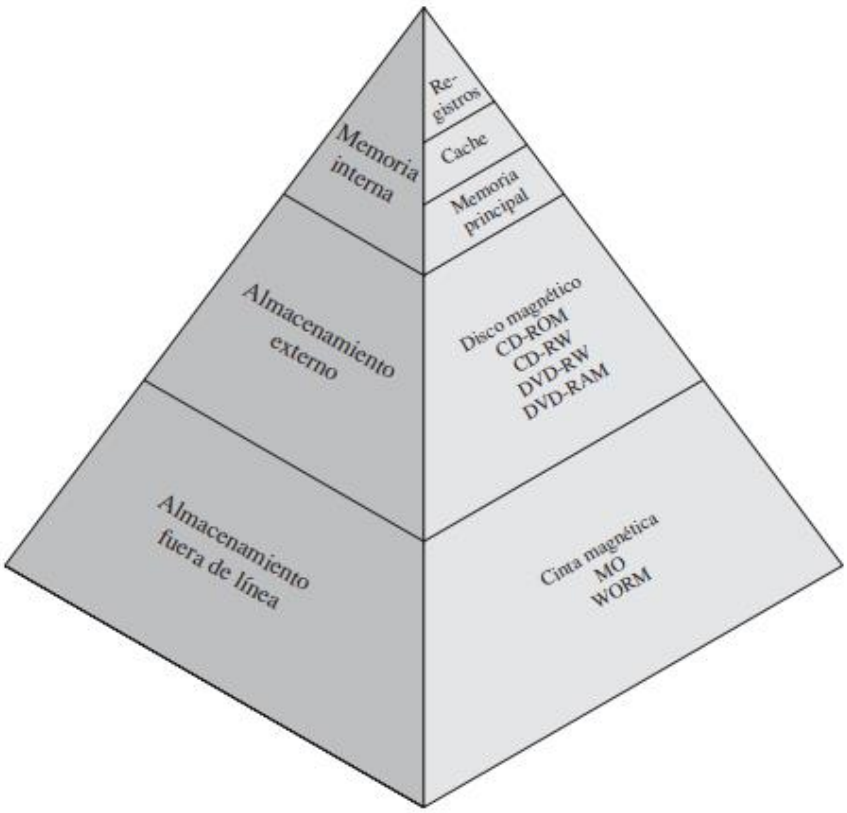
_ En cualquier momento dado, se utilizan diversas tecnologías para implementar los sistemas de memoria. En todo este espectro de tecnologías, se cumplen las siguientes relaciones:

- Cuanto menor tiempo de acceso, mayor coste por bit.
- Cuanto mayor capacidad, menor coste por bit.
- Cuanto mayor capacidad, menor velocidad de acceso.

_ La solución a este dilema consiste en no basarse en un único componente de memoria o en una sola tecnología, sino emplear una jerarquía de memoria como se ve en la pirámide. En donde según se desciende en la jerarquía, ocurre lo siguiente:

- Disminución del coste por bit.
- Aumento de la capacidad.
- Aumento del tiempo de acceso.
- Disminución de la frecuencia de acceso a la memoria por parte del procesador.

_ Por tanto, las memorias más rápidas, caras y pequeñas se complementan con memorias más lentas, baratas y grandes.



Memoria cache

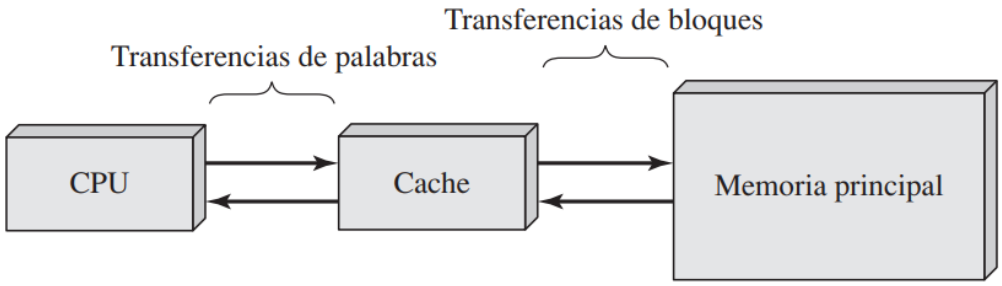
Cache del disco: es una parte de la memoria principal que se puede utilizar como una zona o buffer de almacenamiento intermedio para guardar temporalmente datos que van a ser leídos del disco. Mejora el rendimiento de dos maneras:

- Las escrituras en el disco pueden agruparse. En vez de muchas transferencias de datos de pequeño tamaño, se producen unas pocas transferencias de gran tamaño. Esto mejora el rendimiento del disco y minimiza el grado de implicación del procesador.
- Un programa puede acceder a algunos datos destinados a ser escritos antes del siguiente volcado al disco. En ese caso, los datos se recuperan rápidamente de la cache software en vez de lentamente como ocurre cuando se accede al disco.

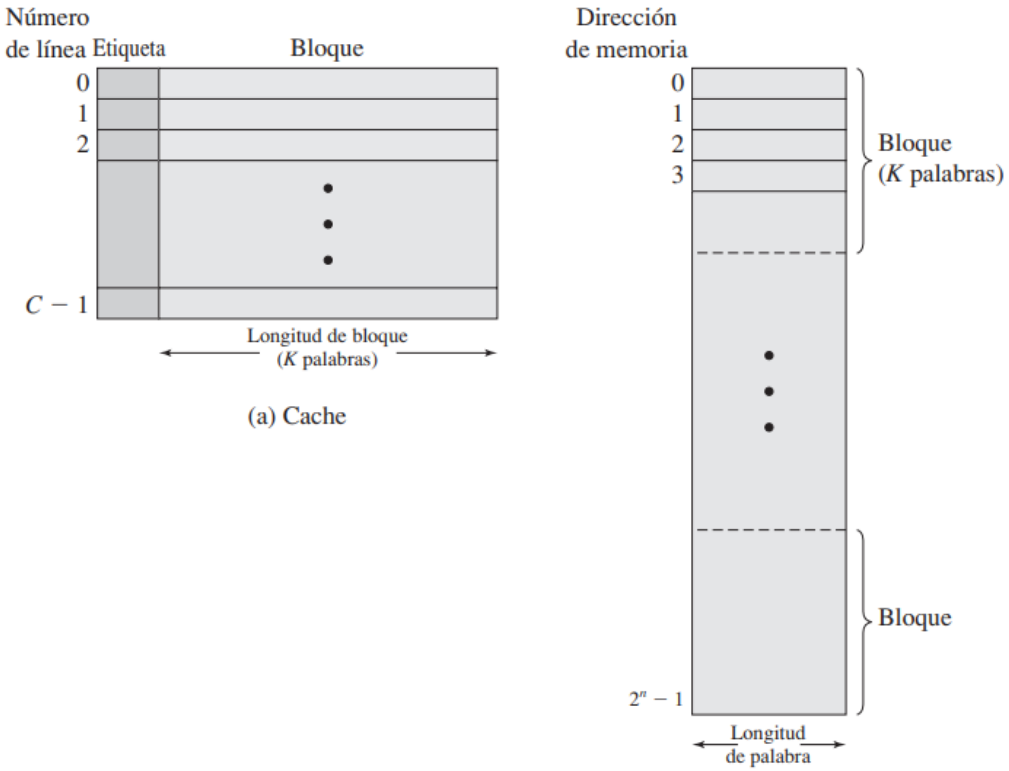
Fundamentos de la cache

_ Aunque la memoria cache es invisible para el sistema operativo, esta interactúa con otros elementos del hardware de gestión de memoria. Es una memoria pequeña y rápida entre el procesador y la memoria principal, cuyo propósito es proporcionar un tiempo de acceso a memoria próximo al de las memorias más rápidas disponibles y, al mismo tiempo, ofrecer un tamaño de memoria grande que tenga el precio de los tipos de memorias de semiconductores menos costosas. La cache contiene una copia de una parte de la memoria principal. Cuando el procesador intenta leer un byte de la

memoria, se hace una comprobación para determinar si el byte está en la cache. Si es así, se le entrega el byte al procesador. En caso contrario, se lee e introduce dentro de la cache un bloque de memoria principal, que consta de un cierto número fijo de bytes, y, a continuación, se le entrega el byte pedido al procesador.



_ A continuación, vemos un gráfico de la estructura de cache/memoria principal:



Diseño de la cache

_ Los elementos fundamentales en el diseño son los siguientes:

Tamaño de la cache: una cache de un tamaño razonablemente pequeño puede tener un impacto significativo en el rendimiento.

Tamaño del bloque: es la unidad de datos que se intercambia entre la cache y la memoria principal. Según el tamaño del bloque se incrementa desde muy pequeño a tamaños mayores, al principio la tasa de aciertos aumentará. La tasa de aciertos significa que la información se encontró en la cache. Según se incrementa el tamaño de bloque, se llevan a la cache más datos útiles. Sin embargo, la tasa de aciertos comenzará a decrecer cuando el tamaño del bloque siga creciendo, ya que la probabilidad de volver a usar los datos recientemente leídos se hace menor que la de utilizar nuevamente los datos que se van a expulsar de la cache.

Función de correspondencia: cuando se lee e incluye un nuevo bloque de datos en la cache, la función de correspondencia determina qué posición de la cache ocupará el bloque. Existen dos restricciones que afectan al diseño de la función de correspondencia:

- En primer lugar, cuando se introduce un bloque en la cache, se puede tener que remplazar otro. Sería deseable hacer esto de manera que se minimizara la probabilidad de que se remplazase un bloque que se necesitara en el futuro inmediato. Cuanto más flexible es la función de correspondencia, mayor grado de libertad a la hora de diseñar un algoritmo de remplazo que maximice la tasa de aciertos.
- En segundo lugar, cuanto más flexible es la función de correspondencia, más compleja es la circuitería requerida para buscar en la cache y determinar si un bloque dado está allí.

Algoritmo de remplazo: selecciona qué bloque remplazar cuando un nuevo bloque va a cargarse en la cache y ésta tiene todos los huecos llenos con otros bloques. Sería deseable remplazar el bloque que probablemente menos se va a necesitar de nuevo. Aunque es imposible identificar tal bloque, una estrategia razonablemente eficiente es remplazar el bloque que ha estado en la cache durante más tiempo sin haberse producido ninguna referencia a él, y esto hace referencia al:

- Algoritmo del menos recientemente usado (Least Recently Used, LRU): en donde se necesitan mecanismos hardware para identificar el bloque menos recientemente usado. Si se altera el contenido de un bloque en la cache, es necesario volverlo a escribir en la memoria principal antes de remplazarlo.

Política de escritura: dicta cuando tiene lugar la operación de escritura en memoria. Una alternativa es que la escritura se produzca cada vez que se actualiza el bloque. Otra opción es que la escritura se realice sólo cuando se remplaza el bloque. La última estrategia minimiza las operaciones de escritura en memoria pero deja la memoria principal temporalmente en un estado obsoleto.

Técnicas de comunicación de E/S

_ Hay tres técnicas para llevar a cabo las operaciones de E/S:

E/S Programada

_ Cuando el procesador ejecuta un programa y encuentra una instrucción relacionada con la E/S, ejecuta esa instrucción generando un mandato al módulo de E/S apropiado. En el caso de la E/S programada, el módulo de E/S realiza la acción solicitada y fija los bits correspondientes en el registro de estado de E/S, pero no realiza ninguna acción para avisar al procesador, es decir, no interrumpe al procesador. Por lo tanto, después de que se invoca la instrucción de E/S, el procesador debe tomar un papel activo para determinar cuándo se completa la instrucción de E/S. Por este motivo, el procesador comprueba periódicamente el estado del módulo de E/S hasta que encuentra que se ha completado la operación. Con esta técnica, el procesador es responsable de extraer los datos de la memoria principal en una operación de salida y de almacenarlos en ella en una operación de entrada. Se incluye instrucciones de E/S de las siguientes categorías:

- Control: utilizadas para activar un dispositivo externo y especificarle qué debe hacer.
- Estado: utilizadas para comprobar diversas condiciones de estado asociadas a un módulo de E/S y sus periféricos.
- Transferencia: utilizadas para leer y/o escribir datos entre los registros del procesador y los dispositivos externos.

E/S Dirigida por interrupciones

_ El problema de esta técnica es que el procesador tiene que esperar mucho tiempo hasta que el módulo de E/S correspondiente esté listo para la recepción o la transmisión de más datos. El procesador, mientras está esperando, debe comprobar repetidamente el estado del módulo de E/S, generando que se degrade gravemente el nivel de rendimiento de todo el sistema.

_ Una alternativa es que el procesador genere un mandato de E/S para un módulo y luego continúe realizando otras operaciones. Después, el módulo de E/S va a interrumpir al procesador para solicitar su servicio cuando esté listo para intercambiar datos con el mismo, en donde el procesador ejecutará la transferencia de datos y después reanudará el procesamiento previo. Para entender cómo funciona esto:

- Desde el punto de vista del módulo de E/S, para una operación de entrada, el módulo de E/S recibe una orden de lectura del procesador, luego pasa a leer los datos de un periférico asociado. Una vez que los datos están en el registro de datos del módulo, el módulo genera una interrupción al procesador a través de una línea de control. El módulo entonces espera hasta que el procesador pida sus datos y cuando se hace la petición, el módulo sitúa sus datos en el bus de datos y ya está listo para otra operación de E/S.

- Desde el punto de vista del procesador, el mismo genera una orden de lectura. Guarda lo que estaba haciendo del programa actual y lo abandona, pasando a hacer otra cosa. Al final de cada ciclo de instrucción, el procesador comprueba si hay interrupciones y elimina esperas innecesarias. Cuando se produce la interrupción del módulo de E/S, el procesador guarda el contexto del programa que se está ejecutando actualmente y comienza a ejecutar un programa de manejo de interrupción que procesa la interrupción. Luego restaura el contexto del programa que había realizado el mandato de E/S (o de algún otro programa) y reanuda su ejecución.

Acceso directo a memoria

_ Esta técnica es más eficiente que la E/S programada simple, pero requiere de la intervención activa del procesador para transferir datos entre la memoria y un módulo de E/S, ya que cualquier transferencia de datos debe pasar a través del procesador. Por lo tanto, ambas formas de E/S sufren dos inconvenientes:

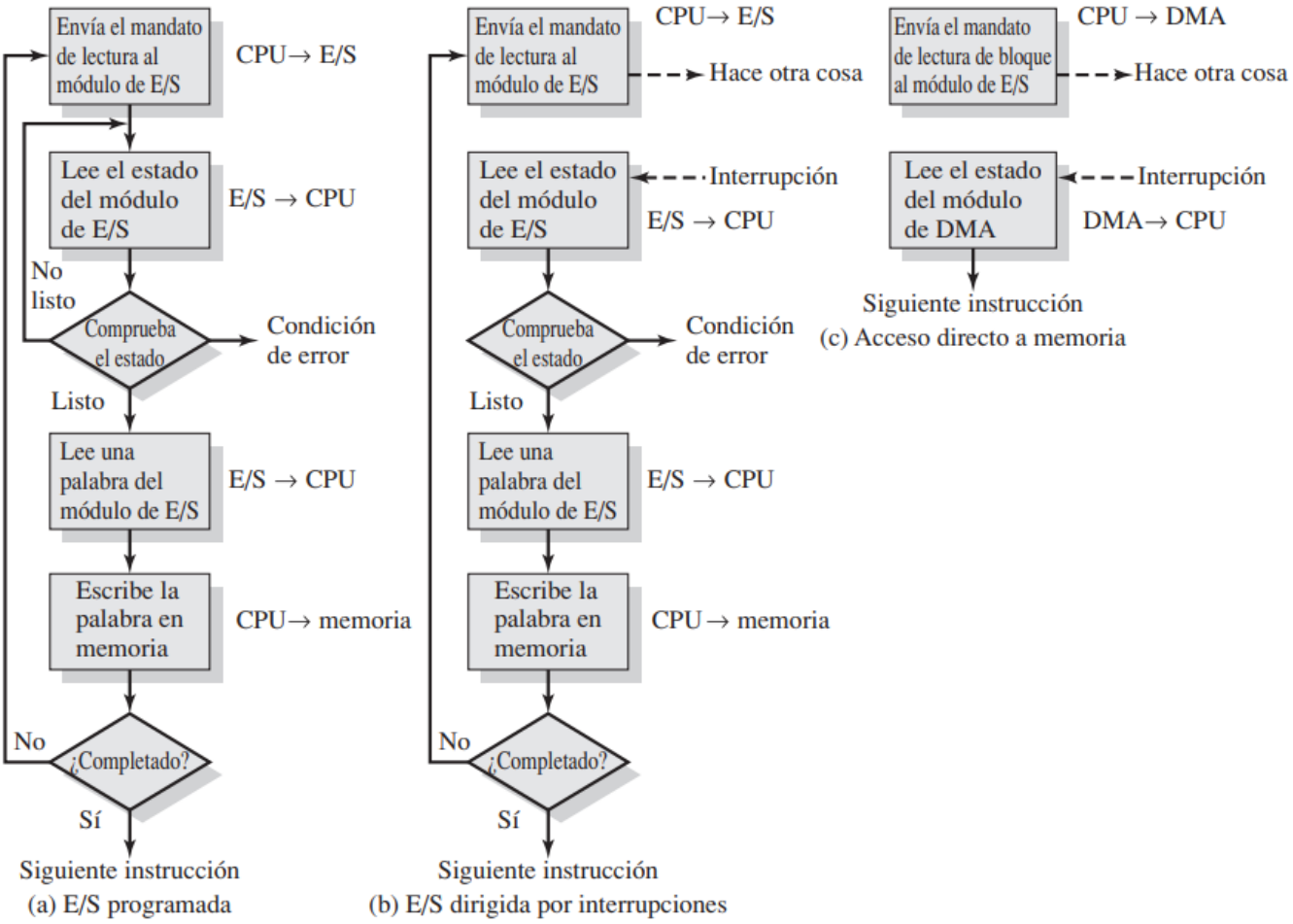
1)_ La tasa de transferencia de E/S está limitada por la velocidad con la que el procesador puede comprobar el estado de un dispositivo y ofrecerle servicio.

2)_ El procesador está involucrado en la gestión de una transferencia de E/S. Cuando se van a transferir grandes volúmenes de datos, se requiere una técnica más eficiente:

- Acceso directo a memoria (Direct Memory Access, DMA): cuando el procesador desea leer o escribir un bloque de datos, genera un mandato al módulo de DMA, enviándole la siguiente información:
  - Si se trata de una lectura o de una escritura.
  - La dirección del dispositivo de E/S involucrado.
  - La posición inicial de memoria en la que se desea leer los datos o donde se quieren escribir.
  - El número de palabras que se pretende leer o escribir.

_ A continuación, el procesador continúa con otro trabajo y ha delegado esta operación de E/S al módulo de DMA, en donde este transferirá el bloque completo de datos, palabra a palabra, hacia la memoria o desde ella sin pasar a través del procesador. Por tanto, el procesador solamente está involucrado al principio y al final de la transferencia.

_ A continuación vemos un diagrama de las tres técnicas para leer un bloque de datos:



Introducción a los sistemas operativos

Introducción

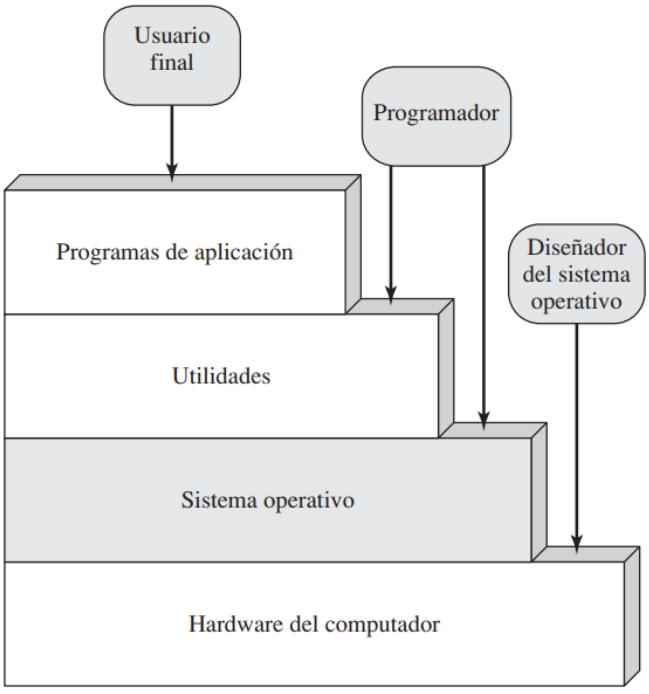
Sistema operativo: es un programa que controla la ejecución de aplicaciones y programas y que actúa como interfaz entre las aplicaciones y el hardware del computador.

Objetivos y funciones de los sistemas operativos

_ Un sistema operativo tiene los siguientes tres objetivos:

- Facilidad de uso: facilita el uso de un computador.
- Eficiencia: permite que los recursos de un sistema de computación se puedan utilizar de una manera eficiente.
- Capacidad para evolucionar: este se debe construir de tal forma que se puedan desarrollar, probar e introducir nuevas funciones en el sistema sin interferir con su servicio.

_ En el siguiente diagrama vemos en forma de capas como el hardware y el software son utilizados para proporcionar aplicaciones a los usuarios. El usuario de dichas aplicaciones, es decir, el usuario final, no se preocupa por los detalles del hardware del computador, por ende este solo ve un sistema de computación en términos de un conjunto de aplicaciones. Una aplicación se puede expresar en un lenguaje de programación y es desarrollada por un programador. Para evitar que un programador tenga que desarrollar una aplicación mediante instrucciones en código máquina que controlen el hardware del computador, ya que es muy complejo, directamente se proporcionan un conjunto de programas de sistema conocidos como utilidades que asisten al programador en las fases de creación de programas, gestión de ficheros y control de los dispositivos de E/S. Entonces, un programador hará uso de estas utilidades cuando desarrolle una aplicación, y las aplicaciones, invocarán a las utilidades durante su ejecución para llevar a cabo ciertas funciones.



_ El programa de sistema o utilidad más importante es el sistema operativo. El sistema operativo oculta los detalles del hardware al programador y le proporciona una interfaz apropiada para utilizar el sistema. Actúa como mediador, haciendo más fácil al programador y a la aplicación el acceso y uso de dichas utilidades y servicios.

Servicios que ofrece el sistema operativo

_ El sistema operativo proporciona normalmente servicios en las siguientes áreas:

Desarrollo de programas: se hace referencia a editores y depuradores (debuggers), para asistir al programador en la creación de los programas. Estas utilidades se conocen como herramientas de desarrollo de programas de aplicación.

Ejecución de programas: se necesita realizar una serie de pasos para ejecutar un programa, en donde las instrucciones y los datos se deben cargar en la memoria principal, los dispositivos de E/S y los ficheros se deben inicializar, y otros recursos deben prepararse. Por lo que, los sistemas operativos realizan estas labores de planificación en nombre del usuario.

Acceso a dispositivos de E/S: cada dispositivo de E/S requiere su propio conjunto de instrucciones para cada operación, por ende, el sistema operativo proporciona una interfaz que esconde esos detalles de forma que los programadores puedan acceder a dichos dispositivos utilizando lecturas y escrituras simples.

Acceso controlado a los ficheros: para el acceso a los ficheros, el sistema operativo debe reflejar una comprensión detallada de la naturaleza del dispositivo de E/S (disco, cinta) y también de la estructura de los datos contenidos en los ficheros del sistema de almacenamiento. Además, puede proporcionar mecanismos de protección para controlar el acceso a los ficheros.

Acceso al sistema: para sistemas compartidos o públicos, el sistema operativo controla el acceso al sistema completo y a recursos específicos del sistema. La función de acceso debe proporcionar protección a los recursos y a los datos, evitando el uso no autorizado de los usuarios y resolviendo conflictos de recursos.

Detección y respuesta a errores: ante la presencia de errores durante la ejecución de un sistema de computación, como errores de hardware internos y externos, por ejemplo un error de memoria, un fallo en un dispositivo, o como errores software, por ejemplo la división por cero, el intento de acceder a una posición de memoria prohibida, o la incapacidad del sistema operativo para conceder la solicitud de una aplicación, para estos casos el sistema operativo debe proporcionar una respuesta que elimine la condición de error y que genere el menor impacto en las aplicaciones que están en ejecución. La respuesta a esto puede ser o finalizar el programa que causó el error hasta reintentar la operación o simplemente informar del error a la aplicación.

Contabilidad: el sistema operativo puede recoger estadísticas de uso de los recursos y puede monitorear parámetros de rendimiento, como el tiempo de respuesta. Esta información es útil para anticipar las necesidades de mejoras futuras y para optimizar

el sistema a fin de mejorar su rendimiento. En un sistema multiusuario, esta información se puede utilizar para facturar a los diferentes usuarios.

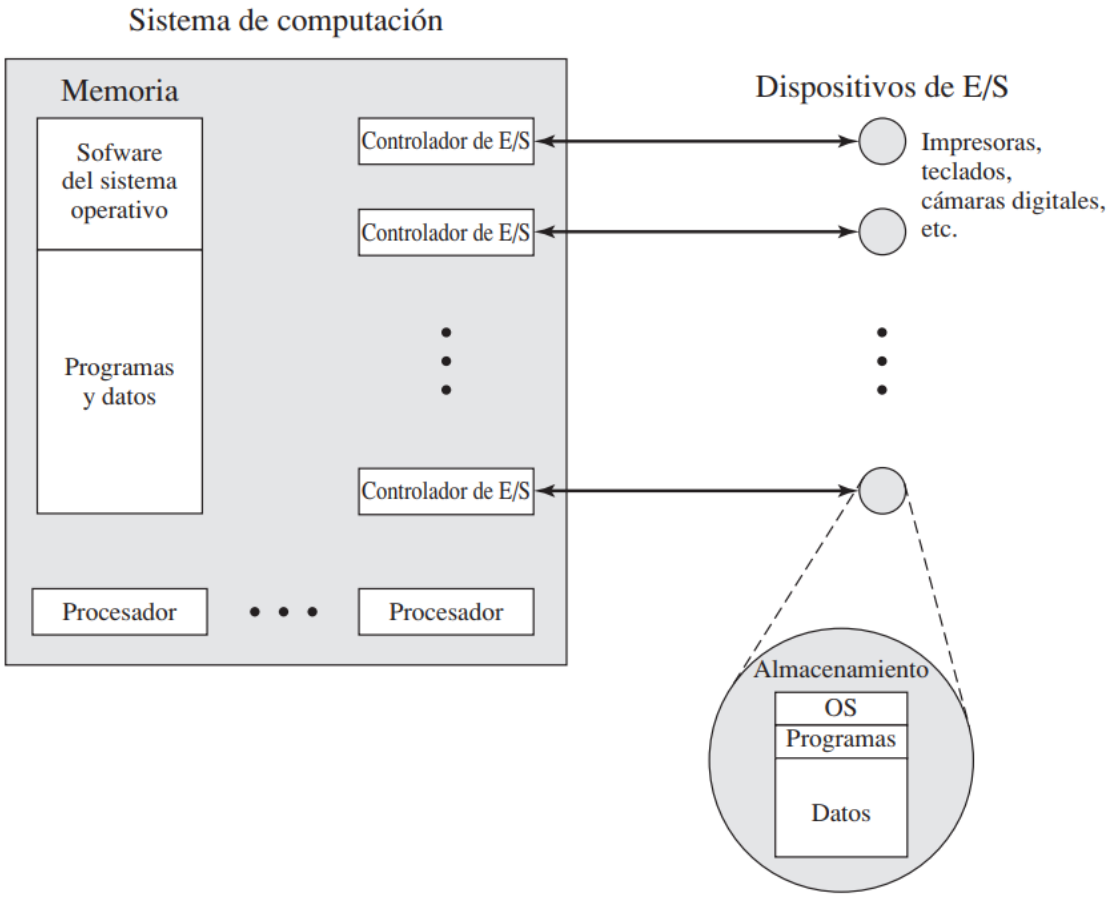
Sistema operativo como gestor de recursos

_ Un computador es un conjunto de recursos que se utilizan para el transporte, almacenamiento y procesamiento de los datos, así como también el control de estas funciones. Entonces, el sistema operativo se encarga de gestionar al computador y tiene el control de las funciones básicas del mismo. Para el control del transporte, el sistema operativo tiene dos aspectos:

- Las funciones del sistema operativo actúan de la misma forma que el resto del software; es decir, se trata de un programa o conjunto de programas ejecutados por el procesador.
- El sistema operativo cede el control y depende del procesador para volver a retomarlo.

_ El sistema operativo, es un conjunto de programas que proporciona instrucciones al procesador, en el que además dirige a este último en el uso de los otros recursos del sistema y en la temporización de la ejecución de otros programas. Pero para que el procesador pueda realizar esto, el sistema operativo debe dejar paso a la ejecución de otros programas, por lo tanto, el sistema operativo deja el control para que el procesador pueda realizar trabajo “útil”, y después de esto retoma el control nuevamente para permitir al procesador siga trabajando.

_ En el siguiente diagrama vemos los principales recursos gestionados por el sistema operativo:



_ Parte del sistema operativo se encuentra en la memoria principal, y esto incluye el kernel, o núcleo, que contiene las funciones del sistema operativo utilizadas con más frecuencia. El sistema operativo y el hardware de gestión de memoria del procesador controlan la asignación de recursos de la memoria principal. Además, el sistema operativo decide cuándo un programa que está en ejecución puede utilizar un dispositivo de E/S, controla el acceso y uso de los ficheros, y debe determinar cuánto tiempo de procesador (que también es un recurso) debe asignarse a la ejecución de un programa de usuario particular.

Evolución de los sistemas operativos

_ Un sistema operativo importante debe evolucionar en el tiempo por las siguientes razones:

- Actualizaciones de hardware más nuevos tipos de hardware.
- Nuevos servicios: en respuesta a la demanda del usuario o en respuesta a las necesidades de los gestores de sistema.
- Resolución o corrección de fallos: ya que cualquier sistema operativo tiene fallos que se descubren con el transcurso del tiempo, y se busca resolverlos.

Procesamiento serie

_ A finales de los años 40 hasta mediados de los años 50, el programador interaccionaba directamente con el hardware del computador ya que no existía ningún sistema operativo. Estas máquinas eran utilizadas desde una consola que contenía luces, interruptores, algún dispositivo de entrada y una impresora. Los programas en código máquina se cargaban a través del dispositivo de entrada y si un error provocaba la parada del programa, las luces indicaban la condición de error. El programador podía entonces examinar los registros del procesador y la memoria principal para determinar la causa de error. Si el programa terminaba de forma normal, la salida aparecía en la impresora. Estos sistemas iniciales presentaban dos problemas principales:

- Planificación: se malgastaba tiempo de procesamiento del computador, al momento de reservar tiempo de máquina para un usuario, en donde este podía tener problemas, ya que si no finalizaba en el tiempo asignado y era forzado a terminar antes de resolver el problema.
- Tiempo de configuración: la preparación incluía cargar un compilador, luego un programa fuente, después salvar el programa compilado y, por último, cargar y montarlo. Si ocurría un error, el usuario tenía que volver al comienzo de la secuencia de configuración. Por lo tanto, se utilizaba una cantidad considerable de tiempo en configurar el programa que se iba a ejecutar.

_ Este modo de operación puede denominarse procesamiento serie, para reflejar el hecho de que los usuarios acceden al computador en serie.

Sistemas en lotes sencillos

_ Las primeras máquinas eran muy caras, y por lo tanto, era importante maximizar su utilización, porque el tiempo malgastado en la planificación y configuración de los trabajos era inaceptable, asi que para mejorar su utilización, se desarrolló el concepto de sistema operativo en lotes, en donde la idea central bajo este esquema de procesamiento es el uso de una pieza de software llamada monitor. Entonces, con este tipo de sistema operativo, el usuario no accede directamente a la máquina sino que envía un trabajo a través de una tarjeta o cinta al operador del computador, que crea un sistema por lotes con todos los trabajos enviados y coloca la secuencia de trabajos en el dispositivo de entrada, para que lo utilice el monitor. Cuando un programa finaliza su procesamiento, devuelve el control al monitor, el cual comienza la carga del siguiente programa.

- El monitor controla la secuencia de eventos. Una gran parte del monitor, llamada monitor residente, debe estar siempre en memoria principal y disponible para la ejecución. El resto del monitor está formado por un conjunto de utilidades y funciones comunes que se cargan como subrutinas en el programa de usuario, al comienzo de cualquier trabajo que las requiera. El monitor lee de uno en uno los trabajos desde el dispositivo de entrada, luego el trabajo actual se coloca en el área de programa de usuario, y se le pasa el control. Cuando el trabajo se ha completado, devuelve el control al monitor, que inmediatamente lee el siguiente trabajo. Por ultimo los resultados de cada trabajo se envían a un dispositivo de salida, (por ejemplo, una impresora), para entregárselo al usuario.
- El procesador posee dos fases, una en la que se pasa el control al trabajo, que simplemente significa que el procesador leerá y ejecutará instrucciones del programa de usuario, en la memoria principal, y la otra frase es que se devuelve el control al monitor, haciendo referencia a que el procesador leerá y ejecutará instrucciones del programa monitor.

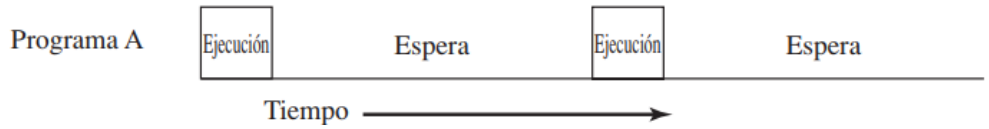
Lenguaje de control de trabajos (Job Control Language, JCL): es un tipo especial de lenguaje de programación utilizado para dotar de instrucciones al monitor. Ya que el monitor realiza una función de planificación que consiste en que en una cola se sitúa un lote de trabajos, y los trabajos se ejecutan lo más rápidamente posible, sin ninguna clase de tiempo ocioso entre medias. Además, el monitor mejora el tiempo de configuración de los trabajos. Con cada uno de los trabajos, se incluye un conjunto de instrucciones en algún formato primitivo de JCL. Este lenguaje es empleado para dar instrucciones al monitor, como por ejemplo que compilador utilizar o que datos utilizar.

Características del hardware: el monitor, o sistema operativo en lotes, es simplemente un programa el cual confía en la habilidad del procesador para cargar instrucciones de diferentes porciones de la memoria principal que de forma alternativa le permiten tomar y abandonar el control. Otras características de hardware que son deseables:

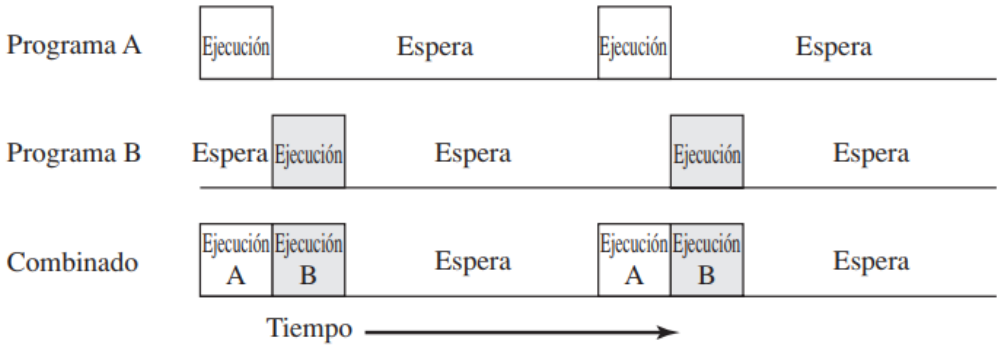
- Protección de memoria: durante la ejecución del programa de usuario, éste no debe alterar el área de memoria que contiene el monitor. Si esto ocurriera, el hardware del procesador debe detectar un error y transferir el control al monitor, en donde este último abortará el trabajo, imprimirá un mensaje de error y cargará el siguiente trabajo.
- Temporizador: se utiliza un temporizador para evitar que un único trabajo monopolice el sistema. Este se activa al comienzo de cada trabajo, y si el temporizador expira, se para el programa de usuario, y se devuelve el control al monitor.
- Instrucciones privilegiadas: ciertas instrucciones a nivel de máquina se denominan privilegiadas y sólo las puede ejecutar el monitor. Si el procesador encuentra estas instrucciones mientras ejecuta un programa de usuario, se produce un error provocando que el control se transfiera al monitor.
- Interrupciones: esta característica proporciona al sistema operativo más flexibilidad para dejar y retomar el control desde los programas de usuario.

Sistemas en lotes multiprogramados

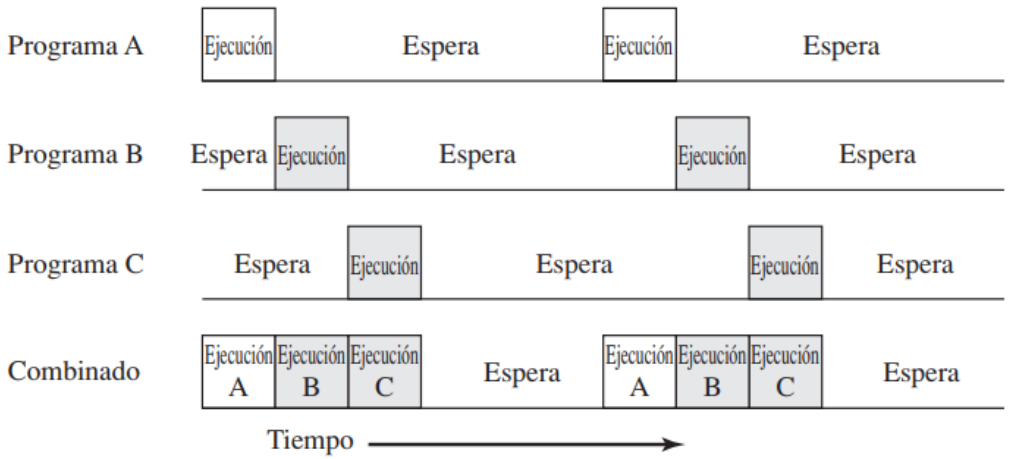
Monoprogramación: en este caso existe un único programa, en donde el procesador se ejecuta durante cierto tiempo hasta que alcanza una instrucción de E/S. Entonces, este debe esperar que la instrucción de E/S concluya antes de continuar. Esto es ineficiente.



Multiprogramación: es el tema central de los sistemas operativos modernos y hace referencia a que como existe suficiente memoria para contener al sistema operativo (monitor residente) y un programa de usuario, suponemos que hay espacio para el sistema operativo y para dos programas de usuario, en donde cuando un trabajo necesita esperar por la E/S, se puede asignar el procesador al otro trabajo, que probablemente no esté esperando por una operación de E/S. A continuación lo vemos:



_ También se puede expandir la memoria para que albergue tres, cuatro o más programas y pueda haber multiplexación entre todos ellos como vemos a continuación:



_ Estos sistemas operativos son bastante sofisticados, comparados con los sistemas monoprogramados. A continuación, tenemos un ejemplo de la ejecución de programas:

	TRABAJO 1	TRABAJO 2	TRABAJO 3
Tipo de trabajo	Computación pesada	Gran cantidad de E/S	Gran cantidad de E/S
Duración	5 minutos	15 minutos	10 minutos
Memoria requerida	50 M	100 M	75 M
¿Necesita disco?	No	No	Sí
¿Necesita terminal?	No	Sí	No
¿Necesita impresora?	No	No	Sí

_ Observamos en la siguiente tabla los efectos de la utilización de recursos sobre la multiprogramación:

	Monoprogramación	Multiprogramación
Uso de procesador	20%	40%
Uso de memoria	33%	67%
Uso de disco	33%	67%
Uso de impresora	33%	67%
Tiempo transcurrido	30 minutos	15 minutos
Productividad	6 trabajos/hora	12 trabajos/hora
Tiempo de respuesta medio	18 minutos	10 minutos

Sistemas de tiempo compartido

_ Es necesario proporcionar un modo interactivo en el cual el usuario interaccione directamente con el computador para algunos trabajos como el procesamiento de transacciones. Hoy en día, los computadores personales dedicados o estaciones de trabajo poseen este modo pero en los años 60, no existía y se desarrolló el concepto de tiempo compartido.

_ La multiprogramación permite al procesador gestionar múltiples trabajos en lotes en un determinado tiempo y además permite gestionar múltiples trabajos interactivos, y es esto último lo que hace referencia a la técnica de tiempo compartido, porque se comparte el tiempo de procesador entre múltiples usuarios. En este sistema, múltiples usuarios acceden simultáneamente al mismo a través de terminales, siendo el sistema operativo el encargado de entrelazar la ejecución de cada programa de usuario en pequeños intervalos de tiempo. Por tanto, si hay n usuarios activos solicitando un servicio a la vez, cada usuario sólo verá 1/n de la capacidad de computación efectiva, sin contar la sobrecarga introducida por el sistema operativo.

_ Tanto el procesamiento en lotes como el tiempo compartido, utilizan multiprogramación. Las diferencias más importantes son:

	Multiprogramación en lotes	**Tiempo compartido**
Objetivo principal	Maximizar el uso del procesador	Minimizar el tiempo de respuesta
Fuente de directivas al sistema operativo	Mandatos del lenguaje de control de trabajos proporcionados por el trabajo	Mandatos introducidos al terminal

CTSS (Compatible Time-Sharing System): fue uno de los primeros sistemas operativos de tiempo compartido desarrollados. La técnica utilizada por este sistema es primitiva comparada con las técnicas de tiempo compartido actuales, pero era muy sencilla, porque minimizaba el tamaño del monitor debido a que un trabajo siempre se cargaba en la misma dirección de memoria, y no había necesidad de utilizar técnicas de reubicación en tiempo de carga.

Principales logros

_ Los sistemas operativos se encuentran entre las piezas de software más complejas jamás desarrolladas. Esto refleja el reto de intentar resolver la dificultad de alcanzar determinados objetivos, algunas veces conflictivos, de conveniencia, eficiencia y capacidad de evolución. A continuación analizamos cinco principales avances teóricos en el desarrollo de los sistemas operativos:

Procesos

_ El concepto de proceso es fundamental en la estructura de los sistemas operativos. Es un término un poco más general que el de trabajo, cuya definición hace referencia a:

- Un programa en ejecución.
- Una instancia de un programa ejecutándose en un computador.
- La entidad que se puede asignar o ejecutar en un procesador.
- Una unidad de actividad caracterizada por un solo hilo secuencial de ejecución, un estado actual, y un conjunto de recursos del sistema asociados.

_ Se puede considerar que un proceso está formado por los siguientes tres componentes:

- Un programa ejecutable.
- Los datos asociados que necesita el programa, como variables, espacio de trabajo, buffers, etc.
- El contexto de ejecución del programa, o estado del proceso, que es el conjunto de datos interno por el cual el sistema operativo es capaz de supervisar y controlar el proceso.

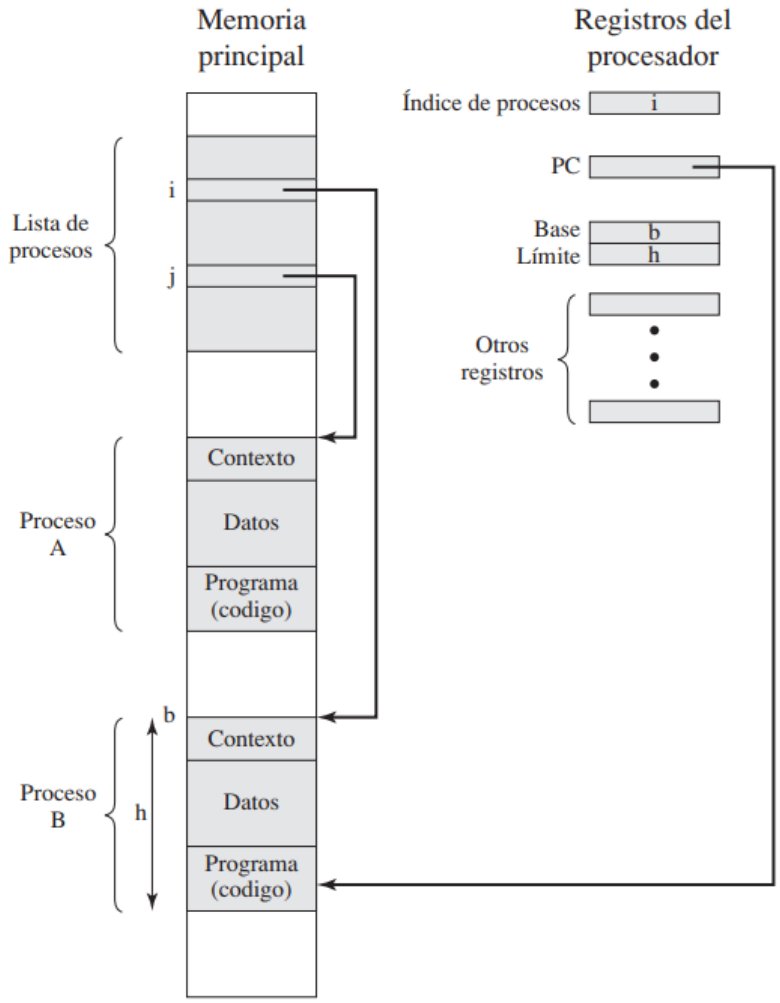
Dificultades en el diseño del software:

_ El diseño del software del sistema para coordinar diversas actividades fue muy difícil. Con la progresión simultánea de muchos trabajos, cada uno de los cuales suponía la realización de numerosos pasos para su ejecución secuencial, era imposible analizar todas las posibles combinaciones de secuencias de eventos. Por lo que si surgían errores, estos eran difíciles de diagnosticar, porque tenían que distinguirse de los errores software y hardware de las aplicaciones. Incluso cuando se detectaba el error, era difícil determinar la causa, porque las condiciones precisas bajo las cuales el error aparecía, eran difíciles de reproducir. En términos generales, existen cuatro causas principales de dichos errores:

- Inapropiada sincronización: es frecuente el caso de que una rutina se suspenda esperando por algún evento en el sistema. En este caso, se necesita una señal procedente de otra rutina, por lo que el diseño inapropiado del mecanismo de señalización puede provocar que las señales se pierdan o se reciban señales duplicadas.

- Violación o fallos de la exclusión mutua: frecuentemente, más de un programa o usuario intentan hacer uso de recursos compartidos simultáneamente, si estos accesos no se controlan, podría ocurrir un error, por ende debe existir algún tipo de mecanismo de exclusión mutua que permita que sólo una rutina en un momento determinado actualice un fichero. Es difícil de verificar que la implementación de este mecanismo sea correcta.
- Funcionamiento no determinista de un programa: los resultados de un programa particular normalmente dependen sólo de la entrada a dicho programa y no de las actividades de otro programa en un sistema compartido, pero cuando los programas comparten memoria, y sus ejecuciones son entrelazadas por el procesador, podrían interferir entre ellos, sobrescribiendo zonas de memoria comunes. Por lo tanto, el orden en el que diversos programas se planifican puede afectar a la salida de cualquier programa particular.
- Interbloqueos: es posible que dos o más programas se queden bloqueados esperándose entre sí. Por ejemplo, dos programas podrían requerir dos dispositivos de E/S para llevar a cabo una determinada, luego uno de los programas ha tomado control de uno de los dispositivos y el otro programa tiene control del otro dispositivo. Cada uno de ellos está esperando a que el otro programa libere el recurso que no poseen. Este interbloqueo puede depender de la temporización de la asignación y liberación de recursos.

<u>Implementación de procesos típica:</u>



Gestión de memoria

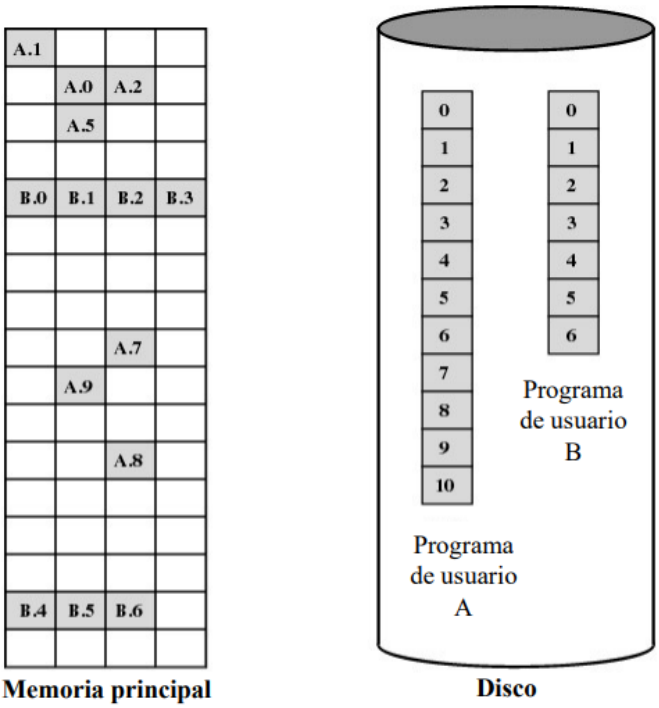
_ Los gestores de sistema necesitan un control eficiente y ordenado de la asignación de los recursos. Para satisfacer estos requisitos, el sistema operativo tiene cinco responsabilidades principales de gestión de almacenamiento:

- Aislamiento de procesos: el sistema operativo debe evitar que los procesos independientes interfieran en la memoria de otro proceso.
- Asignación y gestión automática: los programas deben tener una asignación dinámica de memoria por demanda, en cualquier nivel jerárquico de la memoria, en donde esta asignación debe ser transparente al programador para que no se preocupe de aspectos relacionados con limitaciones de memoria, y asi el sistema operativo puede incrementar la eficiencia, asignando memoria a los trabajos sólo cuando se necesiten.
- Soporte a la programación modular: los programadores deben ser capaces de definir módulos de programación y crear, destruir, y alterar el tamaño de los módulos dinámicamente.
- Protección y control de acceso: la compartición de memoria, en cualquier nivel de la jerarquía de memoria, permite que un programa direccione un espacio de memoria de otro proceso. Esto es deseable cuando se necesita la compartición por parte de determinadas aplicaciones, pero también puede amenazar la integridad de los programas e incluso del propio sistema operativo.
- Almacenamiento a largo plazo: muchas aplicaciones requieren formas de almacenar la información durante largos periodos de tiempo, después de que el computador se haya apagado.

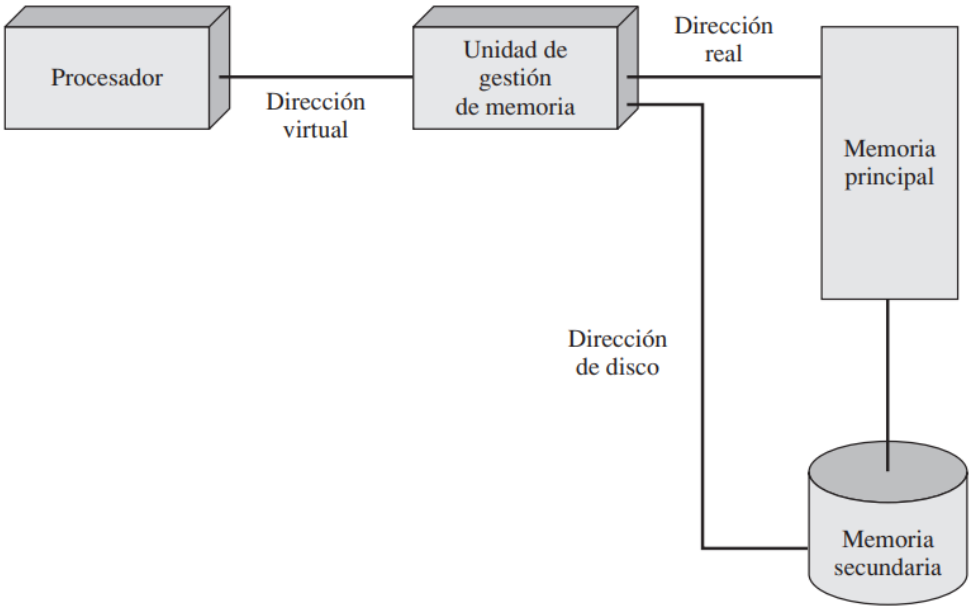
Sistema de archivos o ficheros: el sistema operativo implementa un almacenamiento a largo plazo, con la información almacenada en objetos denominados ficheros. El fichero es un concepto lógico, conveniente para el programador y es una unidad útil de control de acceso y protección para los sistemas operativos.

Memoria virtual: es una utilidad que permite a los programas direccionar la memoria desde un punto de vista lógico, sin importar la cantidad de memoria principal física disponible. La memoria virtual fue concebida como un método para tener múltiples trabajos de usuario residiendo en memoria principal de forma concurrente, y de forma que no exista un intervalo de tiempo de espera entre la ejecución de procesos sucesivos, es decir, mientras un proceso se escribe en almacenamiento secundario y se lee el proceso sucesor.

- La memoria principal está formada por un número de marcos de longitud fija del tamaño de una página. Para ejecutar un programa, algunas o todas sus páginas tienen que estar en el operativo de la memoria principal.
- La memoria secundaria (disco) puede almacenar muchas páginas de longitud fija. Un programa de usuario está formado por varias páginas. Las páginas de todos los programas, incluso los del sistema están en el disco, en forma de archivos.



Direccionamiento de la memoria virtual: respecto a esto, tenemos que el almacenamiento está compuesto por la memoria principal directamente direccionable (a través de instrucciones máquina) y una memoria auxiliar de baja velocidad a la que se accede de forma indirecta, cargando bloques de la misma en memoria principal. El hardware de traducción de direcciones (memory management unit: unidad de gestión de memoria) se interpone entre el procesador y la memoria. Los programas hacen referencia a direcciones virtuales, que son proyectadas sobre direcciones reales de memoria principal. Si una referencia a una dirección virtual no se encuentra en memoria física, entonces una porción de los contenidos de memoria real que son llevados a la memoria auxiliar y los contenidos de la memoria real que se están buscando, son llevados a memoria principal. Durante esta tarea, el proceso que generó la dirección se suspende:



Sistemas de paginación: permiten que los procesos se compriman en un número determinado de bloques de tamaño fijo, llamados páginas. Un programa referencia una palabra por medio de una dirección virtual, que consiste en un número de página y un desplazamiento dentro de la página. Cada página de un proceso se puede localizar en cualquier sitio de memoria principal. El sistema de paginación proporciona una proyección dinámica entre las direcciones virtuales utilizadas en el programa y una dirección real, o dirección física, de memoria principal.

Seguridad y protección de la información: la mayoría del trabajo en seguridad y protección relacionado con los sistemas operativos se puede agrupar en cuatro categorías:

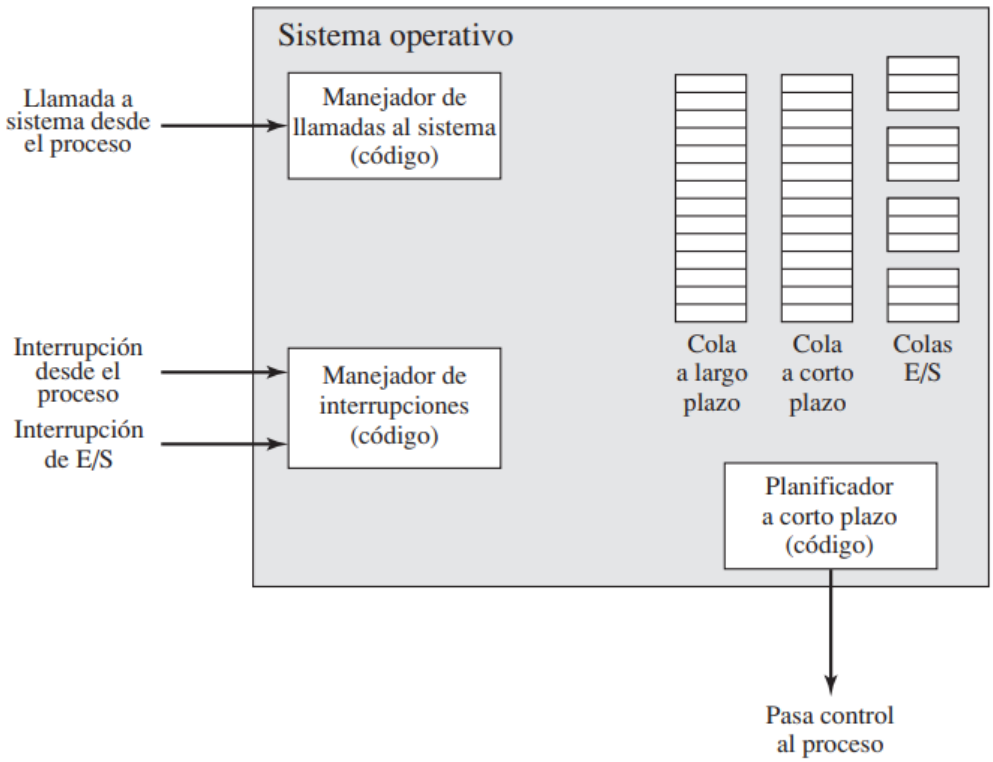
- Disponibilidad: relacionado con la protección del sistema frente a las interrupciones.
- Confidencialidad: asegura que los usuarios no puedan leer los datos sobre los cuales no tienen autorización de acceso.
- Integridad de los datos: protección de los datos frente a modificaciones no autorizadas.
- Autenticidad: relacionado con la verificación apropiada de la identidad de los usuarios y la validez de los mensajes o los datos.

Planificación y gestión de los recursos

_ Una responsabilidad clave de los sistemas operativos es la gestión de varios recursos disponibles para ellos (espacio de memoria principal, dispositivos de E/S, procesadores) y para planificar su uso por parte de los distintos procesos activos. Cualquier asignación de recursos y política de planificación debe tener en cuenta tres factores:

- Equitatividad: se desea que todos los procesos que compiten por un determinado recurso, se les conceda un acceso equitativo a dicho recurso.
- Respuesta diferencial: el sistema operativo puede discriminar entre diferentes clases de trabajos con diferentes requisitos de servicio, y debe tomar las decisiones de asignación y planificación de forma dinámica con el objetivo de satisfacer el conjunto total de requisitos
- Eficiencia: el sistema operativo debe intentar maximizar la productividad, minimizar el tiempo de respuesta, y, en caso de sistemas de tiempo compartido, acomodar tantos usuarios como sea posible.

_ A continuación, vemos los principales elementos del sistema operativo relacionados con la planificación de procesos y la asignación de recursos en un entorno de multiprogramación:



Estructura del sistema

_ La estructura jerárquica de un sistema operativo moderno separa sus funciones de acuerdo a las características de su escala de tiempo y su nivel de abstracción. Se puede ver el sistema como una serie de niveles, en donde cada nivel realiza un subconjunto relacionado de funciones. Cada nivel confía en los niveles inferiores para realizar funciones más primitivas y ocultar los detalles de esas funciones, y además cada uno proporciona servicios a la capa inmediatamente superior. Estos niveles deben definirse de tal forma que los cambios en un nivel no requieran cambios en otros niveles. Por lo tanto, esto ayuda a descomponer un problema en un número de subproblemas más manejables.

Jerarquía de diseño del sistema operativo:

Nivel	Nombre	Objetos	Ejemplos de operaciones
13	Intérprete de mandatos	Entorno de programación de usuario	Sentencias en lenguaje del intérprete de mandatos
12	Procesos de usuario	Procesos de usuario	Salir, matar, suspender, continuar
11	Directorios	Directorios	Crear, destruir, insertar entrada, eliminar entrada, buscar, listar
10	Dispositivos	Dispositivos externos, como impresoras, pantallas y teclados	Abrir, cerrar, leer, escribir
9	Sistema de ficheros	Ficheros	Crear, destruir, abrir, cerrar, leer, escribir
8	Comunicaciones	Tuberías	Crear, destruir, abrir, cerrar, leer, escribir

7	Memoria virtual	Segmentos, páginas	Leer, escribir, cargar
6	Almacenamiento secundario local	Bloques de datos, canales de dispositivo	Leer, escribir, asignar, liberar
5	Procesos primitivos	Procesos primitivos, semáforos, lista de procesos listos	Suspender, continuar, esperar, señalizar
4	Interrupciones	Programas de gestión de interrupciones	Invocar, enmascarar, desenmascarar, reintentar
3	Procedimientos	Procedimientos, pila de llamadas, registro de activación	Marcar la pila, llamar, retornar
2	Conjunto de instrucciones	Pila de evaluación, intérprete de microprogramas, datos escalares y vectoriales	Cargar, almacenar, sumar, restar, saltar
1	Circuitos electrónicos	Registros, puertas, buses, etc.	Poner a 0, transferir, activar, complementar

- Nivel 1: formado por circuitos electrónicos, donde los objetos tratados son registros, celdas de memoria, y puertas lógicas. Las operaciones definidas en estos objetos son acciones, como poner a cero un registro o leer una posición de memoria.
- Nivel 2: formado por el conjunto de instrucciones del procesador. Las operaciones a este nivel son aquéllas permitidas en el conjunto de instrucciones de lenguaje máquina, como adición, resta, carga o almacenamiento.
- Nivel 3: añade el concepto de procedimiento o subrutina, más las operaciones de llamada y retorno (call/return).
- Nivel 4. introduce las interrupciones, que permiten al procesador guardar el contexto actual e invocar una rutina de tratamiento de interrupciones.
- Nivel 5: se introduce la noción de un proceso como un programa en ejecución. Los requisitos fundamentales de los sistemas operativos para dar soporte a múltiples procesos incluyen la habilidad de suspender y continuar los procesos.
- Nivel 6: trata los dispositivos de almacenamiento secundario del computador. Se dan la funciones para posicionar las cabezas de lectura/escritura y la transferencia real de bloques.
- Nivel 7: crea un espacio de direcciones lógicas para los procesos. Organiza el espacio de direcciones virtuales en bloques que pueden moverse entre memoria principal y memoria secundaria.
- Nivel 8: trata con la comunicación de información y mensajes entre los procesos. Una de las herramientas más potentes para este propósito es la tubería o pipe, que es un canal lógico para el flujo de datos entre los procesos.
- Nivel 9: da soporte al almacenamiento a largo plazo en ficheros con nombre. Los datos en el almacenamiento secundario se ven en términos de entidades abstractas y con longitud variable.

- Nivel 10: proporciona acceso a los dispositivos externos utilizando interfaces estándar.
- Nivel 11: responsable para mantener la asociación entre los identificadores externos e internos de los recursos y objetos del sistema. El identificador externo es un nombre que puede utilizar una aplicación o usuario, y el identificador interno es una dirección de otro identificador que puede utilizarse por parte de los niveles inferiores del sistema operativo para localizar y controlar un objeto.
- Nivel 12: proporciona una utilidad completa para dar soporte a los procesos. se da soporte a toda la información necesaria para la gestión ordenada de los procesos.
- Nivel 13: proporciona una interfaz del sistema operativo al usuario. Se denomina shell (caparazón), porque separa al usuario de los detalles de los sistemas operativos y presenta el sistema operativo simplemente como una colección de servicios. El shell acepta mandatos de usuario o sentencias de control de trabajos, los interpreta y crea y controla los procesos que necesita para su ejecución.

Desarrollos que han llevado a los sistemas operativos modernos

Características de los sistemas operativos modernos

Arquitectura micronúcleo o microkernel: asigna sólo unas pocas funciones esenciales al núcleo, incluyendo los espacios de almacenamiento, comunicación entre procesos (IPC), y la planificación básica. Ciertos procesos proporcionan otros servicios del sistema operativo, algunas veces denominados servidores, que ejecutan en modo usuario y son tratados como cualquier otra aplicación por el micronúcleo. Esta técnica desacopla el núcleo y el desarrollo del servidor. Los servidores pueden configurarse para aplicaciones específicas o para determinados requisitos del entorno. La técnica micronúcleo simplifica la implementación, proporciona flexibilidad y se adapta perfectamente a un entorno distribuido. En esencia, un micronúcleo interactúa con procesos locales y remotos del servidor de la misma forma, facilitando la construcción de los sistemas distribuidos.

Multitheading o multihilos: es una técnica en la cual un proceso, ejecutando una aplicación, se divide en una serie de hilos o threads que pueden ejecutar concurrentemente.

Thread o hilo: se trata de una unidad de trabajo. Incluye el contexto del procesador (que contiene el contador del programa y el puntero de pila) y su propia área de datos para una pila (para posibilitar el salto a subrutinas). Un hilo se ejecuta secuencialmente y se puede interrumpir de forma que el procesador pueda dar paso a otro hilo.

Proceso: es una colección de uno o más hilos y sus recursos de sistema asociados (como la memoria, conteniendo tanto código, como datos, ficheros abiertos y dispositivos). Esto corresponde al concepto de programa en ejecución.

Multiprocesamiento simétrico (SMP: Symmetric MultiProcessing): término que se refiere a la arquitectura hardware del computador y también al comportamiento del sistema operativo que explota dicha arquitectura. Se define un multiprocesador simétrico como un sistema de computación aislado con las siguientes características:

- Tiene múltiples procesadores.
- Estos procesadores comparten las mismas utilidades de memoria principal y de E/S, interconectadas por un bus de comunicación u otro esquema de conexión interna.
- Todos los procesadores pueden realizar las mismas funciones (simétrico).

Sistema operativo distribuido: proporciona la ilusión de un solo espacio de memoria principal y un solo espacio de memoria secundario, más otras utilidades de acceso unificadas, como un sistema de ficheros distribuido.

Diseño orientado a objetos: introduce una disciplina al proceso de añadir extensiones modulares a un pequeño núcleo. A nivel del sistema operativo, una estructura basada en objetos permite a los programadores personalizar un sistema operativo sin eliminar la integridad del sistema.

Descripción y control de procesos

Introducción

_ La tarea fundamental de cualquier sistema operativo moderno es la gestión de procesos. El sistema operativo debe reservar recursos para los procesos, permitir a los mismos compartir e intercambiar información, proteger los recursos de cada uno de ellos del resto, y permitir la sincronización entre procesos. Para conseguir alcanzar estos requisitos, el sistema operativo debe mantener una estructura determinada para cada proceso que describa el estado y la propiedad de los recursos y que permite al sistema operativo establecer el control sobre los procesos.

Principales requisitos de los sistemas operativos

_ El diseño de un sistema operativo debe reflejar ciertos requisitos generales:

- El sistema operativo debe intercalar la ejecución de múltiples procesos, para maximizar la utilización del procesador mientras se proporciona un tiempo de respuesta razonable.
- El sistema operativo debe reservar recursos para los procesos conforme a una política específica (por ejemplo, ciertas funciones o aplicaciones son de mayor prioridad) mientras que al mismo tiempo evita interbloqueos1 .
- Un sistema operativo puede requerir dar soporte a la comunicación entre procesos y la creación de procesos, mediante las cuales ayuda a la estructuración de las aplicaciones.

Procesos

Concepto

_ Tenemos diversas definiciones del término proceso, incluyendo:

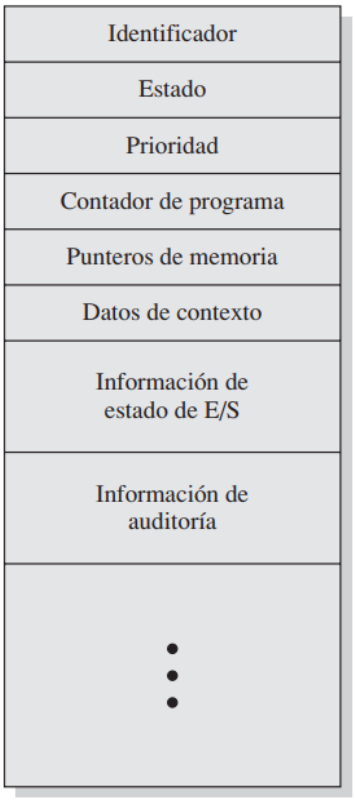
- Un programa en ejecución.
- Una tarea.
- Una instancia de un programa ejecutado en un computador.
- La entidad que se puede asignar y ejecutar en un procesador.
- Una unidad de actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual, y un conjunto de recursos del sistema asociados.

_ También se puede pensar en un proceso como en una entidad que consiste en un número de elementos. Los dos elementos esenciales serían el código de programa (que puede compartirse con otros procesos que estén ejecutando el mismo programa) y un conjunto de datos asociados a dicho código.

_ Supongamos que el procesador comienza a ejecutar este código de programa, y que nos referiremos a esta entidad en ejecución como un proceso. En cualquier instante puntual del tiempo, mientras el proceso está en ejecución, este proceso se puede caracterizar por una serie de elementos, incluyendo los siguientes:

- Identificador: un identificador único asociado a este proceso, para distinguirlo del resto de procesos.
- Estado: si el proceso está actualmente corriendo, está en el estado en ejecución.
- Prioridad: nivel de prioridad relativo al resto de procesos.
- Contador de programa: la dirección de la siguiente instrucción del programa que se ejecutará.
- Punteros a memoria: incluye los punteros al código de programa y los datos asociados a dicho proceso, además de cualquier bloque de memoria compartido con otros procesos.
- Datos de contexto: estos son datos que están presenten en los registros del procesador cuando el proceso está corriendo.
- Información de estado de E/S: incluye las peticiones de E/S pendientes, dispositivos de E/S (por ejemplo, una unidad de cinta) asignados a dicho proceso, una lista de los ficheros en uso por el mismo, etc.
- Información de auditoría: puede incluir la cantidad de tiempo de procesador y de tiempo de reloj utilizados, así como los límites de tiempo, registros contables, etc.

Bloque de control de proceso (Process Control Block - BCP): todos los elementos anteriores se almacenan en esta estructura de datos, que el sistema operativo crea y gestiona. El BCP es la herramienta clave que permite al sistema operativo dar soporte a múltiples procesos y proporcionar multiprogramación.



_ Cuando un proceso se interrumpe, los valores actuales del contador de programa y los registros del procesador (datos de contexto) se guardan en los campos correspondientes del BCP y el estado del proceso se cambia a cualquier otro valor, como bloqueado o listo. El sistema operativo es libre ahora para poner otro proceso en estado de ejecución. El contador de programa y los datos de contexto se recuperan y cargan en los registros del procesador y este proceso comienza a correr. De esta forma, se puede decir que un proceso está compuesto del código de programa y los datos asociados, además del bloque de control de proceso o BCP.

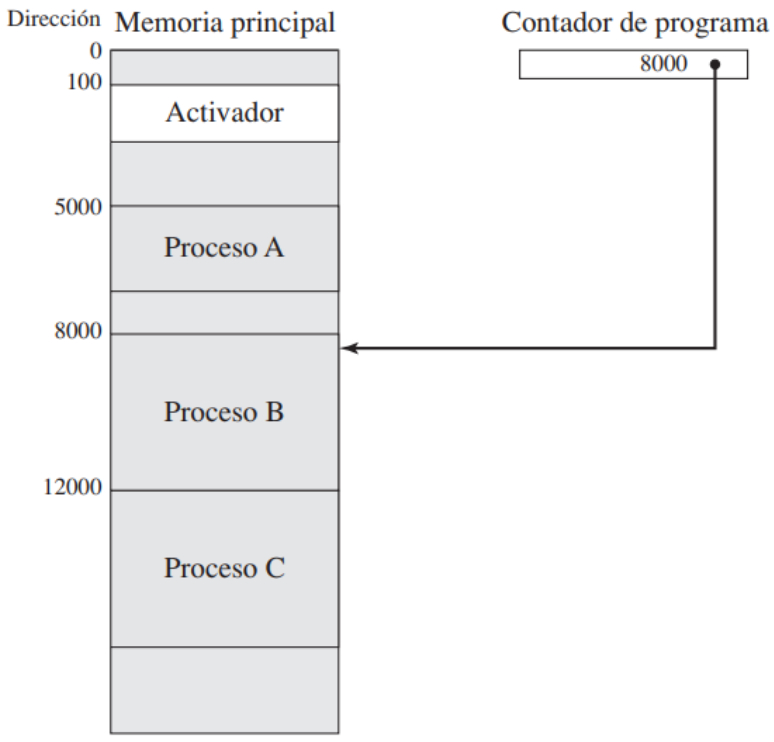
Estados de los procesos

_ Entonces, para que un programa se ejecute, se debe crear un proceso o tarea para dicho programa.

- Desde el punto de vista del procesador, este ejecuta instrucciones de su repertorio de instrucciones en una secuencia dictada por el cambio de los valores del registro contador de programa. A lo largo del tiempo, el contador de programa puede apuntar al código de diferentes programas que son parte de diferentes procesos.
- Desde el punto de vista de un programa individual, su ejecución implica una secuencia de instrucciones dentro del mismo.

Traza del proceso: se puede caracterizar el comportamiento de un determinado proceso, mediante una lista que muestra la secuencia de instrucciones que se ejecutan para dicho proceso o mostrando cómo las trazas de varios procesos se entrelazan.

_ A continuación, vemos un ejemplo en el que se muestra el despliegue en memoria de tres procesos, en donde los tres procesos están representados por programas que residen en memoria principal. De manera adicional, existe un pequeño programa activador (dispatcher) que intercambia el procesador de un proceso a otro.



_ Ahora vemos las trazas de cada uno de los procesos en los primeros instantes de ejecución. Se muestran las 12 primeras instrucciones ejecutadas por los procesos A y C. El proceso B ejecuta 4 instrucciones y se asume que la cuarta instrucción invoca una operación de E/S, a la cual el proceso debe esperar:

5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011
(a) Traza del Proceso A	**(b) Traza del Proceso B**	**(c) Traza del Proceso C**

5000 = Dirección de comienzo del programa del Proceso A.
8000 = Dirección de comienzo del programa del Proceso B.
12000 = Dirección de comienzo del programa del Proceso C.

_ Por último, vemos estas trazas desde el punto de vista del procesador. A continuación, se muestran las trazas entrelazadas resultante de los 52 primeros ciclos de ejecución (por conveniencia los ciclos de instrucciones han sido numerados). En este ejemplo, se asume que el sistema operativo sólo deja que un proceso continúe durante seis ciclos de instrucción, después de los cuales se interrumpe; lo cual previene que un solo proceso monopolice el uso del tiempo del procesador. Vemos que las primeras seis instrucciones del proceso A se ejecutan seguidas de una alarma de temporización (time-out) y de la ejecución de cierto código del activador, que ejecuta seis instrucciones antes de devolver el control al proceso B2 . Después de que se ejecuten cuatro instrucciones, el proceso B solicita una acción de E/S, para la cual debe esperar. Por tanto, el procesador deja de ejecutar el proceso B y pasa a ejecutar el proceso C, por medio del activador. Después de otra alarma de temporización, el procesador vuelve al proceso A. Cuando este proceso llega a su temporización, el proceso B aún estará esperando que se complete su operación de E/S, por lo que el activador pasa de nuevo al proceso C.

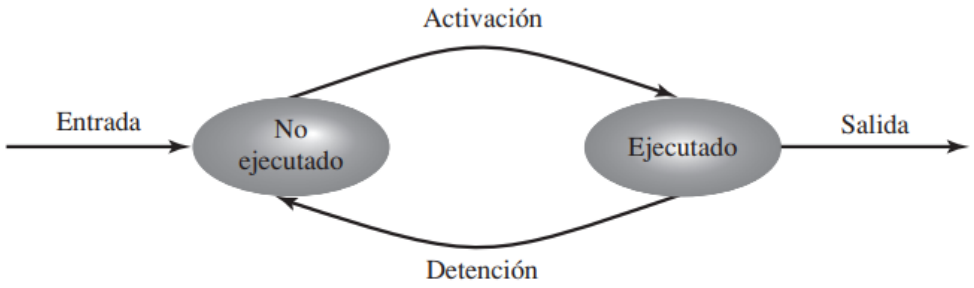
1	5000	
2	5001	
3	5002	
4	5003	
5	5004	
6	5005	
		Temporización
7	100	
8	101	
9	102	
10	103	
11	104	
12	105	
13	8000	
14	8001	
15	8002	
16	8003	
		Petición de E/S
17	100	
18	101	
19	102	
20	103	
21	104	
22	105	
23	12000	
24	12001	
25	12002	
26	12003	
27	12004	
28	12005	
		Temporización
29	100	
30	101	
31	102	
32	103	
33	104	
34	105	
35	5006	
36	5007	
37	5008	
38	5009	
39	5010	
40	5011	
		Temporización
41	100	
42	101	
43	102	
44	103	
45	104	
46	105	
47	12006	
48	12007	
49	12008	
50	12009	
51	12010	
52	12011	
		Temporización

100 = Dirección de comienzo del programa activador.
Las zonas sombreadas indican la ejecución del proceso de activación;
la primera y la tercera columna cuentan ciclos de instrucciones;
la segunda y la cuarta columna las direcciones de las instrucciones que se ejecutan

Modelo de proceso de dos estados

_ La responsabilidad principal del sistema operativo es controlar la ejecución de los procesos; esto incluye determinar el patrón de entrelazado para la ejecución y asignar recursos a los procesos. El primer paso en el diseño de un sistema operativo para el control de procesos es describir el comportamiento que se desea que tengan los procesos. Se puede construir el modelo más simple posible observando que, en un instante dado, un proceso está siendo ejecutando por el procesador o no. En este modelo, un proceso puede estar en dos estados

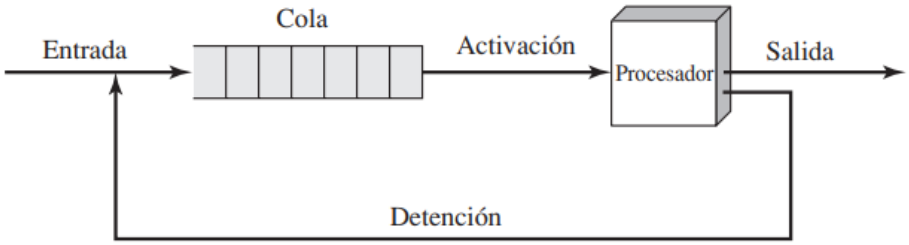
- Ejecutando
- No Ejecutando



(a) Diagrama de transiciones de estados

_ Cuando el sistema operativo crea un nuevo proceso, crea el bloque de control de proceso (BCP) para el nuevo proceso e inserta dicho proceso en el sistema en estado No Ejecutando. El proceso existe, es conocido por el sistema operativo, y está esperando su oportunidad de ejecutar. De cuando en cuando, el proceso actualmente en ejecución se interrumpirá y una parte del sistema operativo, el activador, seleccionará otro proceso a ejecutar. El proceso saliente pasará del estado Ejecutando a No Ejecutando y pasará a Ejecutando un nuevo proceso.

_ Los procesos que no están ejecutando deben estar en una especie de cola, esperando su turno de ejecución. Existe una sola cola cuyas entradas son punteros al BCP de un proceso en particular. Alternativamente, la cola debe consistir en una lista enlazada de bloques de datos, en la cual cada bloque representa un proceso; exploraremos posteriormente esta última implementación. Un proceso que se interrumpe se transfiere a la cola de procesos en espera. Alternativamente, si el proceso ha finalizado o ha sido abortado, se descarta (sale del sistema). En cualquier caso, el activador selecciona un proceso de la cola para ejecutar.



(b) Modelos de colas

Creación y terminación de procesos

Creación de un proceso: cuando se va a añadir un nuevo proceso a aquellos que se están gestionando en un determinado momento, el sistema operativo construye las estructuras de datos que se usan para manejar el proceso y reserva el espacio de direcciones en memoria principal para el proceso. Estas acciones constituyen la creación de un nuevo proceso. Existen cuatro razones o eventos comunes que llevan a la creación de un proceso:

Nuevo proceso de lotes	El sistema operativo dispone de un flujo de control de lotes de trabajos, habitualmente una cinta un disco. Cuando el sistema operativo está listo para procesar un nuevo trabajo, leerá la siguiente secuencia de mandatos de control de trabajos.
Sesión interactiva	Un usuario desde un terminal entra en el sistema.
Creado por el sistema operativo para proporcionar un servicio	El sistema operativo puede crear un proceso para realizar una función en representación de un programa de usuario, sin que el usuario tenga que esperar (por ejemplo, un proceso para controlar la impresión).
Creado por un proceso existente	Por motivos de modularidad o para explotar el paralelismo, un programa de usuario puede ordenar la creación de un número de procesos.

Terminación de procesos: todo sistema debe proporcionar los mecanismos mediante los cuales un proceso indica su finalización, o que ha completado su tarea. Un trabajo por lotes debe incluir una instrucción HALT o una llamada a un servicio de sistema operativo específica para su terminación.

- Instrucción HALT: generará una interrupción para indicar al sistema operativo que dicho proceso ha finalizado.
- Para una aplicación interactiva, las acciones del usuario indicarán cuando el proceso ha terminado. Por ejemplo, en un sistema de tiempo compartido, el proceso de un usuario en particular puede terminar cuando el usuario sale del sistema o apaga su terminal.

_ A continuación, se resumen las razones típicas para la terminación de un proceso.

Finalización normal	El proceso ejecuta una llamada al sistema operativo para indicar que ha completado su ejecución
Límite de tiempo excedido	El proceso ha ejecutado más tiempo del especificado en un límite máximo. Existen varias posibilidades para medir dicho tiempo. Estas incluyen el tiempo total utilizado, el tiempo utilizado únicamente en ejecución, y, en el caso de procesos interactivos, la cantidad de tiempo desde que el usuario realizó la última entrada.
Memoria no disponible	El proceso requiere más memoria de la que el sistema puede proporcionar.
Violaciones de frontera	El proceso trata de acceder a una posición de memoria a la cual no tiene acceso permitido.
Error de protección	El proceso trata de usar un recurso, por ejemplo un fichero, al que no tiene permitido acceder, o trata de utilizarlo de una forma no apropia, por ejemplo, escribiendo en un fichero de sólo lectura.
Error aritmético	El proceso trata de realizar una operación de cálculo no permitida, tal como una división por 0, o trata de almacenar números mayores de los que la representación hardware puede codificar.
Límite de tiempo	El proceso ha esperado más tiempo que el especificado en un valor máximo para que se cumpla un determinado evento.
Fallo de E/S	Se ha producido un error durante una operación de entrada o salida, por ejemplo la imposibilidad de encontrar un fichero, fallo en la lectura o escritura después de un límite máximo de intentos (cuando, por ejemplo, se encuentra un área defectuosa en una cinta), o una operación inválida (la lectura de una impresora en línea).
Instrucción no válida	El proceso intenta ejecutar una instrucción inexistente (habitualmente el resultado de un salto a un área de datos y el intento de ejecutar dichos datos).
Instrucción privilegiada	El proceso intenta utilizar una instrucción reservada al sistema operativo.
Uso inapropiado de datos	Una porción de datos es de tipo erróneo o no se encuentra inicializada.
Intervención del operador por el sistema operativo	Por alguna razón, el operador o el sistema operativo ha finalizado el proceso (por ejemplo, se ha dado una condición de interbloqueo).
Terminación del proceso padre	Cuando un proceso padre termina, el sistema operativo puede automáticamente finalizar todos los procesos hijos descendientes de dicho padre.
Solicitud del proceso padre	Un proceso padre habitualmente tiene autoridad para finalizar sus propios procesos descendientes.

_ Un número de error o una condición de fallo puede llevar a la finalización de un proceso. Por último, en ciertos sistemas operativos, un proceso puede terminarse por parte del proceso que lo creó o cuando dicho proceso padre a su vez ha terminado.

Modelo de proceso de cinco estados

_ Si todos los procesos estuviesen siempre preparados para ejecutar, la gestión de colas sería efectiva. La cola es una lista de tipo FIFO y el procesador opera siguiendo una estrategia cíclica (round-robin o turno rotatorio) sobre todos los procesos disponibles (cada proceso de la cola tiene cierta cantidad de tiempo, por turnos, para ejecutar y regresar de nuevo a la cola, a menos que se bloquee). Sin embargo, esta implementación es inadecuada ya que algunos procesos que están en el estado de No Ejecutando están listos para ejecutar, mientras que otros están bloqueados, esperando a que se complete una operación de E/S. Por lo tanto, utilizando una única cola, el activador no puede seleccionar únicamente los procesos que lleven más tiempo en la cola. En su lugar, debería recorrer la lista buscando los procesos que no estén bloqueados y que lleven en la cola más tiempo.

_ Una forma más natural para manejar esta situación es dividir el estado de No Ejecutando en dos estados, Listo y Bloqueado. Para gestionarlo correctamente, se han añadido dos estados adicionales que resultarán muy útiles. Estos cinco estados en el nuevo diagrama son los siguientes:

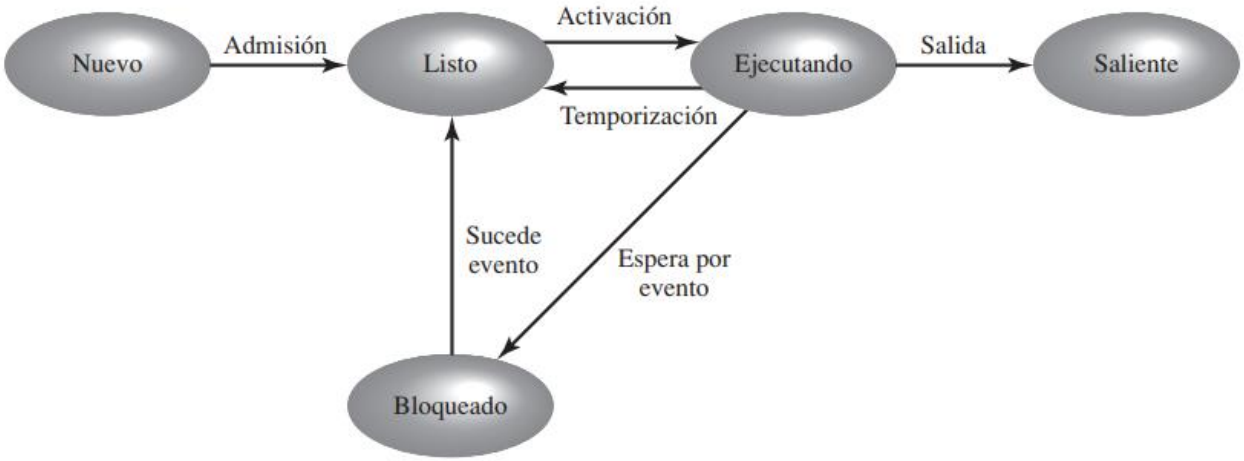
Ejecutando: el proceso está actualmente en ejecución. Para este capítulo asumimos que el computador tiene un único procesador, de forma que sólo un proceso puede estar en este estado en un instante determinado.

Listo: un proceso que se prepara para ejecutar cuando tenga oportunidad.

Bloqueado: un proceso que no puede ejecutar hasta que se cumpla un evento determinado o se complete una operación E/S.

Nuevo: un proceso que se acaba de crear y que aún no ha sido admitido en el grupo de procesos ejecutables por el sistema operativo. Típicamente, se trata de un nuevo proceso que no ha sido cargado en memoria principal, aunque su bloque de control de proceso (BCP) si ha sido creado.

Saliente: un proceso que ha sido liberado del grupo de procesos ejecutables por el sistema operativo, debido a que ha sido detenido o que ha sido abortado por alguna razón.

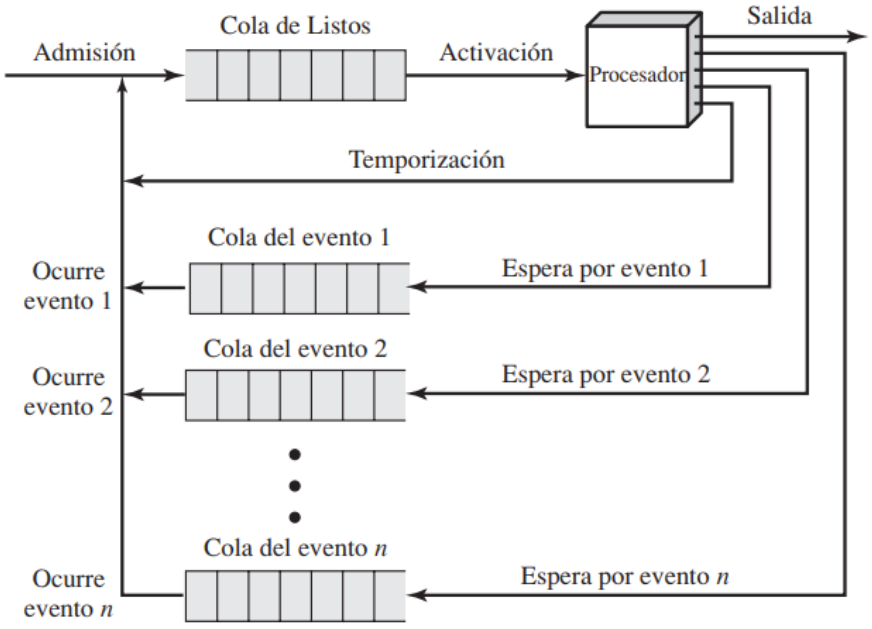


_ El grafico anterior indica que tipos de eventos llevan a cada transición de estado para cada proceso; y las posibles transiciones son las siguientes:

- Null -> Nuevo: se crea un nuevo proceso para ejecutar un programa, en base a cualquier razón de la tabla de creación.
- Nuevo -> Listo: el sistema operativo mueve a un proceso del estado nuevo al estado listo cuando éste se encuentre preparado para ejecutar un nuevo proceso. La mayoría de sistemas fijan un límite basado en el número de procesos existentes o la cantidad de memoria virtual que se podrá utilizar por parte de los procesos existentes. Este límite asegura que no haya demasiados procesos activos y que se degrade el rendimiento sistema.
- Listo -> Ejecutando: cuando llega el momento de seleccionar un nuevo proceso para ejecutar, el sistema operativo selecciona uno de los procesos que se encuentre en el estado Listo. Esta es una tarea la lleva a cabo el planificador.
- Ejecutando -> Saliente: el proceso actual en ejecución se finaliza por parte del sistema operativo tanto si el proceso indica que ha completado su ejecución como si éste se aborta, por cualquiera de las razones de la tabla de terminación.
- Ejecutando -> Listo: la razón más habitual para esta transición es que el proceso en ejecución haya alcanzado el máximo tiempo posible de ejecución de forma ininterrumpida. Existen otras posibles causas alternativas para esta transición, que no están incluidas en todos los sistemas operativos, como los casos en los que el sistema operativo asigna diferentes niveles de prioridad a diferentes procesos. Supóngase, por ejemplo, que el proceso A está ejecutando a un determinado nivel de prioridad, y el proceso B, a un nivel de prioridad mayor, y que se encuentra bloqueado. Si el sistema operativo se da cuenta de que se produce un evento al cual el proceso B está esperando, moverá el proceso B al estado de Listo. Esto puede interrumpir al proceso A y poner en ejecución al proceso B. Decimos, en este caso, que el sistema operativo ha expulsado al proceso A.
- Ejecutando -> Bloqueado: un proceso se pone en el estado Bloqueado si solicita algo por lo cual debe esperar. Una solicitud al sistema operativo se realiza habitualmente por medio de una llamada al sistema; esto es, una llamada del proceso en ejecución a un procedimiento que es parte del código del sistema operativo. Por ejemplo, un proceso ha solicitado un servicio que el sistema operativo no puede realizar en ese momento.
- Bloqueado -> Listo: un proceso en estado Bloqueado se mueve al estado Listo cuando sucede el evento por el cual estaba esperando.
- Listo -> Saliente: esta transición no se muestra en el diagrama de estados. En algunos sistemas, un padre puede terminar la ejecución de un proceso hijo en cualquier momento. También, si el padre termina, todos los procesos hijos asociados con dicho padre pueden finalizarse.
- Bloqueado -> Saliente: se presenta la misma situacion que la transición anterior.

Procesos suspendidos

Necesidad de intercambio o swapping: los tres principales estados descritos (Listo, Ejecutando, Bloqueado) proporcionan una forma sistemática de modelizar el comportamiento de los procesos y diseñar la implementación del sistema operativo. Sin embargo, existe una buena justificación para añadir otros estados al modelo. Para ver este beneficio de nuevos estados, vamos a suponer un sistema que no utiliza memoria virtual. Cada proceso que se ejecuta debe cargarse completamente en memoria principal. Por ejemplo, en el siguiente grafico todos los procesos en todas las colas deben residir en memoria principal:



(b) Múltiples colas de Bloqueados

_ Recordamos que la razón de toda esta compleja maquinaria es que las operaciones de E/S son mucho más lentas que los procesos de cómputo (procesador) y, por tanto, el procesador en un sistema estaría ocioso la mayor parte del tiempo. La diferencia de velocidad entre el procesador y la E/S es tal que sería muy habitual que todos los procesos en memoria se encontrasen a esperas de dichas operaciones.

_ Para solucionar esto tenemos:

- Expandir la memoria principal: para acomodar más procesos, pero hay dos fallos en esta solución. Primero, un coste asociado a la memoria principal, que, desde un coste reducido a nivel de bytes, comienza a incrementarse según nos acercamos a un almacenamiento de gigabytes. Segundo, el apetito de los programas a nivel de memoria ha crecido tan rápido como ha bajado el coste de las memorias. De forma que las grandes memorias actuales han llevado a ejecutar procesos de gran tamaño, no más procesos.

- Swapping (memoria de intercambio): que implica mover parte o todo el proceso de memoria principal al disco. Cuando ninguno de los procesos en memoria principal se encuentra en estado Listo, el sistema operativo intercambia uno de los procesos bloqueados a disco, en la cola de Suspendidos. Esta es una lista de procesos existentes que han sido temporalmente expulsados de la memoria principal, o suspendidos. El sistema operativo trae otro proceso de la cola de Suspendidos o responde a una solicitud de un nuevo proceso. La ejecución continúa con los nuevos procesos que han llegado. El swapping, sin embargo, es una operación de E/S, y por tanto existe el riesgo potencial de hacer que el problema empeore. Cuando todos los procesos en memoria principal se encuentran en estado Bloqueado, el sistema operativo puede suspender un proceso poniéndolo en el estado Suspendido y transfiriéndolo a disco. El espacio que se libera en memoria principal puede usarse para traer a otro proceso.

_ Cuando el sistema operativo ha realizado la operación de swap, tiene dos opciones para seleccionar un nuevo proceso para traerlo a memoria principal: puede admitir un nuevo proceso que se haya creado o puede traer un proceso que anteriormente se encontrase en estado de Suspendido. Pero, esto presenta una dificultad, todos los procesos que fueron suspendidos se encontraban previamente en el estado de Bloqueado en el momento de su suspensión. No sería bueno traer un proceso bloqueado de nuevo a memoria porque podría no encontrarse todavía listo para la ejecución. Todos los procesos en estado Suspendido estaban originalmente en estado Bloqueado, en espera de un evento en particular. Cuando el evento sucede, el proceso no está Bloqueado y está potencialmente disponible para su ejecución.

_ Entonces tenemos dos casos en los que, si un proceso está esperando a un evento (Bloqueado o no) y si un proceso está transferido de memoria a disco (suspendido o no). Para describir estas combinaciones de 2x2 necesitamos cuatro estados:

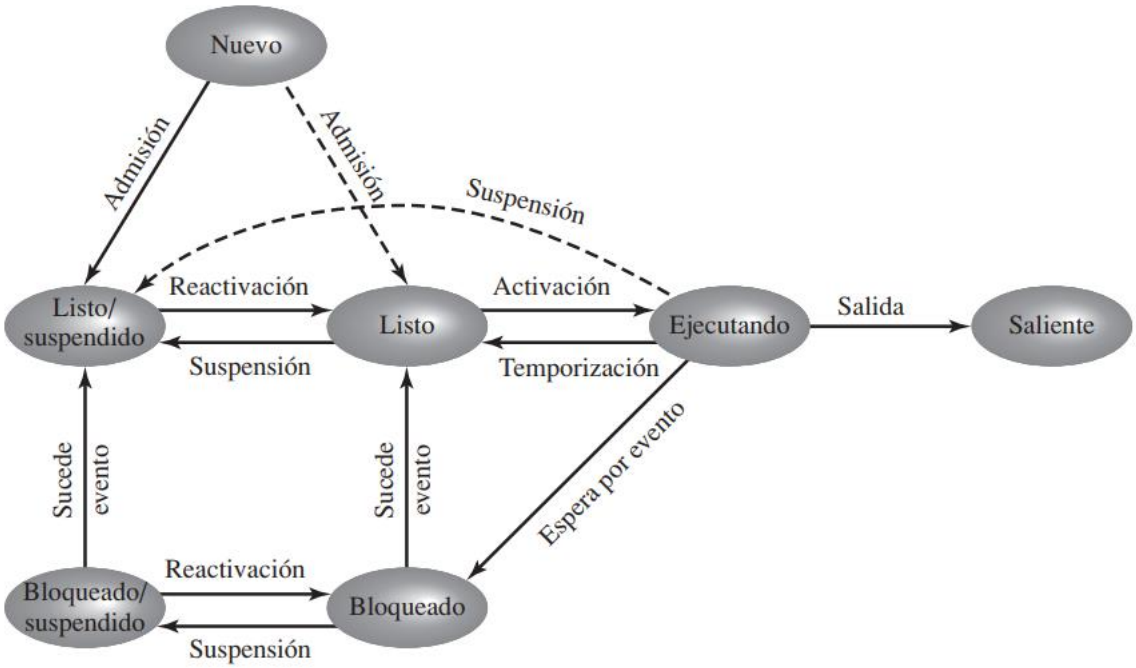
Listo: el proceso está en memoria principal disponible para su ejecución.

Bloqueado: el proceso está en memoria principal y esperando un evento.

Bloqueado/Suspendido: el proceso está en almacenamiento secundario y esperando un evento.

Listo/Suspendido: el proceso está en almacenamiento secundario pero está disponible para su ejecución tan pronto como sea cargado en memoria principal.

_ Ahora vemos el diagrama de transición de estados de procesos con estado suspendidos, en donde las nuevas transiciones más importantes son las siguientes:



(b) Con dos estados Suspendido

- Bloqueado -> Bloqueado/Suspendido: si no hay procesos listos, entonces al menos uno de los procesos bloqueados se transfiere al disco para hacer espacio para otro proceso que no se encuentra bloqueado. Esta transición puede realizarse incluso si hay procesos listos disponibles, si el sistema operativo determina que el proceso actualmente en ejecución o los procesos listos que desea ejecutar requieren más memoria principal para mantener un rendimiento adecuado.
- Bloqueado/Suspendido -> Listo/Suspendido: un proceso en el estado Bloqueado/Suspendido se mueve al estado Listo/Suspendido cuando sucede un evento al que estaba esperando. Esto requiere que la información de estado correspondiente a un proceso suspendido sea accesible para el sistema operativo.
- Listo/Suspendido -> Listo: cuando no hay más procesos listos en memoria principal, el sistema operativo necesitará traer uno para continuar la ejecución. Adicionalmente, puede darse el caso de que un proceso en estado Listo/Suspendido tenga mayor prioridad que cualquiera de los procesos en estado Listo.
- Listo -> Listo/Suspendido: normalmente, el sistema operativo preferirá suspender procesos bloqueados que un proceso Listo, porque un proceso Listo se puede ejecutar en ese momento, mientras que un proceso Bloqueado ocupa espacio de memoria y no se puede ejecutar. Sin embargo, puede ser necesario suspender un proceso Listo si con ello se consigue liberar un bloque suficientemente grande de memoria. También el sistema operativo puede

decidir suspender un proceso Listo de baja prioridad antes que un proceso Bloqueado de alta prioridad, si se cree que el proceso Bloqueado estará pronto listo.

_ Otras transiciones interesantes son las siguientes:

- Nuevo -> Listo/Suspendido y Nuevo a Listo: cuando se crea un proceso nuevo, puede añadirse a la cola de Listos o a la cola de Listos/Suspendidos. En cualquier caso, el sistema operativo puede crear un bloque de control de proceso (BCP) y reservar el espacio de direcciones del proceso. Puede ser preferible que el sistema operativo realice estas tareas internas cuanto antes, de forma que pueda disponer de suficiente cantidad de procesos no bloqueados. Sin embargo, con esta estrategia, puede ocurrir que no haya espacio suficiente en memoria principal para el nuevo proceso; de ahí el uso de la transición Nuevo -> Listo/Suspendido.
- Bloqueado/Suspendido -> Bloqueado: consideramos el escenario en el que un proceso termina, liberando alguna memoria principal. Hay un proceso en la cola de Bloqueados/Suspendidos con mayor prioridad que todos los procesos en la cola de Listos/Suspendidos y el sistema operativo tiene motivos para creer que el evento que lo bloquea va a ocurrir en breve. Bajo estas circunstancias, sería razonable traer el proceso Bloqueado a memoria por delante de los procesos Listos.
- Ejecutando -> Listo/Suspendido: normalmente, un proceso en ejecución se mueve a la cola de Listos cuando su tiempo de uso del procesador finaliza. Sin embargo, si el sistema operativo expulsa al proceso en ejecución porque un proceso de mayor prioridad en la cola de Bloqueado/Suspendido acaba de desbloquearse, el sistema operativo puede mover al proceso en ejecución directamente a la cola de Listos/Suspendidos y liberar parte de la memoria principal.
- De cualquier estado -> Saliente: habitualmente, un proceso termina cuando se está ejecutando, bien porque ha completado su ejecución o debido a una condición de fallo. Sin embargo, en algunos sistemas operativos un proceso puede terminarse por el proceso que lo creó o cuando el proceso padre a su vez ha terminado. Si se permite esto, un proceso en cualquier estado puede moverse al estado Saliente.

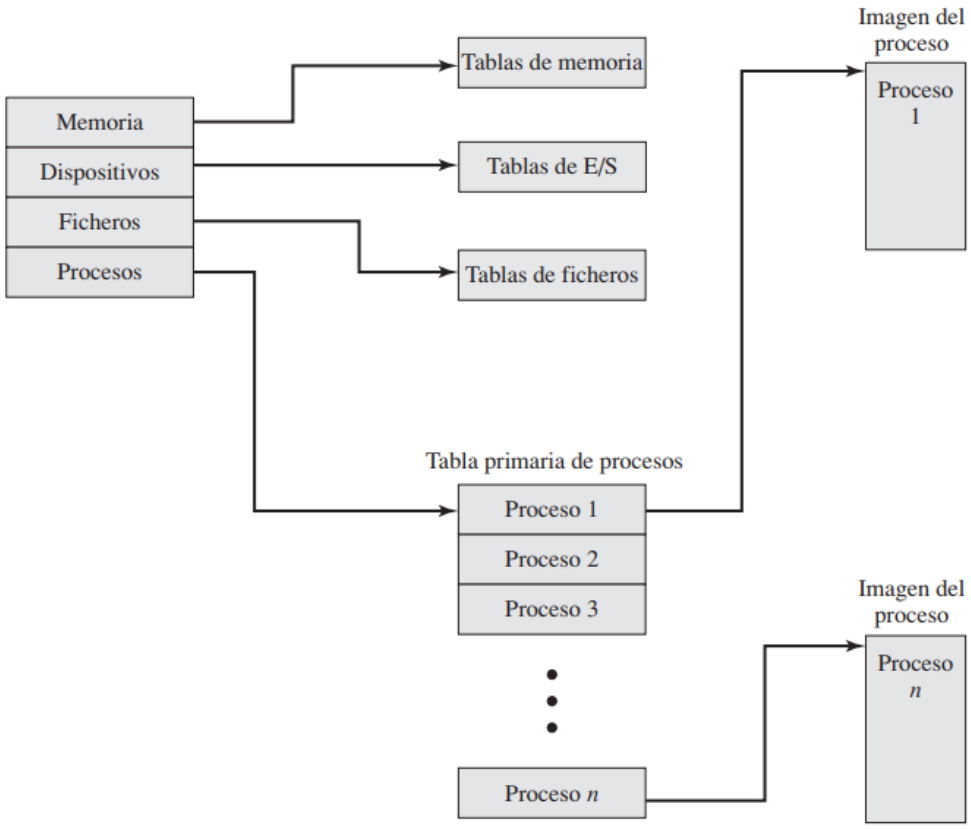
_ Las razones para la suspensión de un proceso son:

Swapping	El sistema operativo necesita liberar suficiente memoria principal para traer un proceso en estado Listo de ejecución.
Otras razones del sistema operativo	El sistema operativo puede suspender un proceso en segundo plano o de utilidad o un proceso que se sospecha puede causar algún problema.
Solicitud interactiva del usuario	Un usuario puede desear suspender la ejecución de un programa con motivo de su depuración o porque está utilizando un recurso.
Temporización	Un proceso puede ejecutarse periódicamente (por ejemplo, un proceso monitor de estadísticas sobre el sistema) y puede suspenderse mientras espera el siguiente intervalo de ejecución.
Solicitud del proceso padre	Un proceso padre puede querer suspender la ejecución de un descendiente para examinar o modificar dicho proceso suspendido, o para coordinar la actividad de varios procesos descendientes.

Descripción de procesos

Estructuras de control del sistema operativo:

_ Si el sistema operativo se encarga de la gestión de procesos y recursos, debe disponer de información sobre el estado actual de cada proceso y cada recurso. El mecanismo universal para proporcionar esta información es la de que el sistema operativo construye y mantiene tablas de información sobre cada entidad que gestiona. El siguiente gráfico muestra cuatro diferentes tipos de tablas mantenidas por el sistema operativo (memoria, E/S, ficheros y procesos):



Tablas de memoria: se usan para mantener un registro tanto de la memoria principal (real) como de la secundaria (virtual). Parte de la memoria principal está reservada para el uso del sistema operativo; el resto está disponible para el uso de los procesos. Los procesos se mantienen en memoria secundaria utilizando algún tipo de memoria virtual o técnicas de swapping. Las tablas de memoria deben incluir la siguiente información:

- Las reservas de memoria principal por parte de los procesos.
- Las reservas de memoria secundaria por parte de los procesos.
- Todos los atributos de protección que restringe el uso de la memoria principal y virtual, de forma que los procesos puedan acceder a ciertas áreas de memoria compartida.
- La información necesaria para manejar la memoria virtual.

Tablas de E/S: para gestionar los dispositivos de E/S y los canales del computador. Pero, en un instante determinado, un dispositivo E/S puede estar disponible o asignado a un proceso en particular. Si la operación de E/S se está realizando, el sistema operativo necesita conocer el estado de la operación y la dirección de memoria principal del área usada como fuente o destino de la transferencia de E/S.

Tablas de ficheros o archivos: estas proporcionan información sobre la existencia de ficheros, su posición en almacenamiento secundario, su estado actual, y otros atributos. La mayoría de, o prácticamente toda, esta información se puede gestionar por el sistema de ficheros, en cuyo caso el sistema operativo tiene poco o ningún conocimiento sobre los ficheros. En otros sistemas operativos, la gran mayoría del detalle de la gestión de ficheros sí que gestiona en el propio sistema operativo.

Tablas de procesos: se utilizan para gestionar los procesos. La parte restante de esta sección se encuentra dedicada a examinar los requisitos que tienen las tablas de procesos. El sistema operativo debe tener alguna información sobre el entorno básico, tal como cuánta memoria física existe, cuáles son los dispositivos de E/S y cuáles son sus identificadores, etc.

Estructuras de control de proceso

Localización de los procesos: un proceso debe incluir un programa o un conjunto de programas a ejecutar. Asociados con estos programas existen unas posiciones de memoria para los datos de variables locales y globales y de cualquier constante definida. Así, un proceso debe consistir en una cantidad suficiente de memoria para almacenar el programa y datos del mismo. Adicionalmente, la ejecución típica de un programa incluye una pila que se utiliza para registrar las llamadas a procedimientos y los parámetros pasados entre dichos procedimientos. Por último, cada proceso está asociado a un número de atributos que son utilizados por el sistema operativo para controlar el proceso. Normalmente, el conjunto de estos atributos se denomina bloque de control del proceso (BCP). Nos podemos referir al conjunto de programa, datos, pila, y atributos, como la imagen del proceso. La posición de la imagen del proceso dependerá del esquema de gestión de memoria que se utilice.

Atributos de proceso: cualquier sistema operativo multiprogramado de la actualidad requiere una gran cantidad de información para manejar cada proceso. Como quedó explicado, esta información se puede considerar que reside en el bloque de control del proceso (BCP). Los diferentes sistemas organizarán esta información de formas diferentes. La siguiente tabla muestra la lista de los tipos de categorías de información que el sistema operativo requiere para cada proceso:

Identificación del proceso

Identificadores

Identificadores numéricos que se pueden guardar dentro del bloque de control del proceso:

- identificadores del proceso
- identificador del proceso que creó a este proceso (proceso padre)
- identificador del usuario

Información de estado del procesador

Registros visibles por el usuario

Un registro visible por el usuario es aquel al que se puede hacer referencia por medio del lenguaje máquina que ejecuta el procesador cuando está en modo usuario. Normalmente, existen de 8 a 32 de estos registros, aunque determinadas implementaciones RISC tienen más de 100.

Registros de estado y control

Hay una gran variedad de registros del procesador que se utilizan para el control de operaciones. Estos incluyen:

- *Contador de programa*: contiene la dirección de la siguiente instrucción a ejecutar.
- *Códigos de condición*: resultan de la operación lógica o aritmética más reciente (por ejemplo, signo, cero, acarreo, igual, desbordamiento).
- *Información de estado*: incluyen los *flags* de interrupciones habilitadas/deshabilitadas, modo ejecución.

Puntero de pila

Cada proceso tiene una o más pilas de sistema (LIFO) asociadas a él. Una pila se utiliza para almacenar los parámetros y las direcciones de retorno de los procedimientos y llamadas a sistema. Un puntero de pila apunta a la parte más alta de la pila.

Información de control de proceso

Información de estado y de planificación

Esta es la información que el sistema operativo necesita para analizar las funciones de planificación. Elementos típicos de esta información son:

- *Estado del proceso*: indica si está listo o no el proceso para ser planificado para su ejecución (por ejemplo, Ejecutando, Listo, Esperando, Detenido).
- *Prioridad*: uno o más campos que se pueden utilizar para escribir la prioridad de planificación del proceso. En algunos sistemas, se necesitan múltiples valores (por ejemplo, por-defecto, actual, mayor-disponible).
- *Información relativa a planificación*: esto dependerá del algoritmo de planificación utilizado. Por ejemplo, la cantidad de tiempo que el proceso estaba esperando y la cantidad de tiempo que el proceso ha ejecutado la última vez que estuvo corriendo.
- *Evento*: identificar el evento al cual el proceso está esperando para continuar su ejecución.

Estructuración de datos

Un proceso puede estar enlazado con otro proceso en una cola, anillo o con cualquier otra estructura. Por ejemplo, todos los procesos en estado de espera por un nivel de prioridad en particular pueden estar enlazados en una cola. Un proceso puede mostrar una relación padre-hijo (creador-creado) con otro proceso. El bloque de control del proceso puede contener punteros a otros procesos para dar soporte a estas estructuras.

Comunicación entre procesos

Se pueden asociar diferentes *flags*, señales, y mensajes relativos a la comunicación entre dos procesos independientes. Alguna o toda esta información se puede mantener en el bloque de control de proceso (BCP).

Privilegios de proceso

Los procesos adquieren privilegios de acuerdo con la memoria a la que van a acceder y los tipos de instrucciones que se pueden ejecutar. Adicionalmente, los privilegios se pueden utilizar para usar utilidades de sistema o servicios.

Gestión de memoria

Esta sección puede incluir punteros a tablas de segmentos y/o páginas que describen la memoria virtual asignada a este proceso.

Propia de recursos y utilización

Se deben indicar los recursos controlados por el proceso, por ejemplo, ficheros abiertos. También se puede incluir un histórico de utilización del procesador o de otros recursos; esta información puede ser necesaria para el planificador.

Control de procesos

Modos de ejecución: muchos procesadores proporcionan al menos dos modos de ejecución. Ciertas instrucciones se pueden ejecutar en modos privilegiados únicamente. Éstas incluirían lectura y modificación de los registros de control, por ejemplo la palabra de estado de programa; instrucciones de E/S primitivas; e instrucciones relacionadas con la gestión de memoria. Adicionalmente, ciertas regiones de memoria sólo se pueden acceder en los modos más privilegiados.

- Modo usuario: es el modo menos privilegiado, porque los programas de usuario típicamente se ejecutan en este modo.
- Modo sistema, modo control o modo núcleo: son distintas formas de llamar al modo más privilegiado, y se refiere al núcleo del sistema operativo, que es la parte del sistema operativo que engloba las funciones más importantes del sistema

_ A continuación, vemos las funciones típicas de un núcleo de sistema operativo:

Gestión de procesos

- Creación y terminación de procesos
- Planificación y activación de procesos
- Intercambio de procesos
- Sincronización de procesos y soporte para comunicación entre procesos
- Gestión de los bloques de control de proceso

Gestión memoria

- Reserva de espacio direcciones para los procesos
- *Swapping*
- Gestión de páginas y segmentos

Gestión E/S

- Gestión de *buffers*
- Reserva de canales de E/S y de dispositivos para los procesos

Funciones de soporte

- Gestión de interrupciones
- Auditoría
- Monitorización

_ El motivo por el cual se usan los otros modos es claro. Se necesita proteger al sistema operativo y a las tablas clave del sistema, por ejemplo los bloques de control de proceso, de la interferencia con programas de usuario. En modo núcleo, el software tiene control completo del procesador y de sus instrucciones, registros, y memoria. Este nivel de control no es necesario y por seguridad no es recomendable para los programas de usuario.

Creación de procesos

_ Una vez que el sistema operativo decide, por cualquier motivo, crear un proceso procederá de la siguiente manera:

1. Asignar un identificador de proceso único al proceso.
2. Reservar espacio para proceso: esto incluye todos los elementos de la imagen del proceso. Para ello, el sistema operativo debe conocer cuánta memoria se requiere para el espacio de direcciones privado (programas y datos) y para la pila de usuario.
3. Inicialización del bloque de control de proceso: la parte de identificación de proceso del BCP contiene el identificador del proceso así como otros posibles identificadores, tal como el indicador del proceso padre.

4. Establecer los enlaces apropiados: por ejemplo, añadir un proceso nuevo a una lista enlazada que se utiliza como cola de planificación.
5. Creación o expansión de otras estructuras de datos: por ejemplo, mantener un archivo de contabilidad.

Cambio de proceso

_ Un cambio de proceso puede ocurrir en cualquier instante en el que el sistema operativo obtiene el control sobre el proceso actualmente en ejecución. La siguiente tabla indica los posibles momentos en los que el sistema operativo puede tomar dicho control:

Mecanismo	Causa	Uso
Interrupción	Externa a la ejecución del proceso actualmente en ejecución.	Reacción ante un evento externo asíncrono.
Trap	Asociada a la ejecución de la instrucción actual.	Manejo de una condición de error o de excepción.
Llamada al sistema	Solicitud explícita.	Llamada a una función del sistema operativo.

_ Algunos ejemplos de interrupciones del sistema son:

Interrupción de reloj: el sistema operativo determinar si el proceso en ejecución ha excedido o no la unidad máxima de tiempo de ejecución, denominada rodaja de tiempo (time slice). Una rodaja de tiempo es la máxima cantidad de tiempo que un proceso puede ejecutar antes de ser interrumpido. En dicho caso, este proceso se puede pasar al estado de Listo y se puede activar otro proceso.

Interrupción de E/S: el sistema operativo determina qué acción de E/S ha ocurrido. Si la acción de E/S constituye un evento por el cual están esperando uno o más procesos, el sistema operativo mueve todos los procesos correspondientes al estado de Listos (y los procesos en estado Bloqueado/Suspendido al estado Listo/Suspendido). El sistema operativo puede decidir si reanuda la ejecución del proceso actualmente en estado Ejecutando o si lo expulsa para proceder con la ejecución de un proceso Listo de mayor prioridad.

Fallo de memoria: el procesador se encuentra con una referencia a una dirección de memoria virtual, a una palabra que no se encuentra en memoria principal. El sistema operativo debe traer el bloque (página o segmento) que contiene la referencia desde memoria secundaria a memoria principal. Después de que se solicita la operación de E/S para traer el bloque a memoria, el proceso que causó el fallo de memoria se pasa al estado de Bloqueado; el sistema operativo realiza un cambio de proceso y pone a ejecutar a otro proceso. Después de que el bloque de memoria solicitado se haya traído, el proceso pasará al estado Listo.

Trap o Cepo: con esto, el sistema operativo conoce si una condición de error o de excepción es irreversible. Si es así, el proceso en ejecución se pone en el estado Saliente y se hace un cambio de proceso. De no ser así, el sistema operativo actuará dependiendo de la naturaleza del error y su propio diseño.

Llamada al sistema o del supervisor: el sistema operativo se puede activar por medio de una llamada al sistema procedente del programa en ejecución. Por ejemplo, si se está ejecutando un proceso y se ejecuta una operación que implica una llamada de E/S, como abrir un archivo.

Cambio de modo

_ En la fase de interrupción, el procesador comprueba que no exista ninguna interrupción pendiente, indicada por la presencia de una señal de interrupción. Si no hay interrupciones pendientes, el procesador pasa a la fase de búsqueda de instrucción, siguiendo con el programa del proceso actual. Si hay una interrupción pendiente, el proceso actúa de la siguiente manera:

1. Coloca el contador de programa en la dirección de comienzo de la rutina del programa manejador de la interrupción.
2. Cambia de modo usuario a modo núcleo de forma que el código de tratamiento de la interrupción pueda incluir instrucciones privilegiadas.

Cambio del estado del proceso: es un concepto diferente del cambio de proceso. Un cambio de modo puede ocurrir sin que se cambie el estado del proceso actualmente en estado Ejecutando. En dicho caso, la salvaguarda del estado y su posterior restauración comportan sólo una ligera sobrecarga. Sin embargo, si el proceso actualmente en estado Ejecutando, se va a mover a cualquier otro estado (Listo, Bloqueado, etc.), entonces el sistema operativo debe realizar cambios sustanciales en su entorno. Los pasos que se realizan para un cambio de proceso completo son:

1. Actualizar el bloque de control del proceso seleccionado.
2. Actualizar las estructuras de datos de la gestión de memoria.
3. Restaurar el contexto del proceso seleccionado.

Ejecución del sistema operativo

Núcleo sin procesos: una visión tradicional, que es común a muchos sistemas operativos antiguos, es la ejecución del sistema operativo fuera de todo proceso. Con esta visión, cuando el proceso en ejecución se interrumpe o invoca una llamada al sistema, el contexto se guarda y el control pasa al núcleo.

- El sistema operativo tiene su propia región de memoria y su propia pila de sistema para controlar la llamada a procedimientos y sus retornos. El sistema operativo puede realizar todas las funciones que necesite y restaurar el contexto del proceso interrumpido, que hace que se retome la ejecución del proceso de usuario afectado. Ejecuta el núcleo del sistema operativo fuera de cualquier proceso.

- El código del sistema operativo se ejecuta como una entidad independiente que requiere un modo privilegiado de ejecución.

Ejecución dentro de los procesos de usuario: una alternativa que es común en los sistemas operativos de máquinas pequeñas (PC, estaciones de trabajo) es ejecutar virtualmente todo el software de sistema operativo en el contexto de un proceso de usuario. Esta visión es tal que el sistema operativo se percibe como un conjunto de rutinas que el usuario invoca para realizar diferentes funciones, ejecutadas dentro del entorno del proceso de usuario.

- Un proceso se ejecuta en modo privilegiado cuando se ejecuta el código del sistema operativo.

Sistemas operativos basados en procesos: en este caso, las principales funciones del núcleo se organizan como procesos independientes. Útil en un entorno de multiprocesador o de varios computadores.

Hilos, SMP y micronúcleos

Introducción

_ Se ha presentado el concepto de un proceso como poseedor de dos características:

Unidad de propiedad de recursos: un proceso incluye un espacio de direcciones virtuales para el manejo de la imagen del proceso.

Unidad de expedición (Planificación/ejecución): la ejecución de un proceso sigue una ruta de ejecución (traza) a través de uno o más programas. Esta ejecución puede estar intercalada con ese u otros procesos. De esta manera, un proceso tiene un estado de ejecución (Ejecutando, Listo, etc.) y una prioridad de activación y ésta es la entidad que se planifica y activa por el sistema operativo.

_ En la mayor parte de los sistemas operativos tradicionales, estas dos características son, realmente, la esencia de un proceso. Además, son dos características independientes y podrían ser tratadas como tales por el sistema operativo. Para distinguir estas dos características:

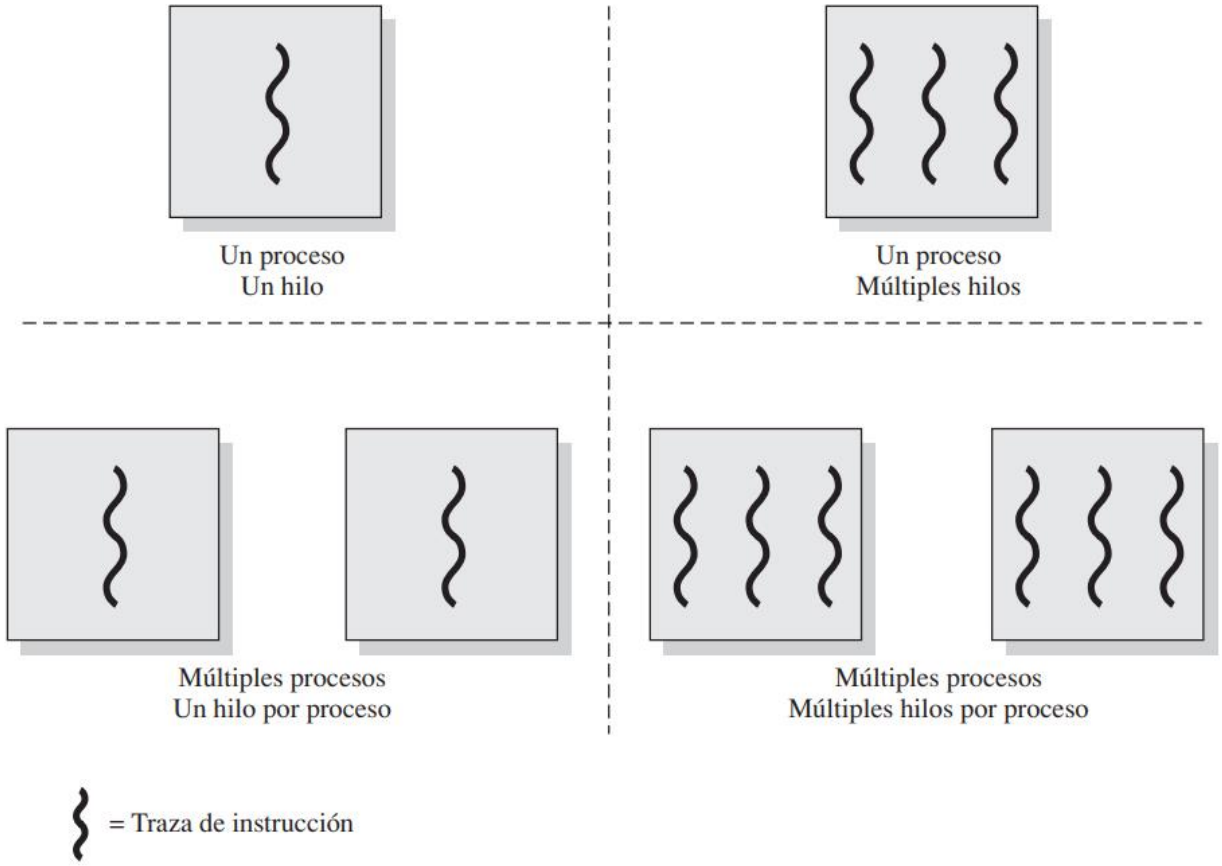
- La unidad de expedición que se activa se suele denominar hilo (thread), o proceso ligero.
- La unidad de propiedad de recursos se suele denominar proceso o tarea.

Procesos e hilos

MULTIHILO

Multihilo: se refiere a la capacidad de un sistema operativo de dar soporte a múltiples hilos de ejecución en un solo proceso. El enfoque tradicional de un solo hilo de ejecución por proceso, en el que no se identifica con el concepto de hilo, se conoce como estrategia monohilo.

- MS-DOS soporta un solo hilo.
- UNIX soporta múltiples procesos de usuarios, pero sólo un hilo por proceso.
- Windows 2000, Solaris, Linux, Mach, y OS/2 soportan múltiples hilos.



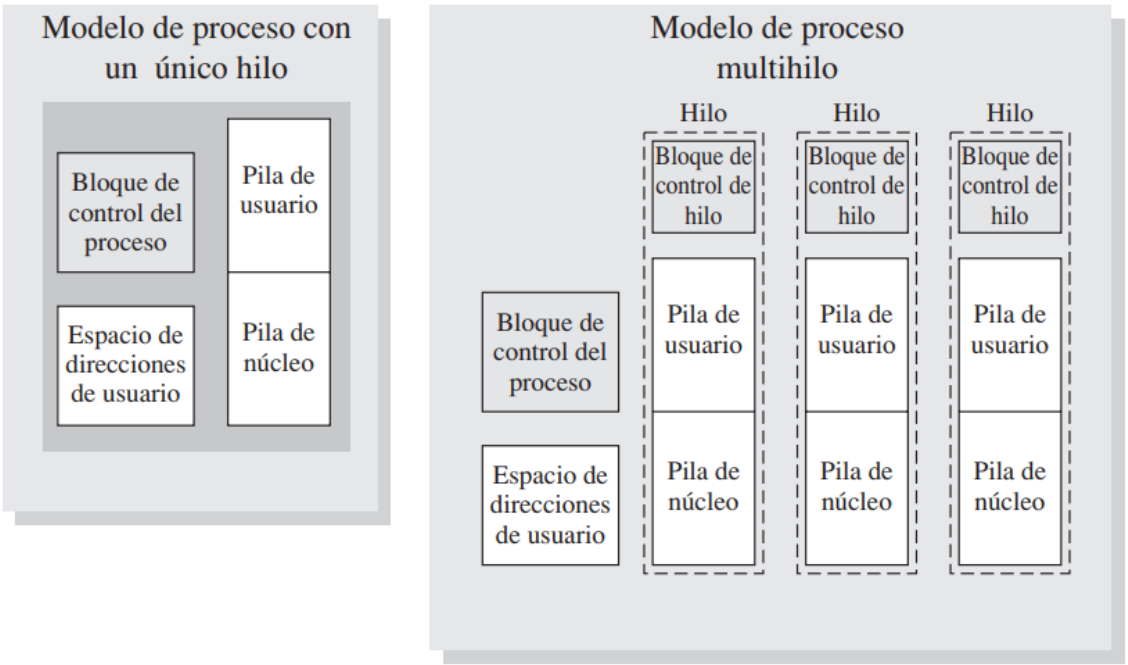
_ En un entorno multihilo, un proceso se define como la unidad de asignación de recursos y una unidad de protección. Se asocian con procesos los siguientes:

- Tienen un espacio de direcciones virtuales que soporta la imagen del proceso.
- Acceso protegido a procesadores, otros procesos (para comunicación entre procesos), archivos y recursos de E/S (dispositivos y canales).

_ Dentro de un proceso puede haber uno o más hilos, cada uno con:

- Un estado de ejecución por hilo (Ejecutando, Listo, etc.).
- Un contexto de hilo que se almacena cuando no está en ejecución; una forma de ver a un hilo es como un contador de programa independiente dentro de un proceso.
- Una pila de ejecución.
- Por cada hilo, espacio de almacenamiento para variables locales.
- Acceso a la memoria y recursos de su proceso, compartido con todos los hilos de su mismo proceso.

_ El siguiente gráfico muestra la diferencia entre hilos y procesos desde el punto de vista de gestión de procesos. En un modelo de proceso monohilo (es decir, no existe el concepto de hilo), la representación de un proceso incluye su bloque de control de proceso y el espacio de direcciones de usuario, además de las pilas de usuario y núcleo para gestionar el comportamiento de las llamadas/retornos en la ejecución de los procesos. Mientras el proceso está ejecutando, los registros del procesador se controlan por ese proceso y, cuando el proceso no se está ejecutando, se almacena el contenido de estos registros. En un entorno multihilo, sigue habiendo un único bloque de control del proceso y un espacio de direcciones de usuario asociado al proceso, pero ahora hay varias pilas separadas para cada hilo, así como un bloque de control para cada hilo que contiene los valores de los registros, la prioridad, y otra información relativa al estado del hilo.



Beneficios de los hilos: los mayores beneficios de los hilos provienen de las consecuencias del rendimiento:

1. Lleva mucho menos tiempo crear un nuevo hilo en un proceso existente que crear un proceso totalmente nuevo.
2. Lleva menos tiempo finalizar un hilo que un proceso.
3. Lleva menos tiempo cambiar entre dos hilos dentro del mismo proceso.
4. Puesto que los hilos de un mismo proceso comparten memoria y archivos, pueden comunicarse entre sí sin invocar al núcleo.

_ De esta forma, si se desea implementar una aplicación o función como un conjunto de unidades de ejecución relacionadas, es mucho más eficiente hacerlo con un conjunto de hilos que con un conjunto de procesos independientes. Un ejemplo de una aplicación que podría hacer uso de hilos es un servidor de archivos. Cada vez que llega una nueva petición de archivo, el programa de gestión de archivos puede ejecutar un nuevo hilo. Ya que un servidor manejará muchas peticiones, se crearán y finalizarán muchos hilos en un corto periodo de tiempo. Si el servidor ejecuta en una máquina multiprocesador, pueden estar ejecutando simultáneamente múltiples hilos del mismo proceso en diferentes procesadores.

Usos de los hilos en un sistema monousuario y multiproceso:

- Trabajo en primer plano y en segundo plano: por ejemplo, en un programa de hoja de cálculo, un hilo podría mostrar menús y leer la entrada de usuario, mientras otro hilo ejecuta los mandatos de usuario y actualiza la hoja de cálculo. Esta forma de trabajo a menudo incrementa la velocidad que se percibe de la aplicación, permitiendo al programa solicitar el siguiente mandato antes de que el mandato anterior esté completado.
- Procesamiento asíncrono: los elementos asíncronos de un programa se pueden implementar como hilos.
- Velocidad de ejecución: un proceso multihilo puede computar una serie de datos mientras que lee los siguientes de un dispositivo. En un sistema multiprocesador pueden estar ejecutando simultáneamente múltiples hilos de un mismo proceso.
- Estructura modular de programas: los programas que realizan diversas tareas o que tienen varias fuentes y destinos de entrada y salida, se pueden diseñar e implementar más fácilmente usando hilos.

_ Existen diversas acciones que afectan a todos los hilos de un proceso y que el sistema operativo debe gestionar a nivel de proceso. La suspensión de un proceso implica la la suspensión de todos los hilos de un proceso, puesto que todos comparten el mismo espacio de direcciones. De forma similar, la terminación de un proceso supone terminar con todos los hilos dentro de dicho proceso.

Funcionalidades de los hilos

_ Los hilos, al igual que los procesos, tienen estados de ejecución y se pueden sincronizar entre ellos. A continuación se analizan estos dos aspectos de las funcionalidades de los hilos:

Estados de los hilos: al igual que con los procesos, los principales estados de los hilos son Ejecutando, Listo y Bloqueado. Generalmente, no tiene sentido aplicar estados de suspensión a un hilo, ya que dichos estados son conceptos de nivel de proceso. Hay cuatro operaciones básicas relacionadas con los hilos que están asociadas con un cambio de estado del mismo:

- Creación: cuando se crea un nuevo proceso, también se crea un hilo de dicho proceso. Posteriormente, un hilo del proceso puede crear otro hilo dentro del mismo proceso, proporcionando un puntero a las instrucciones y los argumentos para el nuevo hilo.
- Bloqueo: cuando un hilo necesita esperar por un evento se bloquea, almacenando los registros de usuario, contador de programa y punteros de pila.
- Desbloqueo: cuando sucede el evento por el que el hilo está bloqueado, el hilo se pasa a la cola de Listos.
- Finalización: cuando se completa un hilo, se liberan su registro de contexto y pilas.

Sincronización de hilos: todos los hilos de un proceso comparten el mismo espacio de direcciones y otros recursos, como por ejemplo, los archivos abiertos. Cualquier alteración de un recurso por cualquiera de los hilos, afecta al entorno del resto de los hilos del mismo proceso. Por tanto, es necesario sincronizar las actividades de los hilos para que no interfieran entre ellos o corrompan estructuras de datos.

Hilos de nivel de usuario y de nivel de núcleo

Hilos de nivel de usuario: en un entorno ULT (User-Level Threads)puro, la aplicación gestiona todo el trabajo de los hilos y el núcleo no es consciente de la existencia de los mismos.

Hilos a nivel de núcleo: en un entorno KLT (Kernel-Level Threads) puro, el núcleo gestiona todo el trabajo de gestión de hilos. Windows, Linux y OS/2 son ejemplos de este enfoque. El núcleo mantiene la información de contexto del proceso y de los hilos. La planificación se realiza en función de los hilos. Son también conocidos en la como hilos Kernel-Supported Threads o Lightweight Processes.

Enfoques combinados: algunos sistemas operativos proporcionan utilidades combinadas ULT/KLT. Solaris es el principal ejemplo de esto. En un sistema combinado, la creación de hilos se realiza por completo en el espacio de usuario, como la mayor parte de la planificación y sincronización de hilos dentro de una aplicación. El

programador debe ajustar el número de KLT para una máquina y aplicación en particular para lograr los mejores resultados posibles.

Relación entre hilos y procesos

Hilos:Procesos	Descripción	Sistemas de Ejemplo
1:1	Cada hilo de ejecución es un único proceso con su propio espacio de direcciones y recursos.	Implementaciones UNIX tradicionales.
M:1	Un proceso define un espacio de direcciones y pertenencia dinámica de recursos. Se pueden crear y ejecutar múltiples hilos en ese proceso.	Windows NT, Solaris, Linux, OS/2, OS/390, MACH.
1:M	Un hilo puede migrar de un entorno de proceso a otro. Esto permite a los hilos moverse fácilmente entre distintos sistemas.	Ra (Clouds), Emerald.
M:N	Combina atributos de M:1 y casos 1:M.	TRIX.

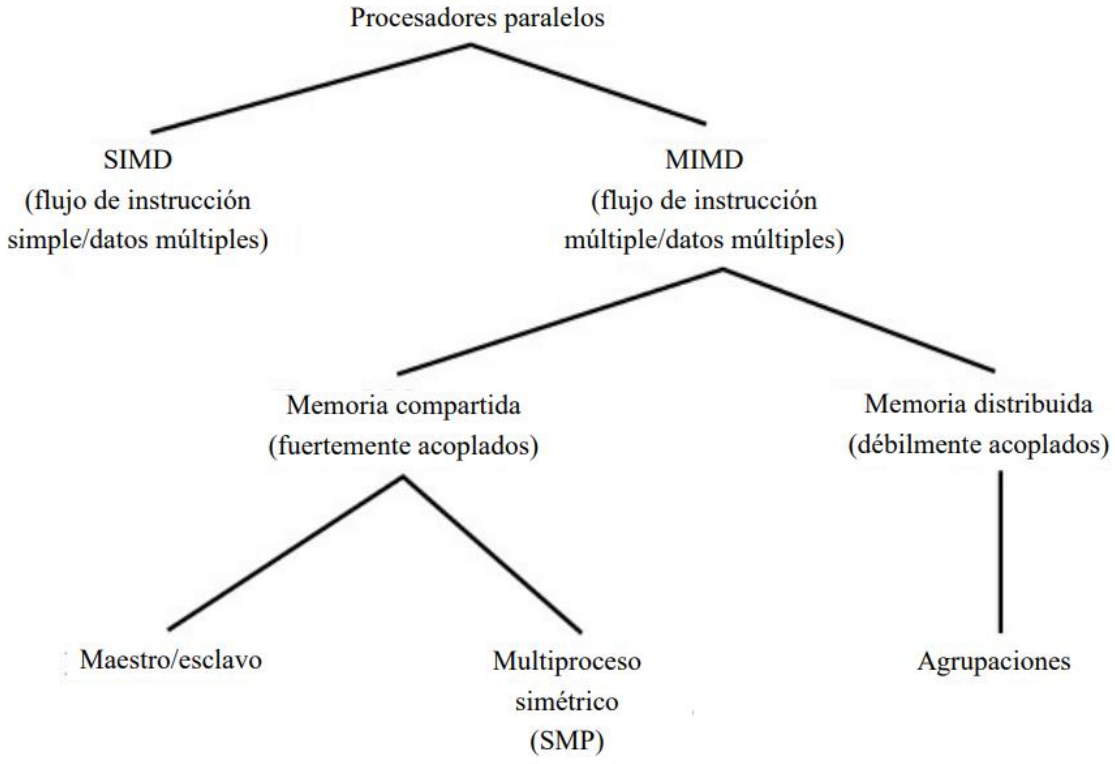
Multiprocesamiento simétrico

Arquitectura SMP

_ Se proponen las siguientes categorías de sistemas de computadores:

- Instrucción simple/dato simple (Single Instruction single data - SISD): un solo procesador ejecuta una única instrucción que opera sobre datos almacenados en una sola memoria.
- Instrucción simple/datos múltiples (Single Instruction multiple data - SIMD): una única instrucción de máquina controla la ejecución simultánea de un número de elementos de proceso. Cada elemento de proceso tiene una memoria de datos asociada, de forma que cada instrucción se ejecuta en un conjunto de datos diferente a través de los diferentes procesadores.
- Instrucción múltiple/dato simple (Multiple Instruction single data - MISD): se transmite una secuencia de datos a un conjunto de procesadores, cada uno de los cuales ejecuta una secuencia de instrucciones diferente. Esta estructura nunca se ha implementado.
- Instrucción múltiple/datos múltiples (Multiple Instruction multiple data - MIMD): un conjunto de procesadores ejecuta simultáneamente diferentes secuencias de instrucciones en diferentes conjuntos de datos.

Multiprocesador simétrico (Symmetric Multiprocessor, SMP): en este, el núcleo se puede ejecutar en cualquier procesador, y normalmente cada procesador realiza su propia planificación del conjunto disponible de procesos e hilos. El núcleo puede construirse como múltiples procesos o múltiples hilos, permitiéndose la ejecución de partes del núcleo en paralelo. El enfoque SMP complica al sistema operativo, ya que debe asegurar que dos procesadores no seleccionan un mismo proceso y que no se pierde ningún proceso de la cola. Se deben emplear técnicas para resolver y sincronizar el uso de los recursos.



_ Consideraciones de diseño de un sistema operativo multiprocesador:

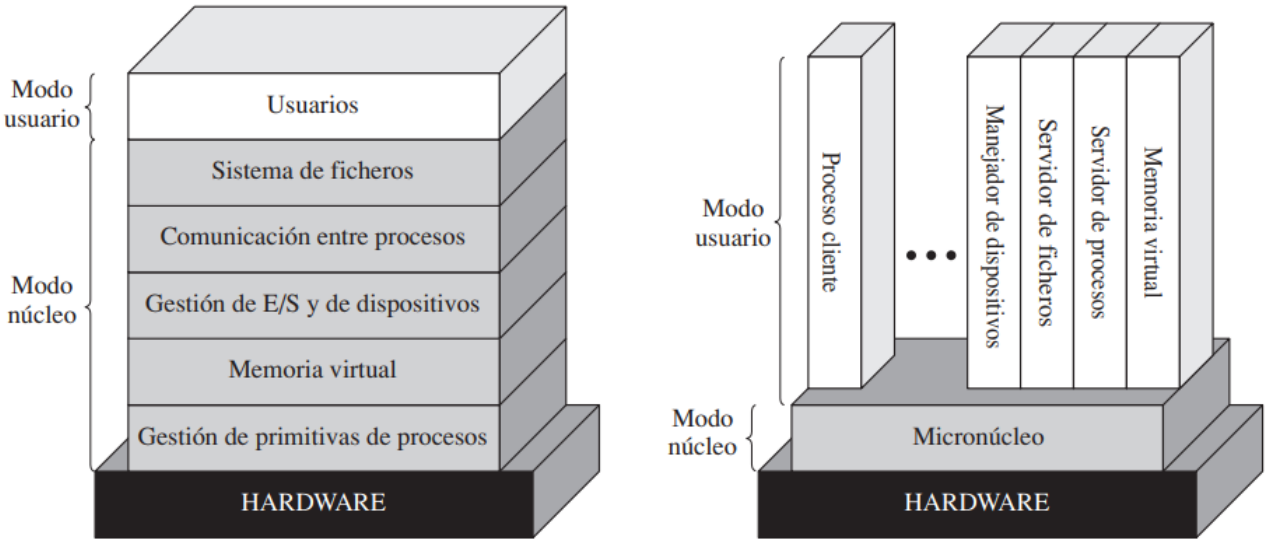
- Procesos o hilos simultáneos concurrentes: las rutinas del núcleo necesitan ser reentrantes para permitir que varios procesadores ejecuten el mismo código del núcleo simultáneamente.
- Planificación: se puede realizar por cualquier procesador, por lo que se deben evitar los conflictos.
- Sincronización: con múltiples procesos activos, que pueden acceder a espacios de direcciones compartidas o recursos compartidos de E/S, se debe tener cuidado en proporcionar una sincronización eficaz.
- Gestión de memoria
- Fiabilidad y tolerancia a fallos: el sistema operativo no se debe degradar en caso de fallo de un procesador.

Micronúcleos

Arquitectura micronúcleo

_ Un micronúcleo es la pequeña parte central de un sistema operativo. La filosofía existente en el micronúcleo es que solamente las funciones absolutamente esenciales del sistema operativo estén en el núcleo. Por otro lado, los servicios y aplicaciones menos esenciales se construyen sobre el micronúcleo y se ejecutan en modo usuario. Muchos servicios que tradicionalmente habían formado parte del sistema operativo ahora son subsistemas que interactúan con el núcleo y entre ellos mismos; algunos ejemplos son:

- Manejadores de dispositivos
- Servidores de archivos
- Gestores de memoria virtual
- Sistemas de ventana
- Servicios de seguridad



(a) Núcleo por capas (b) Micro núcleo

Beneficios de una organización micronúcleo

_ Se pueden encontrar una serie de ventajas del uso de los micronúcleos:

- Interfaz uniforme para las solicitudes realizadas por los procesos: todos los servicios se utilizan mediante paso de mensajes.
- Extensibilidad: permite añadir nuevos servicios.
- Flexibilidad: permite añadir nuevas características y permite reducir las características actuales.
- Portabilidad: en el micronúcleo (y no en los demás servicios) se realizan los cambios necesarios para portar el sistema a un nuevo procesador.
- Fiabilidad: posee un diseño modular, y por otro lado, un pequeño micronúcleo puede probarse de un modo muy riguroso.

- Soporte a sistemas distribuidos: se puede enviar un mensaje sin saber en qué máquina reside el destinatario.
- Sistema operativo orientado a objetos: los componentes son objetos con interfaces claramente definidas que se pueden interconectar para formar un software.

Diseño del micronúcleo

Gestión de memoria a bajo nivel: el micronúcleo tiene que controlar el concepto hardware de espacio de direcciones para hacer posible la implementación de protección a nivel de proceso. Con tal de que el micronúcleo se responsabilice de la asignación de cada página virtual a un marco físico, la parte principal de gestión de memoria, incluyendo la protección del espacio de memoria entre procesos, el algoritmo de reemplazo de páginas y otra lógica de paginación, pueden implementarse fuera del núcleo. Es decir, traduce cada página virtual en un marco de página físico.

Comunicación entre procesos (Interprocess Communication): la forma básica de comunicación entre dos procesos o hilos en un sistema operativo con micronúcleo son los mensajes. Un mensaje incluye una cabecera que identifica a los procesos remitente y receptor y un cuerpo que contiene directamente los datos, un puntero a un bloque de datos, o alguna información de control del proceso.

Gestión de E/S e interrupciones: con una arquitectura micronúcleo es posible manejar las interrupciones hardware como mensajes e incluir los puertos de E/S en los espacios de direcciones. El micronúcleo puede reconocer las interrupciones pero no las puede manejar. Más bien, genera un mensaje para el proceso a nivel de usuario que está actualmente asociado con esa interrupción. De esta forma, cuando se habilita una interrupción, se asigna un proceso de nivel de usuario a esa interrupción y el núcleo mantiene las asociaciones.

Concurrencia: exclusión mutua y sincronización

Introducción

_ Los temas centrales del diseño de sistemas operativos están todos relacionados con la gestión de procesos e hilos:

- Multiprogramación: gestión de múltiples procesos dentro de un sistema monoprocesador.
- Multiprocesamiento: gestión de múltiples procesos dentro de un multiprocesador.
- Procesamiento distribuido: gestión de múltiples procesos que se ejecutan sobre múltiples sistemas de cómputo distribuidos.

Principios de la concurrencia

Concurrencia

_ La concurrencia es fundamental en todas estas áreas y en el diseño del sistema operativo. La concurrencia abarca varios aspectos, entre los cuales están la:

- Comunicación entre procesos.
- Compartición o competencia por recursos.
- Sincronización de la ejecución de múltiples procesos.
- Reserva o asignación de tiempo de procesador para los procesos.

_ Debemos entender que todos estos asuntos no sólo suceden en el entorno del multiprocesamiento y el procesamiento distribuido, sino también en sistemas monoprocesador multiprogramados.

_ La concurrencia aparece en tres contextos diferentes:

- Múltiples aplicaciones: la multiprogramación fue ideada para permitir compartir dinámicamente el tiempo de procesamiento entre varias aplicaciones activas.
- Aplicaciones estructuradas: algunas aplicaciones pueden ser programadas eficazmente como un conjunto de procesos concurrentes.
- Estructura del sistema operativo: las mismas ventajas constructivas son aplicables a la programación de sistemas y, de hecho, los sistemas operativos son a menudo implementados en sí mismos como un conjunto de procesos o hilos.

Dificultades de la concurrencia

_ En primera instancia, puede parecer que el entrelazado y el solapamiento representan modos de ejecución fundamentalmente diferentes y que presentan diferentes problemas. Pero ambas técnicas pueden verse como ejemplos de procesamiento concurrente y ambas presentan los mismos problemas. En el caso de un monoprocesador, los problemas surgen de una característica básica de los sistemas multiprogramados, y es que no puede predecirse la velocidad relativa de ejecución de los procesos. Ésta depende de la actividad de los otros procesos, de la forma en que el sistema operativo maneja las interrupciones y de las políticas de planificación del sistema operativo. Entonces, se plantean las siguientes dificultades:

1. Compartición de recursos globales: la misma está cargada de peligros, ya que, por ejemplo, si dos procesos utilizan ambos la misma variable global y ambos realizan lecturas y escrituras sobre esa variable, entonces el orden en que se ejecuten las lecturas y escrituras es crítico.
2. Gestionar la asignación óptima de recursos: para el sistema operativo esta tarea es complicada, ya que, por ejemplo, el proceso A puede solicitar el uso de un canal concreto de E/S, y serle concedido el control, y luego ser suspendido justo antes de utilizar ese canal por alguna razón.
3. Localizar errores de programación: esto puede llegar a ser muy complicado, porque los resultados son típicamente no deterministas y no reproducibles.

_ Todas las dificultades precedentes se presentan también en un sistema multiprocesador, porque aquí tampoco es predecible la velocidad relativa de ejecución de los procesos.

Ejemplo

_ A continuación tenemos el código de un programa que proporcionará un procedimiento de “eco” de un carácter, en donde la entrada se obtiene del teclado, mediante una tecla cada vez. Cada carácter introducido se almacena en la variable “cent”. Luego se transfiere a la variable “csal” y se envía a la pantalla. Cualquier programa puede llamar repetidamente a este procedimiento para aceptar la entrada del usuario y mostrarla por su pantalla.

```
void eco ()
{
        cent = getchar();
        csal = cent;
        putchar(csal);
}
```

_ Consideramos que tenemos un sistema multiprogramado de un único procesador (monoprocesador) y para un único usuario, en donde el usuario puede saltar de una aplicación a otra, y cada aplicación utiliza el mismo teclado para la entrada y la misma

pantalla para la salida. Dado que cada aplicación necesita usar el procedimiento “eco”, tiene sentido que éste sea un procedimiento compartido que esté cargado en una porción de memoria global para todas las aplicaciones. De este modo, sólo se utiliza una única copia del procedimiento “eco”, para economizar espacio.

_ La compartición de memoria principal entre procesos es útil para permitir una interacción eficiente y próxima entre los procesos. Pero esta interacción puede acarrear problemas. Entonces tenemos la siguiente secuencia:

1. Un proceso P1 invoca el procedimiento “eco” y es interrumpido inmediatamente después de que “getchar” devuelva su valor y sea almacenado en “cent”. En este punto, el carácter introducido más recientemente, es “x”, y está almacenado en “cent”.
2. Luego un proceso P2 se activa e invoca al procedimiento “eco”, que se ejecuta hasta concluir, habiendo leído y mostrado en pantalla un único carácter, “y”.
3. Finalmente se retoma el proceso P1. En este instante, el valor “x” ha sido sobrescrito en “cent” y por lo tanto se ha perdido, ya que en su lugar, “cent” contiene “y”, que es transferido a “csal” y mostrado.

_ Así, el primer carácter se pierde y el segundo se muestra dos veces. La esencia de este problema es la variable compartida global, “cent”, en donde múltiples procesos tienen acceso a esta variable. Si un proceso actualiza la variable global y justo entonces es interrumpido, otro proceso puede alterar la variable antes de que el primer proceso pueda utilizar su valor.

_ Suponemos ahora, que sólo un proceso pueda estar en dicho procedimiento en el momento. Entonces la secuencia anterior arreglada seria:

1. El proceso P1 invoca el procedimiento “eco” y es interrumpido inmediatamente después de que concluya la función de entrada. En este punto, el carácter introducido más recientemente, “x”, está almacenado en “cent”.
2. El proceso P2 se activa e invoca al procedimiento “eco”. Sin embargo, dado que P1 está todavía dentro del procedimiento eco, aunque actualmente suspendido, a P2 se le impide entrar en el procedimiento. Por lo tanto, P2 se suspende esperando la disponibilidad del procedimiento “eco”.
3. En algún momento posterior, el proceso P1 se retoma y completa la ejecución de “eco” y se muestra el carácter correcto, “x”.
4. Cuando P1 sale de “eco”, esto elimina el bloqueo de P2, y cuando P2 sea más tarde retomado, invocará satisfactoriamente el procedimiento “eco”.

_ Por lo tanto, es necesario proteger las variables globales compartidas (así como otros recursos globales compartidos) y que la única manera de hacerlo es controlar el código que accede a la variable. Si imponemos la disciplina de que sólo un proceso al tiempo pueda entrar en eco y que una vez en eco el procedimiento debe ejecutar hasta completarse antes de estar disponible para otro proceso, entonces el tipo de error que se acaba de describir no ocurrirá.

_ Ahora bien, en un sistema multiprocesador, aparecen los mismos problemas de recursos compartidos protegidos, y funcionan las mismas soluciones. Entonces, como primera medida suponemos que no hay mecanismo para controlar los accesos a la variable global compartida:

1. Los procesos P1 y P2 se están ejecutando, cada cual en un procesador distinto. Ambos procesos invocan el procedimiento “eco”.
2. Ocurren los siguientes eventos (los eventos en la misma línea suceden en paralelo):

Proceso P1	**Proceso P2**
•	•
cent = getchar();	•
•	cent = getchar();
csal = cent;	csal = cent;
putchar(csal);	•
•	putchar(csal);
•	•

_ El resultado es que el carácter introducido a P1 se pierde antes de ser mostrado, y el carácter introducido a P2 es mostrado por ambos P1 y P2. Entonces, si añadimos la capacidad de cumplir la disciplina de que sólo un proceso al tiempo pueda estar en “eco” sucede la siguiente secuencia:

1. Los procesos P1 y P2 están ambos ejecutando, cada cual en un procesador distinto. Ambos procesos invocan al procedimiento “eco”.
2. Mientras P1 está dentro del procedimiento “eco”, P2 invoca a “eco”. Dado que P1 está todavía dentro del procedimiento “eco” (ya estando P1 suspendido o ejecutando), a P2 se le bloqueará la entrada al procedimiento. Por lo tanto, P2 se suspende en espera de la disponibilidad del procedimiento “eco”.
3. En algún momento posterior, el proceso P1 completa la ejecución de “eco”, sale del procedimiento y continúa ejecutando. Inmediatamente después de que P1 salga de “eco”, se retoma P2 que comienza la ejecución de “eco”.

_ En el caso de un sistema monoprocesador, el motivo por el que se tiene un problema es que una interrupción puede parar la ejecución de instrucciones en cualquier punto de un proceso. En el caso de un sistema multiprocesador, se tiene el mismo motivo y, además, puede suceder porque dos procesos pueden estar ejecutando simultáneamente y ambos intentando acceder a la misma variable global. Sin embargo, la solución a ambos tipos de problema es la misma, controlar los accesos a los recursos compartidos.

Condición de carrera

_ Una condición de carrera sucede cuando múltiples procesos o hilos leen y escriben datos de manera que el resultado final depende del orden de ejecución de las instrucciones en los múltiples procesos. Vemos un ejemplo:

- Suponemos que dos procesos, P1 y P2, comparten la variable global “a”. En algún punto de su ejecución, P1 actualiza a al valor 1 y, en el mismo punto en su ejecución, P2 actualiza a al valor 2. Así, las dos tareas compiten en una carrera por escribir la variable “a”. En este ejemplo el “perdedor” de la carrera (el proceso que actualiza el último) determina él valor de “a”.

Preocupaciones del sistema operativo

_ Debido a la existencia de la concurrencia, las necesidades o aspectos de diseño y gestión que surgen son:

1. El sistema operativo debe ser capaz de seguir la pista de varios procesos.
2. El sistema operativo debe asignar y retirar varios recursos para cada proceso activo. Estos recursos incluyen:

- Tiempo de procesador
- Memoria
- Ficheros
- Dispositivos de E/S

3. El sistema operativo debe proteger los datos y recursos físicos de cada proceso frente a interferencias involuntarias de otros procesos.
4. El funcionamiento de un proceso y el resultado que produzca, debe ser independiente de la velocidad a la que suceda su ejecución en relación con la velocidad de otros procesos concurrentes.

Interacción de procesos

_ Podemos clasificar las formas en que los procesos interaccionan en base al grado en que perciben la existencia de cada uno de los otros. Tenemos tres posibles grados de percepción más las consecuencias de cada uno:

- Procesos que no se perciben entre sí: son procesos independientes que no se pretende que trabajen juntos, por ejemplo, dos aplicaciones independientes pueden querer ambas acceder al mismo disco, fichero o impresora. El sistema operativo debe regular estos accesos.
- Procesos que se perciben indirectamente entre sí: son procesos que no están necesariamente al tanto de la presencia de los demás mediante sus respectivos ID de proceso, pero que comparten accesos a algún objeto, como un buffer de E/S. Tales procesos exhiben cooperación en la compartición del objeto común.
- Procesos que se perciben directamente entre sí: son procesos capaces de comunicarse entre sí vía el ID del proceso y que son diseñados para trabajar conjuntamente en cierta actividad. Tales procesos exhiben cooperación.

Competencia entre procesos por recursos: Los procesos concurrentes entran en conflicto entre ellos cuando compiten por el uso del mismo recurso. No hay intercambio de información entre los procesos en competencia. No obstante, la ejecución de un proceso puede afectar al comportamiento de los procesos en competencia. En concreto, si dos procesos desean ambos acceder al mismo recurso único, entonces, el sistema operativo reservará el recurso para uno de ellos, y el otro tendrá que esperar. Por tanto, el proceso al que se le deniega el acceso será ralentizado. En un caso extremo, el proceso bloqueado puede no conseguir nunca el recurso y por tanto no terminar nunca satisfactoriamente.

_ En el caso de procesos en competencia, deben afrontarse tres problemas de control:

- Necesidad de exclusión mutua: Supóngase que dos o más procesos requieren acceso a un recurso único no compartible, como una impresora. Durante el curso de la ejecución, cada proceso estará enviando mandatos al dispositivo de E/S, recibiendo información de estado, enviando datos o recibiendo datos. Nos referiremos a tal recurso como un recurso crítico, y a la porción del programa que lo utiliza como la sección crítica del programa. Es importante que solo un programa pueda acceder a su sección crítica en un momento dado. No podemos simplemente delegar en el sistema operativo para que entienda y aplique esta restricción porque los detalles de los requisitos pueden no ser obvios. En el caso de una impresora, por ejemplo, solo se permite que un proceso envíe una orden a la impresora en un momento dado, ya que de otro modo, las líneas de los procesos en competencia se intercalarían.
- Interbloqueo
- Inanición

Cooperación entre procesos vía compartición: este caso cubre procesos que interaccionan con otros procesos sin tener conocimiento explícito de ellos. Por ejemplo, múltiples procesos pueden tener acceso a variables compartidas o a ficheros o bases de datos compartidas. Los procesos pueden usar y actualizar los datos compartidos sin referenciar otros procesos pero saben que otros procesos pueden tener acceso a los mismos datos. Así, los procesos deben cooperar para asegurar que los datos que comparten son manipulados adecuadamente.

- Los mecanismos de control deben asegurar la integridad de los datos compartidos.
- Dado que los datos están contenidos en recursos (dispositivos, memoria), los problemas de control de exclusión mutua, interbloqueo e inanición están presentes de nuevo. La única diferencia es que los datos individuales pueden ser accedidos de dos maneras diferentes, lectura y escritura, y sólo las operaciones de escritura deben ser mutuamente exclusivas.

Cooperación entre procesos vía comunicación: cuando los procesos cooperan vía comunicación, en cambio, los diversos procesos involucrados participan en un esfuerzo común que los vincula a todos ellos. La comunicación proporciona una manera de sincronizar o coordinar actividades varias.

- Típicamente, la comunicación se fundamenta en mensajes de algún tipo, proporcionados como parte del lenguaje de programación o por el núcleo del sistema operativo.
- Dado que en el acto de pasar mensajes los procesos no comparten nada, la exclusión mutua no es un requisito de control en este tipo de cooperación.
- Sin embargo, pueden surgir problemas de interbloqueo, en donde Cada proceso puede estar esperando una comunicación del otro, o problemas de inanición, en donde, Dos procesos se están mandando mensajes mientras que otro proceso está esperando recibir un mensaje.

Requisitos para la exclusión mutua

_ Cualquier mecanismo o técnica que vaya a proporcionar exclusión mutua debería cumplimentar los siguientes requisitos:

1. La exclusión mutua debe hacerse cumplir: solo un proceso debe tener permiso para entrar en la sección crítica por un recurso en un instante dado.
2. Un proceso que se interrumpe en una sección crítica debe hacerlo sin interferir con los otros procesos.
3. No puede permitirse el interbloqueo o la inanición.
4. Cuando ningún proceso esté en una sección crítica, a cualquier proceso que solicite entrar en su sección crítica debe permitírsele entrar sin demora.
5. No se deben hacer suposiciones sobre las velocidades relativas de los procesos ni sobre el número de procesadores.
6. Un proceso permanece dentro de su sección crítica sólo por un tiempo finito.

Exclusión mutua: soporte hardware

Primer intento

Espera activa: todos los otros procesos que intenten entrar en su sección crítica caen en un modo de espera activa. Este término se refiere a una técnica en la cual un proceso no puede hacer nada hasta obtener permiso para entrar en su sección crítica, pero continúa ejecutando una instrucción o conjunto de instrucciones que comprueban la variable apropiada para conseguir entrar.

Corrutinas

_ Están diseñadas para poder pasar el control de la ejecución entre ellas.

- No es una técnica apropiada para dar soporte al procesamiento concurrente.

Segundo intento

_ Cada proceso puede examinar el estado del otro pero no lo puede alterar. Cuando un proceso desea entrar en su sección crítica comprueba en primer lugar el otro proceso. Si no hay otro proceso en su sección crítica fija su estado para la sección crítica.

- Este método no garantiza la exclusión mutua.
- Cada proceso puede comprobar las señales y luego entra en la sección crítica al mismo tiempo.

Tercer intento

_ Dar valor a la señal para entrar en la sección crítica antes de comprobar otros procesos.

- Si hay otro proceso en la sección crítica cuando se ha dado valor a la señal, el proceso queda bloqueado hasta que el otro proceso abandona la sección crítica.
- Se provocará un interbloqueo cuando dos procesos den valor a sus señales antes de entrar en la sección crítica. Entonces cada proceso debe esperar a que el otro abandone la sección crítica.

Cuarto intento

_ Un proceso activa su señal para indicar que desea entrar en la sección crítica, pero debe estar listo para desactivar la variable señal.

- Se comprueban los otros procesos. Si están en la sección crítica, la señal se desactiva y luego se vuelve a activar para indicar que desea entrar en la sección crítica. Esta operación se repite hasta que el proceso puede entrar en la sección crítica.
- Es posible que cada proceso active su señal, compruebe otros procesos y desactive su señal. Esta situación no se mantendrá por mucho tiempo, de modo que no se trata de un interbloqueo. Así pues, se rechaza el cuarto intento.

Una solución correcta

_ Cada proceso tiene un turno para entrar en la sección crítica.

- Si un proceso desea entrar en la sección crítica, debe activar su señal y puede que tenga que esperar a que llegue su turno.

Semáforos

Concepto

_ El principio fundamental de los semáforos es que dos o más procesos pueden cooperar por medio de simples señales, tales que un proceso pueda ser obligado a parar en un lugar específico hasta que haya recibido una señal específica. Para la señalización, se utilizan unas variables especiales llamadas semáforos.

- Para transmitir una señal vía el semáforo "s", el proceso ejecutará la primitiva semSignal(s).
- Para recibir una señal vía el semáforo "s", el proceso ejecutará la primitiva semWait(s).
- Si la correspondiente señal no se ha transmitido todavía, el proceso se suspenderá hasta que la transmisión tenga lugar.

_ El semáforo puede ser visto como una variable que contiene un valor entero sobre el cual sólo están definidas tres operaciones:

1. Un semáforo puede ser inicializado a un valor no negativo.
2. La operación semWait decrementa el valor del semáforo. Si el valor pasa a ser negativo, entonces el proceso que está ejecutando semWait se bloquea. En otro caso, el proceso continúa su ejecución.
3. La operación semSignal incrementa el valor del semáforo. Si el valor es menor o igual que cero, entonces se desbloquea uno de los procesos bloqueados en la operación semWait.

_ Aparte de estas tres operaciones no hay manera de inspeccionar o manipular un semáforo. Las primitivas semWait y semSignal se asumen atómicas.

```
struct semaphore {
    int cuenta;
    queueType cola;
}
void semWait(semaphore s)
{
    s.cuenta—;
    if (s.cuenta < 0)
    {
        poner este proceso en s.cola;
        bloquear este proceso;
    }
}
void semSignal(semaphore s)
{
    s.cuenta++;
    if (s.cuenta <= 0)
    {
        extraer un proceso P de s.cola;
        poner el proceso P en la lista de listos;
    }
}
```

Semáforo binario: una versión más restringida, es conocida como semáforo binario o mutex. Un semáforo binario sólo puede tomar los valores 0 y 1 y se puede definir por las siguientes tres operaciones:

1. Un semáforo binario puede ser inicializado a 0 o 1.
2. La operación semWaitB comprueba el valor del semáforo. Si el valor es cero, entonces el proceso que está ejecutando semWaitB se bloquea. Si el valor es uno, entonces se cambia el valor a cero y el proceso continúa su ejecución.
3. La operación semSignalB comprueba si hay algún proceso bloqueado en el semáforo. Si lo hay, entonces se desbloquea uno de los procesos bloqueados en la operación semWaitB. Si no hay procesos bloqueados, entonces el valor del semáforo se pone a uno.

```
struct binary_semaphore {
    enum {cero, uno} valor;
    queueType cola;
};
void semWaitB(binary_semaphore s)
{
    if (s.valor == 1)
        s.valor = 0;
    else
        {
            poner este proceso en s.cola;
            bloquear este proceso;
    }
}
void semSignalB(binary_semaphore s)
{
    if (esta_vacia(s.cola))
        s.valor = 1;
    else
    {
            extraer un proceso P de s.cola;
            poner el proceso P en la lista de listos;
    }
}
```

_ En principio debería ser más fácil implementar un semáforo binario, y puede demostrarse que tiene la misma potencia expresiva que un semáforo general. Para contrastar los dos tipos de semáforos, el semáforo no-binario es a menudo referido como semáforo con contador o semáforo general.

_ Para ambos, semáforos con contador y semáforos binarios, se utiliza una cola para mantener los procesos esperando por el semáforo.

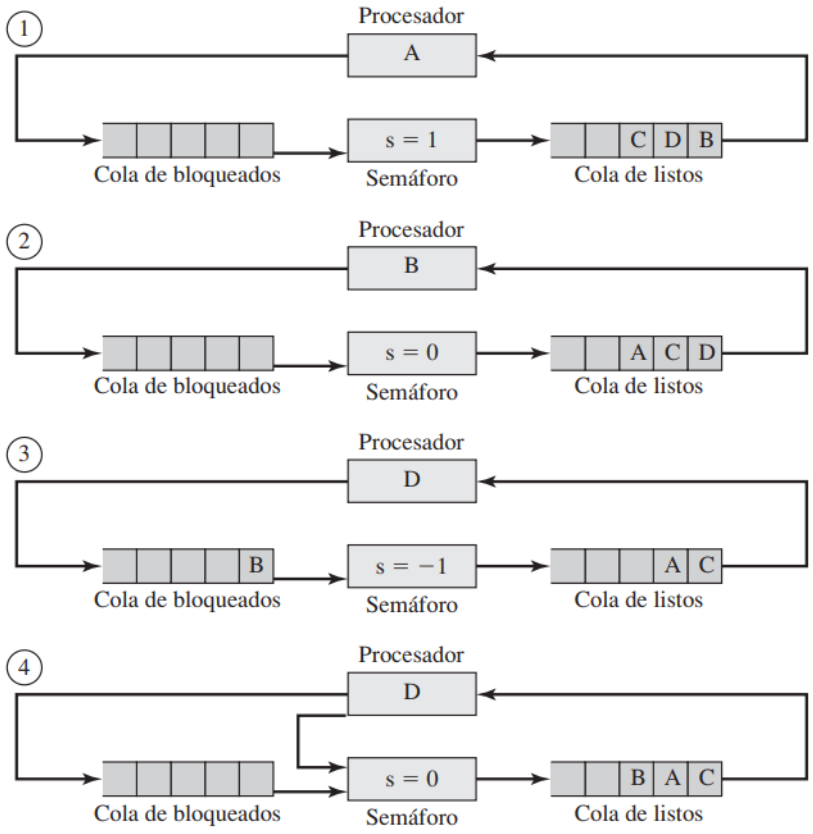
_ Surge la cuestión sobre el orden en que los procesos deben ser extraídos de tal cola. Se utiliza la política FIFO (First In First Out), en donde el proceso que lleve más tiempo bloqueado es el primero en ser extraído de la cola.

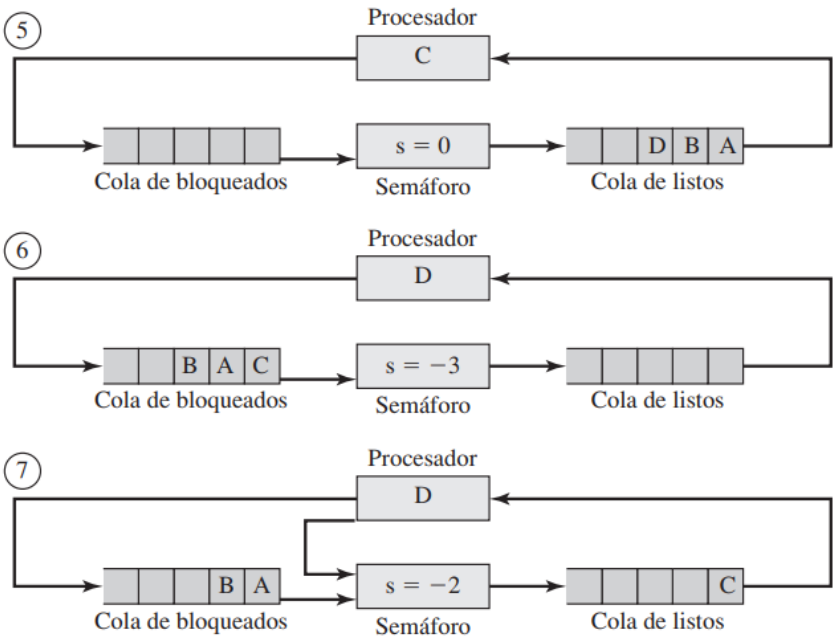
Semáforo fuerte: es un semáforo cuya definición incluye la política FIFO.

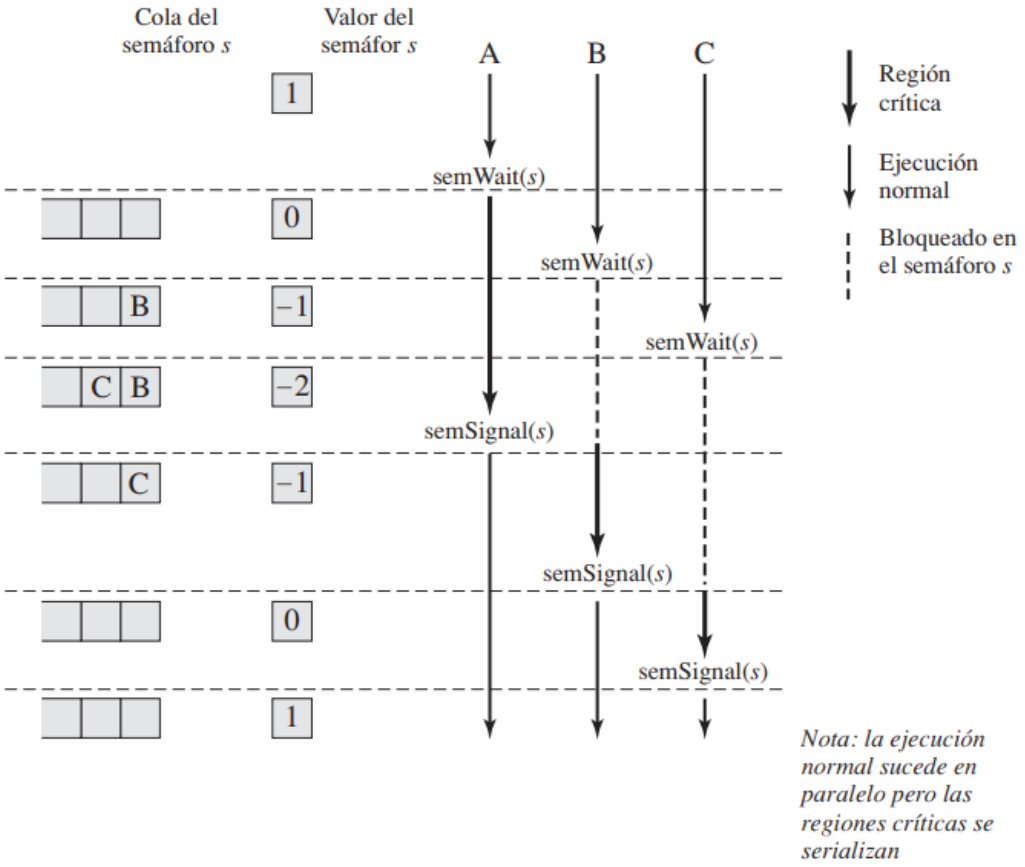
Semáforo débil: es un semáforo que no especifica el orden en que los procesos son extraídos de la cola.

_ El siguiente grafico es un ejemplo de la operación de un semáforo fuerte. Vemos que los procesos A, B y C dependen de un resultado del proceso D.

- Inicialmente, en el paso (1), A se está ejecutando; B, C y D están listos; y el contador del semáforo es 1, indicando que uno de los resultados de D está disponible. Cuando A realiza una instrucción semWait sobre el semáforo s, el semáforo se decrementa a 0 y A puede continuar ejecutando; posteriormente se adjunta a la lista de listos.
- Entonces B se ejecuta en el paso (2), finalmente realiza una instrucción semWait y es bloqueado, permitiendo que D se ejecute en el paso (3).
- Cuando D completa un nuevo resultado, realiza una instrucción semSignal, que permite a B moverse a la lista de listos como vemos en el paso (4).
- D se adjunta a la lista de listos y C comienza a ejecutarse en el paso (5) pero se bloquea cuando realiza una instrucción semWait. De manera similar, A y B ejecutan y se bloquean en el semáforo, permitiendo a D retomar la ejecución en el paso (6).
- Cuando D tiene un resultado realiza un semSignal, que transfiere a C a la lista de listos. Posteriores ciclos de D liberarán A y B del estado Bloqueado.







Nota: la ejecución normal sucede en paralelo pero las regiones críticas se serializan

Problema productor/consumidor

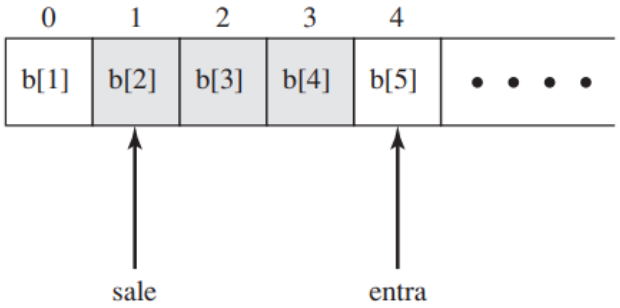
_ El enunciado general es éste: hay uno o más procesos generando algún tipo de datos (registros, caracteres) y poniéndolos en un buffer. Hay un único consumidor que está extrayendo datos de dicho buffer de uno en uno. El sistema está obligado a impedir la superposición de las operaciones sobre los datos. Solo un agente (productor o consumidor) puede acceder al buffer en un momento dado.

_ Para empezar, asumimos que el buffer es infinito y consiste en un vector de datos. En términos abstractos, las funciones productor y consumidor se definen como sigue:

```
productor:
while (true)
{
    /* producir dato v */;
    while ((entra + 1) % n == sale)
        /* no hacer nada */;
    b[entra] = v;
    entra = (entra + 1) % n;
}
```

```
consumidor:
while (true)
{
    while (entra == sale)
    /* no hacer nada */;
    w = b[sale];
    sale = (sale + 1) % n;
    /* consumir dato w */
}
```

_ Vemos la estructura del buffer b. El productor puede generar datos y almacenarlos en el vector a su propio ritmo. Cada vez, se incrementa un índice (entra) sobre el vector. El consumidor procede de manera similar pero debe asegurarse de que no intenta leer de una entrada vacía del vector. Por lo tanto, el consumidor se asegura de que el productor ha avanzado más allá que él (entra > sale) antes de seguir.



Nota: el área sombreada indica la porción del *buffer* que está ocupada

_ Se intenta implementar este sistema utilizando semáforos binarios, en donde en vez de tratar con los índices “entra” y “sale”, simplemente guardamos constancia del número de datos en el buffer usando la variable entera “n” (= entra – sale). El semáforo “s” se utiliza para cumplir la exclusión mutua; el semáforo retardo se usa para forzar al consumidor a esperar (semWait) si el buffer está vacío.

_ Este problema se usa comúnmente para demostrar la aplicación de semáforos para controlar el acceso a recursos compartidos y evitar problemas como la condición de carrera y la inanición.

_ En este caso, hay uno o varios procesos productores que generan datos y los colocan en un búfer ilimitado (como una cola), y hay procesos consumidores que retiran esos datos del búfer para su procesamiento. La dificultad principal radica en asegurar que los productores no intenten poner datos en el búfer si está lleno, y que los consumidores no intenten retirar datos si el búfer está vacío. Además, se debe evitar que múltiples productores o consumidores accedan al búfer al mismo tiempo, lo que podría llevar a problemas de consistencia o pérdida de datos.

_ Entonces, los semáforos se utilizan para controlar el acceso al búfer. Se pueden utilizar dos semáforos, uno para representar el espacio disponible en el búfer (llamémoslo espacio_disponible) y otro para representar los elementos presentes en el búfer (llamémoslo elementos_presentes):

- Los productores deben decrementar espacio_disponible antes de colocar un elemento en el búfer y luego incrementar elementos_presentes para indicar que hay datos disponibles.
- Mientras tanto, los consumidores deben decrementar elementos_presentes antes de retirar un elemento del búfer y luego incrementar espacio_disponible para indicar que hay espacio disponible para más datos.

Monitores

Introducción

_ Los semáforos proporcionan una herramienta potente y flexible para conseguir la exclusión mutua y para la coordinación de procesos. Sin embargo, puede ser difícil producir un programa correcto utilizando semáforos. La dificultad es que las operaciones semWait y semSignal pueden estar dispersas a través de un programa y no resulta fácil ver el efecto global de estas operaciones sobre los semáforos a los que afectan.

_ El monitor es una construcción del lenguaje de programación que proporciona una funcionalidad equivalente a la de los semáforos pero es más fácil de controlar.

Monitor con señal

Monitor: es un módulo software consistente en uno o más procedimientos, una secuencia de inicialización y datos locales. Las principales características de un monitor son las siguientes:

1. Las variables locales de datos son sólo accesibles por los procedimientos del monitor y no por ningún procedimiento externo.
2. Un proceso entra en el monitor invocando uno de sus procedimientos.
3. Sólo un proceso se puede estar ejecutando dentro del monitor en un tiempo determinado, y cualquier otro proceso que haya invocado al monitor se bloquea, en espera de que el monitor quede disponible.

_ Al cumplir la disciplina de sólo un proceso al mismo tiempo, el monitor es capaz de proporcionar exclusión mutua fácilmente. Las variables de datos en el monitor sólo pueden ser accedidas por un proceso a la vez. Así, una estructura de datos compartida puede ser protegida colocándola dentro de un monitor. Si los datos en el monitor representan cierto recurso, entonces el monitor proporciona la función de exclusión mutua en el acceso al recurso.

_ Para ser útil para la programación concurrente, el monitor debe incluir herramientas de sincronización. Por ejemplo, supongamos que un proceso que invoca a un monitor y mientras está en él, deba bloquearse hasta que se satisfaga cierta condición. Se precisa una funcionalidad mediante la cual el proceso no sólo se bloquee, sino que libere el monitor para que algún otro proceso pueda entrar en él. Más tarde, cuando la condición se haya satisfecho y el monitor esté disponible nuevamente, el proceso debe poder ser retomado y permitida su entrada en el monitor en el mismo punto en que se suspendió.

_ Un monitor soporta la sincronización mediante el uso de variables condición que están contenidas dentro del monitor y son accesibles sólo desde el monitor. Las variables condición son un tipo de datos especial en los monitores que se manipula mediante dos funciones:

- cwait(c): suspende la ejecución del proceso llamante en la condición “c”. El monitor queda disponible para ser usado por otro proceso.
- csignal(c): retoma la ejecución de algún proceso bloqueado por un “cwait” en la misma condición. Si hay varios procesos, elige uno de ellos; si no hay ninguno, no hace nada.

_ Las operaciones wait y signal de los monitores son diferentes de las de los semáforos. Si un proceso en un monitor señala y no hay ningún proceso esperando en la variable condición, la señal se pierde.

Paso de mensajes

_ Cuando los procesos interaccionan entre sí, deben satisfacerse dos requisitos fundamentales, sincronización y comunicación. Los procesos necesitan ser sincronizados para conseguir exclusión mutua, y los procesos cooperantes pueden necesitar intercambiar información. Un enfoque que proporciona ambas funciones es el paso de mensajes, el cual es un refuerzo de la exclusión mutua.

_ La funcionalidad real del paso de mensajes se proporciona normalmente en forma de un par de primitivas:

- send(destino, mensaje)
- receive(origen, mensaje)

_ Este es el conjunto mínimo de operaciones necesarias para que los procesos puedan entablar paso de mensajes. El proceso envía información en forma de un mensaje a otro proceso designado por destino. El proceso recibe información ejecutando la primitiva receive, indicando la fuente y el mensaje.

Sincronización

_ La comunicación de un mensaje entre dos procesos implica cierto nivel de sincronización entre los dos. El receptor no puede recibir un mensaje hasta que no lo haya enviado otro proceso. Cuando una primitiva send se ejecuta en un proceso, hay dos posibilidades, o el proceso que envía se bloquea hasta que el mensaje se recibe o no se bloquea. De igual modo, cuando un proceso realiza la primitiva receive, hay dos posibilidades:

1. Si el mensaje fue enviado previamente, el mensaje será recibido y la ejecución continúa.
2. Si no hay mensajes esperando, entonces:
   - El proceso se bloquea hasta que el mensaje llega
   - O el proceso continúa ejecutando, abandonando el intento de recepción.

_ Así, ambos emisor y receptor pueden ser bloqueantes o no bloqueantes. Son tres las combinaciones típicas:

- Envío bloqueante, recepción bloqueante: el emisor y receptor se bloquean hasta que el mensaje se entrega. A esto también se le conoce normalmente como rendezvous.
- Envío no bloqueante, recepción bloqueante: aunque el emisor puede continuar, el receptor se bloqueará hasta que el mensaje solicitado llegue. Esta es probablemente la combinación más útil, ya que permite que un proceso envíe uno o más mensajes a varios destinos tan rápido como sea posible.
- Envío no bloqueante, recepción no bloqueante: ninguna de las partes tiene que esperar.

Direccionamiento

_ Es necesario tener una manera de especificar en la primitiva de envío, qué procesos deben recibir el mensaje. La mayor parte de las implementaciones permiten al proceso receptor indicar el origen del mensaje a recibir. Los diferentes esquemas para especificar procesos en las primitivas send y receive caben dentro de dos categorías:

Direccionamiento directo:

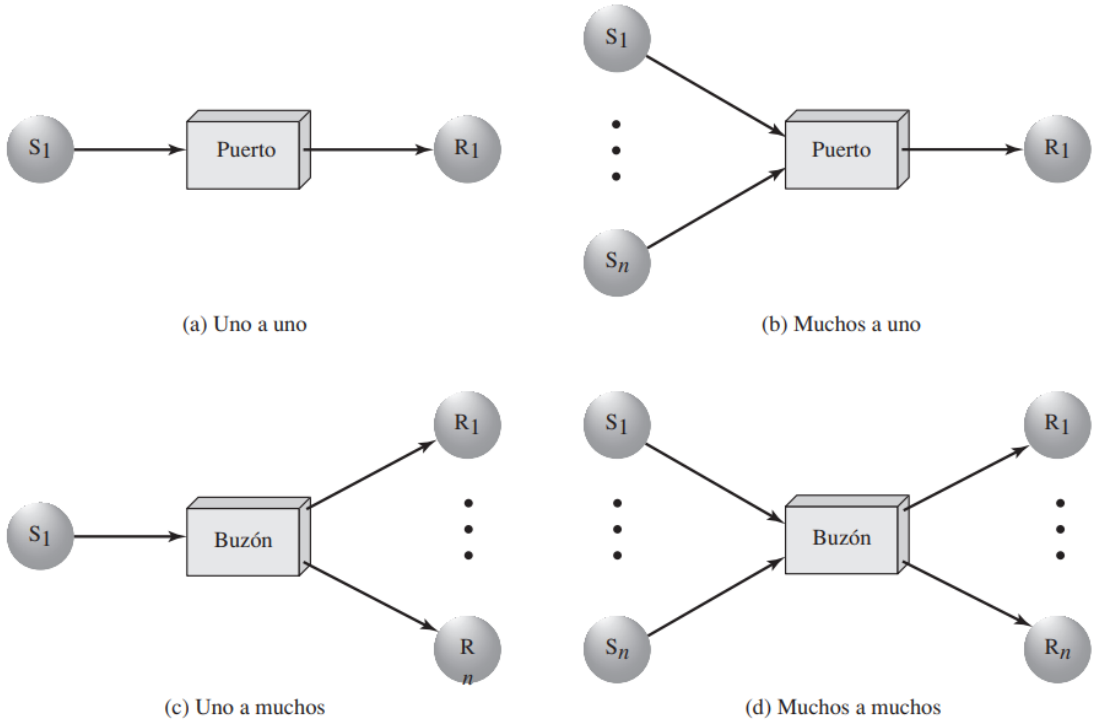
- La primitiva send incluye un identificador específico del proceso destinatario.
- La primitiva receive puede ser manipulada de dos maneras:
   - El proceso deba designar explícitamente un proceso emisor. Así, el proceso debe conocer con anticipación de qué proceso espera el mensaje.

- Uso de direccionamiento implícito: en este caso, la primitiva receive puede utilizar el parámetro origen para devolver un valor cuando se haya realizado la operación de recepción.

Direccionamiento indirecto:

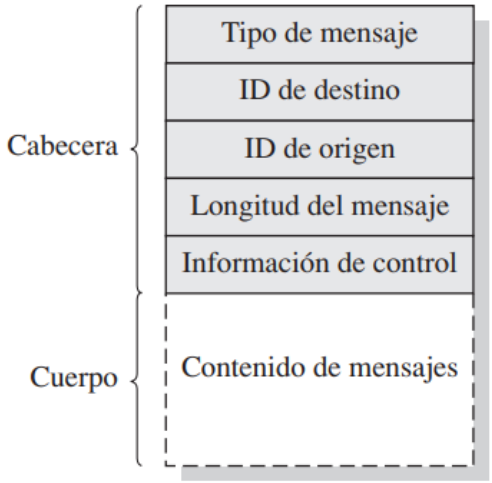
- Los mensajes no se envían directamente por un emisor a un receptor sino que son enviados a una estructura de datos compartida que consiste en colas que pueden contener mensajes temporalmente. Estas colas se conocen generalmente como buzones (mailboxes).
- En caso de que dos procesos se comuniquen, un proceso envía un mensaje al buzón apropiado y otro proceso toma el mensaje del buzón.

_ Una virtud del uso del direccionamiento indirecto es que, desacoplando emisor y receptor, se permite una mayor flexibilidad en el uso de mensajes. La relación entre emisores y receptores puede ser uno-a-uno, muchos-a-uno, uno-a-muchos o muchos-a-muchos.



Formato de mensaje

_ En algunos sistemas operativos, los diseñadores han preferido mensajes cortos de longitud fija para minimizar la sobrecarga de procesamiento y almacenamiento. Si se va a transferir una gran cantidad de datos, los datos pueden estar dispuestos en un archivo y el mensaje puede simplemente indicar el archivo. Una solución más sencilla es permitir mensajes de longitud variable. A continuación vemos un formato típico de mensaje para un sistema operativo que proporciona mensajes de longitud variable:



_ El mensaje está dividido en dos partes:

- Cabecera: contiene información acerca del mensaje.
- Cuerpo: contiene el contenido real del mensaje.

Concurrencia: Interbloqueo e inanición

Fundamentos del interbloqueo

Interbloqueo

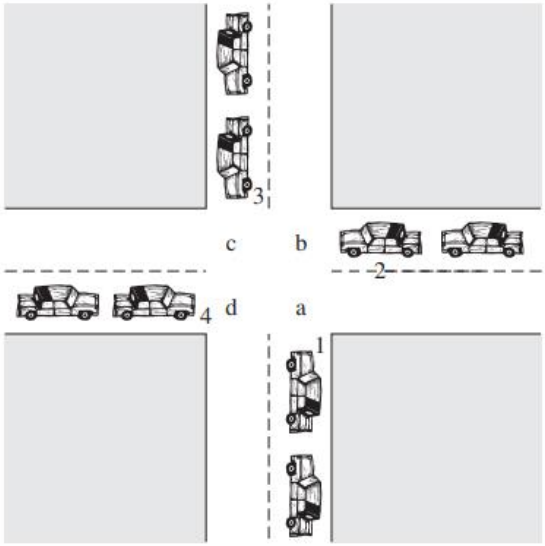
_ Se puede definir el interbloqueo como el bloqueo permanente de un conjunto de procesos que o bien compiten por recursos del sistema o se comunican entre sí. Un conjunto de procesos está interbloqueado cuando cada proceso del conjunto está bloqueado esperando un evento (normalmente la liberación de algún recurso requerido) que sólo puede generar otro proceso bloqueado del conjunto.

- El interbloqueo es permanente porque no puede producirse ninguno de los eventos.
- A diferencia de otros problemas en la gestión de procesos concurrentes, no hay una solución eficiente para esto.
- Todos los interbloqueos involucran necesidades conflictivas que afectan a los recursos de dos o más procesos.

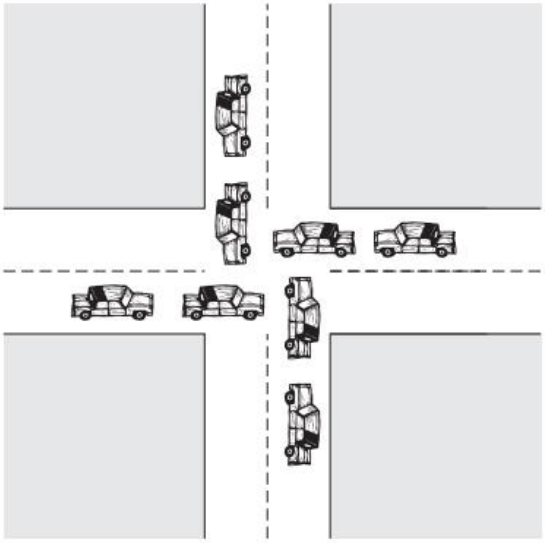
_ Un ejemplo habitual es el interbloqueo del tráfico, y como podemos ver en el gráfico, tenemos una situación en la que cuatro autos han llegado aproximadamente al mismo tiempo a una intersección donde confluyen cuatro caminos. Los cuatro cuadrantes de la intersección son los recursos que hay que controlar. En particular, si los cuatro coches desean cruzar la intersección, los requisitos de recursos son los siguientes:

- El coche 1, que viaja hacia el norte, necesita los cuadrantes a y b.
- El coche 2 necesita los cuadrantes b y c.
- El coche 3 necesita los cuadrantes c y d.

- El coche 4 necesita los cuadrantes d y a.



(a) Posible interbloqueo

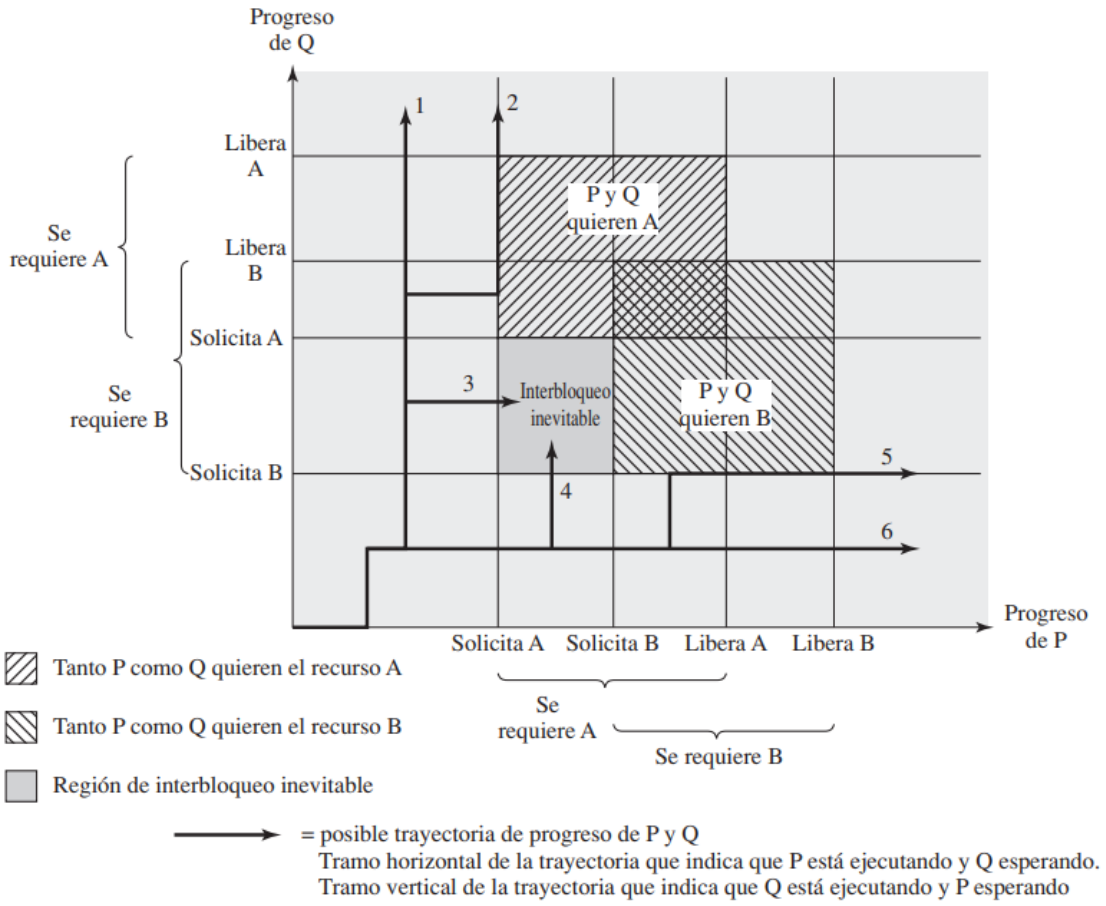

(b) Interbloqueo

_ La norma de circulación habitual es que un coche en un cruce de cuatro caminos debería dar preferencia a otro coche que está justo a su derecha. Esta regla funciona si hay sólo dos o tres coches en la intersección. Por ejemplo, si sólo llegan a la intersección los coches que vienen del norte y del oeste, el del norte esperará y el del oeste proseguirá. Sin embargo, si todos los coches llegan aproximadamente al mismo tiempo, cada uno se abstendrá de cruzar la intersección, produciéndose un interbloqueo.

_ Si los cuatro coches olvidan las normas y entran (cuidadosamente) en la intersección, cada uno posee un recurso (un cuadrante) pero no pueden continuar porque el segundo recurso requerido ya se lo ha apoderado otro coche. De nuevo, se ha producido un interbloqueo. Nótese también que debido a que cada coche tiene justo detrás otro coche, no es posible dar marcha atrás para eliminar el interbloqueo.

_ A continuación, se examina un diagrama de interbloqueo involucrando procesos y recursos del computador. Este gráfico, llamado diagrama de progreso conjunto, muestra el progreso de dos procesos compitiendo por dos recursos. Cada proceso necesita el uso exclusivo de ambos recursos durante un cierto periodo de tiempo. Suponga que hay dos procesos, P y Q, que tienen la siguiente estructura general:

Proceso P	**Proceso Q**
...	...
Solicita A	Solicita B
...	...
Solicita B	Solicita A
...	...
Libera A	Libera B
...	...
Libera B	Libera A
...	...



_ Vemos que el eje x representa el progreso en la ejecución de P, mientras que el eje y representa el de Q. El progreso conjunto de los dos procesos se representa por tanto por una trayectoria que avanza desde el origen en dirección nordeste. Se muestran áreas en las que tanto P como Q requieren el recurso A (líneas ascendentes), áreas en las que ambos procesos requieren el recurso B (líneas descendentes), y áreas en las que ambos requieren ambos recursos.

_ Debido a que se asume que cada proceso requiere el control exclusivo de un recurso, todas éstas son regiones prohibidas; es decir, es imposible que cualquier trayectoria que represente el progreso de la ejecución conjunta de P y Q entre en una de estas regiones. El grafico muestra seis diferentes trayectorias de ejecución, que se pueden resumir de la siguiente manera:

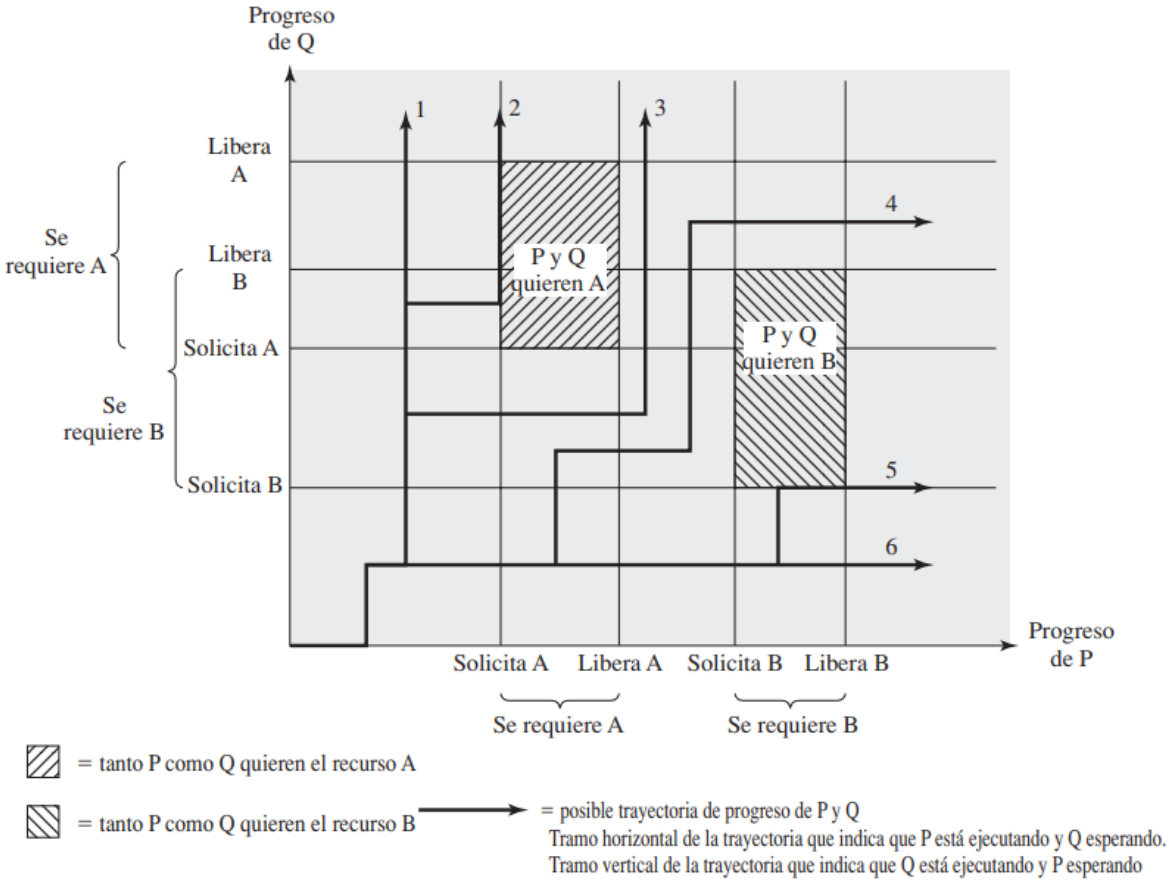
1. Q adquiere B y, a continuación, A, y, más tarde, libera B y A. Cuando P continúe su ejecución, será capaz de adquirir ambos recursos.
2. Q adquiere B y, a continuación, A. P ejecuta y se bloquea al solicitar A. Q libera B y A. Cuando P continúe su ejecución, será capaz de adquirir ambos recursos.
3. Q adquiere B y, a continuación, P adquiere A. El interbloqueo es inevitable, puesto que cuando la ejecución continúe, Q se bloqueará a la espera de A y P a la de B.

4. P adquiere A y, a continuación, Q adquiere B. El interbloqueo es inevitable, puesto que cuando la ejecución continúe, Q se bloqueará a la espera de A y P a la de B.
5. P adquiere A y, a continuación, B. Q ejecuta y se bloquea al solicitar B. P libera A y B. Cuando Q continúe su ejecución, será capaz de adquirir ambos recursos.
6. P adquiere A y, a continuación, B, y, más tarde, libera A y B. Cuando Q continúe su ejecución, será capaz de adquirir ambos recursos.

Región fatal: área sombreada en gris en la gráfico, y está relacionada con el comentario realizado sobre las trayectorias 3 y 4. Si una trayectoria de ejecución entra en esta región fatal, el interbloqueo es inevitable. El interbloqueo es sólo inevitable si el progreso conjunto de los dos procesos crea una trayectoria que entra en la región fatal.

_ La aparición de un interbloqueo depende tanto de la dinámica de la ejecución como de los detalles de la aplicación. Por ejemplo, supóngase que P no necesitase ambos recursos al mismo tiempo de manera que los dos procesos tuvieran la siguiente estructura:

Proceso P	**Proceso Q**
...	...
Solicita A	Solicita B
...	...
Libera A	Solicita A
...	...
Solicita B	Libera B
...	...
Libera B	Libera A
...	...



Recursos

_ Pueden distinguirse dos categorías de recursos:

Reutilizable: es aquél que sólo lo puede utilizar de forma segura un proceso en cada momento y que no se destruye después de su uso. Los procesos obtienen unidades del recurso que más tarde liberarán para que puedan volver a usarlas otros procesos. Algunos ejemplos de recursos reutilizables incluyen procesadores, canales de E/S, memoria principal y secundaria, dispositivos, y estructuras de datos como ficheros, bases de datos y semáforos.

_ Como un ejemplo de recursos reutilizables involucrados en un interbloqueo, consideramos dos procesos que compiten por el acceso exclusivo a un fichero de disco D y a una unidad de cinta C. En el grafico se muestran las operaciones realizadas por los programas implicados.

Proceso P

Paso	Acción
p_0	Solicita (D)
p_1	Bloquea (D)
p_2	Solicita (C)
p_3	Bloquea (C)
p_4	Realiza función
p_5	Desbloquea (D)
p_6	Desbloquea (C)

Proceso Q

Paso	Acción
q_0	Solicita (C)
q_1	Bloquea (C)
q_2	Solicita (D)
q_3	Bloquea (D)
q_4	Realiza función
q_5	Desbloquea (C)
q_6	Desbloquea (D)

_ El interbloqueo se produce si cada proceso mantiene un recurso y solicita el otro. Por ejemplo, ocurrirá un interbloqueo si el sistema de multiprogramación intercala la ejecución de los procesos de la siguiente manera:

- p0 p1 q0 q1 p2 q2

_ Una estrategia para tratar con este interbloqueo es imponer restricciones en el diseño del sistema con respecto al orden en que se pueden solicitar los recursos.

_ Otro ejemplo de interbloqueo con un recurso reutilizable está relacionado con las peticiones de reserva de memoria principal. Suponemos que el espacio disponible para reservar es de 200 Kbytes y que se produce la secuencia siguiente de peticiones:

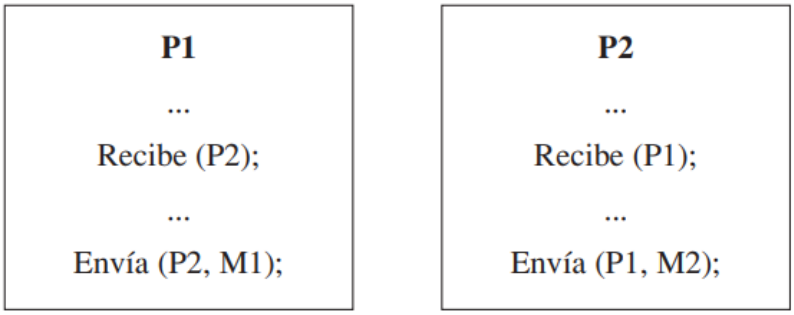
P1
...
Solicita 80 Kbytes;
...
Solicita 60 Kbytes;

P2
...
Solicita 70 Kbytes;
...
Solicita 80 Kbytes;

_ El interbloqueo sucede si ambos procesos progresan hasta su segunda petición. Si no se conoce anticipadamente la cantidad de memoria que va a solicitarse, es difícil tratar con este tipo de interbloqueos mediante restricciones en el diseño del sistema.

Recurso consumible: es aquél que puede ser creado (producirse) y destruido (consumirse) por un proceso. Normalmente, no hay límite en el número de recursos consumibles de un determinado tipo. Un proceso productor desbloqueado puede crear un número ilimitado de estos recursos. Cuando un proceso consumidor adquiere un recurso, el recurso deja de existir. Algunos ejemplos de recursos consumibles son las interrupciones, las señales, los mensajes y la información en buffers de E/S.

_ Como un ejemplo de interbloqueo que involucra recursos consumibles, consideramos el siguiente par de procesos de tal forma que cada proceso intenta recibir un mensaje del otro y, a continuación, le envía un mensaje:



_ Se produce un interbloqueo si la función de recepción (Recibe) es bloqueante (es decir, el proceso receptor se bloquea hasta que se recibe el mensaje). Nuevamente, la causa del interbloqueo es un error de diseño, que pueden ser bastante sutiles y difíciles de detectar. Además, puede darse una rara combinación de eventos que cause el interbloqueo.

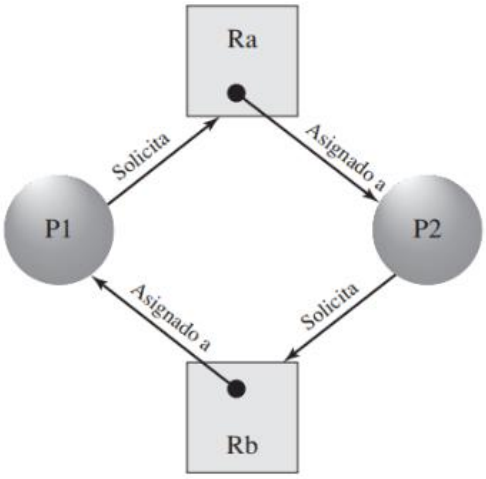
Condiciones para el interbloqueo

_ Deben presentarse tres condiciones de gestión para que sea posible un interbloqueo:

1. Exclusión mutua: solo un proceso puede utilizar un recurso en cada momento. Ningún proceso puede acceder a una unidad de un recurso que se ha asignado a otro proceso.
2. Retención y espera: un proceso puede mantener los recursos asignados mientras espera la asignación de otros recursos.
3. Sin expropiación: no se puede forzar la expropiación de un recurso a un proceso que lo posee.

_ Si se cumplen estas tres condiciones se puede producir un interbloqueo, pero aunque se cumplan puede que no lo haya. Para que realmente se produzca el interbloqueo, se requiere una cuarta condición:

4. Espera circular o circulo vicioso de espera: existe una lista cerrada de procesos, de tal manera que cada proceso posee al menos un recurso necesitado por el siguiente proceso de la lista.



(c) Espera circular

_ Las tres primeras condiciones son necesarias pero no suficientes para que exista un interbloqueo. La cuarta condición es, realmente, una consecuencia potencial de las tres primeras. Es decir, si se cumplen las tres primeras condiciones, se puede producir una secuencia de eventos que conduzca a una espera circular irresoluble. La espera circular irresoluble es de hecho la definición del interbloqueo. La espera circular enumerada como cuarta condición es irresoluble debido a que se cumplen las tres primeras condiciones. Por tanto, las cuatro condiciones de forma conjunta constituyen condiciones necesarias y suficientes para el interbloqueo.

Estrategias para el tratamiento del interbloqueo

Prevención del interbloqueo

_ Se puede prevenir el interbloqueo adoptando una política que elimine una de las condiciones (las 4 condiciones enumeradas previamente). La estrategia de prevención del interbloqueo consiste, en diseñar un sistema de manera que se excluya la posibilidad del interbloqueo. Esto conlleva un uso ineficiente de los recursos y una ejecución ineficiente de los procesos. Se pueden clasificar los métodos de prevención del interbloqueo en dos categorías:

- Método indirecto: interbloqueo es impedir la aparición de una de las tres condiciones necesarias listadas previamente (las tres primeras).
- Método directo: impide que se produzca una espera circular (cuarta condición).

_ A continuación, se examinan las técnicas relacionadas con las cuatro condiciones:

Exclusión mutua: esta condición no puede eliminarse. Si el acceso a un recurso requiere exclusión mutua, el sistema operativo debe proporcionarlo.

Retención y espera: esta condición puede eliminarse estableciendo que un proceso debe solicitar al mismo tiempo todos sus recursos requeridos, bloqueándolo hasta que se le puedan conceder simultáneamente todas las peticiones, pero esta estrategia es insuficiente ya que, en primer lugar, un proceso puede quedarse esperando mucho tiempo hasta que todas sus solicitudes de recursos puedan satisfacerse, cuando podría haber continuado con solamente algunos de los recursos. Y en segundo lugar, los recursos asignados a un proceso pueden permanecer inutilizados durante un periodo de tiempo considerable, durante el cual se impide su uso a otros procesos. Otro problema es que un proceso puede no conocer por anticipado todos los recursos que requerirá.

Sin expropiación: esta condición se puede impedir de varias maneras:

- Si a un proceso que mantiene varios recursos se le deniega una petición posterior, ese proceso deberá liberar sus recursos originales y, si es necesario, los solicitará de nuevo junto con el recurso adicional.
- Alternativamente, si un proceso solicita un recurso que otro proceso mantiene actualmente, el sistema operativo puede expropiar al segundo proceso y obligarle a liberar sus recursos.

Espera circular: esta condición se puede impedir definiendo un orden lineal entre los distintos tipos de recursos. Por ejemplo, si a un proceso le han asignado recursos de tipo R, posteriormente puede pedir sólo aquellos recursos cuyo tipo tenga un orden posterior al de R.

Predicción del interbloqueo

_ Se puede predecir el interbloqueo tomando las apropiadas decisiones dinámicas basadas en el estado actual de asignación de recursos. La predicción del interbloqueo, por otro lado, permite las tres condiciones necesarias pero toma decisiones razonables para asegurarse de que nunca se alcanza el punto del interbloqueo. De esta manera, la predicción permite más concurrencia que la prevención.

- Con la predicción del interbloqueo, se decide dinámicamente si la petición actual de reserva de un recurso, si se concede, podrá potencialmente causar un interbloqueo. La predicción del interbloqueo, por tanto, requiere el conocimiento de las futuras solicitudes de recursos del proceso.

_ Tenemos dos técnicas para predecir el interbloqueo:

- No iniciar un proceso si sus demandas podrían llevar al interbloqueo.
- No conceder una petición adicional de un recurso por parte de un proceso si esta asignación podría provocar un interbloqueo.

Denegación de asignación de recursos o Algoritmo del banquero: consideramos un sistema con un número fijo de procesos y de recursos. En un determinado momento un proceso puede tener cero o más recursos asignados.

- Estado del sistema: refleja la asignación actual de recursos a procesos. Por lo tanto, el estado consiste en los dos vectores:
  - Recursos: cantidad total de cada recurso en el sistema.
  - Disponibles: cantidad total de cada recurso no asignada a ningún proceso.

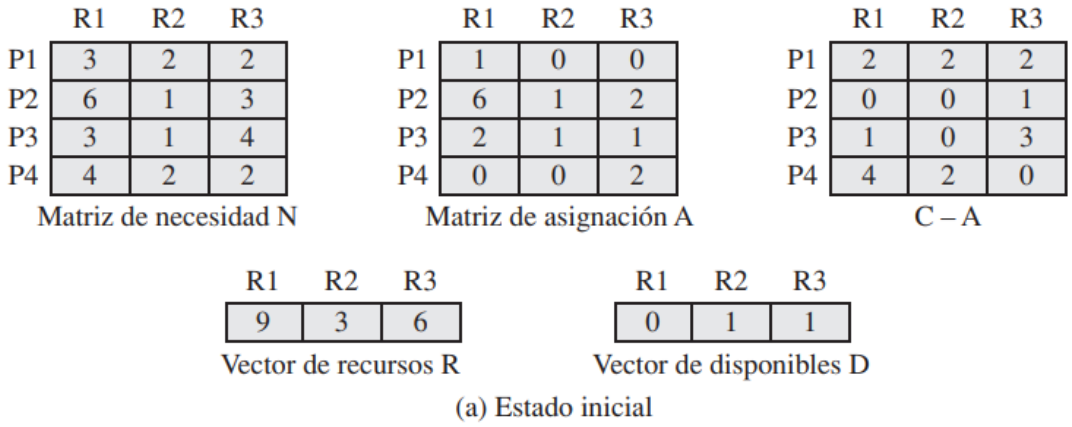
  Y en las dos matrices:

  - Necesidad: necesidades del proceso i con respecto al recurso j.
  - Asignación: asignación actual al proceso i con respecto al recurso j.

_ Tenemos que definir entonces:

- Estado seguro: es aquél en el que hay al menos una secuencia de asignación de recursos a los procesos que no implica un interbloqueo (es decir, todos los procesos pueden ejecutarse al completo).
- Estado inseguro: es un estado que no es seguro.

_ El siguiente grafico muestra el estado de un sistema que consta de cuatro procesos y tres recursos. La cantidad total de recursos R1, R2, y R3 son de 9, 3 y 6 unidades, respectivamente. En el estado actual se han realizado asignaciones a los cuatro procesos, dejando disponibles 1 unidad de R2 y 1 unidad de R3. Nos preguntamos si es un estado seguro, y para saberlo nos preguntamos si alguno de los cuatro procesos puede ejecutarse por completo con los recursos disponibles.



	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Matriz de necesidad N

	R1	R2	R3
P1	1	0	0
P2	6	1	2
P3	2	1	1
P4	0	0	2

Matriz de asignación A

	R1	R2	R3
P1	2	2	2
P2	0	0	1
P3	1	0	3
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Vector de recursos R

R1	R2	R3
0	1	1

Vector de disponibles D

(a) Estado inicial

_ Esto no es posible en el caso de P1, que tiene sólo una unidad de R1 y requiere 2 unidades adicionales, otras 2 de R2, así como 2 unidades de R3. Sin embargo, asignando una unidad de R3 al proceso P2, éste logra tener asignados sus recursos máximos requeridos y puede ejecutarse por completo. Si esto se lleva a cabo, cuando se complete P2, podrá retornar sus recursos al conjunto de recursos disponibles. El estado resultante se muestra a continuación:

	R1	R2	R3
P1	3	2	2
P2	0	0	0
P3	3	1	4
P4	4	2	2

Matriz de necesidad N

	R1	R2	R3
P1	1	0	0
P2	0	0	0
P3	2	1	1
P4	0	0	2

Matriz de asignación A

	R1	R2	R3
P1	2	2	2
P2	0	0	0
P3	1	0	3
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Vector de recursos R

R1	R2	R3
6	2	3

Vector de disponibles D

(b) P2 ejecuta hasta completarse

_ En ese momento, se repetiría la pregunta de si cualquiera de los procesos restantes puede completarse. En este caso, todos los procesos restantes podrían completarse. Supongamos, que se elige P1, asignándole los recursos requeridos, se completa P1, y devuelve todos sus recursos al conjunto de disponibles. El estado resultante se muestra a continuación:

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	3	1	4
P4	4	2	2

Matriz de necesidad N

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	2	1	1
P4	0	0	2

Matriz de asignación A

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	1	0	3
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Vector de recursos R

R1	R2	R3
7	2	3

Vector de disponibles D

(c) P1 ejecuta hasta completarse

_ A continuación, se puede completar P3, alcanzándose el estado del siguiente gráfico.

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	4	2	2

Matriz de necesidad N

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	0	0	2

Matriz de asignación A

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Vector de recursos R

R1	R2	R3
9	3	4

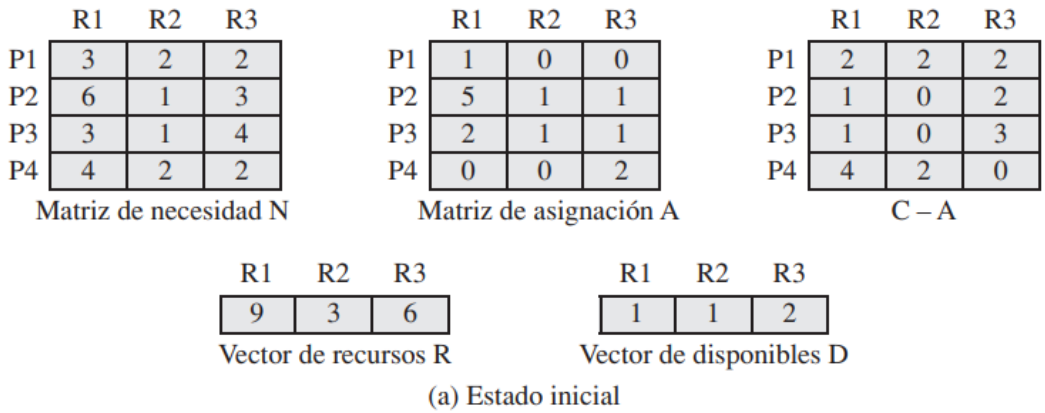
Vector de disponibles D

(d) P3 ejecuta hasta completarse

_ Finalmente, se puede completar P4. En ese instante, todos los procesos se han ejecutado por completo. Por lo tanto, el estado definido en el grafico (a) es seguro.

_ Esta estrategia de predicción del interbloqueo, asegura que el sistema de procesos y recursos está siempre en un estado seguro. Cuando un proceso solicite un conjunto de recursos, suponiendo de que se concede la petición, se actualiza el estado del sistema en consecuencia, y determina si el resultado es un estado seguro. En caso afirmativo, se concede la petición. En caso contrario, se bloquea el proceso hasta que sea seguro conceder la petición.

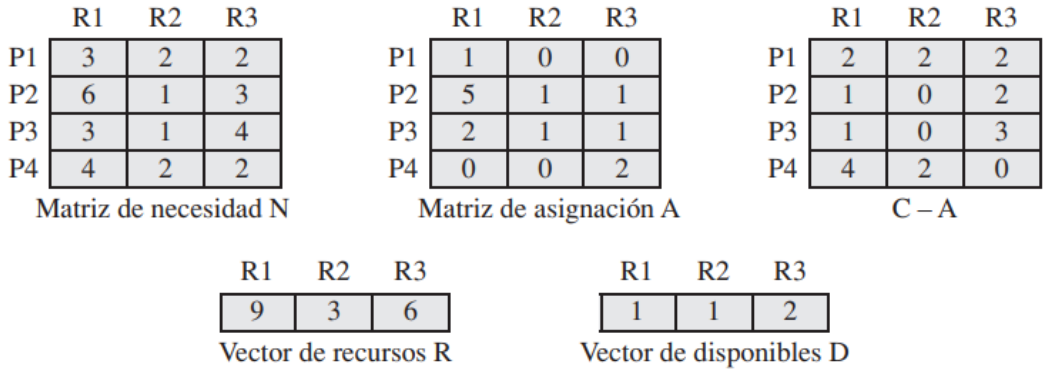
_ Ahora, en el estado definido en el siguiente gráfico, suponemos que P2 solicita una unidad adicional de R1 y otra de R3:



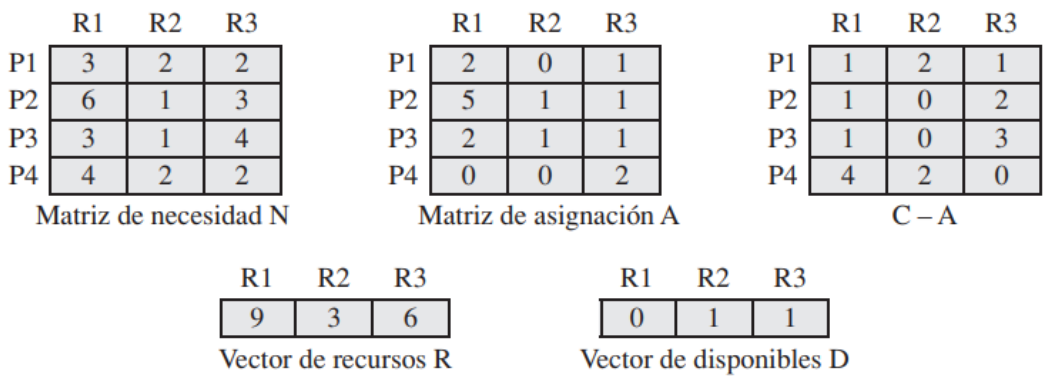
(a) Estado inicial

_ Si se asume que se concede la petición, el estado resultante es el del grafico anterior (a), que ya se ha comprobado previamente que es un estado seguro. Por lo tanto, es seguro conceder la petición.

_ Retornando al estado del siguiente gráfico y suponiendo que P1 solicita una unidad de R1 y otra de R3; si se asume que se concede la petición, el estado resultante es el representado en la grafico (b):



(a) Estado inicial



(b) P1 solicita una unidad de R1 y de R3

_ Nos preguntamos si esto es un estado seguro, y la respuesta es que no, debido a que cada proceso necesitará al menos una unidad adicional de R1, y no hay ninguna disponible. Por lo tanto, basándose en la predicción del interbloqueo, la solicitud de P1 se denegaría y P1 se debería bloquear. Es importante resaltar que el grafico (b) no es un estado de interbloqueo. Sólo indica la posibilidad del interbloqueo.

_ Por lo tanto, la estrategia de predicción de interbloqueo no predice el interbloqueo con certeza; sólo anticipa la posibilidad del interbloqueo y asegura que no haya tal posibilidad. La predicción del interbloqueo tiene la ventaja de que no es necesario expropiar a los procesos ni retroceder su ejecución, como ocurre con la detección del interbloqueo, y es menos restrictivo que la prevención del interbloqueo. Sin embargo, tiene varias restricciones de uso:

- Deben establecerse por anticipado los requisitos máximos de recursos de cada proceso.
- Los procesos involucrados deben ser independientes, es decir, el orden en el que se ejecutan no debe estar restringido por ningún requisito de sincronización.
- Debe haber un número fijo de recursos que asignar.
- Ningún proceso puede terminar mientras mantenga recursos.

_ A modo de resumen, la idea detrás del algoritmo del banquero, es evitar que un estado inseguro (donde los procesos podrían quedar bloqueados indefinidamente) ocurra en un sistema con asignación de recursos limitados a varios procesos. Este algoritmo se utiliza en la gestión de recursos en sistemas operativos para garantizar que no se concedan asignaciones que puedan llevar a un estado de inanición.

- El algoritmo del banquero opera sobre la premisa de que el sistema conoce el número de recursos disponibles y la cantidad máxima de recursos que cada proceso podría necesitar en su ejecución. Utiliza esta información para decidir si se puede conceder una solicitud de recursos de un proceso sin dejar al sistema en un estado inseguro.
- Antes de asignar recursos a un proceso, el algoritmo verifica si la solicitud lleva al sistema a un estado inseguro (donde algún proceso podría quedar bloqueado indefinidamente). Si la asignación no causa un estado inseguro, se otorgan los recursos; de lo contrario, la solicitud se deniega o se pone en espera hasta que el sistema vuelva a un estado seguro.

Detección del interbloqueo

_ Se puede intentar detectar la presencia del interbloqueo (cuando se cumplen las 4 condiciones) y realizar las acciones pertinentes para recuperarse del mismo. Las estrategias de prevención de interbloqueo resuelven el problema del interbloqueo limitando el acceso a los recursos e imponiendo restricciones a los procesos. En el extremo contrario, la estrategia de detección del interbloqueo no limita el acceso a los recursos ni restringe las acciones de los procesos. Con la detección del interbloqueo, los recursos pedidos se conceden a los procesos siempre que sea posible.

Recuperación: una vez que se ha detectado el interbloqueo, se necesita alguna estrategia para recuperarlo. Las siguientes estrategias son posibles:

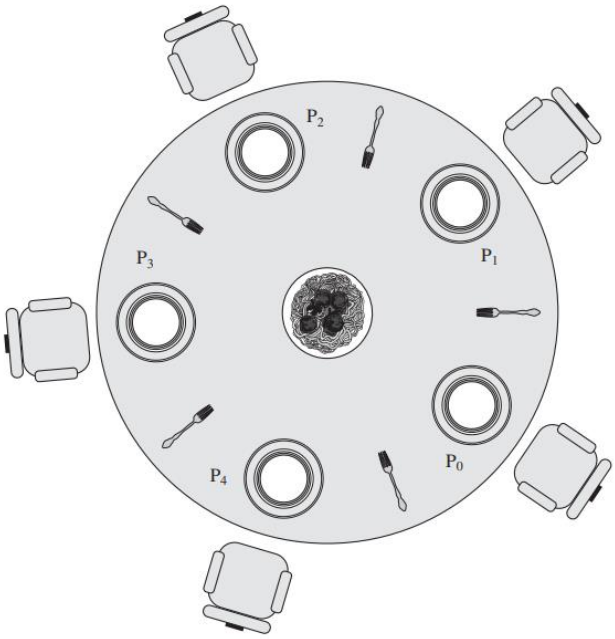
1. Abortar todos los procesos involucrados en el interbloqueo. Esta es la solución adoptada en los sistemas operativos.
2. Retroceder cada proceso en interbloqueo a algún punto de control (checkpoint) previamente definido, y volver a ejecutar todos los procesos. El riesgo de esta técnica es que puede repetirse el interbloqueo original.
3. Abortar sucesivamente los procesos en el interbloqueo hasta que éste deje de existir. Después de cada aborto, se debe invocar de nuevo el algoritmo de detección para comprobar si todavía existe el interbloqueo.
4. Expropiar sucesivamente los recursos hasta que el interbloqueo deje de existir. Un proceso al que se le ha expropiado un recurso debe retroceder a un punto anterior a la adquisición de ese recurso.

_ Para los puntos 3 y 4, el criterio de selección podría ser uno de los siguientes. Se elige el proceso con:

- La menor cantidad de tiempo de procesador consumida hasta ahora.
- La menor cantidad de salida producida hasta ahora.
- El mayor tiempo restante estimado.
- El menor número total de recursos asignados hasta ahora.
- La menor prioridad.

El problema de los filósofos comensales

_ Cinco filósofos viven en una casa, donde hay una mesa preparada para ellos. Básicamente, la vida de cada filósofo consiste en pensar y comer, y después de años de haber estado pensando, todos los filósofos están de acuerdo en que la única comida que contribuye a su fuerza mental son los espaguetis. Debido a su falta de habilidad manual, cada filósofo necesita dos tenedores para comer los espaguetis. La disposición para la comida es simple, hay una mesa redonda en la que está colocado un gran cuenco para servir espaguetis, cinco platos, uno para cada filósofo, y cinco tenedores. Un filósofo que quiere comer se dirige a su sitio asignado en la mesa y, utilizando los dos tenedores situados a cada lado del plato, toma y come algunos espaguetis.



_ El problema consiste en diseñar un ritual (algoritmo) que permita a los filósofos comer. El algoritmo debe satisfacer la exclusión mutua (no puede haber dos filósofos que puedan utilizar el mismo tenedor a la vez) evitando el interbloqueo y la inanición (en este caso, el término tiene un sentido literal, además de algorítmico). Este problema muestra los problemas básicos del interbloqueo y la inanición.

Solución utilizando semáforos: cada filósofo toma primero el tenedor de la izquierda y después el de la derecha. Una vez que el filósofo ha terminado de comer, vuelve a colocar los dos tenedores en la mesa. Esta solución, desgraciadamente, conduce al interbloqueo:

- Si todos los filósofos están hambrientos al mismo tiempo, todos ellos se sentarán, agarran el tenedor de la izquierda y tenderán la mano para tomar el otro tenedor, que no estará allí.
- Para superar el riesgo de interbloqueo, se podrían comprar cinco tenedores adicionales (una solución más higiénica) o enseñar a los filósofos a comer espaguetis con un solo tenedor. Como alternativa, se podría incorporar un asistente que sólo permitiera que haya cuatro filósofos al mismo tiempo en el comedor. Con un máximo de cuatro filósofos sentados, al menos un filósofo tendrá acceso a los dos tenedores.

Solución utilizando un monitor: se define un vector de cinco variables de condición, una variable de condición por cada tenedor. Estas variables de condición se utilizan para permitir que un filósofo espere hasta que esté disponible un tenedor. Además, hay un vector de tipo booleano que registra la disponibilidad de cada tenedor (verdadero significa que el tenedor está disponible). El monitor consta de dos procedimientos:

- Procedimiento obtiene_tenedores: lo utiliza un filósofo para obtener sus tenedores, el de su izquierda y su derecha. Si ambos tenedores están disponibles, el proceso que corresponde con el filósofo se encola en la variable de condición correspondiente. Esto permite que otro proceso filósofo entre en el monitor.
- Procedimiento liberar_tenedores: sirve para hacer que queden disponibles los dos tenedores.

_ En ambos soluciones, un filósofo toma primero el tenedor de la izquierda y después el de la derecha. Pero a diferencia de la solución del semáforo, esta solución del monitor no sufre interbloqueos, porque sólo puede haber un proceso en cada momento en el monitor. Por ejemplo, se garantiza que el primer proceso filósofo que entre en el monitor pueda agarrar el tenedor de la derecha después de que tome el de la izquierda pero antes de que el siguiente filósofo a la derecha tenga una oportunidad de agarrar el tenedor de su izquierda, que es el que está a la derecha de este filósofo.

Planificación uniprocesador

Introducción

_ En un sistema multiprogramado, hay múltiples procesos de modo concurrente en la memoria principal. Los procesos pueden estar usando un procesador o pueden estar esperando el suceso de algún evento, tal como la finalización de una operación de E/S. El procesador o los procesadores se mantienen ocupados ejecutando un proceso mientras el resto espera. La clave de la multiprogramación es la planificación.

Planificación del procesador

Concepto

_ El objetivo de la planificación de procesos es asignar procesos a ser ejecutados por el procesador o procesadores a lo largo del tiempo, de forma que se cumplan los objetivos del sistema tales como el tiempo de respuesta, el rendimiento y la eficiencia del procesador.

- En muchos sistemas, esta actividad de planificación se divide en tres funciones independientes, planificación a largo, medio, y corto-plazo. Los nombres sugieren las escalas de tiempo relativas con que se ejecutan estas funciones.
- La planificación afecta al rendimiento del sistema porque determina qué proceso esperará y qué proceso progresará.

_ La planificación uniprocesador se trata de:

- Maximizar la utilización del procesador.
- Minimizar los tiempos de espera.
- Proporcionar un equilibrio entre la equidad para todos los procesos y la eficiencia global del sistema.

Tipos de planificación del procesador

Planificación a largo plazo: esta determina cuándo se admiten nuevos procesos al sistema. La decisión, de añadir un proceso al conjunto de procesos a ser ejecutados, se realiza cuando se crea un nuevo proceso. Entonces consiste en decidir si se añade un nuevo proceso al conjunto de los que están activos actualmente. Esta planificación se ejecuta relativamente con poca frecuencia, y toma la decisión de grano grueso de admitir o no un nuevo proceso y qué proceso admitir.

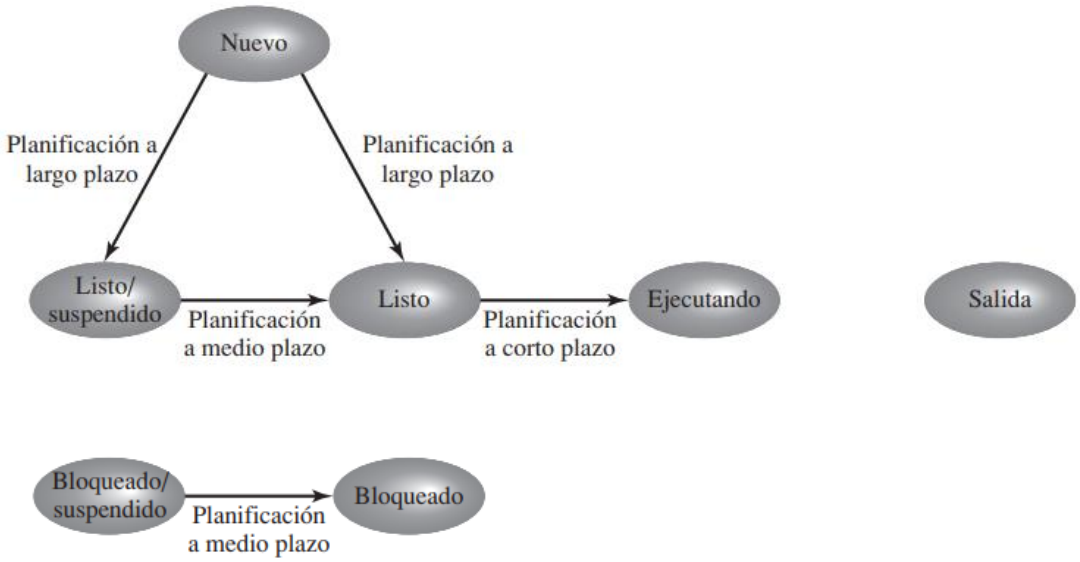
Planificación a medio plazo: esta determina cuándo un programa se trae parcial o totalmente a memoria principal para que pueda ser ejecutado. La decisión de añadir un proceso al número de procesos que están parcialmente o totalmente en la memoria principal es parte de la función de intercambio (swapping function). Entonces, hay que decidir si se añade un proceso a los que están al menos

parcialmente en memoria y que, por tanto, están disponibles para su ejecución. Esta planificación se ejecuta más frecuentemente para tomar decisiones de intercambio.

Planificación a corto plazo: esta determina qué proceso listo será ejecutado por el procesador. La decisión por la que un proceso disponible será ejecutado por el procesador, es decir, conlleva decidir cuál de los procesos listos para ejecutar será ejecutado. Se conoce también como activador y se ejecuta mucho más frecuentemente. El planificador a corto plazo se invoca siempre que ocurre un evento que puede conllevar el bloqueo del proceso actual y que puede proporcionar la oportunidad de expulsar al proceso actualmente en ejecución en favor de otro. Algunos ejemplos de estos eventos son:

- Interrupciones de reloj.
- Interrupciones de E/S.
- Llamadas al sistema.
- Señales (por ejemplo, semáforos).

_ Este unidad se centra en los aspectos relativos a la planificación a corto plazo.



Planificación de la E/S: La decisión por la que un proceso que está pendiente de una petición de E/S será atendido por un dispositivo de E/S disponible.

Criterios de la planificación a corto plazo

_ Para el diseño de un planificador a corto plazo se deben tener en cuenta una serie de criterios. Algunos de estos criterios están relacionados con el comportamiento del sistema tal y como lo percibe el usuario (orientados al usuario), mientras que otros miran la efectividad total del sistema para satisfacer las necesidades de todos los usuarios (orientados al sistema). Algunos criterios se relacionan específicamente con medidas cuantitativas de rendimiento, mientras que otros son de naturaleza más cualitativa. Desde el punto de vista del usuario el tiempo de respuesta es, generalmente, la característica más importante de un sistema, mientras que desde el

punto de vista del sistema son importantes el rendimiento o la utilización del procesador.

Orientados al usuario, relacionados con las prestaciones:

- Tiempo de estancia (turnaround time): tiempo transcurrido desde que se lanza un proceso hasta que finaliza. Incluye el tiempo de ejecución sumado con el tiempo de espera por los recursos, incluyendo el procesador.
- Tiempo de respuesta (response time): tiempo que trascurre desde que se lanza una petición hasta que se comienza a recibir la respuesta. Es una medida mejor que el tiempo de estancia. La planificación debe intentar lograr bajos tiempos de respuesta y maximizar el número de usuarios interactivos con tiempos de respuesta aceptables.
- Fecha tope (deadlines): cuando se puede especificar la fecha tope de un proceso, el planificador debe obligar otros objetivos al de maximizar el porcentaje de fechas tope conseguidas.

Orientados al usuario, otros:

- Previsibilidad: un trabajo dado debería ejecutarse en el mismo tiempo y con el mismo coste a pesar de la carga del sistema. Una gran variación en el tiempo de respuesta o en el tiempo de estancia es malo desde el punto de vista de los usuarios. Puede significar sobrecarga del sistema o la necesidad de eliminar las inestabilidades.

Orientados al sistema, relacionados con las prestaciones:

- Rendimiento: medida de cuánto trabajo está siendo realizado, y la política de planificación debería intentar maximizar el número de procesos completados por unidad de tiempo.
- Utilización del procesador: porcentaje de tiempo que el procesador está ocupado. Para un sistema compartido, es un criterio significativo, pero para un sistema de un solo usuario o sistemas de tiempo real, es menos importante.

Orientados al sistema, otros:

- Equidad: en todos los sistemas, todos los procesos deben ser tratados de la misma manera, y ninguno debe sufrir inanición.
- Imposición de prioridades: cuando se asignan prioridades a los procesos, la política del planificador debería favorecer a los procesos con prioridades más altas.
- Equilibrado de recursos: la política del planificador debería mantener ocupados los recursos del sistema, los procesos que utilicen poco los recursos que en un determinado momento están sobreutilizados, deberían ser favorecidos.

Algoritmos de planificación a corto plazo

_ Se han desarrollado varios algoritmos para tomar decisiones de planificación a corto plazo entre todos los procesos listos para ejecutar. Los sistemas operativos aplican estos algoritmos para asignar tiempo de CPU a los procesos, asegurándose de optimizar el rendimiento y evitar la inanición o el bloqueo. Estos son:

- Primero en llegar, primero en servirse (FCFS - First-Come-First-Served): este selecciona el proceso que más tiempo ha estado esperando servicio.
- Turno rotatorio (Round Robin): utiliza “rodajas de tiempo” para limitar los procesos en ejecución a una pequeño ciclo de tiempo de ejecución, y rota entre todos los procesos listos.
- Primero el proceso más corto (Shortest Process Next): se selecciona el proceso con el menor tiempo de procesamiento esperado, y no expulsa a los procesos.
- Menor tiempo restante (Shortest Remaining Time): se selecciona el proceso con el menor tiempo de procesamiento restante esperado. Un proceso puede ser expulsado cuando otro proceso pasa a listo.
- Primero el de mayor tasa de respuesta (Highest Response Ratio Next): basa la decisión de planificación en una estimación del tiempo de estancia normalizado.
- Retroalimentación (Feedback): establece un conjunto de colas de planificación y sitúa los procesos en las colas basándose en su historia de ejecución y otros criterios.

Planificación multiprocesador y de tiempo real

Planificación multiprocesador

_ Cuando un sistema computador contiene más de un procesador, el diseño de la función de planificación plantea varias cuestiones nuevas. Podemos clasificar los sistemas multiprocesador como sigue:

- Débilmente acoplado o multiprocesador distribuido, o cluster: es una colección de sistemas relativamente autónomos, y cada uno tiene su propia memoria principal y canales de E/S.
- Procesadores de funcionalidad especializada: hay un procesador de propósito general maestro y procesadores especializados que son controlados por el procesador maestro y que le proporcionan servicios, por ejemplo, un procesador de E/S.
- Procesamiento fuertemente acoplado: conjunto de procesadores que comparten la memoria principal y están bajo el control integrado de un único sistema operativo. Este es el que nos interesa en esta unidad.

Aspectos de diseño de planificación

Distribución de tareas: la asignación de procesos entre varios procesadores para mejorar la eficiencia y el rendimiento del sistema.

Algoritmos de asignación: estrategias para distribuir cargas de trabajo, como la planificación asimétrica (tareas diferentes en distintos procesadores) o simétrica (mismas tareas en todos los procesadores).

Sincronización y concurrencia: coordinación entre procesadores para evitar problemas de acceso simultáneo a recursos compartidos y garantizar consistencia.

Balance de carga: distribución equilibrada de tareas entre procesadores para evitar sobrecargas y cuellos de botella que reduzcan la eficiencia global.

Planificación de tiempo real

Proceso en tiempo real

_ Un proceso o tarea de tiempo real es aquel que ejecuta en conexión con algunos procesos o funciones o conjunto de eventos externos al sistema computador y que debe cumplir uno o más plazos para interactuar efectiva y correctamente con el entorno externo. Un sistema operativo de tiempo real es aquel capaz de manejar procesos de tiempo real. En este contexto, los criterios tradicionales de los algoritmos de planificación no tienen cabida, en cambio, el factor clave es cumplir los plazos de tiempo. Estas tareas se pueden clasificar en:

Tarea de tiempo real duro: es aquella que debe cumplir su plazo límite y tienen cierto grado de urgencia, ya que de otro modo, se producirá un daño inaceptable o error fatal en el sistema.

Tarea de tiempo real suave: tiene asociado un plazo límite deseable pero no obligatorio; sigue teniendo sentido planificar y completar la tarea incluso cuando su plazo límite ya haya vencido.

_ Otra característica de las tareas de tiempo son:

Tarea aperiódica: tiene un plazo en el cual debe finalizar o comenzar, o puede tener una restricción tanto de su instante de comienzo como de finalización.

Tarea periódica: el requisito puede ser enunciado como «una vez por periodo T» o «exactamente T unidades a parte». Se le asigna un periodo T.

Características de los sistemas operativos de tiempo real

_ Los sistemas operativos de tiempo real pueden ser caracterizados por tener requisitos únicos en cinco áreas generales:

Determinismo: en los sistemas de tiempo real, las operaciones deben ocurrir en tiempos predecibles y consistentes. Esto significa que las acciones deben completarse dentro de plazos específicos y con mínima variabilidad en el tiempo de respuesta.

Reactividad: capacidad de responder rápidamente a eventos. Los sistemas de tiempo real deben manejar eventos o solicitudes de manera inmediata, sin demoras significativas.

Control del usuario: estos sistemas permiten a los usuarios controlar y ajustar prioridades, plazos y recursos para garantizar la ejecución correcta y oportuna de tareas críticas.

Fiabilidad: esencial en entornos críticos donde se necesitan operaciones confiables y consistentes. Los sistemas operativos de tiempo real deben minimizar los fallos y tener mecanismos de recuperación robustos.

Operación de fallo suave: en lugar de detenerse completamente en caso de fallo, estos sistemas pueden degradarse o ajustarse para mantener la funcionalidad esencial. Esto significa que pueden seguir operando con limitaciones en lugar de un colapso total.

Aspectos de diseño de planificación

_ La planificación de tiempo real se centra en garantizar que las tareas se completen dentro de plazos establecidos:

Garantía de plazos: asegurar que las tareas críticas se completen dentro de límites temporales específicos y estrictos.

Algoritmos de planificación específicos:

- Enfoques estáticos dirigidos por tabla: estos enfoques realizan un análisis antes de la ejecución (de forma estática) para planificar cuándo deben comenzar cada tarea. Es como hacer un cronograma por adelantado.
- Enfoques estáticos expulsivos dirigidos por prioridad: también realizan un análisis previo, pero en lugar de planificar, asignan prioridades a las tareas. Estas prioridades se usan durante la ejecución para decidir cuál tarea se ejecuta primero.
- Enfoques dinámicos basados en un plan: deciden la factibilidad (si se pueden cumplir los plazos) durante la ejecución (de forma dinámica), en lugar de antes. Si una tarea se puede ejecutar dentro de su límite de tiempo, se planifica cuándo comenzará.
- Enfoques dinámicos de mejor esfuerzo: no se realiza ningún análisis de factibilidad antes de la ejecución. En lugar de eso, el sistema intenta cumplir los plazos y, si una tarea no lo cumple, se aborta su ejecución.

Gestión de interrupciones: manejo de eventos y solicitudes que puedan interferir con las tareas críticas, evitando retrasos en su ejecución.

Priorización y determinismo: enfoque en tareas prioritarias con plazos rígidos sobre tareas menos críticas para cumplir con los límites temporales establecidos.