SO 22-23: Bibliografía Tema 2

Stallings

Tarea 2.1 - Diseño e implementación de procesos e hilos: páginas 108-143 y 158-172

Capítulo 3: Descripción y control de procesos

El diseño de un sistema operativo debe reflejar ciertos requisitos generales. Todos los sistemas operativos multiprogramados, desde los sistemas operativos monousuario como Windows 98 hasta sistemas mainframes como IBM z/OS, que son capaces de dar soporte a miles de usuarios, se construyen en torno al concepto de proceso.

La mayoría de los requisitos que un sistema operativo debe cumplir se pueden expresar con referencia a los procesos:

- 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 interbloqueos.
- 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.

3.1. ¿Qué es un proceso?

Conceptos previos

Antes de definir el término proceso, es útil recapitular algunos de los conceptos ya presentados en los Capítulos 1 y 2:

1. Una plataforma de computación consiste en una colección de recursos hardware, como procesador, memoria, módulos de E/S, relojes, unidades de disco y similares.
2. Las aplicaciones para computadores se desarrollan para realizar determinadas tareas. Suelen aceptar entradas del mundo exterior, realizar algún procesamiento y generar salidas.
3. No es eficiente que las aplicaciones estén escritas directamente para una plataforma hardware específica.
4. El sistema operativo se desarrolló para proporcionar una interfaz apropiada para las aplicaciones, rica en funcionalidades, segura y consistente. El sistema operativo es una capa de software entre las aplicaciones y el hardware del computador que da soporte a aplicaciones y utilidades.
5. Se puede considerar que el sistema operativo proporciona una representación uniforme y abstracta de los recursos, que las aplicaciones pueden solicitar y acceder. Los recursos incluyen la memoria principal, las interfaces de red, los sistemas de ficheros, etc. Una vez que el sistema operativo ha creado estas abstracciones de los recursos para que las aplicaciones las usen, debe también controlar su uso. Por ejemplo, un sistema operativo podría permitir compartición y protección de recursos.

Ahora sí, podemos hablar de cómo el SO puede, de forma ordenada, gestionar la ejecución de aplicaciones de forma que:

- Los recursos estén disponibles para múltiples aplicaciones.
- El procesador físico se conmute entre múltiples aplicaciones, de forma que todas lleguen a procesarse.
- El procesador y los dispositivos de E/S se puedan usar de forma eficiente.

El enfoque adoptado por todos los sistemas operativos modernos recae en un modelo bajo el cual la ejecución de una aplicación se corresponde con la existencia de uno o más procesos.

Procesos y bloque de control de procesos

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.

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.

La información de la lista anterior se almacena en una estructura de datos, que se suele llamar bloque de control de proceso (process control block) (Figura 3.1), que el sistema operativo crea y gestiona.

El punto más significativo en relación al bloque de control de proceso, o BCP, es que contiene suficiente información de forma que es posible interrumpir el proceso cuando está corriendo y posteriormente restaurar su estado de ejecución como si no hubiera habido interrupción alguna. 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 (descritos a continuación). 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.

Para un computador monoprocesador, en un instante determinado, como máximo un único proceso puede estar corriendo y dicho proceso estará en el estado en ejecución.

3.2. Estados de los procesos

Se puede caracterizar el comportamiento de un determinado proceso listando la secuencia de instrucciones que se ejecutan para dicho proceso. A esta lista se la denomina traza del proceso. Se puede caracterizar el comportamiento de un procesador mostrando cómo las trazas de varios procesos se entrelazan.

Un modelo de proceso de dos estados

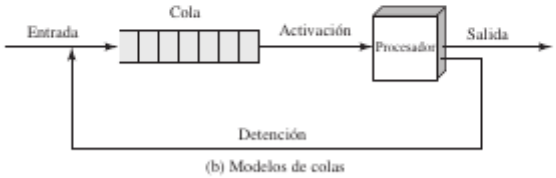
La responsabilidad principal del sistema operativo es controlar la ejecución de los procesos; esto incluye determinar el patrón de entrelazado para la ejecución y asignar recursos a los procesos. El primer paso en el diseño de un sistema operativo para el control de procesos es describir el comportamiento 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 o No Ejecutando, como se muestra en la Figura 3.5a. 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 (dispatcher), seleccionará otro proceso a ejecutar. El proceso saliente pasará del estado Ejecutando a No Ejecutando y pasará a Ejecutando un nuevo proceso.

De este modelo simple, ya se puede apreciar algo del diseño de los elementos del sistema operativo. **Cada proceso debe representarse de tal manera que el sistema operativo pueda seguirle la pista. Es decir, debe haber información correspondiente a cada proceso, incluyendo el estado actual y su localización en memoria; esto es el bloque de control de programa. Los procesos que no están ejecutando deben estar en una especie de cola, esperando su turno de ejecución**. La Figura 3.5b sugiere esta estructura. 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. Podemos describir el comportamiento del activador en términos de este diagrama de colas. 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.

(a) Diagrama de transiciones de estados



(b) Modelos de colas

Figura 3.5. Modelo de proceso de dos estados.

Creación y terminación de procesos

Antes de intentar refinar nuestro sencillo modelo de dos estados, resultará útil hablar de la creación y terminación de los procesos; por último, y de forma independiente del modelo de comportamiento de procesos que se use, la vida de un proceso está acotada entre su creación y su terminación.

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 (como se describió en la Sección 3.3) y reserva el espacio de direcciones en memoria principal para el proceso. Estas acciones constituyen la creación de un nuevo proceso.

Existen cuatro 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.
    - Cuando un proceso lanza otro, al primero se le denomina proceso padre, y al proceso creado se le denomina proceso hijo. Habitualmente, la relación entre procesos necesita comunicación y cooperación entre ellos. Alcanzar esta cooperación es una tarea complicada para un programador.

Terminación de procesos.

- 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 tie- ne 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 alguna razón, el operador o el sistema operativo ha finalizado el por el sistema operativo 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.

Modelo de proceso de cinco estados

Si todos los procesos estuviesen siempre preparados para ejecutar, la gestión de colas proporcionada en la Figura 3.5b 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, hasta con el sencillo ejemplo que vimos antes, esta implementación es inadecuada: 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 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. Esto se muestra la Figura 3.6. 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 proce- sos 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.
    - El SO ha realizado todas las tareas necesarias para crear el proceso pero el proceso en sí, aún no se ha puesto en ejecución. Por ejemplo, un sistema operativo puede limitar el número de procesos que puede haber en el sistema por razones de rendimiento o limitaciones de memoria principal. Mientras un proceso está en el estado Nuevo, la información relativa al proceso que se necesite por parte del sistema operativo se mantiene en tablas de control de memoria principal. Sin embargo, el proceso en sí mismo no se encuentra en memoria principal.
- 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.
    - De forma similar al estado nuevo, un proceso sale del sistema en dos fases. Primero, el proceso termina cuando alcanza su punto de finalización natural, cuando es abortado debido a un error no recuperable, o cuando otro proceso con autoridad apropiada causa que el proceso se aborte. La terminación mueve el proceso al estado Saliente. En este punto, el proceso no es elegible de nuevo para su ejecución. Las tablas y otra información asociada con el trabajo se encuentran temporalmente preservadas por el sistema operativo, el cual proporciona tiempo para que programas auxiliares o de soporte extraigan la información necesaria.

Figura 3.6. Modelo de proceso de cinco estados.

La Figura 3.6 indica que tipos de eventos llevan a cada transición de estado para cada proceso; las posibles transiciones son las siguientes:

- Null Æ Nuevo. Se crea un nuevo proceso para ejecutar un programa. Este evento ocurre por cualquiera de las relaciones indicadas en la tabla 3.1.
- Nuevo Æ 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 Æ Ejecutando. Cuando llega el momento de seleccionar un nuevo proceso para ejecutar, el sistema operativo selecciona uno los procesos que se encuentre en el estado Listo. Esta es una tarea la lleva acabo el planificador. El planificador se estudiará más adelante en la Parte Cuatro.
- Ejecutando Æ 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. Véase tabla 3.2.
- Ejecutando Æ 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; prácticamente todos los sistemas operativos multiprogramados imponen este tipo de restricción de tiempo. Existen otras posibles causas alternativas para esta transición, que no están incluidas en todos los sistemas operativos.

  Es de particular importancia el caso en el cual 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, move- rá 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. Adicionalmente, un proceso puede voluntariamente dejar de utilizar el procesador. Un ejemplo son los procesos que realiza alguna función de auditoría o de mantenimiento de forma periódica.
- Ejecutando Æ 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. Puede solicitar un recurso, como por ejemplo un fichero o una sección compartida de memoria virtual, que no está inmediatamente disponible. Cuando un proceso quiere iniciar una acción, tal como una operación de E/S, que debe completarse antes de que el proceso continúe. Cuando un proceso se comunica con otro, un proceso puede bloquearse mientras está esperando a que otro proceso le proporcione datos o esperando un mensaje de ese otro proceso.
- Bloqueado Æ Listo. Un proceso en estado Bloqueado se mueve al estado Listo cuando sucede el evento por el cual estaba esperando.
- Listo Æ Saliente. Por claridad, 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 Æ Saliente. Se aplican los comentarios indicados en el caso anterior.

La figura 3.8a sugiere la forma de aplicar un esquema de dos colas: la colas de Listos y la cola de Bloqueados. Cada proceso admitido por el sistema, se coloca en la cola de Listos. Cuando llega el momento de que el sistema operativo seleccione otro proceso a ejecutar, selecciona uno de la cola de Listos. En ausencia de un esquema de prioridad, esta cola puede ser una lista de tipo FIFO (first-in-first-out).

Cuando el proceso en ejecución termina de utilizar el procesador, o bien finaliza o bien se coloca en la cola de Listos o de Bloqueados, dependiendo de las circunstancias. Por último, cuando sucede un evento, cualquier proceso en la cola de Bloqueados que únicamente esté esperando a dicho evento, se mueve a la cola de Listos. Esta última transición significa que, cuando sucede un evento, el sistema operativo debe recorrer la cola entera de Bloqueados, buscando aquellos procesos que estén esperando por dicho evento.

En los sistemas operativos con muchos procesos, esto puede significar cientos o incluso miles de procesos en esta lista, por lo que sería mucho más eficiente tener una cola por cada evento. De esta forma, cuando sucede un evento, la lista entera de procesos de la cola correspondiente se movería al estado de Listo (Figura 3. 8b).

Un refinamiento final sería: si la activación de procesos está dictada por un esquema de prioridades, sería conveniente tener varias colas de procesos listos, una por cada nivel de prioridad. El sistema operativo podría determinar cual es el proceso listo de mayor prioridad simplemente seleccionando éstas en orden.

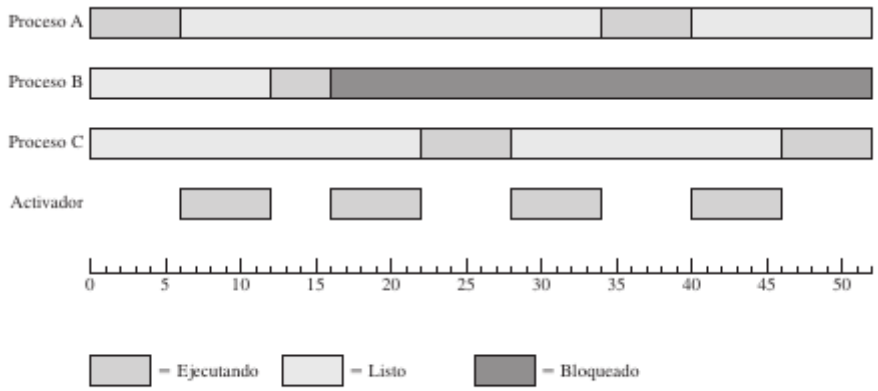


Figura 3.7. Estado de los procesos de la traza de la Figura 3.4.

Procesos suspendidos

La 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. Se han construido algunos sistemas operativos utilizando únicamente estos tres estados. 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.

Recuérdese que la razón de toda esta compleja maquinaria es que las operaciones de E/S son mucho más lentas que los procesos de cómputo y, por tanto, el procesador en un sistema monoprogramado estaría ocioso la mayor parte del tiempo. Es verdad que, en este caso, la memoria almacena múltiples procesos y el procesador puede asignarse a otro proceso si el que lo usa se queda bloqueado. 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.

Por tanto, incluso en un sistema multiprogramado, el procesador puede estar ocioso la mayor parte del tiempo. ¿Qué se puede hacer? La memoria principal puede expandirse para acomodar más procesos. Hay dos fallos en esta solución. Primero, existe 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.

Otra solución es el 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. Pero debido a que la E/S en disco es habitualmente más rápida que la E/S sobre otros sistemas (por ejemplo, comparado con cinta o impresora), el swapping habitualmente mejora el rendimiento del sistema.

Con el uso de swapping tal y como se ha escrito, debe añadirse un nuevo estado a nuestro modelo de comportamiento de procesos (Figura 3.9a): el estado Suspendido. 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 (transferencia a disco de un proceso), 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. Parece que sería preferible traer un proceso que anteriormente estuviese suspendido, para proporcionar dicho servicio en lugar de incrementar la carga total del sistema. Pero, esta línea de razonamiento presenta una dificultad: todos los procesos que fueron suspendidos se encontraban previamente en el estado de Bloqueado en el momento de su suspensión. Claramente no sería bueno traer un proceso bloqueado de nuevo a memoria porque podría no encontrarse todavía listo para la ejecución.

Cuando el evento sucede, el proceso no está Bloqueado y está potencialmente disponible para su ejecución. De esta forma, necesitamos replantear este aspecto del diseño. Hay dos conceptos independientes aquí: 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.

Veamos ahora, en la Figura 3.9b, el modelo de transición de estados que ha sido diseñado. (Las líneas punteadas de la figura indican transiciones posibles pero no necesarias.) Las nuevas transiciones más importantes son las siguientes:

- Bloqueado Æ 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

proce- sos listos que desea ejecutar requieren más memoria principal para mantener un rendimiento adecuado.

- Bloqueado/Suspendido Æ Listo/Suspendido. Un proceso en el estado Bloqueado/Suspendido se mueve al estado Listo/Suspendido cuando sucede un evento al que estaba esperando. Nótese que esto requiere que la información de estado concerniente a un proceso suspendido sea accesible para el sistema operativo.
- Listo/Suspendido Æ 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. En este caso, el sistema operativo puede haberse diseñado para determinar que es más importante traer un proceso de mayor prioridad que para minimizar el efecto del swapping.
- Listo Æ 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 podrían ser las siguientes:

- Nuevo Æ 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 (NuevoÆListo/Suspendido). Por otro lado, deseamos hacer hincapié en que la filosofía de creación de procesos diferida, haciéndolo cuanto más tarde, hace posible reducir la sobrecarga del sistema operativo y le permite realizar las tareas de creación de procesos cuando el sistema está lleno de procesos bloqueados.
- Bloqueado/Suspendido Æ Bloqueado. La incursión de esta transición puede parecer propia de un diseño más bien pobre. Después de todo, si hay un proceso que no está listo para ejecutar y que no está en memoria principal, ¿qué sentido tiene traerlo? Pero considérese el siguiente escenario: 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 Æ 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 Æ Saliente. Habitualmente, un proceso termina cuando 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.

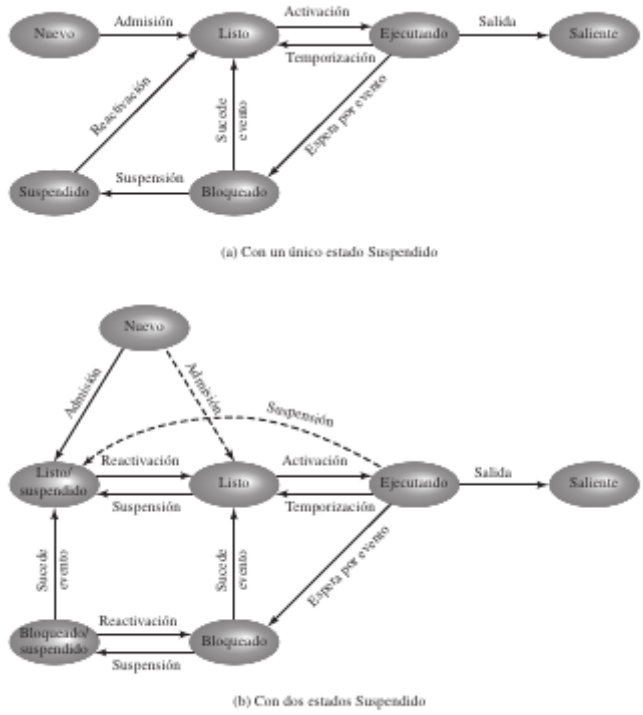


Figura 3.9. Diagrama de transición de estados de procesos con estado suspendidos.

3.3. Descripción de procesos

El sistema operativo controla los eventos dentro del computador, planifica y activa los procesos para su ejecución por el procesador, reserva recursos para los mismos y responde a las solicitudes de servicios básicos de los procesos de usuario. Fundamentalmente, se piensa en el sistema operativo como en la entidad que gestiona el uso de recursos del sistema por parte de los procesos. Este concepto se ilustra en la Figura 3.10.

En un entorno multiprogramado, hay numerosos procesos (P1,... Pn) creados y residentes en memoria virtual. Cada proceso, durante el transcurso de su ejecución, necesita acceder a ciertos recursos del sistema, incluido el procesador, los dispositivos de E/S y la memoria principal. En la figura, el proceso P1 está ejecutando; al menos parte del proceso está en memoria principal, y controla dos dispositivos de E/S. El proceso P2 está también totalmente en memoria principal pero está bloqueado esperando a disponer de un dispositivo de E/S que está asignado a P1. El proceso Pn se encuentra transferido a disco (swapping) y está por tanto suspendido.

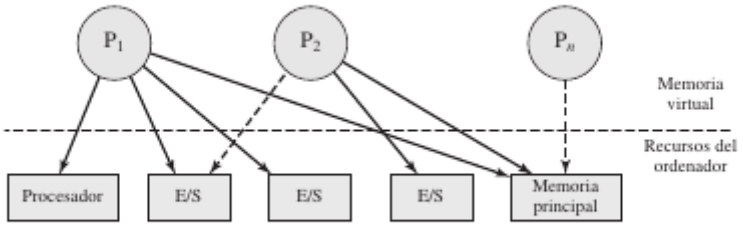


Figura 3.10. Procesos y recursos (reserva de recursos en una instantánea del sistema).

Exploraremos los detalles de la gestión de estos recursos por parte del sistema operativo en los próximos capítulos. Aquí nos interesa una cuestión más fundamental: ¿qué información necesita el sistema operativo para controlar los procesos y gestionar los recursos de estos?

Estructuras de control del SO

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 el siguiente: el sistema operativo construye y mantiene tablas de información sobre cada entidad que gestiona. Se indica una idea general del alcance de esta tarea en la Figura 3.11, que muestra cuatro diferentes tipos de tablas mantenidas por el sistema operativo: memoria, E/S, ficheros y procesos. A pesar de que los detalles difieren de un sistema operativo a otro, fundamentalmente, todos los sistemas operativos mantienen información de estas cuatro categorías.

- Las 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.
- El sistema operativo debe utilizar las 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.
- El sistema operativo también puede mantener las tablas de ficheros. Estas tablas 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.
- En último lugar, el sistema operativo debe mantener tablas de procesos para gestionar los procesos. La parte restante de esta sección se encuentra dedicada a examinar los requisitos que tienen las tablas de procesos.

Antes de proceder con esta explicación, vamos a realizar dos puntualizaciones importantes. Primero, a pesar de que la Figura 3.11 muestra cuatro tipos diferentes de tablas debe quedar claro que estas tablas se encuentran entrelazadas y referenciadas entre sí de alguna manera. Memoria, E/S, y ficheros se gestionan por parte de los procesos, de forma que debe haber algunas referencias en estos recursos, directa o indirectamente, desde las tablas de procesos. Los ficheros indicados en las tablas de ficheros son accesibles mediante dispositivos E/S y estarán, en algún caso, residentes en memoria virtual o principal. Las tablas son también accesibles para el sistema operativo, y además están controladas por la gestión de memoria.

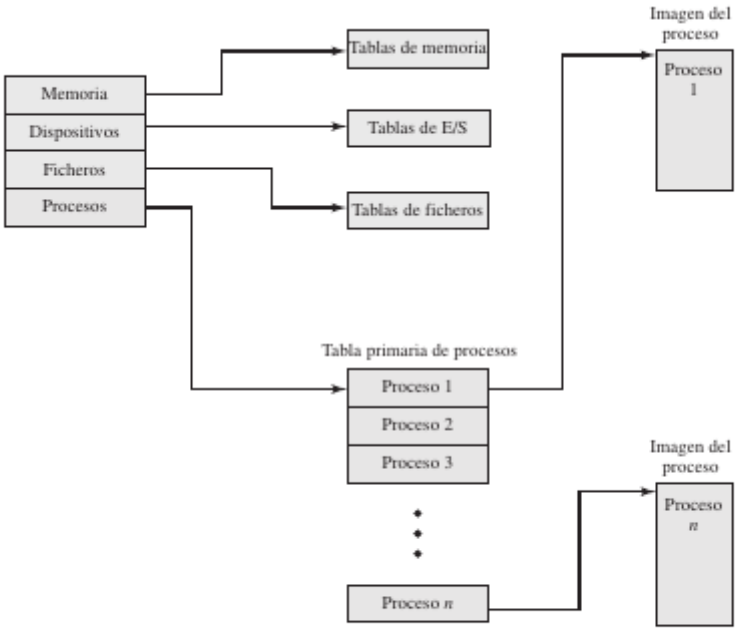


Figura 3.11. Estructura general de las tablas de control del sistema operativo.

Segundo, ¿cómo puede el sistema operativo crear las tablas iniciales? Ciertamente, 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, por ejemplo. Esto es una cuestión de configuración. Esto es, cuando el sistema operativo se inicializa, debe tener acceso a algunos datos de configuración que definen el entorno básico y estos datos se deben crear fuera del sistema operativo, con asistencia humana o por medio de algún software de autoconfiguración.

Estructuras de control del proceso

Se considerará qué información debe conocer el sistema operativo si quiere manejar y controlar los procesos. Primero, debe conocer dónde están localizados los procesos, y segundo, debe conocer los atributos de los procesos que quiere gestionar (por ejemplo, identificador de proceso y estado del mismo).

Localización de los procesos. Antes de que podamos tratar la cuestión de dónde se encuentran los procesos o cuáles son sus atributos, debemos tratar una cuestión más fundamental: ¿qué es la representación física de un proceso? Como mínimo, 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 con- trol del proceso (BCP). Nos podemos referir al conjunto de programa, datos, pila, y atributos, como la **imagen del proceso** (Tabla 3.4).

La posición de la imagen del proceso dependerá del esquema de gestión de memoria que se utilice. En el caso más simple, la imagen del proceso se mantiene como un bloque de memoria contiguo, o continuo. Este bloque se mantiene en memoria secundaria, habitualmente disco. Para que el sistema operativo pueda gestionar el proceso, al menos una pequeña porción de su imagen se debe mantener en memoria principal. Para ejecutar el proceso, la imagen del proceso

completa se debe cargar en memoria principal o al menos en memoria virtual. Asimismo, el sistema operativo necesita conocer la posición en disco de cada proceso y, para cada proceso que se encuentre en memoria principal, su posición en dicha memoria.

Tabla 3.4. Elementos típicos de la imagen de un proceso.

Datos del usuario	La parte modificable del espacio de usuario. Puede incluir datos de programa, área de pila de usuario, y programas que puedan ser modificados.
Programa de usuario	El programa a ejecutar
Pila de sistema	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.
Bloque de control de proceso	Datos necesarios para que el sistema operativo pueda controlar los procesos (*véase* tabla 3.5).

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. Al final de este capítulo y en el próximo se muestran varios ejemplos. Por ahora, exploraremos simplemente el tipo de información que el sistema operativo puede utilizar, sin considerar cualquier detalle de cómo esta información está organizada.

La Tabla 3.5 muestra la lista de los tipos de categorías de información que el sistema operativo requiere para cada proceso. Resulta sorprendente la cantidad de información necesaria para esta gestión. Cuando se vaya apreciando con mayor exactitud las responsabilidades del sistema operativo, esta lista parecerá más razonable.

Tabla 3.5. Elementos típicos de un bloque de control del proceso (BCP).

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.

Podemos agrupar la información del bloque de control del proceso en tres categorías generales

- Identificación del proceso: Con respecto al identificador de proceso, en prácticamente todos los sistemas operativos, a cada proceso se le asocia un identificador numérico único, que puede ser simplemente un índice en la tabla de procesos principal; de otra forma, debe existir una traducción que permita al sistema operativo localizar las tablas apropiadas basándose en dicho identificador de proceso.
- Información de estado del procesador: La información de estado de proceso indica los contenidos de los registros del procesador. Cuando un proceso está ejecutando, esta información está, por supuesto, en los registros. Cuando un proceso se interrumpe, toda la información de los registros debe salvaguardarse de forma que se pueda restaurar cuando el proceso continúe con su ejecución.
    - Se debe notar que, el diseño de todos los procesadores incluye un registro o conjuntos de registros, habitualmente conocidos como palabras de estado de programa (program status word, PSW) que contiene información de estado. La PSW típicamente contiene códigos de condición así como otra información de estado. Como ejemplo de una palabra de estado de un procesador se tiene lo que los procesadores Pentium definen como registros EFLAFS (véase Tabla 3.6). Esta estructura es utilizada por los sistemas operativos (incluyendo UNIX y Windows) que se ejecuten sobre procesadores de tipo Pentium.

Tabla 3.6. Bits del registro EFLAGS de un Pentium.

Bits de control	
AC (*Alignment check*)	Fija si una palabra (o una doble-palabra) se encuentra direccionada en la frontera entre palabras (o dobles-palabras).
ID (*Identification flag*)	Si este bit puede ser puesto a 1 y a 0, este procesador soporta la instrucción CPUID. Esta instrucción proporciona información sobre el vendedor, familia, y modelo.
RF (*Resume flag*)	Permite al programador desactivar las extensiones de depuración de forma que la instrucción pueda volverse a ejecutar después de una excepción de depuración sin causar inmediatamente después otra excepción de depuración.
IOPL (*I/O privilege level*)	Cuando está activado, hace que el procesador genere una excepción para todos los accesos a dispositivo de E/S durante una operación en modo protegido.
DF (*Direction flag*)	Determina cuando una instrucción de procesamiento de cadenas de caracteres incrementa o decrementa los semi-registros de 16 bits SE y DI (para operaciones de 16 bits) o los registros de 32 bits ESE y EDI (para operaciones de 32 bits).
IF (*Interruption enable flag*)	Cuando está puesto a 1, el procesador reconocerá interrupciones externas.
TF (*Trap flag*)	Cuando está puesto a 1, causa una interrupción después de la ejecución de cada instrucción. Su uso está dirigido a la depuración de código.
Bits de modos de operación	
NT (*Nested task flag*)	Indica que la tarea actual está anidada dentro de otra tarea en modo de operación protegido.
VM (*Virtual 8086 mode*)	Permite al programador activar o desactivar el modo virtual 8086, que determina si el procesador funciona como si fuese una máquina 8086.
VIP (*Virtual interrupt pending*)	Usado en el modo virtual 8086 para indicar que una o más interrupciones están a la espera de servicio.
VIF (*Virtual interruption flag*)	Usado en modo virtual 8086 en lugar de IF.
Códigos de condición	
AF (*Auxiliar carry flag*)	Representa el acarreo o resto entre medios bytes de una operación aritmética o lógica de 8 bits usando el registro AL.
CF (*Carry flag*)	Indica el acarreo o el resto por medio del bit situado más a la izquierda después de una operación aritmética. También se modificaron por algunas operaciones de desplazamiento o rotación.
OF (*Overflow flag*)	Indica un desbordamiento (*overflow*) aritmético después de una suma o resta.
PF (*Parity flag*)	Paridad del resultado de una operación aritmética o lógica. 1 indica paridad par; 0 indica paridad impar.
SF (*Sign flag*)	Indica el signo del resultado de una operación aritmética o lógica.
ZF (*Zero flag*)	Indica que el resultado de una operación aritmética o lógica es 0.

- Información de control del proceso: La tercera categoría principal de información del bloque de control de proceso se denomina, a falta de otro nombre mejor, información de control de proceso. Esta información adicional la necesita el sistema operativo para controlar y coordinar varios procesos activos. La última parte de la Tabla 3.5 indica el ámbito de esta información. Según examinemos los detalles de funcionamiento del sistema operativo en los sucesivos capítulos, la necesidad de algunos de los elementos de esta lista parecerá más clara.

El papel del bloque de control de proceso. El bloque de control de proceso es la más importante de las estructuras de datos del sistema operativo. Cada bloque de control de proceso contiene toda la información sobre un proceso que necesita el sistema operativo. Los bloques se leen y/o se modifican por la práctica totalidad de los módulos del sistema operativo, incluyendo aquellos relacionados con la planificación, la reserva de recursos, el procesamiento entre regiones, y el análisis y monitorización del rendimiento. Se puede decir que el conjunto de los bloques de control de proceso definen el estado del sistema operativo.

Esto muestra un aspecto importante en el diseño. Un gran número de rutinas dentro del sistema operativo necesitarán acceder a información de los bloques de control de proceso. Proporcionar acceso directo a estas tablas no es difícil. Cada proceso lleva asociado un único identificador, que puede utilizarse para indexar dentro de una tabla de punteros a bloques de control de proceso.

La dificultad no reside en el acceso sino en la protección. Dos posibles problemas se pueden presentar son:

- Un fallo en una simple rutina, como un manejador de interrupción, puede dañar los bloques de control de proceso, que puede destruir la capacidad del sistema para manejar los procesos afectados.
- Un cambio en la estructura o en la semántica de los bloques de control de procesos puede afectar a un gran número de módulos del sistema operativo.

Estos problemas se pueden ser tratar obligando a que todas rutinas del sistema operativo pasen a través de una rutina de manejador, cuyo único trabajo es proteger a los bloques de control de proceso, y que es la única que realiza el arbitraje en las operaciones de lectura y escritura de dichos bloques. Los factores a equilibrar en el uso de está rutina son, por un lado los aspectos de rendimiento, y por el otro, el grado en el cual el resto del software del sistema puede verificarse para determinar su corrección.

3.4. Control de procesos

Modos de ejecución

Antes de continuar con nuestra explicación sobre cómo el sistema operativo gestiona los procesos, necesitamos distinguir entre los modos de ejecución del procesador, normalmente asociados con el sistema operativo y los asociados con los programas de usuario. 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.

El modo menos privilegiado a menudo se denomina modo usuario, porque los programas de usuario típicamente se ejecutan en este modo. El modo más privilegiado se denomina modo kernel. La Tabla 3.7 lista las funciones que normalmente se encuentran dentro del núcleo del sistema operativo.

Tabla 3.7. 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.

Aparecen dos cuestiones: ¿cómo conoce el procesador en que modo está ejecutando y cómo este modo puede modificarse? En lo referente la primera cuestión, existe típicamente un bit en la palabra de estado de programa (PSW) que indica el modo de ejecución. Este bit se cambia como respuesta a determinados eventos. Habitualmente, cuando un usuario realiza una llamada a un servicio del sistema operativo o cuando una interrupción dispara la ejecución de una rutina del sistema operativo, este modo de ejecución se cambia a modo núcleo y; tras la finalización del servicio, el modo se fija de nuevo a modo usuario.

La mayoría de los sistemas operativos, como Linux, utilizar el Nivel 0 para el núcleo y uno de los otros niveles como nivel de usuario. Cuando llega una interrupción, el procesador limpia la mayor parte de los bits en psr, incluyendo el campo cpl. Esto automáticamente pone en 0 el cpl. Al final de la rutina de manejo de interrupción, la última instrucción que se ejecuta es irt (interrupt return). Esta instrucción hace que el procesador restaure el psr del programa interrumpido, que restaura a su vez el nivel de privilegios de dicho programa. Una secuencia similar ocurre cuando la aplicación solicita una llamada al sistema.

Creación de procesos

En la Sección 3.1 hemos comentado los eventos que llevará creación de uno proceso. Una vez vistas las estructuras de datos asociadas a un proceso, ahora posiciones está en posición de describir brevemente los pasos que llevan a la verdadera creación de un proceso. Una vez que el sistema operativo decide, por cualquier motivo (Tabla 3.1), crear un proceso procederá de la siguiente manera:

1. Asignar un identificador de proceso único al proceso. En este instante, se añade una nueva entrada a la tabla primaria de procesos, que contiene una entrada por proceso. 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. Estos valores se pueden asignar por defecto basándonos en el tipo de proceso, o pueden fijarse en base a la solicitud de creación del trabajo remitido por el usuario. Si un proceso es creado por otro proceso, el proceso padre puede pasar los parámetros requeridos por el sistema operativo como parte de la solicitud de la creación de proceso. Si existe una parte del espacio direcciones compartido por este nuevo proceso, se fijan los enlaces apropiados. Por último, se debe reservar el espacio para el bloque de control de proceso (BCP).

2. 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. En la información de estado de proceso del BCP, habitualmente se inicializa con la mayoría de entradas a 0, excepto el contador de programa (fijado en el punto entrada del programa) y los punteros de pila de sistema (fijados para definir los límites de la pila del proceso).

   La parte de información de control de procesos se inicializa en base a los valores por omisión, considerando también los atributos que han sido solicitados para este proceso. Por ejemplo, el estado del proceso se puede inicializar normalmente a Listo o Listo/Suspendido. La prioridad se puede fijar, por defecto, a la prioridad más baja, a menos que una solicitud

explicita la eleve a una prioridad mayor. Inicialmente, el proceso no debe poseer ningún recurso (dispositivos de E/S, ficheros) a menos que exista una indicación explícita de ello o que haya sido heredados del padre.

3. Establecer los enlaces apropiados. Por ejemplo, si el sistema operativo mantiene cada cola del planificador como una lista enlazada, el nuevo proceso debe situarse en la cola de Listos o en la cola de Listos/Suspendidos.
4. Creación o expansión de otras estructuras de datos. Por ejemplo, el sistema operativo puede mantener un registro de auditoría por cada proceso que se puede utilizar posteriormente a efectos de facturación y/o de análisis de rendimiento del sistema.

Cambio de proceso

A primera vista, la operación de cambio de proceso puede parecer sencilla. En algunos casos, un proceso en ejecución se interrumpe para que el sistema operativo asigne a otro proceso el estado de Ejecutando y de esta forma establecer el turno entre los procesos. Sin embargo, es necesario tener en cuenta algunas consideraciones de diseño.

Primero, ¿qué evento dispara los cambios de proceso? Otra consideración que se debe tener en cuenta es la distinción entre cambio de proceso (process switching) y cambio de modo (mode switching). Por último, ¿qué debe hacer el sistema operativo con las diferentes estructuras que gestiona para proporcionar un cambio de proceso?

Cuándo se realiza el 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 Tabla 3.8 indica los posibles momentos en los que el sistema operativo puede tomar dicho control.

Primero, consideremos las interrupciones del sistema. Realmente, podemos distinguir, como hacen muchos sistemas, dos tipos diferentes de interrupciones de sistema, unas simplemente denominadas interrupciones, y las otras denominadas traps. **Las interrupciones se producen por causa de algún tipo de evento que es externo e independiente al proceso actualmente en ejecución, por ejemplo la finalización de la operación de E/S. Las traps están asociadas a una condición de error o excepción generada dentro del proceso que está ejecutando, como un intento de acceso no permitido a un fichero.**

Dentro de una interrupción ordinaria, el control se transfiere inicialmente al manejador de interrupción, que realiza determinadas tareas internas y que posteriormente salta a una rutina del sistema operativo, encargada de cada uno de los tipos de interrupciones en particular. Algunos ejemplos 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). Esto es, 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.

Con un trap, 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. Se puede intentar un procedimiento de recuperación o simplemente la notificación al usuario, pudiendo implicar tanto un cambio de proceso como la continuación de la ejecución del proceso actual.

Por último, 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. **Esta llamada implica un salto a una rutina que es parte del código del sistema operativo**. La realización de una llamada al sistema puede implicar en algunos casos que el proceso que la realiza pase al estado de Bloqueado.

Tabla 3.8. Mecanismos para la interrupción de la ejecución de un proceso.

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.

Cambio de modo. En el Capítulo 1, discutimos la inclusión de una fase de interrupción como parte del ciclo de una instrucción. Recordando esto, 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.

El procesador, acto seguido, pasa a la fase de búsqueda de instrucción y busca la primera instrucción del programa de manejo de interrupción, que dará servicio a la misma. En este punto, habitualmente, el contexto del proceso que se ha interrumpido se salvaguarda en el bloque de control de proceso del programa interrumpido.

Una pregunta que se puede plantear es, **¿qué constituye el contexto que se debe salvaguardar?** La respuesta es que se trata de toda la información que se puede ver alterada por la ejecución de la rutina de interrupción y que se necesitará para la continuación del proceso que ha sido interrumpido. De esta forma, se debe guardar la parte del bloque de control del proceso que hace referencia a la información de estado del procesador. Esto incluye el contador de programa, otros registros del procesador, y la información de la pila.

¿Se necesita hacer algo más? Eso depende de qué ocurra luego. El manejador de interrupción es habitualmente un pequeño programa que realiza unas pocas tareas básicas relativas a la interrupción. Por ejemplo, borra el flag o indicador que señala la presencia de interrupciones. Puede enviar una confirmación a la entidad que lanzó dicha interrupción, como por ejemplo el módulo de E/S. Y puede realizar algunas tareas internas variadas relativas a los efectos del evento que causó la interrupción. Por ejemplo, si la interrupción se refiere a un evento de E/S, el manejador de interrupción comprobará la existencia o no de una condición de error. Si ha

ocurrido un error, el manejador mandará una señal al proceso que solicitó dicha operación de E/S. Si la interrupción proviene del reloj, el manejador la va a pasar el control al activador, el cual decidirá pasar a otro proceso debido a que la rodaja de tiempo asignada a ese proceso ha expirado.

¿Qué pasa con el resto de información del bloque de control de proceso? Si a esta interrupción le sigue un cambio de proceso a otro proceso, se necesita hacer algunas cosas más. Sin embargo, en muchos sistemas operativos, la existencia de una interrupción no implica necesariamente un cambio de proceso. Es posible, por tanto, que después de la ejecución de la rutina de interrupción, la ejecución se reanude con el mismo proceso. En esos casos sólo se necesita salvaguardar la información del estado del procesador cuando se produce la interrupción y restablecerlo cuando se reanude la ejecución del proceso. Habitualmente, las operaciones de salvaguarda y recuperación se realizan por hardware.

Cambio del estado del proceso. Esta claro, por tanto, que el cambio de modo 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. Salvar el estado del procesador, incluyendo el contador de programa y otros registros.
2. Actualizar el bloque de control del proceso que está actualmente en el estado Ejecutando. Esto incluye cambiar el estado del proceso a uno de los otros estados (Listo, Bloqueado, Listo/Suspendido, o Saliente). También se tienen que actualizar otros campos importantes, inclu- yendo la razón por la cual el proceso ha dejado el estado de Ejecutando y demás información de auditoría.
3. Mover el bloque de control de proceso a la cola apropiada (Listo, Bloqueado en el evento i, Listo/Suspendido).
4. Selección de un nuevo proceso a ejecutar; esta cuestión se analiza con más detalle en la Parte Cuatro.
5. Actualizar el bloque de control del proceso elegido. Esto incluye pasarlo al estado Ejecutando.
6. Actualizar las estructuras de datos de gestión de memoria. Esto se puede necesitar, dependiendo de cómo se haga la traducción de direcciones; estos aspectos se cubrirán en la Parte Tres.
7. Restaurar el estado del procesador al que tenía en el momento en el que el proceso seleccionado salió del estado Ejecutando por última vez, leyendo los valores anteriores de contador de programa y registros.

Por tanto, el cambio de proceso, que implica un cambio en el estado, requiere un mayor esfuerzo que un cambio de modo.

Ejecución del sistema operativo

En el Capítulo 2, señalamos dos aspectos intrínsecos del sistema operativo:

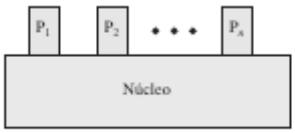
- El sistema operativo funciona de la misma forma que cualquier software, en el sentido que un sistema operativo es un conjunto de programas ejecutados por un procesador.
- El sistema operativo, con frecuencia, cede el control y depende del procesador para recuperar dicho control.

Si el sistema operativo es simplemente una colección de programas y si son ejecutados por el procesador, tal y como cualquier otro programa, ¿es el sistema operativo un proceso del sistema? y si es así ¿cómo se controla? Estas interesantes cuestiones han inspirado unas cuentas direcciones diferentes de diseño. La Figura 3.15 ilustra el rango de diferentes visiones que se encuentran en varios sistemas operativos contemporáneos.

Núcleo sin procesos. Una visión tradicional, que es común a muchos sistemas operativos antiguos, es la ejecución del sistema operativo fuera de todo proceso (Figura 3.15a). 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.

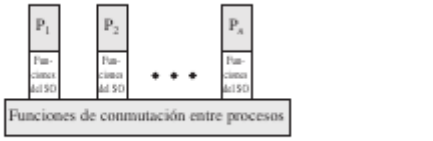
El sistema operativo puede realizar todas las funciones que necesite y restaurar el contexto del proceso interrumpido, que hace que se retome la ejecución del proceso de usuario afectado. De forma alternativa, el sistema operativo puede realizar la salvaguarda del contexto y la activación de otro proceso diferente. Si esto ocurre o no depende de la causa de la interrupción y de las circunstancias puntuales en el momento.

En cualquier caso, el punto clave en este caso es que el concepto de proceso se aplica aquí únicamente a los programas de usuario. El código del sistema operativo se ejecuta como una entidad independiente que requiere un modo privilegiado de ejecución.



(a) Núcleo independiente

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. Esto se muestra en la Figura 3.15b. En un punto determinado, el sistema operativo maneja n imágenes de procesos. Cada imagen incluye no sólo las regiones mostradas en la Figura 3.13, sino que además incluye áreas de programa, datos y pila para los programas del núcleo.



(b) Funciones del SO se ejecutan dentro del proceso de usuario

La Figura 3.16 sugiere la estructura de imagen de proceso típica para esta estrategia. Se usa una pila de núcleo separada para manejar llamadas/retornos cuando el proceso está en modo núcleo. El código de sistema operativo y sus datos están en el espacio de direcciones compartidas y se comparten entre todos los procesos.

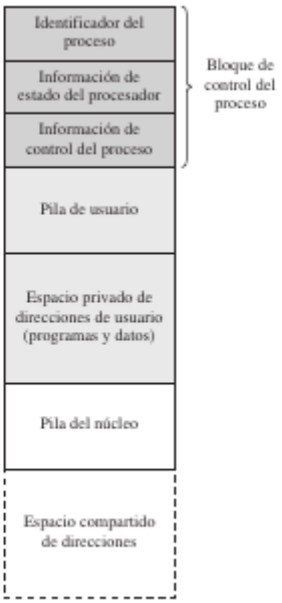


Figura 3.16. Imagen de proceso: el sistema operativo ejecuta dentro del espacio de usuario.

Cuando ocurre una interrupción, trap o llamada al sistema, el procesador se pone en modo núcleo y el control se pasa al sistema operativo. Para este fin, el contexto se salva y se cambia de modo a una rutina del sistema operativo. Sin embargo, la ejecución continúa dentro del proceso de usuario actual. **De esta forma, no se realiza un cambio de proceso, sino un cambio de modo dentro del mismo proceso**. Si el sistema operativo, después de haber realizado su trabajo, determina que el proceso actual debe continuar con su ejecución, entonces el cambio de modo continúa con el programa interrumpido, dentro del mismo proceso. Ésta es una de las principales ventajas de esta alternativa: **se ha interrumpido un programa de usuario para utilizar alguna rutina del sistema operativo, y luego continúa, y todo esto se ha hecho sin incurrir en un doble cambio de proceso.** Sin embargo, si se determina que se debe realizar un cambio de proceso en lugar de continuar con el proceso anterior, entonces el control pasa a la rutina de cambio de proceso. Esta rutina puede o no ejecutarse dentro del proceso actual, depende del sistema. No obstante, significará que, en algún momento, el proceso actual pasará a un estado de no ejecución

De alguna forma, esta visión de un sistema operativo es muy reseñable. De una forma más sencilla, en determinados instantes de tiempo, un proceso salva su estado, elige otro proceso a ejecutar entre aquellos que están listos, y delega el control a dicho proceso. La razón por la cual este esquema no se convierte en una situación arbitraria y caótica es que durante los instantes críticos **el código que se está ejecutando es código de compartido de sistema operativo, no código de usuario. Debido al concepto de modo usuario y modo núcleo, el usuario no puede interferir con las rutinas del sistema operativo, incluso cuando se están ejecutando dentro del entorno del proceso del usuario.** Esto nos recuerda que **existe una clara distinción entre el concepto de proceso y programa y que la relación entre los dos no es uno a uno**. Dentro de un proceso, pueden ejecutarse tanto un programas de usuario como programas del sistema operativo. Los programas de sistema operativo que se ejecutan en varios procesos de usuario son idénticos.

Sistemas operativos basados en procesos. Otra alternativa, mostrada en la Figura 3.15c, es implementar un sistema operativo como una colección de procesos de sistema. Como en las otras opciones, el software que es parte del núcleo se ejecuta en modo núcleo. En este caso, las principales funciones del núcleo se organizan como procesos independientes. De nuevo, debe haber una pequeña cantidad de código para intercambio de procesos que se ejecuta fuera de todos los procesos.

Esta visión tiene diferentes ventajas. Impone una disciplina de diseño de programas que refuerza el uso de sistemas operativos modulares con mínimas y claras interfaces entre los módulos. Adicionalmente, otras funciones del sistema operativo que no sean críticas están convenientemente separadas como otros procesos.

(c) Funciones del SO se ejecutan como procesos independiente

Tarea 2.1 - Diseño e implementación de procesos e hilos: páginas 158-172

Capítulo 4: Hilos, SMP y micronúcleos

Este capítulo analiza algunos conceptos avanzados relativos a la gestión de procesos que se pueden encontrar en los sistemas operativos modernos.

En primer lugar, se muestra cómo el concepto de proceso es más complejo y sutil de lo que se ha visto hasta este momento y, de hecho, contiene dos conceptos diferentes y potencialmente independientes: uno relativo a la propiedad de recursos y otro relativo a la ejecución. En muchos sistemas operativos esta distinción ha llevado al desarrollo de estructuras conocidas como hilos (threads). Después de analizar los hilos se pasa a ver el multiprocesamiento simétrico (Symmetric Multiprocessing, SMP). Con SMP el sistema operativo debe ser capaz de planificar simultáneamente diferentes procesos en múltiples procesadores.

Por último, se introduce el concepto de micronúcleo, una forma útil de estructurar el sistema operativo para dar soporte al manejo de procesos y sus restantes tareas.

4.1. Procesos e hilos

Hasta este momento se ha presentado el concepto de un proceso como poseedor de dos características:

- Propiedad de recursos. Un proceso incluye un espacio de direcciones virtuales para el manejo de la imagen del proceso; como ya se explicó en el Capítulo 3 la imagen de un proceso es la colección de programa, datos, pila y atributos definidos en el bloque de control del proceso. De vez en cuando a un proceso se le puede asignar control o propiedad de recursos tales como la memoria principal, canales E/S, dispositivos E/S y archivos. El sistema operativo realiza la función de protección para evitar interferencias no deseadas entre procesos en relación con los recursos.
- 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. Sin embargo, debe quedar muy claro que **estas dos características son independientes y podrían ser tratadas como tales por el sistema operativo.** Así se hace en diversos sistemas operativos, sobre todo en los desarrollados recientemente. Para distinguir estas dos características, la unidad que se activa se suele denominar hilo (thread), o proceso ligero, mientras que la unidad de propiedad de recursos se suele denominar proceso o tarea.

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. Las dos configuraciones que se muestran en la parte izquierda de la Figura 4.1 son estrategia monohilo.

Un ejemplo de sistema operativo que soporta un único proceso de usuario y un único hilo es el MS-DOS. Otros sistemas operativos, como algunas variedades de UNIX, soportan múltiples procesos de usuario, pero sólo un hilo por proceso. La parte derecha de la Figura 4.1 representa las estrategias multihilo.

El entorno de ejecución de Java es un ejemplo de sistema con un único proceso y múltiples hilos. Lo interesante en esta sección es el uso de múltiples procesos, cada uno de los cuales soporta múltiples hilos. Este enfoque es el de Windows, Solaris, Mach, y OS/2 entre otros. En esta sección se ofrece una descripción general del mecanismo multihilo; más adelante en este capítulo se discuten los detalles de los enfoques de Windows, Solaris y Linux.

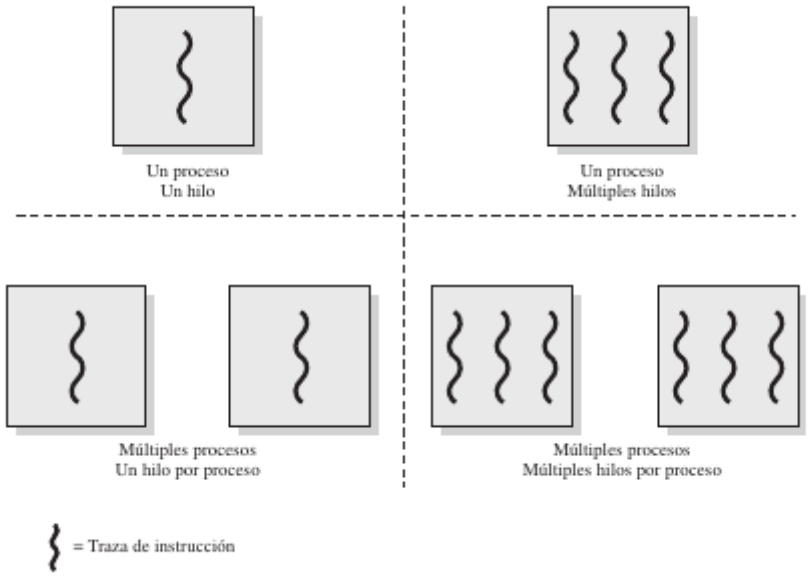


Figura 4.1. Hilos y procesos [ANDE97].

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:

- 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.

Todos los hilos de un proceso comparten el estado y los recursos de ese proceso, residen en el mismo espacio de direcciones y tienen acceso a los mismos datos. Cuando un hilo cambia determinados datos en memoria, otros hilos ven los resultados cuando acceden a estos datos. Si un hilo abre un archivo con permisos de lectura, los demás hilos del mismo proceso pueden también leer ese archivo. 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. Los estudios realizados por los que desarrollaron el sistema operativo Mach muestran que la creación de un hilo es diez veces más rápida que la creación de un proceso en UNIX
2. Lleva menos tiempo finalizar un hilo que un proceso.
3. Lleva menos tiempo cambiar entre dos hilos dentro del mismo proceso.
4. Los hilos mejoran la eficiencia de la comunicación entre diferentes programas que están ejecutando. En la mayor parte de los sistemas operativos, la comunicación entre procesos independientes requiere la intervención del núcleo para proporcionar protección y los mecanismos necesarios de comunicación. Sin embargo, ya que los hilos dentro de un mismo proceso comparten memoria y archivos, se pueden comunicar entre ellos sin necesidad de 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.

A veces los hilos son también útiles en un solo procesador ya que ayudan a simplificar la estructura de programas que realizan varias funciones diferentes. Cuatro ejemplos de uso de hilos en un sistema de multiprocesamiento de un solo usuario:

- 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. Por ejemplo, se puede diseñar un procesador de textos con protección contra un fallo de corriente que escriba el buffer de su memoria RAM a disco una vez por minuto. Se puede crear un hilo cuyo único trabajo sea crear una copia de seguridad periódicamente y que se planifique directamente a través del sistema operativo; no se necesita código adicional en el programa principal que proporcione control de tiempo o que coordine la entrada/salida.
- 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. De esta forma, aunque un hilo pueda estar bloqueado por una operación de E/S mientras lee datos, otro hilo puede estar ejecutando.
- 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.

En un sistema operativo que soporte hilos, la planificación y la activación se realizan a nivel de hilo; de aquí que la mayor parte de la información de estado relativa a la ejecución se mantenga en estructuras de datos a nivel de hilo. Existen, sin embargo, diversas acciones que afectan a todos los hilos de un proceso y que el sistema operativo debe gestionar a nivel de proceso. Suspender un proceso implica expulsar el espacio de direcciones de un proceso de memoria principal para dejar hueco a otro espacio de direcciones de otro proceso. Ya que todos los hilos de un proceso comparten el mismo espacio de direcciones, todos los hilos se suspenden al mismo tiempo. De forma similar, la finalización de un proceso finaliza todos los hilos de ese 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. 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. En particular, si se expulsa un proceso, todos sus hilos se deben expulsar porque comparten el espacio de direcciones del proceso.

Hay cuatro operaciones básicas relacionadas con los hilos que están asociadas con un cambio de estado del hilo:

- 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. Al nuevo hilo se le proporciona su propio registro de contexto y espacio de pila y se coloca en la cola de Listos.
- Bloqueo. Cuando un hilo necesita esperar por un evento se bloquea, almacenando los registros de usuario, contador de programa y punteros de pila. El procesador puede pasar a ejecutar otro hilo en estado Listo, dentro del mismo proceso o en otro diferente.
- 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.

Un aspecto importante es si el bloqueo de un hilo implica el bloqueo del proceso completo. En otras palabras, si se bloquea un hilo de un proceso, ¿esto impide la ejecución de otro hilo del mismo proceso incluso si el otro hilo está en estado de Listo? Sin lugar a dudas, se pierde algo de la potencia y flexibilidad de los hilos si el hilo bloqueado bloquea al proceso entero. Volveremos a este tema a continuación cuando veamos los hilos a nivel de usuario y a nivel de núcleo, pero por el momento consideraremos los beneficios de rendimiento de los hilos que no bloquean al proceso completo.

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. Por ejemplo, si dos hilos de modo simultáneo intentan añadir un elemento a una lista doblemente enlazada, se podría perder un elemento o la lista podría acabar malformada. Los asuntos que surgen y las técnicas que se utilizan en la sincronización de los hilos son, en general, los mismos que en la sincronización de procesos. Estos aspectos y técnicas se tratan en los Capítulos 5 y 6.

Ejemplo: Adobe Pagemaker

Hilos de nivel de usuario y de nivel de núcleo

Otras configuraciones

Tarea 2.1 - Planificación de procesos e hilos: 402-427

Capítulo 9: Planificación uniprocesador

9.1. Tipos de planificación del procesador

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.

Tabla 9.1. Tipos de planificación.

Planificación a largo plazo	La decisión de añadir un proceso al conjunto de procesos a ser ejecutados
Planificación a medio plazo	La decisión de añadir un proceso al número de procesos que están parcialmente o totalmente en la memoria principal
Planificación a corto plazo	La decisión por la que un proceso disponible será ejecutado por el procesador
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

- La planificación a largo plazo se realiza cuando se crea un nuevo proceso. Hay que decidir si se añade un nuevo proceso al conjunto de los que están activos actualmente.
- La planificación a medio plazo es parte de la función de intercambio (swapping function). 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.
- La planificación a corto plazo conlleva decidir cuál de los procesos listos para ejecutar es ejecutado. La Figura 9.2 reorganiza el diagrama de transición de estados de la Figura 3.9b para que se pueda apreciar el anidamiento de las funciones de planificación.

La planificación afecta al rendimiento del sistema porque determina qué proceso esperará y qué proceso progresará. Este punto de vista se presenta en la Figura 9.3, donde se muestran las colas involucradas en los estados de transición de los procesos. Fundamentalmente, la planificación es un problema de manejo de colas para minimizar el retardo en la cola y para optimizar el rendimiento en un entorno de colas.

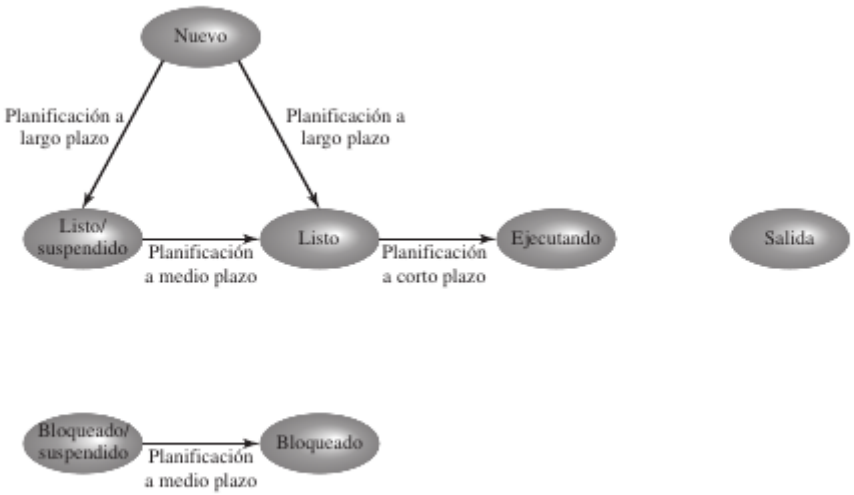


Figura 9.1. Planificación y transiciones de estado de los procesos.

Planificación a largo plazo

El planificador a largo plazo determina qué programas se admiten en el sistema para su procesamiento. De esta forma, se controla el grado de multiprogramación. Una vez admitido, un trabajo o programa de usuario se convierte en un proceso y se añade a la cola del planificador a corto plazo. En algunos sistemas, un proceso de reciente creación comienza en la zona de intercambio, en cuyo caso se añaden a la cola del planificador a medio plazo.

En un sistema por lotes, o en la parte de lotes de un sistema operativo de propósito general, los nuevos trabajos enviados se mandan al disco y se mantienen en una cola de lotes. El planificador a largo plazo creará procesos desde la cola siempre que pueda. En este caso hay que tomar dos decisiones. La primera, el planificador debe decidir cuándo el sistema operativo puede coger uno o más procesos adicionales. La segunda, el planificador debe decidir qué trabajo o trabajos se aceptan y son convertidos en procesos. Consideremos brevemente estas dos decisiones.

La decisión de **cuándo crear un nuevo proceso** se toma dependiendo del grado de multiprogramación deseado. Cuanto mayor sea el número de procesos creados, menor será el porcentaje de tiempo en que cada proceso se pueda ejecutar (es decir, más procesos compiten por la misma cantidad de tiempo de procesador). De esta forma, el planificador a largo plazo puede limitar el grado de multiprogramación a fin de proporcionar un servicio satisfactorio al actual conjunto de procesos. Cada vez que termine un trabajo, el planificador puede decidir añadir uno o más nuevos trabajos. Además, si la fracción de tiempo que el procesador está ocioso excede un determinado valor, se puede invocar al planificador a largo plazo.

La decisión de **qué trabajo admitir el siguiente**, puede basarse en un sencillo «primero en llegar primero en servirse», o puede ser una herramienta para gestionar el rendimiento del sistema. El criterio utilizado puede incluir la prioridad, el tiempo estimado de ejecución y los requisitos de E/S. Por ejemplo, si la información está disponible, el planificador puede intentar encontrar un compromiso entre procesos limitados por el procesador y procesos limitados por la E/S. Además, la decisión puede ser tomada dependiendo de los recursos de E/S que vayan a ser utilizados, de forma que se intente equilibrar el uso de la E/S.

Para los programas interactivos en un sistema de tiempo compartido, la petición de la creación de un proceso, puede estar generada por un usuario intentando conectarse al sistema. Los usuarios de tiempo compartido no se sitúan simplemente en una cola hasta que el sistema los pueda aceptar. Más exactamente, el sistema operativo aceptará a todos los usuarios autorizados hasta que el sistema se sature. La saturación se estima utilizando ciertas medidas prefijadas del

sistema. Llegado a este punto, una petición de conexión se encontrará con un mensaje indicando que el sistema está completo y el usuario debería reintentarlo más tarde.

Planificación a medio plazo

La planificación a medio plazo es parte de la función de intercambio. Los aspectos relacionados se discuten en los Capítulos 3, 7 y 8. Con frecuencia, la decisión de intercambio se basa en la necesidad de gestionar el grado de multiprogramación. En un sistema que no utiliza la memoria virtual, la gestión de la memoria es también otro aspecto a tener en cuenta. De esta forma, la decisión de meter un proceso en la memoria, tendrá en cuenta las necesidades de memoria de los procesos que están fuera de la misma.

Planificación a corto plazo

En términos de frecuencia de ejecución, el planificador a largo plazo ejecuta con relativamente poca frecuencia y toma la decisión de grano grueso de admitir o no un nuevo proceso y qué proceso admitir.

El planificador a medio plazo se ejecuta más frecuentemente para tomar decisiones de intercambio.

El planificador a corto plazo, conocido también como activador, ejecuta mucho más frecuentmente y toma las decisiones de grano fino sobre qué proceso ejecutar el siguiente. 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).

9.2. Algoritmos de planificacion

Criterios de planificación a corto plazo

El objetivo principal de la planificación a corto plazo es asignar tiempo de procesador de tal forma que se optimicen uno o más aspectos del comportamiento del sistema. Generalmente, se establece un conjunto de criterios con los que se pueden evaluar varias políticas de planificación. Los criterios utilizados habitualmente se pueden categorizar en dos dimensiones.

- Primero, se puede hacer una distinción entre criterios orientados al usuario y criterios orientados al sistema. Mientras que los criterios orientados al usuario son importantes en prácticamente todos los sistemas, los criterios orientados al sistema son generalmente menos importantes en sistemas de un solo usuario. En sistemas de un solo usuario, probablemente no es tan importante lograr una alta utilización del procesador o un alto rendimiento como que la respuesta del sistema a las aplicaciones del usuario sea aceptable.
    - Los criterios orientados al usuario están relacionados con el comportamiento del sistema tal y como lo percibe un usuario individual o un proceso. Un ejemplo es el tiempo de respuesta en un sistema interactivo. El tiempo de respuesta es el tiempo que transcurre entre el envío de una petición y la aparición de la respuesta. Esta cantidad es visible por parte del usuario y lógicamente es de su interés. Nos gustaría tener una política de planificación que proporcione «buen» servicio a varios usuarios. En el caso

del tiempo de respuesta se puede definir un límite, por ejemplo, 2 segundos. De esta forma el objetivo del mecanismo de planificación puede ser maximizar el número de usuarios que tienen un tiempo de respuesta medio menor o igual a 2 segundos.
    - Otros criterios son los relacionados con el sistema. Es decir, la atención se centra en el uso efectivo y eficiente del procesador. Un ejemplo es el rendimiento (throughput), que es la tasa con que los procesos se finalizan. Esta medida es valiosa en relación al sistema y es algo que nos gustaría maximizar. Sin embargo, no se centra en los servicios proporcionados al usuario. De esta forma, el rendimiento incumbe al administrador del sistema pero no a los usuarios.
- También es posible clasificar los criterios dependiendo de si están o no relacionados con las prestaciones.
    - Los criterios relacionados con las prestaciones son cuantitativos y generalmente pueden ser medidos. Algunos ejemplos son el tiempo de respuesta y el rendimiento.
    - Los criterios que no están relacionados con las prestaciones, o son cualitativos por naturaleza, o no pueden ser medidos y analizados. Un ejemplo de tales criterios es la previsibilidad. Nos gustaría que el servicio proporcionado a los usuarios tuviera las mismas características a lo largo del tiempo, independientemente de otros trabajos que se estén realizando en el sistema. Hasta cierto punto este criterio puede ser medido calculando las discrepancias en función de la carga de trabajo. Sin embargo, esta medida no es tan directa como medir el rendimiento o el tiempo de respuesta como una función de la carga de trabajo.

La Tabla 9.2 resume los criterios clave de la planificación. Son independientes, y es imposible optimizar todos ellos de forma simultánea. Por ejemplo, proporcionar un buen tiempo de respuesta puede requerir un algoritmo de planificación que cambie entre procesos frecuentemente. Esto incrementa la sobrecarga del sistema, reduciendo las prestaciones. De esta forma, el diseño de las políticas de un planificador implica un compromiso entre requisitos competitivos; los pesos relativos dados a los distintos requisitos dependerán de la naturaleza y el uso dado al sistema.

Tabla 9.2. Criterios de planificación.

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. Es una medida apropiada para trabajos por lotes.
Tiempo de respuesta (*response time*) Para un proceso interactivo, es el tiempo que trascurre desde que se lanza una petición hasta que se comienza a recibir la respuesta. A menudo un proceso puede producir alguna salida al usuario mientras continúa el proceso de la petición. De esta forma, desde el punto de vista del usuario, 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 subordinar otros objetivos al de maximizar el porcentaje de fechas tope conseguidas.
Orientados al usuario, otros
Previsibilidad Un trabajo dado debería ejecutarse aproximadamente 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 una gran oscilación en la sobrecarga del sistema o la necesidad de poner a punto el sistema para eliminar las inestabilidades.
Orientados al sistema, relacionados con las prestaciones
Rendimiento La política de planificación debería intentar maximizar el número de procesos completados por unidad de tiempo. Es una medida de cuánto trabajo está siendo realizado. Esta medida depende claramente de la longitud media de los procesos, pero está influenciada por la política de planificación, que puede afectar a la utilización.
Utilización del procesador Es el porcentaje de tiempo que el procesador está ocupado. Para un sistema compartido costoso, es un criterio significativo. En un sistema de un solo usuario y en otros sistemas, tales como los sistemas de tiempo real, este criterio es menos importante que algunos otros.
Orientados al sistema, otros
Equidad En ausencia de orientación de los usuarios o de orientación proporcionada por otro sistema, los procesos deben ser tratados de la misma manera, y ningún proceso 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. Este criterio también implica planificación a medio plazo y a largo plazo.

El uso de prioridades

En muchos sistemas, a cada proceso se le asigna una prioridad y el planificador siempre elegirá un proceso de prioridad mayor sobre un proceso de prioridad menor. La Figura 9.4 muestra el uso de las prioridades. Por claridad, el diagrama de colas está simplificado, ignorando la existencia de múltiples colas bloqueadas y de estados suspendidos (comparar con la Figura 3.8a). En lugar de una sola cola de procesos listos para ejecutar, se proporcionan un conjunto de colas en orden descendente de prioridad: CL0, CL1, ... CLn, con la prioridad [CLi]>prioridad[CLj] para i<j. Cuando se va a realizar una selección en la planificación, el planificador comenzará en la cola de listos con la prioridad más alta.

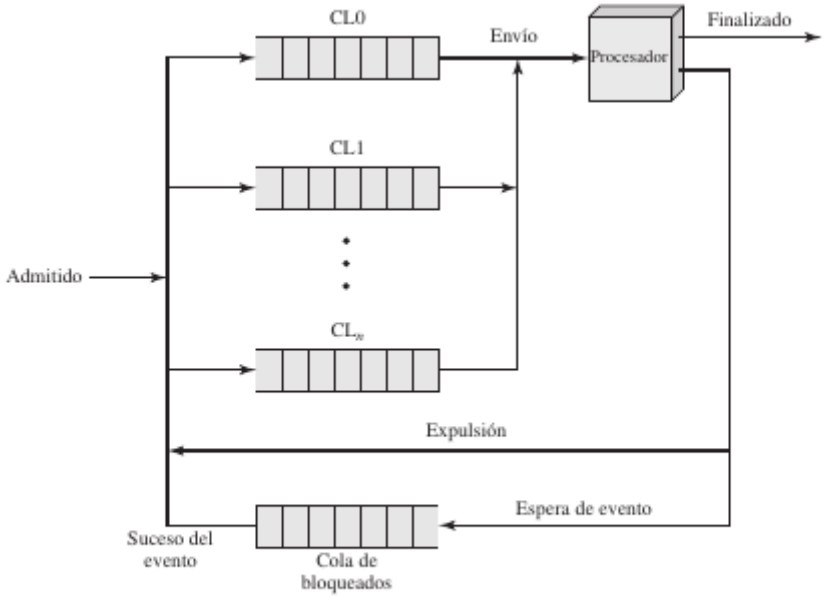


Figura 9.4. Encolamiento con prioridades.

Un problema de los esquemas de planificación con prioridades es que los procesos con prioridad más baja pueden sufrir inanición. Esto sucederá si hay siempre un conjunto de procesos de mayor prioridad listos para ejecutar. Si este comportamiento no es deseable, la prioridad de un proceso puede cambiar con su antigüedad o histórico de ejecución. Posteriormente daremos un ejemplo de esta situación.

Políticas de planificación alternativas

Tabla 9.3. Características de algunas políticas de planificación.

	Función de Selección	Modo de Decisión	Rendimiento	Tiempo de Respuesta	Rendimiento	Efecto sobre los Procesos	Inanición
FCFS	max[*w*]	No expulsiva	No especificado	Puede ser alto especialmente si hay mucha diferencia entre los tiempos de ejecución de los procesos	Mínima	Penaliza procesos cortos; penaliza procesos con mucha E/S	No
Turno Rotatorio *(round robin)*	constante	Expulsiva (por rodajas de tiempo)	Puede ser mucho si la rodaja es demasiado pequeña	Proporciona buen tiempo de respuesta para procesos cortos	Mínima	Tratamiento justo	No
SPN	min[*s*]	No expulsiva	Alto	Proporciona buen tiempo de respuesta para procesos cortos	Puede ser alta	Penaliza procesos largos	Posible
SRT	min[*s*-*e*]	Expulsiva (a la llegada)	Alto	Proporciona buen tiempo de respuesta	Puede ser alta	Penaliza procesos largos	Posible
HRRN	max(*w*+*s*/*s*)	No expulsiva	Alto	Proporciona buen tiempo de respuesta	Puede ser alta	Buen equilibrio	No
Feedback	(ver texto)	Expulsiva (por rodajas de tiempo)	No especificado	No especificado	Puede ser alta	Puede favorecer procesos con mucha E/S	Posible

w = tiempo de espera
e = tiempo de ejecución hasta el momento
s = tiempo total de servicio requerido por el proceso, incluyendo *e*

- **Función de selección:** determina qué proceso, entre los procesos listos, se selecciona para ejecutar. Puede basarse en prioridades o caracterísiticas cuantitativas:
    - w = tiempo usado en el sistema hasta el momento, esperando o ejecutando (tiempo de espera)

    - e = tiempo usado en ejecución hasta el momento (tiempo de CPU)
    - s = tiempo total de servicio requerido por el proceso, incluyendo e. Generalmente, esta cantidad debe ser estimada o proporcionada por el usuario.
- **Modo de decisión:** especifica los instantes de tiempo en que se ejecuta la función de selección. Hay dos categorías:
    - Sin expulsion (nonpreemptive). En este caso, una vez que el proceso está en el estado Ejecutando, continúa ejecutando hasta que (a) termina o (b) se bloquea para esperar E/S o para solicitar algún servicio al sistema operativo.
    - Con expulsión (preemptive). Un proceso ejecutando en un determinado momento puede ser interrumpido y pasado al estado de listo por el sistema operativo. La decisión de expulsar puede ser tomada cuando llega un nuevo proceso, cuando llega una interrupción que pasa un proceso de bloqueado a estado de listo, o periódicamente, basándose en las interrupciones del reloj.
        - Las políticas expulsivas tienen mayor sobrecarga que las no expulsivas, pero pueden proporcionar mejor servicio a la población total de procesos, ya que previenen que cualquier proceso pueda monopolizar el procesador durante mucho tiempo. Además, el coste de la expulsión puede resultar relativamente bajo a través de la utilización de mecanismos eficientes de cambios de proceso (tanta ayuda del hardware como sea posible) y proporcionando una gran cantidad de memoria principal para dejar un alto porcentaje de programas en la memoria principal.
- El tiempo de estancia (turnaround time – TAT) es el tiempo de residencia Tn o tiempo total que un elemento está en el sistema (tiempo de espera, más tiempo de servicio). Algo más útil es el tiempo de estancia normalizado, que es tasa del tiempo de estancia y tiempo de servicio. Este valor indica el retardo relativo experimentado por un proceso. Así, mayor sea el tiempo de ejecución, mayor será el valor absoluto del retardo que se puede tolerar. El menor valor posible de esta tasa es 1,0; valores mayores corresponden a un servicio de nivel menor.

FIFO / FCFS

La directiva de planificación más sencilla es primero en llegar primero en servirse (FCFS), también conocida como primero- entra-primero-sale (FIFO) o como un sistema de colas estricto. En el momento en que un proceso pasa al estado de listo, se une a la cola de listos. Cuando el proceso actualmente en ejecución deja de ejecutar, se selecciona para ejecutar el proceso que ha estado más tiempo en la cola de listos. FCFS funciona mucho mejor para procesos largos que para procesos cortos. Considérese el siguiente ejemplo, basado en [FINK88]:

Proceso	Tiempo de Llegada	Tiempo de Servicio (T_s)	Tiempo de Comienzo	Tiempo de Finalización	Tiempo de Estancia (T_r)	T_r/T_s
W	0	1	0	1	1	1
X	1	100	1	101	100	1
Y	2	1	101	102	100	100
Z	3	100	102	202	199	1,99
Media					100	26

El tiempo de estancia normalizado para el proceso Y es excesivamente grande en comparación con los otros procesos: el tiempo total que está en el sistema es 100 veces el tiempo requerido para su proceso. Esto sucederá siempre que llegue un proceso corto a continuación de un proceso largo. Por otra parte, incluso en este ejemplo extremo, los procesos largos no van mal. El

proceso Z tiene un tiempo de estancia de casi el doble que Y, pero su tiempo de residencia normalizado está por debajo de 2,0.

Otro problema de FCFS es que tiende a favorecer procesos limitados por el procesador sobre los procesos limitados por la E/S. Considere que hay una colección de procesos, uno de los cuales está limitado por el procesador y un número de procesos limitados por la E/S. Cuando el proceso limitado por el procesador está ejecutando, el resto de los procesos debe esperar. Alguno de estos estarán en las colas de E/S (estado bloqueado) pero se pueden mover a la cola de listos mientras que el proceso limitado por el procesador sigue ejecutando. En este punto, la mayor parte de los dispositivos de E/S pueden estar ociosos, incluso aunque exista trabajo potencial que pueden hacer. Cuando el proceso actual en ejecución deja el estado Ejecutando, los procesos listos pasarán al estado de Ejecutando y se volverán a bloquear en un evento de E/S. Si el proceso limitado por el procesador está también bloqueado, el procesador se quedará ocioso. De esta manera, FCFS puede conllevar usos ineficientes del procesador y de los dispositivos de E/S.

FCFS no es una alternativa atractiva por sí misma para un sistema uniprocesador. Sin embargo, a menudo se combina con esquemas de prioridades para proporcionar una planificación eficaz. De esta forma, el planificador puede mantener varias colas, una por cada nivel de prioridad, y despachar dentro de cada cola usando primero en llegar primero en servirse. Veremos un ejemplo de este sistema más adelante, cuando hablemos de la planificación retroalimentada (feedback).

RR

Turno rotatorio (round robin), una forma directa de reducir el castigo que tienen los trabajos cortos con FCFS es la utilización de expulsión basándose en el reloj. La política más sencilla es la del turno rotatorio, también denominada planificación cíclica. Se genera una interrupción de reloj cada cierto intervalo de tiempo. Cuando sucede la interrupción, el proceso actual en ejecución se sitúa en la cola de listos, y se selecciona el siguiente trabajo según la política FCFS. Esta técnica es también conocida como cortar el tiempo (time slicing), porque a cada proceso se le da una rodaja de tiempo antes de ser expulsado.

Con la planificación en turno rotatorio, el tema clave de diseño es la longitud del quantum de tiempo, o rodaja, a ser utilizada. Si el quantum es muy pequeño, el proceso se moverá por el sistema relativamente rápido. Por otra parte, existe una sobrecarga de procesamiento debido al manejo de la interrupción de reloj y por las funciones de planificación y activación. De esta forma, se deben evitar los quantums de tiempo muy pequeños. Una buena idea es que el quantum de tiempo debe ser ligeramente mayor que el tiempo requerido para una interacción o una función típica del proceso. Si es menor, muchos más procesos necesitarán, al menos, dos quantums de tiempo. La Figura 9.6 muestra el efecto que tiene en el tiempo de respuesta.

Nótese que en el caso extremo de un quantum de tiempo mayor que el proceso más largo en ejecución, la planificación en turno rotatorio degenera en FCFS.

La planificación en turno rotatorio es particularmente efectiva en sistemas de tiempo compartido de propósito general o en sistemas de procesamiento transaccional. Una desventaja de la planificación en turno rotatorio es que trata de forma desigual a los procesos limitados por el procesador y a los procesos limitados por la E/S. Generalmente, un proceso limitado por la E/S tiene ráfagas de procesador más cortas (cantidad de tiempo de ejecución utilizada entre operaciones de E/S) que los procesos limitados por el procesador. Si hay una mezcla de los dos tipos de procesos, sucederá lo siguiente: un proceso limitado por la E/S utiliza el procesador durante un periodo corto y luego se bloquea; espera a que complete la operación de E/S y a continuación se une a la cola de listos. Por otra parte, un proceso limitado por el procesador

generalmente utiliza la rodaja de tiempo completa mientras ejecuta e inmediatamente vuelve a la cola de listos. De esta forma, los procesos limitados por el procesador tienden a recibir una rodaja no equitativa de tiempo de procesador, lo que conlleva un mal rendimiento de los procesos limitados por la E/S, uso ineficiente de los recursos de E/S y un incremento en la varianza del tiempo de respuesta.

Primero el proceso más corto SPN

Otor enfoque para reducir el sesgo a favor de los procesos largos inherente al FCFS es la política primero el proceso más corto (SPN). Es una política no expulsiva en la que se selecciona el proceso con el tiempo de procesamiento más corto esperado. De esta forma un proceso corto se situará a la cabeza de la cola, por delante de los procesos más largos.

Un problema de la política SPN es la necesidad de saber, o al menos de estimar, el tiempo de procesamiento requerido por cada proceso. Para trabajos por lotes, el sistema puede requerir que el programador estime el valor y se lo proporcione al sistema operativo. Si la estimación del programador es mucho menor que el tiempo actual de ejecución, el sistema podría abortar el trabajo. En un entorno de producción, ejecutan frecuentemente los mismos trabajos y, por tanto, se pueden recoger estadísticas. Para procesos interactivos, el sistema operativo podría guardar una media del tiempo de ejecución de cada «ráfaga» de cada proceso. La forma más sencilla de cálculo podría ser la siguiente:

$$S_{n+1} = \frac{1}{n}\sum_{i=1}^{n} T_i$$

donde,

- Ti = tiempo de ejecución del procesador para la instancia i-ésima de este proceso (tiempo total de ejecución para los trabajos por lotes; tiempo de ráfaga de procesador para los trabajos interactivos)
- Si = valor predicho para la instancia i-ésima
- S1 = valor predicho para la primera instancia; no calculado

Para evitar volver a calcular la suma completa cada vez, podemos reescribir la ecuación como

$$S_{n+1} = \frac{1}{n} T_n + \frac{n-1}{n} S_n$$

Un riesgo con SPN es la posibilidad de inanición para los procesos más largos, si hay una llegada constante de procesos más cortos. Por otra parte, aunque SPN reduce la predisposición a favor de los trabajos más largos, no es deseable para un entorno de tiempo compartido o de procesamiento transaccional por la carencia de expulsión. Volviendo al peor caso descrito con FCFS, los procesos W, X, Y y Z seguirán ejecutando en el mismo orden, penalizando fuertemente al proceso corto Y.

Menor tiempo restante SRT

Menor tiempo restante (shortest remaining time), la política del menor tiempo restante (SRT) es una versión expulsiva de SPN. En este caso, el planificador siempre escoge el proceso que tiene el menor tiempo de proceso restante esperado. Cuando un nuevo proceso se une a la cola de listos, podría tener un tiempo restante menor que el proceso actualmente en ejecución. Por tanto, el

planificador podría expulsar al proceso actual cuando llega un nuevo proceso.

Al igual que con SPN, el planificador debe tener una estimación del tiempo de proceso para realizar la función seleccionada, y existe riesgo de inanición para los procesos más largos. SRT no tiene el sesgo en favor de los procesos largos que se puede encontrar en FCFS.

A diferencia del turno rotatorio, no se generan interrupciones adicionales, reduciéndose la sobrecarga. Por otra parte, se deben almacenar los tiempos de servicio transcurridos, generando sobrecarga. SRT debería mejorar los tiempos de estancia de SPN, porque a un trabajo corto se le da preferencia sobre un trabajo más largo en ejecución.

Primero el de mayor tasa de respuesta HRRN Highest Response Ratio Next

Hemos utilizado el tiempo de estancia normalizado, que es la tasa entre el tiempo de estancia y el tiempo actual de servicio, como un valor importante. Para cada proceso individual, nos gustaría minimizar esta tasa, y nos gustaría minimizar el valor medio sobre todos los procesos.

En general, no podemos saber por adelantado el tiempo de servicio de los procesos, pero lo podemos aproximar, basándonos en la historia anterior, en alguna entrada de usuario o en un gestor de configuración. Considerar la siguiente tasa:

$$R = \frac{(w+s)}{s}$$

donde

- R = tasa de respuesta
- w = tiempo invertido esperando por el procesador
- s = tiempo de servicio esperado

Si se despacha inmediatamente al proceso con este valor, R es igual al tiempo de estancia normalizado. Observar que el valor mínimo de R es 1,0, que sucede cuando un proceso acaba de entrar en el sistema.

De esta forma, nuestra regla de planificación es como sigue: cuando se completa o bloquea el proceso actual, elegir el proceso listo con el mayor valor de R. Este enfoque es atractivo por que tiene en cuenta la edad del proceso. Mientras que se favorece a los procesos más cortos (un menor denominador lleva a una mayor tasa), el envejecimiento sin servicio incrementa la tasa, por lo que un proceso más largo podría competir con los trabajos más cortos.

Como en SRT y SPN, el tiempo de servicio se debe estimar para usar la política primero el de mayor tasa de respuesta (HRRN).

Feedback

Si no es posible averiguar el tiempo de servicio de varios procesos, SPN, SRT y HRRN no se pueden utilizar. Otra forma de establecer una preferencia para los trabajos más cortos es penalizar a los trabajos que han estado ejecutando más tiempo. En otras palabras, si no podemos basarnos en el tiempo de ejecución restante, nos podemos basar en el tiempo de ejecución utilizado hasta el momento.

La forma de hacerlo es la siguiente. La planificación se realiza con expulsión (por rodajas de tiempo), y se utiliza un mecanismo de prioridades dinámico. Cuando un proceso entra en el sistema, se sitúa en CL0 (véase Figura 9.4). Después de su primera expulsión, cuando vuelve al estado de Listo, se sitúa en CL1. **Cada vez que es expulsado, se sitúa en la siguiente cola de menor prioridad. Un proceso corto se completará de forma rápida, sin llegar a migrar muy**

lejos en la jerarquía de colas de listos. Un proceso más largo irá degradándose gradualmente. De esta forma, se favorece a los procesos nuevos más cortos sobre los más viejos y largos. Dentro de cada cola, excepto en la cola de menor prioridad, se utiliza un mecanismo FCFS. Una vez en la cola de menor prioridad, un proceso no puede descender más, por lo que es devuelto a esta cola repetidas veces hasta que se consigue completar. De esta forma, esta cola se trata con una política de turno rotatorio.

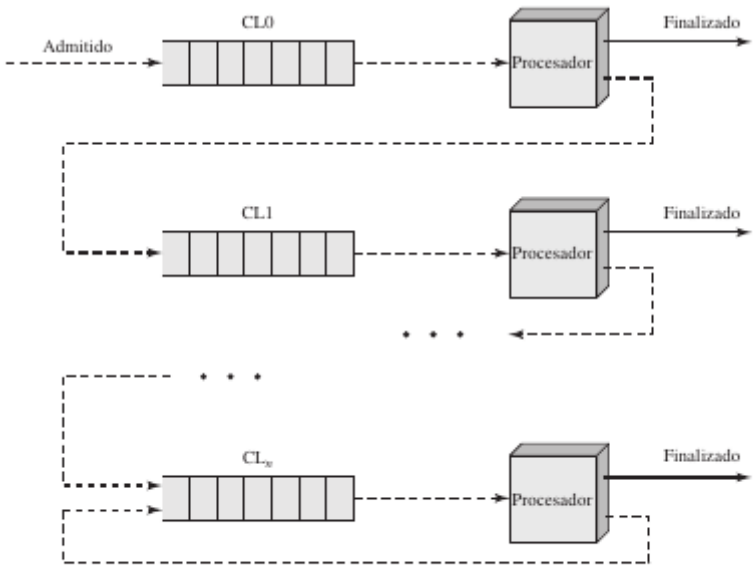


Figura 9.10. Planificación retroalimentada.

Hay una serie de variaciones de este esquema. Una versión sencilla consiste en realizar la expulsión de la misma manera que la política rotatoria: con intervalos periódicos. Nuestro ejemplo muestra esta situación (Figura 9.5 y Tabla 9.5) para una rodaja de una unidad de tiempo. Observar que en este caso, el comportamiento es similar a un turno rotatorio con rodaja de tiempo 1.

Un problema que presenta el esquema simple antes contado e**s que el tiempo de estancia para procesos más largos se puede alargar de forma alarmante. De hecho, puede ocurrir inanición si están entrando nuevos trabajos frecuentemente en el sistema.** Para compensar este problema, podemos variar los tiempos de expulsión de cada cola: un proceso de la cola CL0 tiene una rodaja de una unidad de tiempo; un proceso de la cola CL1 tiene una rodaja de dos unidades de tiempo, y así sucesivamente. En general, a un proceso de la cola CLi se le permite ejecutar 2i unidades de tiempo antes de ser expulsado. Este esquema se muestra en nuestro ejemplo de la Figura 9.5 y Tabla 9.5.

Incluso asignando rodajas de tiempo mayores a las colas de menor prioridad, un proceso grande puede sufrir inanición. **Una posible solución es promover a los procesos a colas de mayor prioridad después de que pasen un determinado tiempo esperando servicio en su cola actual.**

Comparación de rendimiento

La planificación contribución justa (fair-share scheduling)

Todos los algoritmos de planificación discutidos hasta el momento tratan a la colección de procesos listos como un simple conjunto de procesos de los cuales seleccionar el siguiente a ejecutar. Este conjunto de procesos se podría romper con el uso de prioridades, pero es homogéneo.

Sin embargo, en un sistema multiusuario, si las aplicaciones o trabajos de usuario se pueden organizar como múltiples procesos (o hilos), hay una estructura en la colección de procesos que no se reconoce en los planificadores tradicionales. Desde el punto de vista del usuario, la preocupación no es cómo ejecuta un simple proceso, sino cómo ejecutan su conjunto de procesos, que forman una aplicación. De esta forma, sería deseable tomar decisiones de planificación basándose en estos conjuntos de procesos. Este enfoque se conoce normalmente como planificación contribución justa.

Además, el concepto se puede extender a un grupo de usuarios, incluso si cada usuario se representa con un solo proceso. Por ejemplo, en un sistema de tiempo compartido, podríamos querer considerar a todos los usuarios de un determinado departamento como miembros del mismo grupo. Se podrían tomar las decisiones de planificación intentando dar a cada grupo un servicio similar. De esta forma, si muchos usuarios de un departamento se meten en el sistema, nos gustaría ver cómo el tiempo de respuesta se degrada sólo para los miembros de este departamento, más que para los usuarios de otros departamentos.

El término contribución justa indica la filosofía que hay detrás de este planificador. A cada usuario se le asigna una prima de algún tipo que define la participación del usuario en los recursos del sistema como una fracción del uso total de dichos recursos. En particular, a cada usuario se le asigna una rodaja del procesador. Este esquema debería operar, más o menos, de forma lineal. Así, si un usuario A tiene el doble de prima que un usuario B, en una ejecución larga, el usuario A debería ser capaz de hacer el doble de trabajo que el usuario B. El objetivo del planificador contribución justa es controlar el uso para dar menores recursos a usuarios que se han excedido de su contribución justa y mayores recursos a los que no han llegado.

Fair Share Scheduler FSS

FSS considera el histórico de ejecución de un grupo de procesos relacionados, junto con el histórico de ejecución de cada uno de los procesos, para tomar decisiones de planificación. El sistema divide a la comunidad de usuarios en un conjunto de grupos y destina una fracción del procesador a cada grupo. De esta forma, podría haber 4 grupos, cada uno con un 25% del procesador. De hecho, a cada grupo se le proporciona un sistema virtual que ejecuta más lento que un sistema completo.

La planificación se realiza en base a la prioridad y tiene en cuenta la prioridad del proceso, su uso reciente de procesador y el uso reciente de procesador del grupo al que pertenece el proceso. Mayor sea el valor numérico de la prioridad, menor será la prioridad. La siguiente fórmula se aplica al proceso j del grupo k:

$$CPU_j(i) = \frac{CPU_j(i-1)}{2}$$

$$GCPU_k(i) = \frac{GCPU_k(i-1)}{2}$$

$$P_j(i) = \text{Base}_j + \frac{CPU_j(i)}{2} + \frac{GCPU_k(i)}{4 \times W_k}$$

donde

- CPUj(i) = Medida de utilización del procesador por el proceso j en el intervalo i
- GCPUk(i) = Medida de utilización del procesador del grupo k en el intervalo i
- Pj(i) = Prioridad del proceso j al comienzo del intervalo i; valores más pequeños equivalen a prioridades más altas
- Basej = Prioridad base del proceso j
- Wk = Prima asignada al grupo k, con la restricción que 0 < Wk <= 1 y sum Wk = 1