Descripción de procesos: Stalling pp. 108-143

Página 108

En general, todos los SOs multiprogramados se construyen en torno al concepto de proceso. Los SOs deben ser capaces de intercalar la ejecución de distintos procesos, de reservar recursos para estos en base a una política específica evitando interbloqueos y de darles soporte para que se comuniquen entre ellos y para crearlos.

Un proceso se puede definir como una entidad que consiste en un número de elementos. Los dos elementos esenciales serían el código del 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 ID único asociado al proceso, para distinguirlo del resto (PID).
- **Estado:** Si el proceso está actualmente corriendo, está en ejecución.
- **Prioridad:** Nivel de prioridad respecto 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 presentes 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…
- **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…

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), 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. 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.

Página 110

ESTADOS DE LOS PROCESOS

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

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.

El activador (dispatcher) es el programa que intercambia el procesador de un proceso a otro.

Página 113

UN MODELO DE PROCESO DE DOS ESTADOS

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

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.

Página 114

CREACIÓN Y TERMINACIÓN DE PROCESOS

Creación de un proceso. Cuando se va a añadir un nuevo proceso a aquellos que se están gestionando en un determinado momento, el sistema operativo construye las estructuras de datos que se usan para manejar el proceso (como se describió en la Sección 3.3) y reserva el espacio de direcciones en memoria principal para el proceso.

Existen cuatro eventos comunes que llevan a la creación de un proceso. En un entorno por lotes, un proceso se crea como respuesta a una solicitud de trabajo. En un entorno interactivo, un proceso se crea cuando un nuevo usuario entra al sistema. Además, un proceso puede ser creado por otro proceso ya existente por motivos de modularidad o para explotar el paralelismo y también puede ser creado por el SO para proporcionar un servicio.

Cuando un proceso lanza a otro, el proceso que lo lanza es el proceso padre y el lanzado, el proceso hijo.

Terminación de un proceso. Las razones por la que puede terminar un proceso son las siguientes:

- **Finalización normal.** El proceso ejecuta una llamada al SO para indicar que ha terminado.
- **Límite de tiempo excedido.** El proceso ha ejecutado más del establecido.
- **Memoria no disponible.**
- **Violaciones de frontera.** El proceso trata de acceder a una dirección de memoria a la que no tiene acceso.
- **Error de protección.** El proceso intenta usar un recurso para el que no tiene recursos o intenta escribir en un archivo de solo lectura.
- **Error aritmético.** El proceso trata de realizar una operación de cálculo no permitida como dividir por 0.
- **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 en una operación de E/S.
- **Instrucción no válida.** El proceso intenta ejecutar una instrucción inexistente (a menudo el resultado de un salto a un área de datos y el intento de ejecutar dichos datos).
- **Instrucción privilegiada.**
- **Uso inapropiado de datos.** Una porción de datos es de tipo erróneo o no se encuentra inicializada.
- **Intervención del operador o del SO.**
- **Terminación del proceso padre.**
- **Solicitud del proceso padre.**

Página 117

MODELO DE PROCESO DE 5 ESTADOS

Puesto que el modelo anterior es ineficiente teniendo en cuenta que procesos de la cola de no ejecutados puede ser que estén bloqueados, lo cual provocaría que el activador tendría que ir recorriendo toda la cola de procesos no ejecutados buscando aquellos que no estén bloqueados, se propone un modelo con 5 estados, estos son:

- **Nuevo:** Un proceso se acaba de crear pero aún no ha sido admitido en la cola de procesos ejecutables. Esto a menudo se trata de un proceso que tiene el PCB creado pero que aún no ha sido cargado en memoria principal (ya sea porque el número de procesos que puede haber en el sistema por rendimiento o por limitación de memoria o por otra cosa).
- **Listo:** Cola de procesos preparados para ser ejecutados cuando les toque.
- **Ejecutando:** Proceso que está actualmente en ejecución (en el libro supondrán que estamos en computado monoprocesador).
- **Bloqueado:** Un proceso que no se puede ejecutar hasta que se cumpla un evento determinado o hasta que se complete una operación de E/S.
- **Saliente:** Un proceso que se libera del grupo de procesos ejecutables debido a que ha terminado o a que ha sido detenido o abortado por alguna razón.

Cuando un proceso se encuentra en el estado Nuevo, la información relativa al proceso que se necesite por parte del SO se mantiene en tablas de control de memoria principal, pese a que el

propio proceso no se encuentre en memoria principal. Mientras un estado está en este estado, este permanece almacenado en almacenamiento secundario, generalmente en disco.

Cuando un proceso pasa al estado Saliente, el SO mantiene las tablas y otra información asociada con el trabajo temporalmente, de forma que otros programas que puedan necesitar dicha información puedan obtenerla antes de que el SO la elimine. Una vez que todos los programas han obtenido los datos necesarios, el SO elimina la información relativa al proceso que ha finalizado.

Cuando un proceso pasa del estado Listo a Ejecutando es porque el planificador así lo ha decidido.

Cuando un proceso pasa del estado Ejecutando a Listo es a menudo, o bien porque ha cumplido el límite de tiempo que puede ejecutarse de forma ininterrumpida, o bien porque se ha desbloqueado o ha entrado otro proceso porque tiene más prioridad, o incluso porque el propio proceso se lo pida al procesador, este es el caso de los procesos que realizan alguna función de auditoría o de mantenimiento de forma periódica.

Un proceso puede ir del estado Listo al estado Saliente si su proceso padre termina u ordena que el proceso hijo termine, esto mismo pasa también si un proceso pasa del estado bloqueado al estado saliente.

Otra idea es la de tener colas de listos y de bloqueados. El hecho de tener varias colas de bloqueados, una para cada evento hace que el procesador no tenga que buscar por toda la lista de bloqueados qué procesos estaban esperando a qué evento. De forma análoga, en el caso en el que tengamos una planificación por prioridades, si tenemos una cola de listos para cada prioridad, evitaremos que el procesador tenga que ir por todos los procesos comprobando su prioridad a la hora de seleccionar cual se ejecuta.

Página 120

PROCESOS SUSPENDIDOS

En este tipo de modelos vemos que a menudo ocurre un problema y es que, debido a la gran diferencia de velocidad entre el procesador y la E/S, es común ver que todos los procesos en memoria se llegan a encontrar bloqueados esperando por estas operaciones E/S. Debido a esto, incluso en sistemas multiprogramados, el procesador puede llegar a estar ocioso la mayor parte del tiempo.

Una posible solución sería expandir la memoria principal de forma que sea capaz de albergar más procesos, pero tenemos el problema del coste, y pese a que la memoria cada día se va a abaratando más, los procesos cada vez van ocupando más memoria, por lo que este problema no acaba solucionándose.

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, 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. Se debe reconocer, sin embargo, que todos los procesos en estado Suspendido estaban originalmente en estado Bloqueado, en espera de un evento en particular. Cuando el evento sucede, el proceso no está Bloqueado y está potencialmente disponible para su ejecución.

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 2 x 2 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.

La explicación ha asumido que no se utiliza memoria virtual y que un proceso está completamente en memoria principal, o completamente fuera de la misma.

Ahora tenemos la transición de bloqueado a bloqueado/suspendido. Si no hay procesos listos, entonces al menos uno de los bloqueados se transfiere a disco para hacer espacio para otro proceso que no se encuentra bloqueado. Esta transición puede realizarse incluso si hay procesos listos disponibles en caso de que el SO crea que el proceso actualmente en ejecución, o los procesos listos que desea ejecutar requieren más memoria principal para mantener un rendimiento adecuado.

Transición de bloqueado/suspendido a listo/suspendido. Esto ocurre cuando un proceso en el estado bloqueado/suspendido se mueve al estado listo/suspendido porque ha sucedido el evento al que estaba esperando. Esto requiere que el SO tenga información de estado concerniente a los procesos suspendidos.

Otra transición es de listo/suspendido a listo. Cuando no hay más procesos listos en memoria

principal, el SO trae uno para continuar su ejecución. Puede darse el caso de que un proceso en listo/suspendido tenga más prioridad que uno en listo, en este caso, el SO puede haberse diseñado de forma que le de más importancia a traer procesos de mayor prioridad que a minimizar el efecto del swapping.

Otra es de listo a listo/suspendido. Esto puede ocurrir si de esta forma se consigue liberar un bloque grande de memoria que es necesario, o bien si el proceso en cuestión es de baja prioridad y hay uno Bloqueado de alta prioridad que el SO cree que se liberará pronto, en dicho caso, puede decidir suspender el listo antes que el bloqueado.

Otras transiciones adicionales son las siguientes:

De nuevo a listo/suspendido o de nuevo a listo. Ahora es posible añadir un proceso que acaba de ser creado a la cola de listos/suspendidos directamente. Esta transición listo/suspendido puede ser útil de cara a cuando no haya suficiente espacio en memoria principal para el nuevo proceso. Por otro lado, tenemos la creación de procesos diferida, creándolos cuanto más tarde, hace posible reducir la sobrecarga del SO y le permite realizar las tareas de creación de procesos cuando el sistema está lleno de procesos bloqueados.

De bloqueado/suspendido a bloqueado. Esta transición puede parecer un poco inútil, pero pensemos en este escenario: un proceso termina, liberando algo de 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 SO 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.

De ejecutando a 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 a saliente. Si tenemos en cuenta que si un proceso termina, o así lo ordena, mata a su hijo, esto puede ocurrir desde cualquier estado.

Página 124

OTROS USOS PARA LA SUSPENSIÓN DE PROCESOS

Aparte de lo que hemos hablado hasta ahora, puede haber otras razones para suspender un proceso, aquí las enumero todas:

- **Swapping:** El SO necesita liberar memoria principal para traer un proceso al estado Listo.
- **Otras razones del SO:** El SO puede suspender un proceso que considera que puede causar algún problema.
- **Solicitud interactiva del usuario:** Un usuario puede desear suspender la ejecución de un programa con motivo de su depuración o porque está utilizando un recurso.
- **Temporización:** Un proceso puede ejecutarse periódicamente y puede suspenderse mientras espera al siguiente intervalo de ejecución.
- **Solicitud del proceso padre:** Un proceso padre puede querer suspender la ejecución de un

proceso hijo para examinar o modificar dicho proceso suspendido, o para coordinar la actividad de varios procesos hijos.

Página 126

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. A pesar de que los detalles difieren de un sistema operativo a otro, fundamentalmente, todos los sistemas operativos mantienen información de la memoria, la E/S, los ficheros y los procesos.

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 algunos sistemas operativos, la gran mayoría del detalle de la gestión de ficheros sí que se gestiona en el propio sistema operativo.

En último lugar, el sistema operativo debe mantener tablas de procesos para gestionar los 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.

Pese a que hayamos hablado de 4 tipos de tablas distintas, hay que tener en cuenta que realmente, estas 4 tablas están entrelazadas y referenciadas entre sí de alguna manera.

ESTRUCTURAS DE CONTROL DE 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.

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 control del proceso (BCP). Nos podemos referir al conjunto de programa, datos, pila, y atributos, como la imagen del proceso.

La posición de la imagen del proceso dependerá del esquema de gestión de memoria que se utilice. 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.

Los sistemas operativos modernos suponen la existencia de un hardware de paginación que permite el uso de la memoria física no contigua, para dar soporte a procesos parcialmente residentes en la memoria principal. En esos casos, una parte de la imagen del proceso se puede encontrar en dicha memoria principal, con las restantes en memoria secundaria. De esta forma, las tablas mantenidas por sistema operativo deben mostrar la localización de cada página de la imagen del proceso.

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).

Podemos agrupar la información del bloque de control del proceso (PCB) en tres categorías generales:

- Identificación del proceso
  - ID del proceso
  - ID del proceso padre
  - ID de los procesos hijo
  - ID del usuario que es responsable del trabajo

- Información de estado del procesador
  - Registros visibles por el usuario, es decir, aquellos a los que se puede hacer referencia por medio del lenguaje máquina que ejecuta el procesador cuando está en modo usuario. Suele haber entre 8 y 32 de estos registros.

  - Registros de estado y control o palabra de estado de programa (PSW), entre estos tenemos el contador de programa, los códigos de condición (cero, acarreo, igual…) y la información de estado, que incluye los flags de interrupciones habilitadas/deshabilitadas, modo de ejecución…

- Puntero de pila. Cada proceso tiene 1 o más pilas asociadas a él. Una pila se utiliza para almacenar los parámetros y las direcciones de retorno de los procedimientos y llamadas al sistema. El puntero de pila apunta a la parte más alta de esta.

- Información de control del proceso (necesaria para controlar y coordinar varios procesos activos)
  - Información de estado y planificación. Esta es la información que el SO necesita para analizar las funciones de planificación, entre estos elementos tenemos el estado del proceso, su prioridad, la información relativa a la planificación, es decir, cuanto tiempo ha estado esperando, la cantidad de tiempo que el proceso se ejecutó la ultima vez que estuvo corriendo… y el evento, en caso de haberlo, al que el proceso está esperando para poder continuar su ejecución.

  - Estructuración de datos. Un proceso puede estar enlazado con otro proceso en una cola, por ejemplo, en la cola de listos. O bien, un proceso puede una relación padre-hijo con otro proceso. O bien el PCB 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.

Como hemos visto, un bloque de control de proceso puede contener información estructural, incluyendo punteros que permiten enlazar bloques de control de proceso entre sí. De esta forma, las colas que se describieron en la sección anterior pueden implementarse como listas enlazadas de bloques de control de proceso.

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.
Página 135

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

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 sistema, modo control o modo núcleo. Este último término se refiere al núcleo del sistema operativo, que es la parte del sistema operativo que engloba las funciones más importantes del sistema.

Funciones típicas del núcleo del SO:

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

- Gestión de memoria
  - Reserva de espacio de 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 procesador sabe en qué modo está ejecutando un proceso debido a 1 bit que existe en la palabra de estado, este denota el modo en el que se está ejecutando. Cuando se realiza una llamada a un servicio del SO o cuando una interrupción dispara la ejecución de una rutina del SO, este modo cambia a modo núcleo, cuando este servicio finaliza, se vuelve a cambiar a modo usuario.

CREACIÓN DE PROCESOS

Una vez que el SO decida crear un proceso por la razón que sea 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.

2. **Reservar espacio para el 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).

3. **Inicialización del bloque de control de proceso.** La parte de identificación de proceso del BCP contiene el identificador del proceso así como otros posibles identificadores, tal como el indicador del proceso padre. 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.

4. **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.

5. **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

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. Esto puede ocurrir mediante interrupciones, traps o llamadas al sistema.

Tenemos dos tipos de interrupciones del sistema, unas se denominan simplemente interrupciones mientras que las otras se denominan traps. Las primeras se producen por cause 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. Dentro de las interrupciones normales, el control se transfiere inicialmente al manejador de interrupción, que realiza determinadas tareas internas y que posteriormente salta a una rutina del SO, encargada de cada uno de los tipos de interrupciones en particular. Algunos ejemplos son:

- **Interrupción de reloj.** El SO determina si el proceso en ejecución ha excedido o no la unidad máximo de tiempo de ejecución, denominada rodaja de tiempo. En dicho caso, el proceso se pasa al estado listo y se ejecuta otro proceso.

- **Interrupción de E/S.** El SO determina qué acción E/S ha ocurrido, si esta constituye un evento por el cual estaban esperando algunos procesos, los mueve a la cola de Listos (o a la de listos/suspendidos, dependiendo) y entonces el SO decide si continuar con la ejecución del proceso que estaba ejecutando o si lo cambia por alguno de la cola de Listos con mayor prioridad.

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

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.

Cambio de modo. Cuando se cambia de modo debido a una interrupción es necesario salvaguardar el contexto del proceso actual en el BCP del mismo. Este contexto constituye toda la información que se puede ver alterada por la ejecución de la rutina de interrupción y que se necesita para la

continuación posterior del proceso. Esto incluye el contador de programa, otros registros del procesador y la información de la pila.

Habitualmente, las operaciones de salvaguarda y de recuperación se realizan por hardware.

Página 140

Cambiar del estado del proceso. Obviamente, cambiar de modo es distinto a cambiar de proceso. Un cambio de modo puede ocurrir sin que cambie el estado del proceso actualmente ejecutándose. En dicho caso, la salvaguarda del estado y su posterior restauración comportan sólo una ligera sobrecarga. En cambio, si el proceso actualmente en estado Ejecutando va a cambiarse de estado entonces es necesario que el SO realice cambios en su entorno. Los pasos que se realizan para un cambio de estado completo son:

1. Salvar el estado del procesador, incluyendo el contador de programa y otros registros.
2. Actualizar el PCB del proceso que se estaba ejecutando, en el PCB hay que cambiar el estado de Ejecutando al que haya pasado además de información de porqué ha cambiado de estado así como información de auditoría.
3. Mover el PCB a la cola correspondiente (Listos, Bloqueado en el evento i…)
4. Selección de un nuevo proceso a ejecutar.
5. Actualizar el PCB del proceso elegido.
6. Actualizar las estructuras de datos de gestión de memoria.
7. Restaurar el estado del procesador al que tenía en el momento en el que dicho proceso que ha entrado dejo de estar en el estado Ejecutando, leyendo los valores anteriores del contador de programa y de los registros.

Por tanto, si el cambio de modo requiere un cambio de estado, necesita más esfuerzo que un cambio de modo a secas.

Página 139

EJECUCIÓN DEL SO

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 cuantas direcciones diferentes de diseño. Aquí vemos algunas de ellas:

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.

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.

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.

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, dependiendo del diseño del sistema. En algún momento, no obstante, el proceso actual se debe poner en un estado de no ejecución, designando a otro proceso como el siguiente a ejecutar. Durante esta fase, es más conveniente ver la ejecución como fuera de cualquiera de los procesos.
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. Como vemos, 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 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.

Concepto y tipos de Hebras (hilos) en Stalling pp. 158-172

Página 158

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.

- **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.

Estas dos características son independientes y podrían ser tratadas como tales por el sistema operativo. Para distinguir estas dos características, la unidad que se activa se suele denominar hilo (thread), mientras que la unidad de propiedad de recursos se suele denominar proceso o tarea.

MULTIHILO
Esto se refiere a la capacidad de un SO de dar soporte a múltiples hilos de ejecución en un solo proceso.

El enfoque tradicional de un solo hilo en ejecución por proceso se conoce como estrategia monohilo.

El MS-DOS soporta un único proceso y un hilo por proceso (monoprogramado y monohilo).

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 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.

En un entorno multihilo, sigue habiendo un único bloque de control del proceso y un espacio de direcciones de usuario asociado al proceso, pero ahora hay varias pilas separadas para cada hilo, así como un bloque de control para cada hilo que contiene los valores de los registros, la prioridad, y otra información relativa al estado del hilo.

De esta forma, 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.

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. 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.

Un ejemplo de una aplicación que podría hacer uso de hilos es un servidor de archivos. Cada vez que llega una nueva petición de archivo, el programa de gestión de archivos puede ejecutar un nuevo hilo. Ya que un servidor manejará muchas peticiones, se crearán y finalizarán muchos hilos en un corto periodo de tiempo.

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.

Aquí podemos ver 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.

- **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. 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, como por ejemplo la suspensión de un 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.

Sincronización de hilos. Todos los hilos de un proceso comparten el mismo espacio de direcciones y otros recursos, por tanto, es necesario sincronizar las actividades de los hilos para que no interfieran entre ellos o corrompan estructuras de datos.

HILOS DE NIVEL DE USUARIO Y DE NIVEL DE NÚCLEO

Existen dos amplias categorías de implementación de hilos: hilos de nivel de usuario (user-level threads, ULT) e hilos de nivel de núcleo (kernel-level threads, KLT).

Hilos de nivel de usuario. En un entorno ULT puro, la aplicación gestiona todo el trabajo de los hilos y el núcleo no es consciente de la existencia de los mismos. Cualquier aplicación puede programarse para ser multihilo a través del uso de una biblioteca de hilos, que es un paquete de rutinas para la gestión de ULT. La biblioteca de hilos contiene código para la creación y destrucción de hilos, para paso de mensajes y datos entre los hilos, para planificar la ejecución de los hilos, y para guardar y restaurar el contexto de los hilos.

Por defecto, una aplicación comienza con un solo hilo y ejecutando en ese hilo. Esta aplicación y su hilo se localizan en un solo proceso gestionado por el núcleo. En cualquier momento que la aplicación esté ejecutando (el proceso está en estado Ejecutando), la aplicación puede crear un nuevo hilo a ejecutar dentro del mismo proceso. La creación se realiza llamando a la utilidad de creación en la biblioteca de hilos. Se pasa el control a esta utilidad a través de una llamada a procedimiento. La biblioteca de hilos crea una estructura de datos para el nuevo hilo y luego pasa el control a uno de los hilos de ese proceso que esté en estado listo, utilizando algún algoritmo de planificación. Cuando se pasa el control a la biblioteca, se almacena el contexto del hilo actual, y cuando se pasa el control de la biblioteca al hilo, se recupera el contexto de ese hilo. El contexto tiene esencialmente el contenido de los registros del usuario, el contador que programa, y los punteros de pila.

Toda la actividad descrita en el párrafo anterior tiene lugar en el espacio de usuario y dentro de un solo proceso. El núcleo no es consciente de esta actividad. El núcleo continúa planificando el proceso como una unidad y asigna al proceso un único estado (Listo, Ejecutando, Bloqueado, etc.).

Todo esto implica que el estado de los hilos, al ser anónimo para el núcleo, ocurre aparte del estado del proceso. Si el núcleo bloque al proceso, sus hilos se seguirán viendo en estado Ejecutando (los que lo estuviesen haciendo) para la biblioteca de hilos, pese a que no estén corriendo como tal en el procesador. Una vez que el proceso se desbloquee, o vuelva a ejecutarse, los hilos seguirán su funcionamiento como lo estaban haciendo, todo controlado por la biblioteca de hilos. Asimismo, si cuando el proceso deja de estar ejecutándose, justo pilla a la biblioteca de hilos cambiando el estado de los hilos del proceso, no pasa nada, cuando el proceso vuelva a ejecutarse, la biblioteca de hilos completará esta operación.

El uso de ULT en lugar de KLT, presenta las siguientes ventajas:

1. El cambio de hilo no requiere privilegios de modo núcleo porque todas las estructuras de datos de gestión de hilos están en el espacio de direcciones de usuario de un solo proceso. Por consiguiente, el proceso no cambia a modo núcleo para realizar la gestión de hilos. Esto ahorra la sobrecarga de dos cambios de modo (usuario a núcleo; núcleo a usuario).

2. La planificación puede especificarse por parte de la aplicación. Una aplicación se puede beneficiar de un simple algoritmo de planificación cíclico, mientras que otra se podría beneficiar de un algoritmo de planificación basado en prioridades. El algoritmo de planificación se puede hacer a medida sin tocar el planificador del sistema operativo.

3. Los ULT pueden ejecutar en cualquier sistema operativo. No se necesita ningún cambio en el nuevo núcleo para dar soporte a los ULT. La biblioteca de los hilos es un conjunto de utilidades a nivel de aplicación que comparten todas las aplicaciones.

Hay dos desventajas de los ULT en comparación con los KLT:

1. En un sistema operativo típico muchas llamadas al sistema son bloqueantes. Como resultado, cuando un ULT realiza una llamada al sistema, no sólo se bloquea ese hilo, sino que se bloquean todos los hilos del proceso.

2. En una estrategia pura ULT, una aplicación multihilo no puede sacar ventaja del multiproceso. El núcleo asigna el proceso a un solo procesador al mismo tiempo. Por consiguiente, en un determinado momento sólo puede ejecutar un hilo del proceso. En efecto, tenemos multiprogramación a nivel de aplicación con un solo proceso.

Hay formas de afrontar estos dos problemas. Por ejemplo, ambos problemas pueden superarse escribiendo una aplicación de múltiples procesos en lugar de múltiples hilos. Pero este enfoque elimina la principal ventaja de los hilos: cada cambio es un cambio de proceso en lugar de un cambio de hilo, lo que genera una gran sobrecarga.

Otra forma de solucionar el problema de hilos que se bloquean es una técnica denominada jacketing (revestimiento). El objetivo de esta técnica es convertir una llamada al sistema bloqueante en una llamada al sistema no bloqueante. Por ejemplo, en lugar de llamar directamente a una rutina del sistema de E/S, un hilo puede llamar a una rutina jacket de E/S a nivel de aplicación. Con esta rutina jacket, el código verifica si el dispositivo de E/S está ocupado. Si lo está, el hilo entra en estado Bloqueado y pasa el control (a través de la biblioteca de hilos) a otro hilo. Cuando este hilo recupera de nuevo el control, chequea de nuevo el dispositivo de E/S.

Hilos a nivel de núcleo. En un entorno KLT puro, el núcleo gestiona todo el trabajo de gestión de hilos. No hay código de gestión de hilos en la aplicación, solamente una interfaz de programación de aplicación (API) para acceder a las utilidades de hilos del núcleo. Windows es un ejemplo de este enfoque.

Cualquier aplicación puede programarse para ser multihilo. Todos los hilos de una aplicación se mantienen en un solo proceso. El núcleo mantiene información de contexto del proceso como una entidad y de los hilos individuales del proceso. La planificación realizada por el núcleo se realiza a nivel de hilo. Este enfoque resuelve los dos principales inconvenientes del enfoque ULT. Primero, el núcleo puede planificar simultáneamente múltiples hilos de un solo proceso en múltiples procesadores. Segundo, si se bloquea un hilo de un proceso, el núcleo puede planificar otro hilo del mismo proceso. Otra ventaja del enfoque KLT es que las rutinas del núcleo pueden ser en sí mismas multihilo.

La principal desventaja del enfoque KLT en comparación con el enfoque ULT es que la transferencia de control de un hilo a otro del mismo proceso requiere un cambio de modo al núcleo.

De esta forma, mientras que hay una ganancia significativa entre el uso de multihilos KLT en comparación con procesos de un solo hilo, hay una ganancia significativa adicional por el uso de ULT. Sin embargo, depende de la naturaleza de la aplicación involucrada que nos podamos beneficiar o no de la ganancia adicional.

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

en un número (menor o igual) de KLT. El programador debe ajustar el número de KLT para una máquina y aplicación en particular para lograr los mejores resultados posibles.

En los enfoques combinados, múltiples hilos de la misma aplicación pueden ejecutar en paralelo en múltiples procesadores, y una llamada al sistema bloqueante no necesita bloquear el proceso completo. Si el sistema está bien diseñado, este enfoque debería combinar las ventajas de los enfoques puros ULT y KLT, minimizando las desventajas.

OTRAS CONFIGURACIONES

Aparte de la configuración Uno a Uno 1:1, que es en la que tenemos 1 proceso con 1 único hilo, y de la configuración Muchos a Uno M:1, que es en la que tenemos múltiples hilos para 1 proceso también existen otras configuraciones:

Relación muchos a muchos. La idea de tener una relación muchos-a-muchos entre hilos y procesos ha sido explorada en el sistema operativo experimental TRIX. En TRIX, existen los conceptos de dominio de hilo. Un dominio es una entidad estática, que consiste en un espacio de direcciones y «puertos» a través de los cuales se pueden enviar y recibir mensajes. Un hilo es una ruta de ejecución, con una pila de ejecución, estado del procesador e información de planificación.

Al igual que en el enfoque multihilo visto hasta el momento, múltiples hilos podrían ejecutar en un solo dominio, proporcionando las ventajas discutidas anteriormente. Sin embargo, también es posible realizar la actividad de usuario o ejecutar aplicaciones en múltiples dominios. En este caso hay un hilo que se puede mover entre dominios.

El uso de un solo hilo en múltiples dominios está motivado por el deseo de proporcionar herramientas de estructuración al programador. Por ejemplo, considere un programa que hace uso de un subprograma de E/S. En un entorno multiprogramado que permite procesos creados por los usuarios, el programa principal podría generar un nuevo proceso para el manejo de la E/S y continuar ejecutando. Sin embargo, si el futuro progreso del programa principal depende del funcionamiento de la E/S, entonces el programa principal tendrá que esperar a la finalización del otro programa de E/S.

Relación uno-a-muchos. En el campo de los sistemas operativos distribuidos (diseñados para controlar sistemas de computadores distribuidos), ha habido interés en el concepto de un hilo como una entidad que se puede mover entre espacios de direcciones. Un ejemplo importante de esta investigación es el sistema operativo Clouds, y especialmente su núcleo, conocido como Ra.

Desde el punto de vista del usuario, un hilo en Clouds es una unidad de actividad. Un proceso es un espacio de direcciones virtual con un bloque de control de proceso asociado. Una vez creado, un hilo comienza ejecutando en un proceso a través de la invocación de un punto de entrada de un programa en ese proceso. Los hilos se pueden mover de un espacio de direcciones a otro, incluso fuera de los límites de la máquina (es decir, moverse de un computador a otro). Según se mueve el hilo, debe llevarse determinada información con él, tal como el controlador de terminal, los parámetros globales y las guías de planificación (por ejemplo, prioridad).

El enfoque de Clouds proporciona una forma eficiente de aislar a los usuarios y programadores de los detalles del entorno distribuido. La actividad del usuario puede ser representada como un solo hilo, y el movimiento de ese hilo entre máquinas puede ser gestionado por el sistema operativo gracias a información relativa al sistema, tal como la necesidad de acceder a un recurso remoto o el

equilibrado de carga.

Planificación de procesos. Tipos de planificadores de recursos, Criterios de planificación y Políticas de planificación del procesador: Stalling pp.402-421

Página 402

El objetivo de la planificación de procesos es asignar procesos a ser ejecutados por el procesador o procesadores a lo largo del tiempo, de forma que se cumplan los objetivos del sistema tales como el tiempo de respuesta, el rendimiento y la eficiencia del procesador. En muchos sistemas, esta actividad de planificación se divide en tres funciones independientes: planificación a largo-, medio-, y corto-plazo. Los nombres sugieren las escalas de tiempo relativas con que se ejecutan estas funciones.

La planificación a largo plazo se encarga de añadir un proceso al conjunto de procesos para ser ejecutados, a medio plazo se encarga de añadir un proceso al número de procesos que están parcialmente o totalmente en memoria principal y la a corto plazo se encarga de decidir qué procesos se ejecutan. En cambio, la planificación de la E/S se encarga de decidir si un proceso que está pendiente de una petición E/S es atendido por un dispositivo E/S disponible.

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.

Los procesos limitados por el procesador son aquellos que realizan mucho trabajo computacional (se ejecutan mucho tiempo) y ocasionalmente usan los dispositivos E/S. En cambio, los limitados por E/S son aquellos que pasan más tiempo utilizando los dispositivos E/S que el procesador.

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. 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 frecuentemente 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 E/S
- Llamadas al sistemas
- Señales (por ejemplo, semáforos)

CRITERIOS DE LA 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. 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.

Estos criterios de 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 glos distintos requisitos dependerán de la naturaleza y el uso dado al sistema.

En resumen:

Orientados al usuario relacionados con las prestaciones:

- **Tiempo de estancia:** Tiempo transcurrido desde que se lanza un proceso hasta que finaliza.

- **Tiempo de respuesta:** 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.

- **Fecha tope:** 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, no relacionados con las prestaciones:

- **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.

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.

- **Utilización del procesador.** Es el porcentaje de tiempo que el procesador está ocupado.

Orientados al sistema, no relacionados con las prestaciones:

- **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.

Página 408

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.

En este caso, suponiendo que tengamos una cola de listos por cada prioridad, el activador comenzaría revisando la cola 0, la de menor prioridad, si hay un proceso, lo pone a ejecutar, si hay más de 1, aplicará algún tipo de planificación para decidir uno de ellos, y si no hay ninguno, pasará a la cola 1 y así.

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.

POLÍTICAS DE PLANIFICACIÓN ALTERNATIVAS

La función de selección determina qué proceso, entre los procesos listos, se selecciona para su ejecución. La función puede estar basada en prioridades, requisitos sobre los recursos, o las características de ejecución del proceso. En el último caso, hay tres medidas cuantitativas:

- w = tiempo usado en el sistema hasta este momento, esperando o ejecutando
- e = tiempo usado en ejecución hasta el momento
- s = tiempo total de servicio requerido por el proceso, incluyendo e; generalmente, esta cantidad debe ser estimada o proporcionada por el usuario

Por ejemplo, la función de selección max[w] implica una planificación del tipo primero en llegar, primero en servirse (first-come-first-served – FCFS).

El modo de decisión especifica los instantes de tiempo en que se ejecuta la función de selección. Hay dos categorías generales:

- **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.

Primero en llegar, primero en servirse (first-come-first-served). 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. Esto es porque si llega primero un proceso largo y después uno corto, el corto tendrá que comerse todo el tiempo que tarde el largo, de forma que su estancia en el sistema será mucho más larga de lo normal.

Otro problema de FCFS es que tiende a favorecer procesos limitados por el procesador sobre los procesos limitados por la E/S.

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.

Con la planificación en turno rotatorio, el tema clave de diseño es la longitud del quantum de tiempo a ser utilizado. 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.

En este caso también se trata de forma desigual a los procesos limitados por la CPU respecto a los limitados por la E/S, puesto que los limitados por la CPU serán capaces de agotar su quantum completo, mientras que los limitados por la E/S se ejecutarán menos tiempo, se bloquearán por algún suceso E/S y tendrán que volver a ponerse a la cola.

Primero el proceso más corto (shortest process next). Otro 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.

El rendimiento global se mejora significativamente en términos del tiempo de respuesta. Sin embargo, se incrementa la variabilidad de los tiempos de respuesta, especialmente para los procesos más largos, y de esta forma se reduce la predecibilidad.

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.

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.

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.

Retroalimentación (feedback). 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. 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.

Un problema que presenta el esquema simple antes contado es 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.

Planificación de procesos. Tipos de planificadores de recursos, Criterios de planificación y Políticas de planificación del procesador: Carretero pp.154-188

Página 157

En este libro nos introduce el concepto de afinidad estricta, es decir, un proceso puede establecer qué ejemplares de qué recursos (procesadores, por ejemplo) son por los que tiene afinidad, es decir, son los que quiere/necesita usar, de forma que nunca le serán asignados unos que no estén en esa lista.

También se introduce el concepto de afinidad natural, esta no viene especificada en ningún lado, pero se refiere a que si un ejemplar de un recurso ha tratado un proceso (pongamos el ejemplo de un procesador), es preferible que cuando el proceso se bloquee, vuelva a la cola de Listos y vuelva a ejecutarse, lo vuelva a hacer en el mismo procesador para aprovechar la memoria cache de este.

Con recursos también nos referimos a la memoria, el disco y los mecanismos de sincronización.

Página 158

Perfil de uso del procesador

La ejecución de un programa puede verse como una secuencia en la que se va alternando una fase de uso del procesador (ráfaga de procesador) con una fase de bloqueo asociada al uso de un recurso (llamada ráfaga de entrada/salida puesto a menudo, los procesos se bloquean por esta causa).

Página 160

Grado de interactividad

Este grado denota el nivel de interacción directa que existe entre el usuario de la aplicación y la misma. Aquí podemos nombrar los procesos batch, que son procesos que no tienen interacción directa con el usuario.

Notar que un programa con un alto grado de interactividad va a denotar también un alto grado de uso de la E/S.

Los procesos con alto grado de interactividad se ven beneficiados cuando se le da prioridad a los procesos con ráfagas cortas.

Página 160

Puntos de activación del planificador

Lo nuevo es que cuando se produce un cambio de contexto involuntario, como sabemos, se llamará al planificador (cuando sea voluntario también), pues bueno, en este caso, diferenciamos dos casos:

1. Cuando tenemos un núcleo no expulsable, en este caso, habrá que esperar a que se completen las interrupciones y las llamadas al sistema si las había para llamar al planificar y para realizar el cambio de contexto.

2. En cambio, si tenemos un núcleo expulsable, tendremos que esperar a que se completen las interrupciones pero no las llamadas al sistema.

Planificación en Sistemas Multiprocesadores: Stalling pp. 454-462

Página 454

En un multiprocesador la planificación involucra tres aspectos interrelacionados:

- La asignación de procesos a procesadores.
- El uso de la multiprogramación en cada procesador individual.
- La activación del proceso, propiamente dicha.

Asignación de procesos a procesadores. Si se asume que la arquitectura del multiprocesador es uniforme, en el sentido de que ningún procesador tiene una ventaja física particular con respecto al acceso a memoria principal o a dispositivos de E/S, entonces el enfoque más simple de la planificación consiste en tratar cada proceso como un recurso colectivo y asignar procesos a procesadores por demanda. Surge la cuestión de si la asignación debería ser estática o dinámica.

Si un proceso se vincula permanentemente a un procesador desde su activación hasta que concluye, entonces se mantiene una cola a corto plazo dedicada por cada procesador. Una ventaja de esta estrategia es que puede haber menos sobrecarga en la función de planificación, dado que la asignación a un procesador se realiza una vez y para siempre.

Una desventaja de la asignación estática es que un procesador puede estar ocioso, con su cola vacía, mientras otro procesador tiene trabajo acumulado. Para evitar esta situación, puede utilizarse una cola común. Todos los procesos van a una cola global y son planificados sobre cualquier procesador disponible. Así, a lo largo de la vida de un proceso, puede ser ejecutado en diferentes procesadores en diferentes momentos. En una arquitectura de memoria compartida fuertemente acoplada, la información de contexto de todos los procesos estará disponible para todos los procesadores, y por lo tanto el coste de planificación de un proceso será independiente de la identidad del procesador sobre cual se planifica. Otra opción más, es el balance dinámico de carga, en el que los hilos se mueven de una cola de un procesador a la cola de otro procesador; Linux utiliza este enfoque.

Independientemente de cómo los procesos se vinculan a los procesadores, se necesita alguna manera de asignar procesos a procesadores. Se han venido usando dos enfoques: maestro/esclavo y camaradas. Con la arquitectura maestro/esclavo, ciertas funciones clave del núcleo del sistema operativo ejecutan siempre en un procesador concreto. Los otros procesadores sólo pueden ejecutar programas de usuario. El maestro es responsable de la planificación de trabajos. Una vez que el proceso está activo, si el esclavo necesita servicio (por ejemplo, una llamada de E/S), debe enviar una solicitud al maestro y esperar a que el servicio se realice. Este enfoque es bastante simple y sólo requiere mejorar un poco un sistema operativo monoprocesador multiprogramado. La resolución de conflictos se simplifica porque un procesador tiene todo el control de la memoria y de los recursos de E/S. Hay dos desventajas en este enfoque: (1) un fallo en el maestro hace que falle el sistema completo, y (2) el maestro puede llegar a ser un cuello de botella para el rendimiento del sistema.

En la arquitectura camaradas, el núcleo puede ser ejecutado en cualquier procesador, y cada procesador se auto-planifica desde la colección de procesos disponibles. Este enfoque complica el sistema operativo. El sistema operativo debe asegurar que dos procesadores no escogen el mismo proceso y que los procesos no se extravían por ninguna razón. Deben emplearse técnicas para

resolver y sincronizar la competencia en la demanda de recursos.

El uso de la multiprogramación en procesadores individuales. Cuando cada proceso se asocia estáticamente a un procesador para todo su tiempo de vida, surge una nueva pregunta: ¿debería ese procesador ser multiprogramado?

Cuando hay muchos procesadores disponibles, conseguir que cada procesador esté ocupado tanto tiempo como sea posible deja de ser lo más importante. En cambio, la preocupación es proporcionar el mejor rendimiento, medio, de las aplicaciones. Una aplicación que consista en varios hilos, puede ejecutar con dificultad a menos que todos sus hilos estén dispuestos a ejecutar simultáneamente.

Activación de procesos. El último aspecto de diseño relacionado con la planificación multiprocesador, es la elección real del proceso a ejecutar. Hemos visto que en un monoprocesador multiprogramado, el uso de prioridades o de sofisticados algoritmos de planificación basados en el uso pasado, pueden mejorar el rendimiento frente a la ingenua estrategia FCFS (primero en llegar, primero en serservido). Cuando consideramos multiprocesadores, estas complejidades pueden ser innecesarias o incluso contraproducentes, y un enfoque más simple puede ser más eficaz con menos sobrecarga. En el caso de la planificación de hilos, entran en juego nuevos aspectos que pueden ser más importantes que las prioridades o las historias de ejecución. Tratemos cada uno de estos temas por turno.

PLANIFICACIÓN DE PROCESOS

En los sistemas multiprocesador más tradicionales, los procesos no se vinculan a los procesadores. En cambio hay una única cola para todos los procesadores o, si se utiliza algún tipo de esquema basado en prioridades, hay múltiples colas basadas en prioridad, alimentando a un único colectivo de procesadores. En cualquier caso, el sistema puede verse como una arquitectura de colas multiservidor.

Nótese que la diferencia entre los algoritmos de planificación es mucho menor en el caso del biprocesador. Con dos procesadores, un proceso único con un tiempo de servicio largo es mucho menos disruptivo en el caso FCFS; otros procesos pueden utilizar el otro procesador.

La conclusión general es que la disciplina de planificación específica es mucho menos importante con dos procesadores que con uno. Debería ser evidente que esta conclusión es incluso más fuerte a medida que el número de procesadores crece. Así, la disciplina básica FCFS o el uso de FCFS con un esquema estático de prioridades puede ser suficiente en un sistema multiprocesador.

PLANIFICACIÓN DE HILOS

El poder completo de los hilos se vuelve evidente en un sistema multiprocesador. En este entorno, los hilos pueden explotar paralelismo real dentro de una aplicación. Si los hilos de una aplicación están ejecutando simultáneamente en procesadores separados, es posible una mejora drástica de sus prestaciones. Sin embargo, puede demostrarse que para aplicaciones que necesitan una interacción significativa entre hilos (paralelismo de grano medio), pequeñas diferencias en la gestión y planificación de hilos pueden dar lugar a un impacto significativo en las prestaciones.

Entre las muchas propuestas para la planificación multiprocesador de hilos y la asignación a procesadores, destacan cuatro enfoques generales:

- **Compartición de carga.** Los procesos no se asignan a un procesador particular. Se mantiene una cola global de hilos listos, y cada procesador, cuando está ocioso, selecciona un hilo de la cola.

- **Planificación en pandilla.** Un conjunto de hilos relacionados que se planifica para ejecutar sobre un conjunto de procesadores al mismo tiempo, en una relación uno-a-uno.

- **Asignación de procesador dedicado.** Esto es lo opuesto al enfoque de compartición de carga y proporciona una planificación implícita definida por la asignación de hilos a procesadores. Cada proceso ocupa un número de procesadores igual al número de hilos en el programa, durante toda la ejecución del programa. Cuando el programa termina, los procesadores regresan al parque general para la posible asignación a otro programa.

- **Planificación dinámica.** El número de hilos de un proceso puede cambiar durante el curso de su ejecución.

Compartición de carga. La compartición de carga es posiblemente el enfoque más simple y que se surge más directamente de un entorno monoprocesador. Tiene algunas ventajas:

- La carga se distribuye uniformemente entre los procesadores, asegurando que un procesador no queda ocioso mientras haya trabajo pendiente.

- No se precisa un planificador centralizado; cuando un procesador queda disponible, la rutina de planificación del sistema operativo se ejecuta en dicho procesador para seleccionar el siguiente hilo.

- La cola global puede organizarse y ser accesible usando cualquiera de los esquemas expuestos en el Capítulo 9, incluyendo esquemas basados en prioridad y esquemas que consideran la historia de ejecución o anticipan demandas de procesamiento.

Aquí analizamos tres versiones diferentes de compartición de carga:

- **Primero en llegar, primero en ser servido (FCFS).** Cuando llega un trabajo, cada uno de sus hilos se disponen consecutivamente al final de la cola compartida. Cuando un procesador pasa a estar ocioso, coge el siguiente hilo listo, que ejecuta hasta que se completa o se bloquea.

- **Menor número de hilos primero.** La cola compartida de listos se organiza como una cola de prioridad, con la mayor prioridad para los hilos de los trabajos con el menor número de hilos no planificados. Los trabajos con igual prioridad se ordenan de acuerdo con qué trabajo llega primero. Al igual que con FCFS, el hilo planificado ejecuta hasta que se completa o se bloquea.

- **Menor número de hilos primero con expulsión.** Se le da mayor prioridad a los trabajos con el menor número de hilos no planificados. Si llega un trabajo con menor número de hilos que un trabajo en ejecución se expulsarán los hilos pertenecientes al trabajo planificado.

La compartición de carga tiene varias desventajas:

- La cola central ocupa una región de memoria a la que debe accederse de manera que se cumpla la exclusión mutua. De manera que puede convertirse en un cuello de botella si muchos procesadores buscan trabajo al mismo tiempo. Cuando hay sólo un pequeño número de procesadores, es difícil que esto se convierta en un problema apreciable. Sin embargo, cuando el multiprocesador consiste en docenas o quizás cientos de procesadores, la posibilidad de que esto sea un cuello de botella es real.

- Es poco probable que los hilos expulsados retomen su ejecución en el mismo procesador. Si cada procesador está equipado con una cache local, ésta se volverá menos eficaz.

- Si todos los hilos se tratan como un conjunto común de hilos, es poco probable que todos los hilos de un programa ganen acceso a procesadores a la vez. Si se necesita un alto grado de coordinación entre los hilos de un programa, los cambios de proceso necesarios pueden comprometer seriamente el rendimiento.

A pesar de las potenciales desventajas, este es uno de los esquemas más utilizados en los multiprocesadores actuales.

Planificación en pandilla. El concepto de planificar un conjunto de procesos simultáneamente sobre un conjunto de procesadores es anterior al uso de hilos. Vemos los siguientes beneficios:

- Si se ejecutan en paralelo procesos estrechamente relacionados, puede reducirse el bloqueo por sincronización, pueden necesitarse menos cambios de proceso y las prestaciones aumentarán.

- La sobrecarga de planificación puede reducirse dado que una decisión única afecta a varios procesadores y procesos a la vez.

El término planificación en pandilla se ha aplicado a la planificación simultánea de los hilos que componen un proceso individual. La planificación en pandilla es para aplicaciones paralelas de grano medio o fino cuyo rendimiento se ve degradado de forma importante cuando cualquier parte de la aplicación no está ejecutando mientras otras partes están listas para ejecutar. También es beneficiosa para cualquier aplicación paralela incluso para aquellas de rendimiento no tan sensible. La necesidad de planificación en pandilla está ampliamente reconocida, y existen implementaciones en variedad de sistemas operativos multiprocesador.

Una manera obvia en que la planificación en pandilla mejora las prestaciones de una aplicación individual es porque se minimizan los cambios de proceso. Suponga que un hilo de un proceso está ejecutando y alcanza un punto en el que debe sincronizarse con otro hilo del mismo proceso. Si este otro hilo no está ejecutando, pero está listo para ejecutar, el primer hilo quedará colgado hasta que suceda un cambio de proceso en otro procesador que tome el hilo necesario. En una aplicación con coordinación estrecha entre sus hilos, estos cambios reducirán dramáticamente el rendimiento. La planificación simultánea de hilos cooperantes puede también reducir el tiempo de ubicación de recursos. Por ejemplo, múltiples hilos planificados en pandilla pueden acceder a un fichero sin la sobrecarga adicional de bloquearse durante una operación de posicionamiento y lectura/escritura.

Asignación de procesador dedicado. Una forma extrema de la planificación en pandilla es dedicar un grupo de procesadores a una aplicación durante toda la duración de la aplicación. Esto es,

cuando se planifica una aplicación dada, cada uno de sus hilos se asigna a un procesador que permanece dedicado al hilo hasta que la aplicación concluye.

Este enfoque puede parecer un desperdicio extremo del tiempo de procesador. Si un hilo de una aplicación se bloquea esperando por E/S o por sincronización con otro hilo, entonces el procesador de ese hilo se queda ocioso: no hay multiprogramación de los procesadores. Pueden realizarse dos observaciones en defensa de esta estrategia:

1. En un sistema altamente paralelo, con decenas o cientos de procesadores, cada uno de los cuales representa una pequeña fracción del coste del sistema, la utilización del procesador deja de ser tan importante como medida de la eficacia o rendimiento.

2. Evitar totalmente el cambio de proceso durante la vida de un programa debe llevar a una sustancial mejora de velocidad de ese programa.

Claramente, no todas las aplicaciones pueden ser estructuradas de este modo, pero muchos problemas numéricos y otras aplicaciones varias sí pueden tratarse de esta forma.

Tanto la asignación de procesador dedicado como la planificación en pandilla atacan el problema de la planificación tratando el aspecto de la ubicación del procesador. Puede observarse que el problema de la ubicación del procesador en un multiprocesador se parece más al problema de la ubicación de la memoria en un monoprocesador que al problema de la planificación en un monoprocesador.

Planificación dinámica. Para algunas aplicaciones, es posible proporcionar, en el lenguaje o en el sistema, herramientas que permitan que el número de hilos del proceso pueda ser alterado dinámicamente. Esto permitiría que el sistema operativo ajustase la carga para mejorar la utilización.

El sistema operativo es el responsable de particionar los procesadores entre los trabajos. Cada trabajo utiliza los procesadores actualmente en su partición para ejecutar algún subconjunto de sus tareas ejecutables proyectando estas tareas sobre hilos. La decisión apropiada acerca del subconjunto a ejecutar, así como qué hilos suspender cuando el proceso sea expulsado, se le deja a las aplicaciones individuales (quizás a través de un conjunto de rutinas de biblioteca).

En este enfoque, la responsabilidad de planificación del sistema operativo está limitada fundamentalmente a la ubicación del procesador y se realiza de acuerdo a la siguiente política. Cuando un trabajo solicita uno o más procesadores (bien cuando el trabajo llega por primera vez o porque sus requisitos cambian),

1. Si hay procesadores ociosos, utilizarlos para satisfacer la solicitud.

2. En otro caso, si el trabajo que realiza la solicitud acaba de llegar, ubicarlo en un único procesador quitándoselo a cualquier trabajo que actualmente tenga más de un procesador.

3. Si no puede satisfacerse cualquier parte de la solicitud, mantenerla pendiente hasta que un procesador pase a estar disponible, o el trabajo rescinda la solicitud (por ejemplo, si dejan de ser necesarios los procesadores extra).

Cuando se libere uno o más procesadores (incluyendo la terminación de un trabajo),

4. Examinar la cola actual de solicitudes de procesador no satisfechas. Asignar un único procesador a cada trabajo en la lista que no tenga actualmente procesadores (por ejemplo, a todos los recién llegados en espera). Luego volver a examinar la lista, volviendo a asignar el resto de los procesadores siguiendo la estrategia FCFS.

Planificación para Sistemas de Tiempo Real -- Stalling pp. 466-468

En un compendio de algoritmos de planificación de tiempo real, se observa que los distintos enfoques de la planificación dependen de cuando el sistema realiza análisis de planificabilidad; y si lo hace, de si se realiza estática o dinámicamente; y de si el resultado del análisis
produce un plan de planificación de acuerdo al cual se desarrollarán las tareas en tiempo de ejecución. En base a estas consideraciones los autores identifican las siguientes clases de algoritmos:

- **Enfoques estáticos dirigidos por tabla.** En éstos se realiza un análisis estático de la factibilidad de la planificación. El resultado del análisis es una planificación que determina cuando, en tiempo de ejecución, debe comenzar a ejecutarse cada tarea.

- **Enfoques estáticos expulsivos dirigidos por prioridad.** También se realiza un análisis estático, pero no se obtiene una planificación. En cambio, el análisis se utiliza para asignar prioridades a las tareas, y así puede utilizarse un planificador expulsivo tradicional basado en prioridades.

- **Enfoques dinámicos basados en un plan.** La factibilidad se determina en tiempo de ejecución (dinámicamente) en vez de antes de comenzar la ejecución (estáticamente). Una nueva tarea será aceptada como ejecutable sólo si es posible satisfacer sus restricciones de tiempo. Uno de los resultados del análisis de factibilidad es un plan que se usará para decidir cuándo poner en marcha la tarea.

- **Enfoques dinámicos de mejor esfuerzo.** No se realiza análisis de factibilidad. El sistema intenta cumplir todos los plazos y aborta la ejecución de cualquier proceso cuyo plazo haya fallado.

La planificación estática dirigida por tabla es aplicable a tareas que son periódicas. Los datos de entrada para el análisis son: tiempo periódico de llegada, tiempo de ejecución, plazo periódico de finalización, y prioridad relativa de cada tarea. El planificador intenta encontrar un plan que le permita cumplir todos los requisitos de todas las tareas periódicas. Éste es un enfoque predecible pero no flexible, dado que un cambio en cualquiera de los requisitos de las tareas requiere rehacer toda la planificación.

La planificación estática con expulsión dirigida por prioridad hace uso del mecanismo de planificación expulsivo dirigido por prioridades común a la mayoría de los sistemas multiprogramados que no son de tiempo real. En un sistema que no es de tiempo real, pueden

utilizarse en múltiples factores para determinar la prioridad. Por ejemplo, en un sistema de tiempo compartido, la prioridad de un proceso puede cambiar dependiendo de qué consume más, CPU o E/S. En un sistema de tiempo real, la asignación de prioridades está relacionada con las restricciones de tiempo asociadas a cada tarea.

Con la planificación dinámica basada en un plan, cuando llega una nueva tarea, pero antes de que comience su ejecución, se intentará crear un plan que contenga las tareas previamente planificadas así como la nueva. Si la tarea recién llegada puede ser planificada de manera que se cumplan sus plazos sin que ninguna otra tarea planificada anteriormente pierda un plazo, la nueva tarea será aceptada poniéndose en marcha el nuevo plan de planificación.

La planificación dinámica de mejor esfuerzo es en enfoque utilizado en muchos sistemas operativos de tiempo real disponibles comercialmente hoy en día. Cuando llega una tarea, el sistema le asigna una prioridad basada en las características de la misma. De forma característica se utiliza algún tipo de planificación basada en plazos como la planificación del plazo más cercano. Así, las tareas no son periódicas y por tanto no es posible realizar un análisis estático de planificabilidad. Con este tipo de planificación, no sabremos si una determinada restricción de tiempo será satisfecha hasta que venza su plazo o la tarea se complete. Esta es la principal desventaja de esta forma de planificación. La ventaja es que es fácil de implementar.

Problema de la inversión de prioridad -- Stalling pp. 474-477

Página 474

La inversión de prioridad es un fenómeno que puede suceder en cualquier esquema de planificación expulsivo basado en prioridades, pero es particularmente relevante en el contexto de la planificación de tiempo real.

En cualquier esquema de planificación por prioridad, el sistema debe siempre estar ejecutando la tarea de mayor prioridad. La inversión de prioridad sucede cuando las circunstancias dentro de sistema fuerzan a una tarea de mayor prioridad a esperar por una tarea de menor prioridad. Un ejemplo sencillo de inversión de prioridad sucede si una tarea de menor prioridad ha bloqueado un recurso (un dispositivo o un semáforo binario) y una tarea de mayor prioridad intenta bloquear el mismo recurso. La tarea de mayor prioridad pasará a estado bloqueado hasta que el recurso esté disponible. Si la tarea de menor prioridad termina pronto con el recurso y lo libera, la tarea de mayor prioridad puede ser retomada rápidamente y es posible que no se violen restricciones de tiempo real.

Una situación más seria, es la conocida como inversión de prioridad ilimitada, en la cual la duración de la inversión de prioridad depende no sólo del tiempo necesario para conseguir el recurso compartido sino también de las acciones impredecibles de otras tareas no relacionadas.

En sistemas prácticos, se utilizan dos enfoques alternativos para evitar la inversión de prioridad ilimitada: el protocolo de herencia de prioridad y el protocolo de techo de prioridad. La idea básica de la herencia de prioridad es que una tarea de menor prioridad heredará la prioridad de cualquier tarea de mayor prioridad pendiente de un recurso que compartan. Este cambio de prioridad sucede tan pronto la tarea de mayor prioridad se bloquea en el recurso; y debe finalizar cuando la tarea de menor prioridad libera el recurso.

En el enfoque de techo de prioridad, se asocia una prioridad con cada recurso. La prioridad asociada con un recurso es un nivel más alta que la prioridad de su usuario más prioritario. Entonces, el planificador asigna esta prioridad a cualquier tarea que acceda al recurso. Cuando la tarea termina de usar el recurso, su prioridad se restablece al valor normal.

Diapositivas de Linux

Diapositiva 56

Un proceso en Linux se crea con las órdenes fork y clone (menos el proceso 0).

El proceso 0 está creado “a mano” al arrancar el sistema y es el responsable de crear el proceso 1 (init).

El proceso init es el antecesor de cualquier proceso del sistema.

Diapositiva 58

El kernel almacena la lista de procesos como una lista circular doblemente enlazada.

Cada elemento es un descriptor de proceso (PCB).

Diapositiva 61

En linux, a la hora de ver el modelo de estados, puesto que existen las señales, hay que tener en cuenta 2 estados más:

Ahora no tenemos solo el estado bloqueado, sino que tenemos el bloqueado interrumpible y el bloqueado no interrumpible. El interrumpible puede salir del estado de bloqueo o bien cuando ocurra el suceso por el que está bloqueado, o bien cuando le llegue una señal, mientras que desde el estado de bloqueado no interrumpible únicamente puede desbloquearse si ocurre el suceso por el que se ha bloqueado.

Adicionalmente tenemos el estado Parado. A este estado se entra desde el estado Ejecutando cuando otro proceso le envía una señal a este para que se detenga, ya sea para depurarlo, o para tracearlo. Solo podrá salir del estado Parado cuando le llegue una señal para ello.

Diapositiva 64

Cada task_struct tiene un puntero a la task_struct de su padre (llamada parent), a una lista de hijos (llamada children) y a una lista de sus hermanos (llamada sibling).

Diapositiva 65

Si un proceso hace un fork, los procesos creados serán sus hijos, y si hace más de un fork, entre ellos serán hermanos.

Todos los procesos son descendientes del proceso init (con PID 1).

Diapositiva 66

Desde el punto de vista del kernel, no hay distinción entre hebra y proceso. Cada hebra tiene su propio task_struct.

La llamada al sistema clone crea un nuevo proceso o hebra.

Diapositiva 68

A veces es útil que el kernel realice operaciones en segundo plano, para lo cual se crean hebras kernel. Estas no tienen un espacio de direcciones (su puntero mm es NULL).

Se ejecutan únicamente en espacio de kernel y se crean por el kernel al levantar al sistema, mediante la orden clone().

Terminan cuando realizan una operación do_exit() o cuando otra parte del kernel provoca su finalización.

Diapositiva 69

En esta diapositiva y en la siguiente se explica el proceso de formación de un nuevo proceso:

1. Se crea la estructura thread_info (pila kernel) y la task_struct para el nuevo proceso con los valores de la tarea actual.
2. Los elementos de task_struct que deban tener valores distintos a los del padre se inicializan con los valores correctos.
3. Se establece el estado del hijo a TASK_UNINTERRUPTIBLE mientras se realizan las acciones restantes.
4. Se establecen los valores adecuados para los flags task_struct del hijo.
5. Se llama a alloc_pid() para reservar un PID para el proceso.
6. Según cuales sean los flags pasados a clone, duplica o comparte recursos.
7. Se establece el estado del hijo a TASK_RUNNING.
8. Finalmente copy_proccess() termina devolviendo un puntero a la task_struct del hijo.

Diapositiva 71

Un proceso puede terminar o bien porque el programador haya incluido la llamada exit() en el programa, o bien porque el programa alcance el final de su main(), por lo que se llama a exit() de forma implícita o bien porque reciba una señal diciéndole que termine.

El trabajo de liberación lo hace la función do_exit().

Diapositiva 72

Actuación de do_exit():

1. Activa el flag PF_EXITING de task_struct.
2. Para cada recurso que esté utilizando este proceso se decrementa el contador

correspondiente que indica el n.º de procesos que lo están utilizando. Si el contador vale 0, se realiza la operación de destrucción oportuna sobre el recurso, por ejemplo, si fuese una zona de memoria, se liberaría.
3. El valor que se pasa como argumento a exit() se almacena en el campo exit_code() de task_struct (información de terminación del padre).
4. Se manda una señal al padre indicando la terminación de su hijo
5. Si aún tiene hijos, se pone como padre de estos al proceso init (PID = 1), o, dependiendo de las características del grupo de procesos al que pertenezca el proceso, podría ponerse como padre a otro miembro de ese grupo de procesos.
6. Se establece el campo exit_state de task_struct a EXIT_ZOMBIE (es decir, que se mantenga la entrada de la tabla de procesos hasta que el padre haga un wait (EXIT_DEAD)
7. Se llama a schedule() para que el planificador elija un nuevo proceso a ejecutar.

Puesto que esto es lo último que ejecuta un proceso, do_exit() nunca retorna.

Diapositiva 74

En Linux tenemos un planificador modular con distintas clases de planificación:

1. Planificación de tiempo real
2. Planificación neutra o limpia
3. Planificación de la tarea idle (no hay trabajo que realizar)

Diapositiva 75

En base a estos, vemos que cada clase de planificación tiene una prioridad y que el algoritmo de planificación entre estas clases es por prioridades apropiativo.

Cada clase de planificación usa 1 o varias políticas para gestionar sus procesos.

La planificación, además, no opera únicamente sobre el concepto de proceso, sino que maneja conceptos más amplios en el sentido de manejar grupos de procesos: Entidad de planificación

Una entidad de planificación se representa mediante una instancia de la estructura sche_entity.

Diapositiva 76

Políticas manejadas por el planificador CFS:

- SCHED_NORMAL: Se aplica a los procesos normales de tiempo compartido
- SCHED_BATCH: Tareas menos importantes, menor prioridad.
- SCHED_IDLE: Tareas de este tipo tienen una prioridad mínima.

Políticas manejadas por el planificador de tiempo real:

- SCHED_RR: Round Robin
- SCHED_FIFO: FCFS

Diapositiva 77

El proceso en ejecución es siempre el más prioritario.

Las prioridades para los procesos de tiempo real van desde el 0 al 99, mientras que para los procesos normales van desde el 100 al 139.

De esta forma, cuando miramos el valor de nice, si este está a 0, significará que la prioridad es realmente 120, este valor se puede modificar, dandole como mínimo una prioridad de +19 (139), o como mucho (con permisos de administrador), de -20 (100).

Diapositiva 78

Tenemos 2 planificadores, el periódico y principal. El periódico se ejecuta automáticamente para actualizar estadísticas del kernel y para activar el método de planificación periódico de la clase de planificación a la que corresponde el proceso actual.

Si hay que replanificar, el planificador de la clase en cuestión activará el flag TIF_NEED_RESCHED asociado al thread_info y provocará que se llame al planificador principal.

Este se implementa en la función schedule, la cual se invoca de forma implícita cuando un proceso se bloquea o termina o si el kernel ve que el flag mencionado anteriormente está activado.

Diapositiva 80

Actuación del schedule:

1. Determina la actual runqueue y establece el puntero prev a la task_struct del proceso actual

2. Actualiza estadísticas y limpia el flag TIF_NEED_RESCHED

3. Si el proceso actual estaba en un estado TASK_INTERRUPTIBLE y ha recibido la señal que esperaba, se establece su estado a TASK_RUNNING

4. Se llama a pick_next_task de la clase de planificación a la que pertenezca el proceso actual para que se seleccione el siguiente proceso a ejecutar, se establece next con el puntero a la task_struct de dicho proceso seleccionado.

5. Si hay cambio en la asignación de CPU, se realiza el cambio de contexto llamando a context_switch

Diapositiva 81

La clase de planificación CFS (Completely Fair Scheduling):

- Idea general: repartir el tiempo de CPU de forma imparcial, garantizando que todos los procesos se ejecutarán y, dependiendo del número de procesos, asignarles más o menos tiempo de uso de CPU.

- Mantiene datos sobre los tiempos consumidos por los procesos.

- El kernel calcula un peso para cada proceso. Cuanto mayor sea el valor de la prioridad estática de un proceso, menor será el peso que tenga.

- vruntime (virtual runtime) de una entidad es el tiempo virtual que un proceso ha consumido y se calcula a partir del tiempo real que el proceso ha hecho uso de la CPU, su prioridad estática y su peso.

- El valor vruntime del proceso actual se actualiza:
  - periódicamente (el planificador periódico ajusta los valores de tiempo de CPU consumido)
  - cuando llega un nuevo proceso ejecutable
  - cuando el proceso actual se bloquea

- Cuando se decide qué proceso ejecutar a continuación, se elige el que tenga un valor menor de vruntime.

- Para realizar esto CFS utiliza un rbtree (red black tree):
  - estructura de datos que almacena nodos identificados por una clave y que permite una eficiente búsqueda dada una determinado valor de la clave.

- Cuando un proceso va a entrar en estado bloqueado:
  1. se añade a una cola asociada con la fuente del bloqueo
  2. se establece el estado del proceso a TASK_INTERRUPTIBLE o a TASK_NONINTERRUPTIBLE según sea conveniente
  3. es eliminado del rbtree de procesos ejecutables
  4. se llama a schedule para que se elija un nuevo proceso a ejecutar

- Cuando un proceso vuelve del estado bloqueado
  1. se cambia su estado a ejecutable (TASK_RUNNING)
  2. se elimina de la cola de bloqueo en que estaba
  3. se añade al rbtree de procesos ejecutables

Diapositiva 84

La clase de planificación de tiempo real:

- Se define la clase de planificación rt_sched_class.

- Los procesos de tiempo real son más prioritarios que los normales, y mientras existan procesos de tiempo real ejecutables éstos serán elegidos frente a los normales.

- Un proceso de tiempo real queda determinado por la prioridad que tiene cuando se crea, el kernel no incrementa o disminuye su prioridad en función de su comportamiento.

- Las políticas de planificación de tiempo real SCHED_RR y SCHED_FIFO posibilitan que el kernel Linux pueda tener un comportamiento soft real-time (Sistemas de tiempo real no estricto).

- Al crear el proceso también se especifica la política bajo la cual se va a planificar. Existe una llamada al sistema para cambiar la política asignada.

Diapositiva 85

Particularidades en SMP:

- Para realizar correctamente la planificación en un entorno SMP (multiprocesador), el kernel deberá tener en cuenta:
  1. Se debe repartir equilibradamente la carga entre las distintas CPUs
  2. Se debe tener en cuenta la afinidad de una tarea con una determinada CPU
  3. El kernel debe ser capaz de migrar procesos de una CPU a otra (puede ser una operación costosa)

- Periódicamente una parte del kernel deberá comprobar que se da un equilibrio entre las cargas de trabajo de las distintas CPUs y si detecta que una tiene más procesos que otra, reequilibra pasando procesos de una CPU a otra.