Planificadores

Durante su tiempo de vida, los procesos se mueven entre las diversas colas de planificación. El sistema operativo, como parte de la tarea de planificación, debe seleccionar de alguna manera los procesos que se encuentran en estas colas. El proceso de selección se realiza mediante un planificador apropiado.

A menudo, en un sistema de procesamiento por lotes, se envían más procesos de los que pueden ser ejecutados de forma inmediata. Estos procesos se guardan encola en un dispositivo de almacenamiento masivo (normalmente, un disco), donde se mantienen para su posterior ejecución. El planificador a largo plazo o planificador de trabajos selecciona procesos de esta cola y los carga en memoria para su ejecución. El planificador a corto plazo o planificador de la CPU selecciona de entre los procesos que ya están preparados para ser ejecutados y asigna la CPU a uno de ellos.

La principal diferencia entre estos dos planificadores se encuentra en la frecuencia de ejecución. El planificador a corto plazo debe seleccionar un nuevo proceso para la CPU frecuentemente. Un proceso puede ejecutarse sólo durante unos pocos milisegundos antes de tener que esperar por una solicitud de E/S. Normalmente, el planificador a corto plazo se ejecuta al menos una vez cada 100 milisegundos. Debido al poco tiempo que hay entre ejecuciones, el planificador a corto plazo debe ser rápido. Si tarda 10 milisegundos en decidir ejecutar un proceso durante 100 milisegundos, entonces el 10/(100 + 10) = 9 por ciento del tiempo de CPU se está usando (perdiendo) simplemente para planificar el trabajo.

El planificador a largo plazo se ejecuta mucho menos frecuentemente; pueden pasar minutos entre la creación de un nuevo proceso y el siguiente. El planificador a largo plazo controla el grado de multiprogramación (el número de procesos en memoria). Si el grado de multiprogramación es estable, entonces la tasa promedio de creación de procesos debe ser igual a la tasa promedio de salida de procesos del sistema. Por tanto, el planificador a largo plazo puede tener que invocarse sólo cuando un proceso abandona el sistema. Puesto que el intervalo entre ejecuciones es más largo, el planificador a largo plazo puede permitirse emplear más tiempo en decidir qué proceso debe seleccionarse para ser ejecutado.

Es importante que el planificador a largo plazo haga una elección cuidadosa. En general, la mayoría de los procesos pueden describirse como limitados por la E/S o limitados por la CPU. Un proceso limitado por E/S es aquel que invierte la mayor parte de su tiempo en operaciones de E/S en lugar de en realizar cálculos. Por el contrario, un proceso limitado por la CPU genera solicitudes de E/S con poca frecuencia, usando la mayor parte de su tiempo en realizar cálculos. Es importante que el planificador a largo plazo seleccione una adecuada mezcla de procesos, equilibrando los procesos limitados por E/S y los procesos limitados por la CPU. Si todos los procesos son limitados por la E/S, la cola de procesos preparados casi siempre estará vacía y el planificador a corto plazo tendrá poco que hacer. Si todos los procesos son limitados por la CPU, la cola de espera de E/S casi siempre estará vacía, los dispositivos apenas se usarán, y de nuevo el sistema se desequilibrará. Para obtener un mejor rendimiento, el sistema dispondrá entonces de una combinación equilibrada de procesos limitados por la CPU y de procesos limitados por E/S.

En algunos sistemas, el planificador a largo plazo puede no existir o ser mínimo. Por ejemplo, los sistemas de tiempo compartido, tales como UNIX y los sistemas Microsoft Windows, a menudo no disponen de planificador a largo plazo, sino que simplemente ponen todos los procesos nuevos en memoria para que los gestione el planificador a corto plazo. La estabilidad de estos sistemas depende bien de una limitación física (tal como el número de terminales disponibles), bien de la propia

naturaleza autoajustable de las personas que utilizan el sistema. Si el rendimiento desciende a niveles inaceptables en un sistema multiusuario, algunos usuarios simplemente lo abandonarán.

Algunos sistemas operativos, como los sistemas de tiempo compartido, pueden introducir un nivel intermedio adicional de planificación. La idea clave subyacente a un planificador a medio plazo es que, en ocasiones, puede ser ventajoso eliminar procesos de la memoria (con lo que dejan de contender por la CPU) y reducir así el grado de multiprogramación. Después, el proceso puede volver a cargarse en memoria, continuando su ejecución en el punto en que se interrumpió. Este esquema se denomina intercambio. El planificador a medio plazo descarga y luego vuelve a cargar el proceso. El intercambio puede ser necesario para mejorar la mezcla de procesos o porque un cambio en los requisitos de memoria haya hecho que se sobrepase la memoria disponible, requiriendo que se libere memoria.

Cambio de contexto

Las interrupciones hacen que el sistema operativo obligue a la CPU a abandonar su tarea actual, para ejecutar una rutina del kernel. Estos sucesos se producen con frecuencia en los sistemas de propósito general. Cuando se produce una interrupción el sistema tiene que guardar el contexto actual del proceso que se está ejecutando en la CPU, de modo que pueda restaurar dicho contexto cuando su procesamiento concluya, suspendiendo el proceso y reanudándolo después. El contexto se almacena en el PCB del proceso e incluye el valor de los registros de la CPU, el estado del proceso y la información de gestión de memoria. Es decir, realizamos una salvaguarda del estado actual de la CPU, en modo kernel , o en modo usuario, y una restauración del estado para reanudar las operaciones.

La conmutación de la CPU a otro proceso requiere una salvaguarda del estado del proceso actual y una restauración del estado de otro proceso diferente. Esta tarea se conoce como cambio de contexto. Cuando se produce un cambio de contexto, el kernel guarda el contexto del proceso antiguo en su PCB y carga el contexto almacenado del nuevo proceso que se ha decidido ejecutar.

El tiempo dedicado al cambio de contexto es tiempo desperdiciado, dado que el sistema no realiza ningún trabajo útil durante la conmutación. La velocidad del cambio de contexto varía de una máquina a otra, dependiendo de la velocidad de memoria, del número de registros que tengan que copiarse y de la existencia de instrucciones especiales (como por ejemplo, una instrucción para cargar o almacenar todos los registros). Las velocidades típicas son del orden de unos pocos milisegundos.

El tiempo empleado en los cambios de contexto depende fundamentalmente del soporte hardware. Por ejemplo, algunos procesadores (como Ultra\SPARC de Sun) proporcionan múltiples conjuntos de registros. En este caso, un cambio de contexto simplemente requiere cambiar el puntero al conjunto actual de registros. Por supuesto, si hay más procesos activos que conjuntos de registros, el sistema recurrirá a copiar los datos de los registros en y desde memoria, al igual que antes. También, cuanto más complejo es el sistema operativo, más trabajo debe realizar durante un cambio de contexto. Existen técnicas avanzadas de gestión de memoria pueden requerir que con cada contexto se intercambien datos adicionales. Por ejemplo, el espacio de direcciones del proceso actual debe preservarse en el momento de preparar para su uso el espacio de la siguiente tarea. Cómo se conserva el espacio de memoria y qué cantidad de trabajo es necesario para conservar lo depende del método de gestión de memoria utilizado por el sistema operativo.

Operaciones sobre los procesos

En la mayoría de los sistemas, los procesos pueden ejecutarse de forma concurrente y pueden crearse y eliminarse dinámicamente. Por tanto, estos sistemas deben proporcionar un mecanismo para la creación y terminación de procesos.

Creación de procesos

Un proceso puede crear otros varios procesos nuevos mientras se ejecuta; para ello se utiliza una llamada al sistema específica para la creación de procesos. El proceso creador se denomina proceso padre y los nuevos procesos son los hijos de dicho proceso. Cada uno de estos procesos nuevos puede a su vez crear otros procesos, dando lugar a un árbol de procesos.

La mayoría de los sistemas operativos (incluyendo UNIX y la familia Windows de sistemas operativos) identifican los procesos mediante un identificador de proceso unívoco o pid (process identifier), que normalmente es un número entero.

En UNIX, puede obtenerse un listado de los procesos usando el comando ps. Por ejemplo, el comando ps -el proporciona información completa sobre todos los procesos que están activos actualmente en el sistema. Resulta fácil construir un árbol de procesos, trazando recursivamente los procesos padre hasta llegar al proceso init.

En general, un proceso necesitará ciertos recursos (tiempo de CPU, memoria, archivos, dispositivos de E/S) para llevar a cabo sus tareas. Cuando un proceso crea un subproceso, dicho subproceso puede obtener sus recursos directamente del sistema operativo o puede estar restringido a un subconjunto de los recursos del proceso padre. El padre puede tener que repartir sus recursos entre sus hijos, o puede compartir algunos recursos (como la memoria o los archivos) con alguno de sus hijos. Restringir un proceso hijo a un subconjunto de los recursos del padre evita que un proceso pueda sobrecargar el sistema creando demasiados subprocesos.

Además de los diversos recursos físicos y lógicos que un proceso obtiene en el momento de su creación, el proceso padre puede pasar datos de inicialización (entrada) al proceso hijo. Por ejemplo. considere un proceso cuya función sea mostrar los contenidos de un archivo, por ejemplo img.jpg, en la pantalla de un terminal. Al crearse, obtendrá como entrada de su proceso padre el nombre del archivo img.jpg y empleará dicho nombre de archivo, lo abrirá y mostrará el contenido. También puede recibir el nombre del dispositivo de salida. Algunos sistemas operativo s pasan recursos a los procesos hijo. En un sistema así, el proceso nuevo puede obtener como entrada dos archivos abiertos, img.jpg y el dispositivo terminal, y simplemente transferir los datos entre ellos.

Cuando un proceso crea otro proceso nuevo, existen dos posibilidades en términos de ejecución:

1. El padre continúa ejecutándose concurrentemente con su hijo.
2. El padre espera hasta que alguno o todos los hijos han terminado de ejecutarse.

También existen dos posibilidades en función del espacio de direcciones del nuevo proceso:

1. El proceso hijo es un duplicado del proceso padre (usa el mismo programa y los mismos datos que el padre).
2. El proceso hijo carga un nuevo programa.

Para ilustrar estas diferencias, consideremos en primer lugar el sistema operativo UNIX. En UNIX, cada proceso se identifica mediante su identificador de proceso, que es un entero unívoco. Puede crearse un proceso nuevo mediante la llamada al sistema fork ( ). El nuevo proceso consta de una copia del espacio de direcciones del proceso original. Este mecanismo permite al proceso padre comunicarse fácilmente con su proceso hijo. Ambos procesos (padre e hijo) continúan la ejecución en la instrucción que sigue a fork ( ), con una diferencia: el código de retorno para fork () es cero en el caso del proceso nuevo (hijo), mientras que al padre se le devuelve el identificador de proceso (distinto de cero) del hijo.

Normalmente, uno de los dos procesos utiliza la llamada al sistema exec ( ) después de una llamada al sistema fork (), con el fin de sustituir el espacio de memoria del proceso con un nuevo programa. La llamada al sistema exec () carga un archivo binario en memoria (destruyendo la imagen en memoria del programa que contiene la llamada al sistema exec ( )) e inicia su ejecución. De esta manera, los dos procesos pueden comunicarse y seguir luego caminos separados, El padre puede crear más hijos, o, si no tiene nada que hacer mientras se ejecuta el hijo, puede ejecutar una llamada al sistema wait () para auto-excluirse de la cola de procesos preparados hasta que el proceso hijo se complete.

Terminación de procesos

Un proceso termina cuando ejecuta su última instrucción y pide al sistema operativo que lo elimine usando la llamada al sistema exit ( ) . En este momento, el proceso puede devolver un valor de estado (normalmente, un entero) a su proceso padre (a través de la llamada al sistema wait()). El sistema operativo libera la asignación de todos los recursos del proceso, incluyendo las memorias física y virtual, los archivos abiertos y los búferes de E/S.

La terminación puede producirse también en otras circunstancias. Un proceso puede causar la terminación de otro proceso a través de la adecuada llamada al sistema (por ejemplo, TerminateProcess en Win32). Normalmente, dicha llamada al sistema sólo puede ser invocada por el padre del proceso-que se va a terminar. En caso contrario, los usuarios podrían terminar arbitrariamente los trabajos de otros usuarios. Observe que un padre necesita conocer las identidades de sus hijos. Por tanto, cuando un proceso crea un proceso nuevo, se pasa al padre la identidad del proceso que se acaba de crear.

Un padre puede terminar la ejecución de uno de sus hijos por diversas razones, como por ejemplo, las siguientes:

- El proceso hijo ha excedido el uso de algunos de los recursos que se le han asignado. Para determinar si tal cosa ha ocurrido, el padre debe disponer de un mecanismo para inspeccionar el estado de sus hijos.
- La tarea asignada al proceso hijo ya no es necesaria.
- El padre abandona el sistema, y el sistema operativo no permite que un proceso hijo continúe si su padre ya ha terminado.

Algunos sistemas, incluyendo VMS, no permiten que un hijo siga existiendo si su proceso padre se ha completado. En tales sistemas, si un proceso termina (sea normal o anormalmente), entonces todos sus hijos también deben terminarse. Este fenómeno, conocido como terminación en cascada, normalmente lo inicia el sistema operativo.

Para ilustrar la ejecución y terminación de procesos, considere que, en UNIX, podemos terminar un proceso usando la llamada al sistema exit ( ); su proceso padre puede esperar a la terminación del

proceso hijo usando la llamada al sistema wait ( ). La llamada al sistema wait ( ) devuelve el identificador de un proceso hijo completado, con el fin de que el padre puede saber cuál de sus muchos hijos ha terminado. Sin embargo, si el proceso padre se ha completado, a todos sus procesos hijo se les asigna el proceso init como su nuevo padre. Por tanto, los hijos todavía tienen un padre al que proporcionar su estado y sus estadísticas de ejecución.