Dentro del planificador de Linux 2.6

por Amir Elaguizy y Jim "Spamizbad" Battin
Traducción: Javier Smaldone
Original: http://www.arstechnica.com/etc/linux/index.html

Al fin, el kernel 2.6 se encuentra terminado. La navidad nos da bastante tiempo libre para jugar con sus nuevas características. La lista de cambios es bastante extensa; se ha realizado mucho trabajo sobre este nuevo kernel. Uno de los cambios más importantes y visibles es la introducción del planificador (scheduler, en inglés) de O(1). El planificador es la pieza del kernel que asigna porciones de tiempo a programas individuales que se ejecutan en la computadora. Él posibilita que una única CPU pueda ejecutar múltiples programas simultaneamente permitiendo que un programa se ejecute por determinada cantidad de tiempo, luego intercambiando a otro programa, permitiéndole ejecutarse por otra cantidad de tiempo, y así sucesivamente. Por lo tanto, un buen planificador que administre el tiempo de la CPU eficientemente puede dar al usuario una sensación de mayor velocidad de respuesta.

¿Qué demonios es O(1)?

"O(1)" es un ejemplo de la notación "Oh grande" ("Big-Oh", en inglés), usada en ciencias de la computación para descrisluggishbir el tiempo de ejecución de un algoritmo. Un algoritmo es un método "paso a paso" bien definido para cumplir determinada tarea computacional. La notación "Oh grande" no mide realmente el número de milisegundos que toma a un algoritmo ejecutarse; tal medida es altamente dependiente de las especificaciones de la máquina, sistema operativo, etc. En cambio, mide cuánto tarda un algoritmo en proporción al tamaño de su entrada. Por ejemplo, el algoritmo de ordenamiento por inserción es $O(n^2)$, lo que significa que el tiempo que tarda en ordenar una lista de elementos es proporcional al cuadrado del tamaño de la entrada.

Es fácil reconocer la entrada de varios algoritmos. Por ejemplo, la entrada del algoritmo de ordenamiento por inserción es una lista de elementos. Los algoritmos de planificación toman una tarea a ejecutar y deciden cuánto tiempo se le dará; su entrada es el conjunto de tareas que el sistema operativo está manejando. Cuando decimos que el planificador es O(1), queremos decir que toma una cantidad de tiempo constante para ejecutarse, independientemente de la cantidad de tareas. Ya sea que su computadora esté ejecutando 1000 tareas simultaneamente o 10, el planificador siempre tardará la misma cantidad de tiempo en elegir la próxima tarea. Esta es una característica muy deseable para un planificador.

Planificadores y los procesos que planifican

Para entender cómo el planificador asigna el tiempo de CPU y administra la ejecución de los programas, es necesario entender el concepto de "proceso". Un proceso es un conjunto de instrucciones que el procesador ejecuta secuencialmente. Uno o más procesos pueden constituir un programa.

Los procesos pueden tener uno de cinco estados: en ejecución (running), interrumpible (interruptible), no interrumpible (uninterruptible), zombi (zombie) o detenido (stopped). En este artículo, sólo consideraremos los estados "en ejecución" e "interrumpible". Un proceso en el estado "en ejecución" está o bien siendo ejecutado por el procesador, o almacenado en la memoria esperando ser ejecutado. Un proceso "interrumpible" se encuentra actualmente "bloqueado". El bloqueo ocurre cuando un

proceso voluntariamente cede el control de la CPU hasta que se cumpla determinada condición. Esto típicamente ocurre cuando un proceso está esperando una entrada del usuario, disco duro, red u otro recurso externo.

El planificador de Linux 2.6

Los procesos pueden tener varios niveles de prioridad. El planificador de Linux 2.6 utiliza un algoritmo eficiente para favorecer a los procesos de alta prioridad, a la vez que permite ejecutarse a los procesos de baja prioridad. El planificador mantiene una lista de los niveles de prioridad. Cuando llega el momento de seleccionar un proceso, el planificador busca el nivel de prioridad más alto que contenga procesos disponibles. Luego selecciona un proceso del nivel de prioridad deseado. Dado que el número de niveles de prioridad es fijo, el planificador siempre demora una cantidad constante de tiempo.

El planificador tiene otra capacidad importante: "expropiación" ("preemption" en inglés). Esto permite que un proceso pueda ser detenido en cualquier momento, permitiendo que un proceso de mayor prioridad pueda ser iniciado. Esto es muy importante desde el punto de vista del usuario, ya que permite que los procesos que interactúan con el usuario sean ejecutados cuando sea necesario, mientras permite que los procesos de baja prioridad que no interactúan con el usuario se ejecuten en segundo plano (background). Por ejemplo, imagine que está ejecutando un cliente de computación distribuida. Este programa usa la mayor parte de su tiempo de CPU cuando usted no está frente a su computadora. En el momento en que usted comienza a utilizar un programa tal como un navegador web, este último desaloja al cliente de computación distribuida. Su sistema no parecerá lento o pesado, incluso si usted tiene un proceso computacionalmente intensivo ejecutándose en segundo plano.

Los beneficios de la expropiación son tan grandes que los desarrolladores del kernel decidieron hacer que el kernel mismo sea "expropiable" ("preemptible", en inglés). Esto permite a una tarea del kernel, tal como E/S de disco ser "expropiado", por ejemplo, por un evento de teclado. Esto le permite al sistema tener mejores tiempos de respuesta a las demandas del usuario. El kernel 2.6 es capaz de administrar eficientemente sus propias tareas además de los procesos de usuario.

Colas de ejecución y balance de carga

Linux 2.6 es capaz de administrar eficientemente más procesadores que Linux 2.4, la versión estable anterior. La habilidad de manipular distintas cantidades de recursos computacionales, desde pequeñas CPUs empotradas a supercomputadoras masivas de 64 procesadores es llamada "escalabilidad"; y es una de las mayores fortalezas del nuevo kernel. Uno podría pensar que un planificador diseñado para administrar una CPU nunca podría ser adaptado para administrar 64 CPUs; sin embargo, el planificador de Linux 2.6 puede administrar sistemas de muchos procesadores a través del uso de listas especiales llamadas "colas de ejecución". Una cola de ejecución almacena información acerca de los procesos que se están ejecutando en una sola CPU; habiendo una cola de ejecución por cada CPU en el sistema. La información contenida en las colas de ejecución le permiten al planificador transferir el control de una CPU hacia otra, usando un método denominado "balance de carga".

Balance de carga es una manera de asegurar que los recursos de una CPU no serán desperdiciados mientras otra CPU se encuentre sobrecargada. Si el planificador encuentra que una cola de ejecución tiene muchos más procesos en ella que otra, uno o más procesos pueden ser movidos desde la cola mas larga hacia la mas pequeña. El balanceador de carga es invocado cada vez que se vacía una cola de ejecución; si no hay colas de ejecución vacías, es invocado periódicamente. El temporizador periódico le permite al sistema mantener un balance de carga razonable a través de varias CPUs sin tener que

dedicar demasiado tiempo a mover procesos de una CPU hacia otra. Balancear la carga cuando una cola de ejecución se vacía permite al planificador asegurar que el precioso poder de CPU nunca va a desperdiciarse. Hay una excepción a esta regla del balanceo ‹ algunos procesos especiales pueden ser fijados a una cierta cola de ejecución. Este atributo es llamado "afinidad de hilos" ("thread affinity", en inglés), un "hilo" ("thread", en inglés) es simplemente otro nombre para "proceso".

Es importante notar que si el soporte de SMP ("multiprocesamiento simétrico" o "symmetric multiprocessing", en inglés) no se encuentra habilitado cuando uno compila el kernel, no será habilitado el código de balance de carga y no se desperdiciará tiempo tratando de balancear la carga de una única CPU.

Porciones de tiempo (timeslices) y amabilidad (niceness)

Usted puede estarse preguntando cuánto tiempo asigna el planificador a cada proceso. La cantidad de tiempo de CPU que el planificador asigna a un proceso particular es llamada una "porción de tiempo" ("timeslice", en inglés). Luego de que la porción de tiempo de un proceso ha sido utilizada, el proceso es detenido para que pueda ejecutarse el siguiente. Es importante recordar que un proceso puede ser detenido en medio de su porción de tiempo; este es el propósito de la "expropiación". A distintos procesos les son asignadas distintas porciones de tiempo basadas en la prioridad; los procesos de alta prioridad se ejecutar una mayor cantidad de tiempo que los de baja prioridad. La prioridad no es un concepto unitario; cada proceso tiene una prioridad estática y una dinámica.

La prioridad estática, o "amabilidad" ("niceness", en inglés) en la terminología tradicional de UNIX es la medida de cuán importante es un proceso. Puede ser fijada por el usuario o por otros programas. A los procesos con un valor bajo de "amabilidad" se les asigna una porción de tiempo mayor; a aquellos con valores altos, se les asignan porciones de tiempo mas pequeñas. Los valores de "amabilidad" tienen un rango de -20 a +19. El hecho de que los procesos de mayor prioridad tengan valores de "amabilidad" más baja puede resultar confuso si uno ve a la "amabilidad" como una medida de prioridad; en cambio, piense en la "amabilidad" como una medida de la disposición del proceso a ceder ante otros. Los valores de "amabilidad" pueden ser fijados desde la línea de comando con el comando "nice", o a través de algunos programas de monitoreo del sistema. La edición del 5 de noviembre de Linux.Ars incluye una buena introducción a la administración de procesos con tales herramientas.

La prioridad dinámica de un proceso es determinada por el planificador monitoreando su comportamiento durante la ejecución. Los procesos que gastan mucho de su tiempo bloqueándose son conocidos como procesos "cota de E/S" ("I/O bound", en inglés); su comportamiento está acotado por la entrada y la salida. Cuando el planificador reconoce a un proceso "cota de E/S" se le asigna una bonificación negativa (de hasta -5), y por lo tanto una mayor porción de tiempo. En contraste, a los procesos "cota de CPU" se les asigna una bonificación positiva (de hasta +5), y por lo tanto una porción de tiempo menor. Esto previene que los procesos "cota de CPU" controlen el procesador, y permite a las entradas y salidas desarrollarse sin problemas. A veces, un proceso puede intercambiar entre comportamientos "cota de E/S" y "cota de CPU". Por ejemplo, un proceso puede leer cierta información desde el teclado y realizar algunas computaciones basadas en esa entrada. El planificador necesita constantemente estar alerta del comportamiento del proceso; las prioridades dinámicas pueden ser reajustadas de acuerdo a esto.

El planificador lleva un registro de todas las porciones de tiempo con dos listas de procesos. La primera lista contiene a los procesos que todavía disponen de tiempo; la segunda contiene a quellos procesos que ya han agotado su tiempo. Cuando un proceso usa su porción de tiempo, el planificador calcula una nueva porción añadiendo la bonificación de prioridad dinámica a la prioridad estática. El

proceso es luego insertado en la segunda lista. Cuando la primera lista se vacía, la segunda lista la reemplaza y viceversa. Esto permite al planificadorhttp://www.diccionarios.com/nocookie.phtml calcular continuamente las porciones de tiempo con una sobrecarga computacional mínima.

¿Por qué debe ser tan cuidadoso el planificador cuando calcula porciones de tiempo? Un buen planificador debe alcanzar un balance apropiado entre caudal y latencia. Caudal ("throughput", en inglés) es la cantidad de datos que pueden ser transferidos desde una ubicación hacia otra. Latencia es el tiempo que le toma a un proceso responder a una entrada. Este balance es alcanzado ajustando las porciones de tiempo. Un proceso "cota de E/S" necesita buen caudal para cumplir sus tareas rápidamente. Esta es la razón por la cual el planificador le da a este tipo de procesos porciones de tiempo mayores; ellos tienen que hacer y responder a los requerimientos de E/S, y no tiene que esperar demasiado para que otros procesos se ejecuten. Dado que casi todos los procesos pueden beneficiarse de un caudal superior, ¿por qué no darles a todos los procesos una porción de tiempo mayor? Si un planificador hiciera esto, sufriría la latencia. Dado que cada proceso toma un gran tiempo para completar su tarea, otros procesos no serán capaces de responder rápidamente a la entrada del usuario. Un buen balance entre caudal y latencia llevan a una sensación de respuesta veloz para el usuario con el suficiente caudal.

Conclusión

El kernel Linux 2.6 provee un fundamento potente para las nuevas distribuciones de Linux, programas y dispositivos. Se ha puesto un gran esfuerzo en el diseño y testeo de esta nueva versión; este esfuerzo ha sido tan grande que mucha gente piensa que el nuevo kernel debería ser numerado como 3.0 en vez de 2.6. Desarrolladores como Ingo Molnar, Robert Love, Con Kolivas, David Libenzi y Linus Torvalds, además de otros miles de desarrolladores y testeadores nos han entregado esta excelente versión. Exiten otras importantes áreas del kernel que han sido rediseñadas, tales como partes clave del subsistema de E/S pero, tristemente, están más allá del alcance de este artículo. Si está interesado en las cuestiones relacionadas con el kernel le recomendamos Kerneltrap y Kernel Newbies. Los usuarios más ambiciosos pueden desear suscribirse a la Lista de Correo del Kernel Linux (Linux Kernel Mailing List), el medio oficial para desarrollo y conversación sobre el kernel.

Nota del traductor

Esta traducción ha sido realizada de manera independiente y sin la participación de Ars Technica. Mi intención al realizar esta traducción ha sido poner el mismo al alcance de quienes no leen inglés. He tratado de ser fiel al contenido del texto original.

Javier Smaldone
http://www.smaldone.com.ar