

THREADS & MULTITHREADS

Curso: Lenguajes de programación

Alumnos: Rocio Suarez, Percy Ramos

Terminología

* Threads/ hilos 🡪 secuencia mas pequeña de comandos programadas que puede ser administrada independientemente por un planificador, que normalmente es una parte del sistema operativo.
* Subproceso 🡪 componente de un proceso.
* La implementación de subprocesos y procesos varia entre sistemas operativos
* Multithreads/ hilos múltiples 🡪 pueden existir dentro de un proceso, se ejecutan simultáneamente y compartiendo recursos como la memoria, mientras que diferentes procesos no comparten estos recursos. En particular, los hilos de un proceso comparten su código ejecutable y los valores de sus variables en un momento dado.

Historia

Los hilos se presentaron en la multiprogramación OS / 360 en 1967 con un numero variable de tareas (Multiple Variable Tasks o MVT). El término "hilo" se ha atribuido a Víctor A. Vyssotsky. Los programadores de procesos de muchos sistemas operativos modernos soportan directamente el procesamiento en tiempo, multiprocesador y el núcleo del sistema operativo permite a los programadores manipular los hilos exponiendo la funcionalidad requerida a través de la interfaz de llamada del sistema. Algunas implementaciones de subprocesos se llaman hilos de núcleo, mientras que los procesos de peso ligero (Light Weight Processes o LWP) son un tipo específico de hilo de núcleo que comparten el mismo estado e información. Además, los programas pueden tener hilos de espacio de usuario al enhebrar con temporizadores, señales u otros métodos para interrumpir su propia ejecución, realizando una especie de corte de tiempo ad hoc.

Sistemas Single vs Multiprocessor:

Los sistemas con un solo procesador implementan generalmente multithreading (hilos multiples) por el tiempo que rebanan, la unidad de proceso central (CPU) cambia entre las diversas hebras del software. Este cambio de contexto generalmente ocurre con mucha frecuencia y en milisegundos para que los usuarios perciban los hilos o las tareas como ejecutándose en paralelo. En un sistema multiprocesador o multi-core, múltiples subprocesos pueden ejecutarse en paralelo, con cada procesador o núcleo ejecutando un hilo separado simultáneamente; En un procesador o núcleo con subprocesos de hardware, los subprocesos de software independientes también se pueden ejecutar simultáneamente mediante subprocesos de hardware independientes.

Hilos vs procesadores

Los hilos se diferencian de los procesos tradicionales del sistema operativo multitarea en que:

|  |  |
| --- | --- |
| **Procesadores** | **Hilos** |
| Típicamente independientes | Existen como subconjuntos de un proceso |
| Llevan mucha mas información de estado | Múltiples hilos dentro de un proceso comparten el estado, la memoria y otros recursos del proceso. |
| Tienen espacios de direcciones separadas | Comparten su espacio de direcciones |
| Interactúan solo a través de mecanismos de comunicación entre procesos (proporcionados por el sistema) |  |
| El cambio de contexto entre subprocesos en el mismo proceso suele ser mas rápido que el cambio de contexto entre procesos |  |

Hilos

Un solo hilo o ‘threading’ es el proceso de un comando a la vez.

Multihilos

Hilos multiples o ‘multithreading’ se encuentra principalmente en sistemas operativos multitarea. Este es un modelo de programación y ejecución generalizado que permite que múltiples subprocesos existan dentro del contexto de un proceso. Estos hilos comparten los recursos del proceso, pero son capaces de ejecutarse independientemente. El modelo de programación roscado proporciona a los desarrolladores una abstracción útil de la ejecución concurrente. Multithreading también se puede aplicar a un proceso para permitir la ejecución paralela en un sistema de multiprocesamiento.

**Las aplicaciones multiproceso tienen las siguientes ventajas:**

* **Capacidad de respuesta:** el multithreading puede permitir que una aplicación siga respondiendo a la entrada. En un programa de un hilo, si el hilo de ejecución principal bloquea en una tarea de ejecución larga, toda la aplicación puede aparecer para congelarse. Al mover esas tareas de ejecución larga a un subproceso de trabajo que se ejecuta simultáneamente con el subproceso de ejecución principal, es posible que la aplicación siga respondiendo a la entrada del usuario mientras ejecuta tareas en segundo plano. Por otra parte, en la mayoría de los casos multithreading no es la única forma de mantener un programa en respuesta, con señales de E / S y / o Unix no bloqueantes disponibles para obtener resultados similares.
* **Ejecución más rápida:** esto permite operar más rápido en sistemas informáticos que tienen múltiples unidades centrales de procesamiento (CPUs) o uno o más procesadores multi-core, o a través de un grupo de máquinas, porque los hilos del programa prestan naturalmente a la ejecución paralela, asumiendo una independencia suficiente (que no necesitan esperar el uno al otro).
* **Menor consumo de recursos:** mediante el uso de subprocesos, una aplicación puede servir a varios clientes simultáneamente utilizando menos recursos de los que necesitaría al utilizar múltiples copias de proceso de sí mismo. Por ejemplo, el servidor HTTP de Apache utiliza grupos de subprocesos: un grupo de subprocesos de escucha para escuchar solicitudes entrantes y un grupo de subprocesos de servidor para procesar esas solicitudes.
* **Mejor utilización del sistema:** como ejemplo, un sistema de archivos que utiliza múltiples subprocesos puede lograr un mayor rendimiento y una menor latencia, ya que los datos de un medio más rápido (como la memoria caché) pueden ser recuperados por un hilo mientras otro hilo recupera datos de un medio más lento Como almacenamiento externo) con ninguno de los hilos esperando que el otro termine.
* **Compartición y comunicación simplificadas:** a diferencia de los procesos que requieren un mecanismo de paso de mensajes o de memoria compartida para realizar la comunicación entre procesos (IPC), los hilos pueden comunicarse a través de datos, códigos y archivos que ya comparten.
* **Paralelización:** las aplicaciones que buscan utilizar sistemas multicore o multi-CPU pueden utilizar multithreading para dividir datos y tareas en subtareas paralelas y permitir que la arquitectura subyacente administre cómo se ejecutan los subprocesos, ya sea simultáneamente en un núcleo o en paralelo en múltiples núcleos. Los entornos de computación GPU como CUDA y OpenCL utilizan el modelo multithreading donde decenas de cientos de subprocesos se ejecutan en paralelo a través de datos en un gran número de núcleos.

**Multithreading tiene los siguientes inconvenientes:**

* **Sincronización:** dado que los hilos comparten el mismo espacio de direcciones, el programador debe tener cuidado de evitar las condiciones de carrera y otros comportamientos no intuitivos. Para que los datos se manipulen correctamente, los hilos a menudo necesitarán rendezvous en el tiempo para procesar los datos en el orden correcto. Los hilos también pueden requerir operaciones mutuamente exclusivas (a menudo implementadas usando semáforos) para evitar que los datos comunes sean modificados o leídos simultáneamente mientras están en proceso de modificación. El uso descuidado de tales primitivas puede dar lugar a bloqueos.
* **El hilo bloquea un proceso:** una operación ilegal realizada por un hilo bloquea todo el proceso; Por lo tanto, un hilo de comportamiento incorrecto puede interrumpir el procesamiento de todos los otros hilos en la aplicación.

Programación

Los sistemas operativos planifican los subprocesos de forma preventiva o cooperativa. El mulithreading preventivo se considera generalmente el enfoque superior, ya que permite al sistema operativo determinar cuándo debe producirse un cambio de contexto. La desventaja del preemptive multihreading[[1]](#footnote-1) es que el sistema puede hacer un cambio de contexto en un momento inapropiado, causando convoy de bloqueo, inversión de prioridad u otros efectos negativos, lo que puede evitarse mediante multihilo cooperativo. El multithreading cooperativo, por otro lado, depende de los hilos mismos para abandonar el control una vez que están en un punto de parada. Esto puede crear problemas si un subproceso está esperando a que un recurso esté disponible.

Hasta principios de la década de 2000, la mayoría de las computadoras de escritorio sólo disponían de una CPU de un solo núcleo, sin soporte para los hilos de hardware, aunque los hilos seguían siendo utilizados en dichas computadoras.

En 2002, Intel añadió soporte para multithreading simultáneo al procesador Pentium 4, bajo el nombre de hyper-threading.

En 2005, introdujeron el procesador dual-core Pentium D y AMD introdujo el procesador dual-core Athlon 64 X2.

Los procesadores de sistemas embebidos, que tienen requisitos más altos para comportamientos en tiempo real, podrían soportar el multihilo reduciendo el tiempo de conmutación de subprocesos, tal vez asignando un archivo de registro dedicado para cada subproceso en lugar de guardar o restaurar un archivo de registro común.

Procesos, hilos de núcleo, hilos de usuario y fibras

La programación puede hacerse a nivel de kernel o nivel de usuario, y la multitarea puede realizarse de forma preventiva o cooperativa. Esto produce una variedad de conceptos relacionados.

En el nivel del núcleo, un proceso contiene uno o más subprocesos del núcleo, que comparten los recursos del proceso, como memoria y manejadores de archivos; un proceso es una unidad de recursos, mientras que un subproceso es una unidad de programación y ejecución. Normalmente, la programación del núcleo se realiza de forma ‘preemptiva’ o, menos comúnmente, en forma cooperativa. A nivel de usuario, un proceso tal como un sistema de tiempo de ejecución puede programar varios hilos de ejecución. Si éstos no comparten datos, como en Erlang, usualmente se llaman procesos análogos, mientras que si comparten datos usualmente se les llama hilos (usuarios), en particular si se programan de forma preventiva. Los hilos de usuario programados cooperativamente se conocen como fibras; Diferentes procesos pueden programar los subprocesos de usuario de forma diferente. Los subprocesos de usuario pueden ser ejecutados por los subprocesos del kernel de varias maneras (uno a uno, muchos a uno, muchos a muchos). El término "proceso de peso ligero" hace referencia de diversas maneras a hilos de usuario o a mecanismos de núcleo para programar hilos de usuario en hilos de núcleo.

Un proceso es una unidad "pesada" de programación del núcleo, ya que crear, destruir y cambiar procesos es relativamente costoso. Procesa recursos propios asignados por el sistema operativo. Los recursos incluyen memoria (para código y datos), manejadores de archivos, sockets, controladores de dispositivos, ventanas y un bloque de control de procesos. Los procesos son aislados por el aislamiento del proceso y no comparten espacios de direcciones o recursos de archivos excepto a través de métodos explícitos como heredar identificadores de archivos o segmentos de memoria compartida o asignar el mismo archivo de una manera compartida. Crear o destruir un proceso es relativamente costoso, ya que los recursos deben ser adquiridos o liberados. Los procesos son típicamente multitarea de manera preventiva, y la conmutación de procesos es relativamente costosa, más allá del coste básico de la conmutación de contexto, debido a problemas tales como la descarga de caché.

Un hilo de kernel es una unidad "ligera" de programación del kernel. Al menos un hilo de kernel existe dentro de cada proceso. Si existen varios hilos de núcleo en un proceso, comparten los mismos recursos de memoria y archivo. Los subprocesos del núcleo son preemptivos multitarea si el programador de procesos del sistema operativo es preventivo. Los subprocesos del núcleo no poseen recursos excepto una pila, una copia de los registros incluyendo el contador del programa y el almacenamiento local de hilos (si los hay), y por lo tanto son relativamente baratos de crear y destruir. La conmutación de subprocesos también es relativamente barata: requiere un conmutador de contexto (guardar y restaurar registros y puntero de pila), pero no cambia la memoria virtual y, por lo tanto, es compatible con el caché (dejando TLB válido). El núcleo puede asignar un hilo a cada núcleo lógico en un sistema (porque cada procesador se divide en múltiples núcleos lógicos si es compatible con multithreading, o sólo admite un núcleo lógico por núcleo físico si no lo hace), y puede intercambiar hilos que Bloquearse Sin embargo, los subprocesos del kernel tardan mucho más que los subprocesos de usuario que se intercambian.

Los hilos se implementan a veces en las bibliotecas de espacio de usuario, llamadas hilos de usuario. El kernel no tiene conocimiento de ellos, por lo que se gestionan y programan en el espacio de usuario. Algunas implementaciones basan sus subprocesos de usuario en la parte superior de varios hilos del núcleo, para beneficiarse de las máquinas multiprocesador (modelo M: N). En este artículo, el término "hilo" (sin kernel o calificador de usuario) predeterminado se refiere a hilos de kernel. Los subprocesos de usuario implementados por máquinas virtuales también se denominan subprocesos verdes. Los subprocesos de usuario generalmente son rápidos para crear y administrar, pero no pueden aprovechar la multitarea o el multiproceso y se bloquearán si todos los subprocesos del núcleo asociados se bloquean aunque haya algunos subprocesos de usuario listos para ejecutarse.

Las fibras son una unidad de programación aún más ligera que se programan de forma cooperativa: una fibra en ejecución debe "ceder" explícitamente para permitir que otra fibra funcione, lo que hace que su implementación sea mucho más fácil que los hilos del núcleo o del usuario. Una fibra se puede programar para funcionar en cualquier hilo en el mismo proceso. Esto permite que las aplicaciones obtengan mejoras de rendimiento mediante la administración de la programación en sí, en lugar de confiar en el programador del núcleo (que puede no estar ajustado para la aplicación). Los entornos de programación paralelos como Open MP suelen implementar sus tareas a través de fibras. Estrechamente relacionadas con las fibras son las co-rutinas, con la distinción de que las co-rutinas son una construcción a nivel de lenguaje, mientras que las fibras son una construcción a nivel de sistema.

Modelos

**1: 1 (enhebrado a nivel de kernel)**

Los subprocesos creados por el usuario en una correspondencia 1: 1 con entidades programables en el kernel son ​​la implementación de subprocesos más sencilla posible. OS / 2 y Win32 utilizaron este enfoque desde el principio, mientras que en Linux la biblioteca C habitual implementa este enfoque (a través de la NPTL o más antiguos LinuxThreads). Este enfoque también es utilizado por Solaris, NetBSD, FreeBSD, macOS e iOS.

**N: 1 (nivel de usuario de roscado)**

Un modelo N: 1 implica que todos los subprocesos de nivel de aplicación se asignan a una entidad planificada a nivel de kernel; el núcleo no tiene conocimiento de los subprocesos de la aplicación. Con este enfoque, el cambio de contexto puede hacerse muy rápidamente y, además, puede implementarse incluso en núcleos simples que no soportan el enhebrado. Sin embargo, uno de los principales inconvenientes es que no puede beneficiarse de la aceleración de hardware en los procesadores multiproceso o en los equipos con varios procesadores: nunca hay más de un subproceso programado al mismo tiempo. Por ejemplo: Si uno de los subprocesos necesita ejecutar una solicitud de E / S, se bloquea todo el proceso y no se puede utilizar la ventaja de subprocesamiento. Los hilos portátiles de GNU usan el subproceso de nivel de usuario, al igual que los subprocesos de estado.

**M: N (hilo híbrido)**

M: N mapea un número M de hilos de aplicación en un número N de entidades del kernel, [10] o "procesadores virtuales". Se trata de un compromiso entre el nivel de kernel ("1: 1") y el nivel de usuario ("N: 1"). En general, los sistemas de subprocesamiento "M: N" son más complejos de implementar que los subprocesos del kernel o del usuario, ya que se requieren cambios en el código del kernel y del espacio del usuario. En la implementación M: N, la biblioteca de subprocesos es responsable de programar los subprocesos de usuario en las entidades planificables disponibles; Esto hace que el cambio de contexto de los hilos sea muy rápido, ya que evita las llamadas al sistema. Sin embargo, esto aumenta la complejidad y la probabilidad de inversión de prioridad, así como una programación subóptima sin una coordinación extensa (y costosa) entre el planificador del usuario y el planificador del núcleo.

**Ejemplos de implementación híbrida**

* Las activaciones del programador utilizadas por la implementación de la biblioteca de subprocesos POSIX nativos de NetBSD (un modelo M: N en oposición a un modelo de implementación de kernel o espacio de usuario 1: 1)
* Procesos ligeros utilizados por versiones anteriores del sistema operativo Solaris
* Marcel del proyecto PM2.
* El SO de la Tera-Cray MTA-2
* Microsoft Windows 7
* El compilador Haskell de Glasgow (GHC) para el lenguaje Haskell utiliza hilos ligeros que están programados en los hilos del sistema operativo.

**Ejemplos de implementación de fibra**

Las fibras se pueden implementar sin soporte del sistema operativo, aunque algunos sistemas operativos o bibliotecas proporcionan soporte explícito para ellas.

* Win32 suministra una API de fibra (Windows NT 3.51 SP3 y posterior)
* Ruby como hilos verdes
* Netscape Portable Runtime (incluye una implementación de fibras del espacio de usuario)
* Costillas2

Lenguajes de programacion soportados

IBM PL / I (F) incluía soporte para multithreading (llamado multitarea) a finales de los años 1960, y esto continuó en el Optimizing Compiler y versiones posteriores. El compilador IBM Enterprise PL / I introdujo un nuevo modelo de API "thread". Ninguna versión formaba parte del estándar PL / I.

Muchos lenguajes de programación admiten enrutamiento en alguna capacidad. Muchas implementaciones de C y C ++ admiten subprocesos y proporcionan acceso a las API de subprocesamiento nativas del sistema operativo. Algunos lenguajes de programación de nivel superior (y normalmente multiplataforma), como lenguajes Java, Python y .NET Framework, exponen el subproceso a los desarrolladores mientras resumen las diferencias específicas de la plataforma en las implementaciones de subprocesos en el tiempo de ejecución. Varios otros lenguajes de programación y extensiones de lenguaje también tratan de abstraer el concepto de concurrencia y de subprocesos del desarrollador (Cilk, OpenMP, Message Passing Interface (MPI)). Algunos lenguajes están diseñados para el paralelismo secuencial en su lugar (especialmente con GPUs), sin necesidad de concurrencia o hilos (Ateji PX, CUDA).

Algunos lenguajes de programación interpretados tienen implementaciones (por ejemplo, Ruby MRI para Ruby, CPython para Python) que soportan subprocesos y concurrencia pero no ejecución paralela de subprocesos, debido a un bloqueo de intérprete global (GIL). El GIL es un bloqueo de exclusión mutua mantenido por el intérprete que puede impedir que el intérprete interprete simultáneamente el código de aplicaciones en dos o más subprocesos a la vez, lo que limita efectivamente el paralelismo en sistemas de múltiples núcleos. Esto limita el rendimiento principalmente para los subprocesos sujetos al procesador, que requieren el procesador, y no mucho para los vinculados a E / S o vinculados a la red.

Otras implementaciones de lenguajes de programación interpretados, como Tcl que usa la extensión Thread, evitan el límite GIL utilizando un modelo de Apartment donde los datos y el código deben ser explícitamente "compartidos" entre hilos. En Tcl cada hilo tiene en uno o más intérpretes.

Los lenguajes de descripción de hardware de programación basados ​​en eventos como Verilog tienen un modelo de subprocesamiento diferente que admite un número extremadamente grande de subprocesos (para hardware de modelado).

Multihilos en casos prácticos

Una interfaz estandarizada para la implementación de subprocesos es POSIX Threads (Pthreads), que es un conjunto de llamadas de biblioteca de funciones C. Los proveedores de SO son libres de implementar la interfaz como se desee, pero el desarrollador de aplicaciones debe ser capaz de utilizar la misma interfaz a través de múltiples plataformas. La mayoría de plataformas Unix incluyendo Linux soportan Pthreads. Microsoft Windows tiene su propio conjunto de funciones de subproceso en la interfaz process.h para multithreading, como beginthread. Java proporciona otra interfaz estandarizada sobre el sistema operativo host usando la biblioteca de concurrencia Java java.util.concurrent.

Las bibliotecas multihilo proporcionan una llamada de función para crear un nuevo subproceso, que toma una función como un parámetro. A continuación, se crea un subproceso simultáneo que comienza a ejecutar la función pasada y termina cuando la función devuelve. Las bibliotecas de subprocesos también ofrecen funciones de sincronización que permiten implementar funciones de multithreading sin errores de condición de carrera usando mutexes, variables de condición, secciones críticas, semáforos, monitores y otras primitivas de sincronización.

Otro paradigma del uso de subprocesos es el de los conjuntos de subprocesos en los que se crea un número determinado de subprocesos al inicio que luego esperan a que se asigne una tarea. Cuando llega una nueva tarea, se despierta, completa la tarea y vuelve a la espera. Esto evita las funciones relativamente costosas de creación y destrucción de subprocesos para cada tarea realizada y extrae la gestión de subprocesos de la mano del desarrollador de aplicaciones y la deja a una biblioteca o al sistema operativo que mejor se adapta para optimizar la gestión de subprocesos. Por ejemplo, marcos como Grand Central Dispatch y Threading Building Blocks.

En modelos de programación como CUDA diseñados para el cálculo paralelo de datos, una matriz de hilos ejecuta el mismo código en paralelo utilizando sólo su ID para encontrar sus datos en la memoria. En esencia, la aplicación debe diseñarse de manera que cada hilo realice la misma operación en diferentes segmentos de memoria para que puedan funcionar en paralelo y utilizar la arquitectura GPU.

1. ***Preemptive multithreading:*** *el acto de interrumpir temporalmente una tarea que se lleva a cabo por un sistema, sin requerir su cooperación, y con la intención de reanudar la tarea en un momento posterior.* [↑](#footnote-ref-1)