Aula 5

Exercício:

Java e Sistemas de Tempo Real Vale a pena utilizar?

A utilização da linguagem Java em sistemas em tempo real não é difundida por uma série de razões significativas. Estes incluem os efeitos de desempenho não determinísticas inerentes no projeto da linguagem Java, como o carregamento de classe dinâmica, e no Java Runtime Environment (JRE) em si, como o coletor de lixo e compilação de código nativo. A especificação em tempo real para Java (RTSJ) é uma especificação aberta que aumenta a linguagem Java para abrir a porta de forma mais ampla para usar a linguagem para construir sistemas de tempo real. Implementando o RTSJ requer suporte do sistema operacional, o JRE, e a biblioteca de classes Java (JCL).

Requisitos em tempo real

Em tempo real (RT) é um termo amplo utilizado para descrever aplicativos que têm do mundo real requisitos de tempo. Por exemplo, uma interface de usuário lento não satisfaz os requisitos genéricos RT de um usuário médio. A mesma exigência pode ser mais explicitamente formulada como "a aplicação não deve demorar mais do que 0,1 segundos para responder a um clique do mouse." Se a exigência não for cumprida, é um fracasso suave: o aplicativo pode continuar, e o usuário, embora infeliz, ainda pode usá-lo. Em contraste, os aplicativos que devem cumprir estritamente reais requisitos de tempo são normalmente chamados *difícil* aplicativos RT. Uma aplicação controlar o leme de um avião, por exemplo, não deve ser retardada por qualquer razão porque o resultado pode ser catastrófico. O que significa ser uma aplicação RT depende em grande parte de como tolerante a aplicação pode ser a falhas na forma de requisitos de tempo perdido.

Outro aspecto fundamental dos requisitos RT é o tempo de resposta. É fundamental para programadores escrevendo duro ou aplicativos RT suaves para entender a restrição de tempo de resposta. As técnicas necessárias para atender a uma dura resposta de 1 microssegundo são significativamente diferentes das exigidas para atender a uma dura resposta de 100

milissegundos. Na prática, conseguindo tempos de resposta abaixo de dezenas de microssegundos requer uma combinação de hardware e software personalizado, possivelmente sem - ou uma fina - camada do sistema operacional.

Finalmente, os designers de aplicativos RT robustos normalmente precisam de algum nível quantificável de características de desempenho deterministas, a fim de arquiteto uma aplicação para atender aos requisitos de tempo de resposta. Efeitos de desempenho imprevisíveis grandes o suficiente para afetar a capacidade de um sistema para atender aos requisitos de tempo de resposta de um aplicativo torna difícil e talvez até mesmo impossível arquiteto que a aplicação corretamente. Os designers da maioria dos ambientes de execução RT dedicar um esforço considerável para reduzir os efeitos de desempenho não determinísticas para atender as necessidades de tempo de resposta de mais amplo espectro possível a aplicações de RT.

Desafios para aplicativos Java RT

Aplicações Java padrão que funcionam em uma JVM de uso geral em um sistema operacional de propósito geral só pode esperar para atender às exigências macias RT no nível de centenas de milissegundos. Vários aspectos fundamentais da linguagem são responsáveis: gerenciamento de threads, carregamento de classe, Just-in-time (JIT) compilador atividade, e coleta de lixo (GC). Alguns desses problemas podem ser mitigados por projetistas de aplicações, mas apenas com um trabalho significativo.

Gerenciamento de threads

Java padrão não oferece garantias para agendamento de segmento ou de fios de prioridades. Um aplicativo que deve responder a eventos em um tempo bem definido não tem como garantir que outro segmento de baixa prioridade não vai ficar marcada na frente de um fio de alta prioridade. Para compensar, um programador precisaria para particionar um aplicativo em um conjunto de aplicações que o sistema operacional pode, em seguida, executados em diferentes prioridades. Esta separação aumentaria a sobrecarga desses eventos e tornar a comunicação entre os eventos muito mais desafiador.

Classe de carga

A Java-conformant JVM deve atrasar o carregamento de uma classe até que ele é referenciado pela primeira vez por um programa. Carregando uma classe pode levar uma quantidade de tempo variável, dependendo da velocidade do meio (rígido ou outro) a classe é carregada a partir, o tamanho da classe, e as despesas gerais incorridos pelos próprios carregadores de classe. O atraso para carregar uma classe pode geralmente ser tão elevada quanto 10 milissegundos. Se dezenas ou centenas de classes precisam de ser carregado, o próprio tempo de carregamento pode provocar um atraso significativo e possivelmente inesperado. Design da aplicação cuidadosa pode ser usado para carregar todas as classes na aplicação start-up, mas isso deve ser feito manualmente, pois a especificação linguagem Java não permite que a JVM executar essa etapa inicial.

A especificação em tempo real para Java

O RTSJ foi criado para resolver algumas das limitações da linguagem Java que impedem seu uso generalizado em ambientes de execução RT. O RTSJ aborda várias áreas problemáticas, incluindo a programação, gerenciamento de memória, threading, sincronização, tempo, relógios e manipulação de eventos assíncrona.

Agendamento

Sistemas RT precisa controlar estritamente como tópicos são programadas e garantir que eles forem marcados de forma determinística, ou seja, que as roscas são programadas da mesma maneira dado o mesmo conjunto de condições. Embora o JCL define o conceito de prioridade de thread, uma JVM tradicional não é obrigada a cumprir as prioridades. Além disso, implementações não-RT Java geralmente usam um round-robin abordagem de escalonamento de preferência com a ordem de programação imprevisível. Com o RTSJ, verdadeiras prioridades e um agendador preemptivo com prioridade fixa com suporte a herança de prioridade é necessária para tópicos RT. Esta abordagem de escalonamento assegura que o segmento ativo de maior prioridade será sempre a execução e continua a executar até que voluntariamente libera a CPU ou é precedido por uma thread de alta prioridade. A herança de prioridade garante que *inversão de prioridade* é evitada quando um fio de alta prioridade precisa de um recurso mantido por uma thread de baixa prioridade. Inversão de prioridade é um problema significativo para os sistemas RT.

Gerenciamento de memória

Embora alguns sistemas RT pode tolerar atrasos resultantes do coletor de lixo, em muitos casos, estes atrasos são inaceitáveis. Para apoiar as tarefas que não podem tolerar interrupções de GC, o RTSJ define *imortais* e *escopo* áreas de memória para complementar o heap Java padrão. Estas áreas permitem tarefas para usar a memória sem ser obrigada a bloquear se o coletor de lixo precisa libertar memória no heap. Objetos alocados na área de memória imortal são acessíveis a todos os tópicos e nunca são coletados. Porque nunca é coletado, a memória imortal é um recurso limitado que deve ser usada com cuidado. Áreas de memória extensão podem ser criados e destruídos sob o controlo do programador. Cada área de memória âmbito é alocada com uma dimensão máxima e podem ser usados para a atribuição de objeto. Para garantir a integridade de referências entre objetos, o RTSJ define regras que governam como os objetos em uma área de memória (heap, imortal ou extensão) pode se referir a objetos em outras áreas de memória. Mais regras definem quando os objetos em uma memória escopo são finalizados e quando a área de memória pode ser reutilizada. Devido a estas complexidades, o uso recomendado de memória imortal e escopo é limitado a componentes que não podem tolerar pausas do GC.

Tópicos

O RTSJ adiciona suporte para duas novas classes de fio que fornecem a base para a execução de tarefas com o comportamento RT: RealtimeThread e NoHeapRealtimeThread (NHRT). Essas classes fornecem suporte para as prioridades, comportamento periódico, prazos com os manipuladores que podem ser acionados quando o prazo for ultrapassado, eo uso de outros que o heap áreas de memória. NHRTs não pode acessar a pilha e assim, ao contrário de outros tipos de tópicos, NHRTs não são na sua maioria interrompida ou preterida por GC. Sistemas RT normalmente usam NHRTs com prioridades para as tarefas com os requisitos de latência mais apertados, RealtimeThread s para tarefas com requisitos de latência que podem ser acomodados por um coletor de lixo, e encadeamentos Java regulares para tudo o resto. Porque NHRTs não pode acessar a pilha, usando esses segmentos requer um alto grau de cuidado. Por exemplo, até mesmo o uso de classes contêineres do padrão JCL devem ser cuidadosamente geridos para que a classe contêiner não involuntariamente criar objetos temporários ou internos na pilha.

Sincronização

Sincronização deve ser cuidadosamente gerido dentro de um sistema RT para evitar tópicos de alta prioridade de espera de thread de prioridade menor. O RTSJ inclui suporte de herança de prioridade para gerenciar a sincronização quando ocorre, e ele fornece a capacidade para comunicar sem fios para sincronização via ler e escrever filas livre de espera.

Tempo e relógios

Sistemas RT precisa relógios de maior resolução do que as previstas pelo código Java padrão. Os novos HighResolutionTime e Relógioclasses encapsulam estes serviços de tempo.

Manipulação de eventos Asynchronous

Sistemas RT muitas vezes gerenciar e responder a eventos assíncronos. O RTSJ inclui suporte para manipulação de eventos assíncronos desencadeada por uma série de fontes, incluindo temporizadores, sinais do sistema operacional, prazos não cumpridos, e outros eventos definidos pelo aplicativo.