UMA VISÃO GERAL SOBRE THREADS

- Revisão Bibliográfica –

ROSELY SCHEFFER

Universidade Estadual de Maringá - DIN – Departamento de Informática. E-mail: rose.scheffer@grupointegrado.br

RESUMO

Esta pesquisa apresenta uma visão geral sobre threads, baseado em artigos internacionais, fundamentações e conceitos de autores nacionais. As threads são uma parte integral de todo o sistema, por isso a importância de focar o tema, quando estudamos sistemas operacionais.

***Palavras-Chave**: Threads; Sistemas Operacionais*

AN OVERVIEW ABOUT "THREADS"

ABSTRACT

This research shows us an overview about "threads", based on international articles, foundation and national author's concepts. The "threads" are an integral part of a whole system, so it's important to focus on this theme when we are studying operational systems.

***Key-Words**: Threads; Operating Systems*

INTRODUÇÃO

As threads são definidas como processos de pouco peso. São a unidade básica do processador central de um programa sendo um córrego seqüencial da execução dentro de um processo.

Com o intuito de mostrar uma visão geral sobre o assunto, abordaremos conceito, benefícios, utilizações, gerenciamento e características de threads em alguns Sistemas Operacionais, bem como algumas aplicações.

CONCEITO

Processo com entidade própria, com próprio contexto de escalonamento, mas que compartilha a estrutura de dados com seu pai.

Thread ou processo leve é uma unidade básica de utilização de CPU que consiste em: apontador de instruções, conjunto de registradores e espaço de pilhas.

Uma *thread* compartilha com *threads* irmãs: a área do código, a área de dados e recursos do sistema operacional.

Em uma tarefa dotada de múltiplos fluxos de execução, enquanto um fluxo está bloqueado esperando, um outro fluxo na mesma tarefa pode continuar rodando.

Cooperação de múltiplas *threads* em uma mesma tarefa aumenta o *throughput* e *performance*.

O mecanismo de *threads* permite que processos seqüenciais sejam executados paralelamente, apesar de poderem fazer chamadas ao sistema que bloqueiam processos.

Na programação é um processo que faz parte de um processo maior ou programa. Em uma estrutura de dados em forma de árvore, um ponteiro que identifica o nó imediatamente superior (Pai) sendo usado para facilitar o percurso da árvore (CASTRO, 1998).

As threads operam de forma semelhante a processos quanto ao seu estado, podem ser pronto, bloqueado, executando e terminado, apenas uma thread de cada vez em execução na CPU, executa seqüencialmente e pode criar threads filhas.

E se diferem de processos, por exemplo, quando uma thread bloqueia, outra da mesma task pode ser executada, não são independentes umas das outras, podendo invadir o espaço de outra a invalidando, não são protegidas umas das outras.

Threads múltiplas em um único Espaço de Endereçamento podem ser utilizadas para implementar concorrência dentro de um processo. Imagine que você tem um processo que serve requisições de I/O (por exemplo, um servidor lê arquivos ou um sistema de bancos de dados):

a) Quando uma *thread* solicita uma leitura, ela fica bloqueada até que os dados fiquem disponíveis;

b) Um único processo com uma única *thread* não pode executar nada de útil enquanto o disco está sendo acessado;

c) Com *threads* múltiplas, uma *thread* pode continuar a executar enquanto a outra espera pelo I/O;

d) compartilham memória, arquivos abertos e variáveis globais;

e) mantém cópias privadas de pilhas, contador de programa e estado de processador.

CARACTERÍSTICAS

a) Variáveis locais da *thread* são alocadas em memória estática própria

b) Todas as *threads* de um processo compartilham a memória e os recursos desse processo

BENEFÍCIOS

Criar/terminar uma *thread* é mais rápido que criar/terminar um processo

O chaveamento de duas *threads* (do mesmo processo) gasta menos tempo que o chaveamento entre dois processos diferentes

As *threads* de um processo compartilham memória e arquivos e podem se comunicar sem a intermediação (invocação) do núcleo

Aplicações que requerem o compartilhamento de *buffers* (por exemplo, produtores e consumidores) se beneficiam da utilização de *threads.*

UTILIZAÇÕES

Permitir a exploração do paralelismo real oferecido por máquinas multiprocessadores.

Aumentar número de atividades executadas por unidade de tempo (throughput*).*

Aumentar tempo de resposta, possibilidade de associar *threads* a dispositivos de entrada/ saída.

Sobrepor operações de cálculo com operações de entrada e saída.

GERENCIAMENTO DE THREADS

Suspender um processo acarreta a suspensão de todas as suas *threads*, considerando que elas compartilham o mesmo espaço de endereçamento

A terminação de um processo termina todas as *threads* dentro do processo.

Podem ser implementadas em dois níveis diferentes: nível do usuário e nível do núcleo

Nível do Usuário:

Todo o gerenciamento é feito pela aplicação; o núcleo não conhece a existência de *threads; o* chaveamento não requer a intervenção do núcleo (execução em modo privilegiado); o escalonamento é específico da aplicação não passa pelo Kernel; chamadas de sistema normalmente bloqueiam o processo e não pode fazer uso de um processador.

Nível do Núcleo:

O núcleo gerência *threads* (incluindo criação e escalonamento) e o processo mantêm a informação de contexto; o chaveamento de *threads* requer a intervenção do núcleo; se uma thread bloqueia, o núcleo pode executar outra thread do mesmo processo e o núcleo pode escalonar várias *threads* de um mesmo processo em processadores diferentes.

Gerenciamento Misto:

A criação de *threads* é realizada no espaço do usuário, a maior parte do escalonamento e sincronização de *threads* também é feita no espaço do usuário, chamadas bloqueantes das *threads* não bloqueiam o processo e combina as vantagens e minimiza as desvantagens das técnicas puras.

MULTITHREADING

É a execução de vários processos em uma seqüência rápida (multitarefa) dentro de um mesmo programa.

Os sistemas operacionais podem suportar várias *threads* de execução em um único processo / espaço de endereçamento, tais como:

a) MS-DOS suporta uma única thread

b) UNIX suporta vários processos mas somente uma thread por processo

c) Windows NT suporta várias threads por processo

d) JVM único processo, múltiplas threads Windows 2000, Solaris, Linux, Mach, OS/2, suportam vários processos, múltiplas threads.

Vantagens:

O tempo de criação/ destruição de *threads* é inferior ao tempo de criação / destruição de um processo.

Chaveamento de contexto entre *threads* é mais rápido que tempo de chaveamento entre processos.

Como *threads* compartilham o descritor do processo que as porta, elas dividem o mesmo espaço de endereçamento o que permite a comunicação por memória compartilhada sem interação com o núcleo.

WINDOWS NT: PROCESSOS E *THREADS*

Windows NT é um ambiente multi-enfiado em que cada processo contém ao menos uma linha de execução. A maioria de operações longa ocorrerá em uma linha separada do fundo com prioridade baixa, quando o usuário continuar a trabalhar em uma linha separada, normal da prioridade Windows NT também fará exame da vantagem de uma máquina usando processadores múltiplos funcionando cada linha em um processador central separado.

O NT não permite que os programados explorem o paralelismo através de um objeto que seja construído em processos de pouco peso.

Os processos e threads no NT são simplesmente objetos criados e excluídos pelo gerenciador de objetos. Um objeto de processo ou thread contém um cabeçalho com os atributos padrões do objeto. O gerenciador de processos define os atributos armazenados no corpo do objeto de processo e de thread e também fornece os serviços de sistema que recuperam e alteram estes atributos.

THREADS SUN-SOLARIS

Ao contrário do Linux, o Solaris confia pesadamente o ambiente a multithreaded.

As linhas de Solaris são executadas e controladas por uma biblioteca de sistema.

Os processos incluem espaço de endereçamento do usuário, pilha, e o bloco de controle do processo.

As Threads no nível do Usuário são suportadas via biblioteca, o sistema operacional não vê. Threads do Núcleo são as unidades de escalonamento do processador, quando uma thread bloquea o LWP associado bloqueia.

THREADS EM LINUX

As threads em Linux são completamente diferentes da maioria dos outros sistemas operando-se devido à natureza aberta da fonte Linux.

Linux não suporta multithreading, porque estes são mais prováveis de deixar de funcionar. Com a multithreadeds há diversos objetos que são escondidos das aplicações e podem ser compartilhados imediatamente.

O núcleo do Linux copia os atributos do processo corrente para o que está sendo criado. É o procedimento de fork-exec. O Linux, entretanto, prevê uma segunda forma de criação de processos: a clonagem. Um processo clone compartilha os recursos (arquivos abertos, memória virtual, etc.) com o processo original. Quando dois ou mais processos compartilham as mesmas estruturas, eles atuam como se fossem diferentes *threads* no interior de um único processo. O Linux não diferencia as estrutura de dados de *threads* e de processos, e por conseqüência, ambos são tratados indistintivamente por todos os mecanismos de gerência do núcleo. Essa característica é mais visível no escalonamento: *threads* e processos são tratados da mesma forma. A vantagem de criar *threads* está associada ao seu custo de criação (tempo), elas são criadas mais rapidamente que processos, pois não necessitam copiar os atributos do processo original, basta inicializar ponteiros de seu descritor de processos de forma que eles referenciem as áreas já existentes do processo que está sendo clonado.

THREAD JAVA

Todo programa Java tem uma thread que começa a rodar no momento em que o programa é iniciado. Ela é conhecida como a thread principal (*main*) do programa, porque é aquela que começa junto com o programa. A thread principal é importante por dois motivos:

É a thread a partir da qual as threads filhas (c*hild threads*) serão geradas.

É a última thread a encerrar a execução. Quando a thread principal pára, o programa é encerrado.

Em Java é possível lançar várias linhas de execução do mesmo programa. Chamamos a isso de Threads ou MultiThreading. A diferença com os processos e programas acima é que o Java é interpretado. Quem cuida dos vários Threads de um programa é o próprio interpretador Java. Algumas vantagens em relação aos processos:

a) O chaveamento entre os threads é mais rápido que o chaveamento entre processos.

b) A troca de mensagens entre os threads também é mais eficiente.

Java atribui uma prioridade a cada thread, isso determina como a thread será tratada em relação a outras. A prioridade de uma thread é um número inteiro que especifica a prioridade da thread em relação às outras threads. Portanto, como valor absoluto, a prioridade não tem nenhum significado. Ou seja, ao rodar sozinha, uma thread com prioridade mais elevada não rodará mais rápido do que uma thread de prioridade mais baixa. A prioridade é usada para decidir quando será feito o chaveamento de uma thread para outra. Esse procedimento é o que chamamos de chaveamento de contexto (*context switch*). As regras que determinam quando o chaveamento de contexto acontece são bastante simples.

WINDOWS 2000

A unidade de escalonamento do Windows 2000 é o conceito de thread. A cada processo está associado, no mínimo uma thread. Cada thread pode criar outras. Essa organização permite a execução concorrente dos processos, além de possibilitar uma concorrência entre as threads que pertencem a um mesmo processo.

As threads de qualquer processo podem em máquinas multiprocessadoras, ser executadas em qualquer processador. Dessa forma o escalonador do Windows 2000 atribui uma thread pronta para executar para o próximo processador disponível. Múltiplas threads de um mesmo processo podem estar em execução simultaneamente.

Implementa mapeamento uma-para-uma. Cada thread contém: um identificador da thread, conjunto de registradores, pilhas de

usuário e kernel separadas, área de armazenamento de dados privados.

THREADS EM OUTRAS APLICAÇÕES

Real-Time Multiple Video Player Systems

É um sistema que permite exibir vários filmes ou seqüências de vídeos simultaneamente em um mesmo cliente ou em um mesmo computador. É um sistema de monitoramento em empresa ou residência, com várias câmeras posicionadas em pontos estratégicos, que se pode visualizar as imagens de todas estas câmeras ao mesmo tempo em um único computador.

Isso é possível, mas o problema está justamente em conseguir mostrar os vídeos com um bom desempenho e sem atrasos.

Como neste caso há vários processos ou programas rodando ao mesmo tempo, isto tende a degradar o sistema.

Então a sugestão de Chris C.H. Ngam e Kam-Yiu Lam foi a de utilizar duas threads para cada vídeo, uma para receber (receive) os dados das imagens e outra para mostrar esses dados (display), dos vídeos que estiverem sendo exibidos. Além disso, essas threads são executadas utilizando duas técnicas especiais pra acelerar sua execução: 1- Priority Assigment, 2 – Feedback Capture Vídeo.

Quando cliente requisita vários vídeos a um servidor de vídeo, este vai enviar um quadro (frame) de cada vídeo de forma seqüencial, o primeiro quadro de todos os vídeos, o segundo quadro de todos os vídeos, assim sucessivamente até o encerramento. Esse envio seqüencial funciona bem se todos os quadros forem do mesmo tamanho. Mas se o vídeo tiver tamanho diferente, então se aplica o esquema de prioridades atribuídas as threads. Essas threads possuem prioridades de execução, dependendo de alguns fatores, algumas vão executar mais tempos que outras, vão ocupar a CPU durante mais tempo. Aqueles vídeos que estiverem recebendo mais informações, ou mais frames, terão prioridade menor, pois já estará sendo mais rápidos e não precisarão executar mais tempo, e os que estiverem recebendo menos quadros serão os mais lentos e precisarão ficar menos tempo com a CPU, o que significa uma prioridade maior.

E ainda o Feedback Capture Vídeo consiste no cliente enviar um sinal para servidor dizendo ao servidor para diminuir a qualidade do vídeo e, por conseguinte diminuindo também a quantidade de dados enviados para o cliente. Dessa forma, se o cliente não estiver dando conta de receber os vídeos, pode pedir ao servidor para mandar menos informação.

Reducing Pause Time of Conservative Collectors

(Reduzir o Tempo de Pausa em Coletores Conservadores)

Garbage Collector - Programa responsável por verificar quando os programas liberam a memória utilizada por eles, para juntar novamente essa memória para ser usada por outros programas. Os programas ocupam uma quantidade de memória cedida pelo Sistema Operacional, quando terminam sua execução, devolvem a memória para ser usada por outros programas.

O Convervative Garbage Collector, ou Conservador utiliza os métodos convencionais pra recolher a memória não utilizada, mas como esse programa tem que ficar verificando a memória constantemente, ele acaba dividindo o tempo da CPU com os programas (normais) que estão rodando. Logo o garbage vai sofrer pausas. Para diminuir o tempo dessas pausas, Toshio Endo e Kenjiro Taura sugeriram a implementação de várias threads que ficam coletando a memória não usada e devolvendo ao sistema, ou seja, o Garbage Collector não é formado por um único programa, mas por várias threads que ficam recolhendo o “lixo”.

Em um sistema com um único processador, é usada apenas uma thread, mas quando se trata de um sistema com mais de um processador, aí o Garbage Collector executa uma thread em cada processador, justamente para tornar o processo de deslocar memória mais rápido. Em cada processador haverá uma thread independente coletando “lixo”.

Thread pools and work queues

(Grupos de Threads e Filas de Trabalhos)

Brian Goetz sugere como atender aplicações que precisam de muitas threads sendo executadas ao mesmo tempo. Aplicações do tipo Servidor, geralmente utilizam um grande número de Threads, como Servidores Web, de Banco de Dados, de Arquivos e de E-mail.

Isto acontece porque estes servidores recebem várias solicitações de usuários diferentes, querendo conectar-se e executar um serviço. Para cada um dos usuários o sistema cria uma thread para atender estas solicitações e logo após completar a tarefa, as threads são destruídas.

Porém criar e destruir centenas de threads já que os servidores podem criar às vezes milhares de threads por segundo, dependendo da demanda dos usuários, é uma tarefa árdua para o sistema em termos de CPU. Muitas vezes a CPU vai ficar mais tempo criando e destruindo threads do que executando as tarefas solicitadas, como consultar uma homepage ou conectar a um banco de dados.

O autor propõe então utilizar uma abordagem diferente: em vez de criar e destruí threads o tempo todo, o sistema já cria um número razoável delas (100, 1000, 2000, dependendo da aplicação) e não as destrói. Resultado as threads são criadas apenas uma vez, e reutilizadas quantas vezes forem necessárias. Essa grande quantidade de Threads é chamada de Threads Pools.

Outro problema abordado pelo autor de criar e destruir muitas threads, é que este processo consome muita memória podendo levar a uma queda do sistema. Inclusive, o fato de definir A PRIORI um número X de Threads prontas, evita que sejam criadas mais threads do que o sistema suporta, o que levaria à queda do sistema.

Já as Work Queus, ou filas de trabalho, representam as filas de threads que estão disponíveis no Threads Pools. Podendo haver, por exemplo, uma fila de threads em uso, e uma fila de threads disponíveis. Uma vez que uma thread já tenha sido usada, ela volta para a fila de threads disponíveis.

CONCLUSÃO

Observou-se pelos conteúdos estudados que um conceito bastante em voga em sistemas operacionais é o conceito de *threads*. Esse interesse por *threads* está associado com o advento de máquinas multiprocessadoras (SMP) e com a facilidade de exprimir atividades concorrentes. Uma *thread* é usualmente definida como um fluxo de controle no interior de um processo.

Elas são muito úteis quando temos sistemas rodando paralelamente com mais de um processador, porque agilizam a execução de tarefas simultâneas em vários processadores.

Foi possível concluir-se ainda que existem várias maneiras de um sistema operacional executar *threads*. Alguns têm mais facilidade para gerenciar programas com apenas um processo e vários *threads*, do que vários processos e pouco *threads*.

REFERÊNCIAS

BERG. D.J; LEWIS, B.: **Threads Primer: a Guide To Multithreaded Programamming, SunSoft Press**, 1996.

CASTRO, G.; CHAMON, V. **Dicionário de Informática.** 3. ed. Rio de Janeiro: Campus, 1998.

NGAM. Chris C.H.; LAM. Kam-Yiu. **Real-time Multiple Video Payer Systems.**

ENDO. Toshio; TAURA Kenjiro. **Reducing Pause Time of Conservative Collectors**. 2002.

MAIA, Luiz Paulo; MACHADO, Francis B. **Arquitetura de computadores.** 2.ed. Rio de janeiro: LTC,1997.

JONHSON, A.P.; MACAULEY, M.W.S. **High precision Timing within Microsoft Windows: threads, scheduling and system interrupts**. Scotland, UK 2001.

GOETZ. Brian. **Thread Pools and Work Queues,** California, USA. 2002.

Recebido 27 mar. 2007
Aceito 05 ago. 2007