Sistemas Operacionais

Thread

Introdução

- A partir do conceito de múltiplos threads (multithread) é possível projetar e implementar aplicações concorrentes de forma eficiente, pois um processo pode ter partes diferentes do seu código sendo executadas em paralelo.

Ambiente Monothread

- Um programa é uma sequência de instruções, composta por desvios, repetições e chamadas a procedimentos e funções.
- Em um “ambiente monothread”, um processo suporta apenas um programa no seu espaço de endereçamento.

Ambiente Monothread

- Neste ambiente, aplicações concorrentes são implementadas apenas com o uso de múltiplos processos independentes ou subprocessos.
- São exemplos de SO monothread o MS-DOS e as primeiras versões do MS-Windows.

Ambiente Multithread

- Em um “ambiente multithread”, ou seja, com múltiplos threads, não existe a idéia de programas associados a processos, mas, sim, a threads. O processo, neste ambiente, tem pelo menos um thread de execução, mas pode compartilhar o seu espaço de endereçamento com inúmeros outros threads.

Ambiente Multithread

- A grande vantagem no uso de threads é a possibilidade de minimizar a alocação de recursos do sistema, além de diminuir o overhead na criação, troca e eliminação de processos.
- Threads compartilham o processador da mesma maneira que processos e passam pelas mesmas mudanças de estado.

Ambiente Multithread

- Threads são implementados internamente através de uma estrutura de dados denominada “bloco de controle do thread (Thread Control Block – TCB). O TCB armazena, além do contexto de hardware, mais algumas informações relacionadas exclusivamente ao thread, como prioridade, estado de execução e bits de estado.

Ambiente Multithread

- A grande diferença entre aplicações monothread e multithread está no uso do espaço de endereçamento. Processos independentes e subprocessos possuem espaços de endereçamento individuais e protegidos, enquanto threads compartilham o espaço dentro de um mesmo processo. Esta característica permite que o compartilhamento de dados entre threads de um mesmo processo seja mais simples e rápido, se comparados a ambientes monothread.

Ambiente Multithread

- O uso de múltithreads proporciona uma série de benefícios:
  - Programas concorrentes são mais rápidos do que programas concorrentes implementados com múltiplos processos.
  - Como os threads dentro de um processo dividem o mesmo espaço de endereçamento, a comunicação entre eles pode ser realizada de forma rápida e eficiente.
  - Threads em um mesmo processo podem compartilhar facilmente outros recursos.

Ambiente Multithread

- Em ambientes cliente-servidor, threads são essenciais para solicitações de serviços remotos. Em um ambiente monothread, se uma aplicação solicita um serviço remoto, ela pode ficar esperando indefinidamente, enquanto aguarda o resultado. Em um ambiente multithread, um thread pode solicitar o serviço remoto, enquanto a aplicação pode continuar realizando outra atividades.

Ambiente Multithread

- Já para o processo que atende a solicitação, múltiplos threads permitem que diversos pedidos sejam atendidos simultaneamente.

Arquitetura e Implementação

- O conjunto de rotinas disponíveis para que uma aplicação utilize as facilidades dos threads é chamado de “pacote de threads”.
- Threads podem ser oferecidos por uma biblioteca de rotinas fora do núcleo do SO (modo usuário), pelo próprio núcleo do sistema (modo kernel), por uma combinação de ambos (modo híbrido) ou por um modelo conhecido como “schedule activations”.

Arquitetura e Implementação

Ambiente	Arquitetura
Distributed Computing Enviroment (DCE	Modo Usuário
Compaq Open VMS v.6	Modo Usuário
MS-Windows 2000	Modo Kernel
Compaq Unix	Modo Kernel
Compaq Open VMS v.7	Modo Kernel
Sun Solaris v. 2	Modo Híbrido
University of Washington Fast Threads	Scheduler Actvations

Threads em Modo Usuário (TMU)

- São implementados pela aplicação e não pelo SO. Para isso, deve existir uma biblioteca de rotinas que possibilite à aplicação realizar tarefas como criação/eliminação de threads, troca de mensagens entre threads e uma política de escalonamento. Neste modo, o SO não sabe da existência de múltiplos threads, sendo responsabilidade exclusiva da aplicação gerenciar e sincronizar os diversos threads existentes.

Threads em Modo Usuário (TMU)

- A vantagem deste modelo é a possibilidade de implementar aplicações multithreads mesmo em SO que não suportam threads.
- TMU são rápidos e eficientes por dispensarem acessos ao kernel do SO, evitando assim a mudança de modo de acesso (usuário-kernel-usuário).

Threads em Modo Kernel (TMK)

- São implementados diretamente pelo núcleo do SO, através de chamadas a rotinas do sistema que oferecem todas as funções de gerenciamento e sincronização. O SO sabe da existência de cada thread e pode escaloná-los individualmente. No caso de múltiplos processadores, os threads de um mesmo processo podem ser executados simultanemanente.

Threads em Modo Kernel (TMK)

- O grande problema para pacotes em modo kernel é o seu baixo desempenho.

Threads em Modo Híbrido

- Combina as vantagens de threads implementadas em modo usuário e em modo kernel.
- Um processo pode ter vários TMK e, por sua vez, um TMK pode ter vários TMU. O núcleo do sistema reconhece os TMK e pode escaloná-los individualmente.

Threads em Modo Híbrido

- Um TMU pode ser executado em um TMK, em um determinado momento, e no instante seguinte ser executado em outro.
- O programador desenvolve a aplicação em termos de TMU e especifica quantos TMK estão associados ao processo.

Schedule Activations

- Os problemas apresentados no pacote de threads em modo híbrido existem devido à falta de comunicação entre os threads em modo usuário e em modo kernel.
- Este pacote combina o melhor das duas arquiteturas, mas em vez de dividir os threads em modo usuário entre os de modo kernel, o núcleo do sistema troca informações com a biblioteca de threads utilizando uma estrutura de dados chamada “scheduler activations”.