Avaliação Substitutiva

Aluno: Marco Tulio Ferreira

Matrícula: UC22102516

Componente Curricular: Programação Concorrente e Distribuída

Professor: João Robson Santos Martins

Threads são miniprocessos. As menores unidades de processamento que um sistema operacional pode executar. Um tipo de processo dentro de um processo. Um processo pode ter diversos threads dentro dele. Permitem que ocorram múltiplas execuções no mesmo ambiente, com um alto grau de independência uma da outra. Os threads criam a possibilidade de entidades em paralelo compartilharem um espaço de endereçamento e todos os seus dados entre si. Também são mais dos processos e é mais fácil criar ou destruir eles. O uso de threads não resulta em um ganho de desempenho quando todos eles são limitados pela CPU, mas quando há uma computação substancial, os threads permitem que essas atividades se sobreponham, acelerando desse modo a aplicação.

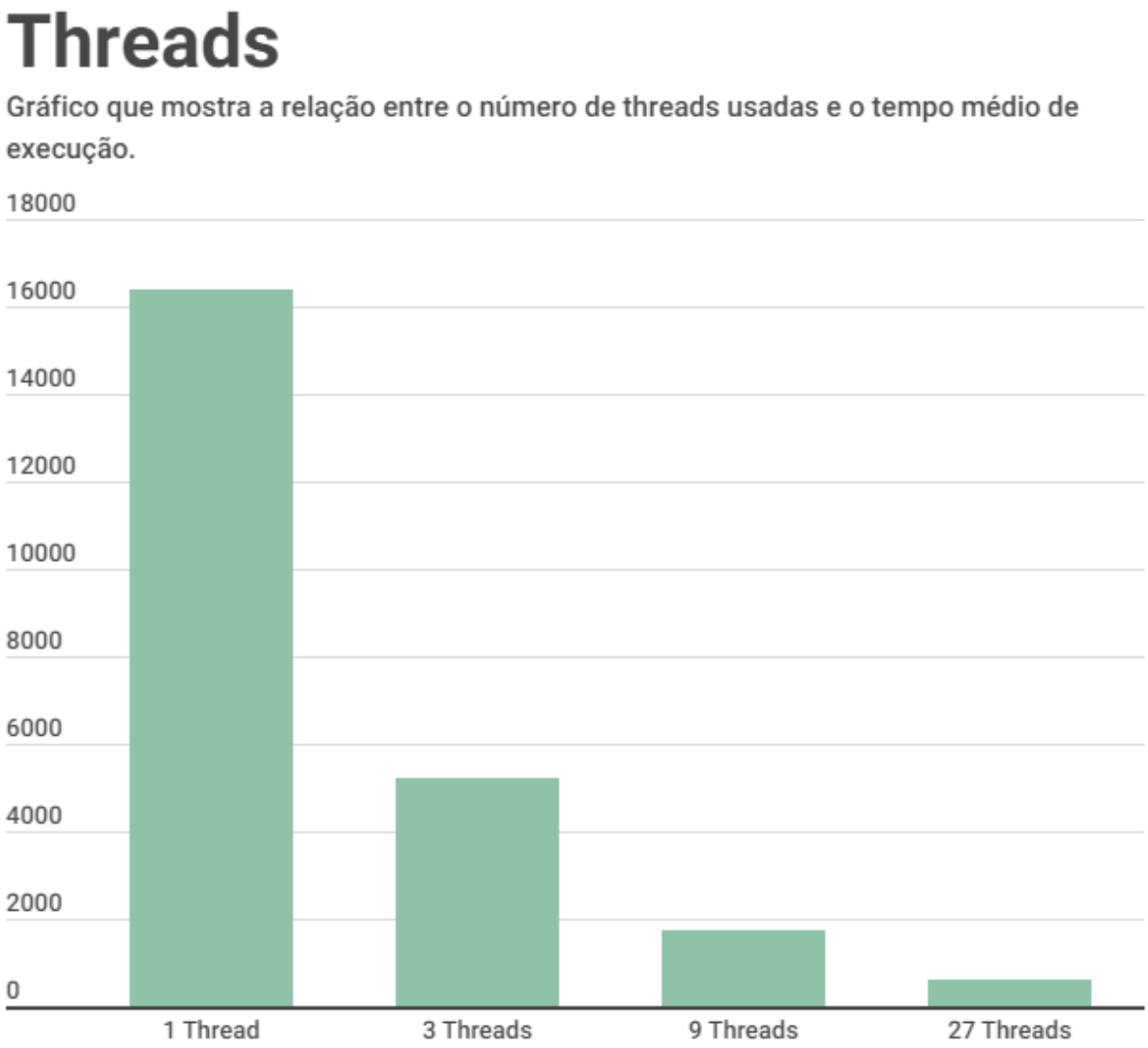
Quando um programa utiliza um único thread, ele processa todas as suas tarefas sequencialmente, como se estivesse manipulando um único arquivo. Isso torna mais fácil realizar operações globais, como buscas e substituições, mas pode ser ineficiente para tarefas que podem ser realizadas em paralelo. Dividir o trabalho em múltiplas threads é como dividir um livro em capítulos ou seções. Cada thread pode tratar de uma parte específica do trabalho de forma independente. Isso permite que diferentes partes do trabalho sejam realizadas simultaneamente, melhorando a eficiência e a responsividade.

Por exemplo, um processador de texto com múltiplas threads poderia ter um thread interativo para lidar com a interface do usuário, um thread de reformatação para atualizar o layout do texto em segundo plano, e um terceiro thread para salvar o arquivo automaticamente em intervalos regulares. Cada thread opera de forma independente, mas compartilham o mesmo espaço de memória, permitindo que o trabalho seja realizado de forma mais eficiente e responsiva.

Por meio do paralelismo presente no funcionamento dos threads, o uso de threads pode diminuir bastante o tempo de execução de um programa, principalmente em programas que envolvem tarefas como entrada e saída de dados. Isso faz com que um programa que usa threads seja mais rápido que um programa que não utiliza esse conceito, onde todas as tarefas são executadas de forma sequencialmente.

A computação concorrente ocorre quando existe a execução de várias atividades de forma intercalada em uma CPU. Em nenhum momento duas atividades são executadas ao mesmo tempo, porém como elas acontecem de forma intercalada, o programa passa a impressão de que estão sendo executadas simultaneamente. Já a computação paralela ocorre quando várias tarefas são executadas de simultaneamente por mais de uma CPU. Isso pode fazer com que a execução

das tarefas seja muito mais rápida. Porém, é importante lembrar da Lei de Amdahl, um código que tem boa parte dele como sequencial limita a capacidade da computação paralela. Por isso, nem sempre é inteligente usar a computação paralela e cabe ao programador decidir baseado no contexto que ele se encontra.



O objetivo do experimento era mostrar como o uso de threads de diferentes tamanhos pode alterar o tempo médio de execução de um programa, especificamente um algoritmo que realiza requisições HTTP. Foram implementadas quatro versões do programa, cada uma utilizando um número diferente de threads: 1 (versão de referência), 3, 9 e 27. Cada versão foi executada 10 vezes e o tempo médio de execução foi calculado.

No gráfico acima, onde o tempo médio de execução dos threads foi medido em milissegundos, podemos ver que o tempo médio de execução diminuiu drasticamente conforme a quantidade de threads aumentaram. O programa com 1 thread (a versão de referência) foi

executado em 16,402 milissegundos, o com 3 threads foi executado em 5,214 milissegundos, o com 9 threads em 1,749 milissegundos e o programa com 27 threads em 603 milissegundos.

Os resultados mostram que a eficiência do uso de threads em programas que envolvem operações como requisições HTTP. Conforme o número de threads foram aumentando, o tempo médio de execução reduziu bastante, pois as requisições passaram a ser processadas de forma paralela. No programa de referência, as requisições foram realizadas de forma sequencial, por isso demorou muito mais tempo que as outras versões do programa.

Com isso, é possível determinar que o uso de threads pode melhorar muito o desempenho de um programa, principalmente se envolve uma tarefa que permite a distribuição da carga de trabalho do algoritmo. Porém, também é importante saber quando que se deve utilizar os threads, pois a implementação de soluções multithreaded deve ser feita com cuidado, utilizando práticas recomendadas para evitar problemas de sincronização e deadlocks.

Referências Bibliográficas:

TANENBAUM, Andrew S.; BOS, Herbert. Sistemas Operacionais Modernos. 4ª ed. Pearson, 2015.

TANENBAUM, Andrew S.; VAN STEEN, Maarten. Sistemas Distribuídos: Princípios e Paradigmas. 2ª ed. Pearson, 2007.