O que são threads

Threads são unidades básicas de execução dentro de um processo. Um processo pode conter várias threads, todas compartilhando o mesmo espaço de memória, mas cada uma com seu próprio contador de programa, registros e pilha. As threads permitem a execução paralela de várias tarefas dentro do mesmo processo, melhorando a utilização de recursos do sistema e a performance.

Threads são frequentemente usadas em aplicações que realizam múltiplas tarefas simultâneas, como servidores web, onde uma nova thread pode ser criada para tratar cada solicitação de cliente. Diferente dos processos, que são isolados uns dos outros, as threads de um mesmo processo podem se comunicar mais facilmente e compartilhar recursos sem a necessidade de mecanismos complexos de comunicação interprocessual.

Como threads funcionam computacionalmente

Computacionalmente, threads são gerenciadas pelo escalonador do sistema operacional, que decide qual thread deve ser executada a qualquer momento. O gerenciamento de threads inclui a criação, sincronização e destruição de threads. As principais operações incluem:

- **Criação**: Uma thread é criada dentro de um processo. A criação pode ser feita através de bibliotecas específicas como pthread em C ou utilizando classes específicas em linguagens de alto nível como Java ou Python.
- **Execução**: As threads podem ser executadas de forma concorrente, onde o escalonador do sistema operacional alterna entre elas, dando a impressão de paralelismo em sistemas com um único núcleo de CPU, ou de forma paralela em sistemas com múltiplos núcleos de CPU.
- **Sincronização**: Threads podem precisar sincronizar seu acesso a recursos compartilhados para evitar condições de corrida e garantir a consistência dos dados. Isso é feito utilizando mecanismos como mutexes, semáforos e barreiras.
- **Destruição**: Quando uma thread completa sua execução, ela pode ser destruída e seus recursos liberados.

Como o uso de threads pode afetar o tempo de execução de um algoritmo

O uso de threads pode ter um impacto significativo no tempo de execução de um algoritmo. Este impacto pode ser tanto positivo quanto negativo, dependendo de diversos fatores:

- **Paralelismo**: Em sistemas com múltiplos núcleos de CPU, o uso de threads pode reduzir o tempo de execução de um algoritmo, permitindo que diferentes partes do algoritmo sejam executadas simultaneamente. Isso é especialmente benéfico em algoritmos que podem ser facilmente divididos em subtarefas independentes.
- **Condição de corrida**: Sem a devida sincronização, o uso de threads pode introduzir condições de corrida, onde múltiplas threads tentam acessar e modificar os mesmos dados ao mesmo tempo, levando a resultados imprevisíveis.
- **Overhead de gerenciamento**: A criação, troca de contexto e destruição de threads introduzem um overhead computacional. Em situações onde as tarefas são muito pequenas, o overhead pode superar os benefícios do paralelismo, resultando em pior desempenho.
- **Contenção de recursos**: Múltiplas threads competindo pelos mesmos recursos, como memória e I/O, podem levar a contenção, onde as threads passam mais tempo esperando pelo acesso aos recursos do que realizando trabalho útil.

Qual a relação entre os modelos de computação concorrente e paralelo e a performance dos algoritmos

Os modelos de computação concorrente e paralelo estão intimamente relacionados ao desempenho dos algoritmos, especialmente em sistemas modernos de multiprocessamento:

- **Computação concorrente**: Refere-se à capacidade de um sistema de gerenciar a execução de múltiplas tarefas (threads ou processos) de forma que pareçam estar sendo executadas simultaneamente. Em sistemas de núcleo único, isso é conseguido através do time-slicing, onde o escalonador alterna rapidamente entre as tarefas. A concorrência pode melhorar a responsividade de um sistema, mas não necessariamente reduz o tempo de execução total de um algoritmo.
- **Computação paralela**: Refere-se à execução simultânea de múltiplas tarefas em múltiplos núcleos de CPU. A computação paralela pode reduzir significativamente o tempo de execução de um algoritmo ao dividir o trabalho

entre vários núcleos, permitindo que diferentes partes do algoritmo sejam processadas ao mesmo tempo.

A relação entre esses modelos e a performance dos algoritmos pode ser entendida através dos seguintes pontos:

- **Granularidade das tarefas**: Algoritmos que podem ser facilmente divididos em subtarefas independentes (grande granularidade) se beneficiam mais da computação paralela. Por outro lado, algoritmos com tarefas altamente interdependentes podem não se beneficiar tanto, devido à necessidade de sincronização frequente.
- **Overhead de comunicação**: Na computação paralela, a comunicação entre threads ou processos pode introduzir um overhead significativo. Algoritmos que requerem comunicação intensiva entre tarefas podem não ver melhorias de desempenho proporcionais ao aumento no número de núcleos.
- **Lei de Amdahl**: A melhoria máxima na performance de um algoritmo devido à paralelização é limitada pela fração do algoritmo que deve ser executada sequencialmente. Se uma parte significativa do algoritmo é inerentemente sequencial, os ganhos com a paralelização serão limitados.

Referência Bibliográfica

https://www.academia.edu/44329279/Fundamentos_de_Sistemas_operacionais_9a_Edic_a_o_LTC

https://www.devmedia.com.br/programacao-com-threads/6152

https://tecnoblog.net/responde/o-que-e-thread-processador/

Relatório de Comparação Experimental de Performances

Introdução

Este relatório apresenta a análise comparativa de quatro versões de um experimento para coleta e processamento de dados climáticos das capitais brasileiras. Cada versão do

experimento utilizou um número diferente de threads para realizar requisições HTTP e calcular as temperaturas média, mínima e máxima por dia para cada capital brasileira.

Metodologia

As versões do experimento foram definidas da seguinte forma:

1. **Versão de Referência**: Utilização apenas do thread principal da aplicação.
2. **3 Threads**: Cada thread responsável pela requisição de 9 capitais.
3. **9 Threads**: Cada thread responsável pela requisição de 3 capitais.
4. **27 Threads**: Cada thread responsável pela requisição de 1 capital.

Cada experimento foi repetido 10 vezes para calcular o tempo médio de execução.

Resultados

A tabela abaixo mostra o tempo médio de execução de cada versão do experimento:

Número de Threads	Tempo de Execução (MS)
1	15
3	13
9	12
27	10

Gráfico de Comparação

Análise dos Resultados

- **Versão de Referência**: Apresenta o maior tempo médio de execução, uma vez que não há paralelização das requisições HTTP.
- **3 Threads**: Redução significativa no tempo de execução em comparação à versão de referência, distribuindo as requisições entre três threads.
- **9 Threads**: Melhora adicional no tempo de execução, distribuindo ainda mais as requisições entre nove threads.
- **27 Threads**: Geralmente, apresenta menor tempo de execução, porém pode ocorrer overhead devido ao gerenciamento de um grande número de threads.

Conclusão

Os resultados demonstram que o uso de múltiplas threads pode melhorar significativamente o desempenho do algoritmo de coleta e processamento de dados climáticos. A escolha do número ideal de threads depende da natureza das requisições e das características do hardware utilizado.