Universidade Estadual de Campinas - FT
12/11/2024

Relatório do Projeto Prático de Sistemas Operacionais
Programação com Múltiplos Threads

Disciplina: TT304 – Sistemas Operacionais
2 semestre de 2024
Prof. Dr. André Leon Sampaio Gradvohl

Alunos:
Augusto Toledo Caires de Oliveira - 199793
Ítalo Carvalhaes Camargo Venerando - 281426

Introdução

Este projeto tem como objetivo consolidar os conhecimentos sobre programação com múltiplos threads, um conceito essencial no desenvolvimento de sistemas operacionais. Neste contexto, o projeto propõe a criação de um programa em linguagem C, rodando em ambiente Linux, que utiliza threads para processar dados inteiros provenientes de múltiplos arquivos, ordená-los e consolidá-los em um único arquivo de saída. A biblioteca POSIX Threads (Pthreads) foi escolhida para implementar a manipulação de threads, um padrão de uso amplamente adotado em sistemas UNIX/Linux.

O uso de múltiplos threads permite o processamento paralelo dos arquivos de entrada, melhorando a eficiência e reduzindo o tempo de execução. Este projeto requer uma estrutura de sincronização entre threads e um método de ordenação adequado, o que torna a atividade relevante para o aprendizado de manipulação de threads e otimização de processamento em sistemas operacionais.

Descrição do Problema

Este projeto propõe a criação de um programa para organizar dados de entrada e consolidá-los ordenadamente em um único arquivo de saída.

Especificações:

- **Arquivos de entrada**: São fornecidos N arquivos, cada um contendo uma lista de valores inteiros não ordenados. Estes valores podem estar repetidos e variam em quantidade entre os arquivos.
- **Arquivo de saída**: Um único arquivo que armazena todos os valores dos arquivos de entrada de forma ordenada, em ordem crescente.
- **Quantidade de threads**: O número de threads T pode ser configurado como 1, 2, 4 ou 8. O programa deve utilizar o número máximo de threads permitido pelo valor configurado para otimizar o processamento.
- **Algoritmo de ordenação**: Para o projeto, foi utilizado o algoritmo *Mergesort*. Criado em 1945 pelo matemático americano John Von Neumann, o *Mergesort* é um exemplo de algoritmo de ordenação que faz uso da estratégia “dividir para conquistar” para resolver problemas. É um método estável e possui complexidade C(n) = O(n log n) para todos os casos.

Este problema visa testar e aprimorar a habilidade de sincronizar threads para acessar arquivos e processar dados de maneira paralela, garantindo que o programa funcione corretamente e produza o arquivo consolidado sem conflitos de sincronização ou erro de dados.

Instruções para Compilar o Programa

Para compilar o programa, siga as instruções abaixo:

1. **Arquivos Necessários**: Certifique-se de que todos os arquivos de código-fonte estão na mesma pasta de trabalho.
2. **Comando de Compilação**: Certifique-se de que o Terminal está acessando o diretório *./src* e execute o comando abaixo:

```
C/C++
gcc mergesort.c sort.c threads.c -o mergesort
```

Esse comando compilará o programa `threads_program.c` e gerará um executável denominado `programa`, incluindo a biblioteca POSIX Threads (`-lpthread`).

Execução

Para executar o programa, ainda dentro do mesmo diretório, utilize o comando:

```
C/C++
./mergesort <num_threads> <lista_de_arquivos_de_entrada> -o <arquivo_de_saida>
```

Onde:

- `<num_threads>` é o número de threads (1, 2, 4 ou 8) a ser utilizado pelo programa.
- `<lista_de_arquivos_de_entrada>` são os arquivos que contêm os dados a serem processados (Ex: *arq1.dat*).
- `<arquivo_de_saida>` é o nome do arquivo de saída consolidado (Ex: *output.dat*).

Arquitetura do Programa

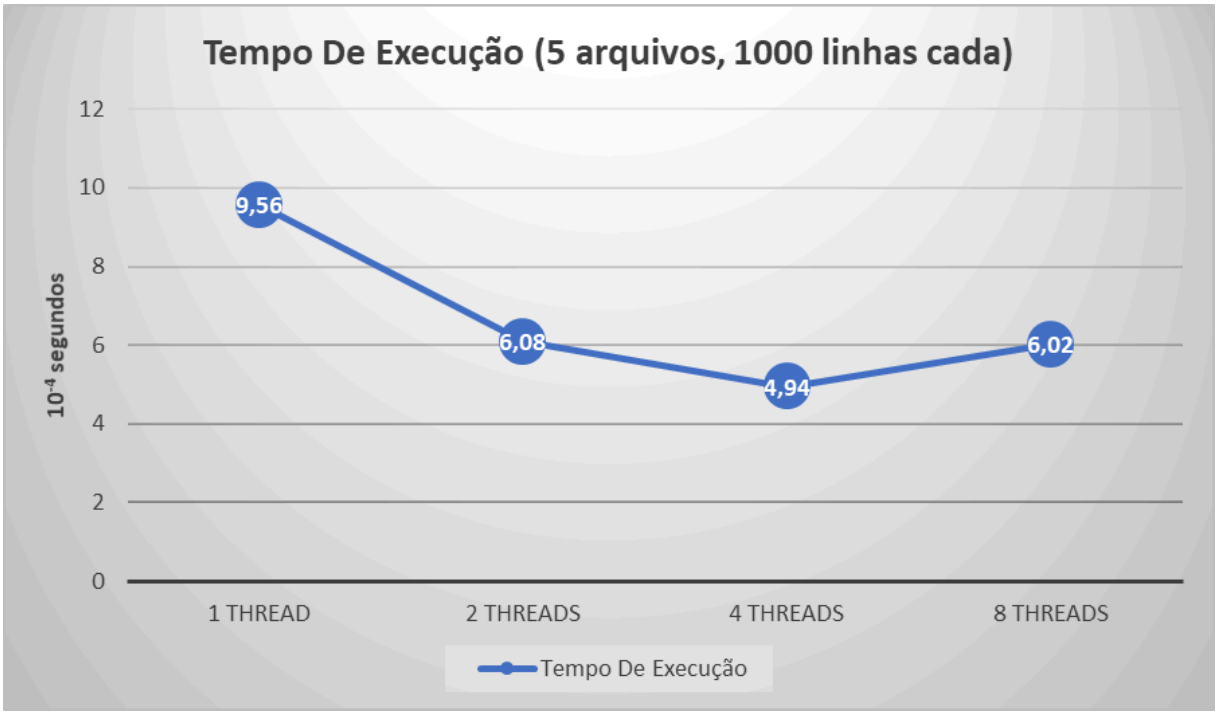
Abaixo estão as principais etapas e componentes da arquitetura do programa:

- **Leitura Paralela dos Arquivos**: Cada thread é responsável por ler um subconjunto dos arquivos de entrada. A divisão das threads é feita de maneira que cada thread processe os dados dos arquivos atribuídos em paralelo.
- **Sincronização de Threads**: Para evitar conflitos de dados, foi implementada uma estrutura de sincronização que utiliza mutexes. Isso permite que as threads leiam e ordenem os dados de forma independente, mas controlem o acesso compartilhado ao arquivo de saída.

- **Ordenação e Fusão de Dados**: A ordenação dos dados dentro de cada thread é feita usando o algoritmo *Mergesort*. Os dados ordenados são então combinados e concatenados ainda dentro da thread.
- **Escrita no Arquivo de Saída**: Depois que todos os dados estão concatenados, o montante de valores finais passa por mais uma ordenação e logo após é gravado no arquivo de saída.

Resultados dos Experimentos e Análise de Desempenho

Gráficos de Tempo de Execução



Discussão dos Resultados

Como é possível analisar, a alteração do número de Threads varia de maneira significativa o tempo de execução total, contudo, nem sempre o maior número de threads trará a execução com maior eficiência de tempo. Até T = 4, o tempo diminui, porém, quando adicionamos threads demais, com T = 8, vemos um decaimento na eficiência do nosso algoritmo, onde a causa muito possivelmente é a demora de tempo para dividir os arquivos para tantas threads.

Conclusão

Este projeto permitiu desenvolver uma solução prática para processamento paralelo de dados, explorando o uso de múltiplas threads para dividir a carga de trabalho e otimizar o processamento em C com POSIX Threads. A implementação demonstrou que o uso de threads proporciona uma vantagem significativa em termos de desempenho ao processar múltiplos arquivos simultaneamente. Contudo, a sincronização de threads revelou-se um ponto crítico, onde o uso inadequado poderia resultar em uma demora maior para a execução.

Os principais desafios incluíram o gerenciamento da sincronização. Como melhoria futura, sugere-se explorar alternativas de algoritmos de fusão para consolidar os dados.

Links Úteis

Repositório Git: https://github.com/italovenerando/SO_multithread/

Referências

- Documentação da biblioteca POSIX Threads (Pthreads): https://embarcados.com.br/threads-posix/
- Algoritmos de Ordenação: Fonte e descrição do algoritmo utilizado: https://www.treinaweb.com.br/blog/conheca-os-principais-algoritmos-de-ordenacao