Relatório do Projeto

Programação Multithread

NOME: Gabriel Matheus Pereira dos Santos **RA:** 281416

Descrição do Problema

Neste trabalho de Sistemas Operacionais, o objetivo é desenvolver um programa em C que abra e leia múltiplos arquivos contendo números inteiros em quantidades variadas. Esses números devem ser ordenados e gravados em um arquivo de saída.

O programa deve criar threads para dividir a ordenação em partes menores e executá-las simultaneamente, visando reduzir o tempo total de processamento. O número de threads (2, 4 ou 8) deve ser definido pelo usuário na linha de comando, assim como os nomes dos arquivos de entrada e saída, eliminando a necessidade de solicitações de dados durante a execução.

Além de gerar o arquivo com os números ordenados, o programa deve calcular e exibir na tela o tempo de execução de cada thread e o tempo total de execução, considerando apenas o tempo gasto na ordenação.

Compilação do Programa

Para uma melhor modularização, dividi o código em quatro partes:

- code.c: Contém a função *int main()* e a função *OrdenacaoVetor*, que será utilizada por cada uma das Threads.
- funcoes.c: Contém as funções principais do programa, como funções de ordenação e funções para ler e criar arquivos. Esse arquivo possui seu *header*: funcoes.h
- funcoesAux.c: Contém as funções auxiliares do programa, que, mesmo realizando partes importantes, não são responsáveis pelo propósito principal do programa. Esse arquivo possui seu *header*: funcoesAux.h
- alocacao.c: Contém funções de alocação dinâmica, já que em diversas partes do código, foi preciso alocar dinamicamente desde inteiros até structs inteiras. Esse arquivo possui seu *header*: alocacao.h

Para compilar todos esses arquivos, seria necessário executar o comando

```
gcc code.c funcoes.c alocacao.c funcoesAux.c –o mergesort -lpthread
```

Entretanto, para deixar a compilação mais fácil, criei um arquivo Makefile. Dessa maneira, para compilar o programa, basta executar o comando *make* no terminal para compilar o programa.

Experimentos

Para os experimentos, considerei a leitura de três arquivos, cada um contendo 1.000 inteiros, totalizando 3.000 números a serem ordenados. Além disso, o programa foi executado com todas as quantidades possíveis de threads (2, 4 e 8), além de uma execução com apenas 1 thread, representando o tempo de execução do programa sem a implementação de multithreading.

- *1 thread*

```
Tempo de execução da thread 0: 0.068803 segundos.
Tempo total de execução: 0.069548
```

- *2 threads*

```
Tempo de execução da thread 0: 0.030961 segundos.
Tempo de execução da thread 1: 0.031565 segundos.
Tempo total de execução: 0.032205
```

- *4 threads*

```
Tempo de execução da thread 0: 0.006155 segundos.
Tempo de execução da thread 1: 0.008338 segundos.
Tempo de execução da thread 2: 0.006132 segundos.
Tempo de execução da thread 3: 0.008433 segundos.
Tempo total de execução: 0.008880
```

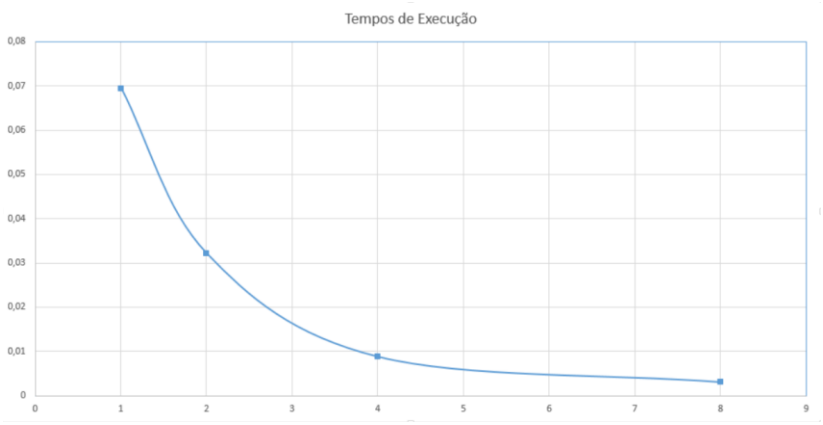
- *8 threads*

```
Tempo de execução da thread 0: 0.002243 segundos.
Tempo de execução da thread 1: 0.001964 segundos.
Tempo de execução da thread 2: 0.002035 segundos.
Tempo de execução da thread 3: 0.002032 segundos.
Tempo de execução da thread 4: 0.002031 segundos.
Tempo de execução da thread 5: 0.002014 segundos.
Tempo de execução da thread 6: 0.002360 segundos.
Tempo de execução da thread 7: 0.002579 segundos.
Tempo total de execução: 0.003149
```

Na próxima página, veremos uma tabela somente com os tempos finais, além de um gráfico representando todos esses tempos, para percebermos as diferenças de maneira mais clara.

Quantidade de Threads	Tempo de Execução (segundos)
1	0.069548
2	0.032205
4	0.008880
8	0.003149

Aqui, já podemos claramente perceber como o uso de threads permitiu uma melhora significativa no tempo de execução do programa. Perceba que, comparando o tempo de execução utilizando 1 thread e 8 threads, o tempo se reduziu em mais de 20 vezes.



Observando o gráfico, é notório como o tempo reduziu de maneira drástica.

Conclusão

Nos exemplos acima, mesmo com um grande número de inteiros a se ordenar, o computador foi capaz de processar tudo de maneira muito rápida. Entretanto, pensando em proporções maiores: dizer que o uso de threads reduziu em 20 vezes o tempo de execução seria o equivalente a dizer que um programa que deveria levar 4 horas para finalizar, conseguiu finalizar em 12 minutos!

Portanto, de acordo com os resultados obtidos, é evidente que o uso de threads é uma ferramenta poderosa que pode reduzir de maneira significativa o tempo de execução de um programa.

LINK VÍDEO: https://youtu.be/Lut4_yy18J0

LINK REPOSITÓRIO GITHUB: https://github.com/gatheus/Threads