Antônio Lima Della Flora

00315056

Laboratório 3 – Multithread

A ideia desse laboratório era implementar um contador de palavras de forma sequencial e paralela e comparar a performance de ambos.

Especificações do computador:

- Modelo: Ryzen 5 3600X
- 6 cores
- 12 threads
- Clock base de 3.8 GHz
- Clock turbo de 4.2 GHz

Implementação sequencial

A versão sequencial do programa implementa uma busca simples de string em arquivo texto. O programa lê o arquivo linha por linha usando fgets() e, para cada linha, utiliza a função strstr() para encontrar todas as ocorrências da string de busca. Um contador é incrementado para cada ocorrência encontrada, e a busca continua na mesma linha a partir da posição seguinte à última ocorrência.

A implementação sequencial é direta e eficiente em sua simplicidade. Por não envolver múltiplas threads, não há necessidade de mecanismos de sincronização ou preocupação com condições de corrida. O acesso sequencial ao arquivo otimiza naturalmente as operações de I/O, resultando em um código previsível e de fácil manutenção.

Implementação paralela

A versão paralela do programa foi desenvolvida utilizando a biblioteca pthread para dividir o trabalho de busca entre múltiplas threads. O programa primeiro lê todo o arquivo para a memória e divide o total de linhas entre as threads disponíveis. Cada thread recebe um segmento específico do arquivo para processar, mantendo seu próprio contador local de ocorrências.

Para evitar problemas de concorrência, a implementação utiliza uma estrutura ThreadArgs que encapsula os dados necessários para cada thread. A divisão do trabalho é feita por linhas completas, e a sincronização ocorre apenas

ao final do processamento, quando os contadores locais são somados para obter o total de ocorrências.

Resultados:

	Tamanho (bytes)	Tempo de execução (1000 buscas)
Sequencial	81 KB	0.09 s
Paralelo – 1 thread	83 KB	0.16 s
Paralelo – 2 threads	83 KB	0.22 s
Paralelo – 4 threads	83 KB	0.38 s
Comando find	82 KB	24.4 s

Os testes revelaram diferenças significativas entre as implementações. A versão sequencial produziu o executável mais compacto com 81 KB, enquanto a versão paralela ocupou 83 KB. O executável que utiliza o comando find do Windows ficou em um tamanho intermediário de 82 KB. Em termos de desempenho, a versão sequencial se destacou significativamente, completando as 1000 buscas em apenas 0,09 segundos.

Contrariando a expectativa, a adição de threads não resultou em melhorias de desempenho. A versão com uma única thread necessitou de 0,16 segundos, representando um aumento de 78% no tempo de execução em comparação com a versão sequencial. Este padrão de piora de performance se intensificou com o aumento de threads: a implementação com duas threads consumiu 0,22 segundos (144% mais lento que o sequencial), e a versão com quatro threads levou 0,38 segundos (322% mais lento). Este comportamento sugere que custo computacional para criar e gerenciar threads supera o benefício potencial da paralelização já que o arquivo de testes é muito pequeno. Posteriormente será feito um teste com um arquivo maior.

A implementação utilizando o comando find do Windows apresentou o desempenho mais inferior, com um tempo de execução de 24,4. Este resultado drasticamente inferior pode ser atribuído ao fato do comando *system* criar um novo processo a cada execução ou ao fato do comando imprimir no console, o que aumenta seu tempo de execução quando repetimos 1000 vezes.

Testando com um arquivo maior:

Como surgiu a hipótese de que a busca com threads só performou pior do que a sequencial devido ao tamanho do arquivo lido, repeti o experimento com um arquivo muito maior. O arquivo utilizado foi um livro do Sherlock Holmes convertido para formato txt com 128 mil linhas e 6 MB.

	Tempo de execução (1000 buscas)
Sequencial	34.2 s
Paralelo – 1 thread	18.07 s
Paralelo – 2 threads	9.74 s
Paralelo – 4 threads	5.14 s
Paralelo – 8 threads	3.7 s
Paralelo – 12 threads	3.5 s
Comando find	8 minutos

Com o arquivo maior, a paralelização demonstrou benefícios substanciais. O desempenho melhorou progressivamente com o aumento do número de threads, até atingir um ponto de saturação em torno de 8-12 threads.

Esse desempenho mostra que a utilização de múltiplas threads pode sim melhorar a performance, mas é necessário que a tarefa realizada seja minimamente demorada, para compensar a demorada causada pelo gerenciamento das threads.

O aumento de performance quase linear até 4 threads indica que para arquivos longos a busca em paralelo é muito superior à busca sequencial.