

Thiago Bazilio e Weberti Silva

RELATÓRIO

Trabalho apresentado Engenharia da Computação da
UNIVALI - Universidade do Vale do Itajaí, para a disciplina
Sistemas Operacionais.
Professor: Felipe Viel

Itajaí - Santa Catarina

Introdução

Este trabalho tem como objetivo apresentar dois projetos, cujo primeiro projeto busca mostrar uma análise comparativa de tempo de processamento utilizando bibliotecas como `time.h`, na qual usou-se multiplicação matricial e posicional para realizar a comparação do sistema single thread e multithread.

No segundo projeto foi feito um algoritmo de ordenação de vetores, onde o vetor é de 200 posições e foi comparado com um sistema de singlethread e multithread.

Projeto 1 códigos importantes e contexto da aplicação para a compreensão dos resultados obtidos.

Foi usado da linguagem c++ para o trabalho, na qual as funções utilizadas para obter os resultados da comparação de single thread e multithread foram:

Alocar Linha: Usado para alocar as linhas das matrizes.

```
//aloca linhas
int** a = new int* [linha];

int** b = new int* [linha];

int** res = new int* [linha];
```

Matricial: Utilizado na multiplicação matricial das matrizes por meio de multithreads.

```
//matricial
void matricial1() {
    for (int i = 0; i < 5000; i++) {
        for (int j = 0; j < coluna; j++) { ... }
    }
}

void matricial2() {
    for (int i = 5000; i < 10000; i++) {
        for (int j = 0; j < coluna; j++) { ... }
    }
}

void matricial3() {
    for (int i = 10000; i < 15000; i++) {
        for (int j = 0; j < coluna; j++) { ... }
    }
}

void matricial4() {
    for (int i = 15000; i < linha; i++) {
        for (int j = 0; j < coluna; j++) { ... }
    }
}
```

Posicional: Utilizado na multiplicação posicional das matrizes por multithreads.

```
//posicional
void posicional1() {
    for (int i = 0; i < 5000; i++) { ... }
}

void posicional2() {
    for (int i = 5000; i < 10000; i++) { ... }
}

void posicional3() {
    for (int i = 10000; i < 15000; i++) { ... }
}

void posicional4() {
    for (int i = 15000; i < linha; i++) { ... }
}
```

Single: Função Singlethread usado na multiplicação.

```
///Single

//matricial
void munica() {
    for (int i = 0; i < linha; i++) {
        for (int j = 0; j < coluna; j++) {
            res[i][j] = a[i][j] * b[j][i];
        }
    }
}

//posicional
void punica() {
    for (int i = 0; i < linha; i++) {
        for (int j = 0; j < coluna; j++) {
            res[i][j] = a[i][j] * b[i][j];
        }
    }
}
```

Resultados obtidos (primeiro projeto)

Simulação: Foi usada duas matrizes de 20.000 por 20.000, que foram preenchidas com números aleatórios entre 1 e 10. O processo mais demorado no projeto foi o preenchimento delas, por conta das 400 milhões de casas para preencher.

```
Console de Depuração do Microsoft Visual Studio
* Matriz 20.000x20.000 *
. . .
. . .
. . .
::Multiplicacao matricial::
Processo multithread demorou 3.626segundos
Processo singlethread demorou 5.066segundos

::Multiplicacao posicional::
Processo multithread demorou 0.294segundos
Processo singlethread demorou 1.017segundos
```

Tabela Comparativa:

Operação ::	single: (s)	multi: (s)	(%)
preencher:	54,892	6,453	850%
matricial:	5,066	3,626	140%
posicional:	1,017	0,294	345%

Conclusão

De acordo com os dados obtidos nesta análise podemos concluir que o modo singlethread se saiu pior, como já esperado. Em comparação, o multithread obteve uma satisfatória melhoria de 140% na multiplicação matricial, que é mais complexa, e de 345% na multiplicação posicional.

Projeto 2 códigos importantes e contexto da aplicação para a compreensão dos resultados obtidos.

O código para sua execução foi utilizado de 3 funções principais, além de seu main. Sendo elas, criar, ordenar e executar.

Função Criar: Foi usada para a criação dos vetores utilizados no projeto.

```
void criar() {  
    /* ... */  
  
    for (int i = 0; i < tamanho; i++) {  
        vetor[i] = (rand() % 99) + 1;  
  
        cout << vetor[i] << " ";  
    }  
}
```

Função Ordenar: Usado na ordenação dos vetores após o início da execução do código.

```
void ordenar(int sta, int end) {  
    for (int i = 0; i < tamanho; i++) {  
        for (int j = sta; j < end; j++) {  
            if (vetor[j] < vetor[i]) {  
                int flex = vetor[i];  
                vetor[i] = vetor[j];  
                vetor[j] = flex;  
            }  
        }  
    }  
    /* ... */  
}
```

Função Executar: Utilizou-se desta função para, as threads do código assim como para a definição do tamanho do vetor.

```
thread o1(ordenar, 0, quarto);  
thread o2(ordenar, quarto, metade);  
thread o3(ordenar, metade, terco);  
thread o4(ordenar, terco, tamanho);  
o1.join();  
o2.join();  
o3.join();  
o4.join();
```

Resultado Multithread:

```
* Vetor de 100.000 *  
Ordenar o vetor com multithread levou 22.799 segundos:
```

Resultado Singlethread:

```
* Vetor de 100.000 *  
Ordenar o vetor com single levou 18.185 segundos:
```

Simulação: Foi usado de exemplo um vetor de 100.000 espaços para ser ordenado, e por mais incrível que pareça, a diferença dessa vez foi oposta ao primeiro projeto.

Tabela Comparativa: em multithread, a ordenação durou 22,799 segundos; em singlethread, a ordenação durou 18.185 segundos.

Conclusão

Sendo assim, podemos analisar que resultados deram positivos para o lado do singlethread, podendo ser por conta da facilidade da tarefa, ou que cada thread atrapalha de certa forma uma às outras. Ao contrário do outro projeto, essa operação fica bem mais simplificada para a arquitetura singlethread.