Laboratório 3

Este laboratório é composto de quatro exercícios que devem ser realizados usando os arquivos fornecidos. São três exercícios obrigatórios e um exercício facultativo. O código deve estar identado, organizado e comentado. A entrega do laboratório deverá ser feita até o dia 06/12 às 23:59, através de um arquivo zip na tarefa do Google Classroom.

Exercício 1: Incremento Atômico de Contador (3 pontos)

Objetivo: Familiarizar-se com a biblioteca de operações atômicas em C++.

Desenvolvimento do Programa: Crie um arquivo `ex1.cpp` e desenvolva um programa em C++ que simule um contador compartilhado entre várias threads, utilizando a biblioteca de operações atômicas.

Instruções:

- Crie um contador global atômico.
- Lance várias threads que incrementam este contador.
- Verifique se o valor final do contador é igual ao número total de incrementos realizados por todas as threads. Note que cada thread deve executar uma operação de incremento.
- Imprima na saída o número de threads utilizado e o valor do contador no formato `Threads: <num_threads> | Contador: <contador>`.

Entrega: O arquivo `ex1.cpp` deve estar em um único arquivo zip com os arquivos dos outros exercícios, e devem ser entregues no Google Classroom.

Exercício 2: Maximização Atômica (3 pontos)

Objetivo: Familiarizar-se com a biblioteca de operações atômicas em C++.

Desenvolvimento do Programa: Crie um arquivo `ex2.cpp` e desenvolva um programa em C++ que crie várias threads que geram números aleatórios e atualizam uma variável para seu valor máximo, utilizando a biblioteca de operações atômicas.

Instruções:

- Crie uma variável atômica para armazenar o valor máximo.
- Lance várias threads que geram números aleatórios e atualizam a variável atômica com o valor máximo encontrado.
- Em cada thread, imprima o ID da thread e o valor aleatório gerado no formato `Thread: <id_thread> | Valor: <valor_aleatorio>`.
- No final do programa, imprima o valor da variável atômica no formato `Variavel Atomica: <var_atomica>`.

Dicas: A biblioteca `rand` do C++ não é `thread-safe`, por isso não utilize-a na geração dos números aleatórios.

Entrega: O arquivo `ex2.cpp` deve estar em um único arquivo zip com os arquivos dos outros exercícios, e devem ser entregues no Google Classroom.

Exercício 3: Cálculo Assíncrono de Soma de Elementos de um Vetor (4 pontos)

Objetivo: Familiarizar-se com as bibliotecas `packaged_task` e `async` do C++.

Desenvolvimento do Programa: Crie dois arquivos (`ex3_1.cpp` e `ex3_2.cpp`) e desenvolva dois programa em C++ que utilizam `std::packaged_task` e `std::async`, respectivamente, para calcular a soma dos elementos de um vetor de forma assíncrona. O vetor deve ser dividido em duas partes, e cada metade deve ser somada em uma tarefa separada, executada assincronamente.

Instruções:

- Inicialização do Vetor:
  - Crie um vetor de inteiros com um tamanho definido (por exemplo, 100 elementos).
  - Preencha o vetor com valores aleatórios ou sequenciais.
- Divisão do Vetor e Cálculo Assíncrono:
  - Divida o vetor em duas partes iguais.
  - Para cada metade do vetor, crie uma tarefa assíncrona usando `std::package_task` ou `std::async` que calculará a soma dos elementos dessa metade.
- Obtenção de Resultados:
  - Aguarde o término das tarefas e obtenha o resultado da soma de cada metade.
  - Some os resultados das duas metades para obter a soma total do vetor.
- Exibição do Resultado:
  - Exiba a soma total do vetor no formato `Soma do Vetor: <soma>`.

Entrega: Os arquivos `ex3_1.cpp` e `ex3_2.cpp` devem estar em um único arquivo zip com os arquivos dos outros exercícios, e devem ser entregues no Google Classroom.

Exercício 4: Merge Sort com Tarefas (Facultativo)

Objetivo: Familiarizar-se com a criação e gerenciamento de tarefas em C++ em um algoritimo de ordenação do Merge Sort.

Desenvolvimento do Programa: Analise o arquivo `ex4.cpp` que contém a implementação serial do algoritmo de ordenação Merge Sort. Seu objetivo é

modificar esse código serial para torná-lo paralelo usando tarefas assíncronas do C++. Você pode utilizar tanto a biblioteca `std::package_task` quanto a `std::async`

Instruções:

- Você receberá o código serial do algoritmo Merge Sort, que já ordena um vetor de inteiros.
- Sua tarefa é modificar esse código para torná-lo paralelo, de forma que a ordenação possa ser acelerada usando tarefas assíncronas.
- A versão paralela do algoritmo Merge Sort deve dividir o vetor em partes menores e utilizar tarefas para ordenar essas partes de forma independente. Em seguida, as partes ordenadas devem ser mescladas de forma adequada.
- Evite condições de corrida e garanta que o número de tarefas criadas não seja muito grande para não prejudicar o desempenho da aplicação.
- Implemente uma função principal que leia a entrada, chame a versão paralela do Merge Sort com o número especificado de threads e aguarde a conclusão de todas elas.
- Teste o programa modificado com diferentes tamanhos de entrada e números de threads para verificar o desempenho da ordenação paralela em comparação com a versão serial. Note que o array dado é muito pequeno, por isso não será possível obter speedup com ele.
- Imprima o vetor ordenado após a conclusão da ordenação para verificar a corretude do algoritmo paralelo.

Entrega: O arquivo `ex4.cpp` deve estar em um único arquivo zip com os arquivos dos outros exercícios, e devem ser entregues no Google Classroom.