INSTITUTO FEDERAL
Espírito Santo
Campus Serra

Curso: Bacharelado em Sistemas de Informação
Disciplina: Sistemas Operacionais
Período: 2023/1
Professor: Flávio Giraldeli

Turma: BSI3
CH: 60 h

Introdução à Programação Multithread com PThreads

Objetivos

- Introduzir o conceito de programação concorrente com múltiplas threads e seções críticas a serem tratadas.
- Analisar comparativamente o desempenho do algoritmo com diferentes números de threads.

Grupos

Deverão ser formados grupos com 2 pessoas (duplas), a serem escolhidos livremente. No caso do número matriculados não ser múltiplo de 2, a decisão deverá ficar a critério do professor.

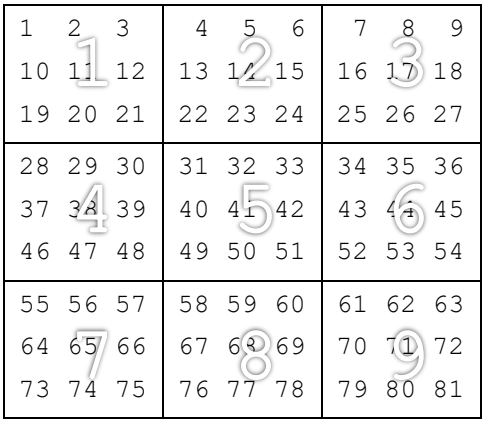
Pontuação

O referido trabalho será avaliado de 0 a 100 e corresponderá a 20% da nota semestral.

Implementação

O objetivo final do algoritmo é: dada uma matriz de números naturais aleatórios (intervalo 0 a 31999) contabilizar quantos números primos existem e o tempo necessário para isso. No entanto, isso será feito de duas formas:

- De **modo serial**, ou seja, a contagem dos primos será feita um a um, um após o outro. Esse será o seu tempo de referência.
- De **modo paralelo**. Para tanto, o trabalho de verificar cada número e se for primo contabilizá-lo consistirá na subdivisão da matriz em "macroblocos" (submatrizes), sem qualquer cópia, baseando-se apenas nos índices.



Tome, como exemplo (ao lado), uma matriz de 9 x 9. Cada macrobloco é composto por 9 elementos (3 x 3). O macrobloco 1 vai da coluna 0 a 2 e da linha 0 a 2, e assim sucessivamente. Os macroblocos serão as unidades de trabalho de cada thread (*paralelismo de dados*, lembra?). Atenção: **Nem a matriz nem os macroblocos deverão ser obrigatoriamente quadradas**. A única exigência é que todos os macroblocos tenham o mesmo tamanho. Além disso, você deve encontrar alguma forma de PARAMETRIZAR essa divisão (usando a diretiva `#define`, por exemplo) a fim de poder efetuar os testes para diferentes tamanhos de macroblocos. **Os macroblocos terão tamanhos que podem variar de desde um <u>único elemento</u> até a <u>matriz toda</u>** (equivalente ao caso serial).

Dito qual é o objetivo final da implementação, segue um passo a passo das diretrizes básicas a serem seguidas pelo algoritmo:

1. Geração de uma matriz de números naturais aleatórios (intervalo 0 a 31999) usando uma semente pré-definida no código, a fim de sempre ter a mesma "matriz aleatória" para todos os testes. A geração de números aleatórios em C se dá com o uso das funções `srand()` e `rand()`. **Essa matriz e a variável que contabiliza o número de números primos encontrados na matriz deverão ser globais (logo, compartilhadas) e únicas. Além disso, a matriz deverá ser alocada dinamicamente (tamanho parametrizado via `#define`).** O tamanho da matriz deverá ser consideravelmente grande a fim de que possam ser efetuadas medidas de desempenho consistentes. O tamanho 10000 x 10000 é um bom começo para os computadores modernos. Mas, **atenção** a essas observações:
    a. Muito cuidado na escolha desse tamanho (dimensões da matriz)! Rode os testes no modo *debug* e, com informações do ambiente de programação (o Visual Studio é muito bom nisso!) e do Gerenciador de Tarefas (no caso do Windows), avalie a quantidade de memória consumida. É fundamental que você se atente para o S.O. não estar fazendo uso significativo do armazenamento secundário como extensão da memória principal ("Memória Virtual"), pois, caso isso aconteça, a medição de tempo não será consistente. Um tamanho razoável é algum que leve a um tempo de busca para o caso serial superior a 5 segundos.

b. Ainda falando sobre as dimensões da matriz, a partir de certo tamanho a quantidade de memória principal consumida será maior que 2 GB e esse é o **limite** para processos de 32 bits no Windows (x86). Isso pode ser facilmente contornado compilando o código nativamente para 64 bits (x64). Novamente, no Visual Studio, isso é um ajuste muito simples feito na própria barra de ferramentas.

2. Efetuar a **Busca Serial** pela quantidade de números primos da matriz gerada acima. Exiba a quantidade de números primos encontrados e o tempo decorrido nessa busca.

   A verificação se um número é primo ou não deve ser feita por uma função com protótipo `int ehPrimo(int n)`, onde `n` é o número a ser verificado. Se ele for primo, a função retorna 1 (true), caso contrário, 0 (false). Atenção: evite fazer essa função de modo "muito otimizado". Não se esqueça: Nós precisamos de uma carga de trabalho grande para testarmos o impacto da paralelização do código. Por outro lado, não implemente essa função de modo tão grosseiro. Dica: teste se há divisores até a raiz quadrada do número. No primeiro momento que encontrar um divisor, já conclua que o número não é primo. Se chegar até o teste `sqrt(número)` e não houver divisor, é porque ele é primo. #Obrigado #DeNada.

3. Efetuar a **Busca Paralela** pela quantidade de números primos na **mesma matriz** usada na busca serial (e utilizando, também, a mesma função de verificação). Cabem, aqui, várias observações:

   a. A escolha do número de threads para o teste principal deverá levar em conta o **número de núcleos físicos** do processador (CPU) que equipa o computador. Se ele tiver N ≥ 2 núcleos reais, crie N threads. Se não souber dessa informação, procure saber de antemão (Você fez AOC com o Giraldeli né? Favor não decepcionar!). Caso o processador possua SMT/HT (*Simultaneous Multithreading*), teste também a quantidade de threads igual a quantidade de núcleos lógicos/virtuais. Faça uma análise dessa comparação: Quantidade de Threads igual ao número de **núcleos físicos** x Quantidade de Threads igual ao número de **núcleos lógicos/virtuais**, estimando, assim, o ganho proporcionado pelo SMT.

   b. A atribuição de qual macrobloco será processado em cada momento deverá ser da seguinte forma: Suponha que foram criadas 4 threads. A thread 1 deverá começar a busca no macrobloco 1, a thread 2 no macrobloco 2, a thread 3 no macrobloco 3 e a thread 4 no macrobloco 4. Devido à natureza aleatória dos números (consequentemente do tempo de verificação se o número é primo ou não depender da magnitude do número) e do escalonador do sistema operacional, nada se pode afirmar sobre que thread terminará sua busca primeiro. No entanto, digamos que a thread 2 termine sua busca primeiro. Ela deverá: somar o número de primos encontrado à variável global que esteja contabilizando o número total de primos da matriz (ou seja, a thread usará uma variável temporária para contar o número de primos e, após terminada a busca no macrobloco, somará esse valor à variável global) e reiniciar a busca no próximo macrobloco que ainda não foi atribuído a nenhuma thread. **A variável que controla quais macroblocos estão livres/alocados deverá ser global (compartilhada)**. Exemplo: Thread 2 termina. Logo, ela verificará se o macrobloco 5 já foi atribuído a alguma thread. Se não, ela "marca-o" como já atribuído e reinicia seus trabalhos nele. Caso contrário, busca pelo próximo macrobloco "livre", até que por fim se esgotem os macroblocos a serem buscados. Neste ponto, a thread termina e a thread principal fica esperando que as demais threads terminem.

   c. **ATENÇÃO**: É proibido criar um vetor para armazenar o número de primos encontrados em cada macrobloco e depois somá-los (ao fim). **Como já mencionado, as variáveis que armazenam o número de primos total, a que controla a alocação do macroblocos e matriz principal deverão ser globais e o acesso compartilhado deverá ser controlado**. **Não fuja das seções críticas**. Trate-as!

   d. Uma vez criadas as N threads, outras não deverão ser criadas. As mesmas threads deverão "buscar trabalho" em outros macroblocos livres, conforme mencionado acima. Quando não houver mais trabalho a ser feito pela thread, permita ela seja encerrada naturalmente.

   e. Faça testes com macroblocos de tamanhos diferentes, entre os extremos: um único elemento e a matriz toda. Anote esses valores, pois serão usados no relatório.

   f. Forneça uma forma prática e fácil de testar o código em *single-thread*, *multi-thread* ou ambos. O professor precisará fazer todos esses testes ao corrigir seu trabalho.

VAMOS FALAR DE PLÁGIO E IA?

Então... todo professor honesto deseja que seus(suas) alunos(as) também tenham honestidade no processo de aprendizagem. Essa honestidade é fundamental para que o professor identifique possíveis deficiências na aprendizagem e possa intervir para tentar solucionar. Em termos mais simples, eu, Giraldeli, preciso saber se você aprendeu ou não os conceitos abordados nesse trabalho. E estou mensurando isso através de uma tarefa (construção de um algoritmo que solucione um problema). Se você abre mão de exercitar seu cérebro e busca soluções mais "fáceis", perde-se completamente o sentido do trabalho.

Hoje há duas formas básicas de você cometer essa desonestidade nesse trabalho: buscando trabalhos de turmas anteriores e/ou consultando diretamente ferramentas de IA (como o chatGPT). Sobre os trabalhos anteriores, é bem simples: eu tenho TODOS arquivados. Já a questão do uso indiscriminado de **ferramentas de IA**, a coisa é mais complicada, já que é difícil afirmar que você a **usou extensivamente no seu trabalho**, o que, acho quase desnecessário dizer, **é proibido**! O que farei nesse caso é também simples: **caso eu desconfie**, por alguma razão (experiência na disciplina, talvez?), que a sua solução não partiu do seu esforço direto, **a dupla será convocada para uma entrevista presencial**. Nessa entrevista eu farei perguntas e possivelmente solicitarei algumas modificações "ao vivo" no código apresentado. Caso seja observada a incapacidade da dupla diante desse cenário, a nota será considerada zero. Caso a dupla se negue a comparecer nessa entrevista, a mesma decisão será tomada: nota zero.

Professor... agora você me deixou preocupado(a)! 👀 Se você não está sendo desonesto(a), você não tem com que se preocupar. Prometo.

Por fim... uma última coisa: isso não é um desafio. Mas, se assim você quiser, eu não vou fugir.

RELATÓRIO

Seja criativo! Monte tabelas, gráficos, etc. Avalie/critique o máximo possível os resultados dos testes. Faça conjecturas e verifique, com testes, se elas estão certas.

Elabore gráficos relativo ao **tempo de processamento** versus diferentes **números de threads** e **tamanhos de macroblocos**, conforme citado anteriormente. Você encontrará resultados bem interessantes quando o tamanho dos macroblocos for muito grande ou muito pequenos! 😉 Mas, lembre-se de testar isoladamente a influência de cada um desses parâmetros. Igualmente, verifique se há algum ganho/perda quando o número de threads for maior que o número de núcleos de processamento.

Testes e Análises obrigatórias

- Mensure o *speedup* ao rodar em múltiplas threads. Analise se esse resultado está coerente com o que a Lei de Amdahl prevê. Mas atenção a diferença entre núcleos físicos e núcleos lógicos (ou virtuais)! O speedup deve ser calculado em cada caso e claramente identificado no relatório do trabalho.
- Teste em dois computadores bem diferentes, tal qual feito em AOC. Compare o desempenho.
- Aumente **muito** o número de threads (algumas centenas ou mais) a fim de que o overhead possa realmente ficar crítico e analise os resultados. Nesse caso, mantenha fixo o tamanho do macrobloco.
- Remova os mutexes que protegem as Regiões Críticas. Isso mesmo... remova as proteções temporariamente, rode o programa com macroblocos pequenos (por quê?) e observe os resultados.

CONCLUSÃO

Pronto, chegou a hora de você responder da maneira mais abrangente possível a simples questão:

O que você pode aprender com esse trabalho?

Capriche na elaboração da resposta.

ENTREGA

A data limite para entrega do trabalho, via seção correspondente no AVA, é o dia **14/06 (quarta-feira)** até as **23:59**.

O trabalho deverá ser entregue comprimido em zip/rar (Relatório em PDF + Arquivo fonte único em .c), com a nomenclatura **[SO 2023-1] Trabalho 1 - Aluno1, Aluno2**. Atenção: É necessário apenas o código fonte, não o projeto todo.

Siga estritamente as regras acima, ou seu trabalho poderá ser desclassificado. Principalmente as normas de nomenclatura. O objetivo é facilitar a correção/análise por parte do professor.

Bom Trabalho!