Sistemas Operacionais I
Programando com Múltiplas Threads

Francisco José da Silva e Silva
Departamento de Informática, UFMA

2º Semestre de 2019

Objetivo, Datas e Observações

- Objetivo: Desenvolver a habilidade de programação com múltiplas threads.
- **Data limite para a entrega: 20 de outubro**.
- Peso no processo de avaliação da disciplina: 30% da segunda nota da disciplina.
- Dúvidas podem e devem ser tiradas com o monitor: **Rodolfo Sobreira Alves / rodolfosalves@lsdi.ufma.br / 981578730 (WhatsApp/Telegram)**. Se sintam a vontade para entrar em contato e agendar um contato presencial.

1 Implementação

O objetivo do programa a ser desenvolvido é: dada uma matriz de números naturais aleatórios (dentro do intervalo 0 a 29999) verificar quantos números primos existem e contabilizar o tempo necessário para realizar esta computação. Isso deve feito de duas formas:

1. De modo serial, ou seja, a contagem dos primos deve ser feita um a um (um número após o outro). O tempo para realizar essa computação será considerado seu tempo de referência.
2. De modo paralelo. Para tanto, o programa deve subdividir a matriz em "macroblocos"(submatrizes) sem efetuar cópias da mesma, baseando-se apenas em índices. A verificação dos números primos de cada submatriz deve ser realizada por uma thread independente. Um exemplo é ilustrado na Figura 1.

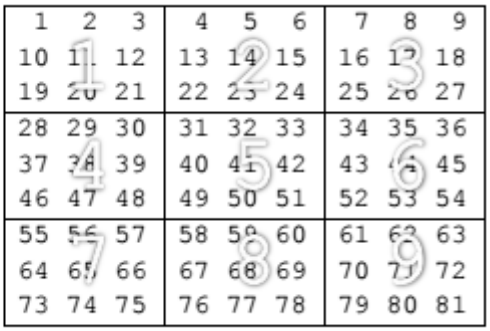

Figura 1: Exemplo de Matriz

A matriz ilustrada na figura é de tamanho 9 x 9 e cada macro-bloco é composto por 9 elementos. O macro-bloco 1 vai da coluna 0 a 2 e da linha 0 a 2, e assim sucessivamente. Os macro-blocos serão as unidades de trabalho de cada thread. Atenção: Nem a matriz nem os macro-blocos deverão ser obrigatoriamente quadradas. A única exigência é que todos os macro-blocos 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 macro-blocos. Os macro-blocos terão tamanhos que podem variar desde um único elemento até a matriz toda (equivalente ao caso serial).

A matriz e a variável que contabiliza o número de números primos encontrados deverão ser globais (logo, compartilhadas entre as threads) e únicas.

Diretrizes básicas a serem seguidas:

- Geração de uma matriz de números naturais aleatórios (intervalo 0 a 29999) 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 pode ser realizada com o uso das funções `srand()` e `rand()`.

- O tamanho da matriz deverá ser consideravelmente grande a fim de que possam ser efetuadas medidas de desempenho consistentes. Para efeito de comparação, em um desktop com 8 GB de RAM e CPU core i5 com 4

  núcleos reais, a verificação serial de uma matriz de 20000 x 20000 levou aproximadamente 42 segundos.

- Para contabilizar o tempo de processador gasto utilize uma ferramenta de *profile*, como o GProf (`https://www.ibm.com/developerworks/`

br/local/linux/gprof_introduction/index.html)

- **Busca serial** pela quantidade de números primos da matriz. Exiba a quantidade de números primos encontrados e o tempo decorrido nessa busca.

- **Busca paralela** pela quantidade de números primos na mesma matriz usada na busca serial. Cabe aqui, algumas dicas:

  A escolha do número de threads para o teste principal deverá levar em conta o número de núcleos reais da CPU que equipa o computador. Se a CPU tiver N ¿= 2 núcleos reais, crie N threads.

  A atribuição de qual macrobloco será processado em cada momento deverá ser feita 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 a natureza aleatória dos números (e 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 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 o marca como já atribuído e reinicia seus trabalhos nele. Caso contrário, busca pelo próximo macrobloco livre, até que por fim se esgote os macroblocos a serem buscados. Neste ponto, a thread termina e a thread principal fica esperando que as demais threads terminem.

  Atenção: É proibido criar um vetor para armazenar a quantidade de números primos encontrados em cada macrobloco e depois somá-los. 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.

2 O que enviar ?

A entrega do trabalho deve ser realizada exclusivamente através do sistema SIGAA. Deve-se enviar um arquivo compactado (tar.gz ou zip) con-

tendo:

- Código fonte do programa implementado
- Um relatório no formato PDF, que deve conter:

  Como compilar e executar seu código;

  Bugs encontrados e não tratados;

  Dificuldades enfrentadas;

  Avaliação do seu Trabalho: Seja Criativo :)

Avalie/critique os resultados dos testes, faça análises relativas ao tempo de processamento considerando as diferentes quantidades de threads utilizadas (Aumente muito, algumas centenas ou mais, o número de threads a fim de que o overhead possa realmente ficar crítico e analise os resultados) e tamanhos de macro-blocos. Você encontrará resultados bem interessantes quando o tamanho dos macroblocos for muito grande ou muito pequenos! Em resumo, brique com seu trabalho, e analise: o que você pôde aprender com esse trabalho?

Boa Sorte :)