INSTITUTO FEDERAL DE MINAS GERAIS

Bacharelado em Ciência da Computação

Sistemas Operacionais I

Trabalho Prático 1

Professor: Raí Caetano de Jesus

Formiga-MG
23 de setembro de 2018

Sumário

1 Introdução

Este documento descreve a especificação do Trabalho Prático 1 da disciplina de Sistemas Operacionais I, que trata do tema multiprogramação e *multithreading*. O trabalho pode ser feito em grupo de até 2 alunos, cuja formação deve ser comunicada com antecedência para o professor da disciplina. O trabalho prático tem o valor de 10 pontos.

O tema do trabalho é programação com a biblioteca PThreads, do POSIX, para resolver um problema de maneira concorrente, usando mais de uma *thread*. Para este propósito o grupo deverá implementar uma multiplicação de matrizes *multithreading*, adaptada neste documento do livro do Silberschatz. Trata-se de um projeto com objetivos didáticos apenas, que não considera questões sobre desempenho e outros aspectos da implementação.

2 Especificação

2.1 Objetivo

Este projeto de programação consiste em implementar uma aplicação `prod-matriz.c` que recebe por linha de comando dois nomes de arquivo, cada qual contendo uma matriz de números reais. O primeiro arquivo refere-se à matriz A_{MK}, com M linhas e K colunas; o segundo arquivo refere-se à matriz B_{KN}, com K linhas e N colunas. Após compilada, a aplicação será executada pela seguinte sintaxe:

```
user@machine$ ./prod-matriz <matrizA> <matrizB>
```

O objetivo da aplicação é calcular o produto das matriz $C_{MN} = A_{MK} \times B_{KN}$, onde cada elemento C_{ij} é dado pela soma dos produtos dos elementos da linha i com a coluna j, isto é

$$C_{ij} = \sum_{k=1}^{K} A_{ik} \times A_{kj}$$

2.2 Entrada e Saída

Os arquivos de entrada são arquivos ASCII e devem ter o formato dado pelo exemplo a seguir, com os elementos da matriz separados por linha.

```
<L>
<C>
<a11> <a12> <a13> ... <a1C>
<a21> <a22> <a23> ... <a2C>
...
<aL1> <aL2> <aL3> ... <aLC>
```

onde `<L>` é o número de linhas da matriz, `<C>` é o número de colunas da matriz, e cada elemento `<aij>` corresponde à um número real em ponto flutuante indexado na linha `i`, coluna `j`.

Para exemplificar, considere a matriz abaixo que deve ser armazenada no arquivo de entrada, digamos matriz01.txt:

$$\begin{bmatrix} -3 & 1 & 7 & 0.5 \\ 2 & 1 & -3 & -10 \\ 5 & 4 & 2 & 4 \end{bmatrix}$$

O arquivo de entrada `matriz01.txt` correspondente a matriz supracitada ficaria como:

```
3
4
-3 1 7 0.5
2 1 -3 -10
5 4 2 4
```

A verificação de erros deve ser feita pela aplicação quanto ao tipo de dados esperado e dimensões (lembre-se que nem todo produto de matriz é possível, pois depende da dimensão de cada fator deste produto). Caso sejam encontrados problemas na entrada de dados, a aplicação deverá liberar os recursos alocados, informar o usuário e encerrar usando o comando `exit()`.

A aplicação deverá alocar memória dinamicamente para receber os elementos das matrizes A e B, assim como reservar memória para guardar os elementos da matriz produto C.

Para cada elemento c_{ij} da matriz C, a aplicação deverá iniciar uma *thread* separada e independente, que irá computar os produtos e somatório. Para matrizes com as dimensões especificadas devem ser criadas, portanto, $M \times N$ *threads*. No fim do processamento, a *thread* principal (i.e., da aplicação) deverá gerar um arquivo de saida de nome resultado.txt com a matriz C resultante, seguindo o mesmo formato de arquivo de entrada descrito acima. Além disso, a matriz C também deve ser exibida no terminal, no mesmo formato armazenado em arquivo.

Para este problema ser implementado corretamente usando *multithreading*, algumas questões devem ser respondidas:

1. Quais parâmetros cada *thread* deverá receber para calcular o elemento c_{ij}?
2. Quais estruturas de dados precisam estar compartilhadas entre as *threads*?
3. Quando que a *thread* principal (da aplicação) pode exibir o resultado computado?

Estas questões devem ser endereçadas pelos membros do grupo. Para auxiliar, as próximas seções dão algumas dicas de possíveis respostas para estas questões.

2.3 Estruturas de dados

O grupo deverá pensar sobre as estruturas de dados a serem usadas pela aplicação em pelo menos duas situações: (i) o que deve ser compartilhado entre as *threads*, como variáveis globais; (ii) o que deve ser informado para cada *thread* em termos de parâmetros.

Primeiramente, cada *thread* irá precisar consultar elementos das matrizes A e B. Portanto, é necessário que as estruturas de dados que implementam as matrizes numéricas sejam visíveis pelas *threads*.

Em segundo lugar, cada *thread* tem a função de calcular cada elemento c_{ij} da matriz. Portanto, os índices da linha e coluna do elemento devem ser informados para que a *thread* saiba onde os laços do cálculo do somatório e dos produtos devem ser executados (ver exemplo `exemplo-threads.zip` no portal meuIFMG ou *classroom* da disciplina para descobrir como passar e recuperar estruturas de dados para uma *thread*). Outra questão que deve ser levantada pelo grupo é se os parâmetros serão criados dinamicamente, então que *thread* cria os parâmetros e que *threads* os liberam da memória quando não tiverem mais utilidade.

2.4 Encerramento das *threads*

Por tratar-se de uma execução concorrente e colaborativa, pode acontecer de uma *thread* ser encerrada antes das demais. Na prática o trabalho de calcular o produto das matrizes está dividido entre várias threads independentes, que são escalonadas pelo *kernel* de acordo com suas regras de prioridade e algoritmos de escalonamentos. O programador não tem controle sobre a execução destas *threads*, mas necessita aguardar que todas se encerrem para exibir a matriz resultante em tela e guardar seus elementos em arquivo. A chamada `pthread_join()` serve a este propósito.

2.5 POSIX PThreads

A aplicação deverá ser implementada na linguagem C, com uso da biblioteca Pthreads (`<pthread.h>`) do POSIX (para mais detalhes, leia os livros da referência básica da disciplina e/ou consulte as páginas de manuais do Linux, usando o comando man pthreads). O aplicativo deverá ser compilado no compilador gcc presente em distribuições Linux.

3 Entrega

O trabalho deverá ser entregue no prazo máximo de 15 dias contados a partir da sua divulgação em sala de aula ou email. Trabalhos entregues fora do prazo serão desconsiderados. O grupo deverá enviar o código-fonte na linguagem C e alguns arquivos contendo matrizes multiplicáveis, todos compactados em um único arquivo no formato `.zip`, e enviados para o *classroom* da disciplina.

4 Barema

O trabalho tem valor de 10 pontos, distribuídos da seguinte maneira:

Critério	**Pontuação**
Entrada e Saída	1,0
Estrutura de dados compartilhadas/passagem de parâmetros	1,0
Criação adequada das threads	4,0
Encerramento adequado das threads	4,0
Total	10 pontos

Entretanto, outros itens poderão ser avaliados de maneira *ad hoc* pelo professor, como comentário de código, modularização, corretude dos resultados, etc. Trabalhos realizados por terceiros ou plagiados valerão zero.