Departamento de Computação e Matemática (DCM)
Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto (FFCLRP)
Universidade de São Paulo (USP)

Segunda Lista de Exercícios de Sistemas Operacionais

1) Escreva um programa chamado *ReverseHello.java*, que cria um thread (“Thread 1”). Este thread cria um segundo thread (“Thread 2”) e assim sucessivamente até chegar ao último thread (“Thread 50”). Cada thread deve imprimir “Olá da Thread <NRO>!!!”, onde NRO corresponde ao número do thread (valor entre 1 e 50). Seu programa deve ser estruturado de modo que os threads exibam suas mensagens na ordem reversa, isto é, o thread pai deve esperar pelo término do thread filho para exibir sua mensagem.

2) Escreva um programa em Java que realize a multiplicação de duas matrizes *A* e *B*, onde *A* contém *M* linhas e *K* colunas e *B* contém *K* linhas e *N* colunas. A *matriz produto* de *A* e *B* é *C*, onde *C* contém *M* linhas e *N* colunas. A entrada na matriz *C* para a linha *i* coluna *j* (C*i,j*) é a soma dos produtos dos elementos para a linha *i* na matriz *A* e coluna *j* na matriz *B*, ou seja:

$$Ci\,j\ = \sum_{n=1}^{k} \quad A_{i,n} \times B_{n,j}$$

Por exemplo, considere *A* uma matriz 3 por 2 e *B* uma matriz 2 por 3, neste caso o elemento $C[3,1]$ seria a soma de $A_{1,1} \times B_{1,1}$ e $A_{1,2} \times B_{2,1}$. Calcule cada elemento C*i, j* em um thread separado. Isso envolverá a criação de M x N threads. Você deve esperar pelo término de todos os threads para escrever a matriz produto C.

3) Escreva um programa Java que calcule o determinante de uma matriz A de ordem n >= 3. Lembre-se que:

i) o determinante de uma matriz *A* de ordem n=2 pode ser calculado como a diferença entre o produto dos termos da diagonal principal e o produto dos termos da diagonal secundária:

$$det(\mathrm{A}) = det\begin{bmatrix} a & b \\ c & d \end{bmatrix} = ad - bc$$

ii) o determinante de uma matriz de ordem n >= 3 pode ser calculado utilizando o Teorema de Laplace:

$$det(A) = \sum_{j=1}^{n} (-1)^{i+j}.a_{ij}.\det(A\_i\_j)$$

onde n é o número de linhas da matriz, *i* é a posição em relação às linhas (qualquer inteiro fixado entre 1 e n), *j* é a posição em relação às colunas e $det(A_{-i\,-j})$ é o determinante da submatriz que exclui a linha *i* e a coluna *j*.

Exemplo com uma matriz 4x4:

Seja a matriz com 4 linhas e 4 colunas

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

Aplicando a fórmula mencionada acima, temos:

$$det(A) = \sum_{j=1}^{4} (-1)^{1+j}.a_{1j}.det(A_{-1-j})$$

Desenvolvendo o determinante pela primeira linha obtemos:

$$\det A = a_{11}.(-1)^{1+1}.\det\left(A_{-1,-1}\right) + a_{12}.(-1)^{1+2}.det\left(A_{-1,-2}\right) + a_{13}.(-1)^{1+3}.det\left(A_{-1,-3}\right) + a_{14}.(-1)^{1+4}.det\left(A_{-1,-4}\right),$$

onde $A_{-i,-j}$ representa a matriz obtida a partir de *A*, com a retirada da *i*-ésima linha e da *j*-ésima coluna.

$$\det A = a_{11}.(-1)^2.\begin{vmatrix} a_{22} & a_{23} & a_{24} \\ a_{32} & a_{33} & a_{34} \\ a_{42} & a_{43} & a_{44} \end{vmatrix} + a_{12}.(-1)^3.\begin{vmatrix} a_{21} & a_{23} & a_{24} \\ a_{31} & a_{33} & a_{34} \\ a_{41} & a_{43} & a_{44} \end{vmatrix} + a_{13}.(-1)^4.\begin{vmatrix} a_{21} & a_{22} & a_{24} \\ a_{31} & a_{32} & a_{34} \\ a_{41} & a_{42} & a_{44} \end{vmatrix} + a_{14}.(-1)^5.\begin{vmatrix} a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{vmatrix}$$

O valor do determinante de uma matriz A deve ser calculado por um thread independente do programa principal. A matriz A deve ser passada como parâmetro pelo construtor da classe

deste thread. O valor do determinante deve ser calculado no corpo do método *run*. Assim, este thread (classe) pode tanto armazenar o valor calculado para o determinante de A em um objeto também passado como parâmetro no construtor ou disponibilizar um método para retornar este valor. Sempre que necessário, um thread que realiza o cálculo do determinante deve criar novos threads filhos para calcular o determinante das submatrizes. Neste caso, o thread pai deve esperar pelo término do cálculo do determinante de todas as submatrizes para calcular seu próprio determinante.

4) Escreva um programa C utilizando threads que receba via linha de comandos um conjunto de números inteiros positivos (*long int*) e verifique, usando um thread separado para cada número, se cada um destes é ou não primo. O thread pode tanto verificar se o número recebido como argumento é ou não primo e escrever este resultado ou retornar esta informação para o thread principal (pai) que então irá escrever este resultado. De qualquer forma, o thread principal deve esperar pelo término da execução de todos os threads filhos criados.

Dicas:

1) Utilize a biblioteca pthread (#include <pthread.h>) e as funções:

- pthread_create(pthread_t *thread, NULL, void* função, void* args);
- pthread_join(pthread_t thread, void *retorno);

5) Implemente um programa em Java que receba via linha de comando um conjunto de números inteiros (*int*), crie um array com estes números, ordene este array usando threads e, ao final do processo de ordenação, escreva o array original e o array ordenado na saída padrão. Crie um thread a partir do algoritmo de ordenação *QuickSort* listado abaixo de forma que as chamadas para realizar a ordenação de um array de inteiros envolva a instanciação de um thread para realizar a tarefa. Por exemplo, em vez de realizar uma recursão simples (linhas [22] e [25]) o algoritmo deve instanciar novos threads responsáveis pela ordenação de parte da lista de inteiros e então aguardar pela conclusão destes threads.

```
[01]void quickSort(int valor[], int esquerda, int direita){
[02]      int i, j, x, y;
```

```
[03]    i = esquerda;
[04]    j = direita;
[05]    x = valor[(esquerda + direita) / 2];
[06]    while(i <= j){
[07]            while(valor[i] < x && i < direita){
[08]                    i++;
[09]            }
[10]            while(valor[j] > x && j > esquerda){
[11]                    j--;
[12]            }
[13]            if(i <= j){
[14]                    y = valor[i];
[15]                    valor[i] = valor[j];
[16]                    valor[j] = y;
[17]                    i++;
[18]                    j--;
[19]            }
[20]    }
[21]    if(j > esquerda){
[22]            quickSort(valor, esquerda, j);
[23]    }
[24]    if(i < direita){
[25]            quickSort(valor, i, direita);
[26]    }
[27]}
```

Dicas:

1) Embora uma dada implementação do Quicksort possa escolher o elemento por meio do qual os demais elementos são ordenados (*pivot*) de diferentes formas, no algoritmo acima é sempre utilizado o elemento central da sublista na atual etapa de recursão (linha [05]). No contexto deste exercício, esta estratégia de escolha do pivot deve ser mantida.

2) O resultado da ordenação pode tanto ser recuperado por meio de um método definido pela classe que implementa o thread de ordenação, quanto uma referência para este objeto pode ser passada por meio do construtor do thread de ordenação.

3) Use a classe ArrayList para trabalhar com uma lista de elementos de tamanho variável.

6) Escreva um programa em Java que receba via linha de comando um conjunto de valores inteiros positivos. O programa deve criar um thread para cada valor recebido para calcular o seu fatorial e escrever o valor calculado usando o seguinte formato:

Thread <índice>: <número>! = <resultado>

Ao final da execução, o thread principal deve aguardar a conclusão de todos os threads antes de encerrar o programa.

Dica:

1) Utilize a classe *BigInteger* para realizar os cálculos de fatorial, garantindo precisão mesmo para números grandes.

Observação:

1) Utilize a classe Matrix.java (em anexo) nos exercícios 2 e 3 para criar e manipular os elementos de uma matriz.