



Universidade Estadual de Campinas  
Faculdade de Tecnologia



# Sistemas Operacionais

## Projeto 1 - Divisão de Matriz

Prof. Dr. André Leon S. Gradvohl

### Grupo “Unicorn Squad”

Ana Luísa Fogarin de S Lima	193948
Isabela Mendes	218123
Larissa Correia da Silva	219765

Limeira, 6 de junho de 2019



## Sumário

<b>Sumário</b>	<b>1</b>
<b>Introdução</b>	<b>2</b>
<b>Objetivo</b>	<b>2</b>
<b>Descrição da solução do problema</b>	<b>2</b>
<b>Instruções para compilar o programa</b>	<b>3</b>
<b>Resultados</b>	<b>3</b>
<b>Conclusão</b>	<b>5</b>
<b>Código fonte</b>	<b>5</b>
<b>Referências</b>	<b>8</b>

## Introdução

Neste trabalho foi desenvolvido o “Projeto 1 - Divide Matriz” da disciplina de Sistemas Operacionais. Além da divisão de uma matriz, utilizou-se o conceito de múltiplas *threads*, registrando os tempos de execução utilizando diferentes quantidades de *threads*.

O programa foi desenvolvido na distro Linux Mint, em linguagem C e utilizando a biblioteca POSIX Threads para implementação das mesmas.

## Objetivo

A partir da utilização de múltiplas *threads*, deve-se dividir uma matriz  $N \times N$  em duas outras duas matrizes: uma possuindo os elementos da diagonal principal e acima da mesma, e outra com os elementos abaixo da diagonal principal.

## Descrição da solução do problema

O usuário fornece a dimensão da matriz, o número de *threads* desejadas e um arquivo “.dat” que terá a matriz que deseja-se dividir. O programa deve dividir essa matriz em outras duas: os valores da diagonal e acima delas ficarão em uma matriz (identificada em vermelho na imagem abaixo), enquanto os valores em posições inferiores à diagonal principal ficarão em outras (identificada em verde na imagem abaixo).

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}$$

Imagem 1 - Matriz

As posições da diagonal principal podem ser identificados por possuírem o número da linha igual ao número da coluna. Para as posições acima da mesma, o número da linha deve ser inferior ao número da coluna. Então, quando o **número da linha for igual ou menor que o**

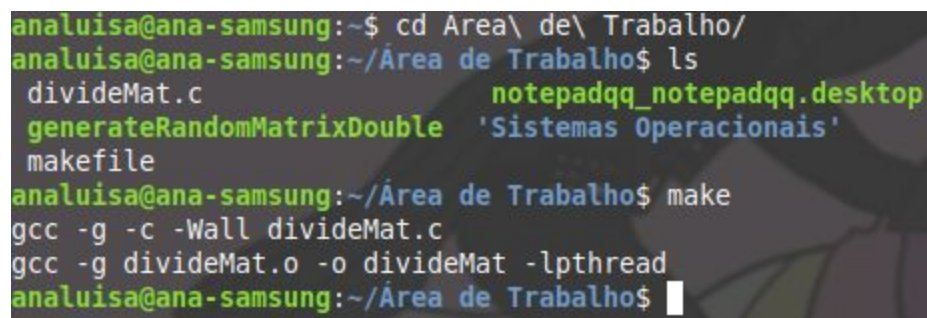
**número da coluna**, ficará na **matriz superior**. Consequentemente, se o **número da linha for maior que o número da coluna**, ficará na **matriz inferior**. Cada matriz será gravada em arquivos com extensão “.diag1” e “.diag2”, respectivamente.

Para esse processo de divisão, o programa utilizará *threads*, que acontecerão paralelamente, cada qual fazendo a análise de algumas posições da matriz. As *threads* devem alternar as posições que estão verificando, para não verificarem mais de uma vez uma mesma posição. Para isso, cada *thread* deve iniciar no elemento situado na posição do mesmo número da *thread* e após isso, a *thread* saltará para o elemento a T posições da que está atualmente.

Link para o GitHub do projeto: <https://github.com/aluisafogarin/ProjetoSO>

## Instruções para compilar o programa

A compilação do programa é feita a partir do arquivo “makefile”. Para isso, os arquivos “divideMat.c” e “makefile” devem estar no mesmo diretório. Através do terminal, acesse o diretório correspondente e execute o comando “make”, para compilar o programa.



```
analuisa@ana-samsung:~$ cd Area\ de\ Trabalho/
analuisa@ana-samsung:~/Área de Trabalho$ ls
divideMat.c          notepadqq_notepadqq.desktop
generateRandomMatrixDouble 'Sistemas Operacionais'
makefile
analuisa@ana-samsung:~/Área de Trabalho$ make
gcc -g -c -Wall divideMat.c
gcc -g divideMat.o -o divideMat -lpthread
analuisa@ana-samsung:~/Área de Trabalho$
```

Imagem 2 - Compilação do Programa

## Resultados

Para os testes, foi utilizado um notebook Samsung X23 (Intel Core i5-7200U, 8GB DDR4). Foi gerada uma matriz 1000x1000 e o programa executado com 2, 4, 8 e 16 *threads*. Os resultados obtidos estão na imagem e no gráfico abaixo.

```
analuisa@ana-samsung:~/Área de Trabalho$ ./generateRandomMatrixDouble 1000 1000
matriz.dat
Gerando matriz 1000 x 1000 para o arquivo matriz.dat
analuisa@ana-samsung:~/Área de Trabalho$ ./divideMat 1000 2 matriz.dat

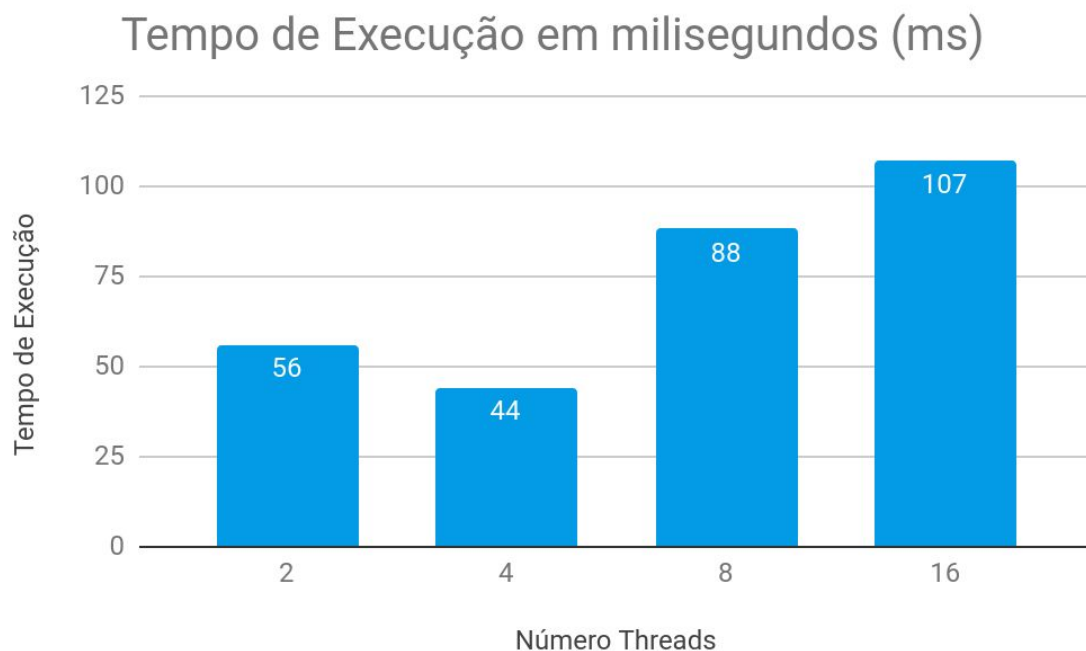
0 programa levou 56.000 ms para separar a matriz.
analuisa@ana-samsung:~/Área de Trabalho$ ./divideMat 1000 4 matriz.dat

0 programa levou 44.000 ms para separar a matriz.
analuisa@ana-samsung:~/Área de Trabalho$ ./divideMat 1000 8 matriz.dat

0 programa levou 88.000 ms para separar a matriz.
analuisa@ana-samsung:~/Área de Trabalho$ ./divideMat 1000 16 matriz.dat

0 programa levou 107.000 ms para separar a matriz.
analuisa@ana-samsung:~/Área de Trabalho$
```

Imagem 3 - Testes com 2, 4, 8 e 16 threads



Link para o vídeo: [https://drive.google.com/file/d/1QVEo\\_MDz1qdFmdzZIWsTdo8-7Ry-oAPJ/view](https://drive.google.com/file/d/1QVEo_MDz1qdFmdzZIWsTdo8-7Ry-oAPJ/view)  
ou [https://drive.google.com/file/d/1Q\\_uQUmqVeFawVOCCTABEuOEiXu5L-13t/view](https://drive.google.com/file/d/1Q_uQUmqVeFawVOCCTABEuOEiXu5L-13t/view)

## Conclusão

O menor tempo obtido foi quando o programa foi executado com 4 *threads*, e o maior tempo foi utilizando 16 *threads*. Assim, observa-se que não é inversamente proporcional a relação número de *threads* e tempo de execução, como seria de se esperar. Além disso, os resultados variam a cada nova execução do programa, pois a forma de execução é diferente em cada uma delas.

## Código fonte

```
1  /*
2  * Grupo Unicorn Squad
3  * Projeto 1 "Divide Matriz" - Sistemas Operacionais
4  *
5  * O programa separa uma matriz NxN em duas matrizes a partir da diagonal principal:
6  * Uma matriz possui os valores da diagonal principal e acima dela;
7  * Uma matriz com os valores abaixo da diagonal principal.
8  *
9  * Autoras:
10 * Ana Luisa Fogarin - 193948
11 * Isabela Mendes - 218123
12 * Larissa Correia - 219765
13 *
14 */
15
16 /* ----- BIBLIOTECAS ----- */
17 #include <stdio.h>
18 #include <stdlib.h>
19 #include <string.h>
20 #include <pthread.h>
21 #include <time.h>
22
23 /* ----- ESTRUTURA ----- */
24 /* Estrutura para passagem dos dados necessarios para as funcoes */
25 typedef struct Argumentos {
26     int N;
27     int T;
28     int posicao;
29     double **matrizGerada;
30     double **matrizCima;
31     double **matrizBaixo;
32 } argumentos;
33
34 /* ----- FUNÇÕES ----- */
35 /* Função que identifica linha e coluna da matriz */
36 int linha(int ordem, int local){
37     return local/ordem;
38 }
39
```

```

40 ▽ int coluna(int ordem, int local){
41     return local%ordem;
42 }
43
44 ▽ /* Funcao que realiza a divisao da matriz */
45 ▽ void *separaMatriz(void *arg){
46     argumentos *x = arg;           //cria um novo objeto da struct para receber os argumentos passados
47     int i = linha(x->N, x->posicao); //descobre em qual linha o elemento que a thread ira analisar esta
48     int j = coluna(x->N, x->posicao); //descobre em qual coluna o elemento que a thread ira analisar esta
49
50 ▽ /* Laço while permite proporciona a repeticao ate que atinja o limite de elementos da matriz*/
51 ▽ while(x->posicao <= (x-> N*x->N)-1) {
52     if (i <= j) //Define matriz superior
53     {
54         x->matrizCima[i][j] = x->matrizGerada[i][j];
55         x->matrizBaixo[i][j] = 0; //Se nao eh o elemento desejado, iguala a zero
56     }
57     else { //Define matriz inferior
58         x->matrizBaixo[i][j] = x->matrizGerada[i][j];
59         x->matrizCima[i][j] = 0; //Se nao eh o elemento desejado, iguala a zero
60     }
61
62     /* Muda a posicao para realizar a divisao no proximo elemento desejado */
63     x->posicao += x->T;
64     i = linha(x->N, x->posicao);
65     j = coluna(x->N, x->posicao);
66 }
67     return(NULL);
68 }
69 ▽ /* ----- *
70 *
71 *
72 * ----- PROGRAMA PRINCIPAL ----- */
73 ▽ int main (int argc, char *argv[]) {
74     int N; //N = dimensao;
75     int T; //T = numero de threads
76     int i, j;
77     char nomeMatriz[30];
78

```

```

79 ▽ /* Atribui as variaveis os valores passados juntos a inicialização do programa */
80 ▽ /* Afunção "atoi" faz conversao de tipo char para int */
81 ▽ N = atoi(argv[1]);
82 ▽ T = atoi(argv[2]);
83 ▽ strcpy(nomeMatriz, argv[3]);
84
85 ▽ /* Cria as threads*/
86 ▽ pthread_t threads[T];
87 ▽ /* Cria um objeto da struct */
88 ▽ argumentos dados[T];
89
90 ▽ /* Cria e aloca dinamicamente as matrizes matrizes
91 *
92 * Cria um vetor em que cada posição tem um double* (linhas da matriz)
93 * Para cada posição double* do vetor, cria um novo vetor double (colunas da matriz)
94 */
95
96 double **matrizGerada = (double **) malloc(sizeof(double *) * N);
97 for (i=0; i < N; i++) {
98     matrizGerada[i] = (double *) malloc(sizeof(double) * N);
99 }
100
101 double **matrizCima = (double **) malloc(sizeof(double) * N);
102 for (i=0; i < N; i++) {
103     matrizCima[i] = (double *) malloc(sizeof(double) * N);
104 }
105
106 double **matrizBaixo = (double **) malloc(sizeof(double) * N);
107 for (i=0; i < N; i++) {
108     matrizBaixo[i] = (double *) malloc(sizeof(double) * N);
109 }
110
111 ▽ /* Cria e abre arquivo para ler a matriz fornecida*/
112 FILE *arq;
113 arq = fopen(nomeMatriz, "r");
114 if (arq == NULL) {
115     puts("Ocorreu um erro ao abrir o arquivo da matriz!\n");
116     return 1;
117 }

```



```

119 ▽ /* Le a matriz do arquivo */
120 ▽ for (i=0; i < N; i++) {
121 ▽     for (j=0; j < N; j++) {
122 ▽         fscanf(arq,"%lf", &matrizGerada[i][j]);
123 ▽     }
124 ▽ }
125
126 ▽ /* Fecha arquivo da matriz */
127 ▽ fclose(arq);
128
129 ▽ /* Remove a extensao do nome da matrizGerada para poder criar arquivos das matrizes desejadas da forma desejada */
130 ▽ const char c[2] = ".";
131 ▽ char *nomeSemExt;
132 ▽ nomeSemExt = strtok(nomeMatriz, c);
133 ▽ /* A função "strtok(string1, string2) aponta o ponteiro atribuido a mesma para a string1 delimitada pela string2,
134 ▽ * nesse caso, passa para nomeSemExt a string nomeMatriz ate encontrar o caractere "." (atribuido na variavel c),
135 ▽ * permitindo assim, criar os nomes dos arquivos para gravas as matrizes resultantes
136 ▽ */
137
138 ▽ /* Define variaveis para contar o tempo */
139 ▽ clock_t tempIni;
140 ▽ clock_t tempFim;
141
142 ▽ /* ----- Implementa thread ----- */
143 ▽ tempIni = clock();
144 ▽ for(i=0; i < T; i++) {
145 ▽     /* Passa os valores necessarios para a struct*/
146 ▽     dados[i].N = N;
147 ▽     dados[i].T = T;
148 ▽     dados[i].matrizGerada = matrizGerada;
149 ▽     dados[i].matrizCima = matrizCima;
150 ▽     dados[i].matrizBaixo = matrizBaixo;
151 ▽     dados[i].posicao = i;
152
153 ▽     /* Cria as threads */
154 ▽     pthread_create(&threads[i], NULL, separaMatriz, (void *)&dados[i]);
155
156 ▽ }
157
158 ▽
159 ▽ for (i=0; i < T; i++) {
160 ▽     pthread_join(threads[i], NULL);
161 ▽ }
162 ▽ /* Termina a contagem de tempo e atribui a tempoTotal o tempo decorrido */
163 ▽ tempFim = clock();
164 ▽ double tempoTotal = (tempFim - tempIni) * 1000 / CLOCKS_PER_SEC;
165 ▽
166 ▽ /* ----- */
167 ▽
168 ▽ /* Cria a string com o nome desejado para gravação da matriz superior*/
169 ▽ char primeiraMatriz[30];
170 ▽ strcpy(primeiraMatriz, nomeSemExt);
171 ▽ strcat(primeiraMatriz, ".diag1");
172 ▽
173 ▽ /* Cria e abre arquivo para a matriz superior */
174 ▽ arq = fopen(primeiraMatriz, "w+");
175 ▽ if (arq == NULL) {
176 ▽     puts("Ocorreu um erro ao abrir o arquivo da matriz diag1!\n");
177 ▽     return 1;
178 ▽ }
179 ▽
180 ▽ /* Grava matrizCima no arquivo */
181 ▽ for (i=0; i < N; i++) {
182 ▽     for (j=0; j < N; j++) {
183 ▽         fprintf(arq," %lf ", matrizCima[i][j]);
184 ▽     }
185 ▽     fprintf(arq,"\n");
186 ▽ }
187 ▽
188 ▽ /* Fecha o arquivo */
189 ▽ fclose(arq);
190 ▽
191 ▽ /* Cria a string com o nome desejado para gravação da matriz inferior */
192 ▽ char segundaMatriz[30];
193 ▽ strcpy(segundaMatriz, nomeSemExt);
194 ▽ strcat(segundaMatriz, ".diag2");
195 ▽
196 ▽ /* Cria e abre arquivo para a matriz inferior */
197 ▽ arq = fopen(segundaMatriz, "w+");

```



```

195 ▾ /* Cria e abre arquivo para a matriz inferior */
196   arq = fopen(segundaMatriz, "w+");
197 ▾   if (arq == NULL) {
198       puts("Ocorreu um erro ao abrir o arquivo da matriz diag2!\n");
199       return 1;
200   }
201
202 ▾   /* Grava matrizBaixo no arquivo */
203   for (i=0; i < N; i++) {
204 ▾       for (j=0; j < N; j++) {
205 ▾           fprintf(arq, " %lf ", matrizBaixo[i][j]);
206       }
207       fprintf(arq, "\n");
208   }
209
210 ▾   /* Fecha o arquivo */
211   fclose(arq);
212
213 ▾   /* Libera a memoria alocada pelas matrizes */
214   free(matrizGerada);
215   free(matrizCima);
216   free(matrizBaixo);
217
218 ▾   /* Exibe na tela o tempo que o programa levou */
219   printf("\n0 programa levou %.3lf ms para separar a matriz.\n", tempoTotal);
220
221   return 0;
222 }

```

## Referências

Alocação de Matrizes: <https://www.pucsp.br/~so-comp/cursoc/aulas/ca70.html>

Threads: <https://computing.llnl.gov/tutorials/pthreads>

Makefile: <http://solver.assistedcoding.eu/makefilegen>

Biblioteca Time.h: [https://www.tutorialspoint.com/c\\_standard\\_library/time\\_h.htm](https://www.tutorialspoint.com/c_standard_library/time_h.htm)