UNIVERSIDADE ESTADUAL DE CAMPINAS - FACULDADE DE TECNOLOGIA
BACHARELADO EM SISTEMAS DE INFORMAÇÃO
SISTEMAS OPERACIONAIS

BRUNO DO VALLE SAES ADEGAS 195045
JULIANA PINHO MARCHI 177291
LETÍCIA SOUSA DE OLIVEIRA 201506

Relatório Sistemas Operacionais

Projeto 1: “ Localiza na matriz”

LIMEIRA - SP
2018

1. Introdução

O projeto proposto pela disciplina de Sistemas Operacionais tinha como objetivo o desenvolvimento de um programa na linguagem de programação “C”, o qual utilizasse múltiplas Threads para a busca de um determinado valor em uma matriz contida em um arquivo.

2. Algoritmo em alto nível

O programa desenvolvido tem cinco diferentes entradas, sendo elas o número de linhas, número de colunas, nome do arquivo, número de threads que o programa utilizará e o valor a ser procurado na matriz.

```
#include <stdio.h>  // scanf-printf
#include <pthread.h>   // Threads
#include <stdlib.h>   // Arquivo


// ---------- Declaracao de variaveis necessarias ----------

//Struct que armezanara as posicoes que o numero foi procurado
struct posicao
{
    int x;
    int y;
    struct posicao *prox;
};

typedef struct posicao Posicao;

//Dados sobre a lista/struct
int qtdno=0;
Posicao *ini=NULL;

//Arquivo
FILE *arq;
char FileName[25];

//Dados inseridos pelo usuario
//Esses dados sao globais pqe as threads vao usa-los
int linhas=0, colunas=0, nthread=0, base=0;
double nprocurado=0;

//Semaforos para a organizacao de acesso ao arquivo
pthread_mutex_t lock;
pthread_mutex_t nos;

// ----------------- Prototipos de funcoes -----------------
void OrganizaLista();
void InsereNo(int x, int y);
void *Busca(void *vargp);
```

```
// ----------------------- Codigos -------------------------
int main()
{
    //Variaveis locais
    int i, j, erro;
    double num;
    char ch;

    //Inicializacao dos semaforos
    if (pthread_mutex_init(&lock, NULL) != 0)
    {
             printf("\n mutex init failed\n");
             exit(0);
    }
    if (pthread_mutex_init(&nos, NULL) != 0)
    {
            printf("\n mutex init failed\n");
             exit(0);
    }

    //Insercao de dados
    printf("Numero de linhas: ");
    scanf("%d", &linhas);
    printf("Numero de colunas: ");
    scanf("%d", &colunas);
    printf("Nome do arquivo: ");
    scanf("%s", FileName);
    printf("Numero de threads: ");
    scanf("%d", &nthread);
    printf("Digite o numero a ser procurado: " );
    scanf("%lf", &nprocurado);

    //Abre o arquivo para somente leitura
    arq = fopen(FileName, "r");

    //Teste do arquivo
    if (arq == NULL)
    {
         perror("ERRO; Arquivo não encontrado: ");
         exit(EXIT_FAILURE);
    }


    //Copiando a matriz do arquivo para a memoria
    double matriz[linhas][colunas];
    //Matriz de tamanho de acordo com os valores inseridos pelo usuario

    for(i=0; i<linhas; i++)
    {
        for(j=0; j<colunas; j++)
        {
               fscanf(arq, "%lf", &num);
               matriz[i][j] = num;
        }
    }
```

```
    //Criando vetores para as threads
    pthread_t thread_id[nthread];

    //Criando as threads
    for(i=0; i<nthread; i++)
    {
        erro = pthread_create(&thread_id[i], NULL, &Busca, matriz);
        if(erro) //Teste para ver se houve erro na criacao da thread
        {
               printf("ERRO; A thread não pôde ser criada: return code from
pthread_create() is %d\n", erro);
                    exit(-1);
        }
    }


    for(i=0; i<nthread; i++) //Juncao das threads para prosseguir com a execucao do
programa
    {
        pthread_join(thread_id[i], NULL);
    }


    //Teste para ver se o numero foi encontrado na matriz
    if(qtdno == 0)
    {
        printf("\n\nElemento não encontrado\n\n");
    }
    else
    {
        OrganizaLista(); // Chamada de funcao que organiza os nos da lista(posicoes
[x,y] do numero procurado na matriz)

        printf("Posições que contém o valor %lf:", nprocurado);
        Posicao *percorre = ini;
        while(percorre != NULL)
        {
               //Exibicao das posicoes do numero encontrado (+1 somado para
facilitar a visualizacao na matriz)
               printf(" [%d,%d]",(percorre->x)+1, (percorre->y)+1);
               percorre = percorre->prox;

        }
               printf("\n\n");
    }

    fclose(arq); //Fecha o arquivo
    pthread_exit(NULL); //Encerra a thread principal
    exit(0); //Finaliza execucao

}
```

```
void *Busca(void *matriz) //Funcao que sera chamada pelas threads e que busca o
numero pela matriz
{
    //Operacao down no semaforo, para que nao sejam lidos/salvos dois valores
iguais
    pthread_mutex_lock(&nos);
    int i = base;
    base++;

    pthread_mutex_unlock(&nos); //Operacao up no semaforo

    //Declaracao das variaveis locais
    int nexec=0,j=0, k=0;
    double *m = (double *)matriz; //Ponteiro para matriz


    //Calculos para linhas e posicionamento do ponteiro

    nexec=linhas/nthread; //nexec - Numero de linhas que a thread vai procurar

    if(linhas%nthread != 0 && ((linhas%nthread) > i)) //Caso o numero de linhas nao
seja divisivel pelo numero de threads
    {
        nexec++; //Atribui uma linha "sobrante" para a thread em questao
    }

    m = m + (colunas*i);

    //Busca pela matriz
    for(j = 0; j<nexec; j++) //o j eh as "linhas" que serao procuradas (quantidade)
    {
        for(k=0; k<colunas; k++) //Percorre a linha
        {
              if((*m) == nprocurado) //Caso o valor apontado por 'm' seja igual ao
numero inserido pelo usuario
              {
                    //Operacao down no semaforo, para que nao haja confusao na
insercao de nos
                    pthread_mutex_lock(&lock);

                    InsereNo(i+(nthread*j), k); //Eh salva a posicao desse numero
na matriz

                    //Operacao up no semaforo, para outra thread possa realizar a
mesma operacao se necessario
                    pthread_mutex_unlock(&lock);
              }
              m++; //Vai para a proxima coluna da matriz
        }

        m = m + (colunas*(nthread-1)); //Apos uma linha ser lida, o ponteiro e
posicionado para a proxima linha a ser lida pela thread
    }
    pthread_exit(NULL); //Encerra a thread
    return NULL;
}
```

```
//Insercao de no(estruturas) que contem o [x,y] do elemento encontrado na matriz
void InsereNo(int x, int y)
{
    Posicao *novo = (Posicao *) malloc(sizeof(Posicao));

    novo->x = x;
    novo->y = y;
    novo->prox = NULL;
    if(ini == NULL)
        ini = novo;
    else
    {
        Posicao *percorre;
        percorre = ini;
        while(percorre->prox != NULL)
        {
               percorre = percorre->prox;
        }
        percorre->prox = novo;
    }
    qtdno++;
}

void OrganizaLista() //BubbleSort utilizado
{
    int i, aux,j;
    Posicao *percorre1;
    Posicao *percorre2;

    for(i=0; i<qtdno; i++)
    {
        percorre1 = ini;
        percorre2 = ini->prox;

        while(percorre2 != NULL)
        {
               if(percorre1->x > percorre2->x)
               {
                      aux = percorre1->x;
                      percorre1->x = percorre2->x;
                      percorre2->x = aux;
                      aux = percorre1->y;
                      percorre1->y = percorre2->y;
                      percorre2->y = aux;
               }
               else if(percorre1->x > percorre2->x && percorre1->y > percorre2->y)
               {
                      aux = percorre1->y;
                      percorre1->y = percorre2->y;
                      percorre2->y = aux;
               }
               percorre1 = percorre2;
               percorre2 = percorre2->prox;
        }
    }
```

}

3. Instruções para a compilação

1º) Entrar no repositório da equipe no GitHub pelo link:
https://github.com/xLeticiaOliveira/SOProjeto.git
2º) Baixar a pasta compactada e descompactar;
3º) Verificar se o arquivo de entrada* está no mesmo diretório que o código fonte;
4º) Abrir o terminal e entrar no diretório correspondente;
5º) Compilar o código com o seguinte comando :
"gcc Projeto1.c -o Projeto1 -lpthread";
6º) Executar o programa (Imagem 1).

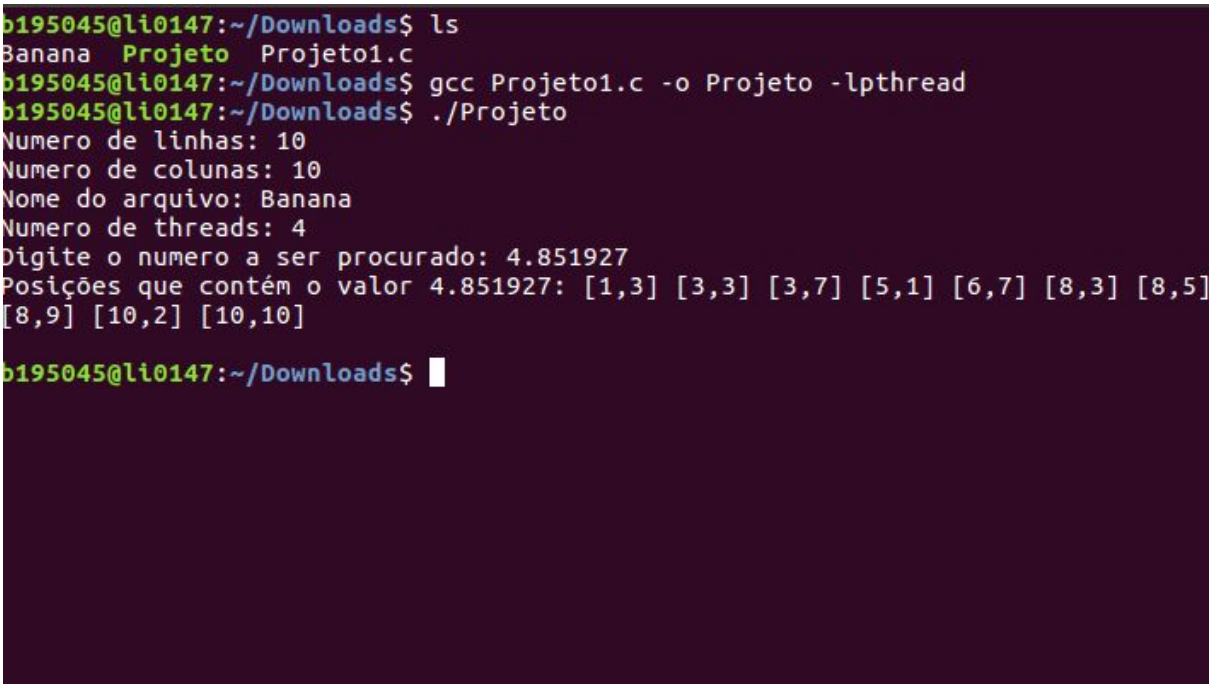


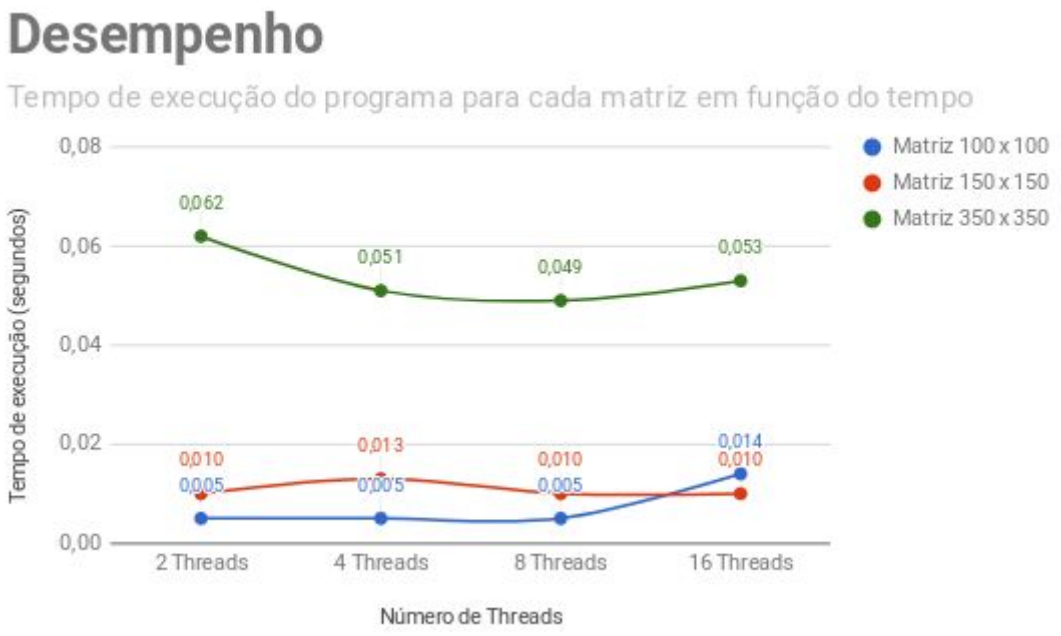
Imagem 1 : Execução do Programa "Projeto 1".
Fonte: Grupo.

Obs: Caso o sistema operacional Linux não tenha a biblioteca POSIX threads instalada, execute o seguinte comando no terminal : **"urpmi lpthread".**

* Para a criação do arquivo de entrada, foi utilizado o código disponibilizado pelo professor André Leon: "generateRandomMatrixDouble.c". O código em questão está na pasta "Gera Matriz Aleatória", contida no repositório do grupo Kokonut. Para instruções para sua execução, leia o README contido na pasta.

4. Conclusão

O gráfico abaixo representa os tempos de execução do programa desenvolvido para a busca de determinado valor em três diferentes matrizes quadradas.



Para a matriz de 100 x 100, é possível observar que o desempenho do programa é estável, ou seja, não há ganhos nem perdas quando utiliza-se 2, 4 ou 8 threads, porém, ao utilizar 16 threads, seu desempenho decaiu, demorando cerca de 150% a mais do que as execuções anteriores. Isto decorre do fato que a matriz é pequena, fazendo com que menos threads sejam mais eficientes do que uma grande quantidade, pois o tempo necessário para criação e organização das mesmas supera o tempo de busca do valor utilizando menos threads.

Na execução para a matriz 150 x 150, pode-se observar que o desempenho foi estável, tendo somente uma pequena flutuação quando executado com 4 threads, fato provavelmente ocasionado pelo tempo de execução da CPU em si, não representando uma variação efetiva no desempenho do programa.

No teste realizado com uma matriz 350 x 350, pode-se observar que, na execução com 2 threads, o tempo foi maior do que quando executado com uma maior quantidade. Isto porque, com mais threads, há uma maior distribuição de linhas, de modo que o tempo de “organização” das threads é compensado nas buscas simultâneas, fazendo-a ser mais eficiente do que a busca feita por menos threads. Entretanto, é observável que, a partir de 8 threads, o desempenho a piora, aumentado o tempo de execução.

Para tanto, pode-se concluir que para a utilização de múltiplas threads é necessário que se estude o cenário onde elas serão aplicadas, pois, se utilizadas incorretamente, podem piorar o desempenho do programa.

Planilha com os dados obtidos nas execuções disponível em:
https://docs.google.com/spreadsheets/d/1OXr10csALoyktbXrQjIDXsJQLzjrN7o5m-mz0i9F5SY/edit?usp=sharing