Projeto de Sistemas Operacionais

Projeto 1

Link do repositório GIT:https://github.com/liraRaphael/SO-projeto

Link do video do programa em funcionamento: https://youtu.be/dKFQWkIHiLE

Giovanni Bassetto - 216968

Jonathan Bozza Gonçalves - 176497

Raphael Lira dos Santos - 223865

1)Solução do Problema

A maneira que utilizamos para resolver o problema foi primeiro criando as duas matrizes que serão alteradas e lendo a matriz principal (Também lê-se o tamanho da matriz):

```
void
iniciaMatrizes(double **
aux){
                                    int
                                          i;
                                    for(i=0;i<tamanhoMatriz;i++){
                                          aux[i] = (double *)
                               malloc(tamanhoMatriz * sizeof(double));

                                    }


                               }


int ler (char *
nome){
                    FILE
                          //crio "arq" - leitura
                          * arq;

                    int
                             i,j;



                    //abre arquivo
                    arq = fopen(nome , "r");

                    //caso o arquivo não for aberto, dê erro
                    if(arq == NULL)
                          return 1;
```

```
i = 0;
//começa a sequencia de gravação da matriz
while(!feof(arq) && i < tamanhoMatriz){

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

        i++;
    }
```

Depois de ler a matriz inicial, o programa utiliza dois “For”s para alterar as matrizes e deixá-las da maneira correta, as threads são usadas aqui para facilitar essa operação:

```
void * operaMatrizes(void *
args){

                        dadosMatriz * dados = args;


                        int
                                //variaveis para o loop
                                i,j;

                                //transforme a matriz diag1

                    for(i=dados->id;i<tamanhoMatriz;i+=nThread){
                                for(j=i;j<tamanhoMatriz;j++){
                                        diag1[i][j] =
matriz[i][j];
                                        diag2[i][j] = 0;
                                }
                        }
```

```
        //transforme a matriz diag2

        for(i=dados->id;i<tamanhoMatriz;i+=nThread){
                for(j=0;j<i;j++){
                        diag2[i][j] =
matriz[i][j];
                        diag1[i][j] = 0;
                }
        }


}
```

Após serem alteradas as matrizes são então gravadas no arquivo indicado:

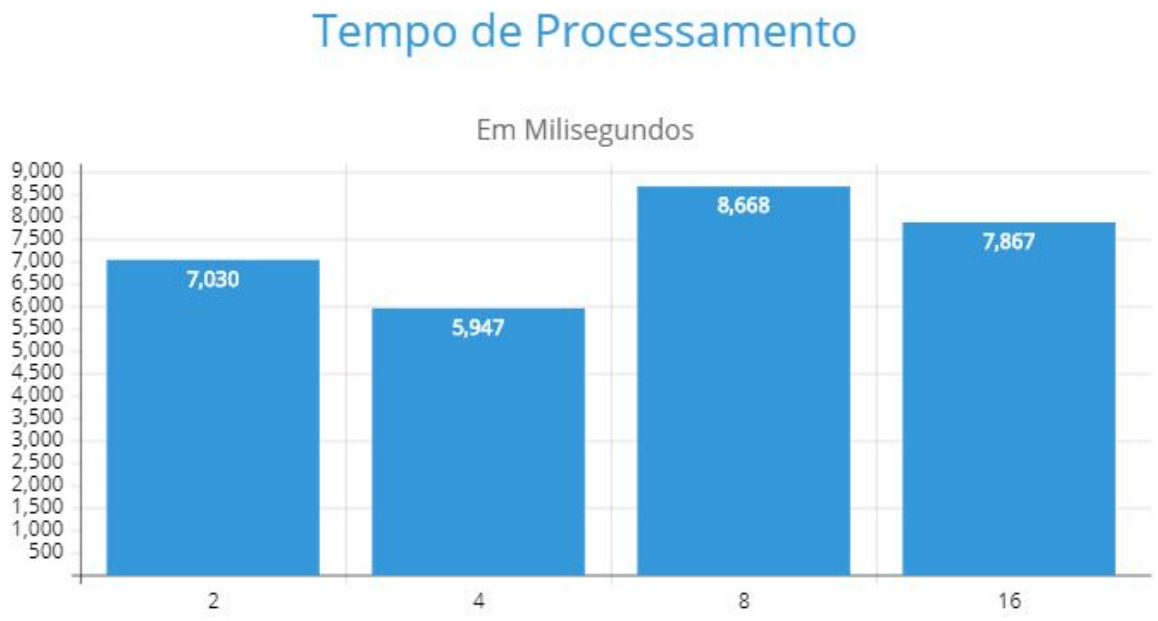
```
int gravar(double ** aux, char
* nome){

        FILE
                //crio "arq" - gravação
                * arq;

        int
                //responsável por fazer o loop de
gravação
                i,j;

        //abre arquivo
        arq = fopen(nome , "w+");

        //caso o arquivo não for aberto, dê
erro
        if(arq == NULL)
                return 1;

        //começa a sequencia de gravação da
matriz
```

```
        for(i=0;i<tamanhoMatriz;i++){
                for(j=0;j<tamanhoMatriz;j++){

fprintf(arq,"\t%lf",aux[i][j]);
                }

                //quebra a linha
                fprintf(arq,"\n");
        }


        //fecha o arquivo
        fclose(arq);

  return 0;

}
```

2)Instruções de Compilação

Para compilar o programa, basta utilizar um sistema Linux, ir para a pasta onde o programa se encontra e digitar “Gcc main.c -o Prog -lpthead” no prompt de comando. Após usar o comando acima, utiliza-se o comando “./divideMat M T D” onde M = Tamanho da Matriz(Exemplo, trocar M por 1000 resulta em uma matriz 1000x1000), T = Número de Threads (2 ,4 ,8 ou 16) e D = Arquivo que contém os dados da matriz (No exemplo do vídeo utilizamos o 1000.dat).

3)Gráfico de Tempo



4)Conclusão

A conclusão que tiramos desse trabalho em relação as threads foi: O número de Threads que teve melhores resultados foi 4 (5,947 Milissegundos), enquanto o que teve pior resultado foi 8 (8,668 Milissegundos). Isso mostra que apenas aumentar o número de threads não resulta em um aumento de velocidade. Isso ocorre porque com um número desnecessário de threads muitas delas acabam ficando em stand-by por muito tempo e a troca de threads por si só gera uma demora maior do que se fosse utilizado um número menor de threads.