UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE TECNOLOGIA

Geovanne Silva Pereira - 250328

Heloisa Feitosa Rocha - 260803

PROJETO DE SISTEMAS OPERACIONAIS

GRUPO: THREAD MASTERS

Limeira

2024

1. Introdução

Ao longo deste arquivo você conferirá o desenvolvimento de um projeto puramente realizado em linguagem C para compilação em linux com o objetivo de compilação de threads. Além de, ter a aplicação desenvolvida possui análises em cima dos resultados obtidos em testes.

2. Instruções de compilação

Para que a compilação do arquivo do seja viável é recomendável que:

I. Configurando o ambiente de trabalho

- Utilize um sistema operacional Linux;
- Tenha um compilador de linguagem C chamado GCC configurado, no caso para verificar se ele está instalado basta abrir o terminal e digitar `gcc --version`, caso ele esteja instalado retornará uma versão, caso contrário execute:

```
sudo apt update
sudo apt install gcc
```

- Crie uma pasta para o projeto com o comando:

```
mkdir TT304-ProgramacaoComMultiplosThreads
```

- Mova os arquivos para a pasta do projeto com o comando:

```
mv/caminho/para/mergesort.c ~/Desktop/TT304-Thread-Masters/
```

- Entre na pasta do projeto com o comando:

```
cd ~/Desktop/TT304-Thread-Masters
```

II. Crie códigos Makefile

O código que será compilado é mergesort.c e para que possamos ter uma compilação facilitada temos um arquivo makefile, que permite a utilização de comandos automáticos ao invés de digitados um a um. Pensando nisso:

- Abra o editor de texto *nano* no terminal e crie um arquivo chamado Makefile e digite: `nano Makefile`;
- Dentro do arquivo, cole o seguinte conteúdo:

```
C/C++

# Nome do compilador que vamos usar
CC = gcc

CFLAGS = -Wall -pthread

# Nomeação do executável final
TARGET = mergesort

# Arquivo de código fonte
SOURCES = mergesort.c
```

```
# Criação do programa
all: $(TARGET)

$(TARGET): $(SOURCES)
	$(CC) $(CFLAGS) -o $(TARGET) $(SOURCES)

clean:
	rm -f $(TARGET) *.o
```

- Por fim, salve e feche o arquivo com Ctrl + X e depois confirme com Y.

III. Realize a compilação

- Com o makefile pronto, você pode compilar o programa utilizando um único comando no terminal, que é o `make`;

IV. Faça a execução do programa

Para a realização da execução do programa basta aplicar:
./mergesort T arq1.dat arq2.dat arq3.dat arq4.dat arq5.dat -o saida.dat
Em que:

- T é o número de threads - deve ser selecionado por você entre os valores 1, 2, 4 ou 8;
- os .dat são os arquivos de entrada com os números a serem organizados;
- saida.dat é o nome do arquivo onde o programa vai salvar os números organizados;

V. Limpeza

- Depois da compilação você pode limpar os arquivos temporários para manter o diretório organizado, e para isso basta rodar o `make clean`, para remover o programa compilado e outros arquivos temporários que possam ter sido criados no processo de compilação.
  Obs.: Lembrando que este passo é opcional.

3. Desenvolvimento do projeto

3.1. Implementação do código

3.1.1. Bibliotecas

```C/C++
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h> //permite a manipulação de threads
#include <time.h>
#include <string.h>
```

3.1.2. Funções

C/C++

```
void merge(int *data, int esquerda, int meio, int direita) {
    int n1 = meio - esquerda + 1;
    int n2 = direita - meio;
    int *E = malloc(sizeof(int) * n1);
    int *D = malloc(sizeof(int) * n2);
    if (E == NULL || D == NULL) {
        fprintf(stderr, "Erro ao alocar memória para mesclagem.\n");
        exit(EXIT_FAILURE);
    }
    for (int i = 0; i < n1; i++)
        E[i] = data[esquerda + i];
    for (int j = 0; j < n2; j++)
        D[j] = data[meio + 1 + j];

    int i = 0, j = 0, k = esquerda;
    while (i < n1 && j < n2) {
        if (E[i] <= D[j]) {
            data[k++] = E[i++];
        } else {
            data[k++] = D[j++];
        }
    }

    while (i < n1) {
        data[k++] = E[i++];
    }

    while (j < n2) {
        data[k++] = D[j++];
    }

    free(E);
    free(D);
}
void merge_sort(int *data, int esquerda, int direita) {
    if (esquerda < direita) {
        int meio = esquerda + (direita - esquerda) / 2;
        merge_sort(data, esquerda, meio);
        merge_sort(data, meio + 1, direita);
        merge(data, esquerda, meio, direita);
    }
}
void *thread_sort(void *arg) {
    struct ThreadData *t_data = (struct ThreadData *)arg;
    struct timespec inicio, fim;
```

```
    clock_gettime(CLOCK_MONOTONIC, &inicio);

    merge_sort(t_data->data, t_data->esquerda, t_data->direita);

    clock_gettime(CLOCK_MONOTONIC, &fim);
    t_data->tempo_execucao = (fim.tv_sec - inicio.tv_sec);
    t_data->tempo_execucao += (fim.tv_nsec - inicio.tv_nsec) / 1e9;

    pthread_mutex_lock(&mutex_thread);
    thread_tempo_execucao[t_data->thread_id] = t_data->tempo_execucao;
    pthread_mutex_unlock(&mutex_thread);

    pthread_exit(NULL);
}
void ler_arquivos(int **data, int *tamanho_total, int argc, char *argv[],
int index_inicial) {
    int *all_data = NULL;
    int total = 0;

    for (int i = index_inicial; i < argc - 2; i++) {
        FILE *fp = fopen(argv[i], "r");
        if (fp == NULL) {
            fprintf(stderr, "Erro ao abrir o arquivo %s\n", argv[i]);
            exit(EXIT_FAILURE);
        }
        int num;
        int tamanho = 0;
        int capacidade = 1000;
        int *dados_do_arquivo = malloc(sizeof(int) * capacidade);
        if (dados_do_arquivo == NULL) {
                  fprintf(stderr, "Erro ao alocar memória para leitura do
arquivo.\n");
            exit(EXIT_FAILURE);
        }

        while (fscanf(fp, "%d", &num) != EOF) {
            if (tamanho >= capacidade) {
                capacidade *= 2;
                  dados_do_arquivo = realloc(dados_do_arquivo, sizeof(int) *
capacidade);
                if (dados_do_arquivo == NULL) {
                      fprintf(stderr, "Erro ao realocar memória para leitura
do arquivo.\n");
                   exit(EXIT_FAILURE);
                }
            }
            dados_do_arquivo[tamanho++] = num;
        }
```

```
        fclose(fp);

        all_data = realloc(all_data, sizeof(int) * (total + tamanho));
        if (all_data == NULL) {
                  fprintf(stderr, "Erro ao realocar memória para todos os
dados.\n");
            exit(EXIT_FAILURE);
        }
        memcpy(all_data + total, dados_do_arquivo, sizeof(int) * tamanho);
        total += tamanho;
        free(dados_do_arquivo);
    }
    *data = all_data;
    *tamanho_total = total;
}
```

3.1.3. Código completo

C/C++

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
#include <string.h>

#define MAX_INT 2147483647

struct ThreadData {
    int *data;
    int esquerda;
    int direita;
    int thread_id;
    double tempo_execucao;
};

int num_threads;
pthread_mutex_t mutex_thread = PTHREAD_MUTEX_INITIALIZER;
double *thread_tempo_execucao;
void merge(int *data, int esquerda, int meio, int direita) {
    int n1 = meio - esquerda + 1;
    int n2 = direita - meio;
    int *E = malloc(sizeof(int) * n1);
    int *D = malloc(sizeof(int) * n2);
```

```c
    if (E == NULL || D == NULL) {
        fprintf(stderr, "Erro ao alocar memória para mesclagem.\n");
        exit(EXIT_FAILURE);
    }

    for (int i = 0; i < n1; i++)
        E[i] = data[esquerda + i];
    for (int j = 0; j < n2; j++)
        D[j] = data[meio + 1 + j];

    int i = 0, j = 0, k = esquerda;
    while (i < n1 && j < n2) {
        if (E[i] <= D[j]) {
            data[k++] = E[i++];
        } else {
            data[k++] = D[j++];
        }
    }

    while (i < n1) {
        data[k++] = E[i++];
    }

    while (j < n2) {
        data[k++] = D[j++];
    }

    free(E);
    free(D);
}

void merge_sort(int *data, int esquerda, int direita) {
    if (esquerda < direita) {
        int meio = esquerda + (direita - esquerda) / 2;
        merge_sort(data, esquerda, meio);
        merge_sort(data, meio + 1, direita);
        merge(data, esquerda, meio, direita);
    }
}

void *thread_sort(void *arg) {
    struct ThreadData *t_data = (struct ThreadData *)arg;
    struct timespec inicio, fim;
    clock_gettime(CLOCK_MONOTONIC, &inicio);

    merge_sort(t_data->data, t_data->esquerda, t_data->direita);

    clock_gettime(CLOCK_MONOTONIC, &fim);
```

```
    t_data->tempo_execucao = (fim.tv_sec - inicio.tv_sec);
    t_data->tempo_execucao += (fim.tv_nsec - inicio.tv_nsec) / 1e9;

    pthread_mutex_lock(&mutex_thread);
    thread_tempo_execucao[t_data->thread_id] = t_data->tempo_execucao;
    pthread_mutex_unlock(&mutex_thread);

    pthread_exit(NULL);
}

void ler_arquivos(int **data, int *tamanho_total, int argc, char *argv[],
int index_inicial) {
    int *all_data = NULL;
    int total = 0;

    for (int i = index_inicial; i < argc - 2; i++) {
        FILE *fp = fopen(argv[i], "r");
        if (fp == NULL) {
            fprintf(stderr, "Erro ao abrir o arquivo %s\n", argv[i]);
            exit(EXIT_FAILURE);
        }

        int num;
        int tamanho = 0;
        int capacidade = 1000;
        int *dados_do_arquivo = malloc(sizeof(int) * capacidade);
        if (dados_do_arquivo == NULL) {
                  fprintf(stderr, "Erro ao alocar memória para leitura do
arquivo.\n");
            exit(EXIT_FAILURE);
        }

        while (fscanf(fp, "%d", &num) != EOF) {
            if (tamanho >= capacidade) {
                capacidade *= 2;
                  dados_do_arquivo = realloc(dados_do_arquivo, sizeof(int) *
capacidade);
                if (dados_do_arquivo == NULL) {
                      fprintf(stderr, "Erro ao realocar memória para leitura
do arquivo.\n");
                   exit(EXIT_FAILURE);
                }
            }
            dados_do_arquivo[tamanho++] = num;
        }
        fclose(fp);

        all_data = realloc(all_data, sizeof(int) * (total + tamanho));
```

```
        if (all_data == NULL) {
                    fprintf(stderr, "Erro ao realocar memória para todos os
dados.\n");
            exit(EXIT_FAILURE);
        }
        memcpy(all_data + total, dados_do_arquivo, sizeof(int) * tamanho);
        total += tamanho;
        free(dados_do_arquivo);
    }

    *data = all_data;
    *tamanho_total = total;
}

int main(int argc, char *argv[]) {
    if (argc < 5) {
        fprintf(stderr, "Uso incorreto. Exemplo:\n");
             fprintf(stderr, "./mergesort 4 arq1.dat arq2.dat arq3.dat -o
saida.dat\n");
        exit(EXIT_FAILURE);
    }

    num_threads = atoi(argv[1]);
       if (num_threads != 1 && num_threads != 2 && num_threads != 4 &&
num_threads != 8) {
        fprintf(stderr, "Número de threads deve ser 1, 2, 4 ou 8.\n");
        exit(EXIT_FAILURE);
    }

    char *arquivo_de_saida = NULL;
    for (int i = 2; i < argc; i++) {
        if (strcmp(argv[i], "-o") == 0 && i + 1 < argc) {
            arquivo_de_saida = argv[i + 1];
            break;
        }
    }

    if (arquivo_de_saida == NULL) {
        fprintf(stderr, "Arquivo de saída não especificado.\n");
        exit(EXIT_FAILURE);
    }

    int *data = NULL;
    int tamanho_total = 0;

    ler_arquivos(&data, &tamanho_total, argc, argv, 2);

    pthread_mutex_init(&mutex_thread, NULL);
```

```
    thread_tempo_execucao = malloc(sizeof(double) * num_threads);
    if (thread_tempo_execucao == NULL) {
         fprintf(stderr, "Erro ao alocar memória para tempos de execução das
threads.\n");
        exit(EXIT_FAILURE);
    }

    pthread_t *threads = malloc(sizeof(pthread_t) * num_threads);
     struct ThreadData *thread_data_array = malloc(sizeof(struct ThreadData)
* num_threads);
    int segment_size = (tamanho_total + num_threads - 1) / num_threads;

    for (int i = 0; i < num_threads; i++) {
        thread_data_array[i].data = data;
        thread_data_array[i].esquerda = i * segment_size;
        thread_data_array[i].direita = (i + 1) * segment_size - 1;
        if (thread_data_array[i].direita >= tamanho_total) {
            thread_data_array[i].direita = tamanho_total - 1;
        }
        thread_data_array[i].thread_id = i;
        thread_data_array[i].tempo_execucao = 0;

                    if   (pthread_create(&threads[i],   NULL,   thread_sort,
&thread_data_array[i]) != 0) {
            fprintf(stderr, "Erro ao criar thread %d.\n", i);
            exit(EXIT_FAILURE);
        }
    }

    for (int i = 0; i < num_threads; i++) {
        pthread_join(threads[i], NULL);
    }

    int current_segments = num_threads;
    int *indices = malloc(sizeof(int) * current_segments);
    if (indices == NULL) {
               fprintf(stderr,  "Erro  ao  alocar  memória  para  índices  de
mesclagem.\n");
        exit(EXIT_FAILURE);
    }

    for (int i = 0; i < current_segments; i++) {
        indices[i] = thread_data_array[i].esquerda;
    }

    int *dados_ordenados = malloc(sizeof(int) * tamanho_total);
    if (dados_ordenados == NULL) {
```

```
        fprintf(stderr, "Erro ao alocar memória para dados ordenados.\n");
        exit(EXIT_FAILURE);
    }

    for (int i = 0; i < tamanho_total; i++) {
        int valor_minimo = MAX_INT;
        int segmento_min = -1;
        for (int j = 0; j < current_segments; j++) {
            if (indices[j] <= thread_data_array[j].direita) {
                if (data[indices[j]] < valor_minimo) {
                    valor_minimo = data[indices[j]];
                    segmento_min = j;
                }
            }
        }
        if (segmento_min != -1) {
            dados_ordenados[i] = data[indices[segmento_min]];
            indices[segmento_min]++;
        } else {
            fprintf(stderr, "Erro durante a mesclagem dos segmentos.\n");
            exit(EXIT_FAILURE);
        }
    }

    memcpy(data, dados_ordenados, sizeof(int) * tamanho_total);

    free(dados_ordenados);
    free(indices);

    double soma_thread_exec_time = 0.0;
    for (int i = 0; i < num_threads; i++) {
             printf("Tempo de execução da Thread %d: %f segundos.\n", i,
thread_tempo_execucao[i]);
        soma_thread_exec_time += thread_tempo_execucao[i];
    }
         printf("Tempo total de execução das threads: %f segundos.\n",
soma_thread_exec_time);

    FILE *fp_out = fopen(arquivo_de_saida, "w");
    if (fp_out == NULL) {
              fprintf(stderr, "Erro ao abrir o arquivo de saída %s\n",
arquivo_de_saida);
        exit(EXIT_FAILURE);
    }

    for (int i = 0; i < tamanho_total; i++) {
        fprintf(fp_out, "%d\n", data[i]);
    }
```

```
    fclose(fp_out);
    free(data);
    free(thread_tempo_execucao);
    free(threads);
    free(thread_data_array);
    pthread_mutex_destroy(&mutex_thread);

    return 0;
}
```

3.2. Arquivos

Para execução do código consideramos 5 arquivos de entrada como correspondentes de teste (disponibilizados no repositório do github) para nossa compilação e obtenção de resultados. Logo, dentro de cada arquivo de entrada possuímos 1.000 inteiros presentes que servirão como valores a serem executados dentro das threads e posteriormente serem exibidos e ordenados no arquivo final.

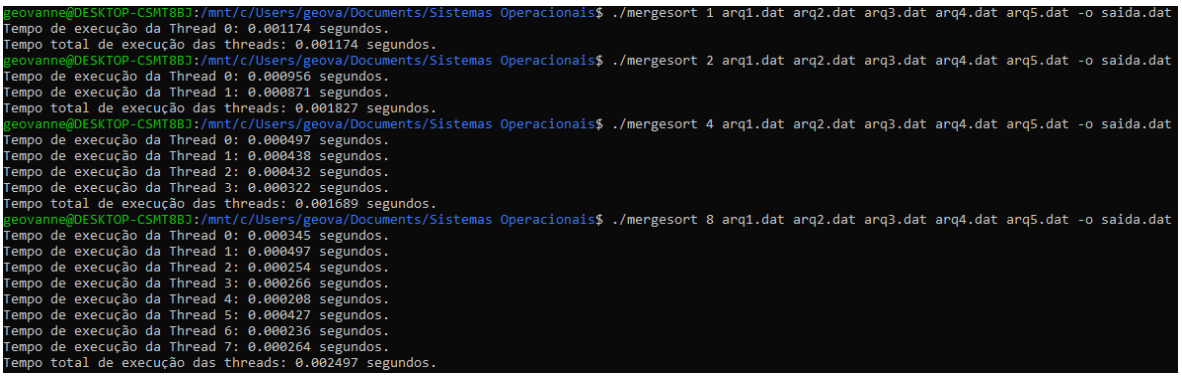
3.3. Arquivo de saída

Já no arquivo gerado como resultado da compilação do programa, temos os valores de compilação ordenados representando na saída do código, considerando que todas as threads dos arquivos tenham sido utilizadas.

3.4. Saída do programa

A saída do programa já exibe o formato de tempo de execução de cada thread seguido do cálculo de tempo total de execução, além de considerar trazer os valores em segundos, de modo que confirme o propósito do código.

Imagem 1 - Representação da saída do programa em teste



Fonte: Acervo pessoal, 2024.

4. Análise do projeto

4.1. Dados obtidos

Nesta seção apresentaremos os resultados obtidos em testes considerando entradas de 1, 2, 4 e 8 threads na compilação, tendo os retornos de cada uma das threads e o tempo de execução total descritos em tabela e um gráfico que representa um comparativos entre o tempo total de execução para as threads de entrada.

Tabela 1 - Valores de tempos de execução por threads em segundos

Thread	Thread 0 em s	Thread 1 em s	Thread 2 em s	Thread 3 em s	Thread 4 em s	Thread 5 em s	Thread 6 em s	Thread 7 em s	Tempo Total em segundos
1	0,001174								0,001174
2	0,000956	0,000871							0,001827
4	0,000497	0,000438	0,000432	0,000322					0,001689
8	0,000345	0,000497	0,000254	0,000266	0,000208	0,000427	0,000236	0,000264	0,002497

Fonte: Acervo pessoal, 2024.

Gráfico 1 - Comparativo de tempo de execução por threads

Fonte: Acervo pessoal, 2024.

4.2. Considerações

Tendo em vista que o programa pode considerar múltiplos valores de threads, podemos dizer que em sua análise geral de acordo com a tabela um e o gráfico de tempo de execução por thread, podemos afirmar que:

Com a utilização de uma única thread tivemos um retorno de resolução mais rápida, considerando que o conceito mergesort foi aplicado, em que, um número maior de threads ao executarem o arquivo precisam realizar divisão e conquista dos itens o que automaticamente acarreta em um tempo maior de execução para cada thread.

Entretanto, fazendo a utilização de 2 threads para compilação tivemos um tempo total em segundo superando a utilização de 4 threads, pois os tempos de compilação das threads 0 e 1, no programa com 2 threads foi mais do que o dobro da execução no programa com 4 threads, o que impacta no tempo total deixando assim a utilização de menos threads com um tempo total de execução mais do que o com mais elementos, contrariando o que aconteceu nos casos de 1 e 8 threads.

Já com a utilização de 8 threads, temos que o tempo total somado foi superior aos demais, mas a maioria dos tempos individuais foram menores, o que diminui a sobrecarga, em que se pode dizer que se tem benefícios para paralelização.

De modo geral, temos um programa funcional dentro do seu propósito, atingindo com os requisitos especificados e trazendo justamente como retorno um arquivo de saída com valores ordenados de acordo com a entrada de cada arquivo no programa. Portanto, ao longo do projeto foi possível entender a importância e aplicação de threads como um conteúdo geral.

5. Anexos

5.1. Link do repositório

O código está armazenado em um repositório da plataforma GitHub, em que, foi utilizado apenas para subir a versão final do código ao invés de realizar commits a cada alteração pois, ao longo do desenvolvimento consideramos apenas atualizações locais em máquina em virtude da realização de teste com a aplicação do WLS. Portanto, para acessar o repositório basta clicar [AQUI].

5.2. Link do vídeo

Já o vídeo está disponibilizado na plataforma youtube como não listado o que permite a visualização somente a partir de um link específico e para isto, basta clicar [AQUI]