CSC 452/552 Operations Systems

Project 3 Threads

Name: Tanmai Kumar Ghosh

Bronco ID: 114224102

Date: 11/12/2024

1. Project Overview

In Project 3, a multi-threaded merge sort algorithm was developed in C using pthreads, with a focus on creating efficient, thread-safe code. The primary goal was to parallelize the sorting process by splitting the input array into equal-sized segments, each handled by a separate thread. Additionally, the project involved creating a driver application to test performance across different array sizes and thread counts, along with a comprehensive analysis of the algorithm's scalability and efficiency.

2. Project Management Plan

a) Task 3: **Make Thread Safe**

In Task 3, a multi-threaded, thread-safe merge sort function (**mergesort_mt**) was implemented in C using **pthreads**. The input array was divided into equal chunks, with each chunk assigned to a separate thread to be sorted using the existing single-threaded merge sort (**mergesort_s**). Since each thread worked on a unique, non-overlapping section of the array, no locking mechanisms were necessary, eliminating the risk of data races. Once sorting was completed, **pthread_join** was used to synchronize the threads, followed by a sequential merge of the sorted chunks. This approach enabled efficient parallel processing, reduced synchronization complexity, and ensured robust memory management by correctly handling dynamic allocations.

b) Task 4: **Driver App**

The driver application was tested with various array sizes and thread counts. Results showed that larger arrays required more time to sort, while increasing the number of threads initially reduced the sorting time. However, after a certain point, further increasing the thread count led to slower runtimes due to overhead.

c) Task 5: **Add Bash File**

The provided bash file was modified to generate a plot displaying performance across multiple thread counts. The provided bash file returns error. **createplot.sh** is the listing of command lines generating the plot. The setting of the plot is declared in the **graph.plt** file. The generated plot is saved in the **student_plot.png** image file. To facilitate the plot generate capabilities, installation of **gnuplot** is required.

d) Task 6: **Analysis**

The reflection of the plot analysis is documented in the **analysis.md** markdown.

3. Project Deliveries

a) How to compile and use my code?

1. Run the project

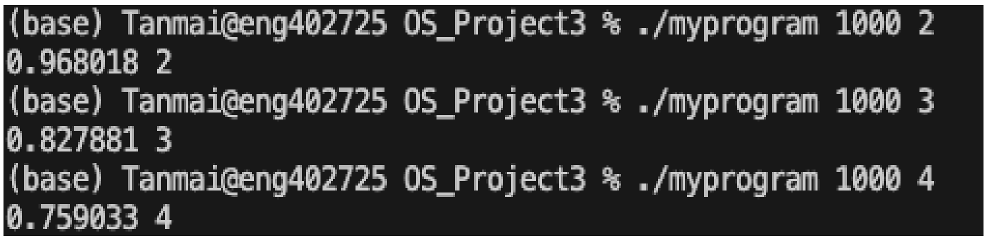
   ./myprogram <array_size> <num_threads>

2. Example Usage:

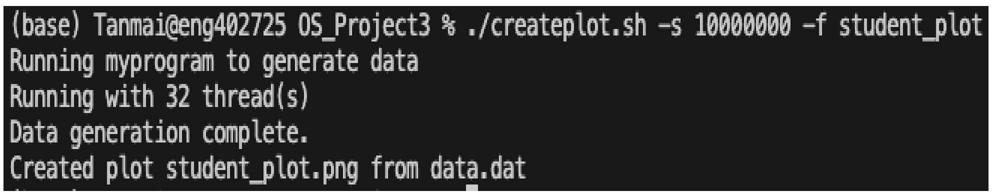
   ./myprogram 1000 2

b) Any self-modification: I have not made any changes except **lab.c, createplot.sh, and main.c.** In the **main.c**, it was required to free the memory allocation of A_(see the line 28.)

c) Summary of Results.



```
(base) Tanmai@eng402725 OS_Project3 % ./myprogram 1000 2
0.968018 2
(base) Tanmai@eng402725 OS_Project3 % ./myprogram 1000 3
0.827881 3
(base) Tanmai@eng402725 OS_Project3 % ./myprogram 1000 4
0.759033 4
```



```
(base) Tanmai@eng402725 OS_Project3 % ./createplot.sh -s 10000000 -f student_plot
Running myprogram to generate data
Running with 32 thread(s)
Data generation complete.
Created plot student_plot.png from data.dat
```

4. Self-Reflection of Project

This project provided practical experience with multi-threading in C using **pthreads**, focusing on memory management, thread synchronization, and performance optimization. A key lesson was the importance of balancing thread count to reduce overhead while maximizing parallelism. It also enhanced skills in performance testing and analysis using automated scripts and **gnuplot**, deepening understanding of thread-safe programming and scalable multi-threaded design.

5. Comments for Project (optional)

6. Use of AI for debugging (optional)