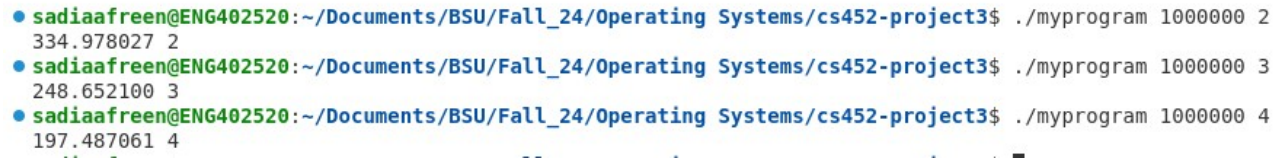
CSC 452/552 Operations Systems
Project 3 Threads
Name: Sadia Afreen
Bronco ID: 114248372
Date: 11.11.2024

1. **Project Overview:**
   In Project 3, a multi-threaded merge sort algorithm in C is implemented using pthreads, focusing on implementing efficient, thread-safe code. The main objective was to parallelize the sorting process by dividing the input array into equal-sized chunks, each processed by separate threads. The project also included creating a driver application for performance testing across various array sizes and thread counts, and performing a detailed analysis of the algorithm's scalability and efficiency.
2. **Project Management Plan**
   a) **Task 3 (Make Thread Safe):** In Task 3, a thread-safe, multi-threaded merge sort (`*mergesort_mt*`) is implemented using pthreads in C. The input array was split into equal chunks, with each chunk assigned to a separate thread for sorting using the existing single-threaded merge sort (`*mergesort_s*`). Explicit locking mechanisms are avoided since each thread operated on distinct, non-overlapping sections of the array, preventing data races. After sorting, threads are synchronized using *`pthread_join`* and performed a sequential merge of the sorted chunks. This design ensured efficient parallel processing, minimized synchronization complexity, and demonstrated robust memory management by handling dynamic allocations correctly.
   b) **Task 4 (Driver App):** Driver app is run using different sizes of array and different number of threads. It was observed that large arrays took longer time to sort and increasing number of threads reduced the time taken. However, after increasing the number of threads too many times, the runtime slowed again due to overhead.
   c) **Task 5 (Add Bash Files):** The bash file given is modified so that it can generate a plot to show performance for multiple threads. *gnuplot* needed to be installed.
   d) **Task 6 (Complete the Analysis):** After successful generation of the plot, the result is observed from the plot and the analysis written in Analysis.md.

3. **Project Deliveries**
   a) **Compilation and Using the code:** To compile the code, run `*make*` in the terminal first. After successful compilation, run the program using `.\*myprogram*`. If no argument is given, the program will not run, and show: "*usage: ./myprogram <array_size> <num_threads>*". The arguments are following:
      - <array_size> : Number of elements in the array
      - <num_threads> : Number of threads
   b) **Self-modification:** No self-modification was done except on the *lab.c* file.
   c) **Summary of Results:** The summary of the results after executing is given below:



Here 3 runs are shown with an array of size 1000000 and 2, 3, and 4 number of threads, respectively. The first part of the output is the runtime of the program and the second part is the number of threads. It can be observed clearly that when the number of threads increased, the runtime reduced drastically.

4. Self-Reflection of Project 3:

This project provided valuable hands-on experience with multi-threading concepts and the practical challenges of implementing concurrency in C using pthreads. It deepened my understanding of efficient memory management, thread synchronization, and performance optimization. One key takeaway was the importance of balancing the number of threads to minimize overhead while maximizing parallelism. The project also enhanced my skills in testing and analyzing performance using automated scripts and visualization tools like *gnuplot*. Overall, I gained a deeper appreciation for the complexity of thread-safe programming and learned strategies to design efficient, scalable multi-threaded solutions.