CSC 452/552 Operations Systems

Project 3 Threads

Name: Brooklyn Grant

Bronco ID: 114142733

Date: 11/6/2024

1. Project Overview

In this project, I extended an existing mergesort algorithm to work with multiple threads. The goal was to make the merge sort thread safe and look at its performance with different numbers of threads I implemented synchronization to handle concurrent processing and used shell scripts to visualize the sorting time against the number of threads.

1. Project Management Plan
   1. Task 3:
      1. I implemented the function called mergesort\_mt. The way I did this was I started by determining the chunk size by dividing the size of the array by the number of threads to use to determine how many chunks there would be to sort then I made a four loop that would list the start and the end of each chunk and create a thread for each chunk this thread would call a function that I created called calling\_mergesort\_with\_void(). I created this function because calling merge sort of a single thread took in multiple arguments and if we wanted to call it through a thread we must only have one argument it being args. Then I waited for the threads to finish sorting and merged the sorted chunks at the end. And voila the list is sorted! I ran the test and the test file and they all came back passing.
   2. Task 4:
      1. A screen shot of a computer screen

         Description automatically generated
      2. To make sure that this program ran as intended I ran the my program driver application to allow me to input an array size and a number of threads as arguments. This helped me run multiple test cases to analyze the sorting time with different thread counts by myself. It looked like 3 threads was slower than 2 consistently!
   3. Task 5:
      1. A graph with purple lines

         Description automatically generated
      2. I used the script and the PLT file to generate a graph called student\_plot.PNG. It allowed me to visualize the sorting performance across multiple threads. This graph is very interesting and it looks like the linearly upwards direction meaning the more threads the longer it takes.
   4. Task 6:
      1. All information included in my analysis.md file in my code.
2. Project Deliveries
   1. How to compile and use my code?
      1. To compile the project, run make. This will build the main program (myprogram). Use ./myprogram <array\_size> <num\_threads> to run the program directly, where <array\_size> is the size of the array to sort, and <num\_threads> is the number of threads. Run make check if you want to use the tests in the test-lab.c file.
   2. Any self-modification?
      1. Yes in main.c I edited the main function to free \_A at the end because it was causing issues running the program. I also removed the -fsanitize=address flag in the make file while generating because it was coming up with addresssantizer:deadlysignal errors all the time.
   3. Summary of Results.
      1. The results showed a pattern where increasing the number of threads improved performance but after a certain threshold the overhead from the additional threads outweighed the benefits.
3. Self-Reflection of Project

This project was very valuable in learning how to work with threads and understanding parallel processing period I learned that while adding threads can improve in performance there are diminishing returns due to overhead and synchronization costs using the graphing made it easier to visualize these effects which help my understanding of performance. I also had some challenges with address sanitizer, and I was able to learn more about resources and debugging issues with memory management like valgrind and GDP.

1. Comments for Project (optional)
2. Use of AI for debugging (optional)