CS232 Operating Systems
Assignment 4: Multi-threading

CS Program
Habib University

Fall 2023
Due Date: 3 December 2023 @ 11:59PM

1 Introduction

In this assignment, we will put the knowledge of multi-threading using POSIX Threads API to use and implement a multi-threaded file processor.

2 Objective

The goal of this assignment is to develop a multi-threaded C program that processes a large dataset concurrently. You will use synchronization mechanisms to ensure thread safety and efficient resource utilization.

3 Required Tasks

Write a C program that reads a dataset from a file and processes it using multiple threads. The dataset will be a text file containing a linear list of integers from 1 to 1000000 (data_small.txt) / 1 to 10000000 (data_medium.txt) / 1 to 100000000 (data_large.txt) elements. The program should accomplish the following tasks:

1. Read the dataset from a file (e.g., "data_small.txt") into a dynamic array. **Do not use large static array instead malloc the exact amount of memory as required**. The name will be given from the second command line argument. **Do not hard code the filename**.
2. Calculate the sum, the minimum and maximum values in the dataset using a single thread. Note the amount of time it takes to process it.
3. Create a specified number of worker threads (let's say n) to process the dataset. Use the POSIX Threads API. Each thread will process a portion of the dataset (see point 4). The number of threads will be given from the third command line argument. If the user does not provide the number of worker threads to launch, the program should launch 4 threads.
4. Partition the data into n worker threads such that each thread has a chunk of data to work on.
5. Implement a thread-safe mechanism to compute the sum of the dataset. Each thread should maintain a *partial sum*, and the main thread should combine these *partial sums* to compute the *final sum*. Calculate the amount of time it takes to compute the sum using multiple threads.
6. Implement a thread-safe mechanism to find the maximum and minimum values in the dataset. Calculate the amount of time it takes to compute the minimum and maximum values using multiple threads.

7. Write the sum, minimum, and maximum values using both single and multi-threaded version of your code to the output console.
8. Ensure proper resource management and thread synchronization to avoid data races and memory leaks.
9. Implement proper error handling for file operations and thread creation.
10. Do not forget to free the memory you had created dynamically.

4 Input

4.1 Single Threaded Program

Your program should take one additional command line argument. The first command line argument is the name of the program. The second command line argument will be the data file to be processed. If the user fails to provide the desired number of arguments, appropriate message should be printed on the console to guide the user about the issue.

4.2 Multi-Threaded Program

Your program should take two additional command line arguments. The first command line argument is the name of the program. The second command line argument will be the data file to be processed. The third command line argument will be the number of threads to launch for processing. If the user fails to provide the desired number of arguments, appropriate message should be printed on the console to guide the user about the issue. If the user does not provide the number of threads, then default 4 threads should be launched for processing.

5 Timing your C code

Below we share code snippets to help you time your C code to check its performance. Note that the timing code should be called many times and the average time should be used as a measure. This is because often times there are overheads that might cause the performance to drop so averaging ensures that the overall behavior of your code over several runs is calculated as a measure of its performance. It also helps remove any bias that may arise.

5.1 For single thread program

Use the `clock` function to help obtain a reasonable time estimate for a single threaded program as shown in the following listing.

```c
#include <stdio.h>
#include <time.h>

int main() {
    // Start the timer
    clock_t start_time = clock();

    // Your code to be timed

    // Stop the timer
    clock_t end_time = clock();

    // Calculate elapsed time
    double elapsed_time = (double)(end_time - start_time) / CLOCKS_PER_SEC;

    // Print the elapsed time
    printf("Elapsed time: %f seconds\n", elapsed_time);

    return 0;
```

```
}
```

Listing 1: Timing your C code in a single-threaded program.

5.2 For multiple threads in a thread safe manner

Timing a piece of C code in a *thread-safe* manner involves using the `clock_gettime` function to measure time precisely. Here's an example of how to time a code segment within a thread-safe context:

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

#define NUM_THREADS 4

void* thread_function(void* arg) {
    // Start the timer
    struct timespec start_time, end_time;
    clock_gettime(CLOCK_MONOTONIC, &start_time);

    // Your code to be timed
    // ...

    // Stop the timer
    clock_gettime(CLOCK_MONOTONIC, &end_time);

    // Calculate the elapsed time
    double elapsed_time = (end_time.tv_sec - start_time.tv_sec) +
                          (end_time.tv_nsec - start_time.tv_nsec) / 1e9;

    // Print the elapsed time from this thread
    printf("Thread %ld took %f seconds\n", (long)arg, elapsed_time);

    pthread_exit(NULL);
}

int main() {
    pthread_t threads[NUM_THREADS];

    for (long i = 0; i < NUM_THREADS; i++) {
        if (pthread_create(&threads[i], NULL, thread_function, (void*)i) != 0) {
            perror("Thread creation error");
            exit(1);
        }
    }

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

    return 0;
}
```

Listing 2: Timing your C code in a multi-threaded program.

6 Submission and Rubric

6.1 Submission

You will submit the following:

1. `file_processor_singlethreaded.c` containing your complete C for single threaded implementation.

2. `file_processor_multithreaded.c` containing your complete C for multi-threaded implementation.
3. `makefile` for compiling and running of your C code. It should contain `compile`, `build` and `clean` targets.
4. PDF report on the data structures and algorithms you have used in your implementation.
5. Timing comparison of processing the data using a single thread vs multiple threads. Try with different configurations and report your findings.

Compress all of these files into a zip file and rename it with your CS registration number (CSxxyyyy.zip). Submit the zip file on the Assignment 4 submission module on LMS.

6.2 Rubric

The details are given in the following. Note that you might also be called for a viva at the instructor's discretion.

6.2.1 Marks

- single threaded implementation works as expected: 10 marks
- multi-threaded implementation works as expected: 20 marks
- multi-threaded synchronization works as expected: 20 marks
- file and thread creation functions have proper error handling: 20 marks
- makefile: 10 marks
- initialization using command line arguments works correctly: 10 marks
- submission (code legible, commented, PDF correclty formatted): 10 marks

6.2.2 Penalties

- code doesn't compile: -100 marks
- code has warnings (compile with -Wall): -20 marks
- code has memory leaks: -30 marks
- makefile without required targets: -10*number of missing targets
- program crashes: -30 marks
- late submission: -20 marks for missing deadline + -10*num days

Mark obtained = max (marks+penalties, 0)

7 Using chatGPT or other AI software

You are not allowed to use any AI software to obtain the code for this assignment. Appropriate tool will be used to evaluate your submission for AI tool usage. If you are found using such a tool, you will be given a straight 0 and an Academic Conduct will be filed against you for academic dishonesty.

8 Plagiarism Policy

We have zero tolerance for plagiarism. Every submission will be screened using a plagiarism detection software. If there is any evidence of plagiarism, the case will be reported to the Office of Academic Conduct and all offenders will get a 0. This is applicable even for cases when the code is copied or a significant amount has been obtained from an online repository on open source platforms like bitbucket or github without proper attribution. In case you are taking any material from online sources, we expect that a proper credit/reference is given to the source.