6. Multi-thread Programming

Overview

In this assignment, you will practice multi-thread programming using Pthread(POSIX thread). You will learn how to create threads and use mutex locks to ensure safe concurrent access to shared data.

Here are some hints for your reference:

1. **Threads Creation:**

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void*), void *arg);
```

2. **Thread Termination:**

```
void pthread_exit(void *value_ptr);
```

3. **Wait for Thread Termination:**

```
int pthread_join(pthread_t thread, void **value_ptr);
```

4. **Initialize or Destroy a Mutex:**

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

```
int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);
```

```
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

5. **Lock and Unlock a Mutex:**

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

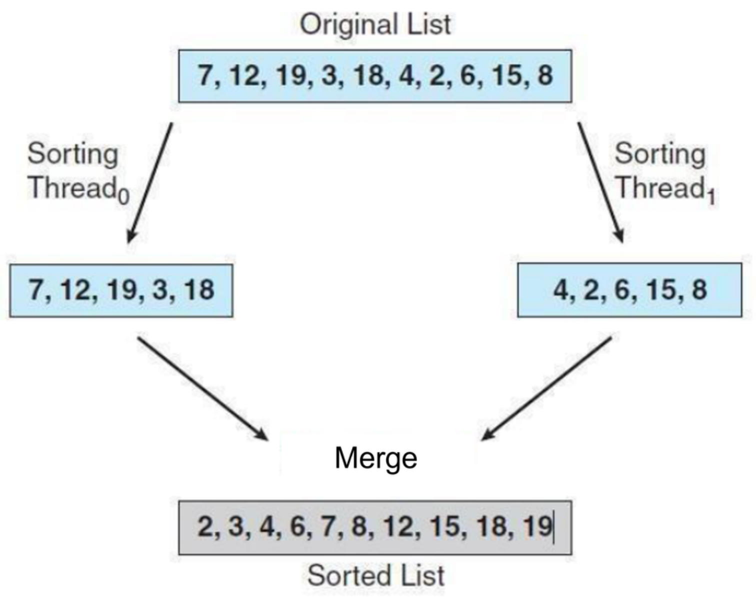
```
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

Part I (Merge Sort)

Descriptions

In Part I, you must implement a multi-threaded merge sort.

The program works as follows:
• A list of integers is divided into two smaller lists of equal size. You have to create **two** separate threads (sorting threads) to sort each sub-list in ascending order using the merge sort algorithm.
• Then, the two sub-lists are merged into a single sorted list after both sorting threads are done.

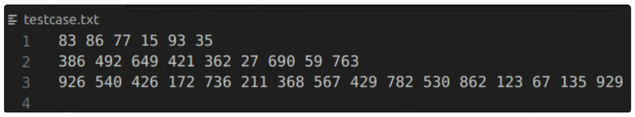


You will receive the file `testcase.txt` containing multiple lines, each with a series of integers representing a distinct test case. Your task is to process these test cases and record your results in `output.txt`. Ensure that the format of your output matches the input format provided in `testcase.txt`.

Sample I/O

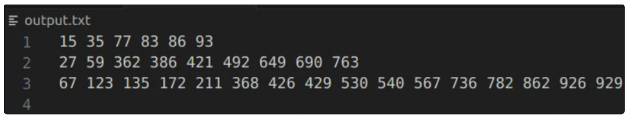
Numbers are separated by a space in each test case and different test cases are separated by a newline (\n).

Sample Input:


testcase.txt
```
83 86 77 15 93 35
386 492 649 421 362 27 690 59 763
926 540 426 172 736 211 368 567 429 782 530 862 123 67 135 929

```

Sample Output:


output.txt
```
15 35 77 83 86 93
27 59 362 386 421 492 649 690 763
67 123 135 172 211 368 426 429 530 540 567 736 782 862 926 929

```

Notes

- The output filename must be `output.txt`.
- Only merge sort using multi-thread programming is acceptable.
- Each test case can contain up to 10,000 integers, with each integer ranging between -2,147,483,648 and 2,147,483,647.
- The program should accept input and output file paths as command-line arguments(argc, argv).
- Use `gcc -o hw6_1 -pthread hw6_1.c` to compile your code.
- To run the program and produce the output, use the command `./hw6_1 testcase.txt output.txt`.

Part II (Dot Product)

Descriptions

Below is a program that calculates the dot product of two vectors `vector1` and `vector2`, each containing 10,000 elements. We need to use of pthread mutex to ensure safe concurrent access to shared data. Your objective is to modify certain sections of this code to ensure its correct execution

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

#define VECTOR_SIZE 10000
#define NUM_THREADS 8

// Define the vectors
int *vector1, *vector2;

int inner_product = 0;

// Structure to hold thread-specific data
typedef struct {
    unsigned int thread_id;
    unsigned int start;
    unsigned int end;
} ThreadData;

// Function to calculate the inner product for a portion of the vectors
void* calculate_inner_product(void* arg) {
    ThreadData* data = (ThreadData*)arg;

    for (int i = data->start; i < data->end; i++) {
        inner_product += vector1[i] * vector2[i];
    }

    pthread_exit(NULL);
}

int main() {
    pthread_t threads[NUM_THREADS];
    ThreadData thread_data[NUM_THREADS];

    // Initialize the vectors
    vector1 = (int *)malloc(sizeof(int)*10000);
    vector2 = (int *)malloc(sizeof(int)*10000);
    for(int i = 0; i < VECTOR_SIZE; i++) {
        vector1[i] = 1;
        vector2[i] = 2;
    }

    int elements_per_thread = VECTOR_SIZE / NUM_THREADS;
    int remaining_elements = VECTOR_SIZE % NUM_THREADS;

    // Create threads and calculate inner product in parallel
    for (int i = 0; i < NUM_THREADS; i++) {
        thread_data[i].thread_id = i;
        thread_data[i].start = i * elements_per_thread;
        thread_data[i].end = (i + 1) * elements_per_thread;
        if (i == NUM_THREADS - 1) {
            // Include remaining elements in the last thread
            thread_data[i].end += remaining_elements;
        }

        pthread_create(&threads[i], NULL, calculate_inner_product, &thread_data[i]);
    }

    // Wait for all threads to finish
    for (int i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }

    printf("Inner Product: %d\n", inner_product);

    return 0;
}
```

It's important to maintain the main structure of the program; a complete rewrite is not necessary. Focus on identifying the specific lines of code that are susceptible to race conditions and apply pthread mutex to safeguard them.

Question: After implementing modifications to the sample code in Part II, consider whether the multi-threaded version will outperform the serial version provided below. What factors contribute to the difference in performance?

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

#define VECTOR_SIZE 10000

// Define the vectors
int *vector1, *vector2;

int inner_product = 0;

int main() {
    // Initialize the vectors
    vector1 = (int *)malloc(sizeof(int)*10000);
    vector2 = (int *)malloc(sizeof(int)*10000);
    for(int i = 0; i < VECTOR_SIZE; i++) {
        vector1[i] = 1;
        vector2[i] = 2;
    }

    // Calculate inner product
    for (int i = 0; i < VECTOR_SIZE; i++) {
        inner_product += vector1[i] * vector2[i];
    }

    printf("Inner Product: %d\n", inner_product);

    return 0;
}
```

Notes

- Use `gcc -o hw6_2 -pthread hw6_2.c` to compile your code.

Submission

Please submit a **zip** file to E3, which contains your source code and report.

Source Code (60%):

- hw6_1.c (40%)
- hw6_2.c (20%)

The program must implemented using C.

Make sure your code can be compiled on **Ubuntu 22.04 AMD64**.

Make sure your outputs are correct.

Report (40%):

- Part I (15%)
  - Explain how you implement your code clearly with screenshots of your functions.
- Part II (25%)
  - Describe which lines of code you modify with detailed explanations. (15%)
  - Answer the above question. (10%)

The name of the zip file should be <student_id>.zip, and the structure of the file should be as the following:

<stduent_id>.zip
|- <student_id>/
|- hw6_1.c
|- output.txt
|- hw6_2.c
|- hw6.pdf

For questions, please contact TA Yuwei Liao <awei6209@gmail.com>