Homework 3

CS211 - Fall 2024

(You should try to answer first and then compare your solution with the ChatGPT solution)

This homework is designed to help you become familiar with key concepts related to threads and how to write a multi-threaded program.

Question 1: Below is the code that is slightly modified from what we wrote in the class (the file hw3_1.c is attached on the Canvas)

```
1  #include <stdio.h>
2  #include <pthread.h>
3
4  void* foo(void* args) {
5      printf("He he %d\n", (int)args);
6      //exit(NULL);       // question 1b
7      return NULL;
8  }
9
10 int main() {
11
12     int n_thread = 10;
13     pthread_t* thread_ids = malloc(n_thread*sizeof(pthread_t));
14
15     for (int i = 0; i < n_thread; i++)
16          pthread_create(&thread_ids[i], NULL, foo, i);
17
18     //exit(NULL); // question 1c
19     for (int i = 0; i < n_thread; i++)
20          pthread_join(thread_ids[i], NULL);
21 }
```

(a) Run this code multiple times. Do you get the same result? Why?

(b) Uncomment line 6, then run the code multiple times again. How does the result change? Why?

(c) Uncomment line 18 (while keeping line 6 commented), then run the code multiple times again. How does the result compare to part (a)? Why?

Question 2: In the code below (as in the file hw3_2.c attached on Canvas):
(i) First, we implement a data structure that represents a sorted array of integers and a function to insert an integer into this sorted array.
(ii) Next, in the main function, we create two threads. The first thread inserts the even numbers 0, 2, …, 2n, while the second thread inserts the odd numbers 2n + 1, 2n - 1, …, 3, 1 into the sorted array.
(iii) Finally, we print the sorted array. If everything works correctly, the output will show the numbers 0, 1, …, 2n + 1 in ascending order.

Your tasks are:
(a) Read the code to understand how it works.
(b) Compile and run the code with different values of n (e.g., 10, 20, 50, 100) to verify if the code works as expected.
(c) Explain the results you observe.

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

#define MAX_SIZE 100000  // Maximum size of the array

// Shared sorted array and its size
int sorted_array[MAX_SIZE];
int size = 0;

// Function to insert an integer into the sorted array
void insert_sorted(int data) {
    // Find the correct position to insert the new element
    int i = size - 1;
    while (i >= 0 && sorted_array[i] > data) {
        sorted_array[i + 1] = sorted_array[i];
        i--;
    }
    // Insert the new element into the correct position
    sorted_array[i + 1] = data;
    size++;  // Increase the size of the array
}
// Function to print the sorted array
void print_array() {
    for (int i = 0; i < size; i++) {
        printf("%d ", sorted_array[i]);
    }
    printf("\n");
}

// Thread function to insert even numbers from 0 to 2n
void* insert_even_numbers(void* arg) {
    int n = *(int*)arg;
    for (int i = 0; i <= 2 * n; i += 2) {
        insert_sorted(i);
    }
    return NULL;
}

// Thread function to insert odd numbers from 2n + 1 to 1
void* insert_odd_numbers(void* arg) {
    int n = *(int*)arg;
    for (int i = 2 * n + 1; i >= 1; i -= 2) {
        insert_sorted(i);
    }
    return NULL;
}
```

```c
// Main function
int main(int argc, char* args[]) {
    int n = atoi(args[1]);

    // Create two threads
    pthread_t even_thread, odd_thread;

    // Start the thread to insert even numbers
    pthread_create(&even_thread, NULL, insert_even_numbers, &n);

    // Start the thread to insert odd numbers
    pthread_create(&odd_thread, NULL, insert_odd_numbers, &n);

    // Wait for both threads to finish
    pthread_join(even_thread, NULL);
    pthread_join(odd_thread, NULL);

    // Print the final sorted array
    print_array();

    return 0;
}
```

Question 3: If A and B are nxn matrices, then the product matrix C is defined as

$$C(i,j) = \sum_{k=1}^{n} A(i,k) * B(k,j)$$

(a) Write a function to calculate C given matrices A and B using the formula above.

(b) Write a function to calculate C using k threads, where each thread calculates n/k rows of matrix C.

(c) Randomize matrices A and B as 1000×1000 matrices, and try different values of k (e.g., 2, 4, …, 20) to measure how multi-threaded programming improves the calculation speed.