CS431 Operating Systems
Programming Assignment 3 - Multi-Threaded Programming Using Pthreads

1 Overview

Multi-threaded programs offer several benefits over their single-threaded counterparts. Better responsiveness, higher CPU utilization, more throughput, and greater scalability are just a few. The purpose of this programming assignment is to gain practice writing multi-threaded programs using the POSIX Thread (Pthread) library.

2 Requirements

Refer to the program code given in the file *multithreaded_addition.c*. The program as written first creates a global array of 10,000 integers and populates each element with a pseudorandom number between 1 and 100. It then uses a single thread (created using `pthread_create()`) to compute and output the sum of all integers in the array.

Your job is to modify the program such that the task of computing the sum of the 10,000 integers is divided among five threads. Each thread will compute a partial sum using a portion of the entire original array, with those partial results collected and added together to compute and output a single sum, just as before.

Each thread will operate only on its own portion of the array, and among the five threads each element of the array will only be read and counted once. Therefore, with a correctly written program there is no possibility threads will either interfere with each other or double count any array element in the overall sum.

3 Thread Implementation

After creating a new thread, the program calls `thread_join`, which causes the main program to block until the thread completes its execution. In your program the main thread should join all five threads it creates in order to retrieve the results of their partial sum computation.

I suggest using an array of type `pthread_t` as in the example program *threads.c*, although you are not required to do so.

3.1 Thread Parameters

The parameter to the thread function is a pointer to a struct named `limits`, which contains three fields:

- An `int` variable named *begin* that represents the starting index of the subarray to sum.
- An `int` variable named *end* that represents the ending index of the subarray to sum.
- An `int` variable named *sum* that represents the sum of all elements of the array between *begin* and *end*, inclusive.

Each thread should sum the elements of the array from *begin* to *end* and place the result in *sum*.

3.2 Void Pointers

In Pthreads the parameter to the thread function is always of type `void*`, or a void pointer. A void pointer is nothing more than a pointer, or a variable referencing an address in memory, to what could be any data type. It could point to an `int`, a `double`, a `char`, an abstract data type, or any other type. For that reason, it is necessary to cast a void pointer to some other type of pointer before it can be dereferenced.

For instance, if you have a variable named `ptr` that is declared to be a void pointer but actually points to an `int`, you must first cast `ptr` to an `int` pointer, then dereference the `int` pointer to retrieve the value stored in memory, e.g.,

```
int value = *(int*)ptr;   // (int*) casts the void* variable ptr to an
                          // int pointer. The preceding * deferences
                          // the int pointer to get an int value.
                          // You can then assign that value to an int
                          // variable
```

In the thread function, the parameter `args` is a void pointer. To use it in the function, it is first cast to a `struct limits*`, or a pointer to a `limits` struct. Because `args` is a pointer to an aggregate type, the arrow notation (`->`) is used to access the fields, i.e.,

```
int first = ((struct limits*)args)->begin;
int last = ((struct limits*)args)->end;
```

4 Building Programs with Pthreads

When building the program, be sure to add the linker option `-lpthread` so that the Pthread library is correctly linked to the executable program, i.e.,

```
gcc -o multithreaded_addition multithreaded_addition.c -lpthread
```

5 Deliverables

Submit the C code source file for your program through the WorldClass assignment dropbox no later than the due date specified in the weekly assignment. **Be sure to add your name to the header block of comments**.

6 Academic Integrity

I recognize that there are only so many ways to write code that meets the requirements of this assignment, and you may consult with other students in this class on the general approach to a solution. Having said that, sharing code, pseudocode, or other implementation details is not permitted, and I expect all work to be your own as evidenced by:

- Program structure, control flow, variable and function names (aside from C library functions, of course), etc. should be manifestly distinct from others' work.
- Your program should be commented in a manner sufficient for me to ascertain that you understand what the code you submit does.