Assignment #4: Multithreaded Summation

CS201 Spring 2020
10 points
due Friday, Mar. 6th, 11:59 pm
second-chance (for 9 points) due Friday, Mar. 20th, 11:59 pm

1 Threads

You'll write a multithreaded C program that will sum the elements in a global array. Each thread will be like a Minion: you'll give it a starting index and an ending index, and it will sum the values in the global array starting at the start index and ending at the end index. You'll then add up the results from each thread to get the total sum.

1.1 Declarations

You will have to do this assignment on Linux or macOS. Use these `#define` statements:

```
#define NUM_THREADS 4
#define NUM_ELEMENTS 64000
```

and declare a global array:

```
double A[NUM_ELEMENTS];
```

Declare this structure:

```
typedef struct {
  int firstIndex;
  int lastIndex;
  double theSum;
} SumInfo;
```

and write a multithreaded program to sum the elements in the array `A`.

You should write this function:

```
void *doSum(void *data);
```

The `data` parameter will point to a `SumInfo` struct. The function will add up the values in the array `A` from index `firstIndex` to index `lastIndex`, inclusive, and put the result in `theSum`.

Initialize your array with random values in the interval $[0, 1)$. For Linux and macOS, the function `double drand48()` returns a uniformly distributed double in the in the range $[0, 1)$.

1.2 Compiling and linking

You'll have to include `stdlib.h` and `pthread.h`.

On Linux, you will have to compile your program this way:

```
$ gcc msum.netid.c -lpthread
```

On macOS, you do not need the `-lpthread`.

2 What to Do

Do the actual implementations in C of the program described above. Put your structure definition and function prototype in `msum.netid.h`, and put your code in `msum.netid.c`; submit these two files to Blackboard.

You will need to write `main()` function. Your `main()` will create the threads and then sum the results from each thread. Leave your `main()` in your file when you submit it.

3 Testing

How will you know if this is working? Have each thread print the start index and end index. Make sure they look correct. You can also add up the values directly in your main and compare results. The two results might not match exactly (as a consequence of dealing with floating-point numbers), but they should be "close". Specifically, you should see $|sum_1 - sum_2| \leq tol * |sum_1|$, where tol is, for example, 10^{-8}.

4 Extra Credit

For a challenge, and five points extra credit, write a multi-threaded merge sort. Use the same structure as the summation assignment, but have each thread sort a region of the global array, specified by `firstIndex` and `lastIndex`. After the threads are done, you still have to do the merge!

Put your code in a file `msort.netid.c` and your declarations in `msort.netid.h`.