Sorting Contest using Threads - Prog 4
CECS 325 – System Programming with C++
Fall 2022
Due: 10/19/2022

In program 3 you created a sort that was (hopefully) faster than the BubbleSort that I gave you. You can use that for this program. If you did not submit a working sort for assignment #3, you can use the BubbleSort I gave you in program 3. (Use mySortA.cpp)

This assignment uses threads to perform parallel processing – allowing you to speed up your sort program substantially. This sample program below shows 4 threads. You should use 8 threads in your program.

Your program will read in 1 million unsorted numbers from numbers.txt into an array. Then you will split the array into 8 sections – each section will have 125,000 numbers. You will call pthread_create( ) 8 times and pass each section of the array to the thread to sort using the sort algorithm you used in the previous assignment. Once all the threads have returned, you will merge 2 adjacent sections into a sorted super section, then 2 adjacent super sections into another super section, and finally the 2 remaining super sections into one single sorted array. Then print the array to a file. Then you will call the verifySort (that you wrote) to verify the output file is sorted. verifySort( ) will print the number of integers in the file and display if they are sorted or not.

You will run this shell file (sortRace.sh – that you wrote) and submit a screenshot of the results:

:: start shell file
generate 1000000 100000 999999
time sort numbers.txt > sysSort.out
time mySort numbers.txt mySort.out
verifySort mySort.out :: this function shows how many numbers in the file and if they are sorted.

What to submit:

1) Your program source code for the sort (mySort.cpp)
2) Your program that verifies if the file is sorted (verifySort.cpp)
3) Your shell (batch) file (sortRace.sh)
4) Screenshot showing the output of the shell command

Below is a program that creates 4 threads, each of which print out one word from a quote from the Matrix movie: “there is no spoon”. You can examine this program to see how threads work in a C++ program.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <thread>
#include <iostream>
#include <unistd.h>
using namespace std;
```

```
// Create a structure to pass parameters to print_message in pthread_create
struct message{
    char *quote; // character pointer - null terminated character array
    int index;
    int wait;
};

// This is the function that will be called in pthread_create.
// Notice this returns a void pointer
void *print_message(void *ptr )
{
     message *arg = (message *) ptr;
     //cout << "-->Entering Thread "<<arg->index<<endl;
     cout << arg->quote<<endl;
     //cout << "<--Exiting Thread "<<arg->index<<endl;
     return NULL;
}

char quotes[][10] = {"there", "is", "no","spoon"};

int main()
{
    message argList[4]; // array of arguments to pass to each thread
    for (int i=0; i<4; i++)
    {
        argList[i].quote = quotes[i];
        argList[i].index = i;
        argList[i].wait = rand()%5;
    }

     pthread_t thread0, thread1, thread2, thread3;

     int  iret0, iret1, iret2, iret3;

    /* Create independent threads each of which will execute function */

     iret0 = pthread_create( &thread0, NULL, print_message, (void*)
&argList[0]);
     iret1 = pthread_create( &thread1, NULL, print_message, (void*)
&argList[1]);
     iret2 = pthread_create( &thread2, NULL, print_message, (void*)
&argList[2]);
     iret3 = pthread_create( &thread3, NULL, print_message, (void*)
&argList[3]);

    // The return value of pthread_create is 0 if successful and non-zero if
there is a problem
    /*
     cout << "Thread 0 returns:"<<iret0<<endl;
     cout << "Thread 1 returns:"<<iret1<<endl;
     cout << "Thread 2 returns:"<<iret2<<endl;
     cout << "Thread 3 returns:"<<iret3<<endl;
    */
     /* Wait till threads are complete before main continues. Unless we  */
     /* wait we run the risk of executing an exit which will terminate   */
     /* the process and all threads before the threads have completed.   */
```

```
    pthread_join( thread0, NULL);
    pthread_join( thread1, NULL);
    pthread_join( thread2, NULL);
    pthread_join( thread3, NULL);

    return 0;
}
```