Sorting Contest using pThreads - Prog 4
CECS 325-02 – System Programming with C++
Spring 2024
Due: 3/26/24

In program 3 you created O(n-squared) soring algorithm – the Bubble Sort even though President Obama told you not to. It was painfully slow partly because you were only using one CPU even though your computer has more than one CPU. You will take advantage of multi-processing in this assignment using the exact same sort function that you used in Prog 3. You will see a remarkable improvement in time.

This assignment uses pthreads 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.dat into an array. Then you will logically 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, and continue to do that until all the smaller sorted sections are in one large sorted section – which will be the entire array of 1 million numbers. Then print the array to a file. Then you will call sort -c -n to verify the output file is sorted.

In fact you can use the same sortrace.sh script file for this assignment. You should see substantial improvement in your sorting times.

You will run the shell file (sortRace.sh – that you used for Prog 3) and submit a screenshot of the results:

What to submit:
1) Your new pthread program source code for the sort (mySort.cpp)
2) Your generate.cpp program (same one you used in Prog 3.
3) The sortrace.sh file you use
4) The sortrace.log file

Below is a program that creates 4 pthreads, 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 pthreads work in a C++ program.

When you compile your program, you need to include “-pthread” on the compile line

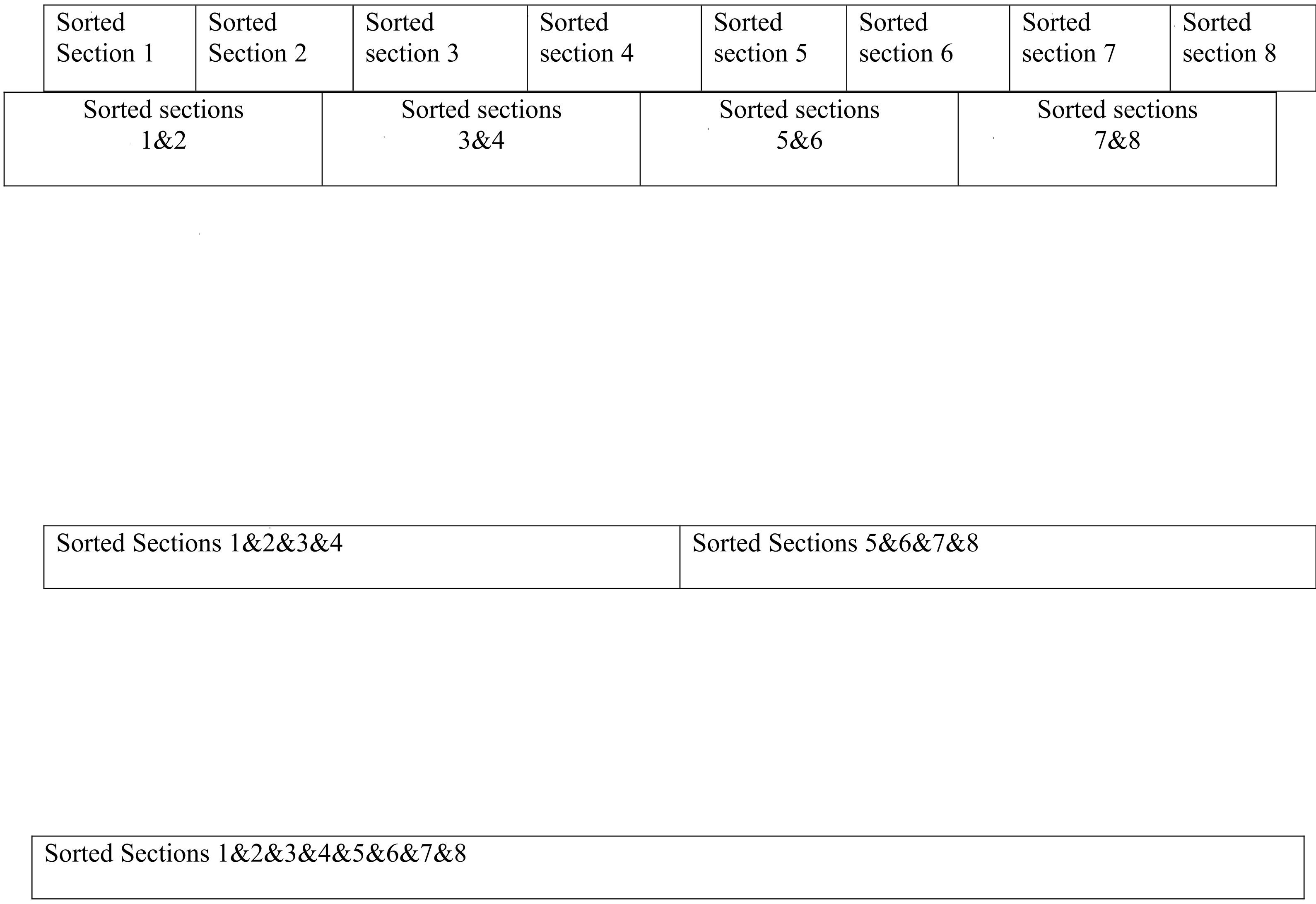
```
$ c++ matrix.cpp -o matrix -pthread
```

```
//#include <stdio.h>
//#include <stdlib.h>
#include <pthread.h>
#include <iostream>
#include <string>
```

```
using namespace std;

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

int fibo(int n)
{
     if (n == 0 || n == 1)
          return 1;
     else
          return fibo(n-1) + fibo(n-2);
}

// This is the function that will be called in pthread_create.
// Notice this returns a void pointer
void *print_message(void *ptr )
{
     // cast the incoming unknown pointer to a message pointer
     message *arg = (message *) ptr;
     // print out the index and quote
     cout << "("<<arg->index<<"):"<<arg->quote << endl;
     cout << "("<<arg->index<<"):"<<"Fibo:"<<arg->fiboNum<<" = "<<fibo(arg-
>fiboNum)<<endl;
     return NULL;
}

// initialze 4 strings in an array called quotes
string quotes[4] = {"there", "is", "no", "spoon"};

int main()
{
    message m0;
    m0.quote = quotes[0];
    m0.index = 0;
    m0.fiboNum = 40;

    message m1;
    m1.quote = quotes[1];
    m1.index = 1;
    m1.fiboNum = 41;

        message m2;
    m2.quote = quotes[2];
    m2.index = 2;
    m2.fiboNum = 37;

    message m3;
    m3.quote = quotes[3];
    m3.index = 3;
    m3.fiboNum = 38;
```

```
     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*) &m0);
     iret1 = pthread_create( &thread1, NULL, print_message, (void*) &m1);
     iret2 = pthread_create( &thread2, NULL, print_message, (void*) &m2);
     iret3 = pthread_create( &thread3, NULL, print_message, (void*) &m3);

    // 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;
}
```

Diagram of how the 8 sorted sections will be merged together:



Extra requirement:
Since you already have a working function called bubble, you will not make any changes to that function. You will copy the bubble function from your Prog 3 and use the exact same function in Prog4. However you will notice that pthread_create( ) has 4 parameters.

Parameter 1: the address of the thread that is created
Parameter 2: NULL – we will not worry about this
Parameter 3: the name of the function to run in the thread – we will further discuss this below
Parameter 4: a pointer to a structure that holds the parameters for the function in parameter 3

So you will create a bridge function that pthread_create( ) will call and then bridge function will call bubble.

Here’s an example of the bridge function:

```
struct sortStuff{
     int *start;
     int size;
};

void *bridge(void *ptr)
{
     sortStuff *arg  = (sortStuff *)ptr;
     bubble(arg->start, arg->size);
}



int main()
{
     int *numbers = new int[1000000]; // dynamic memory
     pthread_t t0;
     sortStuff ss0;
     ss0.start = &numbers[0];  // start at the beginning of the array
     ss0.size = 125000;  // sort the first 125000 numbers only

     sortStuff ss1;
     ss1.start = &numbers[125000];  // start at 125,000th numbe in array
     ss1.size = 125000;  // sort 125000 numbers only

     pthread_create(&t0, NULL, bridge, (void*) &ss0);
     pthread_create(&t0, NULL, bridge, (void*) &ss1);

     /*
     sort the rest of the sections
     then merge all sections - 2 at a time
     */
}
```

Additional requirements: You will use dynamic memory to create the array in your program and you will clean up that memory when you no longer need it.