Sorting Contest using Threads - Prog 4
CECS 325-02 – System Programming with C++
Fall 2024
Due: 10/31/24

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 threads to perform parallel processing – allowing you to speed up your sort program substantially. This sample program below shows 4 threads. You should use 16 threads in your program.

Your program will read in 1 million unsorted numbers from numbers.dat into a dynamically allocated array. Then you will logically split the array into 16 sections – each section will have 62,500 numbers. You will create 16 threads and pass each section of the array to the thread to sort using the Bubble 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.

You will run the new shell file (sortrace.sh) listed at the bottom of this assignment.

Additional requirements:

- Create a global variable which will keep track of all the swaps that take place when your program runs. There will probably be about 16 billion swaps – int type goes up to 2.1 billion – so you cannot use an int for this.

- This variable needs to be global so all 16 processes can access it. Keep track of all the swaps for each process, and at the end of the bubblesort, print out the "[process name" and the swap count for the process. You get to determine what the process name is and pass that through the bubblesort parameter list when you create a thread.

- Also add the number of process swaps to the global variable keeping track of ALL swaps. Since there are 16 threads that will be trying to access the global swap count, you need to use a mutex variable to protect the global variable and ensure only one thread can access it at a time.

- You will use dynamic memory to create the main array of 1,000,000 in your program and you will clean up that memory when you no longer need it. You will also use dynamic arrays for any other arrays you need – for example when you do the merge.

What to submit:
1) Your new thread program source code for the sort (sort.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

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

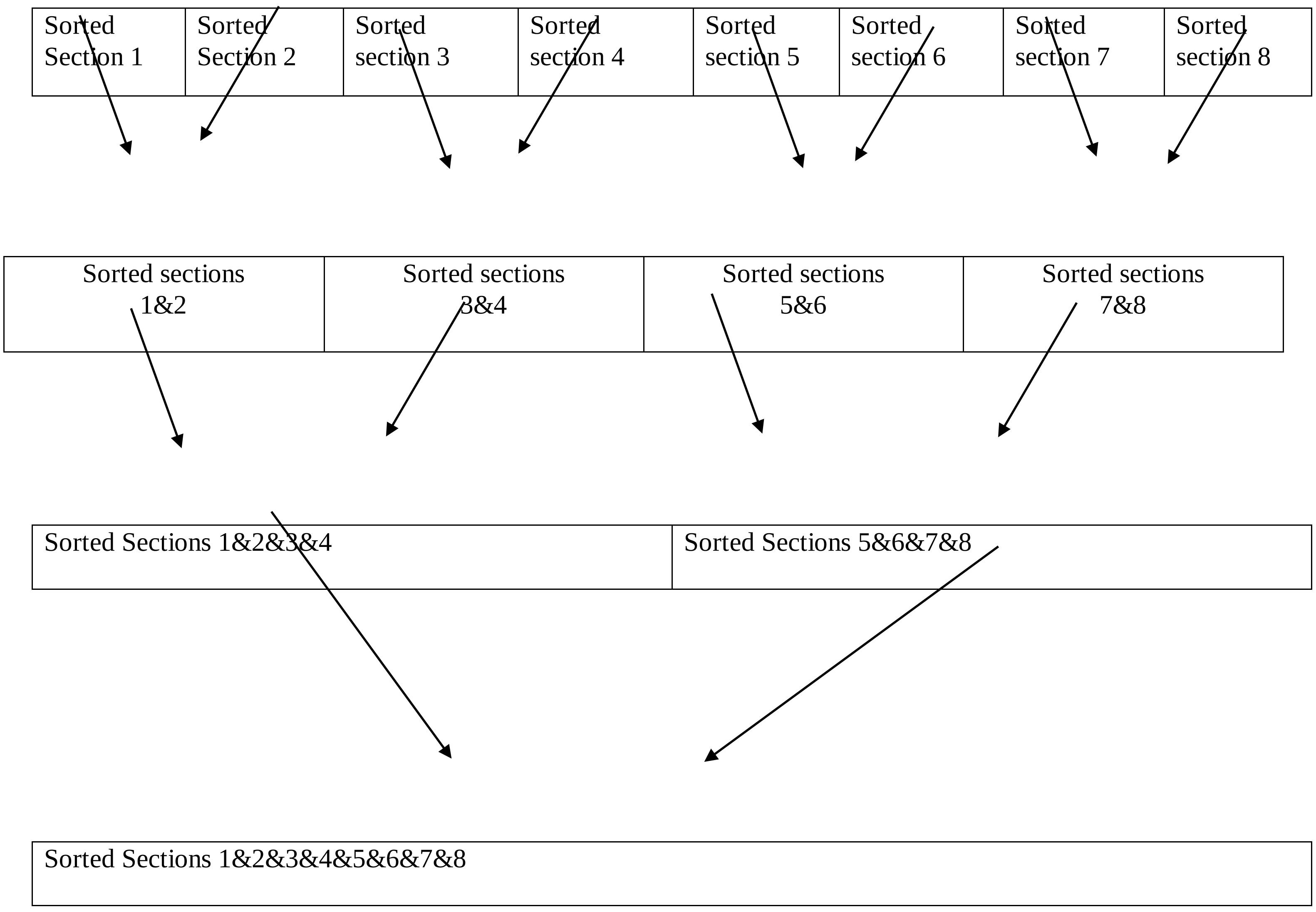
$ c++ matrix.cpp -o matrix -pthread

Below is a program that creates 4 threads, each of which print out one word from a quote from the Matrix movie: "I know Kung Fu". You can examine this program to see how threads work in a C++ program.

```
// filename: matrix.cpp
// compile line:  %c++ matrix.cpp -o matrix -pthread

#include <string>
#include <thread>  // include the thread library
#include <iostream>
using namespace std;

// this is the function that is called by each thread
void print_message(string quote, int threadNum;)
{
     cout "thread:"<< threadNum<< " "<< quote<<endl;
}


int main()
{
     // Create 4 threads. Threads launch a process upon creation.
     // Pass the function name and all parameters.
     // In this case, a function name and string.

     string quote1 = "I";
     string quote2 = "know";
     string quote3 = "Kung";
     string quote4 = "Fu";

     thread thread1(print_message, quote1, 1);
     thread thread2(print_message, quote2, 2);
     thread thread3(print_message, quote3, 3);
     thread thread4(print_message, quote4, 4);

     // force the threads to join back to the main program
     thread1.join();
     thread2.join();
     thread3.join();
     thread4.join();

     return 0;
}
```

Notice: You are using a merge function in this program, NOT the Merge Sort. You will sort each section using the Bubble Sort, then you will merge 2 adjacent sections into one section.

Diagram of how the 8 sorted sections will be merged together. Remember – you are to use 16.



This is the shell script you should use. Some commands are slightly different for Windows vs Mac.

```
############################################################
# this file should be called sortrace.sh
# it must have execute privilege set to run
# run it as a background task like this:
#          $ rm sortrace.log            # start with fresh log file
#          $ sortrace.sh >> sortrace.log &    # run in the background
############################################################
echo "=======Start======="
whoami
date
```

```
echo My machine has this many processors
nproc                   # this is for Windows machines
sysctl -n hw. Ncpu      # this is for Mac machines
echo Generating 1000000 random numbers
sleep 1
./generate 1000000 -1000000 1000000  # you have to write generate.cpp
sleep 1
echo Starting system sort
sleep 1
{ time sort -n numbers.dat > systemsort.out; } 2>> sortrace.log          # this line is for Windows
{ time sort -n numbers.dat > systemsort.out; } 2>&1>> sortrace.log     # modification for Mac
sleep 1
echo Starting my sort
sleep 1
{ time ./mysort numbers.dat  mysort.out; } 2>> sortrace.log    # this line is for Windows
{ time ./mysort numbers.dat  mysort.out; } 2>&1>> sortrace.log     # modification for Mac computers
sleep 1
ls -l systemsort.out
ls -l mysort.out
echo Comparing systemsort.out to mysort.out
diff systemsort.out mysort.out 2>> sortrace.log
echo All done with diff compare
echo "=======End======="
date
```

You must write the following programs

- generate.cpp
  - **Purpose:** generate a file called numbers.dat
  - **Description:** The numbers.dat file will have COUNT random numbers between MIN and MAX
  - **Usage**: generate COUNT MIN MAX
  - **Example**: generate 1000000 -1000000 1000000
  - This program accepts 3 command line arguments as shown above
  - If there are not 3 command line arguments, the program fails and prints error message
- mysort.cpp
  - **Purpose**: read numbers from input file, sort, write sorted numbers to the output file
  - **Description**: Uses bubble sort function on an array of integers
  - **Usage**: mysort numbers.dat mysort.out
  - It accepts 2 command line arguments which is the input file name and the output filename. It expects there to be 1,000,000 numbers in the input file. However, if there is a parameter -test on the command line, then it will expect 10,000 numbers in the input file.
  - **Example**: mysort numbers.dat mysort.out -test <- this will expect 10000 numbers in numbers.dat

What to submit:

- mysort.cpp
- generate.cpp
- sortrace.sh
- sortrace.log

Helpful Links:

Mutex in C++: https://www.geeksforgeeks.org/std-mutex-in-cpp/

Merge 2 sorted arrays: https://www.geeksforgeeks.org/merge-two-sorted-arrays/