

PROGRAMMING ASSIGNMENT 5

DUE: Monday October 26, 11:59 PM

DESCRIPTION

Create a program to sort an array of integers using the Merge Sort algorithm. The program will use several threads for different tasks and semaphores to coordinate these tasks. Please refer to the diagram provided for this assignment for a better understanding of the different components of your program.

Your program **MUST** include the following:

1. Five threads. The thread ids **MUST** be “th1”, “th2”, “th3”, “th4”, and “th5”.
2. The function associated with “th1” **MUST** be named “createArray”. It will use an input parameter (array size n) via a command line argument to create a one-dimensional array to be sorted. The array will consist of n random integers between -1000 to 1000, inclusive. After the array is created, “th1” will print to the console each element separated by a tab ‘\t’. See sample output below.
3. The function associated with “th2” **MUST** be named “sortFirstHalf”. It will sort the first n/2 elements of the array by calling the “mergeSort” function which will be described below.
4. The function associated with “th3” **MUST** be named “sortSecondHalf”. It will sort the second n/2 elements of the array by calling the “mergeSort” function.
5. “th2” and “th3” **MUST** run concurrently. This will provide speed-up of the sorting process. See sample output below.
6. The function associated with “th4” **MUST** be named “mergeTwoHalves”. It will combine first half of the sorted arrays with second half of the sorted array by calling the “merge” function which will be described below.
7. The function associated with “th5” **MUST** be named “printSortedArray”. It will print all the elements in the sorted array separated by a tab ‘\t’. See sample output below.

Your program will also include two functions. As the diagram for this assignment shows “th2” and “th3” will first call the function “mergeSort” and then the function “merge”.

8. The function “mergeSort” will divide the array into equal halves repeatedly until there is only one element left and then call the “merge” function to sort the array.

9. Below is a pseudocode for “mergeSort”. To confirm that “th2” and “th3” are running concurrently, as soon as you enter the “mergeSort” function and if “th2” is running, you must print “*” followed by the first element of the array that is passed into “mergeSort. If “th3” is running then print “**” followed by the first element of the array that is passed into “mergeSort. Use a tab (“\t”) in between prints. See sample output below.

```

1. procedure mergeSort( var a as array )
2.
3.   if ( n == 1 ) return a
4.
5.   var l1 as array = a[0] ... a[n/2]
6.   var l2 as array = a[n/2+1] ... a[n]
7.
8.   l1 = mergeSort( l1 )
9.   l2 = mergeSort( l2 )
10.
11.  return merge( l1, l2 )
12. end procedure

```

IF is called by thread 2
 Print (“*”+a[0]+ “\t”)
 ELSE IF called by thread 3
 Print (“**”+a[0]+ “\t”)

10. The function “merge” will combine the divided arrays sorting them in *ascending order*. Below is a pseudocode for “merge”.

```

1. procedure merge( var a as array, var b as array )
2.
3.   var c as array
4.   while ( a and b have elements )
5.     if ( a[0] > b[0] )
6.       add b[0] to the end of c
7.       remove b[0] from b
8.     else
9.       add a[0] to the end of c
10.      remove a[0] from a
11.    end if
12.  end while
13.
14.  while ( a has elements )
15.    add a[0] to the end of c
16.    remove a[0] from a
17.  end while
18.
19.  while ( b has elements )
20.    add b[0] to the end of c
21.    remove b[0] from b
22.  end while
23.
24.  return c
25.
26. end procedure

```

COMPILER COMMAND

Your program must be able to compile with the command **gcc assign5.c -o assign5 -lpthread**

Sample INPUT

./assign5 200

Where 200 is the input size n for the random array.

Sample OUTPUT

```

Random generated integers are in the following order:
-474 -120 -910 490 236 354 667 522 -31 368 -862 -742 -559 -157 941 781 625 466 -63 344 467 -891 -849
595 284 -20 585 478 -832 -113 714 -748 324 884 388 -881 -285 -26 641 241 -659 779 499 -661 -822 -4
880 883 -539 -386 146 929 -276 -145 891 -993 835 -48 42 561 396 -245 813 -281 -884 120 -605 -169 -966
964 629 993 372 127 890 551 -878 -991 353 -416 -820 -944 -931 -539 -89 517 -975 -698 -532 67 863 864
80 232 -861 496 -648 -465 884 -987 -872 512 553 580 -885 442 58 875 8 961 -985 -254 574 -916 287
516 158 -210 -657 626 857 784 46 -207 -5 186 846 -96 721 286 -93 406 -646 -983 -95 -451 16 513
570 -976 473 997 -873 46 639 534 87 -284 -677 13 980 738 -258 25 -470 746 -790 934 -351 488 219
14 -107 574 -312 356 -878 784 -132 -897 285 898 -980 612 -580 296 146 587 -909 26 -870 -930 -680 -119
6 -592 -817 -137 -659 -611 -649 560 583 -199 -310 748 -844 -630 452 24

----- THREAD 2 AND 3 ARE CURRENTLY SORTING THE NUMBERS -----
*-474 *-474 *-474 *-474 *-474 *-474 *-474 *-474 *-474 *-474 *-474 *-474 *-474 *-474 *-474 *-474 *-474 *-474 *-474 *-474 *-474
*-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910
*-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910
*-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910
*-872 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910
*-872 *-872 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910
*-872 *-910 *-872 *-872 *-872 *-872 *-872 *-872 *-872 *-872 *-872 *-872 *-872 *-872 *-872 *-872 *-872 *-872 *-872 *-872 *-872
*-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985
*-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985
*-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985
*-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985
*-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985
*-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910
*-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910 *-910
*-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985
*-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985 *-985

The elements of array have been sorted as follows:
-997 -993 -991 -985 -983 -976 -975 -964 -944 -931 -930 -916 -910 -899 -896 -890 -897 -891 -884 -881 -880 -878 -878
872 -870 -862 -861 -849 -844 -832 -822 -820 -817 -805 -790 -740 -742 -698 -680 -677 -673 -661 -659 -659 -657 -649
648 -646 -630 -611 -605 -595 -592 -576 -559 -539 -532 -516 -500 -474 -470 -465 -451 -416 -386 -351 -312 -310
285 -281 -276 -254 -250 -245 -210 -207 -204 -199 -169 -157 -145 -137 -132 -120 -119 -113 -107 -96 -95 -93 -89
63 -48 -31 -26 -20 -13 -5 -4 8 16 24 25 26 42 46 46 50 67 87 96 114 120 127
46 146 158 186 207 219 232 238 241 284 285 286 296 308 324 344 353 354 356 368 372 380 396
06 442 452 466 467 473 478 480 486 489 500 503 505 512 513 517 522 534 551 553 560 561
74 574 587 612 625 626 629 639 641 667 784 714 721 738 746 748 764 779 781 803 804 813 835
46 857 863 864 875 884 890 891 898 929 934 941 961 980 993 997

```

This section is printing all the random number generated by th1.

This section is printed by the function “mergeSort” As you can see * and ** are mixed that is because th2 (*) and th3 (**) are run CONCURRENTLY.

Note: you must try a large array size and or run the program couple times to see this pattern.

This section is printing the entire sorted array by th5.

SUBMISSION

Submit assign5.c file through Canvas.

GRADING CRITERIA

- **Early submission (by Oct 24, 11:59pm) will receive a 5% bonus**
- **Late submissions will lose points as stated on the syllabus – 10% for each day late.**
- **A program that does not compile will result in a zero!**
- **Functionality:**
 - 3% - for your full name in the first line as a comment and code organization
 - 15% - for implementing the merge sort algorithm correctly
 - 8% - for “mergeSort” function
 - 7% - for “merge” function
 - 1% - for every thread and associated function named as requested. 7% total
 - 7% - for every thread and associated function that performs correctly. 35% total
 - 10% - for correctly creating and initializing semaphores.
 - 25% - for placing the semaphores correctly within the program
 - 5% - for correctly releasing semaphores at the appropriate place during execution.

PLAGIARISM!

Your program must be your original work, as stated and described in the syllabus. If you are unsure about whether some open source code can be used, contact the TA.

You are allowed to use any portion of code that is included in the course slides published in canvas.