CSC139: Operating System Principles
Spring 2018
Second Assignment
Thread Synchronization
Due Wednesday, 14th, 2018

In this assignment, you will experiment with thread creation and synchronization using both busy waiting and semaphores. You will use the POSIX thread library. Your job is to write a program that generates a large array of random numbers as specified below and then searches for the minimum number in the array using the four different schemes described below. In this assignment you will be measuring the time taken in each case. You are given a template (MTFindMin_template.c) that has the timing code in it. Use the given template.

Your program will take the following three command-line arguments:

1. The array size (a positive integer between 1 and 100 000 000)
2. The number of threads (a positive integer between 1 and 16)
3. The index at which a zero must be placed in the input array. If this index is negative, no zero should be placed.

Your program will perform the following steps:

1. Generate the input data by first filling the array with random numbers in the range 1 to MAX_RANDOM_NUMBER (defined in the given template). Your program should then check for the third command-line argument. If it is a valid non-negative index, your program should set the value at that index to zero.
2. Do sequential search for the minimum (without creating any threads) and print the search time. Your sequential search should terminate as soon as a zero is found.
3. Based on the number of threads (T), compute the indices for dividing the array into T equal divisions. For each division i, indices[i][0] should be set to the division number (i), indices[i][1] should be set to the start index, and indices[i][2] should be set to the end index. For example, if the array size (N) is 1000 and T is 2, the indices will be indices[0][0]=0, indices[0][1]=0, indices[0][2]=499
   indices[1][0]=1, indices[1][1]=500, indices[1][2]=999
4. Search for the minimum using multi-threading by creating T threads and having each thread search one division of the array. You will need to implement the following three synchronization schemes and print the search time for each scheme (as shown in the given template):
   First Scheme: The parent waits for all children.
   Second Scheme: The parent keeps checking on the children in a busy waiting loop and terminates as soon as one child finds a zero or when all children have completed scanning their divisions. The parent must then cancel all threads. **Do not use any semaphores in this scheme**.
   Third Scheme: The parent waits on a semaphore that gets signaled by one of the children either when that child finds a zero or when it detects that all children have completed scanning their divisions. The parent must then cancel all threads.

Use the following command to compile your code:
g++ -O3 MTFfindMin.c -lpthread -o MTFindMin
Note that we are using the –O3 option to enable a high level of compiler optimization.

When your program is working correctly, run the following tests on a *dedicated machine*:
1. Array size = 100M, T=2, index for zero =50M+1
2. Array size = 100M, T=4, index for zero =75M+1
3. Array size = 100M, T=8, index for zero =88M
4. Array size = 100M, T=2, index for zero =-1 (no zero)
5. Array size = 100M, T=4, index for zero =-1 (no zero)
6. Array size = 100M, T=8, index for zero =-1 (no zero)

The tests must be run on a dedicated machine to ensure the accuracy of the timing results. You can find dedicated machines for running the timing tests in Labs 2001, 2003 and 3009.

Note: In the code that implements the second scheme **(**parent keeps checking on the children in a busy-waiting loop), make sure that the shared variables that the busy-waiting loop checks on are declared as *volatile*. This will prevent the compiler from optimizing out the checking code and replacing the loop with an infinite loop (while(true)).

Submission

Submit the following on Canvas:

1. Your source code in one source file named MTFindMin.c
   Please don't use any other file names.
   In your source file, please include a header that has the following information:
   Your name
   Your section number
   The OSs on which you have tested your program (Linux, Mac, etc). **Your code must compile and work correctly on the Athena machine in our ECS computing environment.**
   The hardware configuration of the machine that you ran your tests on. To get cpu information on Linux, you can use one of the following commands:
   lscpu
   cat /proc/cpuinfo

2. A brief report including the following:
   a. Six tables summarizing the results of the tests that you ran (one table for each test)
   b. A discussion of the results and the conclusions that you have drawn. Don't forget to report the number of threads on the machine that you have used. The interpretation of the results will be highly dependent on the number of hardware threads on your machine. Timing tests must be run on a dedicated machine.

Extra Work (25%)

Re-implement this assignment using multiple processes instead of multiple threads. Use the fork() system call to create multiple processes. Each process will search for the minimum value in one division of the array. The array should be divided exactly the same way, that is, the code that divides the array in the parent will be exactly the same. Also, the parent process should synchronize with the child processes using the same synchronization schemes described above (wait for all, busy wait, and semaphore-based synchronization). In this case, shared variables must be placed in shared memory instead of being declared as global variables, because each process has its own address space. After doing the implementation, run the same timing experiments that you ran with multiple threads. Then write a brief report discussing your results and comparing the multi-thread solution with the multi-process solution in terms of speed and implementation complexity. The extra work will be graded out of 25, and the required work will be graded out of 100. So, you can earn up to 25% more

points on this assignment if you do the extra work. The number of extra points will depend on the quality of your work.