Ferris Hussein

Testing Implementation

randomGenerator():

- In this function I take in an array size within the parameters and I allocate the index with values from one to the array size sequentially. I decided to swap every index for array sizes up to 1000 randomly. For example, an array size of 900 will have 900 swaps of two random indexes between 0 and 900. An array size over 1000, will be swapped a total of 75% of the array size. For example, an array size of 10,000 would have 7,500 swaps.

test1():

- In this test I generate a random array of size 1000 with my random number generator. I specified the size of my subarray (in the variable 'block') as 250. I made my target 1001 which should not exist in my random array. Resultingly, the program exits and prints "TARGET NOT FOUND" for processes and threads.

test2():

- In this test I wanted to run a simple test. I generate a random array of size 54321 and I made my target 12345 with a subarray size ('block' size) of 250. Depending on whether I compile with 'make proc' or 'make thread', my program accurately finds the index of the target through processes or threads.

test3():

- In this test I generated a random array of size 2344 and I made my target value 1. My subarray size ('block' size) is 250. I wanted to see if I can find the very first value through threads and processes. Depending on whether I compile with 'make proc' or 'make thread', my program accurately finds the index of the target through processes or threads.

test4():

- In this test I generated a random array of size 100,000 and I made my target value 32,982. My subarray size ('block' size) is 250. I wanted to see how my program would respond to larger array sizes. Depending on whether I compile with 'make proc' or 'make thread', my program accurately finds the index of the target through processes or threads.

test5() and test6():

- Tests 5 and 6 Ire for experimental purposes. I wanted to compare processes/threads with different subarray sizes ('block' sizes) as Ill as look for the largest target possible in the array.
- In test5, I generated a random array of size 2019 and I made my target value 2019. My subarray size ('block' size) is 100 in this case.

- In test6, In test5, I generated a random array of size 2019 and I made my target value 2019. My subarray size (‘block’ size) is 250 in this case.
- Depending on whether I compile with ‘make proc’ or ‘make thread’, my program accurately finds the index of the target through processes or threads in both tests.
- When I run allTests(), I can see how the number of processes/threads affects how long the search takes.

allTests():

- In this function I successfully calculated the average time, minimum time, maximum time, standard deviation, and the time it takes to switch between processes/threads. In this function I invoke tests 1-6, 100 times and calculate the data specified accordingly. The statistics are printed accordingly at the bottom after the program is finished running and they are unique to each test.

tradeOff():

- In my main function in searchtest.c, I commented out the tradeOff() function to be invoked if needed. This is an extra test implementation where I test the time it takes to run both processes and threads with the same parameters. I capped off my array size at 20000 to compare the first 80 processes and threads amongst each other. My subarray size (‘block’ size) is 250 in this case. I created a loop that continuously checks the next 250 elements. For example, if I are currently at index 500, I will search for my index up to 750 (depending on ‘make proc’ or ‘make thread’). Then, my next iteration will begin at 750 and will search for the

Extra Credit:

- For the extra credit portion of this project I want to calculate average amount of time it takes to switch between processes and kernel threads. I calculated this by taking the average of each test (after 100 iterations) and dividing it by the number of processes/threads. The results are printed at the bottom after the program is finished running and is dependent on whether I compile with ‘make proc’ or ‘make thread’. The results are unique to each test.