Assignment 2

**Exercise 2: Threads and Concurrency**

**Q1. Threads and concurrency (10 points)**

1. Using Amdahl’s Law, calculate the speedup gain of an application that has a 70% parallel component for (a) two processing cores and (b) eight processing cores.

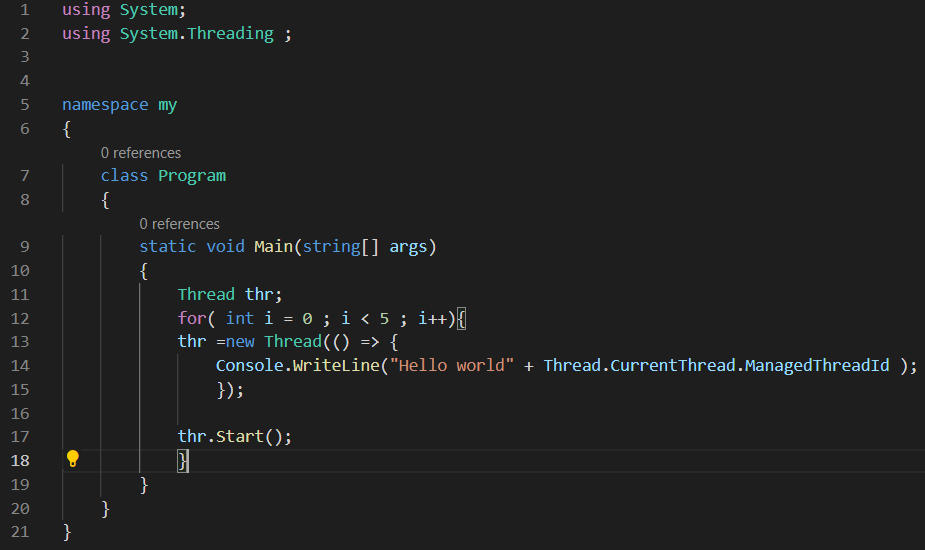
Amdahl’s Law is speedup <= 1 / (S + (1 –S) / N) where S is the portion of the application that must be performed serially, and N is the number of processing cores.

(a) For 2 processing cores and 60 percent parallel component S is 30 percent or 0.3 and N is 2. So, speedup is <= 1/ (0.3 + (1 –0.3)/2) - Speedup gain is 1.538 .   
  
(b)For 4 processing cores and 60 percent parallel component S is 30 percent or 0.3 and N is 8. So, speedup is <= 1/ (0.3 + (1 –0.3)/8) - Speedup gain is 2.58 .

2. Why a thread creation is  faster than process creation? hint: what resources are used when a thread is created? How do they differ from those used when a process is created?  
thread creation is  faster than process creation because :

* Process means any program is in execution but Thread means segment of a process.
* Process consume more resources but Thread consume less resources.
* Process is isolated but Threads share memory.
* There are many "tables" that a Process must have ( open file table …)   
  If you create a process all these tables have to be initialized.  
  If you create a thread they don't (because the thread uses the ones of its process).

3. Write a short program that prints "Hello world [thread id]"  from five different threads, where thread id is the id of the thread



**Q2.** **Multi-threaded merge sort (20 points)**

Merge sort is an efficient, general-purpose, comparison-based sorting algorithm (read: [**https://www.programiz.com/dsa/merge-sort)**](https://www.programiz.com/dsa/merge-sort)).

Design a multi threaded version for the merge sort (hint: look at the "fork and join" strategy learnt in the class). Assume that your function takes an array of integers.

**a) Describe your multi threaded merge-sort algorithm in details.**

We would Take merge Sort Algorithm and Divide the array to \*N parts , run merge sort algorithm on every single section every section on one thread ,  
join threads together , subsequently we have N sorted sections   
Run merge function on these sections to get one sorted array   
  
  
  
N- Number of Threads

**b) Add a descriptive diagram visualizing your multi threaded merge sort algorithm.**

Array Size = 8

Sorted Size 8

UnSorted size2

Merge Sorted

divide Unsorted

Sorted size 2

1

1

1

1

1

1

1

1

Sorted size 2

divide Unsorted

Merge Sorted

UnSorted size2

1

1

1

1

Sorted size 2

divide Unsorted

Merge Sorted

UnSorted size2

UnSorted size2

divide Unsorted

1

1

1

1

Merge Sorted

Sorted size 2

Thread 4

Thread 2

Thread 3

1 Thread

Threads Number N=4

**c) Write a pseudo code for your multi threaded merge-sort version.  Add descriptive comments.**

**MergeSort(arr[], l, r)**

If r > l

**1.** Find the middle point to divide the array into two halves:

middle m = l+ (r-l)/2

**2.** Call mergeSort for first half:

Call mergeSort(arr, l, m)

**3.** Call mergeSort for second half:

Call mergeSort(arr, m+1, r)

**4.** Merge the two halves sorted in step 2 and 3:

Call merge(arr, l, m, r)

ArrayNumbers Size n

threads <- threads array Size N   
for i in Len(Threads)   
 subArray - > from (i \*(n/N) to (i +1\*(n/N))) כאן מקצים לכל תראיד חלק מסוים מהמערך

create Thread (thread[i], mergeSort ,subArray)

sortedArr <- merge All Sub Arrays Together

**Q3. Distributed multi-threaded text search (70 points)**

In this exercise your will implement a multi-threaded version of text search in a file.

Your program (DistributedSearch) takes four parameter and should work as follow:

> DistributedSearch.exe *textfile.txt StringToSearch nThreads Delta*

**textfile:** the text file to search in

**StringToSearch:** the string to search in the text file (it contains only letters & digits and no spaces)

**nThreads:** How many threads will be created for the search

**Delta:** The delta (distance) between letters to search in the file. For example, the string HELLO WORLD" contains the string "ELLO" with delta=0, and the string HLO with delta=1 (**H**E**L**L**O**)

The program output is the location of the string in the file, otherwise output "**not found**".

Note: the beginning of the file is at location 0 and so on.

For example: if a textfile.txt contains "HELLO WORLD"

> DistributedSearch.exe textfile.txt "ELL" 1 0

output: 1

 > DistributedSearch.exe textfile.txt "OW" 1 1

output: 4

 > DistributedSearch.exe textfile.txt "ABC" 1 1

output "not found".

**Search Function -**  
 search function try to find the sub string , we have subString and arr it finds the first appearance of the first letter in the subString in the array and saves its index , and then make delta jumps meanwhile adding the letters to candidate (string variable) ,after iterations in the number to the subString size if the candidate is eqauls we stop the Program and print the index .

Main -

In The main ; we have buffer size 10k and thread pool each 10k we read from the file , we start a new thread with search function and the buffer we read .

Checking suspected areas ( the connection between threads ) after we done we check all the connections between threads with buffer in size (2n) - 2 times of substring length , while the first n in the buffer are the last n chars in the first thread and the other n in the start of the second buffer , like that we cover all the file .

Guidelines:

1) Choose your strategy. A simple strategy is to split the search between the n threads. Note that there are several strategies.

2) You may search only for the first occurrence of the string in the file.

3) Write a code with detailed comments.

4) Describe your algorithm in details in the word/pdf file of this exercise.

5) Add a a descriptive diagram visualizing your distributed search.

6) You may test your program on sample text files (e.g., Alice’s Adventures in Wonderland, by Lewis Carroll can be found here: [**http://www.gutenberg.org/files/11/11-0.txt**](http://www.gutenberg.org/files/11/11-0.txt))

7) Note that we may test your program with very large text files. Consider it when you design your program.

8) We will test your program on large text files.

9) Part of the grade will be given for the efficient programs in term of speed.

**10) There will be a bonus of up to 10 points for the fastest program**

**Good Luck!**