**OS – Assignment 2 – Theoretical (& code) Part**

*Question n1 :* ***Amdahl's Law***

(1) Given a program where 75% of the execution time is parallelizable, calculate the theoretical maximum speedup using 4 processors.

(2) A program has a parallelizable portion of 60% and a sequential portion of 40%. How many processors are needed to achieve a speedup of 3?

(3) If a program has 90% of its execution time parallelizable and the remaining 10% is sequential, calculate the speedup achieved with 32 processors.

(4) For a program where 85% can be parallelized, what is the maximum theoretical speedup if there are an infinite number of processors?

(5) How does the speedup change for a program with 80% parallelizable code if the number of processors is increased from 4 to 32? Provide a detailed calculation.

**(1)** Formula of speedup given by:

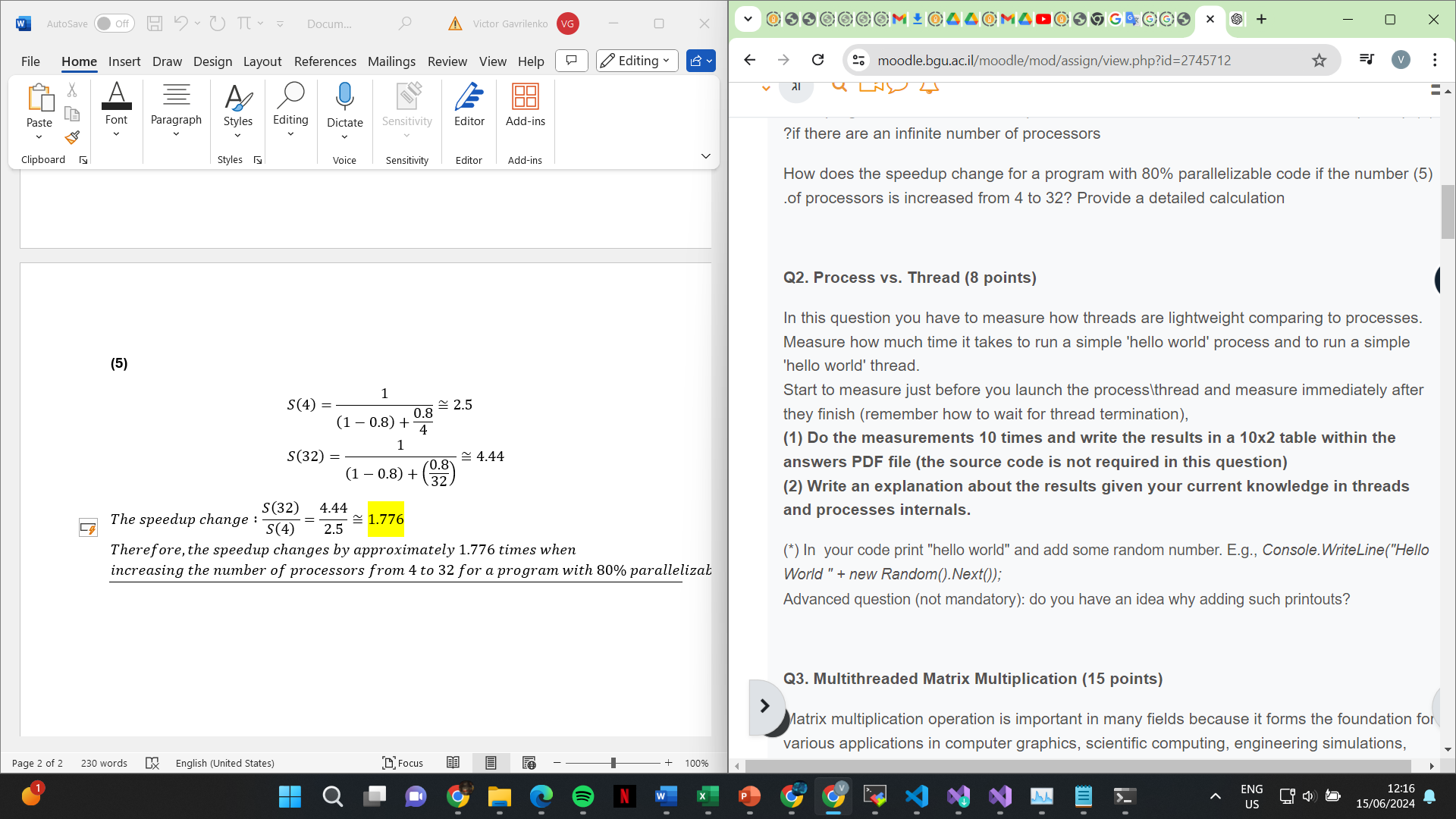
.

**(2)**

**(3)**

**(4)**

**(5)**



(1)

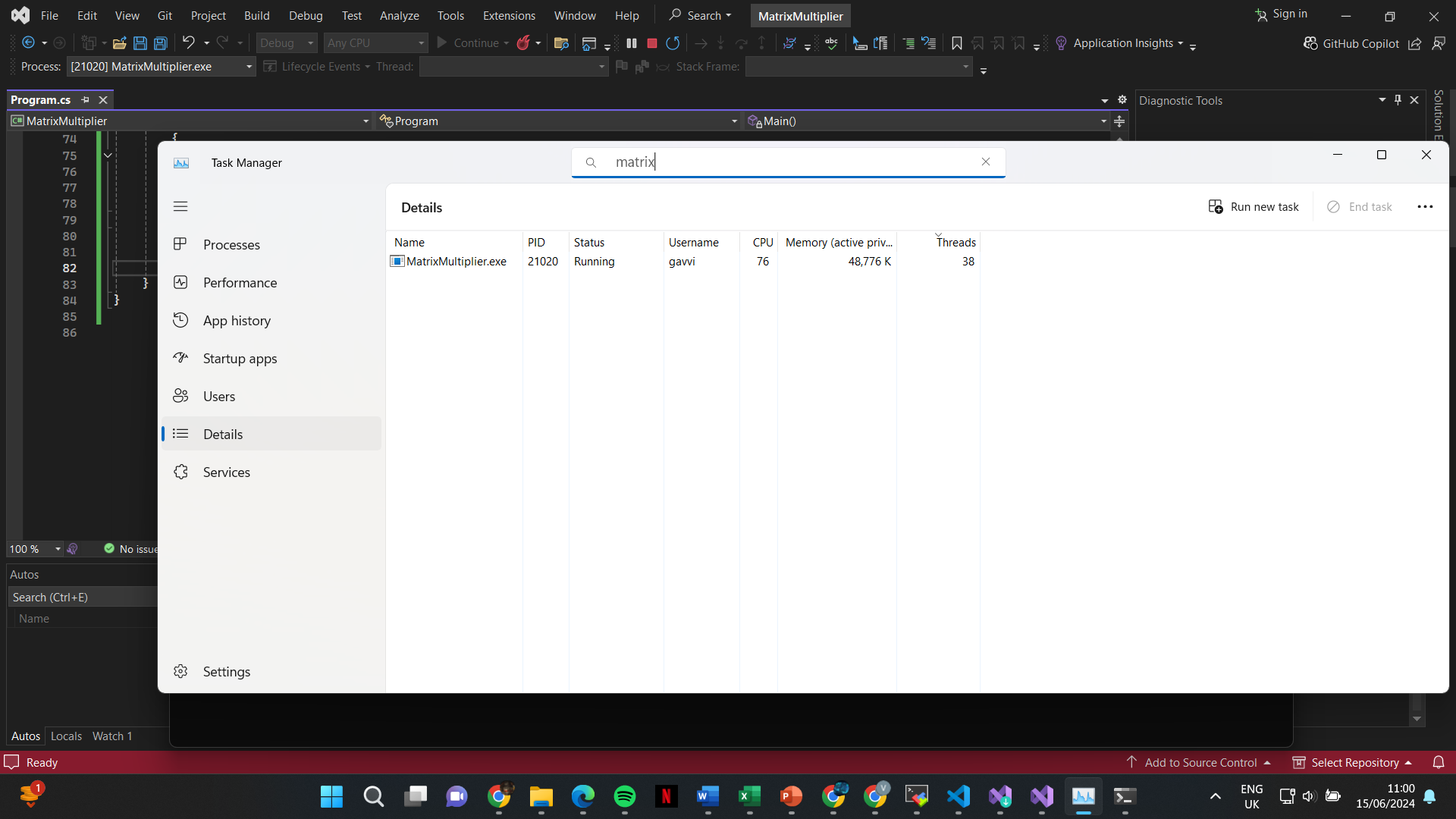
A screenshot of a computer

Description automatically generated

(2) Processes, which run independently with their own memory space and resources allocated by the OS, exhibited longer execution times. This is due to significant overhead including memory allocation, context switching, and potentially disk I/O operations. In contrast, threads, sharing memory within the same process, started swiftly with shorter execution times. Threads' efficiency stems from minimal setup compared to processes, highlighting their advantage for concurrent tasks within a program

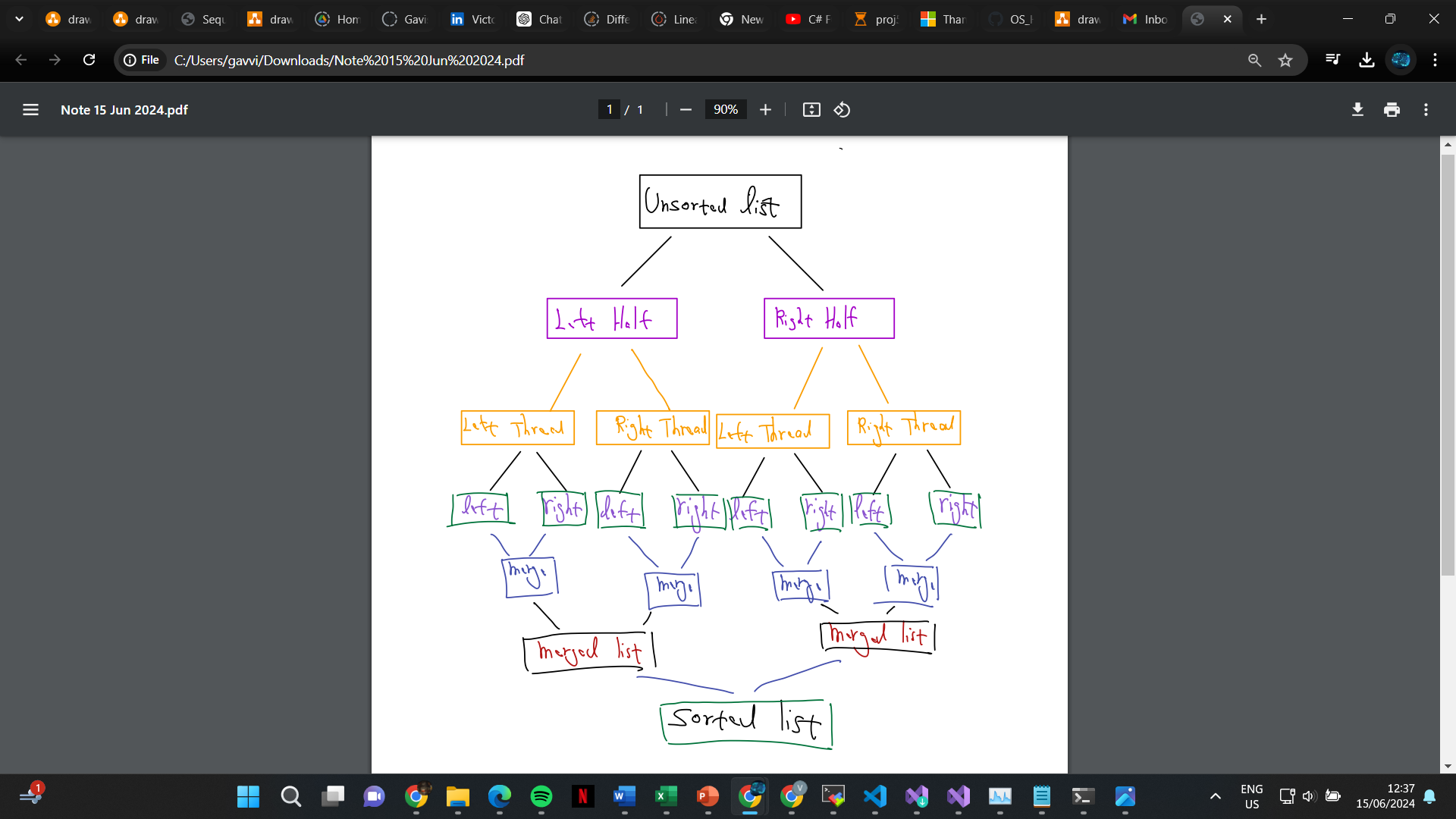
(3) The Console.writeline isn’t thread safe. When we print out the random numbers we can see that the prints from the threads overwrite the prints from the process. From that fact we can in fer that that the threads and process don’t have a synchronized access to the Console, and their prints overwrite each other.

Q3.



Q4:

1. Multi-threaded merge-sort divides a list of strings into halves, sorting each recursively. If the list size is below a threshold (), it sorts sequentially to avoid thread overhead. Larger lists spawn two threads to sort halves concurrently. After sorting, results are merged using This approach balances thread utilization and sorting efficiency, crucial for large datasets where parallel processing improves performance significantly.
2. **Example with**  .(In our test we tried with both low such as 4, and high such as 1000, with higher values we got fast execution times)



1. + d) + e) are in the class submission