Haidar Kahla
200513520

SWE 449

Final Exam

Parallel programming

Q1:

"Parallel programming" refers to a programing technique where tasks are broken down into smaller subtasks that can be executed simultaneously or concurrently on multiple processors or cores a computer. The goal is to improve performance by dividing the workload among multiple processing units.

- This approach reduces overall execution time and improves performance by distributing the workload effectively among processing units, enabling faster completion of complex computations and tasks.

Q2:

A thread in parallel programming is a lightweight unit of execution that runs concurrently with other threads within a single process. Threads share the same memory space, allowing them to efficiently execute tasks simultaneously. while the main execution flow of a program which typically follows a linear sequence of instructions, thread also enable concurrent of tasks.

Haidar Kahla

200515520

Q3:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Threading.Tasks;

public class FileSearcher
{
public async Task<List<string>>
SearchKeywordInFilesAsync(string keyword,
List<string> filePaths)
{
List<Task<List<string>>> searchTasks =
        new List<Task<List<string>>>();
foreach (var filePath in filePaths)
{ searchTasks.Add(Task.Run(() =>
    SearchKeywordInFile(keyword, filepath)));
}
 try
{ await Task.WhenAll(searchTasks);
List<string> results = new List<string>();
foreach (var task in SearchTasks)
{ results.AddRange(await task); }
return results;
}
 { catch (Exception exc)
 { Console.WriteLine($"Error: {ex.Message}");
   throw; } }

private List<string> SearchKeywordInFile(
   string keyword, string filePath)
{ List<string> matchingLines = new List<string>();
try
{ using (StreamReader reader = new
        StreamReader(filePath))
{ string line;
while (line = reader.ReadLine())!=null)
{
if line.Contains(keyword, StringComparison.
                OrdinalIgnoreCase))
 { matchingLines.Add($"File:
  {Path.GetFileName(filePath)}, Line: {line}"); } }
} } } catch (Exception ex)
 { Console.WriteLine($"Error: {ex.Message}");
 } return matchingLines;
 }
}
```

Haidar Kabla
200713520

SWE 449
Final Exam

Q4 ?

```
using System;
using System.Threading;

class Program
{ static void Main (String [] args)
    int number = 10;

BigInteger result = CalculateFactorial
                    MultiThreaded (number);

Console.WriteLine ($"Factorial of {number} is: {result}");
}

static BigInteger CalculateFactorialMultiThreadal (
                    int number) {

    if (number < 0)
{ throw new ArgumentExceptin ("Number
    must be non negative.");
} else if (number == 0 || number == 1)
  { return 1; }

int ThreadCount = Environment.
                  ProcessorCount;

BigInteger [] partialResults = new
                  BigInteger [ThreadCount];

Thread [] threads = new Thread [ThreadCount];

int itemPerThread = number / ThreadCount;

for (int i = 0; i < ThreadCount; i++)

{ int start = i * itemPerThread + 1;
  int end (i == ThreadCount - 1) ? number :
    start + itemPerThread - 1;

threads[i] = new Thread () =>
{ partialResults [i] =
    CalculatePartialFactorial (start, end);
});
threads [i].start (); }

foreach (Thread thread in threads)
  { Thread.Join (); }

BigInteger finalResult = 1;
foreach (BigInteger partialResult in partialResults)
{ finalResult *= partialResult; }

return finalResult; }
```

Haidar Kabbara

200513120

SWE 449

Final exam

```
static BigInteger calculatePartialFactorial(int start, int end)
{
    BigInteger result = 1;
    for (int i = start; i <= end; i++)
    {
        result *= i;
    }
    return result;
}
```