OPERATING SYSTEMS: THREADING PROGRAMMING ACTIVITY

A Documentary Research Paper Presented to
Computer Studies Department of College of Science
TECHNOLOGICAL UNIVERSITY OF THE PHILIPPINES

In Partial Fulfillment of the
Requirements for
OPERATING SYSTEMS

BSCS 2C-M
Manansala, Rhea Miguela M.
Morales, Carlos Miguel T.
Torres, Paul Adrian O.

PROF. PEGARINO AMADOR, Jr.

JUNE 2023

I. RATIONALE

Threading is a programming concept that allows multiple threads of execution to run concurrently within a single process. In simpler terms, it involves creating and managing multiple independent flows of control, known as threads, within a program. Threads can be used for various purposes, such as handling user input while performing background tasks, implementing parallel algorithms, improving responsiveness in graphical user interfaces, and managing concurrent access to shared resources.

1. **Fibonacci** – Calculating Fibonacci numbers involves recursive computations, where each number depends on the two preceding numbers. By using threading, you can divide the Fibonacci sequence into smaller chunks and assign them to different threads. Each thread can independently calculate a portion of the sequence, allowing parallel execution and speeding up the overall computation. This can be particularly beneficial when dealing with larger Fibonacci numbers, as the computation time can be significantly reduced.
2. **Square** – When you need to calculate the square of a large set of numbers, threading can be utilized to distribute the workload among multiple threads. Each thread can take a subset of numbers and compute their squares concurrently. By parallelizing the calculations, you can leverage the available processing power of modern multi-core CPUs and perform the square calculations more quickly. This can be especially useful when dealing with many elements or when performing square computations repeatedly.
3. **Sort Numbers** – Depending on the sorting algorithm used, threading can be employed to divide the sorting process into multiple independent steps. For example, in a parallel merge sort, the initial array can be divided into smaller sub-arrays that are sorted concurrently by different threads. Then, the sorted sub-arrays can be merged in parallel to produce the final sorted array. This approach allows for efficient parallelization of the sorting process, which can significantly speed up the sorting of large datasets.

II. ALGORITHM, CODE AND RUN PROGRAM

CODE FOR MAIN

```
import java.util.Random;

public class Main {
    public static void main(String[] args) {
        Fibonacci fibonacci = new Fibonacci( pid: 1001);
        Square square = new Square( pid: 1002);
        Sortnumbers sortNumbers = new Sortnumbers( pid: 1003);
        try {
            fibonacci.start();
            fibonacci.join();
            square.start();
            square.join();
            sortNumbers.start();
            sortNumbers.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("All processes complete.");
    }
}
```

CODE FOR FIBONACCI

```
2 usages
class Fibonacci extends Thread {
    2 usages
    private int num;
    2 usages
    private int pid;

    1 usage
    public Fibonacci(int pid) {
        Random rand = new Random();
        this.num = rand.nextInt( bound: 20);
        this.pid = 24215;
    }

    public void run() {
        System.out.println("------------------------------------------------------------");
        System.out.println("Fibonacci Number Series (Process ID: " + pid + "):");
        int firstnum = 0;
        int secondnum = 1;
        System.out.print(firstnum + ", ");
        System.out.print(secondnum + ", ");
        for (int i = 2; i < num; i++){
            int next = firstnum + secondnum;
            System.out.print(next + ", ");
            firstnum = secondnum;
            secondnum = next;
        }
        System.out.println("\n------------------------------------------------------------");
    }
}
```

CODE FOR SQUARE

```
2 usages
class Square extends Thread {
    6 usages
    private int n;
    2 usages
    private int pid;

    1 usage
    public Square(int pid) {
        Random rand = new Random();
        this.n = rand.nextInt( bound: 25);
        this.pid = 24216;
    }

    public void run() {
        System.out.println("Square of Numbers (Process ID: " + pid + "):");
        int num = n;
        int square = n * n;
        System.out.print(n + " x "+ n + " = "+ square + " ");
        System.out.println("\n----------------------------------------");
    }
}
```

CODE FOR SORT NUMBERS

```
1 usage
public Sortnumbers(int pid) {
    Random rand = new Random();
    n = 10;
    num = new int[n];
    for (int i = 0; i < num.length; i++) {
        num[i] = rand.nextInt( bound: 100);
    }
    this.pid = 24217;
}

public void run() {
    System.out.println("Sorted Numbers (Process ID: " + pid + "):");
    for (int i = 0; i < num.length; i++) {
        for (int j = i + 1; j < num.length; j++) {
            if (num[j] < num[i]) {
                int temp = num[j];
                num[j] = num[i];
                num[i] = temp;
            }
        }
    }
    for (int number : num) {
        System.out.print(number + ", ");
    }
    System.out.println("\n----------------------------------------");
}
}
```

RUN PROGRAM

```
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:D:\IDE Apps\IntelliJ IDEA
------------------------------------------------------------
Fibonacci Number Series (Process ID: 24215):
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584,
------------------------------------------------------------
Square of Numbers (Process ID: 24216):
4 x 4 = 16
----------------------------------------
Sorted Numbers (Process ID: 24217):
2, 21, 34, 50, 56, 60, 61, 81, 87, 87,
----------------------------------------
All processes complete.
```