Report on Advanced Programming Assignment: Multithreading

Sepehr Rezaee

June 7, 2024

1 Introduction

This report details the implementation and reasoning behind the solutions to the Advanced Programming assignment focusing on multi-threading. The primary objectives were to solve three distinct problems using multi-threaded programming techniques and understand the use of synchronization tools like semaphores.

2 Solutions Implementation

2.1 Calculate Pi

The Calculate Pi problem required calculating the value of Pi up to 1000 decimal places. Various algorithms were evaluated for this purpose, with a focus on those well-suited for parallel execution.

2.1.1 Algorithms Considered

- The Leibniz Formula for Pi, which is straightforward but converges slowly.
- The Bailey-Borwein-Plouffe (BBP) formula, which allows the n-th digit of Pi to be calculated without needing the preceding digits.
- The Chudnovsky algorithm, which is extremely fast in terms of convergence and is commonly used in record-setting calculations of Pi.

2.1.2 Final Choice and Implementation

The Chudnovsky algorithm was chosen for the final implementation due to its rapid convergence. The implementation involved using Java's `BigDecimal` for precision handling and a multithreaded approach to split the computation of the series into manageable tasks processed in parallel. Here is a snippet of the implementation:

```
// Pi computation logic using Chudnovsky algorithm
public BigDecimal computePi(int precision) {
    BigDecimal result = new BigDecimal(0);
    // computation logic here
    return result.setScale(precision, RoundingMode.HALF_UP);
}
```

2.2 Semaphore Usage

The semaphore exercise involved managing access to a shared resource by multiple threads. A semaphore with a permit of 2 was used to ensure that no more than two threads could access the critical section at any given time.

2.2.1 Implementation Details

The semaphore was incorporated directly into the shared resource access method, with debug statements included to trace the access pattern. The implementation ensured thread safety and demonstrated the proper use of semaphores in controlling access to critical sections.

```
Semaphore semaphore = new Semaphore(2);

public void accessResource() {
    semaphore.acquire();
    try {
        // Access the resource
    } finally {
        semaphore.release();
    }
}
```

3 Discussion on the Calculate Pi Problem

The Chudnovsky algorithm's key advantage is its extremely fast convergence, which significantly reduces the computational effort required to calculate many digits of Pi. By implementing this algorithm in a multithreaded fashion, the computation can be further expedited by leveraging multiple cores of modern processors.

4 Semaphore Explanation and Use Cases

Semaphores are synchronization tools used to control access to shared resources. A semaphore manages a set of permits, which threads must acquire before accessing the resource and release after use. Semaphores are particularly useful

in scenarios where resources can accommodate limited concurrent access, as demonstrated in this assignment.

5 Conclusion

This report has detailed the approaches and implementations used to solve the given problems in the assignment. Through this exercise, practical understanding of multithreading, synchronization mechanisms, and the mathematical computation of Pi has been enhanced.

6 References and Resources

- Official Java Documentation: `https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Semaphore.html`
- Pi Calculation and its Algorithms: `https://www.piday.org/million/`