Misato Seki

CPU Scheduling Algorithms

Multithreading and Performance Analysis

Report
CENTRIA UNIVERSITY OF APPLIED SCIENCES
Bachelor of Engineering, Information Technology
April 2023

ABSTRACT

Centria University of Applied Sciences	Date April 2023	Author Misato Seki
Degree programme Bachelor of Engineering, Information Technology		
Name of thesis CPU Scheduling Algorithms - Multithreading and Performance Analysis		
Centria supervisor		**Pages** 5 + 6
Instructor representing commissioning institution or company		

In this report, we will implement CPU scheduling algorithms using multi-threading. we will also compare their performance, such as CPU utilization, throughput, turnaround time, latency, and response time, and analyse the characteristics of each scheduling algorithm.

Through implementing various CPU scheduling algorithms in a multi-threaded program, we felt that there are some suitable algorithms and some unsuitable algorithms. We will explain the reason later, but the FCFS algorithm is not suitable for this program. When implementing a CPU scheduling algorithm, it is necessary to confirm that the algorithm is indeed suitable or not.

When comparing the performance of the programs, the single-threaded program performed slightly better than the multi-threaded program with the FCFS algorithm. This may be because, in implementing the FCFS algorithm in the multi-threaded program, we had to change the threads to be processed one at a time. In other words, the next thread could not be started until the first thread was finished, and thus could not take advantage of the multi-threaded feature.
The reason why the Priority Scheduling algorithm performed better when compared to the SJN algorithm is that the SJN algorithm determines priorities after estimating and calculating the processing time, whereas the Priority Scheduling algorithm is based on the programmer’s order. Scheduling algorithm was less demanding because the priorities were specified on the programmer's side.

Key words

ABSTRACT

CONTENTS

1 INTRODUCTION

In this report, we will implement CPU scheduling algorithms using multi-threading. we will also compare their performance, such as CPU utilization, throughput, turnaround time, latency, and response time, and analyse the characteristics of each scheduling algorithm.

2 HOW TO INCORPORATE

2.1 First-Come, First-Served (FCFS)

The FCFS (First-Come, First-Served) algorithm is implemented by using threads[i].join() to wait for the completion of the previous thread before starting the execution of each thread. This ensures that each thread is executed in turn. (PICTURE.1)

2.2 Shortest Job Next (SJN)

The Task class is implemented to estimate the time required to compute a task. The Task class contains three fields: start, end, and estimatedProcessingTime. estimatedProcessingTime estimates the time to process the task and is used to sort the Task object. The program sorts tasks based on the value of estimatedProcessingTime, processing the tasks in order of shortest to longest time. (PICTURE.2)

2.3 Priority Scheduling

To implement the Priority Scheduling algorithm, we need to set a priority for each thread and determine the order in which the threads are executed. To set the priority of each thread, THREAD_COUNT - i is used. This ensures that the first thread has the highest priority, and the last thread has the lowest priority. The priority of each thread is also set using the setPriority method. (PICTURE.3)

2.4 Round Robin (RR)

To implement the round robin scheduling algorithm, this code pauses a thread and moves on to the next thread when its execution time exceeds a certain amount of time. This ensures that each thread executes fairly and that tasks are distributed in a balanced manner.
A function is also implemented to calculate the schedule threshold for each subtask. This function simply returns the intermediate value of the subtasks to ensure that the tasks are evenly divided. (PICTURE.4)

2.5 Multilevel Queue Scheduling

Two priority queues (lowPriorityQueue and highPriorityQueue) are used to schedule tasks with different priorities. BlockingQueue is used to implement the queue. The put method is used to add tasks to the queue, and the take method is used to prioritize and take tasks with higher priority. We also use Callable to define tasks and store the total value calculated from each thread in a Future object. Finally, when all tasks are completed, the ExecutorService is shut down and the totals are output. (PICTURE.5)

3 COMPARISON WITH SINGLE THREADED PROGRAM

When comparing programs implementing the FCFS algorithm, the single-threaded program performed slightly better on all comparison items. In terms of the programs implementing the SJN algorithm and the Priority Scheduling algorithm, we found that the multi-threaded program performed better in all comparison items.
When comparing the SJN algorithm and the Priority Scheduling algorithm, the Priority Scheduling algorithm performed slightly better, regardless of whether the program was multi-threaded or single-threaded. (PICTURE.6, 7, 8)

FCFS	Multi-threaded	Single-threaded
CPU utilization	651.858791 ms	639.87075 ms
Throughput	1.5340745784312059 tasks/sec	1.56281561549734839tasks/sec
Turnaround time	162.96469775 ms	159.9676875 ms
Waiting time	651.858791 ms	639.87075 ms
Response time	651.858791 ms	639.87075 ms

SJN	Multi-threaded	Single-threaded
CPU utilization	189.742416 ms	648.571958 ms
Throughput	5.2703028720789569 tasks/sec	1.5418489616536899 tasks/sec
Turnaround time	47.435604 ms	162.1429895 ms
Waiting time	189.742416 ms	648.571958 ms
Response time	189.742416 ms	648.571958 ms

PS	Multi-threaded	Single-threaded
CPU utilization	187.365459 ms	637.267125 ms
Throughput	5.337163025336499 tasks/sec	1.5692006707548266 tasks/sec
Turnaround time	46.84136475 ms	159.31678125 ms
Waiting time	187.365459 ms	637.267125 ms
Response time	187.365459 ms	637.267125 ms

4 CONCLUSION

Through implementing various CPU scheduling algorithms in a multi-threaded program, we felt that there are some suitable algorithms and some unsuitable algorithms. We will explain the reason later, but the FCFS algorithm is not suitable for this program. When implementing a CPU scheduling algorithm, it is necessary to confirm that the algorithm is indeed suitable or not.

When comparing the performance of the programs, the single-threaded program performed slightly better than the multi-threaded program with the FCFS algorithm. This may be because, in implementing the FCFS algorithm in the multi-threaded program, we had to change the threads to be processed one at a time. In other words, the next thread could not be started until the first thread was finished, and thus could not take advantage of the multi-threaded feature.
The reason why the Priority Scheduling algorithm performed better when compared to the SJN algorithm is that the SJN algorithm determines priorities after estimating and calculating the processing time, whereas the Priority Scheduling algorithm is based on the programmer’s order. Scheduling algorithm was less demanding because the priorities were specified on the programmer's side.

REFERENCES

chatGPT
https://chat.openai.com/chat

APPENDIX

```java
public class MultiThreadingSumFCFS {
    private static final int THREAD_COUNT = 4;
    private static final int TASK_SIZE = 250000000;
    private static final int N = THREAD_COUNT * TASK_SIZE;
    private static long sum = 0;

    public static void main(String[] args) throws InterruptedException {
        Thread[] threads = new Thread[THREAD_COUNT];
        for (int i = 0; i < THREAD_COUNT; i++) {
            final int start = i * TASK_SIZE + 1;
            final int end = (i + 1) * TASK_SIZE;
            threads[i] = new Thread(() -> {
                long threadSum = compute(start, end);
                synchronized (MultiThreadingSumFCFS.class) {
                    sum += threadSum;
                }
                System.out.println("Thread " + Thread.currentThread().getId() + ": Computed sum from " + start + " to " + end + " = " + threadSum);
            });
            threads[i].start();
            threads[i].join(); //FCFS algorithm: wait for completion of each thread
        }

        System.out.println("Parent: Sum of numbers from 1 to " + N + " = " + sum);
    }

    private static long compute(int start, int end) {
        long threadSum = 0;
        for (int i = start; i <= end; i++) {
            threadSum += i;
        }
        return threadSum;
    }
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
misatoseki@MisatonoMacBook-Air Session14 % java MultiThreadingSumFCFS
Thread 14: Computed sum from 1 to 250000000 = 31250000125000000
Thread 15: Computed sum from 250000001 to 500000000 = 93750000125000000
Thread 16: Computed sum from 500000001 to 750000000 = 156250000125000000
Thread 17: Computed sum from 750000001 to 1000000000 = 218750000125000000
Parent: Sum of numbers from 1 to 1000000000 = 500000000500000000
```

PICTURE.1 code and result (FCFS)

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

public class MultiThreadingSumSJN {
    private static final int THREAD_COUNT = 4;
    private static final int TASK_SIZE = 250000000;
    private static final int N = THREAD_COUNT * TASK_SIZE;
    private static long sum = 0;

    public static void main(String[] args) throws InterruptedException {
        List<Task> tasks = new ArrayList<>();
        for (int i = 0; i < THREAD_COUNT; i++) {
            final int start = i * TASK_SIZE + 1;
            final int end = (i + 1) * TASK_SIZE;
            Task task = new Task(start, end);
            tasks.add(task);
        }

        // sort tasks by estimated processing time
        Collections.sort(tasks, Comparator.comparing(Task::getEstimatedProcessingTime));

        Thread[] threads = new Thread[THREAD_COUNT];
        for (int i = 0; i < THREAD_COUNT; i++) {
            Task task = tasks.get(i);
            threads[i] = new Thread(() -> {
                long threadSum = compute(task.getStart(), task.getEnd());
                synchronized (MultiThreadingSumSJN.class) {
                    sum += threadSum;
                }
                System.out.println("Thread " + Thread.currentThread().getId() + ": Computed sum from " + task.getStart() + " to " + task.getEnd() + " = " + threadSu
            });
            threads[i].start();
        }

        for (Thread thread : threads) {
            thread.join();
        }

        System.out.println("Parent: Sum of numbers from 1 to " + N + " = " + sum);
    }

    private static long compute(int start, int end) {
        long threadSum = 0;
        for (int i = start; i <= end; i++) {
            threadSum += i;
        }
        return threadSum;
    }
     private static class Task {
         private int start;
         private int end;
         private int estimatedProcessingTime;

         public Task(int start, int end) {
             this.start = start;
             this.end = end;
             this.estimatedProcessingTime = (end - start) / TASK_SIZE;
         }

         public int getStart() {
             return start;
         }

         public int getEnd() {
             return end;
         }

         public int getEstimatedProcessingTime() {
             return estimatedProcessingTime;
         }
     }
 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
misatoseki@MisatonoMacBook-Air Session14 % java MultiThreadingSumSJN
Thread 16: Computed sum from 500000001 to 750000000 = 156250000125000000
Thread 17: Computed sum from 750000001 to 1000000000 = 218750000125000000
Thread 14: Computed sum from 1 to 250000000 = 31250000125000000
Thread 15: Computed sum from 250000001 to 500000000 = 93750000125000000
Parent: Sum of numbers from 1 to 1000000000 = 500000000500000000
```

PICTURE.2 code and result (SJN)

```java
public class MultiThreadingSumPS {
    private static final int THREAD_COUNT = 4;
    private static final int TASK_SIZE = 250000000;
    private static final int N = THREAD_COUNT * TASK_SIZE;
    private static long sum = 0;

    public static void main(String[] args) throws InterruptedException {
        Thread[] threads = new Thread[THREAD_COUNT];
        for (int i = 0; i < THREAD_COUNT; i++) {
            final int start = i * TASK_SIZE + 1;
            final int end = (i + 1) * TASK_SIZE;
            final int priority = THREAD_COUNT - i; // 優先度を設定
            threads[i] = new Thread(() -> {
                long threadSum = compute(start, end);
                synchronized (MultiThreadingSum.class) {
                    sum += threadSum;
                }
                System.out.println("Thread " + Thread.currentThread().getId() + ": Computed sum from " + start + " to " + end + " = " + threadSum);
            });
            threads[i].setPriority(priority); // 優先度をセット
            threads[i].start();
        }

        for (Thread thread : threads) {
            thread.join();
        }

        System.out.println("Parent: Sum of numbers from 1 to " + N + " = " + sum);
    }

    private static long compute(int start, int end) {
        long threadSum = 0;
        for (int i = start; i <= end; i++) {
            threadSum += i;
        }
        return threadSum;
    }
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
misatoseki@MisatonoMacBook-Air Session14 % java MultiThreadingSumPS
Thread 14: Computed sum from 1 to 250000000 = 31250000125000000
Thread 17: Computed sum from 750000001 to 1000000000 = 218750000125000000
Thread 16: Computed sum from 500000001 to 750000000 = 156250000125000000
Thread 15: Computed sum from 250000001 to 500000000 = 93750000125000000
Parent: Sum of numbers from 1 to 1000000000 = 500000000500000000
```

PICTURE.3 code and result (Priority Scheduling)

MultiThreadingSumRR.java

```java
public class MultiThreadingSumRR {
    private static final int THREAD_COUNT = 4;
    private static final int TASK_SIZE = 250000000;
    private static final int N = THREAD_COUNT * TASK_SIZE;
    private static long sum = 0;

    public static void main(String[] args) throws InterruptedException {
        Thread[] threads = new Thread[THREAD_COUNT];
        for (int i = 0; i < THREAD_COUNT; i++) {
            final int start = i * TASK_SIZE + 1;
            final int end = (i + 1) * TASK_SIZE;
            threads[i] = new Thread(() -> {
                long threadSum = 0;
                int current = start;
                while (current <= end) {
                    int next = current + TASK_SIZE / THREAD_COUNT;
                    if (next > end) {
                        next = end;
                    }
                    long subTaskSum = compute(current, next);
                    synchronized (MultiThreadingSumRR.class) {
                        sum += subTaskSum;
                    }
                    System.out.println("Thread " + Thread.currentThread().getId() + ": Computed sum from " + current + " to " + next + " = " + subTaskSum);
                    current = next + 1;
                    // Round robin scheduling: switch to next thread if thread execution time exceeds 100ms
                    try {
                        Thread.sleep(100);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    Thread.yield();
                }
            });
            threads[i].start();
        }

        for (Thread thread : threads) {
            thread.join();
        }

        System.out.println("Parent: Sum of numbers computed by all threads = " + sum);
    }
    // Compute the sum of numbers from start to end (inclusive)
    private static long compute(int start, int end) {
        long subTaskSum = 0;
        for (int i = start; i <= end; i++) {
            subTaskSum += i;
        }
        return subTaskSum;
    }
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
misatoseki@MisatonoMacBook-Air Session14 % java MultiThreadingSumRR
Thread 16: Computed sum from 500000001 to 562500001 = 33203125593750001
Thread 15: Computed sum from 250000001 to 312500001 = 17578125343750001
Thread 17: Computed sum from 750000001 to 812500001 = 48828125843750001
Thread 14: Computed sum from 1 to 62500001 = 1953125093750001
Thread 16: Computed sum from 562500002 to 625000002 = 37109375718750002
Thread 14: Computed sum from 62500002 to 125000002 = 5859375218750002
Thread 17: Computed sum from 812500002 to 875000002 = 52734375968750002
Thread 15: Computed sum from 312500002 to 375000002 = 21484375468750002
Thread 15: Computed sum from 375000003 to 437500003 = 25390625593750003
Thread 14: Computed sum from 125000003 to 187500003 = 9765625343750003
Thread 17: Computed sum from 875000003 to 937500003 = 56640626093750003
Thread 16: Computed sum from 625000003 to 687500003 = 41015625843750003
Thread 15: Computed sum from 437500004 to 500000000 = 29296873718749994
Thread 14: Computed sum from 187500004 to 250000000 = 13671874468749994
Thread 17: Computed sum from 937500004 to 1000000000 = 60546872218749994
Thread 16: Computed sum from 687500004 to 750000000 = 44921872968749994
Parent: Sum of numbers computed by all threads = 500000000500000000
```

PICTURE.4 code and result (Round Robin)

MultiThreadingSumMQS.java

```java
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.*;

public class MultiThreadingSumMQS {
    private static final int THREAD_COUNT = 4;
    private static final int TASK_SIZE = 250000000;
    private static final int N = THREAD_COUNT * TASK_SIZE;
    private static long sum = 0;
    private static final int LOW_PRIORITY = 1;
    private static final int HIGH_PRIORITY = 2;

    private static BlockingQueue<Future<Long>> lowPriorityQueue = new LinkedBlockingQueue<>();
    private static BlockingQueue<Future<Long>> highPriorityQueue = new LinkedBlockingQueue<>();

    public static void main(String[] args) throws InterruptedException, ExecutionException {
        List<Future<Long>> futures = new ArrayList<>();
        ExecutorService executor = Executors.newFixedThreadPool(THREAD_COUNT);

        for (int i = 0; i < THREAD_COUNT; i++) {
            final int start = i * TASK_SIZE + 1;
            final int end = (i + 1) * TASK_SIZE;
            final int priority = i % 2 == 0 ? LOW_PRIORITY : HIGH_PRIORITY;
            Callable<Long> task = () -> compute(start, end);

            if (priority == LOW_PRIORITY) {
                lowPriorityQueue.put(executor.submit(task));
            } else {
                highPriorityQueue.put(executor.submit(task));
            }
        }

        while (!lowPriorityQueue.isEmpty() || !highPriorityQueue.isEmpty()) {
            if (!highPriorityQueue.isEmpty()) {
                Future<Long> future = highPriorityQueue.take();
                sum += future.get();
            } else if (!lowPriorityQueue.isEmpty()) {
                Future<Long> future = lowPriorityQueue.take();
                sum += future.get();
            }
        }

        executor.shutdown();
        System.out.println("Parent: Sum of numbers from 1 to " + N + " = " + sum);
    }
    private static long compute(int start, int end) {
        long threadSum = 0;
        for (int i = start; i <= end; i++) {
            threadSum += i;
        }
        System.out.println("Thread " + Thread.currentThread().getId() + ": Computed sum from " + start + " to " + end + " = " + threadSum);
        return threadSum;
    }
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
misatoseki@MisatonoMacBook-Air Session14 % java MultiThreadingSumMQS
Thread 17: Computed sum from 750000001 to 1000000000 = 218750000125000000
Thread 16: Computed sum from 500000001 to 750000000 = 156250000125000000
Thread 14: Computed sum from 1 to 250000000 = 31250000125000000
Thread 15: Computed sum from 250000001 to 500000000 = 93750000125000000
Parent: Sum of numbers from 1 to 1000000000 = 500000000500000000
```

PICTURE.5 code and result (Multilevel Queue Scheduling)

```
misatoseki@MisatonoMacBook-Air Session14 % java MultiThreadingSumFCFScopy
Thread 14: Computed sum from 1 to 250000000 = 31250000125000000
Thread 15: Computed sum from 250000001 to 500000000 = 93750000125000000
Thread 16: Computed sum from 500000001 to 750000000 = 156250000125000000
Thread 17: Computed sum from 750000001 to 1000000000 = 218750000125000000
Parent: Sum of numbers from 1 to 1000000000 = 500000000500000000
CPU Usage: 651.858791 ms
Throughput: 1.534074578431205E9 tasks/second
Turnaround Time: 162.96469775 ms
Waiting Time: 651.858791 ms
Response Time: 651.858791 ms
misatoseki@MisatonoMacBook-Air Session14 % java MultiThreadingSumFCFSSingle
Computed sum from 1 to 250000000
Computed sum from 250000001 to 500000000
Computed sum from 500000001 to 750000000
Computed sum from 750000001 to 1000000000
Parent: Sum of numbers from 1 to 1000000000 = 500000000500000000
CPU Usage: 639.87075 ms
Throughput: 1.5628156154973483E9 tasks/second
Turnaround Time: 159.9676875 ms
Waiting Time: 639.87075 ms
Response Time: 639.87075 ms
```

PICTURE.6 comparison result (FCFS)

```
misatoseki@MisatonoMacBook-Air Session14 % java MultiThreadingSumSJNcopy
Thread 17: Computed sum from 750000001 to 1000000000 = 218750000125000000
Thread 14: Computed sum from 1 to 250000000 = 31250000125000000
Thread 15: Computed sum from 250000001 to 500000000 = 93750000125000000
Thread 16: Computed sum from 500000001 to 750000000 = 156250000125000000
Parent: Sum of numbers from 1 to 1000000000 = 500000000500000000
CPU Usage: 189.742416 ms
Throughput: 5.270302872078956E9 tasks/second
Turnaround Time: 47.435604 ms
Waiting Time: 189.742416 ms
Response Time: 189.742416 ms
misatoseki@MisatonoMacBook-Air Session14 % java SingleThreadingSumSJN
Computed sum from 1 to 250000000 = 31250000125000000
Computed sum from 250000001 to 500000000 = 93750000125000000
Computed sum from 500000001 to 750000000 = 156250000125000000
Computed sum from 750000001 to 1000000000 = 218750000125000000
Parent: Sum of numbers from 1 to 1000000000 = 500000000500000000
CPU Usage: 648.571958 ms
Throughput: 1.541848961653689E9 tasks/second
Turnaround Time: 162.1429895 ms
Waiting Time: 648.571958 ms
Response Time: 648.571958 ms
```

PICTURE.7 comparison result (SJN)

```
misatoseki@MisatonoMacBook-Air Session14 % java MultiThreadingSumPScopy
Thread 14: Computed sum from 1 to 250000000 = 31250000125000000
Thread 17: Computed sum from 750000001 to 1000000000 = 218750000125000000
Thread 16: Computed sum from 500000001 to 750000000 = 156250000125000000
Thread 15: Computed sum from 250000001 to 500000000 = 93750000125000000
Parent: Sum of numbers from 1 to 1000000000 = 500000000500000000
CPU Usage: 187.365459 ms
Throughput: 5.33716302533649E9 tasks/second
Turnaround Time: 46.84136475 ms
Waiting Time: 187.365459 ms
Response Time: 187.365459 ms
misatoseki@MisatonoMacBook-Air Session14 % java SingleThreadedSumPS
Parent: Sum of numbers from 1 to 1000000000 = 500000000500000000
CPU Usage: 637.267125 ms
Throughput: 1.5692006707548265E9 tasks/second
Turnaround Time: 159.31678125 ms
Waiting Time: 637.267125 ms
Response Time: 637.267125 ms
```

PICTURE.8 comparison result (Priority Scheduling)