Operating System Course Report - First Half of the Semester

B class

October 8, 2024

Contents

1 Introduction

This report summarizes the topics covered during the first half of the Operating System course. It includes theoretical concepts, practical implementations, and assignments. The course focuses on the fundamentals of operating systems, including system architecture, process management, CPU scheduling, and deadlock handling.

2 Course Overview

2.1 Objectives

The main objectives of this course are:

- To understand the basic components and architecture of a computer system.
- To learn process management, scheduling, and inter-process communication.
- To explore file systems, input/output management, and virtualization.
- To study the prevention and handling of deadlocks in operating systems.

2.2 Course Structure

The course is divided into two halves. This report focuses on the first half, which covers:

- Basic Concepts and Components of Computer Systems
- System Performance and Metrics
- System Architecture of Computer Systems
- Process Description and Control
- Scheduling Algorithms
- Process Creation and Termination

- Introduction to Threads
- File Systems
- Input and Output Management
- Deadlock Introduction and Prevention
- User Interface Management
- Virtualization in Operating Systems

3 Topics Covered

3.1 Basic Concepts and Components of Computer Systems

This section explains the fundamental components that make up a computer system, including the CPU, memory, storage, and input/output devices.

3.2 System Performance and Metrics

3.2.1 *Performance Metrics* Pada CPU

Kalau kita membahas kinerja matriks pada komputer, maka kita tidak lepas dari CPU. CPU atau *Central Processing Unit* adalah komponen utama dari komputer yang bertanggung jawab untuk mengeksekusi instruksi-instruksi yang diberikan kepada komputer. Kinerja sebuah sistem sangat dipengaruhi oleh bagaimana CPU mengeksekusi instruksi. Jika kita ingin memaksimalkan kinerja sebuah sistem, kita tentu perlu meminimalkan waktu eksekusi karena kinerja berbanding terbalik dengan waktu eksekusi.

Untuk menentukan waktu eksekusi CPU untuk sebuah program, Anda dapat mencari tahu jumlah total siklus clock yang dibutuhkan program dan mengalikannya dengan waktu siklus clock. Setiap program terdiri dari sejumlah instruksi dan setiap instruksi membutuhkan sejumlah siklus clock untuk dieksekusi. Jika Anda mengetahui jumlah total siklus clock per program dan mengetahui waktu siklus clock untuk setiap siklus clock ini, maka waktu eksekusi CPU dapat dihitung sebagai hasil perkalian jumlah total siklus clock

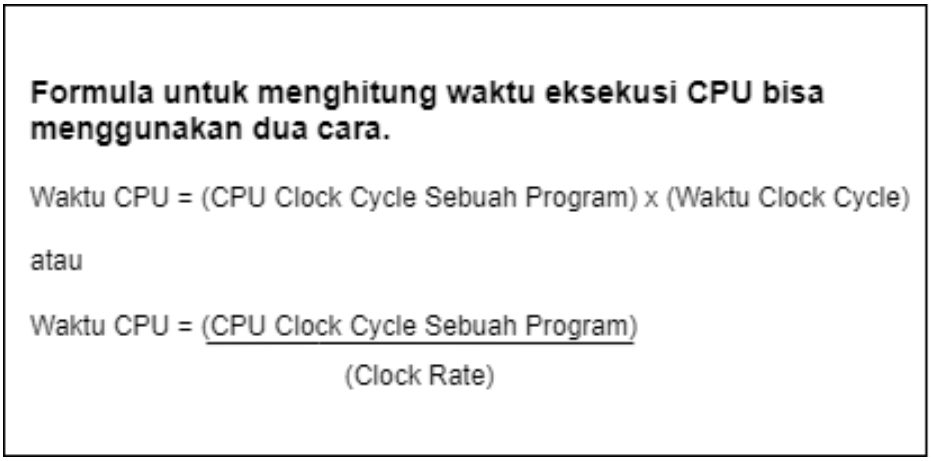

Figure 1: Rumus waktu CPU

CPU per program dengan waktu siklus clock. Karena waktu siklus clock dan clock rate saling terkait, hal ini juga dapat ditulis sebagai jumlah siklus clock CPU untuk sebuah program dibagi dengan clock rate.

Karena waktu eksekusi CPU merupakan hasil dari kedua faktor ini, kita dapat meningkatkan kinerja dengan mengurangi durasi waktu siklus clock atau jumlah siklus clock yang diperlukan untuk suatu program. Kecepatan clock pada dasarnya bergantung pada organisasi CPU tertentu.

3.2.2 Mengapa *Performance Metrics* Penting

Mengukur kinerja menggunakan data yang akurat sangat penting dalam sebuah sistem karena memberikan gambaran nyata tentang performa yang sedang berlangsung. Dengan data yang tepat, manajemen dapat mengambil keputusan yang lebih terinformasi, sehingga memperkecil risiko kesalahan strategi. Selain itu, pengukuran kinerja juga membantu mengidentifikasi area yang membutuhkan perbaikan, sehingga organisasi dapat berfokus pada peningkatan yang signifikan. Memantau perkembangan dan kemajuan kinerja secara berkala meningkatkan akuntabilitas, baik di tingkat individu maupun tim, sehingga setiap anggota lebih bertanggung jawab atas hasil kerjanya. Pada akhirnya, penggunaan *performance metrics* yang tepat dapat mendorong peningkatan berkelanjutan, dengan memberikan dasar yang kuat bagi evaluasi dan inovasi secara sistematis dalam mencapai tujuan jangka

panjang.

3.2.3 Fungsi *Performance Metrics* pada sistem

Metrics performance berperan penting dalam berbagai aspek dalam sistem operasi. Berikut adalah beberapa area di mana metrics performance digunakan :

- ***Benchmark***

  Benchmark biasanya menggunakan berbagai metrics performance untuk mengukur dan melaporkan hasil kinerjanya. Misalnya, sebuah benchmark untuk prosesor mungkin mengukur metrics seperti kecepatan clock, jumlah instruksi per detik (IPS), atau waktu komputasi untuk tugas tertentu.

- ***Bandwidth***

  mengukur jumlah data yang dapat ditransmisikan melalui jaringan dalam waktu tertentu. Bandwidth yang terbatas membatasi kemampuan sistem untuk menangani banyak permintaan atau mentransfer data dalam jumlah besar, yang pada akhirnya memperlambat waktu muat atau respon. Pada sistem yang sangat bergantung pada transfer data, seperti aplikasi berbasis web atau layanan cloud, bandwidth adalah faktor penting yang mempengaruhi performa keseluruhan.

- ***Error Rate***

  menghitung persentase permintaan yang gagal diproses atau menghasilkan kesalahan. Metrik ini penting untuk menilai stabilitas dan keandalan sistem. Tingkat kesalahan yang tinggi menunjukkan masalah pada sistem, seperti bug dalam perangkat lunak, kesalahan konfigurasi, atau keterbatasan sumber daya. Error rate yang tinggi dapat mengganggu pengguna dan menurunkan kepercayaan mereka terhadap sistem.

- **Pengaturan Kualitas Layanan (QoS)**

  Dalam lingkungan yang membutuhkan kualitas layanan tertentu, metrics performance digunakan untuk memastikan bahwa aplikasi dan layanan memenuhi SLA (Service Level Agreement). Metrics seperti latensi, throughput, dan waktu respons digunakan untuk mengatur dan memantau QoS.

- **Optimasi dan Tuning**

  Metrics performance digunakan untuk mengidentifikasi peluang optimasi dan tuning. Misalnya, dengan memantau penggunaan CPU dan memori, administrator sistem dapat menyesuaikan parameter kernel, konfigurasi aplikasi, dan pengaturan sistem lainnya untuk meningkatkan kinerja.

Metrics performance berperan penting dalam berbagai aspek pengelolaan dan operasional sistem operasi. Dari monitoring dan manajemen sumber daya hingga optimasi, diagnosa, dan keamanan, metrics ini memberikan wawasan yang diperlukan untuk menjaga sistem tetap berjalan dengan efisien, aman, dan sesuai dengan kebutuhan kinerja yang diharapkan.

3.3 System Architecture of Computer Systems

Describes the architecture of modern computer systems, focusing on the interaction between hardware and the operating system.

3.4 Process Description and Control

Processes are a central concept in operating systems. This section covers:

- Process states and state transitions
- Process control block (PCB)
- Context switching

3.5 Scheduling Algorithms

This section covers:

- First-Come, First-Served (FCFS)
- Shortest Job Next (SJN)
- Round Robin (RR)

It explains how these algorithms are used to allocate CPU time to processes.

3.6 Process Creation and Termination

Details how processes are created and terminated by the operating system, including:

- Process spawning
- Process termination conditions

3.7 Introduction to Threads

This section introduces the concept of threads and their relation to processes, covering:

- Single-threaded vs. multi-threaded processes
- Benefits of multithreading

3.8 File Systems

File systems provide a way for the operating system to store, retrieve, and manage data. This section explains:

- File system structure
- File access methods
- Directory management

3.9 Input and Output Management

Input and output management is key for handling the interaction between the system and external devices. This section includes:

- Device drivers
- I/O scheduling

3.10 Deadlock Introduction and Prevention

Explores the concept of deadlocks and methods for preventing them:

- Deadlock conditions
- Deadlock prevention techniques

3.11 User Interface Management

This section discusses the role of the operating system in managing the user interface. Topics covered include:

- Graphical User Interface (GUI)
- Command-Line Interface (CLI)
- Interaction between the user and the operating system

3.12 Virtualization in Operating Systems

Virtualization allows multiple operating systems to run concurrently on a single physical machine. This section explores:

- Concept of virtualization
- Hypervisors and their types
- Benefits of virtualization in modern computing

4 Assignments and Practical Work

4.1 Assignment 1: Process Scheduling

Implementasikan algoritma scheduling short job first dan round robin, dan bandingkan average waiting time dan average turnaround time kedua algortima tersebut, yang mana yang lebih baik dalam menangani proses.

JAWABAN

Short Job First (SJF) algorithms

```
1    def sjf_non_preemptive(Jumlah_proses):
2        Jumlah_proses.sort(key=lambda x: x[1])
3
4        waktu_tunggu = 0
5        total_waktu_tunggu = 0
6
```

```
        for pid, burst_time in Jumlah_proses:
            total_waktu_tunggu += waktu_tunggu
            waktu_tunggu += burst_time

        n = len(Jumlah_proses)
        avg_waktu_tunggu = total_waktu_tunggu / n

        print(total_waktu_tunggu)
        print(f"\nAverage Waiting Time: {avg_waktu_tunggu}")
```

Round Robin (RR) algorithms

```
    def round_robin(jumlah_proses, quantum):
        n = len(jumlah_proses)
        remaining_time = [bt for _, bt in jumlah_proses]
        waktu_tunggu = [0] * n
        time = 0

        while True:
            done = True
            for i in range(n):
                if remaining_time[i] > 0:
                    done = False
                    if remaining_time[i] > quantum:
                        time += quantum
                        remaining_time[i] -= quantum
                    else:
                        time += remaining_time[i]
                        waktu_tunggu[i] = time -
                            jumlah_proses[i][1]
                        remaining_time[i] = 0
            if done:
                break

        avg_waktu_tunggu = sum(waktu_tunggu) / n
        print(f"\nAverage Waiting Time: {avg_waktu_tunggu:.2f
            }")
```

```
INPUT :
jumlah_proses = [(1, 10), (2, 5), (3, 8)]

OUTPUT
Average Waiting Time : 6.0       (Short Job First)

Average Waiting Time : 10.6      (Round Robin)
```

Figure 2: Output

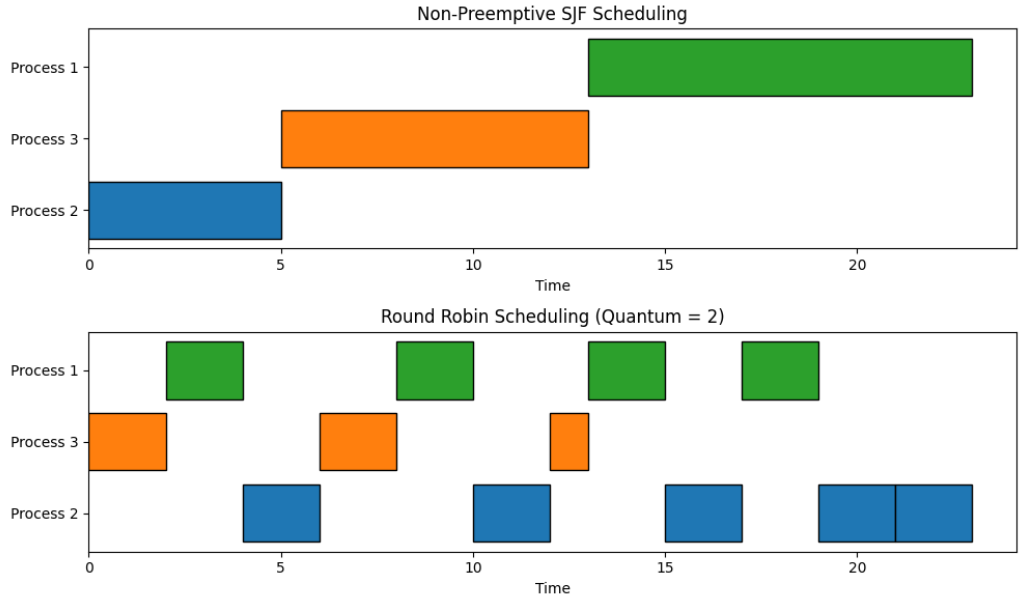

Figure 3: Perbandingan SJF dan RR(quantum = 2)

pada kedua output rata-rata waktu tunggu untuk kedua algoritma, bisa dilihat bahwa rata-rata waktu tunggu sjf lebih ke-

cil dibanding round robin, salah satu alasannya karna sifat RR yang memberikan setiap proses sejumlah waktu yang sama (quantum) dan memaksa semua proses menunggu giliran mereka secara bergilir.

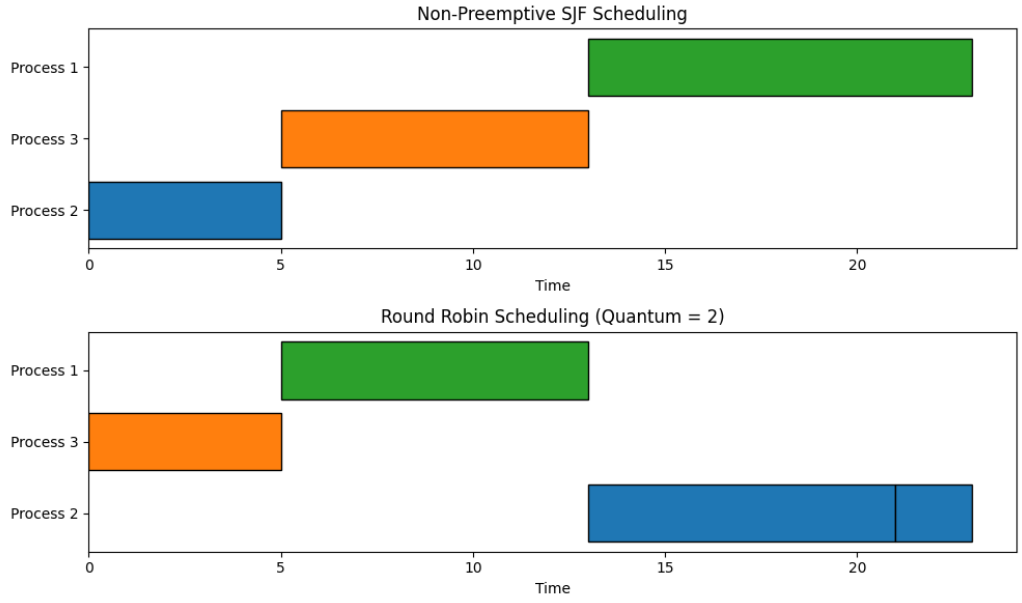

Figure 4: Perbandingan SJF dan RR(quantum = 8)

Kita dapat mengurangi kelemahan Round Robin dengan memilih quantum yang lebih besar, bisa terlihat pada visualisasi diatas bahwa waiting time RR sama dengan SJF. Tetapi tetap saja, SJF sering lebih baik dalam kasus banyak proses dengan burst time yang bervariasi.

4.2 Assignment 2: Deadlock Handling

Simulasikan bagaimana algoritma safety bekerja untuk mendeteksi deadlock.

JAWABAN

```
1
2     import numpy as np
3
4     def is_safe(Jumlah_proses, avail, max_need, allocation):
5         P, R = len(Jumlah_proses), len(avail)
6         need = max_need - allocation
7         finish = [False] * P
8         safe_sequence = []
9         work = avail.copy()
10
11        for _ in range(P):
12            for p in range(P):
13                if not finish[p] and all(need[p] <= work):
14                    work += allocation[p]
15                    finish[p] = True
16                    safe_sequence.append(p)
17                    break
18            else:
19                print("Sistem␣tidak␣dalam␣keadaan␣aman.")
20                return False, []
21
22        print("Sistem␣dalam␣keadaan␣aman.")
23        return True, safe_sequence
24
25
26    Jumlah_proses = [0, 1, 2, 3, 4]
27    avail = np.array([3, 3, 2])
28    max_need = np.array([[7, 5, 3],
29                         [3, 2, 2],
30                         [9, 0, 2],
31                         [2, 2, 2],
32                         [4, 3, 3]])
33    allocation = np.array([[0, 1, 0],
34                           [2, 0, 0],
35                           [3, 0, 2],
36                           [2, 1, 1],
37                           [0, 0, 2]])
38
39    is_safe_state, safe_sequence = is_safe(Jumlah_proses,
          avail, max_need, allocation)
40
41    if is_safe_state:
42        print("Sistem␣dalam␣urutan␣aman:", safe_sequence)
```

```
43      else:
44          print("Tidak ada urutan aman, sistem dalam keadaan
                deadlock.")
```

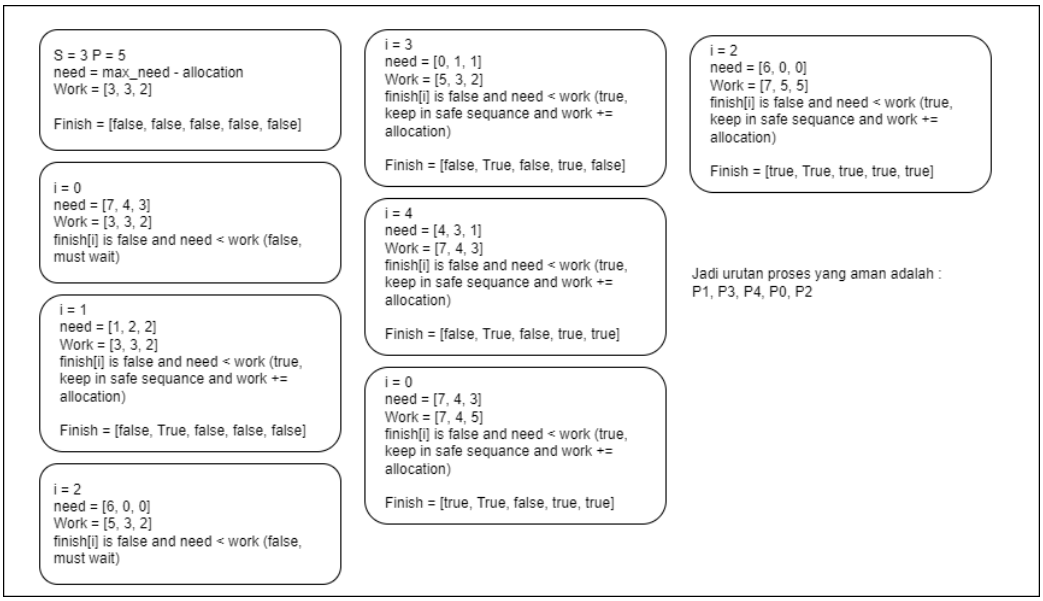

Figure 5: Output dan Penjelasannya

Dengan menggunakan metode banker's untuk algoritma safety kita bisa memastikan bahwa sistem aman jika melakukan proses-proses tertentu, dimana kita memeriksa setiap proses dan memastikan bahwa alokasi sumber daya untuk setiap proses aman dilakukan sehingga tidak akan menghasilkan deadlock. Pada output akan menampilkan status sistem apakah sistem aman atau tidak dan menampilkan urutan proses yang aman.

4.3 Assignment 3: Multithreading and Amdahl's Law

Simulasikan scenario multithreading untuk menghitung jumlah kuadrat dari rentang angka yang diberikan dan menjumlahkannya, dan terapkan Amdahl law untuk menghitung prediksi kecepatan seiring penambahan thread untuk mengetahui penggunaan jumlah thread yang optimal.

JAWABAN

```
1
2       import threading
3       import time
4
5       def compute_squares(start, end, results, index,
            execution_times):
6           start_time = time.time()
7           print(f"Thread {index} mulai menghitung dari {start}
                hingga {end}...")
8           result = 0
9           for number in range(start, end):
10              time.sleep(0.0001)
11              result += number * number
12          results[index] = result
13          end_time = time.time()
14          execution_times[index] = end_time - start_time
15          print(f"Thread {index} selesai menghitung dalam {
                execution_times[index]:.4f} detik.")
16
17      total_range = 10000
18      num_threads = 4
19      step = total_range // num_threads
20
21
22      results = [0] * num_threads
23      execution_times = [0] * num_threads
24
25      threads = []
26
27      start_time = time.time()
28      for i in range(num_threads):
29          start = i * step
30          end = (i + 1) * step if i != num_threads - 1 else
                total_range
31          thread = threading.Thread(target=compute_squares,
                args=(start, end, results, i, execution_times))
32          threads.append(thread)
33          thread.start()
34
35      for thread in threads:
```

```
        thread.join()
    end_time = time.time()

    total_execution_time = end_time - start_time
    total_result = sum(results)

    def amdahl_law(P, N):
        return 1 / ((1 - P) + (P / N))


    sequential_fraction = 0.1
    P = 1 - sequential_fraction

    print("Semua thread selesai.")
    print(f"Hasil perhitungan: {total_result}")
    print(f"Waktu eksekusi total: {total_execution_time:.4f} detik")

    print("Speedup teoretis menurut Hukum Amdahl:")
    for N in range(1, num_threads + 1):
        speedup = amdahl_law(P, N)
        print(f"Speedup teoretis dengan {N} thread: {speedup:.2f}x")
```

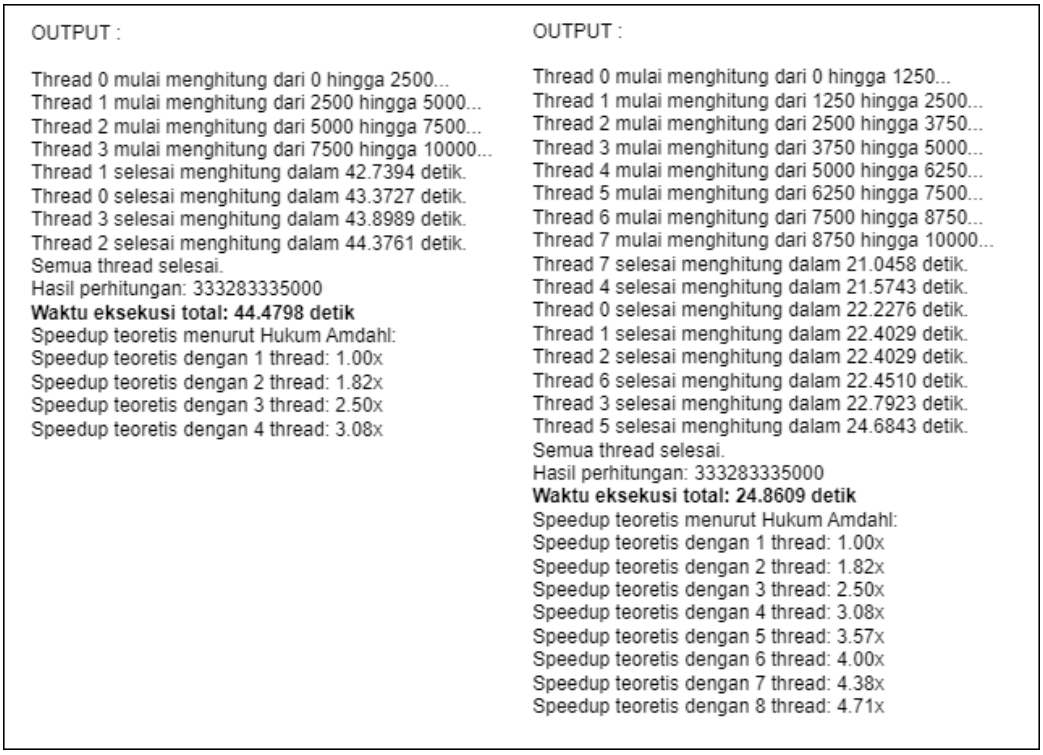

Figure 6: Eksekusi dengan 4 threads (kiri) dan eksekusi dengan 8 threads (kanan)

pada algoritma diatas multithreading berfungsi untuk membagi proses berdasarkan jumlah thread yang digunakan sehingga kita bisa membuat waktu eksekusi lebih singkat, dengan menggunakan algoritma amdah's law kita bisa melihat rasio peningkatan kecepatan seiring meningkatnya jumlah thread yang digunakan. Bisa dilihat pada perbedaan output diatas, dengan meningkatkan jumlah thread dari 4 ke 8 waktu eksekusi juga meningkatkan.

4.4 Assignment 4: Simple Command-Line Interface (CLI) for User Interface Management

Students were tasked with creating a simple **CLI** for user interface management. The CLI should support basic commands such as file manipulation

(creating, listing, and deleting files), process management, and system status reporting.

4.5 Assignment 5: File System Access

In this assignment, students implemented file system access routines, including:

- File creation and deletion
- Reading from and writing to files
- Navigating directories and managing file permissions

5 Conclusion

The first half of the course introduced core operating system concepts, including process management, scheduling, multithreading, and file system access. These topics provided a foundation for more advanced topics to be covered in the second half of the course.

References

[1] GeeksforGeeks. (n.d.). Computer organization: Performance of computer. GeeksforGeeks. Retrieved October 1, 2024, from https://www.geeksforgeeks.org/computer-organization-performance-of-computer/