

Operating System Course Report - First Half of the Semester

B class

October 8, 2024

Contents

1	Introduction	3
2	Course Overview	3
2.1	Objectives	3
2.2	Course Structure	3
3	Topics Covered	4
3.1	Basic Concepts and Components of Computer Systems	4
3.2	System Performance and Metrics	4
3.2.1	<i>Performance Metrics</i> Pada <i>CPU</i>	4
3.2.2	Mengapa <i>Performance Metrics</i> Penting	5
3.2.3	Fungsi <i>Performance Metrics</i> pada Sistem	6
3.3	System Architecture of Computer Systems	7
3.4	Process Description and Control	7
3.5	Scheduling Algorithms	7
3.6	Process Creation and Termination	8
3.7	Introduction to Threads	8
3.8	File Systems	8
3.9	Input and Output Management	8
3.10	Deadlock Introduction and Prevention	8
3.11	User Interface Management	9
3.12	Virtualization in Operating Systems	9
4	Assignments and Practical Work	9
4.1	Assignment 1: Process Scheduling	9
4.2	Assignment 2: Deadlock Handling	12
4.3	Assignment 3: Multithreading and Amdahl's Law	14
4.4	Assignment 4: Simple Command-Line Interface (CLI) for User Interface Management	18
4.5	Assignment 5: File System Access	18
5	Conclusion	18

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*

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

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

Formula untuk menghitung waktu eksekusi CPU bisa menggunakan dua cara.

Waktu CPU = (CPU Clock Cycle Sebuah Program) x (Waktu Clock Cycle)

atau

$$\text{Waktu CPU} = \frac{(\text{CPU Clock Cycle Sebuah Program})}{(\text{Clock Rate})}$$

Figure 1: Rumus waktu CPU

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

Karena waktu eksekusi CPU merupakan hasil dari kedua faktor ini, kita dapat meningkatkan kinerja dengan mengurangi durasi waktu *clock cycle* atau jumlah *clock cycle* 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 membuat keputusan yang lebih terinformasi, sehingga memperkecil risiko kesalahan dalam strategi. Selain itu, pengukuran kinerja juga membantu mengidentifikasi area yang memerlukan 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 untuk evaluasi dan inovasi secara sistematis dalam mencapai tujuan jangka

panjang.

3.2.3 Fungsi *Performance Metrics* pada Sistem

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

- ***Benchmark***

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

- ***Bandwidth***

Mengukur jumlah data yang dapat ditransmisikan melalui jaringan dalam jangka 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 respons. Pada sistem yang sangat bergantung pada transfer data, seperti aplikasi berbasis web atau layanan *cloud*, *bandwidth* adalah faktor penting yang memengaruhi 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. Tingkat kesalahan yang tinggi dapat mengganggu pengguna dan menurunkan kepercayaan mereka terhadap sistem.

- **Pengaturan Kualitas Layanan (*QoS*)**

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

- **Optimasi dan Tuning**

Performance metrics digunakan untuk mengidentifikasi peluang optimasi dan tuning. Misalnya, dengan

Performance metrics berperan penting dalam berbagai aspek pengelolaan dan operasional sistem operasi. Dari monitoring dan manajemen sumber daya hingga optimasi, diagnosis, dan keamanan, metrik 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 penjadwalan *Shortest Job First* (SJF) dan *Round Robin* (RR), lalu bandingkan rata-rata waktu tunggu (*average waiting time*) dan rata-rata waktu penyelesaian (*average turnaround time*) dari kedua algoritma tersebut. Manakah 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
7     for pid, burst_time in Jumlah_proses:
8         total_waktu_tunggu += waktu_tunggu
9         waktu_tunggu += burst_time
10
11     n = len(Jumlah_proses)
12     avg_waktu_tunggu = total_waktu_tunggu / n
13
14     print(total_waktu_tunggu)
15     print(f"\nAverage Waiting Time: {avg_waktu_tunggu}")

```

Round Robin (RR) algorithms

```

1     def round_robin(jumlah_proses, quantum):
2         n = len(jumlah_proses)
3         remaining_time = [bt for _, bt in jumlah_proses]
4         waktu_tunggu = [0] * n
5         time = 0
6
7         while True:
8             done = True
9             for i in range(n):
10                 if remaining_time[i] > 0:
11                     done = False
12                     if remaining_time[i] > quantum:
13                         time += quantum
14                         remaining_time[i] -= quantum
15                     else:
16                         time += remaining_time[i]
17                         waktu_tunggu[i] = time -
18                             jumlah_proses[i][1]
19                         remaining_time[i] = 0
20
21                 if done:
22                     break
23
24         avg_waktu_tunggu = sum(waktu_tunggu) / n
25         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

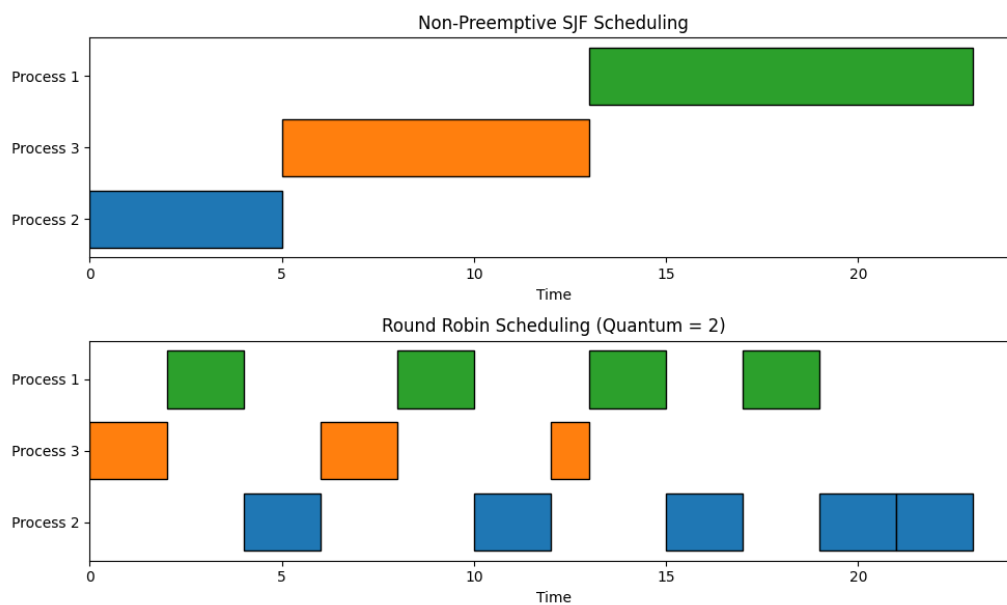


Figure 3: Perbandingan SJF dan RR(quantum = 2)

Dari hasil perbandingan rata-rata waktu tunggu pada kedua algoritma, dapat dilihat bahwa rata-rata waktu tunggu SJF

lebih kecil dibandingkan dengan *Round Robin*. Salah satu alasannya adalah karena sifat RR yang memberikan setiap proses sejumlah waktu yang sama (*quantum*) dan memaksa semua proses menunggu giliran mereka secara bergantian.

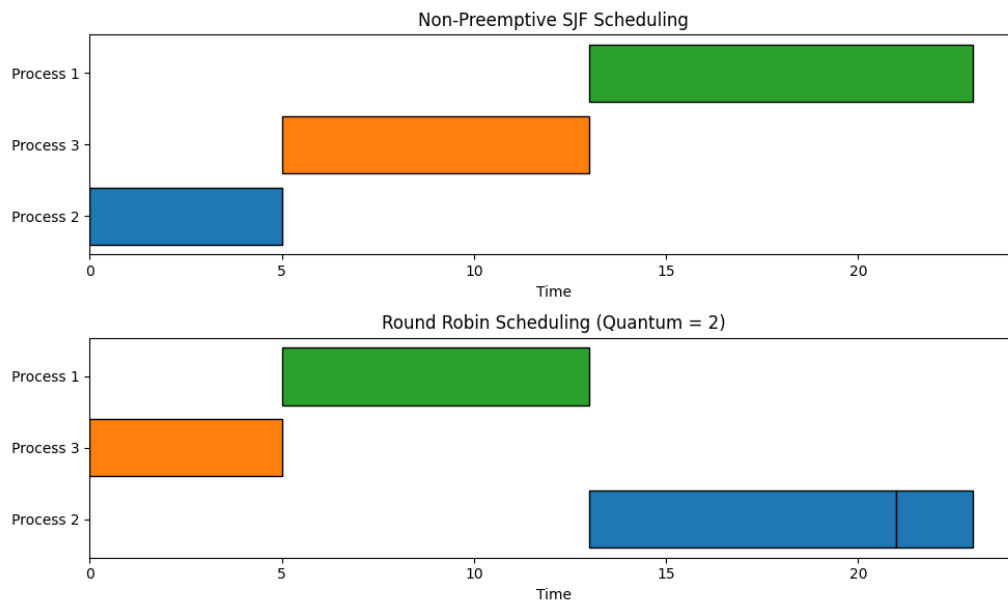


Figure 4: Perbandingan SJF dan RR(quantum = 8)

Kita dapat mengurangi kelemahan *Round Robin* dengan memilih *quantum* yang lebih besar. Pada visualisasi di atas terlihat bahwa waktu tunggu RR dapat sama dengan SJF. Namun, tetap saja, SJF sering kali lebih baik dalam kasus dengan banyak proses yang memiliki *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,
40                                           avail, max_need, allocation)
41
42     if is_safe_state:
43         print("Sistem dalam urutan aman:", safe_sequence)
```

```

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

```

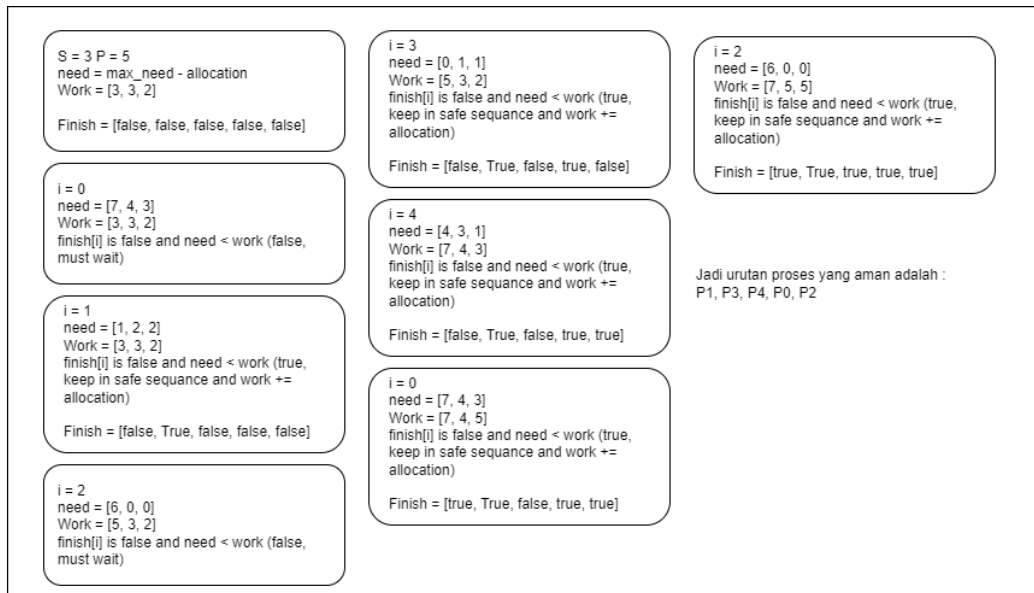


Figure 5: Output dan Penjelasannya

Dengan menggunakan metode *Banker's* untuk algoritma *Safety*, kita bisa memastikan bahwa sistem aman jika menjalankan proses-proses tertentu. Di sini, kita memeriksa setiap proses dan memastikan bahwa alokasi sumber daya untuk setiap proses dapat dilakukan dengan aman sehingga tidak akan menyebabkan *deadlock*. Pada output, sistem akan menampilkan status apakah sistem aman atau tidak, dan menampilkan urutan proses yang aman.

4.3 Assignment 3: Multithreading and Amdahl's Law

Simulasikan skenario *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,
6 execution_times):
7     start_time = time.time()
8     print(f"Thread_{index}_mulai_menghitung_dari_{start}_
9         hingga_{end}...")
10    result = 0
11    for number in range(start, end):
12        time.sleep(0.0001)
13        result += number * number
14    results[index] = result
15    end_time = time.time()
16    execution_times[index] = end_time - start_time
17    print(f"Thread_{index}_selesai_menghitung_dalam_{
18        execution_times[index]:.4f}_detik.")
19
20
21 total_range = 10000
22 num_threads = 4
23 step = total_range // num_threads
24
25
26 results = [0] * num_threads
27 execution_times = [0] * num_threads
28
29 threads = []
30
31 start_time = time.time()
32 for i in range(num_threads):
33     start = i * step
34     end = (i + 1) * step if i != num_threads - 1 else
35         total_range
36     thread = threading.Thread(target=compute_squares,
37         args=(start, end, results, i, execution_times))
38     threads.append(thread)
39     thread.start()
40
41 for thread in threads:
```

```

36         thread.join()
37     end_time = time.time()
38
39     total_execution_time = end_time - start_time
40     total_result = sum(results)
41
42     def amdahl_law(P, N):
43         return 1 / ((1 - P) + (P / N))
44
45
46     sequential_fraction = 0.1
47     P = 1 - sequential_fraction
48
49     print("Semua thread selesai.")
50     print(f"Hasil perhitungan: {total_result}")
51     print(f"Waktu eksekusi total: {total_execution_time:.4f} detik")
52
53     print("Speedup teoretis menurut Hukum Amdahl:")
54     for N in range(1, num_threads + 1):
55         speedup = amdahl_law(P, N)
56         print(f"Speedup teoretis dengan {N} thread: {speedup:.2f}x")

```


OUTPUT :	OUTPUT :
Thread 0 mulai menghitung dari 0 hingga 2500... Thread 1 mulai menghitung dari 2500 hingga 5000... Thread 2 mulai menghitung dari 5000 hingga 7500... Thread 3 mulai menghitung dari 7500 hingga 10000... Thread 1 selesai menghitung dalam 42.7394 detik. Thread 0 selesai menghitung dalam 43.3727 detik. Thread 3 selesai menghitung dalam 43.8989 detik. Thread 2 selesai menghitung dalam 44.3761 detik. Semua thread selesai. Hasil perhitungan: 333283335000 Waktu eksekusi total: 44.4798 detik Speedup teoretis menurut Hukum Amdahl: Speedup teoretis dengan 1 thread: 1.00x Speedup teoretis dengan 2 thread: 1.82x Speedup teoretis dengan 3 thread: 2.50x Speedup teoretis dengan 4 thread: 3.08x	Thread 0 mulai menghitung dari 0 hingga 1250... Thread 1 mulai menghitung dari 1250 hingga 2500... Thread 2 mulai menghitung dari 2500 hingga 3750... Thread 3 mulai menghitung dari 3750 hingga 5000... Thread 4 mulai menghitung dari 5000 hingga 6250... Thread 5 mulai menghitung dari 6250 hingga 7500... Thread 6 mulai menghitung dari 7500 hingga 8750... Thread 7 mulai menghitung dari 8750 hingga 10000... Thread 7 selesai menghitung dalam 21.0458 detik. Thread 4 selesai menghitung dalam 21.5743 detik. Thread 0 selesai menghitung dalam 22.2276 detik. Thread 1 selesai menghitung dalam 22.4029 detik. Thread 2 selesai menghitung dalam 22.4029 detik. Thread 6 selesai menghitung dalam 22.4510 detik. Thread 3 selesai menghitung dalam 22.7923 detik. Thread 5 selesai menghitung dalam 24.6843 detik. Semua thread selesai. Hasil perhitungan: 333283335000 Waktu eksekusi total: 24.8609 detik Speedup teoretis menurut Hukum Amdahl: Speedup teoretis dengan 1 thread: 1.00x Speedup teoretis dengan 2 thread: 1.82x Speedup teoretis dengan 3 thread: 2.50x Speedup teoretis dengan 4 thread: 3.08x Speedup teoretis dengan 5 thread: 3.57x Speedup teoretis dengan 6 thread: 4.00x Speedup teoretis dengan 7 thread: 4.38x Speedup teoretis dengan 8 thread: 4.71x

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

Pada algoritma di atas, *multithreading* berfungsi untuk membagi proses berdasarkan jumlah *thread* yang digunakan, sehingga kita bisa membuat waktu eksekusi menjadi lebih singkat. Dengan menggunakan algoritma *Amdahl's Law*, kita bisa melihat rasio peningkatan kecepatan seiring meningkatnya jumlah *thread* yang digunakan. Dapat dilihat pada perbedaan output di atas, bahwa dengan meningkatkan jumlah *thread* dari 4 ke 8, waktu eksekusi juga meningkat.

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] TestingXperts. (2022, May 10). Performance testing metrics: A detailed guide for businesses. Digital Assurance. Retrieved October 8, 2024, from <https://www.testingxperts.com>
- [2] GeeksforGeeks. (n.d.). Computer organization: Performance of computer. GeeksforGeeks. Retrieved October 1, 2024, from <https://www.geeksforgeeks.org/computer-organization-performance-of-computer/>

- [3] GeeksforGeeks. (2024, August 20). Program for round robin scheduling for the same arrival time. GeeksforGeeks. Retrieved October 8, 2024, from <https://www.geeksforgeeks.org/program-for-round-robin-scheduling-for-the-same-arrival-time/>