Operating System Course Report - First Half of the Semester

A class

October 10, 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 Performa Sistem dan Metrik Sistem Komputer

3.2.1 Performa Sistem

3.2.2 Metrik Sistem Utama

3.2.3 Metrik Sistem Spesifik

3.2.4 Pengukuran dan Optimasi Performa

Performa sistem menjadi aspek penting yang mempengaruhi pengalaman pengguna serta efisiensi operasi. Untuk memastikan sistem berjalan optimal, perlu dilakukan pengukuran yang tepat terhadap berbagai komponen, serta optimasi yang berkelanjutan guna memperbaiki kinerja. Pengukuran performa membantu dalam memahami titik-titik lemah sistem, sementara optimasi bertujuan untuk meningkatkan efisiensi, mengurangi waktu respon, dan memaksimalkan penggunaan sumber daya.

1. Pengukuran performa

   Pengukuran performa mengacu pada proses mengukur sejauh mana suatu sistem, aplikasi, atau proses memenuhi tujuan performa yang

telah ditentukan. Hal ini sangat penting karena memberikan gambaran yang jelas mengenai efektivitas, efisiensi, dan kualitas suatu sistem atau aplikasi. Pengukuran ini melibatkan metrik seperti waktu respon, *throughput*, dan penggunaan sumber daya (CPU, memori, jaringan). Hasil pengukuran dapat digunakan sebagai dasar untuk melakukan optimasi, identifikasi *bottleneck*, serta perbaikan sistem yang lebih lanjut agar performa sistem dapat lebih stabil dan optimal dalam memenuhi kebutuhan pengguna.

(a) *Benchmarking*

Benchmarking adalah metode pengujian serangkaian program dengan cara membandingkan performa suatu sistem terhadap performa standar untuk mendapatkan performa relatif dari komponen PC atau sistem[2]. Tujuan dari *benchmarking* adalah untuk memberikan gambaran yang jelas tentang performa sistem komputer sehingga dapat dipastikan bahwa teknologi atau perangkat yang digunakan optimal dan memenuhi standar industri. Ada dua jenis dari *benchmarking*:

- *Synthetic benchmarking*: Simulasi skenario tertentu untuk mengukur potensi maksimum performa sistem, seperti menggunakan SPEC CPU untuk menguji kemampuan komputasi CPU.
- *Real-world Benchmarking*: Pengukuran performa sistem menggunakan aplikasi nyata dalam kondisi operasional sehari-hari, seperti Adobe Premiere Pro untuk menguji kecepatan *rendering* video.

(b) *Profiling*

Menurut Altvater [1], profiling adalah metode untuk menganalisis performa aplikasi atau sistem secara mendalam dengan fokus pada penggunaan sumber daya internal. *Profiling* membantu mengidentifikasi bagian dari sistem atau program yang mengkonsumsi sumber daya paling banyak, seperti CPU, memori, I/O, dan waktu eksekusi. Dengan demikian, *profiling* digunakan untuk menemukan dan memperbaiki "*bottleneck*" dalam aplikasi atau sistem, memungkinkan pengembang untuk melakukan optimasi yang tepat. Berikut beberapa metode dalam *profiling*:

- *CPU profiling*: Mengukur penggunaan CPU untuk mengidentifikasi kode yang paling memakan waktu.
- *Memory profiling*: Mengukur alokasi memori dan menemukan kebocoran memori.
- *I/O profiling*: Menganalisis performa operasi *input/output* seperti *file* atau jaringan.
- *Function-level profiling*: Menganalisis fungsi dalam aplikasi, melihat frekuensi pemanggilan dan durasi.

(c) *Monitoring*

Monitoring adalah proses pengamatan dan pengukuran performa sistem secara *real-time* dan berkelanjutan untuk memastikan sistem berjalan optimal serta mendeteksi masalah atau potensi gangguan[4]. *Monitoring* sangat penting untuk menjaga stabilitas dan performa sistem komputer atau aplikasi, terutama dalam lingkungan produksi, di mana *uptime* dan keandalan menjadi prioritas. Beberapa teknik yang sering digunakan dalam *monitoring* antara lain:

- *Real-time monitoring*:
  Memantau sistem secara langsung dan memberikan informasi performa atau kegagalan segera setelah terjadi. *Real-time monitoring* sangat penting untuk aplikasi dengan kebutuhan *uptime* tinggi, seperti layanan berbasis *cloud*, sistem *e-commerce*, atau *server*.
- *Historical monitoring*:
  Mengumpulkan data performa selama jangka waktu tertentu dan menyimpannya untuk dianalisis kemudian. Ini penting untuk analisis tren dan perencanaan kapasitas, karena memungkinkan tim IT untuk melihat bagaimana sistem telah beroperasi dalam jangka waktu tertentu.
- *Threshold-based monitoring*:
  Sistem monitoring yang memberikan notifikasi ketika nilai performa melebihi atau di bawah batas tertentu. Misalnya, jika penggunaan CPU melebihi 90% atau penggunaan memori terlalu rendah, sistem akan mengirimkan peringatan kepada administrator. Metode ini memungkinkan deteksi dini masalah performa dan membantu mencegah *downtime* sistem. Selain CPU dan memori, *threshold* juga dapat diterapkan pada

metrik lain seperti penggunaan *disk*, lalu lintas jaringan, atau waktu respons aplikasi. Administrator dapat menyesuaikan nilai ambang batas sesuai dengan kebutuhan spesifik sistem mereka.

Aspek	Benchmarking	Profiling	Monitoring
Fokus	Performa keseluruhan terhadap standar	Analisis mendalam per bagian sistem	Pemantauan performa *real-time*
Tujuan	Perbandingan performa	Optimasi performa	Menjaga kestabilan sistem
Dilakukan saat	Di bawah kondisi spesifik (uji beban)	Saat pengembangan atau *testing*	Selama sistem berjalan (operasional)
Hasil utama	Angka perbandingan	Identifikasi *bottleneck*	Data penggunaan sumber daya

Table 1: Perbedaan dari *Benchmarking*, *Profiling*, dan *Monitoring*

2. Strategi Optimasi Performa

   Strategi optimasi performa bertujuan untuk mengatasi beberapa masalah yang telah diidentifikasi atau diukur performanya[3].

   (a) Peningkatan *hardware*

   Peningkatan *hardware* mengacu pada peningkatan kapasitas fisik perangkat keras komputer untuk meningkatkan performa sistem. Beberapa cara peningkatan *hardware* antara lain:

   - Penambahan CPU/*Core*: Meningkatkan eksekusi proses dan *multitasking*.
   - Penambahan RAM: Menjalankan lebih banyak aplikasi tanpa bergantung pada *disk*.
   - Penggunaan SSD: Mempercepat baca/tulis data dibanding HDD.
   - Jaringan lebih Cepat: Mempercepat transfer data dengan *bandwidth* tinggi.

(b) Optimalisasi *Software*

Optimalisasi *software* bertujuan untuk memaksimalkan efisiensi kode program dan cara perangkat lunak bekerja pada *hardware* yang tersedia. Berikut adalah beberapa langkah untuk mengoptimalkan *software*:

- Optimalisasi algoritma: Mengurangi kompleksitas untuk mempercepat proses.
- *Caching*: Menyimpan data sering diakses untuk mempercepat.
- *Database tuning*: Indeksasi dan optimasi *query* untuk eksekusi lebih cepat.
- Kompresi data: Mengurangi ukuran data untuk transfer lebih cepat.
- Pengurangan *latency*: Menggunakan CDN untuk akses lebih cepat.

(c) *Load balancing*

Load balancing adalah teknik distribusi beban kerja di beberapa *server* atau sumber daya untuk memastikan bahwa tidak ada satu *server* yang kelebihan beban, sementara *server* lainnya tidak terpakai secara maksimal. *Load balancing* dapat diterapkan pada sistem yang berbasis *server* untuk aplikasi *web*, basis data, atau layanan *cloud*.

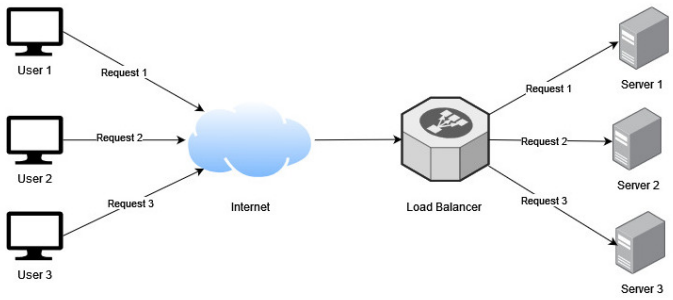

Figure 1: Ilustrasi proses *load balancing*

Pada gambar 1, tiga pengguna (*User* 1, *User* 2, *User* 3) mengirimkan permintaan melalui *internet*. Permintaan tersebut didistribusikan oleh *load balancer* ke beberapa *server* (*Server* 1, *Server*

2, *Server* 3), memastikan beban kerja tersebar merata dan tidak ada *server* yang kelebihan beban.

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

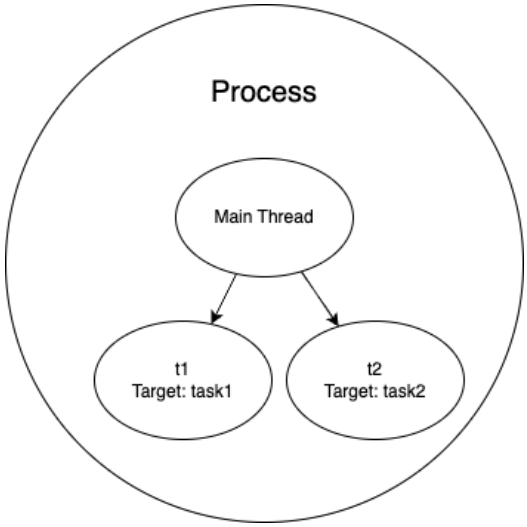

Figure 2: Ini adalah gambar contoh dari multithreading.

Seperti yang terlihat pada Gambar 2, inilah cara menambahkan gambar dengan keterangan.

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

Students were tasked with implementing various process scheduling algorithms (e.g., FCFS, SJN, and RR) and comparing their performance under different conditions.

4.1.1 Group 1

```
    class Process:
    def __init__(self, pid, arrival_time, burst_time):
        self.pid = pid
        self.arrival_time = arrival_time
        self.burst_time = burst_time
        self.completion_time = 0
        self.turnaround_time = 0
        self.waiting_time = 0
```

Header 1	Header 2	Header 3
Row 1, Column 1	Row 1, Column 2	Row 1, Column 3
Row 2, Column 1	Row 2, Column 2	Row 2, Column 3

Table 2: Your table caption

4.2 Assignment 2: Deadlock Handling

In this assignment, students were asked to simulate different deadlock scenarios and explore various prevention methods.

4.3 Assignment 3: Multithreading and Amdahl's Law

This assignment involved designing a multithreading scenario to solve a computationally intensive problem. Students then applied **Amdahl's Law** to calculate the theoretical speedup of the program as the number of threads increased.

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

4.4.1 Group 2

Soal:

Buatlah sebuah program Python sederhana yang berfungsi sebagai *Command-Line Interface* (CLI) untuk manajemen antarmuka pengguna. Program ini harus mendukung beberapa perintah dasar, seperti manipulasi *file* (membuat, menampilkan daftar, dan menghapus *file*), manajemen proses, dan pelaporan status sistem. Berikut adalah ketentuan-ketentuan dari program yang harus dibuat:

- Buat *file* dengan nama tertentu.
- Tampilkan daftar *file* dalam direktori saat ini.
- Hapus *file* berdasarkan nama *file*.
- Tampilkan daftar proses yang sedang berjalan.
- Tampilkan status penggunaan CPU dan memori saat ini.

Jawaban:

Untuk menjawab soal ini, kita akan membuat program Python sederhana yang berfungsi sebagai *Command-Line Interface* (CLI) untuk manajemen *interface* pengguna. Program ini akan mendukung beberapa fitur utama seperti manipulasi *file*, manajemen proses, dan pelaporan status sistem. Berikut adalah implementasi program tersebut:

```
import os
import psutil

def tampilkan_welcome_screen():
    print(
        """
    \033[32m ___  ___       _____ _      _____\033[0m
    \033[32m|   \/  |      /  __ \ |    |_   _|\033[0m
    \033[32m| .  . |_   _| /  \/ |      | |  \033[0m
    \033[32m| |\/| | | | | |    | |      | |  \033[0m
    \033[32m| |  | | |_| | \__/\ |_____| |_ \033[0m
    \033[32m\_|  |_/\__, |\____/\_____/\___/\033[0m
    \033[32m         __/ |                  \033[0m
```

```
    \033[32m        |___/                    \033[0m

    \033[35mPenggunaan: python mycli.py\033[0m

    \033[36mMengelola file dan menampilkan informasi
                                    sistem\033[0m
    """)

def tampilkan_daftar_file():
    try:
        files = os.listdir('.')
        print("Daftar file:", ", ".join(files) if files
                                        else "Tidak ada
                                        file")
    except Exception as e:
        print(f"Gagal menampilkan daftar file: {e}")

def tampilkan_daftar_proses():
    try:
        for proc in psutil.process_iter(['pid', 'name']):
            print(f"PID: {proc.info['pid']}, Nama: {proc.
                                            info['name']}"
                                            )
    except Exception as e:
        print(f"Gagal menampilkan daftar proses: {e}")

def tampilkan_status_sistem():
    try:
        cpu = psutil.cpu_percent()
        mem = psutil.virtual_memory()
        print(f"CPU: {cpu}%, RAM: {mem.percent}% ({mem.
                                        used/1e9:.1f}/{mem
                                        .total/1e9:.1f} GB
                                        )")
    except Exception as e:
        print(f"Gagal menampilkan status sistem: {e}")

def main():
    tampilkan_welcome_screen()

    perintah = {
        '1': ('Buat file',
                lambda: open(input("Nama file: "), 'w').
                                                close()),
        '2': ('Tampilkan daftar file',
```

```
                tampilkan_daftar_file),
        '3': ('Hapus file',
                lambda: os.remove(input("Nama file: "))),
        '4': ('Tampilkan daftar proses',
                tampilkan_daftar_proses),
        '5': ('Tampilkan status sistem',
                tampilkan_status_sistem),
        '6': ('Keluar',
                exit)
    }

    while True:
        print("\n" + "\n".join(f"{k}. {v[0]}" for k, v in
                                perintah.items())
                                )
        try:
            perintah[input("Pilih perintah: ")][1]()
        except Exception as e:
            print(f"Error: {e}")

if __name__ == "__main__":
    main()
```

Hasil *output*:

Figure 3: *output assignment* 4

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] Altvater, A. (2023). *What is code profiling? learn the 3 types of code profilers.* Stackify. Diakses pada 1 oktober 2024, dari https://stackify.com/what-is-code-profiling/

[2] Bhat, A. (2021). *Benchmarking in computer.*Benchmarking in computer. Medium. Diakses pada 1 oktober 2024, dari https://bhatabhishekylp.medium.com/benchmarking-in-computer-c6d364681512

[3] GeeksforGeeks. (2024). *Performance of computer in Computer Organization.* Diakses pada 1 oktober 2024, dari https://www.geeksforgeeks.org/computer-organization-performance-of-computer/

[4] Wikimedia Foundation. (2023). *Monitoring.* Wikipedia. Diakses pada 1 oktober 2024, dari https://en.wikipedia.org/wiki/Monitoring