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

Performa sistem komputer adalah kemampuan suatu sistem untuk menjalankan tugas-tugas komputasi sesuai dengan spesifikasi dan parameter yang telah ditetapkan. Performa ini mencakup kecepatan, efisiensi, dan ketepatan dalam menyelesaikan berbagai proses dan operasi yang diminta oleh pengguna atau aplikasi.

1. Faktor Pengaruh performa

   (a) CPU (*(Central Processing Unit)*
   CPU atau prosesor adalah otak dari komputer yang bertanggung jawab untuk menjalankan instruksi dan proses komputasi. Kecepatan *CPU* diukur dalam *gigahertz (GHz)*; semakin tinggi frekuensi, semakin cepat proses eksekusi instruksi. Jumlah *core* pada CPU juga penting; semakin banyak *core*, semakin baik komputer dalam menangani *multitasking* dan aplikasi yang membutuhkan banyak sumber daya[4].

(b) GPU *(Graphics Processing Unit)*

GPU bertanggung jawab untuk menangani pemrosesan grafis dan rendering visual. Dalam aplikasi yang memerlukan performa grafis tinggi seperti game dan desain 3D, GPU yang kuat menjadi sangat penting. GPU modern juga digunakan untuk komputasi paralel di bidang kecerdasan buatan dan pembelajaran mesin, GPU dapat menyelesaikan tugas sederhana dan berulang jauh lebih cepat karena dapat memecah tugas menjadi komponen yang lebih kecil dan menyelesaikannya secara paralel[5].

(c) RAM *(Random Access Memory)*

RAM adalah memori sementara yang digunakan oleh sistem untuk menyimpan data yang aktif digunakan atau diakses oleh CPU dengan cepat. Jumlah dan kecepatan RAM biasanya meningkatkan performa sistem dengan memungkinkan lebih banyak data digunakan tanpa harus menggunakan penyimpanan yang lebih lambat[6].

2. Keterkaitan Hardware

(a) Keterkaitan antara RAM dan CPU

CPU bergantung pada RAM untuk menyimpan data sementara yang sedang diproses. Ketika CPU menjalankan tugas, ia membutuhkan data yang dapat diakses dengan cepat. RAM menyediakan ruang penyimpanan sementara yang memungkinkan CPU untuk mengakses data dengan kecepatan tinggi, yang membantu mencegah penundaan dalam pemrosesan. Semakin besar kapasitas dan kecepatan RAM, semakin cepat CPU dapat memproses data, terutama dalam situasi *multitasking* ketika banyak aplikasi berjalan secara bersamaan.

(b) Keterkaitan antara CPU dan GPU

CPU dan GPU bekerja bersama-sama untuk membagi tugas yang memerlukan pemrosesan berat. CPU menangani tugas-tugas umum seperti logika, kontrol aplikasi, dan aliran data, sementara GPU menangani tugas-tugas yang membutuhkan komputasi paralel, seperti rendering grafis atau komputasi numerik. Kinerja keseluruhan sistem meningkat ketika CPU dan GPU dapat bekerja secara seimbang. Jika salah satu komponen terlalu lambat dibandingkan den-

gan yang lain, bisa terjadi *bottleneck*, ketika salah satu perangkat keras menahan kinerja perangkat lainnya.

(c) Keterkaitan antara Penyimpanan dan RAM

RAM dan penyimpanan juga bekerja sama untuk memastikan kelancaran operasi sistem. Ketika aplikasi atau data yang dibutuhkan oleh CPU tidak dapat seluruhnya ditampung di RAM, sistem akan menggunakan penyimpanan sebagai memori virtual. Jika penyimpanan yang digunakan adalah SSD (*Solid State Drive*), maka akses data dari penyimpanan ke RAM menjadi jauh lebih cepat dibandingkan dengan HDD (*Hard Disk Drive*) tradisional, sehingga mempercepat pemuatan aplikasi dan respons sistem secara keseluruhan.

3.2.2 Metrik Sistem Utama

1. Throughput

Throughput adalah jumlah output yang dapat diselesaikan oleh sistem dalam jangka waktu tertentu, misalnya jumlah permintaan yang dilayani oleh server dalam satu detik. Contoh Kasus: Server Web: Sebuah server web yang mampu melayani 200 permintaan HTTP per detik memiliki throughput sebesar 200 request per second. Jika throughput rendah, server mungkin membutuhkan peningkatan kapasitas. Optimasi: Throughput dapat ditingkatkan dengan mempercepat prosesor atau meningkatkan bandwidth jaringan untuk menangani lebih banyak permintaan secara simultan.

2.Latency Definisi: Latency adalah waktu yang diperlukan untuk menyelesaikan satu unit kerja dari awal hingga akhir, misalnya waktu tunggu antara saat data dikirim dan diterima. Contoh Kasus: Sistem Jaringan: Dalam jaringan, jika waktu yang dibutuhkan untuk mengirim paket data dari satu komputer ke komputer lain adalah 20 milidetik, ini mencerminkan tingkat latency. Latency yang tinggi dapat menyebabkan keterlambatan dalam komunikasi, yang dapat mengganggu aplikasi seperti video conference. Optimasi: Latency dapat dikurangi dengan menggunakan jalur komunikasi yang lebih cepat atau memperbaiki routing jaringan untuk menghindari keterlambatan.

Utilization Definisi: Utilization mengukur tingkat penggunaan sumber daya sistem, seperti CPU atau memori, dan biasanya dinyatakan dalam persentase. Contoh: Server Database: Jika sebuah server database menunjukkan CPU utilization sebesar 90sangat keras, yang bisa menjadi tanda

overutilization. Overutilization dapat menyebabkan penurunan performa keseluruhan sistem. Optimasi: Utilization dapat dikurangi dengan menambahkan lebih banyak CPU, menyeimbangkan beban kerja di antara server lain, atau mengoptimalkan kode aplikasi untuk efisiensi yang lebih tinggi.

Response time Definisi: Response time adalah waktu yang diperlukan oleh sistem untuk merespons permintaan dari pengguna, misalnya waktu yang diperlukan untuk memproses pembayaran dalam aplikasi e-commerce.

3.3 System Performance and Metrics

3.3.1 Metrik Sistem Spesifik

1. *Cycle Per Instructions* (CPI)

   Cycle Per Instructions adalah salah satu metrik utama yang digunakan untuk mengevaluasi performa *Central Processing Unit* (CPU). CPI mengukur rata-rata jumlah siklus clock yang dibutuhkan oleh CPU untuk mengeksekusi satu instruksi.

   Formula umum CPI:

   CPI = *Total Cycle/Instructions*

   Rendahnya CPI berarti CPU dapat mengeksekusi instruksi lebih efisien, menunjukkan performa yang lebih baik sedangkan tingginya CPI menunjukkan bahwa CPU memerlukan lebih banyak siklus clock untuk mengeksekusi instruksi, yang bisa berarti instruksi tersebut lebih kompleks atau ada masalah dalam *pipeline* CPU.

   Contoh Penggunaan CPI

   Misalkan kamu memiliki dua CPU:

   CPU A memiliki CPI 1.2

   CPU B memiliki CPI 1.8

   Jika keduanya berjalan pada frekuensi yang sama, katakanlah 3 GHz, CPU A akan lebih efisien karena membutuhkan lebih sedikit siklus per instruksi. Ini berarti CPU A dapat menyelesaikan lebih banyak instruksi dalam waktu yang sama dibandingkan CPU B.

2. *Floating Point Operations Per Second* (FLOPS)

Floating Point Operations Per Second adalah metrik yang digunakan untuk mengukur kemampuan komputasi *floating-point* dari sebuah komputer, yang sangat penting dalam aplikasi yang memerlukan kalkulasi numerik berat, seperti simulasi ilmiah, pemodelan 3D, atau *machine learning*.

Contoh Penggunaan FLOPS:

Bayangkan kamu bekerja di bidang simulasi ilmiah yang memerlukan perhitungan numerik yang intensif. Sistem A memiliki kapasitas 2 TFLOPS, sementara Sistem B memiliki 5 TFLOPS. Sistem B akan mampu menyelesaikan simulasi tersebut lebih cepat, karena dapat melakukan lebih banyak operasi *floating-point* per detik.

3. *Input/Output Operations Per Second* (IOPS)

   Input/Output Operations Per Second mengukur performa dari perangkat penyimpanan (seperti SSD, HDD) dalam hal jumlah operasi *input/output* yang dapat diproses dalam satu detik. IOPS sering digunakan untuk mengevaluasi performa *disk* dan sistem penyimpanan.

   Contoh Penggunaan IOPS:

   Misalkan kamu mengelola server database. SSD A memiliki 100,000 IOPS, sementara SSD B memiliki 50,000 IOPS. SSD A akan mampu menangani lebih banyak operasi baca/tulis per detik, sehingga lebih cocok untuk lingkungan database yang memerlukan akses data cepat dan simultan.

3.3.2 Pengukuran dan Optimasi Performa

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.

   (a) Benchmarking

   Benchmarking adalah metode pengujian serangkaian program dengan cara membandingkan performa suatu sistem terhadap per-

forma standar untuk mendapatkan performa relatif dari komponen PC atau sistem. 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 2 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

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. *Tools* yang digunakan yaitu gprof dan Perf.
- Memory Profiling: Mengukur alokasi memori dan menemukan kebocoran memori. *Tools* yang digunakan yaitu Valgrind dan Heap Profiler.
- I/O Profiling: Menganalisis performa operasi input/output seperti file atau jaringan. *Tools* yang digunakan yaitu IOTop dan dstat.
- Function-Level Profiling: Menganalisis fungsi dalam aplikasi, melihat frekuensi pemanggilan dan durasi. *Tools* yang digunakan yaitu Xdebug (PHP) dan py-spy (Python).

(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. Monitoring sangat penting untuk menjaga stabilitas dan performa sistem komputer atau aplikasi, terutama dalam lingkungan produksi, ketika uptime dan keandalan menjadi prioritas. Ada beberapa teknik yang sering digunakan dalam monitoring:

- 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 atau alarm 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.

2. Strategi Optimasi Performa

(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*.

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

- Penggunaan SSD: Mempercepat baca/tulis data dibanding HDD.
- Jaringan lebih Ccpat: 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*.

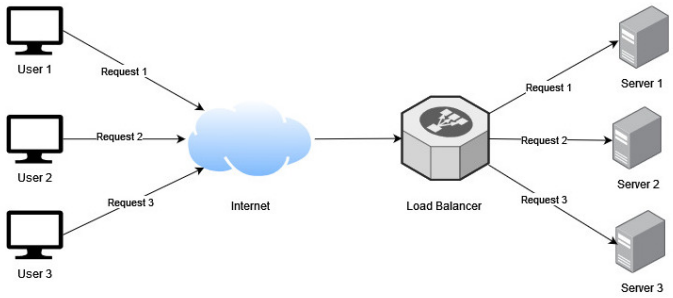

Figure 1: Ilustrasi proses *load balancing*

Pada gambar 3, 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.

3.4 System Architecture of Computer Systems

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

3.5 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

3.6 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.

3.7 Process Creation and Termination

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

- Process spawning
- Process termination conditions

3.8 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

Seperti yang terlihat pada Gambar 4, inilah cara menambahkan gambar dengan keterangan.

3.9 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

Figure 2: Ini adalah gambar contoh dari multithreading.

3.10 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

3.11 Deadlock Introduction and Prevention

Explores the concept of deadlocks and methods for preventing them:

- Deadlock conditions
- Deadlock prevention techniques

3.12 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

3.13 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

4.1.1 Group 2

Soal:

Buatlah sebuah program Python sederhana yang dapat mensimulasikan algoritma penjadwalan proses dengan menggunakan Round Robin Scheduling. Program ini harus mendukung beberapa fitur berikut:

- Menerima daftar proses dengan burst time masing-masing.
- Mengatur proses dengan kuantum waktu tetap.
- Menampilkan urutan eksekusi proses serta waktu selesai masing-masing.
- Hitung waktu rata-rata turnaround time dan waiting time.

Jawaban: Untuk menyelesaikan soal ini, berikut adalah implementasi dari algoritma Round Robin Scheduling sederhana dalam Python:

```
python
Copy code
    while True:
        done = True
```

```
        for i in range(n):
            if remaining_time[i] > 0:
                done = False  # Ada proses yang belum selesai
                if remaining_time[i] > quantum:
                    t += quantum
                    remaining_time[i] -= quantum
                else:
                    t += remaining_time[i]
                    waiting_time[i] = t - burst_time[i]
                    remaining_time[i] = 0
        if done:
            break

    # Hitung turnaround time
    for i in range(n):
        turnaround_time[i] = burst_time[i] + waiting_time[i]

    print("Proses\tBurst Time\tWaiting Time\tTurnaround Time"
                                    )
    for i in range(n):
        print(f"{processes[i]}\t{burst_time[i]}\t\t{
                                        waiting_time[i]}\t\t{
                                        turnaround_time[i]}")

    print(f"\nRata-rata Waiting Time: {sum(waiting_time) / n
                                    :.2f}")
    print(f"Rata-rata Turnaround Time: {sum(turnaround_time)
                                    / n:.2f}")

if __name__ == "__main__":
    processes = ['P1', 'P2', 'P3']
    burst_time = [24, 3, 3]
    quantum = 4
    round_robin(processes, burst_time, quantum)
```

4.2 Assignment 2: Deadlock Handling

4.2.1 Grup 2

Soal:

Buatlah sebuah program Python sederhana yang mensimulasikan beberapa skenario deadlock dan mengeksplorasi berbagai metode pencegahan deadlock. Program ini harus mendukung beberapa fitur berikut:

- Simulasi skenario deadlock dengan penggunaan dua atau lebih sumber daya oleh beberapa proses.
- Implementasi metode pencegahan deadlock seperti pengurutan sumber daya, batasan jumlah maksimum sumber daya yang dapat dialokasikan, atau penggunaan algoritma Banker's.
- Visualisasi atau laporan mengenai status deadlock dan bagaimana pencegahannya dapat diterapkan.
- Tampilkan hasil dari skenario deadlock yang disimulasikan dan solusi pencegahannya.

Jawaban:

Untuk menjawab soal ini, kita akan membuat program Python sederhana yang mensimulasikan skenario deadlock serta menerapkan metode pencegahannya. Berikut adalah implementasi dari program tersebut:

```
import threading
import time

# Simulasi sumber daya
class SumberDaya:
    def __init__(self, nama):
        self.nama = nama
        self.lock = threading.Lock()

def gunakan(self, pengguna):
    with self.lock:
        print(f"{pengguna} menggunakan {self.nama}.")
        time.sleep(1)  # Simulasi waktu penggunaan sumber
                                        daya

# Fungsi untuk simulasi deadlock
def deadlock_scenario(sumber1, sumber2, pengguna):
    print(f"{pengguna} mencoba mengakses {sumber1.nama}."
                                        )
    sumber1.gunakan(pengguna)

    print(f"{pengguna} mencoba mengakses {sumber2.nama}."
                                        )
    sumber2.gunakan(pengguna)

# Pencegahan deadlock: mengatur urutan pengaksesan sumber
                                    daya
```

```
def no_deadlock_scenario(sumber1, sumber2, pengguna):
    sumber_list = sorted([sumber1, sumber2], key=lambda s
                                    : s.nama)

    print(f"{pengguna} mencoba mengakses {sumber_list[0].
                                    nama}.")
    sumber_list[0].gunakan(pengguna)

    print(f"{pengguna} mencoba mengakses {sumber_list[1].
                                    nama}.")
    sumber_list[1].gunakan(pengguna)

def main():
    # Membuat sumber daya
    sumber_a = SumberDaya('SumberA')
    sumber_b = SumberDaya('SumberB')

# Simulasi deadlock
thread1 = threading.Thread(target=deadlock_scenario, args
                                =(sumber_a, sumber_b, "
                                Proses1"))
thread2 = threading.Thread(target=deadlock_scenario, args
                                =(sumber_b, sumber_a, "
                                Proses2"))

print("\nSimulasi Deadlock:")
thread1.start()
thread2.start()

thread1.join()
thread2.join()

# Simulasi tanpa deadlock dengan pengurutan akses sumber
                                daya
print("\nSimulasi Pencegahan Deadlock:")
thread3 = threading.Thread(target=no_deadlock_scenario,
                                args=(sumber_a, sumber_b,
                                "Proses1"))
thread4 = threading.Thread(target=no_deadlock_scenario,
                                args=(sumber_b, sumber_a,
                                "Proses2"))

thread3.start()
thread4.start()
```

```
    thread3.join()
    thread4.join()

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

4.3 Assignment 3: Multithreading and Amdahl's Law

4.3.1 Grup 2

Soal:

Buatlah sebuah program Python yang mensimulasikan skenario multithreading untuk menyelesaikan masalah komputasi intensif. Setelah itu, terapkan Hukum Amdahl untuk menghitung kecepatan teoretis program tersebut ketika jumlah thread ditingkatkan. Program harus memenuhi beberapa kriteria berikut:

- Selesaikan masalah komputasi intensif (misalnya, penjumlahan bilangan dalam rentang besar) menggunakan beberapa thread.
- Hitung waktu eksekusi untuk setiap jumlah thread yang digunakan.
- Terapkan Hukum Amdahl untuk menghitung kecepatan teoretis ketika menambah jumlah thread.
- Bandingkan hasil teoritis dan hasil nyata dari waktu eksekusi program.

Jawaban:

Berikut adalah implementasi program yang menggunakan multithreading untuk menyelesaikan masalah komputasi intensif, diikuti dengan perhitungan kecepatan teoretis menggunakan Hukum Amdahl:

```
        import threading
import time
import os

# Fungsi komputasi intensif: penjumlahan bilangan dari 1
                                hingga N
def komputasi_intensif(n, hasil, index):
    total = 0
    for i in range(1, n+1):
        total += i
```

```
    hasil[index] = total
    print(f"Thread-{index+1} selesai, hasil: {total}")

# Fungsi untuk menghitung waktu eksekusi menggunakan
                                multithreading
def hitung_waktu_multithreading(n, jumlah_thread):
    hasil = [0] * jumlah_thread
    threads = []
    bagian = n // jumlah_thread

    start_time = time.time()

    for i in range(jumlah_thread):
        batas_atas = (i+1) * bagian if i != jumlah_thread - 1
                                        else n
        t = threading.Thread(target=komputasi_intensif, args=
                                        (batas_atas, hasil, i)
                                        )
        threads.append(t)
        t.start()

    for t in threads:
        t.join()

    end_time = time.time()

    total_hasil = sum(hasil)
    waktu_eksekusi = end_time - start_time
    print(f"Total hasil: {total_hasil}, Waktu eksekusi dengan
                                    {jumlah_thread} thread: {
                                    waktu_eksekusi:.4f} detik"
                                    )
    return waktu_eksekusi

# Hukum Amdahl: menghitung kecepatan teoretis
def hitung_kecepatan_teoretis(serial_fraction, jumlah_thread)
                                    :
    return 1 / (serial_fraction + (1 - serial_fraction) /
                                    jumlah_thread)

def main():
    n = 10**7  # Besar masalah (penjumlahan hingga N)
    jumlah_threads = [1, 2, 4, 8]  # Daftar jumlah thread
                                    yang akan diuji
    serial_fraction = 0.1  # Asumsi fraksi yang tidak bisa
```

```
                                    diparalelkan (10%)

    waktu_eksekusi_awal = None

    # Bandingkan waktu eksekusi dengan jumlah thread yang
                                    berbeda
    for thread_count in jumlah_threads:
        waktu_eksekusi = hitung_waktu_multithreading(n,
                                        thread_count)

        if thread_count == 1:
            waktu_eksekusi_awal = waktu_eksekusi  # Simpan
                                            waktu eksekusi
                                            serial (1 thread)

        # Hitung kecepatan teoretis menggunakan Hukum Amdahl
        kecepatan_teoretis = hitung_kecepatan_teoretis(
                                        serial_fraction,
                                        thread_count)
        print(f"Kecepatan teoretis dengan {thread_count}
                                        thread: {
                                        kecepatan_teoretis:.4f
                                        }x")

        # Hitung kecepatan nyata berdasarkan hasil eksekusi
        kecepatan_nyata = waktu_eksekusi_awal /
                                        waktu_eksekusi
        print(f"Kecepatan nyata dengan {thread_count} thread:
                                         {kecepatan_nyata:.4f}
                                        x\n")

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

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 (mem-

buat, 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 buat_file(nama_file):
    try:
        with open(nama_file, 'w') as f:
            f.write('')  # Membuat file kosong
        print(f"File '{nama_file}' berhasil dibuat.")
    except Exception as e:
        print(f"Gagal membuat file: {e}")

def hapus_file(nama_file):
    try:
        os.remove(nama_file)
        print(f"File '{nama_file}' berhasil dihapus.")
    except FileNotFoundError:
        print(f"File '{nama_file}' tidak ditemukan.")
    except Exception as e:
        print(f"Gagal menghapus file: {e}")

def tampilkan_daftar_file():
    try:
        files = os.listdir('.')
```

```
            if files:
                print("Daftar file di direktori saat ini:")
                for file in files:
                    print(file)
            else:
                print("Tidak ada file di direktori saat ini."
                                                    )
        except Exception as e:
            print(f"Gagal menampilkan daftar file: {e}")

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

    def tampilkan_status_sistem():
        try:
            cpu_percent = psutil.cpu_percent(interval=1)
            memory = psutil.virtual_memory()
            print(f"Penggunaan CPU: {cpu_percent}%")
            print(f"Total Memori: {memory.total / (1024 ** 3)
                                            :.2f} GB")
            print(f"Memori yang Digunakan: {memory.used / (
                                            1024 ** 3):.2f} GB
                                             ({memory.percent}
                                            %)")
        except Exception as e:
            print(f"Gagal menampilkan status sistem: {e}")

    def main():
        while True:
            print("\nPilih perintah:")
            print("1. Buat file")
            print("2. Tampilkan daftar file")
            print("3. Hapus file")
            print("4. Tampilkan daftar proses")
            print("5. Tampilkan status sistem")
            print("6. Keluar")
```

```
            pilihan = input("Masukkan pilihan (1-6): ")

            if pilihan == '1':
                nama_file = input("Masukkan nama file yang
                                        akan dibuat: "
                                        )
                buat_file(nama_file)
            elif pilihan == '2':
                tampilkan_daftar_file()
            elif pilihan == '3':
                nama_file = input("Masukkan nama file yang
                                        akan dihapus:
                                        ")
                hapus_file(nama_file)
            elif pilihan == '4':
                tampilkan_daftar_proses()
            elif pilihan == '5':
                tampilkan_status_sistem()
            elif pilihan == '6':
                print("Keluar dari program.")
                break
            else:
                print("Pilihan tidak valid. Silakan coba lagi
                                        .")

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

4.5 Assignment 5: File System Access

4.5.1 Group 2

Soal:

Buatlah sebuah program Python sederhana untuk melakukan akses sistem file. Program ini harus mendukung beberapa perintah berikut:

- Membaca isi file teks.
- Menambahkan teks ke dalam file yang sudah ada.
- Menampilkan status akses file, seperti siapa pemiliknya, waktu modifikasi terakhir, dan izin akses file.

Jawaban: Berikut adalah implementasi dari program Python yang dapat mengakses sistem file dan mendukung fitur-fitur yang diminta:

```
python
Copy code
def baca_file(nama_file):
    try:
        with open(nama_file, 'r') as f:
            print(f"\nIsi file '{nama_file}':")
            print(f.read())
    except FileNotFoundError:
        print(f"File '{nama_file}' tidak ditemukan.")
    except Exception as e:
        print(f"Gagal membaca file: {e}")

def tambah_teks(nama_file, teks):
    try:
        with open(nama_file, 'a') as f:
            f.write(teks + '\n')
        print(f"Teks berhasil ditambahkan ke file '{
                                    nama_file}'.")
    except Exception as e:
        print(f"Gagal menambahkan teks: {e}")

def status_file(nama_file):
    try:
        stat_info = os.stat(nama_file)
        print(f"\nStatus file '{nama_file}':")
        print(f"Pemilik: {stat_info.st_uid}")
        print(f"Waktu modifikasi terakhir: {datetime.
                                    fromtimestamp(
                                    stat_info.st_mtime
                                    )}")
        print(f"Izin akses: {stat.filemode(stat_info.
                                    st_mode)}")
    except FileNotFoundError:
        print(f"File '{nama_file}' tidak ditemukan.")
    except Exception as e:
        print(f"Gagal menampilkan status file: {e}")

def main():
    while True:
        print("\nPilih perintah:")
        print("1. Baca file")
        print("2. Tambah teks ke file")
```

```
            print("3. Tampilkan status file")
            print("4. Keluar")

            pilihan = input("Masukkan pilihan (1-4): ")

            if pilihan == '1':
                nama_file = input("Masukkan nama file yang
                                   akan dibaca: "
                                   )
                baca_file(nama_file)
            elif pilihan == '2':
                nama_file = input("Masukkan nama file yang
                                   akan
                                   ditambahkan
                                   teks: ")
                teks = input("Masukkan teks yang ingin
                                   ditambahkan: "
                                   )
                tambah_teks(nama_file, teks)
            elif pilihan == '3':
                nama_file = input("Masukkan nama file untuk
                                   menampilkan
                                   status: ")
                status_file(nama_file)
            elif pilihan == '4':
                print("Keluar dari program.")
                break
            else:
                print("Pilihan tidak valid. Silakan coba lagi
                                   .")

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

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] *Vaia. CPU Performance: Improvement Influencing Factors.*

[5] *AWS. GPU vs CPU - Difference Between Processing Units.*

[12pt]article amsmath graphicx hyperref listings color pythonhighlight

Operating System Course Report - First Half of the Semester A class October 10, 2024

Contents

6 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.

7 Course Overview

7.1 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.

7.2 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

8 Topics Covered

8.1 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.

8.2 Performa Sistem dan Metrik Sistem Komputer

8.2.1 Performa Sistem

sistem komputer adalah kemampuan suatu sistem untuk menjalankan tugas-tugas komputasi sesuai dengan spesifikasi dan parameter yang telah ditetapkan. Performa ini mencakup kecepatan, efisiensi, dan ketepatan dalam menyelesaikan berbagai proses dan operasi yang diminta oleh pengguna atau aplikasi.

1. Faktor Pengaruh Performa
   (a) CPU (Central Processing Unit)
   CPU atau prosesor adalah otak dari komputer yang bertanggung jawab untuk menjalankan instruksi dan proses komputasi. Kecepatan CPU diukur dalam gigahertz (GHz); semakin tinggi frekuensi, semakin cepat proses eksekusi instruksi. Jumlah core pada CPU juga penting; semakin banyak core, semakin baik komputer dalam menangani multitasking dan aplikasi yang membutuhkan banyak sumber daya[4].

(b) GPU (Graphics Processing Unit)
GPU bertanggung jawab untuk menangani pemrosesan grafis dan rendering visual. Dalam aplikasi yang memerlukan performa grafis tinggi seperti game dan desain 3D, GPU yang kuat menjadi sangat penting. GPU modern juga digunakan untuk komputasi paralel di bidang kecerdasan buatan dan pembelajaran mesin, GPU dapat menyelesaikan tugas sederhana dan berulang jauh lebih cepat karena dapat memecah tugas menjadi komponen yang lebih kecil dan menyelesaikannya secara paralel[5].

(c) RAM (Random Access Memory)
RAM adalah memori sementara yang digunakan oleh sistem untuk menyimpan data yang aktif digunakan atau diakses oleh CPU dengan cepat. Jumlah dan kecepatan RAM biasanya meningkatkan performa sistem dengan memungkinkan lebih banyak data digunakan tanpa harus menggunakan penyimpanan yang lebih lambat[5].

2. Keterkaitan Hardware

(a) Keterkaitan antara RAM dan CPU
CPU bergantung pada RAM untuk menyimpan data sementara yang sedang diproses. Ketika CPU menjalankan tugas, ia membutuhkan data yang dapat diakses dengan cepat. RAM menyediakan ruang penyimpanan sementara yang memungkinkan CPU untuk mengakses data dengan kecepatan tinggi, yang membantu mencegah penundaan dalam pemrosesan. Semakin besar kapasitas dan kecepatan RAM, semakin cepat CPU dapat memproses data, terutama dalam situasi multitasking ketika banyak aplikasi berjalan secara bersamaan.

(b) Keterkaitan antara CPU dan GPU
CPU dan GPU bekerja bersama-sama untuk membagi tugas yang memerlukan pemrosesan berat. CPU menangani tugas-tugas umum seperti logika, kontrol aplikasi, dan aliran data, sementara GPU menangani tugas-tugas yang membutuhkan komputasi paralel, seperti rendering grafis atau komputasi numerik. Kinerja keseluruhan sistem meningkat ketika CPU dan GPU dapat bekerja secara seimbang. Jika salah satu komponen terlalu lambat dibandingkan dengan yang lain, bisa terjadi bottleneck,

ketika salah satu perangkat keras menahan kinerja perangkat lainnya.

(c) Keterkaitan antara Penyimpanan dan RAM
RAM dan penyimpanan juga bekerja sama untuk memastikan kelancaran operasi sistem. Ketika aplikasi atau data yang dibutuhkan oleh CPU tidak dapat seluruhnya ditampung di RAM, sistem akan menggunakan penyimpanan sebagai memori virtual. Jika penyimpanan yang digunakan adalah SSD (Solid State Drive), maka akses data dari penyimpanan ke RAM menjadi jauh lebih cepat dibandingkan dengan HDD (Hard Disk Drive) tradisional, sehingga mempercepat pemuatan aplikasi dan respons sistem secara keseluruhan.

8.2.2 Metrik Sistem Utama

1. Throughput

Throughput adalah jumlah output yang dapat diselesaikan oleh sistem dalam jangka waktu tertentu, misalnya jumlah permintaan yang dilayani oleh server dalam satu detik. Contoh Kasus: Server Web: Sebuah server web yang mampu melayani 200 permintaan HTTP per detik memiliki throughput sebesar 200 request per second. Jika throughput rendah, server mungkin membutuhkan peningkatan kapasitas. Optimasi: Throughput dapat ditingkatkan dengan mempercepat prosesor atau meningkatkan bandwidth jaringan untuk menangani lebih banyak permintaan secara simultan.

2.Latency Definisi: Latency adalah waktu yang diperlukan untuk menyelesaikan satu unit kerja dari awal hingga akhir, misalnya waktu tunggu antara saat data dikirim dan diterima. Contoh Kasus: Sistem Jaringan: Dalam jaringan, jika waktu yang dibutuhkan untuk mengirim paket data dari satu komputer ke komputer lain adalah 20 milidetik, ini mencerminkan tingkat latency. Latency yang tinggi dapat menyebabkan keterlambatan dalam komunikasi, yang dapat mengganggu aplikasi seperti video conference. Optimasi: Latency dapat dikurangi dengan menggunakan jalur komunikasi yang lebih cepat atau memperbaiki routing jaringan untuk menghindari keterlambatan.

Utilization Definisi: Utilization mengukur tingkat penggunaan sumber daya sistem, seperti CPU atau memori, dan biasanya dinyatakan dalam persentase. Contoh: Server Database: Jika sebuah server database menunjukkan CPU utilization sebesar 90sangat keras, yang bisa menjadi tanda overutilization. Overutilization dapat menyebabkan penurunan performa keseluruhan sistem. Optimasi: Utilization dapat dikurangi dengan menambahkan lebih banyak CPU, menyeimbangkan beban kerja di antara server lain, atau mengoptimalkan kode aplikasi untuk efisiensi yang lebih tinggi.

Response time Definisi: Response time adalah waktu yang diperlukan oleh sistem untuk merespons permintaan dari pengguna, misalnya waktu yang diperlukan untuk memproses pembayaran dalam aplikasi e-commerce.

8.2.3 Metrik Sistem Spesifik

1. CPI

   CPI adalah salah satu metrik utama yang digunakan untuk mengevaluasi performa CPU (Central Processing Unit). CPI mengukur rata-rata jumlah siklus clock yang dibutuhkan oleh CPU untuk mengeksekusi satu instruksi.

   Formula umum CPI:

   CPI = Total Cycle Total/Instructions

   Rendahnya CPI berarti CPU dapat mengeksekusi instruksi lebih efisien, menunjukkan performa yang lebih baik sedangkan tingginya CPI menunjukkan bahwa CPU memerlukan lebih banyak siklus clock untuk mengeksekusi instruksi, yang bisa berarti instruksi tersebut lebih kompleks ata ada masalah dalam pipeline CPU.

   Contoh Penggunaan CPI

   Misalkan kamu memiliki dua CPU:

   CPU A memiliki CPI 1.2

   CPU B memiliki CPI 1.8

   Jika keduanya berjalan pada frekuensi yang sama, katakanlah 3 GHz, CPU A akan lebih efisien karena membutuhkan lebih sedikit siklus per instruksi. Ini berarti CPU A dapat menyelesaikan lebih banyak instruksi dalam waktu yang sama dibandingkan CPU B.

2. Floating Point Operations Per Second (FLOPS)

   FLOPS adalah metrik yang digunakan untuk mengukur kemampuan komputasi floating-point dari sebuah komputer, yang sangat penting dalam aplikasi yang memerlukan kalkulasi numerik berat, seperti simulasi ilmiah, pemodelan 3D, atau machine learning.

   Contoh Penggunaan FLOPS:

   Bayangkan kamu bekerja di bidang simulasi ilmiah yang memerlukan perhitungan numerik yang intensif. Sistem A memiliki kapasitas 2 TFLOPS, sementara Sistem B memiliki 5 TFLOPS. Sistem B akan mampu menyelesaikan simulasi tersebut lebih cepat, karena dapat melakukan lebih banyak operasi floating-point per detik.

3. Input/Output Operations Per Second (IOPS)

   IOPS mengukur performa dari perangkat penyimpanan (seperti SSD, HDD) dalam hal jumlah operasi input/output yang dapat diproses dalam satu detik. IOPS sering digunakan untuk mengevaluasi performa disk dan sistem penyimpanan.

   Contoh Penggunaan IOPS:

   Misalkan kamu mengelola server database. SSD A memiliki 100,000 IOPS, sementara SSD B memiliki 50,000 IOPS. SSD A akan mampu menangani lebih banyak operasi baca/tulis per detik, sehingga lebih cocok untuk lingkungan database yang memerlukan akses data cepat dan simultan.

8.2.4 Pengukuran dan Optimasi Performa

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.

   (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. 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 2 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

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. *Tools* yang digunakan yaitu gprof dan Perf.
- Memory Profiling: Mengukur alokasi memori dan menemukan kebocoran memori. *Tools* yang digunakan yaitu Valgrind dan Heap Profiler.
- I/O Profiling: Menganalisis performa operasi input/output seperti file atau jaringan. *Tools* yang digunakan yaitu IOTop dan dstat.
- Function-Level Profiling: Menganalisis fungsi dalam aplikasi, melihat frekuensi pemanggilan dan durasi. *Tools* yang digunakan yaitu Xdebug (PHP) dan py-spy (Python).

(c) Monitoring

Monitoring adalah proses pengamatan dan pengukuran performa sistem secara real-time dan berkelanjutan untuk memas-

tikan sistem berjalan optimal serta mendeteksi masalah atau potensi gangguan. Monitoring sangat penting untuk menjaga stabilitas dan performa sistem komputer atau aplikasi, terutama dalam lingkungan produksi, di mana uptime dan keandalan menjadi prioritas. Ada beberapa teknik yang sering digunakan dalam monitoring:

- 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 atau alarm 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.

2. Strategi Optimasi Performa

   (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.

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 2: Perbedaan dari Benchmarking, Profiling, dan Monitoring

- Jaringan lebih Ccpat: 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 diterap-

kan pada sistem yang berbasis *server* untuk aplikasi *web*, basis data, atau layanan *cloud*.

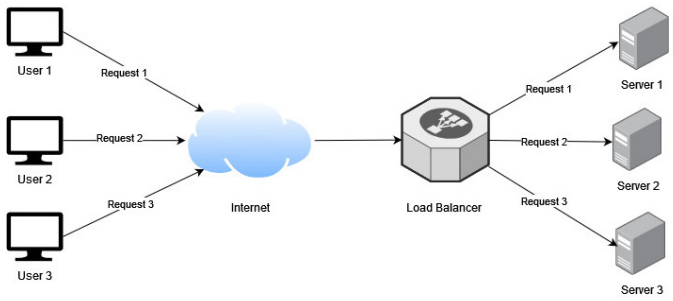

Figure 3: Ilustrasi proses *load balancing*

Pada gambar 3, 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.

8.3 System Architecture of Computer Systems

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

8.4 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

8.5 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.

8.6 Process Creation and Termination

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

- Process spawning
- Process termination conditions

8.7 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

Seperti yang terlihat pada Gambar 4, inilah cara menambahkan gambar dengan keterangan.

8.8 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

Figure 4: Ini adalah gambar contoh dari multithreading.

8.9 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

8.10 Deadlock Introduction and Prevention

Explores the concept of deadlocks and methods for preventing them:

- Deadlock conditions
- Deadlock prevention techniques

8.11 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

8.12 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

9 Assignments and Practical Work

9.1 Assignment 1: Process Scheduling

9.1.1 Group 2

Soal:

Buatlah sebuah program Python sederhana yang dapat mensimulasikan algoritma penjadwalan proses dengan menggunakan Round Robin Scheduling. Program ini harus mendukung beberapa fitur berikut:

- Menerima daftar proses dengan burst time masing-masing.
- Mengatur proses dengan kuantum waktu tetap.
- Menampilkan urutan eksekusi proses serta waktu selesai masing-masing.
- Hitung waktu rata-rata turnaround time dan waiting time.

Jawaban: Untuk menyelesaikan soal ini, berikut adalah implementasi dari algoritma Round Robin Scheduling sederhana dalam Python:

```
python
Copy code
    while True:
        done = True
        for i in range(n):
            if remaining_time[i] > 0:
                done = False  # Ada proses yang belum
                                                selesai
                if remaining_time[i] > quantum:
                    t += quantum
                    remaining_time[i] -= quantum
                else:
                    t += remaining_time[i]
                    waiting_time[i] = t - burst_time[i]
                    remaining_time[i] = 0
        if done:
            break

    # Hitung turnaround time
    for i in range(n):
        turnaround_time[i] = burst_time[i] + waiting_time[i
                                            ]

    print("Proses\tBurst Time\tWaiting Time\tTurnaround
                                        Time")
    for i in range(n):
        print(f"{processes[i]}\t{burst_time[i]}\t\t{
                                        waiting_time[i]}\t\t
                                        {turnaround_time[i]}
                                        ")

    print(f"\nRata-rata Waiting Time: {sum(waiting_time) /
                                        n:.2f}")
    print(f"Rata-rata Turnaround Time: {sum(turnaround_time
                                        ) / n:.2f}")

if __name__ == "__main__":
    processes = ['P1', 'P2', 'P3']
    burst_time = [24, 3, 3]
    quantum = 4
    round_robin(processes, burst_time, quantum)
```

9.2 Assignment 2: Deadlock Handling

9.2.1 Grup 2

Soal:

Buatlah sebuah program Python sederhana yang mensimulasikan beberapa skenario deadlock dan mengeksplorasi berbagai metode pencegahan deadlock. Program ini harus mendukung beberapa fitur berikut:

- Simulasi skenario deadlock dengan penggunaan dua atau lebih sumber daya oleh beberapa proses.
- Implementasi metode pencegahan deadlock seperti pengurutan sumber daya, batasan jumlah maksimum sumber daya yang dapat dialokasikan, atau penggunaan algoritma Banker's.
- Visualisasi atau laporan mengenai status deadlock dan bagaimana pencegahannya dapat diterapkan.
- Tampilkan hasil dari skenario deadlock yang disimulasikan dan solusi pencegahannya.

Jawaban:

Untuk menjawab soal ini, kita akan membuat program Python sederhana yang mensimulasikan skenario deadlock serta menerapkan metode pencegahannya. Berikut adalah implementasi dari program tersebut:

```
import threading
import time

# Simulasi sumber daya
class SumberDaya:
    def __init__(self, nama):
        self.nama = nama
        self.lock = threading.Lock()

def gunakan(self, pengguna):
    with self.lock:
        print(f"{pengguna} menggunakan {self.nama}.")
        time.sleep(1)  # Simulasi waktu penggunaan
                                    sumber daya

# Fungsi untuk simulasi deadlock
```

```
    def deadlock_scenario(sumber1, sumber2, pengguna):
        print(f"{pengguna} mencoba mengakses {sumber1.nama}
                                            .")
        sumber1.gunakan(pengguna)

        print(f"{pengguna} mencoba mengakses {sumber2.nama}
                                            .")
        sumber2.gunakan(pengguna)

    # Pencegahan deadlock: mengatur urutan pengaksesan
                                        sumber daya
    def no_deadlock_scenario(sumber1, sumber2, pengguna):
        sumber_list = sorted([sumber1, sumber2], key=lambda
                                             s: s.nama)

        print(f"{pengguna} mencoba mengakses {sumber_list[0
                                            ].nama}.")
        sumber_list[0].gunakan(pengguna)

        print(f"{pengguna} mencoba mengakses {sumber_list[1
                                            ].nama}.")
        sumber_list[1].gunakan(pengguna)

    def main():
        # Membuat sumber daya
        sumber_a = SumberDaya('SumberA')
        sumber_b = SumberDaya('SumberB')

    # Simulasi deadlock
    thread1 = threading.Thread(target=deadlock_scenario,
                                        args=(sumber_a, sumber_b
                                        , "Proses1"))
    thread2 = threading.Thread(target=deadlock_scenario,
                                        args=(sumber_b, sumber_a
                                        , "Proses2"))

    print("\nSimulasi Deadlock:")
    thread1.start()
    thread2.start()

    thread1.join()
    thread2.join()

    # Simulasi tanpa deadlock dengan pengurutan akses
                                        sumber daya
```

```
    print("\nSimulasi Pencegahan Deadlock:")
    thread3 = threading.Thread(target=no_deadlock_scenario,
                                    args=(sumber_a,
                                   sumber_b, "Proses1"))
    thread4 = threading.Thread(target=no_deadlock_scenario,
                                    args=(sumber_b,
                                   sumber_a, "Proses2"))

    thread3.start()
    thread4.start()

    thread3.join()
    thread4.join()

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

9.3 Assignment 3: Multithreading and Amdahl's Law

9.3.1 Grup 2

Soal:

Buatlah sebuah program Python yang mensimulasikan skenario multithreading untuk menyelesaikan masalah komputasi intensif. Setelah itu, terapkan Hukum Amdahl untuk menghitung kecepatan teoretis program tersebut ketika jumlah thread ditingkatkan. Program harus memenuhi beberapa kriteria berikut:

- Selesaikan masalah komputasi intensif (misalnya, penjumlahan bilangan dalam rentang besar) menggunakan beberapa thread.
- Hitung waktu eksekusi untuk setiap jumlah thread yang digunakan.
- Terapkan Hukum Amdahl untuk menghitung kecepatan teoretis ketika menambah jumlah thread.
- Bandingkan hasil teoritis dan hasil nyata dari waktu eksekusi program.

Jawaban:

Berikut adalah implementasi program yang menggunakan multithreading untuk menyelesaikan masalah komputasi intensif, diikuti dengan perhitungan kecepatan teoretis menggunakan Hukum Amdahl:

```
        import threading
import time
import os

# Fungsi komputasi intensif: penjumlahan bilangan dari 1
                                hingga N
def komputasi_intensif(n, hasil, index):
    total = 0
    for i in range(1, n+1):
        total += i
    hasil[index] = total
    print(f"Thread-{index+1} selesai, hasil: {total}")

# Fungsi untuk menghitung waktu eksekusi menggunakan
                                multithreading
def hitung_waktu_multithreading(n, jumlah_thread):
    hasil = [0] * jumlah_thread
    threads = []
    bagian = n // jumlah_thread

    start_time = time.time()

    for i in range(jumlah_thread):
        batas_atas = (i+1) * bagian if i != jumlah_thread -
                                        1 else n
        t = threading.Thread(target=komputasi_intensif,
                                    args=(batas_atas,
                                    hasil, i))
        threads.append(t)
        t.start()

    for t in threads:
        t.join()

    end_time = time.time()

    total_hasil = sum(hasil)
    waktu_eksekusi = end_time - start_time
    print(f"Total hasil: {total_hasil}, Waktu eksekusi
                                dengan {jumlah_thread}
                                thread: {waktu_eksekusi
```

```
                                    :.4f} detik")
    return waktu_eksekusi

# Hukum Amdahl: menghitung kecepatan teoretis
def hitung_kecepatan_teoretis(serial_fraction,
                              jumlah_thread):
    return 1 / (serial_fraction + (1 - serial_fraction) /
                                    jumlah_thread)

def main():
    n = 10**7  # Besar masalah (penjumlahan hingga N)
    jumlah_threads = [1, 2, 4, 8]  # Daftar jumlah thread
                                    yang akan diuji
    serial_fraction = 0.1  # Asumsi fraksi yang tidak bisa
                                    diparalelkan (10%)

    waktu_eksekusi_awal = None

    # Bandingkan waktu eksekusi dengan jumlah thread yang
                                    berbeda
    for thread_count in jumlah_threads:
        waktu_eksekusi = hitung_waktu_multithreading(n,
                                        thread_count)

        if thread_count == 1:
            waktu_eksekusi_awal = waktu_eksekusi  # Simpan
                                        waktu eksekusi
                                        serial (1 thread
                                        )

        # Hitung kecepatan teoretis menggunakan Hukum
                                        Amdahl
        kecepatan_teoretis = hitung_kecepatan_teoretis(
                                        serial_fraction,
                                        thread_count)
        print(f"Kecepatan teoretis dengan {thread_count}
                                        thread: {
                                        kecepatan_teoretis:.
                                        4f}x")

        # Hitung kecepatan nyata berdasarkan hasil eksekusi
        kecepatan_nyata = waktu_eksekusi_awal /
                                        waktu_eksekusi
        print(f"Kecepatan nyata dengan {thread_count}
                                        thread: {
```

```
                                                kecepatan_nyata:.4f}
                                                x\n")

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

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

9.4.1 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 buat_file(nama_file):
        try:
```

```
            with open(nama_file, 'w') as f:
                f.write('')  # Membuat file kosong
            print(f"File '{nama_file}' berhasil dibuat.")
        except Exception as e:
            print(f"Gagal membuat file: {e}")

    def hapus_file(nama_file):
        try:
            os.remove(nama_file)
            print(f"File '{nama_file}' berhasil dihapus.")
        except FileNotFoundError:
            print(f"File '{nama_file}' tidak ditemukan.")
        except Exception as e:
            print(f"Gagal menghapus file: {e}")

    def tampilkan_daftar_file():
        try:
            files = os.listdir('.')
            if files:
                print("Daftar file di direktori saat ini:")
                for file in files:
                    print(file)
            else:
                print("Tidak ada file di direktori saat ini
                                                    .")
        except Exception as e:
            print(f"Gagal menampilkan daftar file: {e}")

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

    def tampilkan_status_sistem():
        try:
            cpu_percent = psutil.cpu_percent(interval=1)
```

```
            memory = psutil.virtual_memory()
            print(f"Penggunaan CPU: {cpu_percent}%")
            print(f"Total Memori: {memory.total / (1024 **
                                        3):.2f} GB")
            print(f"Memori yang Digunakan: {memory.used / (
                                        1024 ** 3):.2f}
                                        GB ({memory.
                                        percent}%)")
        except Exception as e:
            print(f"Gagal menampilkan status sistem: {e}")

    def main():
        while True:
            print("\nPilih perintah:")
            print("1. Buat file")
            print("2. Tampilkan daftar file")
            print("3. Hapus file")
            print("4. Tampilkan daftar proses")
            print("5. Tampilkan status sistem")
            print("6. Keluar")

            pilihan = input("Masukkan pilihan (1-6): ")

            if pilihan == '1':
                nama_file = input("Masukkan nama file yang
                                        akan dibuat:
                                        ")
                buat_file(nama_file)
            elif pilihan == '2':
                tampilkan_daftar_file()
            elif pilihan == '3':
                nama_file = input("Masukkan nama file yang
                                        akan dihapus
                                        : ")
                hapus_file(nama_file)
            elif pilihan == '4':
                tampilkan_daftar_proses()
            elif pilihan == '5':
                tampilkan_status_sistem()
            elif pilihan == '6':
                print("Keluar dari program.")
                break
            else:
                print("Pilihan tidak valid. Silakan coba
                                        lagi.")
```

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

9.5 Assignment 5: File System Access

9.5.1 Group 2

Soal:

Buatlah sebuah program Python sederhana untuk melakukan akses sistem file. Program ini harus mendukung beberapa perintah berikut:

- Membaca isi file teks.
- Menambahkan teks ke dalam file yang sudah ada.
- Menampilkan status akses file, seperti siapa pemiliknya, waktu modifikasi terakhir, dan izin akses file.

Jawaban: Berikut adalah implementasi dari program Python yang dapat mengakses sistem file dan mendukung fitur-fitur yang diminta:

```
python
Copy code
def baca_file(nama_file):
    try:
        with open(nama_file, 'r') as f:
            print(f"\nIsi file '{nama_file}':")
            print(f.read())
    except FileNotFoundError:
        print(f"File '{nama_file}' tidak ditemukan.")
    except Exception as e:
        print(f"Gagal membaca file: {e}")

def tambah_teks(nama_file, teks):
    try:
        with open(nama_file, 'a') as f:
            f.write(teks + '\n')
        print(f"Teks berhasil ditambahkan ke file '{
                                        nama_file}'.")
    except Exception as e:
        print(f"Gagal menambahkan teks: {e}")
```

```
def status_file(nama_file):
    try:
        stat_info = os.stat(nama_file)
        print(f"\nStatus file '{nama_file}':")
        print(f"Pemilik: {stat_info.st_uid}")
        print(f"Waktu modifikasi terakhir: {datetime.
                                    fromtimestamp(
                                    stat_info.
                                    st_mtime)}")
        print(f"Izin akses: {stat.filemode(stat_info.
                                    st_mode)}")
    except FileNotFoundError:
        print(f"File '{nama_file}' tidak ditemukan.")
    except Exception as e:
        print(f"Gagal menampilkan status file: {e}")

def main():
    while True:
        print("\nPilih perintah:")
        print("1. Baca file")
        print("2. Tambah teks ke file")
        print("3. Tampilkan status file")
        print("4. Keluar")

        pilihan = input("Masukkan pilihan (1-4): ")

        if pilihan == '1':
            nama_file = input("Masukkan nama file yang
                                    akan dibaca:
                                    ")
            baca_file(nama_file)
        elif pilihan == '2':
            nama_file = input("Masukkan nama file yang
                                    akan
                                    ditambahkan
                                    teks: ")
            teks = input("Masukkan teks yang ingin
                                    ditambahkan:
                                    ")
            tambah_teks(nama_file, teks)
        elif pilihan == '3':
            nama_file = input("Masukkan nama file untuk
                                    menampilkan
                                    status: ")
```

```
                status_file(nama_file)
            elif pilihan == '4':
                print("Keluar dari program.")
                break
            else:
                print("Pilihan tidak valid. Silakan coba
                                                lagi.")

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

10 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://bhatabhishek-ylp.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] *Vaia. CPU Performance: Improvement Influencing Factors.*

[5] *AWS. GPU vs CPU - Difference Between Processing Units.*

[6] *Algor Cards. CPU Performance and Factors Affecting It.*