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 System Performance and Metrics

This section introduces various system performance metrics used to measure the efficiency of a computer system, including throughput, response time, and utilization.

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

3.6.1 *Process Creation*

(Isi materi disini)

3.6.2 Siapa saja yang membuat proses

Proses dalam sistem operasi diciptakan oleh beberapa entitas, baik oleh pengguna, sistem operasi itu sendiri, atau proses lain. Setiap entitas memainkan peran penting dalam memastikan bahwa tugas yang dijalankan oleh komputer dilakukan dengan efektif. Berikut adalah penjelasan mendetail tentang setiap entitas yang dapat menciptakan proses.

- *Process spawning* Pembuatan proses baru oleh entitas yang ada (pengguna, sistem operasi, atau proses lain) dengan cara menyalin atau memodifikasi proses yang sudah ada (Silberschatz, Galvin, & Gagne, 2018).
- *Process termination conditions* Kondisi yang menyebabkan sebuah proses dihentikan, bisa karena proses telah selesai, terjadi kesalahan, atau dihentikan secara manual oleh entitas yang membuatnya (Silberschatz, Galvin, & Gagne, 2018).

Pengguna sebagai Pencipta Proses Pengguna adalah salah satu entitas yang paling sering menciptakan proses di dalam sistem operasi. Setiap kali pengguna membuka sebuah program atau aplikasi, sebuah proses baru diciptakan untuk menjalankan program tersebut.

- Ketika pengguna membuka aplikasi seperti *browser*, gim, atau pemutar video, sebuah proses baru dibuat oleh sistem operasi. Proses ini akan dijalankan oleh CPU untuk mengerjakan tugas-tugas aplikasi tersebut.
- Setiap kali Anda membuka *Google Chrome*, sistem operasi menciptakan sebuah proses baru. Jika Anda membuka lebih dari satu tab, setiap tab mungkin dibuat sebagai proses terpisah untuk meningkatkan stabilitas dan performa.
- Sistem operasi akan mengalokasikan sumber daya yang dibutuhkan proses, seperti memori, waktu CPU, dan akses ke perangkat keras lain seperti jaringan dan penyimpanan.

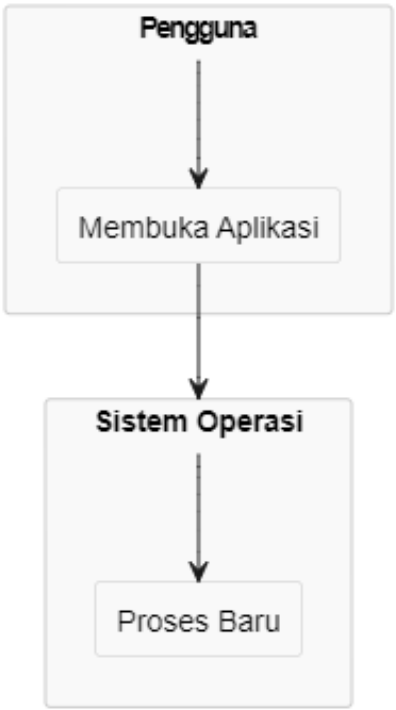

Figure 1: Pengguna menciptakan proses baru ketika membuka aplikasi.

Pengguna adalah entitas langsung yang memulai proses ketika berinteraksi dengan perangkat lunak. Setiap aplikasi yang dijalankan pengguna akan mengarah pada pembuatan proses baru yang membutuhkan manajemen sumber daya oleh sistem operasi.

Sistem Operasi sebagai Pencipta Proses Sistem operasi juga bertindak sebagai pencipta proses, terutama untuk proses-proses latar belakang yang diperlukan untuk menjaga komputer berjalan dengan baik. Proses-proses ini sering kali tidak terlihat oleh pengguna tetapi tetap penting untuk fungsionalitas dasar sistem.

- Sistem operasi menciptakan proses-proses yang menjalankan tugas penting seperti pengelolaan jaringan, penjadwalan tugas, dan pengelolaan file. Tugas-tugas ini dilakukan tanpa memerlukan interaksi langsung dari pengguna.
- Beberapa proses latar belakang, yang dikenal sebagai *daemon*, selalu berjalan untuk menangani tugas-tugas seperti pengelolaan memori atau pengaturan koneksi jaringan.
- Sistem operasi secara otomatis menangani kapan proses-proses ini dibuat, diatur, dan diakhiri, berdasarkan kebutuhan sistem dan kondisi saat ini.

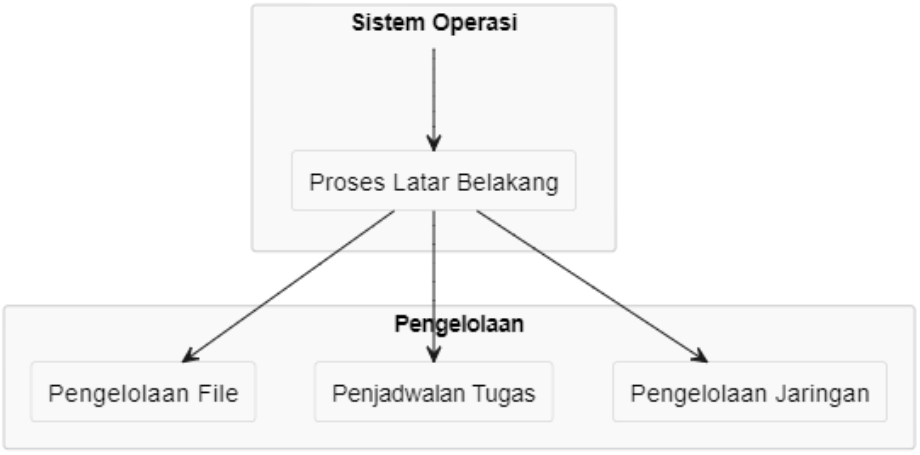

Figure 2: Sistem operasi menciptakan proses latar belakang untuk menjaga kestabilan sistem.

Proses yang diciptakan oleh sistem operasi tidak membutuhkan interaksi pengguna secara langsung, namun sangat penting untuk menjaga stabilitas dan efisiensi komputer. Contoh dari proses-proses ini termasuk manajemen memori, keamanan sistem, dan pengelolaan perangkat keras.

Proses Lain sebagai Pencipta Proses (*Parent-Child Relationship*) Selain pengguna dan sistem operasi, sebuah proses juga dapat menciptakan proses lain, yang dikenal sebagai *spawning*. Dalam sistem operasi, sebuah proses induk (*parent process*) dapat menciptakan proses anak (*child process*) untuk menjalankan tugas-tugas tertentu secara paralel atau terpisah.

- *Proses Parent-Child* Ketika proses induk menciptakan proses anak, keduanya dapat berjalan secara bersamaan. Proses anak dapat mewarisi sebagian besar sumber daya dari proses induk, namun tetap dapat berjalan secara independen.
- *Pembuatan Sub-Proses* Proses induk mungkin menciptakan sub-proses untuk menjalankan bagian-bagian spesifik dari suatu program, misalnya, dalam pengelolaan tab di *web browser*.
- **Contoh :** *Google Chrome* membuat proses terpisah untuk setiap tab yang dibuka oleh pengguna. Setiap tab berfungsi sebagai proses anak dari proses utama *Chrome*, yang mengelola seluruh operasi aplikasi.

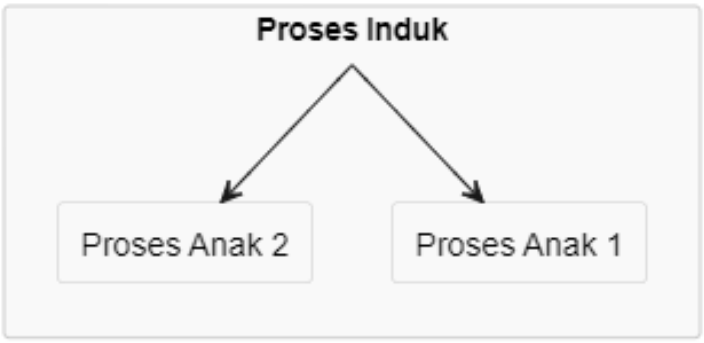

Figure 3: Proses induk menciptakan proses anak.

Penciptaan proses oleh proses lain memungkinkan multitasking dan pembagian beban kerja dalam sistem operasi. Ini penting dalam aplikasi yang membutuhkan banyak tugas secara bersamaan tanpa mengganggu tugas utama.

Scheduler sebagai Pengelola Proses Meskipun bukan entitas yang secara langsung menciptakan proses, *scheduler* dalam sistem operasi memainkan peran penting dalam mengalokasikan waktu CPU kepada setiap proses yang sedang berjalan (Stallings et al., 2017).

- *Pembagian Waktu CPU Scheduler* memastikan bahwa setiap proses mendapatkan giliran yang adil untuk menggunakan CPU, terutama ketika banyak proses berjalan bersamaan.
- *Manajemen Prioritas* Proses dengan prioritas lebih tinggi, seperti proses sistem, dapat diberikan waktu CPU lebih sering dibandingkan proses dengan prioritas lebih rendah, seperti aplikasi yang tidak aktif.
- *Pengelolaan Multitasking* Dengan bantuan *scheduler*, sistem operasi dapat menjalankan banyak aplikasi secara bersamaan tanpa menurunkan performa keseluruhan.

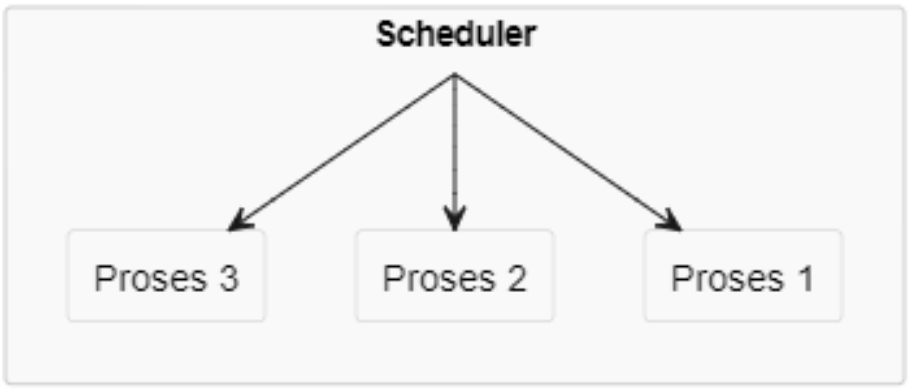

Figure 4: *Scheduler* mengelola alokasi waktu CPU untuk setiap proses.

Meskipun *scheduler* tidak menciptakan proses, ia berperan penting dalam pengelolaan proses, memastikan bahwa semua proses berjalan dengan efisien dan sistem tetap responsif terhadap tugas-tugas pengguna.

3.6.3 Proses *Termination*

- Definisi Proses *Termination* (isi materi disini)
- Jenis-jenis *Termination* (Isi materi disini)
- Apa yang Terjadi Setelah *Termination* (isi materi disini)

References

[1] Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* (10th ed.). Wiley.

[2] Stallings, W. (2017). *Operating Systems: Internals and Design Principles* (9th ed.). Pearson.

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

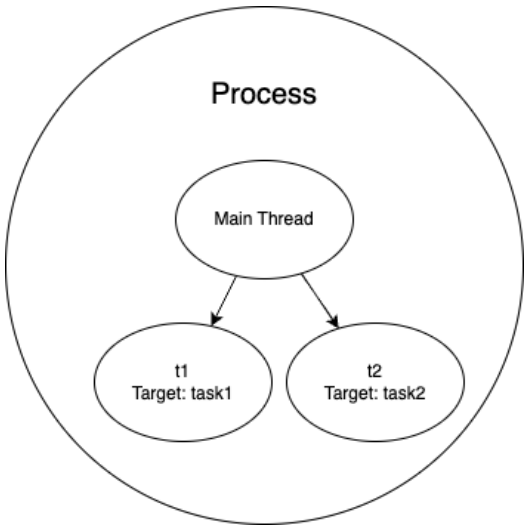

Figure 5: Ini adalah gambar contoh dari multithreading.

Seperti yang terlihat pada Gambar 5, 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 1: 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.

4.3.1 Group 6

Ragilnata, seorang mahasiswa dengan kulit gelap yang suka bercanda, sedang duduk di ruang lab sambil melihat *Instagram* ketika dosennya, Pak Abdul, tiba-tiba memberi tugas mendadak tentang *multithreading*.

"Eh Ragilnata, sudah saatnya kamu mengerjakan tugas daripada hanya melihat media sosial terus," kata Pak Abdul dengan nada marah.

Ragilnata diminta untuk membuat program yang memproses data gambar berukuran besar untuk mendeteksi objek tertentu (misalnya, pisang). Mengingat gambar-gambar ini cukup besar dan berat secara komputasi, Ragilnata memutuskan untuk menggunakan *multithreading* agar dapat menyelesaikan tugas ini dengan lebih cepat, karena dia tidak ingin mengerjakan ini sendirian semalaman.

Total waktu yang diperlukan untuk menyelesaikan proses deteksi objek tanpa *multithreading* adalah sepuluh detik. Dari keseluruhan proses, sekitar 70% dari total eksekusi dapat diparalelkan, sedangkan sisanya harus dieksekusi secara serial karena batasan dari algoritma deteksi objeknya.

Dengan informasi ini, Ragilnata mencoba menghitung berapa kecepatan (*speedup*) yang dapat dicapai berdasarkan **Amdahls Law** jika ia menggunakan empat *thread*. Sebelum mulai melakukan pengkodean, Ragilnata berpikir, "Apakah cukup menggunakan empat *thread*? Atau cukup dua saja agar lebih hemat sumber daya?"

Pertanyaan:

1. Hitung berapa *speedup* (S) yang bisa Ragilnata capai jika dia menggunakan empat *thread*.

2. Apakah hasil *speedup* tersebut signifikan dibandingkan dengan jika Ragilnata hanya menggunakan dua *thread*? Hitung juga *speedup* untuk dua *thread* dan bandingkan hasilnya.

3. Menurut kamu, berapa jumlah *thread* yang sebaiknya digunakan oleh Ragilnata, dan kenapa?

Jawaban:

1. **Speedup dengan 4 thread**
   Amdahl's Law:
   $$S = \frac{1}{(1-P) + \frac{P}{N}}$$
   Di mana:

   P = proporsi bagian yang bisa diparalelkan = 0.7
   N = jumlah thread = 4

   Hasil perhitungan menggunakan Python

```python
# Definisikan variabel
P = 0.7
N = 4

# Fungsi untuk menghitung speedup berdasarkan
                                    Amdahl's Law
 def amdahl_speedup(P, N):
    return 1 / ((1 - P) + (P / N))

# Hitung speedup
speedup = amdahl_speedup(P, N)
speedup

#Hasil : 2.1052631578947367
```

   Jadi, speedup dengan 4 thread adalah sekitar **2.105 kali** lebih cepat.

2. **Speedup dengan 2 thread**

```python
# Update jumlah thread menjadi 2
N = 2

# Hitung speedup dengan 2 thread
speedup_with_2_threads = amdahl_speedup(P, N)
speedup_with_2_threads

#Hasil : 1.5384615384615383
```

   Jadi, speedup dengan 2 thread adalah sekitar **1.538 kali** lebih cepat.
   Speedup dengan 4 thread (2.105) lebih signifikan dibandingkan dengan

2 thread (1.538). Artinya, menggunakan 4 thread lebih efisien dalam hal kecepatan.

3. **Jumlah thread optimal**
   Jumlah thread bergantung pada resource yang tersedia. Jika Ragilnata punya resource yang cukup (seperti CPU dengan core yang banyak), maka menggunakan 4 thread lebih baik karena speedup-nya lebih signifikan. Namun, jika resource terbatas, 2 thread juga sudah memberikan peningkatan yang lumayan tanpa membebani sistem.

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.