Operating System Course Report - First Half of the Semester

B class

October 8, 2024

Contents

1 Introduction

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

2 Course Overview

2.1 Objectives

The main objectives of this course are:

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

2.2 Course Structure

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

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

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

3 Topics Covered

3.1 Basic Concepts and Components of Computer Systems

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

3.2 System Performance and Metrics

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

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

- Process spawning
- Process termination conditions

3.6.1 Terminasi Normal Proses

> Proses terminasi normal adalah kondisi di mana suatu proses menyelesaikan fungsinya dengan baik tanpa mengalami kesalahan atau pengecualian. Ini adalah akhir eksekusi proses yang terkontrol dan disengaja. Dalam kode program, biasanya terdapat instruksi atau pernyataan yang menunjukkan akhir dari proses, seperti *'return'* di fungsi utama dalam bahasa C atau Java. Proses ini merupakan bagian dari siklus hidup normal sebuah proses.

Skenario umum untuk terminasi normal:

- Penyelesaian proses: Proses telah menyelesaikan semua tugasnya dan mencapai titik akhir secara alami.
- Terminasi yang diinisiasi oleh pengguna: Pengguna secara eksplisit meminta proses untuk berhenti, biasanya melalui perintah atau antarmuka.
- Pengembalian dari fungsi: Sebuah proses dapat berhenti ketika sebuah fungsi yang dijalankan mengembalikan nilai yang menandakan penyelesaian.

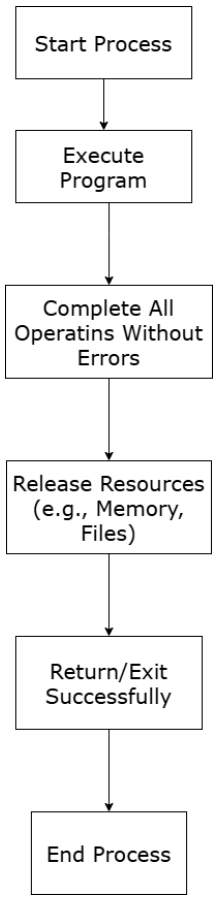

Figure 1: Terminasi Normal

Karakteristik terminasi normal:

- Proses berakhir secara teratur dan dapat diprediksi.
- Tidak ada kesalahan atau pengecualian yang tidak terduga yang menyebabkan proses berhenti sebelum waktunya.
- Proses melepaskan semua sumber daya yang digunakannya (misalnya, memori, file, koneksi jaringan) sebelum terminasi.

Contoh terminasi normal:

- Sebuah proses peramban web menyelesaikan pemuatan halaman web.
- Sebuah proses editor teks menyimpan dokumen dan kemudian keluar.
- Sebuah tugas latar belakang menyelesaikan pekerjaan yang dijadwalkan dan berakhir.

3.6.2 Terminasi Tidak Normal Proses

Terminasi tidak normal terjadi ketika sebuah proses berhenti secara tidak teratur akibat adanya kesalahan yang tidak dapat ditangani dengan baik. Hal ini sering kali disebabkan oleh kondisi tak terduga yang membuat proses tidak dapat melanjutkan eksekusi.

Penyebab terminasi tidak normal:

- Kesalahan logika: Kesalahan dalam kode program, seperti pembagian dengan nol, akses array di luar batas, atau penggunaan pointer yang tidak valid.
- Kehabisan sumber daya: Proses kehabisan sumber daya yang dibutuhkan untuk beroperasi, seperti memori, deskriptor file, atau waktu CPU.
- Sinyal: Proses menerima sinyal yang memaksanya untuk berhenti, seperti sinyal *SIGKILL* (berhenti segera) atau *SIGSEGV* (kesalahan segmentasi).
- Kesalahan perangkat keras: Kerusakan perangkat keras dapat menyebabkan proses berhenti secara tiba-tiba.

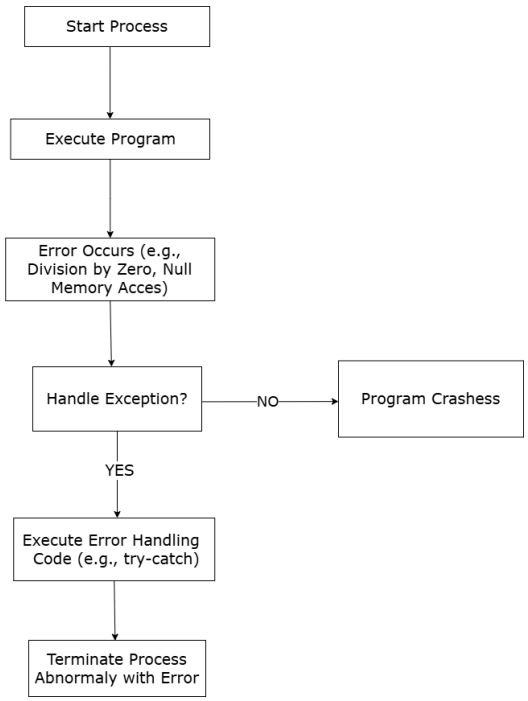

Figure 2: Terminasi Tidak Normal

- *Deadlock*: Dua atau lebih proses saling menunggu satu sama lain, sehingga tidak ada yang dapat melanjutkan eksekusi.
- Interupsi: Terjadinya interupsi yang tidak dapat ditangani oleh sistem operasi.

Kontras dengan terminasi tidak normal:

- Terminasi tidak normal terjadi ketika sebuah proses berhenti secara tak terduga karena kesalahan, pengecualian, atau keadaan tak terduga lainnya. Hal ini dapat menyebabkan perilaku tidak terkendali, kebocoran sumber daya, atau ketidakstabilan sistem.

Dampak terminasi tidak normal:

- Kehilangan data: Data yang sedang diproses oleh proses yang berhenti secara tidak normal mungkin hilang atau rusak.
- Ketidakstabilan sistem: Terminasi tidak normal dapat menyebabkan sistem menjadi tidak stabil atau bahkan *crash.*
- Kerusakan data: Data yang disimpan di disk atau memori dapat rusak jika proses tidak sempat menyimpan data dengan benar sebelum berhenti.
- Gangguan proses lain: Terminasi tidak normal suatu proses dapat mempengaruhi proses lain yang berinteraksi dengannya.

Mekanisme penanganan terminasi tidak normal:

Sistem operasi: Sistem operasi berusaha mendeteksi dan menangani terminasi tidak normal. Ini melibatkan mekanisme seperti:

- Penanganan pengecualian: Sistem operasi menangkap pengecualian yang terjadi selama eksekusi proses dan mengambil tindakan yang sesuai, seperti menghentikan proses atau menampilkan pesan kesalahan.
- Pengawasan memori: Sistem operasi memantau penggunaan memori oleh proses untuk mencegah akses memori yang tidak sah.
- Pengawasan waktu: Sistem operasi membatasi waktu eksekusi suatu proses untuk mencegah proses berjalan terlalu lama dan menghabiskan sumber daya sistem.

- *Debugging*: Pengembang dapat menggunakan debugger untuk melacak kesalahan yang menyebabkan terminasi tidak normal dan memperbaiki kode program.

Contoh terminasi tidak normal:

- Program *crash*: Sebuah program tiba-tiba berhenti bekerja dan menampilkan pesan kesalahan.
- *Blue Screen of Death (BSOD)*: Sistem operasi Windows mengalami kegagalan fatal dan menampilkan layar biru.
- *Kernel panic*: Sistem operasi Linux mengalami kegagalan yang sangat serius dan tidak dapat melanjutkan operasi.

Pencegahan terminasi tidak normal:

- Pengujian yang memadai: Melakukan pengujian menyeluruh pada perangkat lunak untuk menemukan dan memperbaiki *bug* sebelum di-deploy.
- Penanganan pengecualian: Menulis kode yang dapat menangani berbagai jenis pengecualian yang mungkin terjadi.
- Manajemen memori yang baik: Memastikan alokasi dan dealokasi memori dilakukan dengan benar untuk mencegah kebocoran memori.
- Validasi input: Memeriksa semua input pengguna untuk memastikan bahwa input tersebut valid dan tidak menyebabkan kesalahan.

- Wahab, A., Mubarok, R. (2019). Pengenalan Sistem Operasi (Edisi Revisi). Penerbit Andi.
- Hidayat, M., Santoso, A. (2018). "Analisis Penanganan Error dan Exception Handling Pada Bahasa Pemrograman Java." Jurnal Ilmu Komputer dan Teknologi Informasi, 5(1), 55-62.

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 3: Ini adalah gambar contoh dari multithreading.

Seperti yang terlihat pada Gambar 3, 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.

4.4 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

4.5.1 Group 6

- File creation and deletion

Tulis sebuah program Python yang membuat file bernama *example.txt*, menuliskan string *"Hello, World!"* ke dalam file tersebut, kemudian membaca isinya dan menghapus file tersebut.

```python
import os

# Membuat file dan menulis ke dalamnya
with open("example.txt", "w") as file:
    file.write("Hello, World!")

# Membaca isi file
with open("example.txt", "r") as file:
    content = file.read()

    print("Isi file:", content)

# Menghapus file
if os.path.exists("example.txt"):
    os.remove("example.txt")
    print("File berhasil dihapus.")
else:
    print("File tidak ditemukan.")
```

- Reading from and writing to files

Tulis sebuah program Python yang membaca data dari file input.txt dan menuliskan data tersebut ke file baru bernama output.txt.

```
# Membuat file input.txt dan menulis data ke dalamnya
with open("input.txt", "w") as infile:
    infile.write("Ini adalah data yang akan disalin ke
                                    output.txt.")

# Membaca isi file input.txt
with open("input.txt", "r") as infile:
    data = infile.read()

# Menulis data ke file output.txt
with open("output.txt", "w") as outfile:
    outfile.write(data)

print("Data berhasil dipindahkan dari 'input.txt' ke '
                                output.txt'.")
```

- Navigating directories and managing file permissions

Tulis sebuah program Python yang membuat sebuah direktori bernama *new directory*, kemudian mengatur izin file dari example.txt menjadi hanya dapat dibaca *(read-only)*.

```
import os

# Membuat direktori baru
os.makedirs("new_directory", exist_ok=True)
    print("Direktori 'new_directory' berhasil dibuat.")

# Membuat file example.txt di dalam direktori baru
file_path = os.path.join("new_directory", "example.txt")
with open(file_path, "w") as file:
    file.write("This is a test file.")

# Mengatur izin file menjadi read-only
os.chmod(file_path, 0o444)  # 0o444 adalah izin untuk
                                read-only
    print(f"Izin file '{file_path}' diatur menjadi read-
                                only.")
```

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.