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 yang Membuat Proses

(Isi materi disini)

3.6.3 Proses *Termination*

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

  Setelah proses dibuat, proses tersebut mulai berjalan dan melakukan apa pun tugasnya. Namun, tidak ada yang bertahan selamanya, bahkan tidak proses. Cepat atau lambat proses baru akan berhenti, biasanya karena salah satu kondisi berikut :

  1. Normal *Exit* (voluntary)
  2. Error *Exit* (voluntary)
  3. Fatal *Exit* (involuntary)
  4. *Killed by another process (involuntary)*

Sebagian besar proses berhenti karena mereka telah menyelesaikan tugasnya. Saat menjadi compiler telah mengkompilasi program yang diberikan padanya, kompilator mengeksekusi panggilan sistem untuk memberitahu sistem operasi yang sudah selesai. Panggilan ini keluar di UNIX dan *ExitProcess* di Windows. Program berorientasi layar juga mendukung penghentian sukarela. Kata prosesor, browser Internet, dan program serupa selalu memiliki ikon atau menu item yang dapat diklik pengguna untuk memberi tahu proses untuk menghapus file sementara yang dimilikinya buka lalu akhiri

Alasan kedua untuk penghentian adalah karena proses tersebut menemukan kesalahan fatal.

Alasan ketiga penghentian adalah kesalahan yang disebabkan oleh proses, seringkali karena bug program. Contohnya termasuk menjalankan instruksi ilegal, referensi memori yang tidak ada, atau membaginya dengan nol. Di beberapa sistem (misalnya, UNIX), sebuah proses dapat memberi tahu sistem operasi bahwa ia ingin menangani kesalahan tertentu itu sendiri, di mana kasus proses ini ditandai (terputus) bukannya dihentikan ketika salah satu kesalahan terjadi.

Alasan keempat suatu proses mungkin berhenti adalah bahwa proses tersebut menjalankan panggilan sistem yang memberi tahu sistem operasi untuk menghentikan beberapa proses lain. Di UNIX panggilan ini adalah membunuh. Fungsi Win32 yang sesuai adalah *TerminateProcess*.

Setelah sebuah proses dihentikan di sistem operasi, ada beberapa hal yang terjadi di balik layar untuk memastikan semuanya berjalan dengan lancar. Mari kita bahas ini dengan cara yang santai.

1. Pembebasan Sumber Daya

   Setiap proses yang berjalan di komputer menggunakan berbagai sumber daya, seperti memori, CPU, *file*, dan perangkat keras lainnya. Ketika proses tersebut dihentikan, semua sumber daya yang telah dipinjam oleh proses itu harus dikembalikan ke sistem agar bisa digunakan oleh proses lain.

2. Tanda Bahwa Proses Selesai

   Ketika sebuah proses dihentikan, OS akan menandai bahwa proses tersebut sudah selesai. Ini penting agar OS tidak mencoba men-

jalankan kembali proses yang sudah selesai. Selain itu, informasi tentang bagaimana proses itu berakhir (apakah sukses atau terjadi error) disimpan oleh OS.

Exit Status: Setelah proses berhenti, biasanya ada semacam kode yang dikembalikan untuk menunjukkan apakah proses itu berhasil atau ada masalah. Ini disebut *exit* status. Jika proses berjalan lancar, *exit* status-nya mungkin 0(nol). Kalau ada masalah, *exit* status-nya bisa berupa angka lain yang menunjukkan *error*.

3. Membersihkan Proses dari Daftar

   Sistem operasi memiliki daftar semua proses yang sedang berjalan. Ketika sebuah proses selesai, OS akan menghapus proses itu dari daftar. Ini seperti menandai bahwa pekerjaan selesai sehingga OS tidak lagi memperhitungkan proses tersebut.

4. Menginformasikan Proses Lain

   Kadang-kadang, ada proses lain yang bergantung pada proses yang baru saja dihentikan. OS akan memberitahukan proses-proses tersebut bahwa proses yang mereka tunggu sudah selesai. Ini memungkinkan proses lain untuk melanjutkan tugasnya.

5. *Logging* atau Pencatatan

   Terakhir, beberapa sistem operasi akan mencatat bahwa sebuah proses telah dihentikan dalam *log* sistem. Ini seperti mencatat dalam buku harian tentang apa yang telah terjadi. Informasi ini bisa berguna jika ada masalah atau jika kamu ingin melihat apa yang telah dilakukan komputer di masa lalu.

References

[1] Octaviano, A., Junianto, M. B. S., & Saprudin, S. (2022). SISTEM OPERASI.

[2] Pangera, A. A., & Ariyus, D. (2005). Sistem Operasi. Penerbit Andi.

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

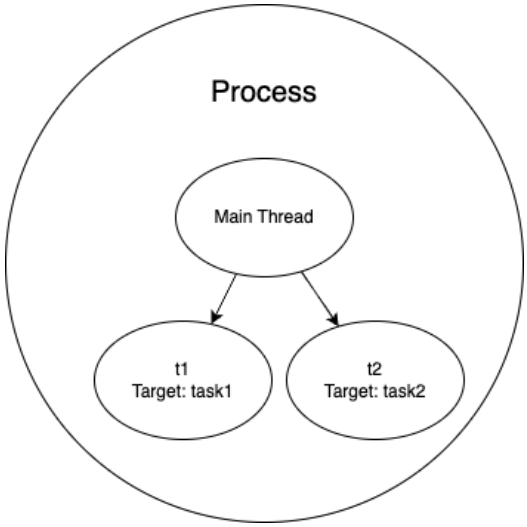

Figure 1: Ini adalah gambar contoh dari multithreading.

Seperti yang terlihat pada Gambar 1, 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.

4.4.1 Group6

Tuliskan implementasi CLI sederhana yang dapat melakukan perintah berikut:

4.4.2 Group 6

Soal:

Buatlah *Simple CLI* menggunakan Python, lalu lakukan serangkaian operasi berikut dan catat hasilnya:

1. Buat sebuah berkas bernama `laporan.txt`.
2. Tampilkan daftar berkas dalam direktori saat ini.
3. Tampilkan status sistem.
4. Tampilkan daftar proses yang sedang berjalan.
5. Hapus berkas `laporan.txt` yang telah dibuat sebelumnya.
6. Tampilkan kembali daftar berkas dalam direktori saat ini.

Berikan tangkapan layar atau salin keluaran dari setiap langkah di atas.

Jawaban:

```
import os
import psutil
import datetime

def create_file(filename):
    with open(filename, 'w') as f:
        f.write("This is a new file.")
    return f"File '{filename}' created successfully."

def list_files():
    files = os.listdir('.')
    return "\n".join(files)

def delete_file(filename):
    if os.path.exists(filename):
        os.remove(filename)
```

```
        return f"File '{filename}' deleted successfully."
    else:
        return f"File '{filename}' not found."

def list_processes():
    processes = []
    for proc in psutil.process_iter(['pid', 'name']):
        processes.append(f"{proc.info['pid']}: {proc.info['name']}")
    return "\n".join(processes[:10])  # Limiting to first 10 for brevity

def system_status():
    cpu_percent = psutil.cpu_percent()
    memory = psutil.virtual_memory()
    disk = psutil.disk_usage('C:\\')  # Ganti '/' dengan 'C:\\'

    return f"CPU Usage: {cpu_percent}%\nMemory Usage: {memory.percent}%

def main():
    print("Welcome to the Simple CLI Simulator!")
    print("Type 'help' for a list of commands.")

    while True:
        command = input(">>> ").strip().lower()

        if command == "exit":
            print("Goodbye!")
            break
        elif command == "help":
            print("Available commands:")
            print("  create <filename> - Create a new file")
            print("  list - List files in the current directory")
            print("  delete <filename> - Delete a file")
            print("  processes - List running processes")
            print("  status - Show system status")
            print("  exit - Exit the CLI")
        elif command.startswith("create "):
            filename = command.split(" ", 1)[1]
            print(create_file(filename))
```

```
        elif command == "list":
            print(list_files())
        elif command.startswith("delete "):
            filename = command.split(" ", 1)[1]
            print(delete_file(filename))
        elif command == "processes":
            print(list_processes())
        elif command == "status":
            print(system_status())
        else:
            print("Unknown command. Type 'help' for a list of commands.")

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

Output:

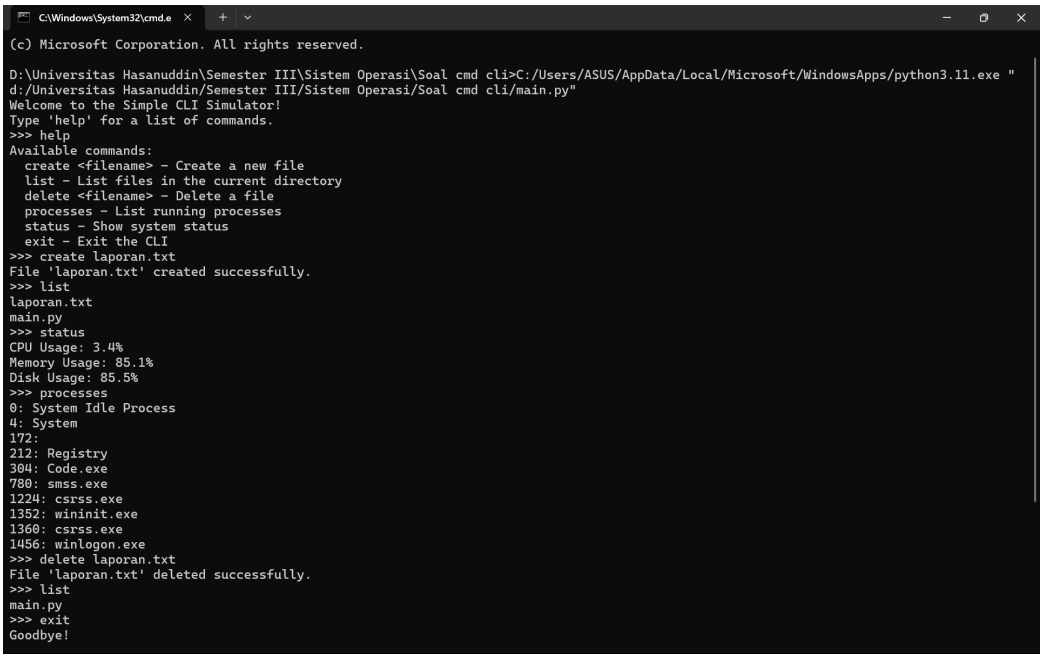

Figure 2: *Output* Kode

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.