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

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

- Process spawning
- Process termination conditions

3.7 Introduction to Threads

This section introduces the concept of threads and their relation to processes, covering:

- Single-threaded vs. multi-threaded processes
- Benefits of multithreading

3.8 File Systems

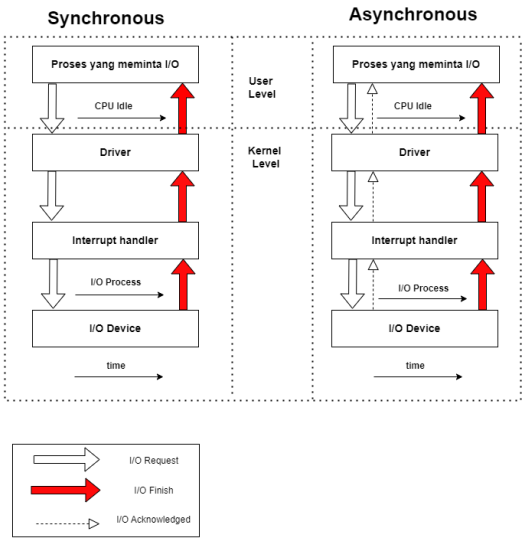
File systems provide a way for the operating system to store, retrieve, and manage data. This section explains:

- 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:

3.9.1 Struktur *Input/Output*



Gambar 1 : Struktur I/O *Synchronous* dan *Asynchronous.*

Nurasyifa, dkk (2024) menjelaskan bahwa struktur I/O terbagi menjadi dua yaitu:

1. Struktur I/O *Synchronous*

   (a) Karakteristik :

   - Proses yang meminta I/O akan menunggu hingga operasi I/O selesai.
   - CPU akan *idle* (tidak melakukan pekerjaan lain) selama menunggu operasi I/O.

   (b) Alur Kerja:

   - Proses aplikasi *user level* meminta operasi I/O.
   - Permintaan diteruskan ke *driver* perangkat.

- *Driver* akan melakukan operasi I/O dan menunggu hingga selesai.
- Setelah selesai, *driver* akan memberitahu proses aplikasi.
- Proses aplikasi kemudian dapat melanjutkan pekerjaannya.

(c) Kelebihan: Sederhana untuk diimplementasikan.

(d) Kekurangan:

- Efisiensi CPU rendah karena CPU harus menunggu.
- Respons sistem menjadi lambat, terutama untuk operasi I/O yang lama.

2. Struktur I/O *Asynchronous*

(a) Karakteristik

- Proses yang meminta I/O tidak perlu menunggu, melainkan dapat melanjutkan pekerjaan lain.
- CPU dapat melakukan tugas lain selama operasi I/O berlangsung.

(b) Alur Kerja:

- Proses aplikasi meminta operasi I/O.
- Permintaan diteruskan ke *driver* perangkat.
- *Driver* akan memulai operasi I/O dan memberikan kendali kembali ke CPU.
- Saat operasi I/O selesai, perangkat akan mengirimkan interupsi ke CPU.
- CPU akan merespons interupsi dengan menjalankan *interrupt handler.*
- *Interrupt handler* akan memberitahu proses aplikasi bahwa operasi I/O telah selesai.

(c) Kelebihan

- Efisiensi CPU tinggi karena CPU dapat melakukan tugas lain.
- Respons sistem lebih cepat.

(d) Kekurangan:

- Implementasinya lebih kompleks
- Membutuhkan mekanisme penanganan interupsi

3.9.2 Metode Operasi Sistem *Input/Output*

1. I/O Terprogram

   Metode ini melibatkan CPU secara langsung dalam proses pengiriman dan penerimaan data dari perangkat I/O. Dalam I/O terprogram, CPU harus memeriksa secara terus-menerus *(polling)* apakah perangkat I/O siap untuk melakukan transfer data. CPU kemudian mentransfer data secara manual antara perangkat dan memori. Ini menyebabkan CPU terlibat penuh dalam operasi I/O dan tidak dapat melakukan tugas lain selama proses berlangsung. Salah satu kelemahan utama metode ini adalah efisiensinya yang rendah karena CPU harus menunggu perangkat I/O untuk siap.

2. I/O *Interrupt Driven*

   Pada metode ini, perangkat I/O mengirim sinyal interupsi kepada CPU ketika perangkat tersebut siap untuk menerima atau mengirim data. CPU tidak perlu terus-menerus memeriksa status perangkat I/O, sehingga bisa mengerjakan tugas lain sampai menerima interupsi. Ketika interupsi terjadi, CPU menghentikan sementara tugas yang sedang dilakukan, menangani interupsi untuk menyelesaikan operasi I/O, kemudian kembali melanjutkan tugas sebelumnya. Metode ini lebih efisien dibandingkan I/O terprogram karena CPU tidak selalu terikat dengan perangkat I/O dan bisa mengerjakan tugas lain sambil menunggu interupsi.

3. *Direct Memory Access* (DMA)

   DMA adalah metode yang memungkinkan perangkat I/O mengakses memori secara langsung tanpa melibatkan CPU dalam proses transfer data. Dengan menggunakan DMA *controller*, data dapat ditransfer antara perangkat I/O dan memori secara mandiri, sementara CPU hanya diberi tahu ketika operasi transfer selesai. Metode ini sangat efisien karena memungkinkan CPU untuk melakukan tugas lain tanpa harus terlibat langsung dalam proses I/O, sehingga cocok untuk transfer data dalam jumlah besar atau operasi I/O yang intensif.

3.9.3 *Interrupt Handler*

Apabila suatu transfer selesai maka pengontrol biasanya menyebabkan suatu *interupt* yang memaksa CPU untuk menunda menjalankan programnya dan mulai menjalankan prosedur khusus yang disebut *interupt handler*. *Interupt handler* adalah kondisi dimana CPU menunda program yang sedang dijalankan dan berganti menjalankan program khusus pada saat akses memori langsung sedang berlangsung dan minta *interupt*.

References

[1] Alifah, N., Deanda, G. V., Juniwan, Aribowo, D. (2023).Peran Teknologi Input dan Output dalam Pengembangan Perangkat Keras.*Jurnal Kendali Teknik dan Sains,*.1(4), 123-136.

[2] Gallo, K. (2024,September20). *builtin.* Retrieved from https://builtin.com/hardware/i-o-input-output

[3] Nurasyifa, N., Sheril, J. A., Maulana, M. R., Muiz, A.,Febrian.(2024). ANALISIS INPUT/OUTPUT:PERANGKAT DAN INTERFACE PADA ORGANISASI ARSITEKTUR KOMPUTER.*Jurnal Multidisiplin Saintek,* 3(6), 100-111.

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 Group 9

- **Tugas Pemrograman CLI : Manajemen Antarmuka Pengguna**

Deskripsi Tugas: Buatlah sebuah program *Command Line Interface (CLI)* sederhana untuk manajemen antarmuka pengguna. Program ini harus mendukung perintah-perintah dasar berikut:

1. Manipulasi berkas baru:
    - Membuat Berkas baru
    - Menampilkan daftar berkas dalam direktori saat ini.
    - Menghapus berkas yang dipilih.
2. Manajemen Proses
    - Menampilkan daftar proses yang sedang berjalan di sistem.
    - Menghentikan proses berdasarkan PID *(Process ID).*
3. Pelaporan Status Sistem
    - Menampilkan penggunaan CPU saat ini.
    - Menampilkan penggunaan memori saat ini.
    - Menampilkan waktu aktif sistem *(uptime).*

Ketentuan :

1. Program harus berjalan di lingkungan CLI *(Command Line Interface)* tanpa antarmuka grafis.
2. Pengguna dapat memasukkan perintah melalui CLI untuk menjalankan fungsi yang diinginkan.
3. Buat dokumentasi singkat mengenai cara penggunaan program (petunjuk perintah).

Fitur Tambahan (Opsional):

- Menyediakan opsi untuk menampilkan daftar direktori saat ini.
- Menampilkan informasi jaringan seperti alamat IP atau kecepatan jaringan.

- Membuat *log* aktivitas untuk setiap perintah yang dijalankan.

Contoh Perintah CLI yang Diharapkan:

- *create-file*
- *list-files*
- *delete-file*
- *list-processes*
- *kill-process*
- *cpu-status*
- *memory-status*
- *system-uptime*

Jawaban:

```
import os
import psutil
import time

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

def list_files():
    """Menampilkan daftar file di direktori saat ini."""
    files = os.listdir('.')
    print("Daftar berkas:")
    for f in files:
        print(f"- {f}")

def delete_file(filename):
    """Menghapus file."""
    try:
        os.remove(filename)
```

```
        print(f"File '{filename}' berhasil dihapus.")
    except Exception as e:
        print(f"Gagal menghapus file: {e}")

def list_processes():
    """Menampilkan daftar proses yang berjalan."""
    processes = psutil.pids()
    for pid in processes:
        p = psutil.Process(pid)
        print(f"PID: {pid}, Name: {p.name()}, Status: {p.
                                        status()}")

def kill_process(pid):
    """Menghentikan proses berdasarkan PID."""
    try:
        p = psutil.Process(int(pid))
        p.terminate()
        print(f"Proses dengan PID {pid} berhasil dihentikan."
                                            )
    except Exception as e:
        print(f"Gagal menghentikan proses: {e}")

def cpu_status():
    """Menampilkan status penggunaan CPU."""
    cpu_percent = psutil.cpu_percent(interval=1)
    print(f"Penggunaan CPU: {cpu_percent}%")

def memory_status():
    """Menampilkan status penggunaan memori."""
    mem = psutil.virtual_memory()
    print(f"Total Memori: {mem.total // (1024 ** 2)} MB")
    print(f"Memori Terpakai: {mem.used // (1024 ** 2)} MB")
    print(f"Memori Bebas: {mem.available // (1024 ** 2)} MB")

def system_uptime():
    """Menampilkan uptime sistem."""
    uptime_seconds = time.time() - psutil.boot_time()
    uptime_str = time.strftime("%H:%M:%S", time.gmtime(
                                    uptime_seconds))
    print(f"System Uptime: {uptime_str}")

def main():
    while True:
        print("\nPerintah CLI yang tersedia:")
        print("1. create-file <nama_berkas>")
```

```
        print("2. list-files")
        print("3. delete-file <nama_berkas>")
        print("4. list-processes")
        print("5. kill-process <pid>")
        print("6. cpu-status")
        print("7. memory-status")
        print("8. system-uptime")
        print("9. exit")

        command = input("\nMasukkan perintah: ").split()

        if len(command) == 0:
            continue

        if command[0] == "create-file" and len(command) == 2:
            create_file(command[1])
        elif command[0] == "list-files":
            list_files()
        elif command[0] == "delete-file" and len(command) ==
                                        2:
            delete_file(command[1])
        elif command[0] == "list-processes":
            list_processes()
        elif command[0] == "kill-process" and len(command) ==
                                         2:
            kill_process(command[1])
        elif command[0] == "cpu-status":
            cpu_status()
        elif command[0] == "memory-status":
            memory_status()
        elif command[0] == "system-uptime":
            system_uptime()
        elif command[0] == "exit":
            print("Keluar dari program.")
            break
        else:
            print("Perintah tidak valid. Coba lagi.")

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

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.