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:

3.4.1 *Process Description*

Menurut Silberschatz, Galvin, dan Gagne (2009), sebuah proses adalah sebuah program yang sedang dijalankan pada komputer. Setiap proses membutuhkan sumber daya, seperti CPU, memori, dan perangkat *input/output*, untuk bisa berfungsi dengan baik. Komputer harus mengelola berbagai proses agar semua program dapat berfungsi dengan baik. Dalam pengelolaan ini, setiap proses memiliki *process state* dan *Process Control Block* (PCB) yang membantu sistem untuk mengawasi dan mengatur kondisi serta informasi penting mengenai proses tersebut.

3.4.2 Process States and State Transitions

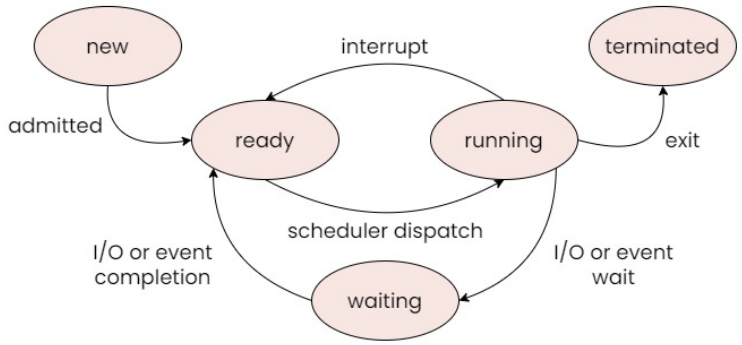

Figure 1: Gambar 1: *Process states and state transitions*

Dalam sistem operasi (OS), manajemen cara program berjalan dan berinteraksi dengan sumber daya sistem sangat penting untuk kinerja yang efisien. Model *5-process-state* adalah kerangka dasar yang digunakan oleh OS untuk mengategorikan dan mengendalikan perilaku proses.

Model ini membagi siklus hidup sebuah proses menjadi lima status yang berbeda, masing-masing mewakili tahap yang berbeda dalam eksekusinya. Memahami status-status ini membantu dalam mengoptimalkan penggunaan sumber daya, memastikan *multitasking* yang lancar, dan menjaga stabilitas sistem. (GeeksForGeeks, 2024)

Siklus hidup proses dalam sistem operasi terdiri dari beberapa status, seperti yang terlihat pada gambar di atas. Status-status tersebut meliputi:

- ***New***: Proses baru saja dibuat dan sedang menunggu untuk dimasukkan dalam daftar proses yang siap dijalankan. Proses akan berada dalam status ini hingga sistem menerima proses tersebut.
- ***Ready***: Setelah diterima, proses akan berpindah ke status *ready*. Proses sudah siap untuk dieksekusi namun harus menunggu hingga *scheduler* memilihnya untuk dieksekusi.
- ***Running***: Proses sedang dieksekusi oleh CPU. Setelah dijadwalkan, proses akan berpindah dari status *ready* ke *running*.
- ***Waiting***: Proses sedang menunggu operasi I/O atau suatu *event* tertentu. Ketika proses membutuhkan input dari perangkat keras atau menunggu suatu *event*, ia akan berpindah dari status running ke *waiting*.
- ***Terminated***: Proses telah selesai dieksekusi. Setelah eksekusi selesai, proses akan berpindah dari status *running* ke *terminated*, di mana ia tidak lagi menjadi bagian dari sistem.

Transisi antar status tersebut meliputi:

- ***Admitted***: Ketika sebuah proses baru dibuat, ia akan berpindah dari status *new* ke *ready* setelah diterima oleh sistem operasi.
- ***Scheduler Dispatch***: Ketika CPU siap untuk mengeksekusi proses, *scheduler* akan memindahkan proses dari status *ready* ke *running*.
- **I/O or *Event Wait***: Ketika proses membutuhkan operasi I/O atau menunggu suatu kejadian, proses akan berpindah dari *running* ke *waiting*.
- **I/O or *Event Completion***: Setelah operasi I/O atau kejadian selesai, proses yang berada di status *waiting* akan berpindah kembali ke status *ready*.
- ***Interrupt***: Jika proses yang sedang berjalan terganggu (misalnya terganggu oleh *scheduler* untuk memberikan giliran kepada proses lain), proses akan berpindah dari *running* ke *ready*.
- ***Exit***: Ketika proses selesai dieksekusi, ia akan berpindah dari *running* ke *terminated*.

3.4.3 *Process Control Block* (PCB)

3.4.4 *Process Control*

References

[1] Silberschatz, A., Galvin, P. B., & Gagne, G. (2009). *Operating System Concepts* (8th ed.). Hoboken, NJ: Wiley.

[2] GeeksForGeeks. (2024, 22 Juli). *5 State Process Model in Operating System*. Diakses pada 10 Oktober 2024, dari `https://www.geeksforgeeks.org/5-state-process-model-in-operating-system/`.

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

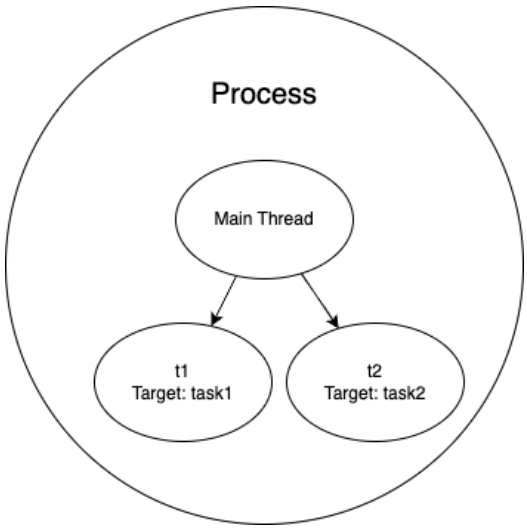

Figure 2: Ini adalah gambar contoh dari multithreading.

Seperti yang terlihat pada Gambar 2, 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

4.1.2 Group 4

Pertanyaan:

Implementasikan algoritma *First-Come, First-Served* (FCFS), *Shortest Job Next* (SJN), dan *Round Robin* (RR) dengan *time quantum* = 4 untuk menghitung waktu tunggu (*waiting time*) setiap proses serta rata-rata waktu tunggu proses berikut:

Process ID	**Burst Time (ms)**
P1	5
P2	3
P3	8
P4	6

Table 2: Tabel 1: *Process ID and their respective Burst Time*

Dari ketiga algoritma tersebut, algoritma mana yang memiliki waktu tunggu rata-rata (*average waiting time*) yang paling rendah?

Jawaban:

Untuk menjawab soal tersebut kita dapat menggunakan kode di bawah ini

```
        # Fungsi untuk algoritma First-Come, First-Served (
                                        FCFS)
        def fcfs(processes):
    print("\nFirst-Come, First-Served (FCFS)")
    waiting_time = 0
    total_waiting_time = 0
    for p in processes:
        print(f"Process {p[0]} | Burst Time: {p[1]} | Waiting
                                         Time: {waiting_time}"
                                        )
        total_waiting_time += waiting_time
        waiting_time += p[1]
    avg_waiting_time = total_waiting_time / len(processes)
    print(f"Average Waiting Time: {avg_waiting_time:.2f}")
    return avg_waiting_time

# Fungsi untuk algoritma Shortest Job Next (SJN)
def sjn(processes):
    print("\nShortest Job Next (SJN)")
    processes_sjn = sorted(processes, key=lambda x: x[1])  #
                                    Urutkan berdasarkan Burst
                                    Time
    waiting_time = 0
    total_waiting_time = 0
    for p in processes_sjn:
        print(f"Process {p[0]} | Burst Time: {p[1]} | Waiting
                                         Time: {waiting_time}"
                                        )
        total_waiting_time += waiting_time
        waiting_time += p[1]
    avg_waiting_time = total_waiting_time / len(processes_sjn
                                    )
    print(f"Average Waiting Time: {avg_waiting_time:.2f}")
    return avg_waiting_time

# Fungsi untuk algoritma Round Robin (RR)
def round_robin(processes, quantum):
    print("\nRound Robin (RR)")
    remaining_burst_time = [p[1] for p in processes]
    waiting_time = [0] * len(processes)
    t = 0  # Waktu saat ini

    while True:
        done = True
        for i in range(len(processes)):
```

```
            if remaining_burst_time[i] > 0:
                done = False
                if remaining_burst_time[i] > quantum:
                    t += quantum
                    remaining_burst_time[i] -= quantum
                else:
                    t += remaining_burst_time[i]
                    waiting_time[i] = t - processes[i][1]
                    remaining_burst_time[i] = 0
        if done:
            break

    for i, p in enumerate(processes):
        print(f"Process {p[0]} | Burst Time: {p[1]} | Waiting
                                        Time: {waiting_time[i
                                       ]}")
    avg_waiting_time = sum(waiting_time) / len(processes)
    print(f"Average Waiting Time: {avg_waiting_time:.2f}")
    return avg_waiting_time

# Daftar proses dengan format: [Process ID, Burst Time]
processes = [[1, 5], [2, 3], [3, 8], [4, 6]]

# Panggil fungsi untuk setiap algoritma scheduler
avg_fcfs = fcfs(processes)
avg_sjn = sjn(processes)
avg_rr = round_robin(processes, quantum=4)

# Mencari algoritma dengan rata-rata waktu tunggu terendah
min_avg_time = min(avg_fcfs, avg_sjn, avg_rr)
if min_avg_time == avg_fcfs:
    best_algorithm = "First Come, First Serve"
elif min_avg_time == avg_sjn:
    best_algorithm = "Shortest Job Next"
else:
    best_algorithm = "Round Robin"

print(f"\nAlgoritma dengan rata-rata waktu tunggu paling
                                rendah adalah {best_algorithm}
                                 dengan rata-rata waktu tunggu
                                 sebesar {min_avg_time:.2f}.")
```

Output Kode:

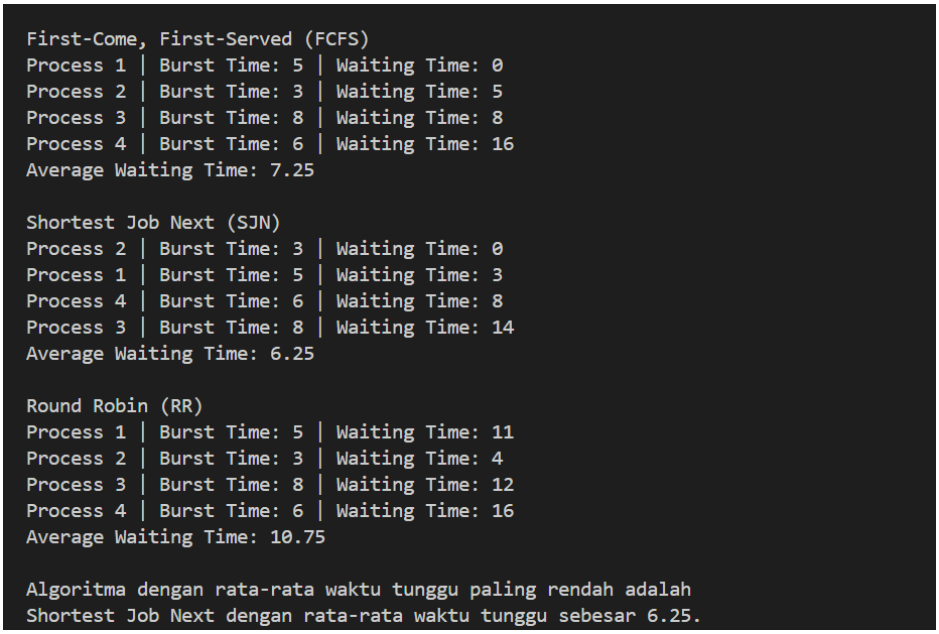

Figure 3: Gambar 2: *Output of Scheduling Algorithm Program*

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 4

Pertanyaan: Buatlah sebuah program Python yang berfungsi sebagai CLI sederhana untuk manajemen antarmuka pengguna! Program tersebut harus mendukung perintah dasar seperti manipulasi *file* (membuat, menampilkan daftar, dan menghapus *file*) serta pelaporan status sistem.

Jawaban:

```
import os
import platform

# Membuat file baru dengan nama yang diberikan
def create_file(filename):
    with open(filename, 'w') as f:
        pass
    print(f"File '{filename}' created successfully.")

# Menampilkan daftar file dari sebuah direktori
def list_files(directory='.'):
    try:
        files = os.listdir(directory)
        if files:
            print("Files in directory:", directory)
            for file in files:
                print(f"- {file}")
        else:
            print(f"No files found in directory: {directory}"
                                                )
    except FileNotFoundError:
        print(f"Directory '{directory}' not found.")

# Menghapus file dengan nama yang diberikan
def delete_file(filename):
    if os.path.isfile(filename):
        os.remove(filename)
        print(f"File '{filename}' deleted successfully.")
    else:
        print(f"File '{filename}' not found.")

# Menampilkan status sistem sederhana
def show_system_status():
    print(f"Operating System: {platform.system()} {platform.
                                    release()}")
    print(f"CPU Count: {os.cpu_count()}")
```

```
# Fungsi utama
def main():
    while True:
        print("\nCLI Simple Management Tool")
        print("1. Create File")
        print("2. List Files")
        print("3. Delete File")
        print("4. Show System Status")
        print("5. Exit")
        choice = input("Enter your choice (1-5): ")

        if choice == '1':
            filename = input("Enter the name of the file to
                                        create: ")
            create_file(filename)
        elif choice == '2':
            directory = input("Enter the directory to list
                                        files (default is
                                        current directory)
                                        : ") or '.'
            list_files(directory)
        elif choice == '3':
            filename = input("Enter the name of the file to
                                        delete: ")
            delete_file(filename)
        elif choice == '4':
            show_system_status()
        elif choice == '5':
            print("You have successfully exited the CLI.")
            break
        else:
            print("Invalid choice, please select a number
                                        between 1 and 5.")

# Eksekusi program utama
if __name__ == "__main__":
    main()
```

Berikut adalah *output* dari program CLI yang dijalankan di *command line*.

Figure 4: Gambar 3: Membuat *file* dengan nama 'tes.txt'

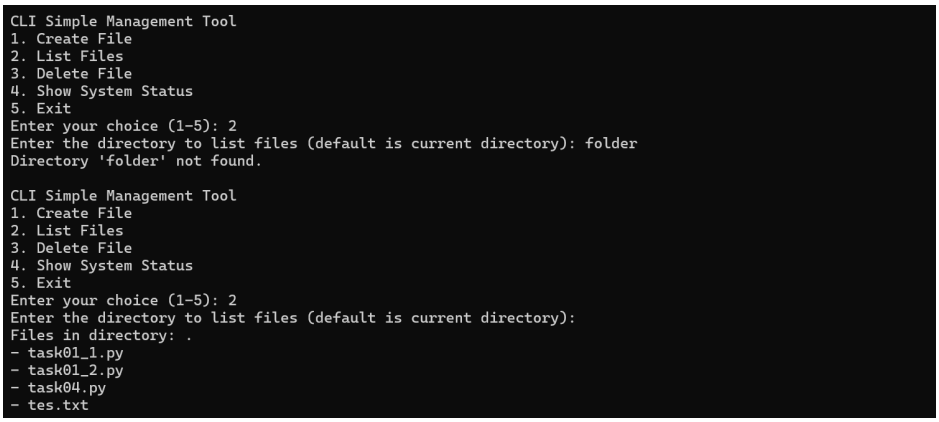

Figure 5: Gambar 4: Menampilkan daftar *file* dalam direktori

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.

Figure 6: Gambar 5: Menghapus *file* bernama 'tes.txt'

Figure 7: Gambar 6: Menampilkan status sistem sederhana

Figure 8: Gambar 7: Keluar dari program