Operating System Course Report - First Half of the Semester

B class

October 8, 2024

Contents

1 Introduction **4**

2 Course Overview **4**
2.1 Objectives 4
2.2 Course Structure 4

3 Topics Covered **5**
3.1 Basic Concepts and Components of Computer Systems 5
3.2 System Performance and Metrics 5
3.3 System Architecture of Computer Systems 5
3.4 Process Description and Control 5
3.4.1 Definition of Process 5
3.4.2 Process states and state transitions 6
3.4.3 Process Control Block (PCB) 7
3.4.4 Context Switching 7
3.4.5 Process Management By Operating Systems 7
3.5 Scheduling Algorithms 7
3.6 Process Creation and Termination 8
3.7 Introduction to Threads 8
3.8 File Systems 9
3.9 Input and Output Management 9
3.10 Deadlock Introduction and Prevention 9
3.11 User Interface Management 9
3.12 Virtualization in Operating Systems 10

4 Assignments and Practical Work **10**
4.1 Assignment 1: Process Scheduling 10
4.1.1 Group 1 10
4.2 Assignment 2: Deadlock Handling 11
4.2.1 Contoh Kasus Deadlock 11
4.2.2 Implementasi Algoritma Bankir 11
4.2.3 Penjelasan Kode 13
4.3 Assignment 3: Multithreading and Amdahl's Law 13
4.4 Assignment 4: Simple Command-Line Interface (CLI) for User Interface Management 14
4.4.1 Tujuan Membuat CLI 14

4.4.2 Fitur CLI 14
4.4.3 Penjelasan Fitur 15
4.4.4 **Contoh Soal** 15
4.4.5 **Contoh Kode Python** 16
4.4.6 **Penjelasan Kode** 18
4.4.7 **Output Program CLI Sederhana** 18
4.5 Assignment 5: File System Access 21

5 Conclusion **21**

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

3.4.1 Definition of Process

Proses adalah program yang sedang dieksekusi, yang terdiri dari instruksi yang berjalan pada CPU, serta informasi yang diperlukan untuk menjalankan program tersebut. Setiap proses memiliki siklus hidup yang melibatkan berbagai keadaan (states) dan transisi antar keadaan tersebut.

3.4.2 Process states and state transitions

Keadaan proses merujuk pada status atau fase yang dilalui oleh suatu proses selama siklus hidupnya. Setiap proses dapat berada dalam salah satu dari beberapa keadaan yang berbeda, dan transisi antar keadaan ini diatur oleh sistem operasi untuk memastikan manajemen proses yang efisien. Berikut adalah beberapa keadaan utama yang biasanya diidentifikasi dalam sistem operasi:

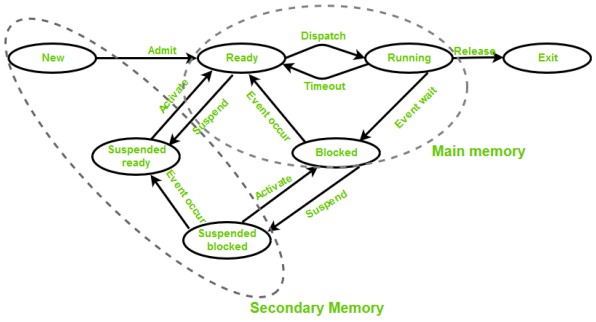

Figure 1: gambar keadaan dalam proses

- **New (Baru)**: Proses baru saja dibuat. Pada keadaan ini, proses sedang dalam tahap inisialisasi, di mana sistem operasi mengalokasikan sumber daya yang diperlukan seperti memori dan ID proses.
- **Ready (Siap)**: Proses telah siap untuk dieksekusi tetapi belum mendapatkan akses ke CPU. Proses ini menunggu dalam antrean ready hingga sistem operasi memilihnya untuk dieksekusi. Proses dalam keadaan ini dapat dipindahkan kembali ke antrean jika CPU sedang sibuk.
- **Running (Berjalan)**: Proses sedang dieksekusi oleh CPU. Pada keadaan ini, instruksi proses sedang dijalankan. CPU memberikan waktu eksekusi untuk proses dalam keadaan ini.
- **Waiting (Menunggu)**: Proses tidak dapat melanjutkan eksekusi hingga suatu event tertentu terjadi seperti menyelesaikan operasi I/O. Ketika menunggu, proses tidak dapat menggunakan CPU dan akan pindah ke antrean waiting.

- **Terminated (Selesai)**: Proses telah selesai menjalankan semua instruksinya. Semua sumber daya yang dialokasikan untuk proses tersebut dibebaskan, dan sistem operasi memperbarui status proses menjadi terminated.

Transisi antara keadaan proses ini adalah bagian penting dari manajemen proses. Berikut adalah beberapa transisi umum:

- **New Ready**: Ketika proses berhasil dibuat dan siap untuk dijadwalkan.
- **Ready Running**: Ketika CPU memilih proses dari antrean ready untuk dieksekusi.
- **Running Waiting**: Ketika proses membutuhkan I/O atau menunggu event tertentu (misalnya menyelesaikan pembacaan dari disk).
- **Running Ready**: Ketika proses di-preempt karena waktu eksekusi habis atau ada proses dengan prioritas lebih tinggi yang memerlukan CPU.
- **Waiting Ready**: Ketika event yang ditunggu oleh proses selesai, proses tersebut kembali ke antrean ready.
- **Running Terminated**: Ketika proses selesai menjalankan semua instruksinya.

3.4.3 Process Control Block (PCB)

3.4.4 Context Switching

3.4.5 Process Management By Operating Systems

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

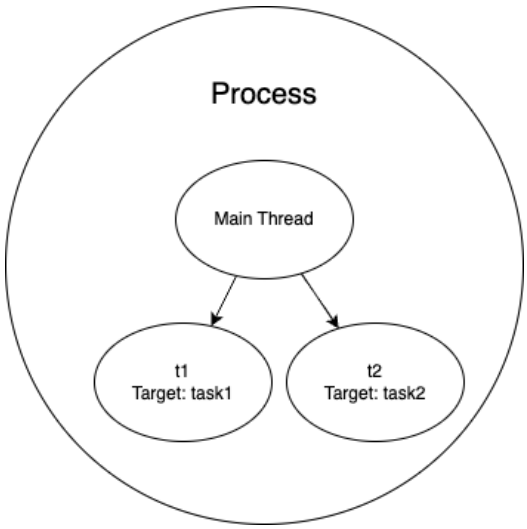

Figure 2: Ini adalah gambar contoh dari multithreading.

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

4.2 Assignment 2: Deadlock Handling

4.2.1 Contoh Kasus Deadlock

Misalkan terdapat tiga proses yang bersaing untuk mendapatkan tiga jenis sumber daya: A, B, dan C. Setiap jenis sumber daya memiliki 10 unit. Proses-proses tersebut adalah sebagai berikut:

- P1 memiliki 3 unit A, 2 unit B, dan 2 unit C, serta membutuhkan tambahan 1 unit A, 1 unit B, dan 3 unit C.
- P2 memiliki 2 unit A, 3 unit B, dan 3 unit C, serta membutuhkan tambahan 2 unit A, 1 unit B, dan 2 unit C.
- P3 memiliki 1 unit A, 2 unit B, dan 2 unit C, serta membutuhkan tambahan 4 unit A, 2 unit B, dan 2 unit C.

Apakah sistem ini dalam kondisi deadlock? Jika iya, bagaimana cara menghindarinya menggunakan Algoritma Bankir? (Silberschatz, 2018) [4]

4.2.2 Implementasi Algoritma Bankir

Berikut adalah contoh kode Python untuk mensimulasikan Algoritma Bankir:

```
# Algoritma Bankir dalam Python

# Jumlah proses dan sumber daya
P = 3  # Jumlah proses
R = 3  # Jumlah jenis sumber daya (A, B, C)

# Sumber daya yang tersedia
available = [3, 3, 2]  # A = 3, B = 3, C = 2 (Sumber daya
                                yang tersedia)

# Permintaan maksimum dari masing-masing proses
max_demand = [
    [7, 5, 3],  # Permintaan maksimum P1 untuk A, B, C
    [3, 2, 2],  # Permintaan maksimum P2
    [9, 0, 2]   # Permintaan maksimum P3
]

# Sumber daya yang saat ini dialokasikan untuk masing-
                                masing proses
```

```
allocated = [
    [3, 2, 2],  # Sumber daya yang dialokasikan untuk P1
    [2, 0, 0],  # Sumber daya yang dialokasikan untuk P2
    [3, 0, 2]   # Sumber daya yang dialokasikan untuk P3
]

# Menghitung matriks kebutuhan (max_demand - allocated)
def calculate_need(max_demand, allocated):
    need = []
    for i in range(P):
        need.append([max_demand[i][j] - allocated[i][j]
                                        for j in range(R)]
                                        )
    return need

# Fungsi untuk memeriksa apakah sistem dalam kondisi aman
def is_safe(available, max_demand, allocated):
    need = calculate_need(max_demand, allocated)
    work = available[:]
    finish = [False] * P
    safe_sequence = []

    while len(safe_sequence) < P:
        found_process = False
        for i in range(P):
            if not finish[i]:
                if all(need[i][j] <= work[j] for j in
                                                range(R)):
                    # Jika proses dapat diselesaikan
                    for j in range(R):
                        work[j] += allocated[i][j]
                    finish[i] = True
                    safe_sequence.append(i)
                    found_process = True

        if not found_process:
            break

    if len(safe_sequence) == P:
        print(f"Sistem berada dalam keadaan aman.\nUrutan
                                        aman: {
                                        safe_sequence}")
        return True
    else:
        print("Sistem berada dalam kondisi deadlock.")
```

```
            return False

    # Memeriksa apakah sistem dalam keadaan aman
    is_safe(available, max_demand, allocated)
```

4.2.3 Penjelasan Kode

- **available**: Jumlah sumber daya yang tersedia untuk masing-masing jenis (A, B, dan C).
- **max_demand**: Matriks yang menunjukkan permintaan maksimum sumber daya untuk setiap proses.
- **allocated**: Matriks yang menunjukkan jumlah sumber daya yang telah dialokasikan untuk masing-masing proses.
- **calculate_need**: Fungsi untuk menghitung sumber daya yang masih dibutuhkan oleh setiap proses.
- **is_safe**: Fungsi utama yang menentukan apakah sistem berada dalam keadaan aman (safe state) atau dalam kondisi deadlock, menggunakan Algoritma Bankir.

Jika sistem berada dalam kondisi aman, program akan menampilkan pesan:

```
Sistem berada dalam keadaan aman.
Urutan aman: [0, 2, 1]
```

Jika sistem berada dalam kondisi deadlock, program akan menampilkan pesan:

```
Sistem berada dalam kondisi deadlock.
```

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

Tugas ini bertujuan untuk membuat Command-Line Interface (CLI) sederhana yang memfasilitasi manajemen antarmuka pengguna. CLI ini memungkinkan pengguna untuk melakukan operasi dasar terkait pengelolaan file dan proses, serta memantau status sistem melalui perintah yang mudah.

4.4.1 Tujuan Membuat CLI

Membuat CLI yang mampu menangani:

- **Manipulasi file**: membuat, menampilkan, dan menghapus file.
- **Manajemen proses**: menampilkan dan menghentikan proses berdasarkan PID (Process ID).
- **Pelaporan status sistem**: menampilkan informasi memori dan ruang penyimpanan yang digunakan.

4.4.2 Fitur CLI

CLI harus mampu menerima perintah pengguna untuk:

- Membuat file baru.
- Menampilkan daftar file dalam direktori saat ini.
- Menghapus file yang ada.
- Menampilkan daftar proses yang berjalan.
- Menghentikan proses berdasarkan PID.
- Menampilkan status memori dan ruang penyimpanan sistem.

4.4.3 Penjelasan Fitur

1. **Manipulasi File**
    - **Membuat File**: Membuat file teks sesuai keinginan pengguna.
    - **Menampilkan Daftar File**: Menampilkan semua file dalam direktori saat ini.
    - **Menghapus File**: Menghapus file berdasarkan nama yang diberikan.
2. **Manajemen Proses**
    - **Menampilkan Daftar Proses**: Menampilkan semua proses berjalan, lengkap dengan PID dan nama proses.
    - **Menghentikan Proses**: Menghentikan proses berdasarkan PID yang diberikan.
3. **Pelaporan Status Sistem**
    - **Status Memori**: Menampilkan penggunaan memori sistem.
    - **Ruang Penyimpanan**: Menampilkan status penggunaan ruang penyimpanan

4.4.4 Contoh Soal

Soal: Buatlah sebuah program Python yang berfungsi sebagai CLI sederhana untuk melakukan:

1. Membuat file teks baru dengan nama dan isi yang diinginkan.
2. Menampilkan daftar file dalam direktori saat ini.
3. Menghapus file yang dipilih pengguna.
4. Menampilkan proses yang sedang berjalan di sistem.
5. Menghentikan proses berdasarkan **PID**.
6. Menampilkan informasi status sistem, seperti penggunaan memori dan ruang penyimpanan.

4.4.5 Contoh Kode Python

Berikut adalah implementasi Python untuk menjawab soal tersebut

```
import os
import psutil  # Library untuk mengelola proses dan
                                status sistem

# Fungsi untuk membuat file
def create_file():
    filename = input("Masukkan nama file: ")
    content = input("Masukkan isi file: ")
    with open(filename, "w") as file:
        file.write(content)
    print(f"File '{filename}' berhasil dibuat.")

# Fungsi untuk menampilkan daftar file
def list_files():
    files = os.listdir()
    print("Daftar file di direktori saat ini:")
    for file in files:
        print(f"- {file}")

# Fungsi untuk menghapus file
def delete_file():
    filename = input("Masukkan nama file yang ingin
                                    dihapus: ")
    if os.path.exists(filename):
        os.remove(filename)
        print(f"File '{filename}' berhasil dihapus.")
    else:
        print("File tidak ditemukan.")

# Fungsi untuk menampilkan daftar proses
def list_processes():
    print("Proses yang sedang berjalan:")
    for proc in psutil.process_iter(['pid', 'name']):
        print(f"PID: {proc.info['pid']}, Nama: {proc.info
                                        ['name']}")

# Fungsi untuk menghentikan proses berdasarkan PID
def kill_process():
    pid = int(input("Masukkan PID proses yang ingin
                                    dihentikan: "))
    try:
```

```
            p = psutil.Process(pid)
            p.terminate()
            print(f"Proses dengan PID {pid} berhasil
                                            dihentikan.")
        except psutil.NoSuchProcess:
            print("Proses tidak ditemukan.")

    # Fungsi untuk menampilkan status sistem
    def system_status():
        memory = psutil.virtual_memory()
        storage = psutil.disk_usage('/')
        print(f"Status Memori: {memory.percent}% digunakan")
        print(f"Ruang Penyimpanan: {storage.percent}%
                                        digunakan")

    # Fungsi utama CLI
    def cli():
        while True:
            print("\n=== CLI Sederhana ===")
            print("1. Buat file")
            print("2. Tampilkan daftar file")
            print("3. Hapus file")
            print("4. Tampilkan daftar proses")
            print("5. Hentikan proses")
            print("6. Tampilkan status sistem")
            print("7. Keluar")
            choice = input("Pilih opsi (1-7): ")

            if choice == '1':
                create_file()
            elif choice == '2':
                list_files()
            elif choice == '3':
                delete_file()
            elif choice == '4':
                list_processes()
            elif choice == '5':
                kill_process()
            elif choice == '6':
                system_status()
            elif choice == '7':
                print("Keluar dari CLI.")
                break
            else:
                print("Pilihan tidak valid, silakan coba lagi
```

```
                                                    .")

    # Memulai CLI
    cli()
```

4.4.6 Penjelasan Kode

- **create_file**: Fungsi ini membuat file baru berdasarkan nama dan isi yang dimasukkan oleh pengguna.
- **list_files**: Menampilkan semua file di direktori saat ini.
- **delete_file**: Menghapus file yang dipilih pengguna.
- **list_processes**: Menampilkan daftar proses yang berjalan di sistem.
- **kill_process**: Menghentikan proses yang dipilih berdasarkan **PID**.
- **system_status**: Menampilkan informasi penggunaan memori dan ruang penyimpanan sistem.

4.4.7 Output Program CLI Sederhana

Berikut adalah contoh output dari program CLI sederhana:

1. **Membuat File**

```
=== CLI Sederhana ===
1. Buat file
2. Tampilkan daftar file
3. Hapus file
4. Tampilkan daftar proses
5. Hentikan proses
6. Tampilkan status sistem
7. Keluar
Pilih opsi (1-7): 1
Masukkan nama file: contoh.txt
Masukkan isi file: Ini adalah contoh isi file.
File 'contoh.txt' berhasil dibuat.
```

2. **Menampilkan Daftar File**

```
=== CLI Sederhana ===
1. Buat file
2. Tampilkan daftar file
3. Hapus file
4. Tampilkan daftar proses
5. Hentikan proses
6. Tampilkan status sistem
7. Keluar
Pilih opsi (1-7): 2
Daftar file di direktori saat ini:
- contoh.txt
```

3. **Menghapus File**

```
=== CLI Sederhana ===
1. Buat file
2. Tampilkan daftar file
3. Hapus file
4. Tampilkan daftar proses
5. Hentikan proses
6. Tampilkan status sistem
7. Keluar
Pilih opsi (1-7): 3
Masukkan nama file yang ingin dihapus: contoh.txt
File 'contoh.txt' berhasil dihapus.
```

4. **Menampilkan Daftar Proses**

```
=== CLI Sederhana ===
1. Buat file
2. Tampilkan daftar file
3. Hapus file
4. Tampilkan daftar proses
5. Hentikan proses
```

```
6. Tampilkan status sistem
7. Keluar
Pilih opsi (1-7): 4
Proses yang sedang berjalan:
PID: 1234, Nama: python
PID: 5678, Nama: bash
```

5. **Menghentikan Proses**

```
=== CLI Sederhana ===
1. Buat file
2. Tampilkan daftar file
3. Hapus file
4. Tampilkan daftar proses
5. Hentikan proses
6. Tampilkan status sistem
7. Keluar
Pilih opsi (1-7): 5
Masukkan PID proses yang ingin dihentikan: 1234
Proses dengan PID 1234 berhasil dihentikan.
```

6. **Menampilkan Status Sistem**

```
=== CLI Sederhana ===
1. Buat file
2. Tampilkan daftar file
3. Hapus file
4. Tampilkan daftar proses
5. Hentikan proses
6. Tampilkan status sistem
7. Keluar
Pilih opsi (1-7): 6
Status Memori: 45.0% digunakan
Ruang Penyimpanan: 60.0% digunakan
```

7. **Keluar dari CLI**

```
=== CLI Sederhana ===
1. Buat file
2. Tampilkan daftar file
3. Hapus file
4. Tampilkan daftar proses
5. Hentikan proses
6. Tampilkan status sistem
7. Keluar
Pilih opsi (1-7): 7
Keluar dari CLI.
```

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.

References

[1] GeeksforGeeks. (n.d.). *Process states in operating system*. Diakses pada 30 September 2024, dari `https://www.geeksforgeeks.org/process-states-in-operating-system/`

[2] Tutorialspoint. (n.d.). *Operating system - Process control block*. Diakses pada 30 September 2024, dari `https://www.tutorialspoint.com/operating_system/os_process_control_block.htm`

[3] Wikipedia contributors. (n.d.). Context switch. Dalam *Wikipedia, The Free Encyclopedia*. Diakses pada 30 September 2024, dari `https://en.wikipedia.org/wiki/Context_switch`

[4] Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* (10th ed.). Wiley. Diakses pada 30 September 2024, dari `https://www.wiley.com/en-us/Operating+System+Concepts%2C+10th+Edition-p-9781119471158`