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
3.5 Scheduling Algorithms 6
3.6 Process Creation and Termination 6
3.7 Introduction to Threads 6
3.8 File Systems 6
3.9 Input dan Output Management 7
3.9.1 Apa itu Manajemen I/O? 7
3.9.2 Fungsi Utama Modul I/O 7
3.9.3 Struktur Modul I/O 9
3.9.4 Control dan Timing 9
3.10 Deadlock Introduction and Prevention 10
3.11 User Interface Management 10
3.12 Virtualization in Operating Systems 11

4 Assignments and Practical Work **11**
4.1 Assignment 1: Process Scheduling 11
4.1.1 Group 1 11
4.1.2 12
4.1.3 12
4.1.4 12
4.1.5 12
4.1.6 12
4.1.7 12
4.1.8 12
4.1.9 Group 9 12
4.2 Assignment 2: Deadlock Handling 15

4.2.1 . . . 15
4.2.2 . . . 15
4.2.3 . . . 15
4.2.4 . . . 15
4.2.5 . . . 15
4.2.6 . . . 15
4.2.7 . . . 15
4.2.8 . . . 15
4.2.9 Group 9 . . . 15
4.3 Assignment 3: Multithreading and Amdahl's Law . . . 19
4.4 Assignment 4: Simple Command-Line Interface (CLI) for User Interface Management . . . 20
4.5 Assignment 5: File System Access . . . 20

5 Conclusion **20**

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

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

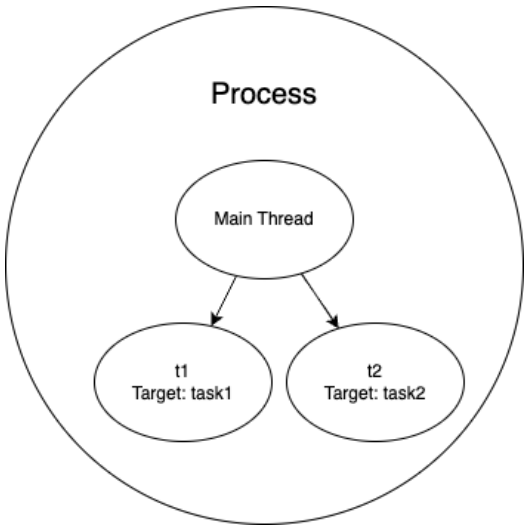

Figure 1: Ini adalah gambar contoh dari multithreading.

3.9 Input dan Output Management

3.9.1 Apa itu Manajemen I/O?

Manajemen I/O merupakan entitas yang bertanggung jawab mengontrol perangkat eksternal dan untuk pertukaran data antarperangkat tersebut dengan memori dan CPU. Perangkat eksternal itu seperti *keyboard, mouse,* dan lain sebagainya.

3.9.2 Fungsi Utama Modul I/O

Fungsi utama modul I/O dikategorikan menjadi lima, yaitu:

- *Control dan timing*: Modul I/O berfungsi untuk mengatur aliran data antara sistem inti komputer (seperti CPU dan memori) dan perangkat eksternal. Fungsi ini membutuhkan *control dan timing* yang tepat karena ada perbedaan kecepatan dan mekanisme antara CPU dan perangkat eksternal yang biasanya lebih lambat. Modul I/O harus memastikan bahwa data dikirim dan diterima sesuai dengan waktu yang tepat, tanpa menyebabkan konflik atau kerusakan data.

- Komunikasi CPU: Modul I/O harus mampu berkomunikasi baik dengan CPU maupun dengan perangkat eksternal. Komunikasi ini men-

cakup beberapa aspek:

  - *Command Decoding* (Dekode Perintah): Modul I/O menerima perintah dari CPU, yang biasanya dikirim melalui sinyal pada *bus control*. Contoh perintah untuk modul I/O pada disk adalah seperti *READ SECTOR, WRITE SECTOR, SEEK* nomor *track, dan SCAN record ID.* Beberapa perintah ini mungkin menyertakan parameter yang dikirim melalui bus data.
  - Data: Data ditransfer antara CPU dan modul I/O melalui bus data. Ini adalah proses pertukaran data yang terjadi setelah perintah diterima dan diproses oleh modul I/O.
  - *Status Reporting* (Laporan Status): Karena perangkat eksternal sering kali lebih lambat daripada CPU, modul I/O harus melaporkan statusnya kepada CPU, misalnya dengan sinyal status seperti *BUSY* (sibuk) atau *READY* (siap). Ini penting agar CPU tahu apakah modul I/O siap untuk melakukan operasi atau sedang sibuk dengan perintah lain.
  - *Address Recognition* (Pengenalan Alamat): Sama seperti memori memiliki alamat unik, perangkat I/O juga memiliki alamat yang unik. Modul I/O harus mampu mengenali alamat unik dari perangkat-perangkat yang dikendalikannya untuk mengarahkan data ke perangkat yang benar.

- Komunikasi Perangkat: Modul I/O juga harus bisa berkomunikasi langsung dengan perangkat eksternal. Komunikasi ini meliputi:
  - Perintah (instruksi yang dikirim ke perangkat).
  - Informasi Status (kondisi perangkat, seperti siap atau sibuk).
  - Data (informasi yang ditransfer antara perangkat dan CPU).

- *Data Buffering*: Modul I/O bertugas untuk menyiapkan data dari atau untuk perangkat eksternal melalui *buffering* (penyimpanan sementara data) sehingga data dapat diproses dengan lebih efisien, mengatasi perbedaan kecepatan antara CPU dan perangkat eksternal.

- Deteksi Kesalahan: Modul I/O juga bertanggung jawab untuk mendeteksi dan melaporkan kesalahan yang terjadi saat komunikasi dengan perangkat eksternal. Kesalahan ini bisa berupa masalah mekanis

(seperti kertas yang menggulung pada printer) atau kesalahan elektris (seperti kesalahan bit saat mentransfer data). Modul I/O kemudian akan melaporkan kesalahan ini ke CPU agar dapat diatasi.

3.9.3 Struktur Modul I/O

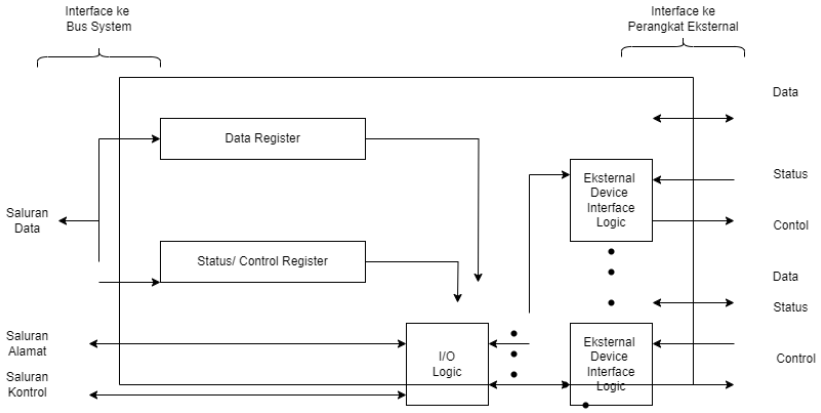

Figure 2: Struktur Modul I/O

Penjelasan:

- Data, alamat, dan kontrol dikirim dari sistem komputer (*bus system*) ke modul I/O.
- *I/O Logic* mengelola aliran data, kontrol, dan alamat yang masuk.
- *Data Register* menyimpan data sementara, dan *Status/Control Register* mengatur status dan kontrol perangkat.
- *External Device Interface Logic* menjembatani data, status, dan kontrol ke perangkat eksternal.
- Data, status, dan kontrol diolah dan dikirim ke perangkat eksternal, dan informasi status kembali ke sistem komputer.

3.9.4 Control dan Timing

Control dan Timing berfungsi untuk mengatur agar kecepatan transfer, data yang berbeda-beda antar periferal dapat tersinkronisasi. Misalnya, kontrol

pemindahan data dari sebuah perangkat eksternal ke CPU. Contoh kontrol pemindahan data dari sebuah perangkat eksternal ke CPU meliputi langkah-langkah berikut:

1. CPU meminta modul I/O untuk memeriksa status perangkat yang terhubung.
2. Modul I/O memberikan jawabannya tentang status perangkat.
3. Bila perangkat sedang beroperasi dan berada dalam keadaan siap untuk mengirimkan, maka CPU meminta pemindahan data, dengan menggunakan perintah tertentu ke modul I/O.
4. Modul I/O akan memperoleh unit data (misalnya, 8 atau 16 bit) dari perangkat eksternal.
5. Data akan dipindahkan dari modul I/O ke CPU.

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

4.1.3

4.1.4

4.1.5

4.1.6

4.1.7

4.1.8

4.1.9 Group 9

- Soal : Diberikan sebuah skenario yang terdapat beberapa proses yang ingin dijalankan oleh CPU. Setiap proses memiliki waktu kedatangan (*arrival time*) dan waktu burst (*burst time*). Hitunglah rata-rata waktu tunggu (*average waiting time*) dari proses-proses tersebut menggunakan tiga algoritma penjadwalan CPU, yaitu:
  - *First Come First Serve* (FCFS)
  - *Shortest Job Next* (SJN)
  - *Round Robin* (RR) dengan quantum sebesar 4

Proses	***burst_times***	***waiting_times***
1	10	0
2	5	1
3	8	2
4	6	3
5	2	4

Table 2: Tabel Data Proses

- Penyelesaian Kode Python :

```
# Fungsi untuk menghitung waktu tunggu rata-rata
def hitung_rata2_waktu(n, burst_times, waiting_times):
    total_wt = sum(waiting_times)
    rata2_wt = total_wt / n
    return rata2_wt
```

```
# Algoritma FCFS
def fcfs(proses, n, burst_times, waktu_kedatangan):
    waiting_times = [0] * n

    # Menghitung waktu tunggu
    for i in range(1, n):
        waiting_times[i] = waiting_times[i - 1] +
                                        burst_times[i - 1]

    rata2_wt = hitung_rata2_waktu(n, burst_times,
                                    waiting_times)
    return rata2_wt

# Algoritma SJN
def sjn(proses, n, burst_times, waktu_kedatangan):
    # Urutkan proses berdasarkan waktu burst
    proses_urut = sorted(zip(proses, burst_times,
                                    waktu_kedatangan), key
                                    =lambda x: (x[1], x[2]
                                    ))

    waiting_times = [0] * n
    waktu_sekarang = 0

    for i in range(n):
        id_proses, burst, kedatangan = proses_urut[i]

        if waktu_sekarang < kedatangan:
            waktu_sekarang = kedatangan

        waiting_times[id_proses - 1] = waktu_sekarang -
                                        kedatangan
        waktu_sekarang += burst

    rata2_wt = hitung_rata2_waktu(n, burst_times,
                                    waiting_times)
    return rata2_wt

# Algoritma Round Robin
def round_robin(proses, n, burst_times, waktu_kedatangan,
                                kuantum):
    sisa_burst = burst_times[:]
    waiting_times = [0] * n
    waktu_sekarang = 0
    selesai = False
```

```
    while not selesai:
        selesai = True
        for i in range(n):
            if sisa_burst[i] > 0:
                selesai = False
                if sisa_burst[i] > kuantum:
                    waktu_sekarang += kuantum
                    sisa_burst[i] -= kuantum
                else:
                    waktu_sekarang += sisa_burst[i]
                    waiting_times[i] = waktu_sekarang -
                                                    burst_times
                                                    [i]
                    sisa_burst[i] = 0

    rata2_wt = hitung_rata2_waktu(n, burst_times,
                                  waiting_times)
    return rata2_wt

# Kode utama
def main():
    # Data proses yang diberikan
    proses = [1, 2, 3, 4, 5]
    burst_times = [10, 5, 8, 6, 2]
    waktu_kedatangan = [0, 1, 2, 3, 4]
    kuantum = 4

    n = len(proses)

    # Menjalankan setiap algoritma
    rata2_wt_fcfs = fcfs(proses, n, burst_times,
                                  waktu_kedatangan)
    rata2_wt_sjn = sjn(proses, n, burst_times,
                                  waktu_kedatangan)
    rata2_wt_rr = round_robin(proses, n, burst_times,
                                  waktu_kedatangan,
                                  kuantum)

    # Menampilkan hasil
    print("Algoritma FCFS")
    print(f"Rata-rata Waktu Tunggu: {rata2_wt_fcfs:.2f}\n
                                  ")

    print("Algoritma SJN")
```

```
    print(f"Rata-rata Waktu Tunggu: {rata2_wt_sjn:.2f}\n"
                                    )

    print("Algoritma Round Robin")
    print(f"Rata-rata Waktu Tunggu: {rata2_wt_rr:.2f}\n")

if __name__ == "__main__":
    main()

# Output:
# Algoritma FCFS
# Rata-rata Waktu Tunggu: 15.40

# Algoritma SJN
# Rata-rata Waktu Tunggu: 10.60

# Algoritma Round Robin
# Rata-rata Waktu Tunggu: 19.40
```

4.2 Assignment 2: Deadlock Handling

In this assignment, students were asked to simulate different deadlock scenarios and explore various prevention methods.

4.2.1

4.2.2

4.2.3

4.2.4

4.2.5

4.2.6

4.2.7

4.2.8

4.2.9 Group 9

- Soal : Kondisi awal:

- Proses A membutuhkan R1 untuk mulai berjalan dan menunggu R2 untuk menyelesaikan.
- Proses B membutuhkan R2 untuk mulai berjalan dan menunggu R1 untuk menyelesaikan.

Gunakan Algoritma Banker's untuk mengecek apakah sistem dalam keadaan aman atau terjadi deadlock, dengan data berikut:

1. Proses A dan Proses B membutuhkan masing-masing maksimal 3 unit dari R1 dan R2.
2. Saat ini, Proses A telah dialokasikan 1 unit dari R1 dan 2 unit dari R2.
3. Proses B telah dialokasikan 2 unit dari R1 dan 1 unit dari R2.
4. Tersisa 2 unit R1 dan 1 unit R2 di sistem.

- Penyelesaian Kode Python:

```
# Fungsi untuk mensimulasikan skenario deadlock
def simulasi_deadlock():
    print("Mensimulasikan Skenario Deadlock...")
    # Dua sumber daya dan dua proses
    sumber_daya = [1, 1]  # Sumber daya yang tersedia (R1
                                dan R2)
    proses_A = [1, 0]  # Proses A awalnya membutuhkan
                                sumber daya R1
    proses_B = [0, 1]  # Proses B awalnya membutuhkan
                                sumber daya R2

    # Kedua proses mencoba untuk mendapatkan sumber daya
    if sumber_daya[0] >= proses_A[0]:
        sumber_daya[0] -= proses_A[0]
        print("Proses A mendapatkan Sumber Daya 1")

    if sumber_daya[1] >= proses_B[1]:
        sumber_daya[1] -= proses_B[1]
        print("Proses B mendapatkan Sumber Daya 2")

    # Sekarang kedua proses mencoba untuk mendapatkan
                                sumber daya satu sama
                                lain
    if sumber_daya[1] >= 1 and sumber_daya[0] < 1:
```

```
        print("Proses A menunggu Sumber Daya 2, tetapi
                                    dipegang oleh
                                    Proses B")

    if sumber_daya[0] >= 1 and sumber_daya[1] < 1:
        print("Proses B menunggu Sumber Daya 1, tetapi
                                    dipegang oleh
                                    Proses A")

    print("Deadlock telah terjadi!\n")

# Algoritma Banker's untuk pencegahan deadlock
def algoritma_bankir(proses, sumber_daya_maks,
                             sumber_daya_alokasi,
                             sumber_daya_tersedia):
    print("Menerapkan Algoritma Banker's...")
    n = len(proses)  # Jumlah proses
    m = len(sumber_daya_maks[0])  # Jumlah jenis sumber
                                  daya

    # Menghitung matriks kebutuhan
    kebutuhan = [[0] * m for _ in range(n)]
    for i in range(n):
        for j in range(m):
            kebutuhan[i][j] = sumber_daya_maks[i][j] -
                                        sumber_daya_alokasi
                                        [i][j]

    selesai = [False] * n  # Melacak apakah proses telah
                                  selesai
    urutan_aman = []    # Daftar untuk menyimpan urutan
                                  aman jika ditemukan

    while len(urutan_aman) < n:
        dialokasikan = False
        for i in range(n):
            # Memeriksa apakah proses dapat dieksekusi
                                        dengan aman
            if not selesai[i] and all(kebutuhan[i][j] <=
                                        sumber_daya_tersedia
                                        [j] for j in
                                        range(m)):
                # Jika bisa, tambahkan sumber daya yang
                                            dialokasikan
                                             kembali
```

```
                                                ke sumber
                                                daya
                                                tersedia
                for k in range(m):
                    sumber_daya_tersedia[k] +=
                                                    sumber_daya_alokasi
                                                    [i][k]

                # Tandai proses sebagai selesai dan
                                                tambahkan
                                                ke urutan
                                                aman
                urutan_aman.append(proses[i])
                selesai[i] = True
                dialokasikan = True
                print(f"Proses {proses[i]} dieksekusi
                                                dengan
                                                aman")

        # Jika tidak ada proses yang bisa dieksekusi di
                                        iterasi ini, ada
                                        deadlock
        if not dialokasikan:
            print("Deadlock terdeteksi oleh Algoritma
                                            Banker's -
                                            Sistem tidak
                                            dalam keadaan
                                            aman\n")
            return False

    print("Sistem dalam keadaan aman. Urutan aman:",
                                    urutan_aman, "\n")
    return True

# Kode Utama
def main():
    print("Memulai Simulasi Deadlock dan Pencegahan
                                    Menggunakan Algoritma
                                    Banker's...\n")

    # Mensimulasikan skenario deadlock
    simulasi_deadlock()

    # Menerapkan Algoritma Banker's untuk mencegah
                                    deadlock
```

```
    proses = ["P1", "P2"]
    sumber_daya_maks = [[3, 3], [4, 2]]  # Sumber daya
                                         maksimum yang
                                         dibutuhkan oleh setiap
                                          proses
    sumber_daya_alokasi = [[1, 2], [2, 1]]  # Sumber daya
                                          yang saat ini
                                         dialokasikan ke setiap
                                          proses
    sumber_daya_tersedia = [2, 1]  # Total sumber daya
                                         yang tersedia

    algoritma_bankir(proses, sumber_daya_maks,
                                         sumber_daya_alokasi,
                                         sumber_daya_tersedia)

if __name__ == "__main__":
    main()

# Output:
# Memulai Simulasi Deadlock dan Pencegahan Menggunakan
                                Algoritma Banker's...

# Mensimulasikan Skenario Deadlock...
# Proses A mendapatkan Sumber Daya 1
# Proses B mendapatkan Sumber Daya 2
# Deadlock telah terjadi!

# Menerapkan Algoritma Banker's...
# Proses P1 dieksekusi dengan aman
# Proses P2 dieksekusi dengan aman
# Sistem dalam keadaan aman. Urutan aman: ['P1', 'P2']
```

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

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.