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

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

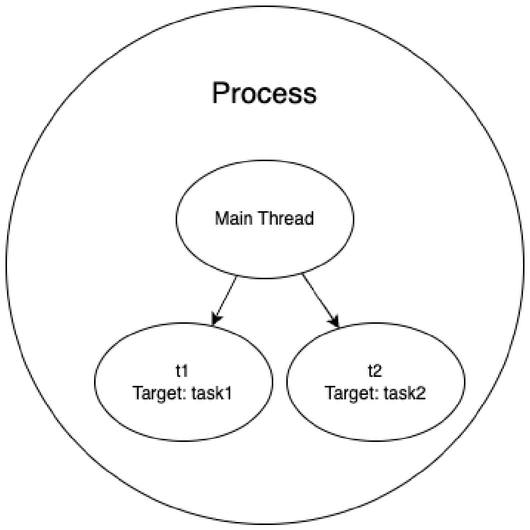

Figure 1: Ini adalah gambar contoh dari multithreading.

3.9 Input 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
    1. Pengertian Deadlock
    2. Kondisi Terjadinya Deadlock
    3. Model Deadlock
    4. Deteksi Dan Pemulihan Deadlock
       Untuk mendeteksi deadlock tanpa ada pencegahan, ada 2 cara yaitu:
        - Deteksi deadlock dengan satu sumber tiap tipe

Sistem ini hanya mungkin untuk memiliki satu bluRay Record, satu plotter, dan satu tape drive pada setiap tipe.

– Deteksi deadlock dengan banyak sumber daya pada tiap tipe

  Pendeteksian sistem ini dengan penyajian algoritma berbasis matriks untuk menangani deadlock, contohnya menggunakan algoritma safety.

Algoritma Safety

Algoritma Safety, sama halnya dengan algoritma banker, merupakan algoritma penanganan deadlock. Namun, algoritma safety ini lebih kepada memastikan apakah suatu sistem dapat dikatakan aman (safe state) atau tidak aman (unsafe state). Algoritma ini cukup sederhana, hanya mengecek kondisi dimana sistem dikatakan dalam keadaan aman (safe state) maupun tidak (unsafe state).

Untuk rumusnya yaitu:

$$\begin{aligned} \text{If} \quad & \text{Need} \leq \text{Available} \\ \text{Then} \quad & \text{execute process} \\ & \text{new Available} = \text{Available} + \text{Allocation} \\ \text{Else} \quad & \text{do not execute go forward} \end{aligned}$$

atau implementasi ke dalam kode Python:

```
class Process:
    def is_safe(available, max,
                                    allocation
                                    ):
        num_processes = len(max)
        num_resources = len(available)

        work = available[:]
        finish = [False] * num_processes
        safe_sequence = []

        while len(safe_sequence) <
                                        num_processes
                                        :
            made_progress = False
```

```
                    for i in range(num_processes)
                                                     :

                        if not finish[i] and all(
                                                        need
                                                        [
                                                        i
                                                        ]
                                                        [
                                                        j
                                                        ]

                                                        <
                                                        =

                                                        work
                                                        [
                                                        j
                                                        ]

                                                        for

                                                        j

                                                        in

                                                        range
                                                        (
                                                        num_resource
                                                        )
                                                        )
                                                        :

                            # If Need <=
                                                          Availabl

                            print(f"Process {i}
                                                          can

                                                          be

                                                          executed
                                                          .

                                                          Need
```

```
:

{
need
[
i
]
}
,

Availabl
:

{
work
}
"
)

# Execute the process
work = [work[j] +

allocati
[
i
]
[
j
]

for

j

in

range
(
num_reso
)
]

finish[i] = True
safe_sequence.append(

i
)
```

```
                            made_progress = True
                            print(f"Process {i}
                                                    executed
                                                    .
                                                    New
                                                    Availabl
                                                    :
                                                    {
                                                    work
                                                    }
                                                    "
                                                    )
                        if not made_progress:
                            print("Do not execute go
                                                forward
                                                "
                                                )
                            return False, []
                    print("System is in a safe state.
                                              ")
                    print(f"Safe sequence: {
                                              safe_sequence
                                              }"
                                              )
                    return True, safe_sequence
```

Contoh soalnya:

Sebutkan urutan prosesnya berdasarkan algoritma safety! Cek sistem safe atau nonsafe!

Jawab:

P0 → *need available*
743 332
do not execute P0 go forward
P1 → *need available*

Proses	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2	7	4	3
P1	2	0	0	3	2	2				1	2	2
P2	3	0	2	9	0	2				6	0	0
P3	2	1	1	2	2	2				4	3	1
P4	0	0	2	4	3	3				4	3	1

Table 1: Tabel Algoritma Safety

122 332

Execute P1

new $available = available + allocation = 332 + 200 \rightarrow 532$

P2 $\rightarrow$ *need available*

600 532

do not execute P2 go forward

P3 $\rightarrow$ *need available*

011 532

Execute P3

new $available = available + allocation = 532 + 211 \rightarrow 743$

P4 $\rightarrow$ *need available*

431 743

Execute P4

new $available = available + allocation = 743 + 002 \rightarrow 745$

P0 $\rightarrow$ *need available*

743 745

Execute P0

new $available = available + allocation = 745 + 010 \rightarrow 755$

Proses	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2	7	4	3
P1	2	0	0	3	2	2	5	3	2	1	2	2
P2	3	0	2	9	0	2	7	4	3	6	0	0
P3	2	1	1	2	2	2	7	4	5	4	3	1
P4	0	0	2	4	3	3	7	5	5	4	3	1

Table 2: Tabel Algoritma Safety

Urutan eksekusi proses: ¡ P_1, P_3, P_4, P_0, P_2 ¿
Kesimpulan: Sistem dalam keadaan *safe state* dan memenuhi kriteria *safety.*

Pemulihan dari deadlock memerlukan strategi khusus yang dirancang untuk mengatasi situasi ini, memungkinkan sistem untuk kembali ke keadaan operasional normal. Pendekatan pemulihan deadlock umumnya melibatkan identifikasi proses yang terlibat, evaluasi dampak dari menghentikan atau mengalihkan proses tersebut, serta penerapan langkah-langkah yang dapat memulihkan sistem tanpa kehilangan integritas data. Adapun pemulihan deadlock dengan 3 cara yaitu:

– *Preempetion* (Pencabutan Sumber Daya)

  Sistem ini memungkinkan untuk mengambil sumberdaya yang ada kemudian memberikannya ke proses yang lain. Namun proses ini biasanya lebih sulit diterapkan karena diperlukan penangguhan proses. Proses ini mungkin membutuhkan intervesi manual terutama sistem operati batch pada sistem.

– *Rollback* (Kembali ke titik sebelumnya)

  Apabila terdeteksi adanya deadlock maka sistem dapat melakukan checkpointed secara berkala. Maksud dari checkpointing sendiri adalah status ditulis ulang pada file agar dapat direload nanti, proses ini dapat berupa gambar gambar memory dan status sumberdaya. Apabila terdapat data yang sama dengan yang sudah ada, maka tidak ditulis. Penulisa hanya dilakukan apabila data masih baru. Apabila terjadi deadlock proses akan dikembalikan sebelum sumberdaya terjadi deadlock.

  - *Terminating Processes* (Menghentikan proses)

    Cara yang lebih drastis untuk mengatasi deadlock adalah dengan menghentikan satu atau lebih proses yang terlibat. Untuk itu, sistem harus secara berkala menyimpan checkpoint dari status proses dan sumber daya. Ketika deadlock terdeteksi, sistem akan melakukan rollback ke checkpoint terakhir yang aman. Proses yang dihentikan akan dipilih berdasarkan kriteria seperti prioritas, waktu yang telah digunakan, atau sumber daya yang telah dialokasikan.

- 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

```python
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
```

4.2 Assignment 2: Deadlock Handling

Dalam soal ini mahasiswa diminta untuk menghitung kondisi ketersediaan sumber daya (available) untuk setiap proses dan mengevaluasi apakah proses tersebut dapat dieksekusi tanpa mengakibatkan deadlock.

4.2.1 Group 10

Hitunglah available selanjutnya dan tentukan proses urutan eksekusi proses menggunakan Algoritma Safety!

Proses	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	1	0	2	7	5	3	2	3	1	6	5	1
P1	2	1	0	3	2	2				1	1	2
P2	3	0	3	9	0	4				6	0	1
P3	2	1	1	4	2	2				2	1	1

Table 3: Tabel Algoritma Safety

Jawab:
Untuk menyelesaikan soal tersebut, kita bisa gunakan rumus algoritma safety

$$\begin{aligned} \text{If} \quad & \text{Need} \leq \text{Available} \\ \text{Then} \quad & \text{execute process} \\ & \text{new Available} = \text{Available} + \text{Allocation} \\ \text{Else} \quad & \text{do not execute go forward} \end{aligned}$$

atau implementasi ke dalam kode python

```
class Process:
def is_safe(available, max, allocation):
num_processes = len(max)
num_resources = len(available)

work = available[:]
finish = [False] * num_processes
safe_sequence = []

while len(safe_sequence) < num_processes:
    made_progress = False
    for i in range(num_processes):
        if not finish[i] and all(need[i][j] <= work[j]
                                 for j in range(
                                 num_resources)):
            # If Need <= Available
            print(f"Process {i} can be executed. Need: {
                                        need[i]},
                                        Available: {
                                        work}")
            # Execute the process
            work = [work[j] + allocation[i][j] for j in
                                        range(
                                        num_resources)
                                        ]
            finish[i] = True
            safe_sequence.append(i)
            made_progress = True
            print(f"Process {i} executed. New Available:
                                        {work}")

    if not made_progress:
        print("Do not execute go forward")
        return False, []
print("System is in a safe state.")
print(f"Safe sequence: {safe_sequence}")
return True, safe_sequence
```

Maka,

$$
\begin{aligned}
P0 \quad &\rightarrow \quad \text{Need} \leq \text{Available} \\
& 651 \leq 231 \\
& \text{do not execute P0, go forward} \\
P1 \quad &\rightarrow \quad \text{Need} \leq \text{Available} \\
& 112 \leq 231 \\
& \text{Execute P1} \\
& \text{new Available} = \text{Available} + \text{Allocation} = 231 + 210 \rightarrow 441 \\
P2 \quad &\rightarrow \quad \text{Need} \leq \text{Available} \\
& 601 \leq 441 \\
& \text{do not execute P2, go forward} \\
P3 \quad &\rightarrow \quad \text{Need} \leq \text{Available} \\
& 211 \leq 441 \\
& \text{Execute P3} \\
& \text{new Available} = \text{Available} + \text{Allocation} = 441 + 211 \rightarrow 652 \\
P0 \quad &\rightarrow \quad \text{Need} \leq \text{Available} \\
& 651 \leq 652 \\
& \text{Execute P0} \\
& \text{new Available} = \text{Available} + \text{Allocation} = 652 + 102 \rightarrow 754
\end{aligned}
$$

Proses	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	1	0	2	7	5	3	2	3	1	6	5	1
P1	2	1	0	3	2	2	4	4	1	1	1	2
P2	3	0	3	9	0	4	6	5	2	6	0	1
P3	2	1	1	4	2	2	7	5	4	2	1	1

Table 4: Tabel Algoritma Safety

Jadi, Urutan eksekusi Prosesnya adalah $\langle P_1, P_3, P_4, P_0, P_2 \rangle$. Dalam urutan ini proses tidak mengalami deadlock (safety)

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.