

Operating System Course Report - First Half of the Semester

A class

October 10, 2024

Contents

1	Introduction	3
2	Course Overview	3
2.1	Objectives	3
2.2	Course Structure	3
3	Topics Covered	4
3.1	Basic Concepts and Components of Computer Systems	4
3.2	System Performance and Metrics	4
3.3	System Architecture of Computer Systems	4
3.4	Process Description and Control	4
3.5	Scheduling Algorithms	5
3.6	Process Creation and Termination	5
3.6.1	<i>Process Creation</i>	5
3.6.2	Siapa yang Membuat Proses	11
3.6.3	Proses <i>Termination</i>	11
3.7	Introduction to Threads	12
3.8	File Systems	13
3.9	Input and Output Management	13
3.10	Deadlock Introduction and Prevention	13
3.11	User Interface Management	13
3.12	Virtualization in Operating Systems	14
4	Assignments and Practical Work	14
4.1	Assignment 1: Process Scheduling	14
4.1.1	Group 1	14
4.1.2	Group 6	15
4.2	Assignment 2: Deadlock Handling	18
4.3	Assignment 3: Multithreading and Amdahl's Law	18
4.4	Assignment 4: Simple Command-Line Interface (CLI) for User Interface Management	19
4.5	Assignment 5: File System Access	19
4.5.1	Group 6	19
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

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

3.6.1 *Process Creation*

- Definisi dan Konsep Dasar *Process Creation*

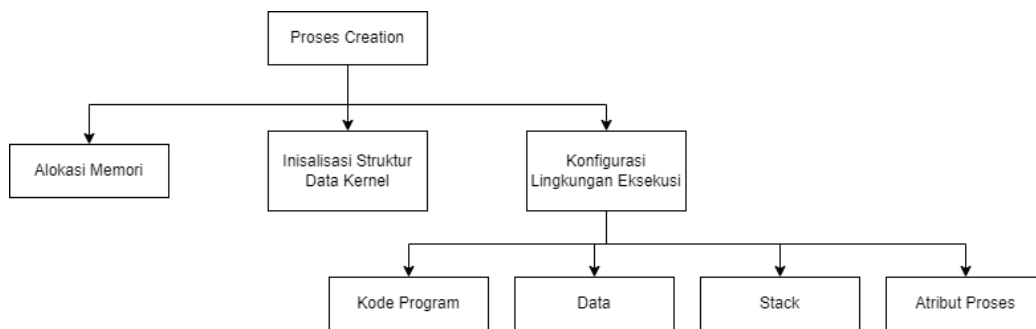


Figure 1: Diagram *Process Creation* dan Komponen Proses

Process Creation, atau penciptaan proses, adalah salah satu fungsi fundamental dalam sistem operasi modern. Ini merupakan mekanisme yang memungkinkan sistem operasi untuk membuat unit eksekusi baru, yang kita kenal sebagai proses, sehingga program dapat dijalankan di CPU. Dalam konteks sistem operasi, proses didefinisikan sebagai program yang sedang dieksekusi, yang terdiri atas kode program, data, *stack*, dan berbagai atribut yang dikelola oleh sistem operasi (Tanenbaum & Bos, 2015).

Ketika sebuah proses baru dibuat, sistem operasi harus melakukan serangkaian operasi kompleks untuk memastikan bahwa proses tersebut memiliki semua sumber daya yang diperlukan untuk eksekusi yang benar.

Ini termasuk alokasi ruang memori, inisialisasi struktur data *kernel*, dan konfigurasi lingkungan eksekusi. Proses penciptaan ini sangat penting karena menjadi fondasi bagi *multitasking* dan manajemen sumber daya dalam sistem operasi modern.

- Alasan Dibuatnya Proses

Ada beberapa alasan mengapa sistem operasi perlu membuat proses baru. Pertama, dan yang paling umum, adalah untuk menjalankan program baru. Ketika pengguna atau sistem meminta eksekusi sebuah program, sistem operasi harus membuat proses baru untuk mengakomodasi program tersebut. Ini melibatkan pembacaan file *executable*, alokasi memori, dan persiapan lingkungan eksekusi (Silberschatz et al., 2018).

Kedua, proses *creation* mendukung konsep *multiprogramming*. *Multiprogramming* memungkinkan beberapa program berjalan secara bersamaan (atau lebih tepatnya, bergantian dengan sangat cepat) pada sistem dengan satu atau lebih *core*. Ini meningkatkan efisiensi penggunaan *core* dan responsivitas sistem secara keseluruhan. Sistem operasi membuat dan mengelola beberapa proses, serta beralih di antara mereka dengan cepat untuk memberikan ilusi bahwa mereka berjalan secara bersamaan pada *core* yang ada.

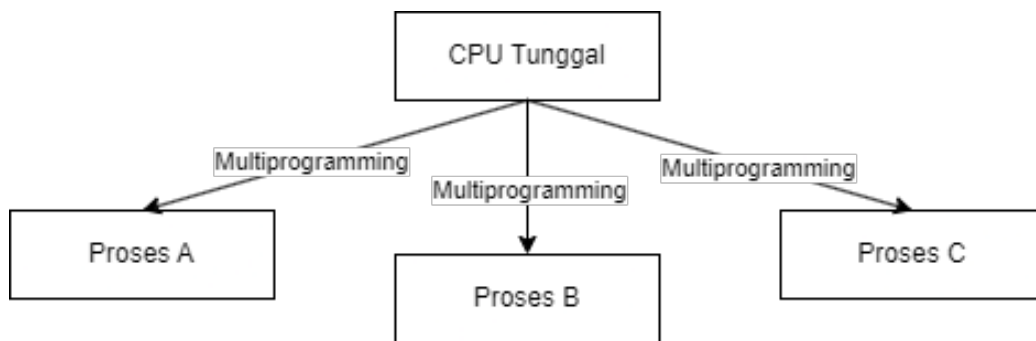


Figure 2: Diagram Ilustrasi *Multiprogramming*

Ketiga, proses *creation* memfasilitasi pemrosesan paralel pada sistem dengan beberapa *core*. Dalam skenario ini, sistem operasi dapat membuat beberapa proses yang benar-benar berjalan secara bersamaan pada *core* yang berbeda. Setiap *core* dapat menangani tugas secara

independen, sehingga meningkatkan *throughput* sistem secara signifikan dan memungkinkan eksekusi beberapa tugas secara simultan.

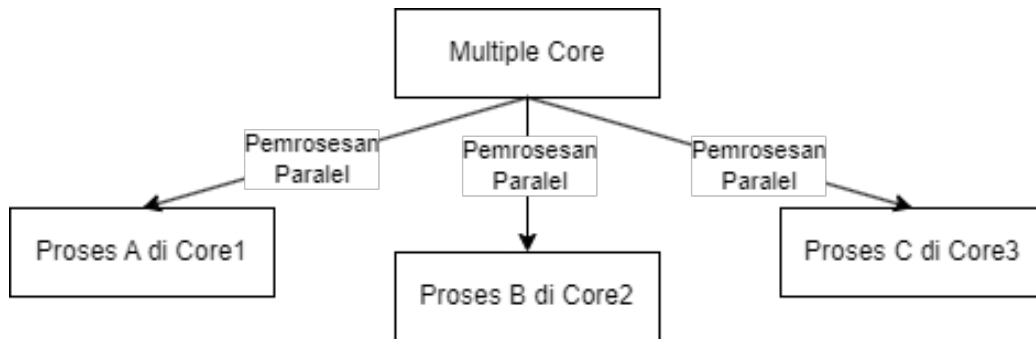


Figure 3: Diagram Ilustrasi Pemerrosesan Paralel

- Elemen Penting Dalam *Process Creation*

- ID Proses

Setiap proses yang dibuat oleh sistem operasi diberi sebuah pen-
genal unik yang disebut *Process ID* atau PID. PID ini sangat
penting untuk manajemen proses oleh sistem operasi. Menurut
Love (2010), di sistem Linux, PID diimplementasikan sebagai bilan-
gan bulat yang diincrement setiap kali proses baru dibuat. Ketika
nilai PID mencapai nilai maksimum (biasanya 32.768 pada banyak
sistem), sistem akan melakukan *wrap-around* dan mulai lagi dari
nilai terendah yang tersedia.

Implementasi PID melibatkan struktur data yang efisien untuk
alokasi dan dealokasi yang cepat. Misalnya, sistem Linux meng-
gunakan bitmap untuk melacak PID yang tersedia. Ini memung-
kinkan alokasi PID baru dalam waktu konstan, yang sangat pent-
ing mengingat frekuensi tinggi operasi pembuatan proses dalam
sistem modern.

- Pewarisan

Konsep pewarisan dalam konteks *process creation* mengacu pada
transfer atribut dan sumber daya dari proses induk ke proses anak.
Ini adalah aspek kunci dari model proses UNIX dan turunannya.
Ketika sebuah proses anak dibuat (misalnya melalui pemanggilan
sistem *fork()* di UNIX), ia mewarisi banyak atribut dari induknya.

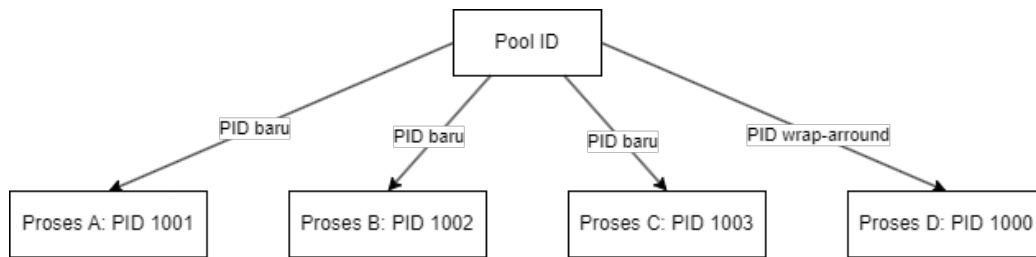


Figure 4: Diagram Alokasi PID

Atribut yang diwariskan biasanya meliputi prioritas penjadwalan, grup proses, direktori kerja, dan variabel lingkungan. Selain itu, proses anak juga mewarisi salinan dari tabel *file descriptor* induk, yang berarti ia memiliki akses ke file yang sama yang terbuka oleh induknya saat *fork()* terjadi (Bovet & Cesati, 2005).

Namun, penting untuk dicatat bahwa pewarisan ini tidak selalu berarti penyalinan langsung. Banyak sistem operasi modern menggunakan teknik *copy-on-write* untuk efisiensi. Dalam skema ini, proses anak awalnya berbagi halaman memori fisik yang sama dengan induknya. Hanya ketika salah satu proses mencoba memodifikasi sebuah halaman, salinan terpisah dibuat. Ini secara signifikan mengurangi *overhead* pembuatan proses.

– Alokasi Sumber daya

Alokasi sumber daya adalah aspek kritis dari *process creation*. Ketika sebuah proses baru dibuat, sistem operasi harus mengalokasikan berbagai sumber daya yang dibutuhkan proses untuk berfungsi. Ini meliputi memori, *file descriptor*, dan waktu CPU.

Alokasi memori melibatkan penciptaan ruang alamat virtual untuk proses baru. Pada sistem modern, ini biasanya melibatkan pengaturan struktur *page table* yang memetakan alamat virtual ke alamat fisik. Sistem operasi juga harus mengalokasikan memori untuk *stack* proses, yang digunakan untuk menyimpan variabel lokal dan informasi pemanggilan fungsi.

Menurut Arpaci-Dusseau (2018), alokasi sumber daya juga melibatkan inisialisasi berbagai struktur data *kernel* yang terkait dengan proses. Ini termasuk *Process Control Block* (PCB) atau *task_struct* di Linux, yang menyimpan semua informasi yang dibu-

tuhkan *kernel* untuk mengelola proses.

– Hubungan Induk-anak

Sistem operasi memelihara hubungan hierarkis antara proses, yang dikenal sebagai hubungan induk-anak. Ketika sebuah proses membuat proses baru, proses asli disebut induk, dan proses baru disebut anak. Hubungan ini penting untuk berbagai aspek manajemen proses, termasuk terminasi proses dan propagasi sinyal.

Di sistem berbasis UNIX, hubungan induk-anak diimplementasikan melalui *pointer* dalam struktur proses. Setiap proses memiliki *pointer* ke proses induknya, serta daftar proses anaknya. Ini memungkinkan penjelajahan yang efisien dari pohon proses, yang diperlukan untuk operasi seperti terminasi proses dan pembersihan sumber daya (Bovet & Cesati, 2005).

Hubungan induk-anak juga memainkan peran penting dalam penanganan sinyal dan pengumpulan status terminasi. Ketika sebuah proses berakhir, sistem operasi mengirim sinyal *SIGCHLD* ke proses induk, memungkinkannya untuk melakukan tindakan yang sesuai, seperti mengumpulkan status keluar dari proses anak atau membersihkan sumber daya.

• Tahapan *Process Creation*

Proses creation adalah operasi kompleks yang melibatkan beberapa tahap. Meskipun implementasi spesifik dapat bervariasi antar sistem operasi, tahapan umumnya adalah sebagai berikut:

– Permintaan Proses

Proses *creation* biasanya dimulai dengan permintaan eksplisit, baik dari pengguna (misalnya, menjalankan program dari *command line*) atau dari proses yang sudah berjalan (misalnya, server web yang membuat proses baru untuk menangani koneksi masuk). Dalam sistem berbasis UNIX, ini sering dipicu oleh pemanggilan sistem *fork()* atau salah satu variannya.

Ketika *fork()* dipanggil, sistem operasi pertama-tama memeriksa apakah ada sumber daya sistem yang cukup untuk mendukung proses baru. Ini termasuk pemeriksaan batas jumlah proses sistem dan batas per-pengguna. Jika sumber daya tidak mencukupi,

pemanggilan sistem akan gagal dengan kode kesalahan yang sesuai (Love, 2010).

– Penugasan PID

Jika sumber daya mencukupi, langkah berikutnya adalah mengalokasikan PID untuk proses baru. Seperti disebutkan sebelumnya, ini biasanya melibatkan pemilihan bilangan bulat berikutnya yang tersedia dari *pool* PID. Sistem operasi harus memastikan bahwa PID yang dipilih unik dan belum digunakan oleh proses lain yang sedang berjalan.

Di Linux, alokasi PID diimplementasikan dengan cara yang aman untuk *thread* (*thread-safe*) guna menghindari *race condition* di sistem multiprosesor. Ini melibatkan penggunaan struktur data *atomic* dan mekanisme *locking* untuk memastikan bahwa dua proses tidak pernah mendapatkan PID yang sama (Love, 2010).

– Alokasi Sumber Daya

Setelah PID dialokasikan, sistem operasi mulai mengalokasikan sumber daya yang diperlukan untuk proses baru. Ini adalah tahap yang kompleks dan melibatkan beberapa sub-langkah:

1. Alokasi struktur *kernel*: Sistem mengalokasikan dan menginisialisasi *Process Control Block* (PCB) atau ekuivalennya. PCB menyimpan semua informasi yang diperlukan sistem untuk mengelola proses, termasuk status proses, prioritas penjadwalan, dan *pointer* ke struktur data terkait lainnya.
2. Alokasi memori: Sistem membuat ruang alamat virtual untuk proses baru. Pada sistem modern, ini biasanya melibatkan pengaturan struktur *page table*. Jika proses baru adalah hasil dari *fork()*, sistem mungkin menggunakan teknik *copy-on-write* untuk efisiensi.
3. Inisialisasi *stack*: Sistem mengalokasikan dan menginisialisasi *stack* untuk proses baru. *Stack* digunakan untuk menyimpan variabel lokal, parameter fungsi, dan alamat kembali (*return address*).
4. Duplikasi deskriptor file: Jika proses baru adalah hasil dari *fork()*, sistem menduplikasi tabel *file descriptor* dari proses induk.

5. Inisialisasi penjadwalan: Sistem menginisialisasi struktur data penjadwalan untuk proses baru, termasuk pengaturan prioritas awal dan statistik penggunaan CPU.

- Pembuatan PCB

Process Control Block (PCB) adalah struktur data inti yang digunakan sistem operasi untuk melacak status dan sumber daya proses. Pembuatan PCB melibatkan inisialisasi berbagai *field* dengan nilai yang sesuai.

Pembuatan PCB adalah langkah kritis dalam proses *creation* karena struktur ini menjadi referensi utama sistem operasi untuk semua operasi yang melibatkan proses tersebut di masa mendatang.

Setelah semua tahapan ini selesai, proses baru siap untuk dijadwalkan dan dieksekusi. Pada titik ini, sistem operasi dapat memilih untuk segera menjadwalkan proses baru untuk eksekusi, atau menempatkannya dalam antrian proses yang siap, tergantung pada kebijakan penjadwalan dan beban sistem saat itu.

3.6.2 Siapa yang Membuat Proses

(Isi materi disini)

3.6.3 Proses *Termination*

- Definisi Proses *Termination*
(isi materi disini)
- Jenis-jenis *Termination*
(Isi materi disini)
- Apa yang Terjadi Setelah *Termination*
(isi materi disini)

References

- [1] Arpaci-Dusseau, R. H., & Arpaci-Dusseau, A. C. (2018). *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books.

- [2] Bovet, D. P., & Cesati, M. (2005). *Understanding the Linux Kernel* (3rd ed.). O'Reilly Media.
- [3] Love, R. (2010). *Linux Kernel Development* (3rd ed.). Addison-Wesley Professional.
- [4] Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* (10th ed.). John Wiley & Sons.
- [5] Tanenbaum, A. S., & Bos, H. (2015). *Modern Operating Systems* (4th ed.). Pearson.

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

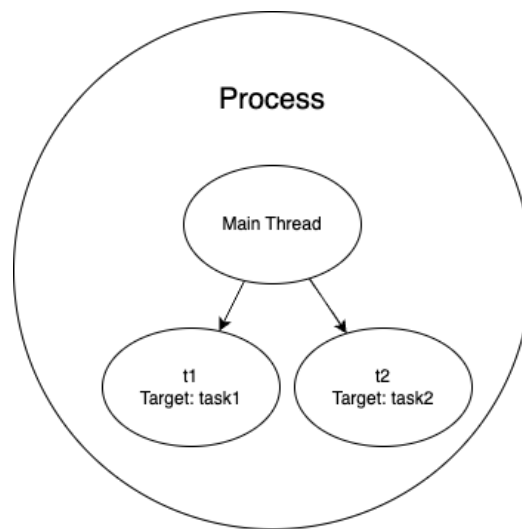


Figure 5: Ini adalah gambar contoh dari multithreading.

Seperti yang terlihat pada Gambar 5, 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 6

Soal:

Implementasikan algoritma penjadwalan proses FCFS (*First Come First Serve*), SJN (*Shortest Job Next*), dan RR (*Round Robin*) menggunakan Python. Bandingkan kinerja ketiga algoritma tersebut dengan menggunakan set data berikut:

Proses	Waktu Kedatangan	Waktu Burst
P1	0	10
P2	1	4
P3	3	2
P4	5	1

Table 2: Data Proses Untuk Penjadwalan

Untuk algoritma *Round Robin*, gunakan kuantum waktu dua unit. Hitung waktu tunggu rata-rata dan waktu penyelesaian rata-rata untuk setiap algoritma.

Jawaban:

```
class Process:
    def __init__(self, pid, arrival_time, burst_time):
        self.pid = pid
        self.arrival_time = arrival_time
        self.burst_time = burst_time
        self.remaining_time = burst_time
        self.completion_time = 0
        self.waiting_time = 0

    def fcfs(processes):
        time = 0
        for p in processes:
            if time < p.arrival_time:
                time = p.arrival_time
            p.waiting_time = time - p.arrival_time
            time += p.burst_time
            p.completion_time = time

    def sjn(processes):
        time = 0
        remaining = processes.copy()
        completed = []
```

```

while remaining:
    available = [p for p in remaining if p.
                  arrival_time <= time]
    if not available:
        time += 1
        continue

    next_process = min(available, key=lambda p: p.
                       burst_time)
    next_process.waiting_time = time - next_process.
        arrival_time
    time += next_process.burst_time
    next_process.completion_time = time
    remaining.remove(next_process)
    completed.append(next_process)

return completed

def round_robin(processes, quantum):
    time = 0
    remaining = processes.copy()
    queue = []

    while remaining or queue:
        # Tambahkan proses yang baru tiba ke antrian
        new_arrivals = [p for p in remaining if p.
                        arrival_time <= time and p not in queue]
        queue.extend(new_arrivals)
        for p in new_arrivals:
            remaining.remove(p)

        if not queue:
            time += 1
            continue

        current_process = queue.pop(0)
        if current_process.remaining_time <= quantum:
            time += current_process.remaining_time
            current_process.completion_time = time
            current_process.waiting_time += time -
                current_process.arrival_time -
                current_process.burst_time
        else:
            time += quantum

```



```

        current_process.remaining_time -= quantum
        current_process.waiting_time += time -
            current_process.arrival_time - (
                current_process.burst_time -
                current_process.remaining_time)

        # Tambahkan proses yang baru tiba selama
        # eksekusi quantum
        new_arrivals = [p for p in remaining if p.
            arrival_time <= time and p not in queue]
        queue.extend(new_arrivals)
        for p in new_arrivals:
            remaining.remove(p)

        queue.append(current_process)

    def calculate_averages(processes):
        avg_waiting_time = sum(p.waiting_time for p in
            processes) / len(processes)
        avg_turnaround_time = sum(p.completion_time - p.
            arrival_time for p in processes) / len(processes)
        return avg_waiting_time, avg_turnaround_time

# Data proses
processes = [
    Process("P1", 0, 10),
    Process("P2", 1, 4),
    Process("P3", 3, 2),
    Process("P4", 5, 1)
]

# FCFS
fcfs_processes = [Process(p.pid, p.arrival_time, p.burst_time
    ) for p in processes]
fcfs(fcfs_processes)
fcfs_avg_waiting, fcfs_avg_turnaround = calculate_averages(
    fcfs_processes)

# SJN
sjn_processes = [Process(p.pid, p.arrival_time, p.burst_time)
    for p in processes]
sjn_completed = sjn(sjn_processes)
sjn_avg_waiting, sjn_avg_turnaround = calculate_averages(
    sjn_completed)

```

```
# Round Robin
rr_processes = [Process(p.pid, p.arrival_time, p.burst_time)
    for p in processes]
round_robin(rr_processes, 2)
rr_avg_waiting, rr_avg_turnaround = calculate_averages(
    rr_processes)
```

Output:

```
D:\Universitas Hasanuddin\Semester III\Sistem Operasi\Soal cmd cli>C:/Users/ASUS/AppData/Local/Microsoft/WindowsApps/python3.11.exe "
d:/Universitas Hasanuddin/Semester III/Sistem Operasi/Soal cmd cli/main3.py"
Algoritma | Waktu Tunggu Rata-rata | Waktu Penyelesaian Rata-rata
FCFS      | 7.75                    | 12.00
SJN       | 6.25                    | 10.50
RR (q=2)  | 9.25                    | 9.25
```

Figure 6: *Output* Kode

Kesimpulan:

Dari hasil *output* tersebut, kita bisa menyimpulkan bahwa:

- Algoritma SJN memberikan waktu tunggu dan waktu penyelesaian yang paling rendah di antara ketiga algoritma tersebut.
- FCFS memiliki waktu tunggu dan waktu penyelesaian yang tertinggi.
- Algoritma *Round Robin* dengan kuantum waktu dua unit menunjukkan waktu penyelesaian yang efisien, tetapi waktu tunggu rata-rata yang lebih tinggi dibandingkan SJN.

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.

4.5 Assignment 5: File System Access

4.5.1 Group 6

Soal:

Anda diminta untuk melakukan tugas-tugas berikut:

1. Membuat direktori baru bernama `ProyekA`.
2. Di dalam direktori `ProyekA`, buat tiga berkas teks baru bernama `file1.txt`, `file2.txt`, dan `file3.txt`.
3. Tuliskan "Hello World" ke dalam berkas `file1.txt`.
4. Tampilkan isi dari `file1.txt` ke layar.
5. Hapus berkas `file2.txt`.
6. Ganti nama berkas `file3.txt` menjadi `file_final.txt`.

Tulis kode python untuk menyelesaikan tugas di atas.

Jawaban:

```
import os

def main():
    # 1. Membuat direktori baru bernama ProyekA
    os.mkdir("ProyekA")
    print("Direktori_ProyekA_berhasil_dibuat.")

    # Pindah ke direktori ProyekA
    os.chdir("ProyekA")
    print("Berpindah_ke_direktori_ProyekA.")

    # 2. Membuat tiga berkas teks baru
```

```

for file in ["file1.txt", "file2.txt", "file3.txt"]:
    with open(file, "w") as f:
        pass
print("Berkas_file1.txt,_file2.txt,_dan_file3.txt_
      berhasil_dibuat.")

# 3. Menuliskan "Hello World" ke dalam file1.txt
with open("file1.txt", "w") as f:
    f.write("Hello_World")
print("Berhasil_menulis_'Hello_World'_ke_file1.txt.")

# 4. Menampilkan isi dari file1.txt ke layar
print("\nIsi_dari_file1.txt:")
with open("file1.txt", "r") as f:
    print(f.read())

# 5. Menghapus berkas file2.txt
os.remove("file2.txt")
print("\nBerkas_file2.txt_berhasil_dihapus.")

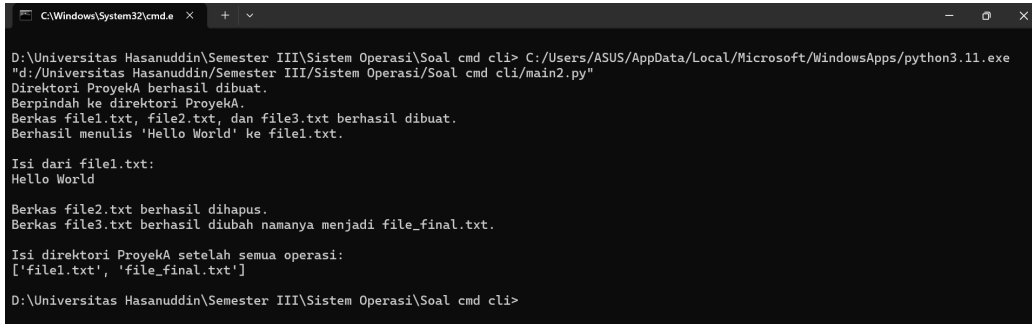
# 6. Mengganti nama berkas file3.txt menjadi file_final.
txt
os.rename("file3.txt", "file_final.txt")
print("Berkas_file3.txt_berhasil_diubah_namanya_menjadi_
      file_final.txt.")

# Menampilkan isi direktori akhir
print("\nIsi_direktori_ProjekA_setelah_semua_operasi:")
print(os.listdir())

if __name__ == "__main__":
    main()

```

Output:



```
C:\Windows\System32\cmd.exe X + v - [icon] X
D:\Universitas Hasanuddin\Semester III\Sistem Operasi\Soal cmd cli> C:/Users/ASUS/AppData/Local/Microsoft/WindowsApps/python3.11.exe
'd:/Universitas Hasanuddin/Semester III/Sistem Operasi/Soal cmd cli/main2.py'
Direktori ProyekA berhasil dibuat.
Berpindah ke direktori ProyekA.
Berkas file1.txt, file2.txt, dan file3.txt berhasil dibuat.
Berhasil menulis 'Hello World' ke file1.txt.

Isi dari file1.txt:
Hello World

Berkas file2.txt berhasil dihapus.
Berkas file3.txt berhasil diubah namanya menjadi file_final.txt.

Isi direktori ProyekA setelah semua operasi:
['file1.txt', 'file_final.txt']

D:\Universitas Hasanuddin\Semester III\Sistem Operasi\Soal cmd cli>
```

Figure 7: *Output*

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.