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.

3.1.1 Interaksi Antara *Brainware*, *Hardware*, dan *Software*

Brainware

Brainware adalah orang yang menggunakan, menjalankan, memanfaatkan, dan mengoperasikan perangkat komputer. Istilah brainware lebih sering ditujukan kepada pengguna komputer yang mampu mengoperasikan komputer dengan canggih, baik dari segi perangkat lunak (*software*) maupun perangkat keras (*hardware*) (Putri and Gischa, 2021).

Jenis-Jenis *Brainware*

Dalam bidang teknologi dan komputer, terdapat beragam jenis pekerjaan, masing-masing dengan peran dan tanggung jawab yang berbeda. Di bawah ini, terdapat penjelasan secara singkat:

- ***Project Manager***
  Jenis *brainware* ini bertanggung jawab untuk merencanakan, mengoordinasikan, dan mengawasi proyek teknologi dari awal hingga akhir.

- **Teknisi Komputer**
  Individu yang ahli dalam memperbaiki, memelihara, dan memasang perangkat keras dan jaringan.

- **Operator**
  Operator adalah pengguna yang mengoperasikan sistem komputer atau perangkat lunak untuk melakukan tugas-tugas tertentu.

- ***Database Administrator***
  Database Administrator biasanya melakukan pencadangan dan pemulihan data, mengelola izin akses, dan memastikan basis data beroperasi secara optimal.

Demikianlah penjelasan tentang jenis-jenis *brainware* (Fadillah, 2023).

Hubungan antara *Brainware*, *Hardware*, dan *Software*

Agar dapat berfungsi secara normal, komputer harus memiliki perangkat keras dan perangkat lunak. Perangkat lunak memberikan nilai fungsional pada perangkat keras.

Namun, adanya *brainware* juga sangat penting. *Brainware* memberikan nilai tambahan pada komputer sehingga memiliki nilai operasional. Tanpa *brainware*, komputer akan menjadi robot yang tidak memiliki nilai operasional (Bestari, 2022).

Jadi, interaksi komputer tidak hanya pada perangkatnya saja, tetapi juga harus diperhatikan dari nilai operasionalnya.

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.

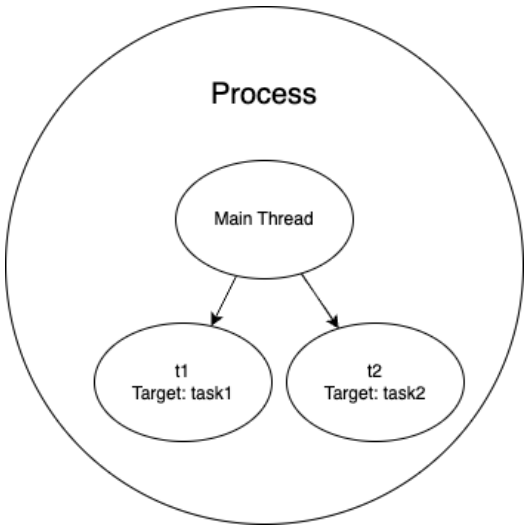

Figure 1: Ini adalah gambar contoh dari multithreading.

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

Soal

Diberikan 4 proses dengan *burst time* dan *arrival time* sebagai berikut:

Process	*Burst Time*	*Arrival Time*
P1	9	0
P2	3	1
P3	7	2
P4	2	3

Hitunglah *waiting time* (WT) dan *average waiting time* (AWT) untuk algoritma berikut:

- *First-Come, First-Served* (FCFS)
- *Round Robin* (RR) dengan *Time Quantum* = 2
- *Shortest Job First* (SJF)

Urutkan algoritma berdasarkan rata-rata waktu tunggu yang paling cepat.

Jawaban

FCFS:

```
process = ['P1', 'P2', 'P3', 'P4']
burst_time = [9, 3, 7, 2]
arrival_time = [0, 1, 2, 3]

def fcfs(process, burst_time, arrival_time):
    n = len(process)
    wt = [0] * n
    service_time = [0] * n

    for i in range(1, n):
        service_time[i] = service_time[i - 1] + burst_time[i
                                                    - 1]
        wt[i] = service_time[i] - arrival_time[i]
        if wt[i] < 0:
            wt[i] = 0

    awt = sum(wt) / n
    return wt, awt

wt_fcfs, awt_fcfs = fcfs(process, burst_time, arrival_time)
print("FCFS Waiting Time: ", wt_fcfs)
print("FCFS Average Waiting Time: ", awt_fcfs)
```

Output:

- FCFS WT: [0, 8, 10, 16] (WT dari P1–P4 berturut-turut)
- FCFS AWT: 8.5

RR, *Time Quantum* = 2:

```
def round_robin(process, burst_time, arrival_time,
                                time_quantum):
    n = len(process)
    rem_bt = burst_time[:]
    wt = [0] * n
    t = 0  # Current time

    while True:
        done = True
        for i in range(n):
            if rem_bt[i] > 0:
                done = False
                if rem_bt[i] > time_quantum:
                    t += time_quantum
                    rem_bt[i] -= time_quantum
                else:
                    t += rem_bt[i]
                    wt[i] = t - burst_time[i] - arrival_time[
                                                        i]
                    rem_bt[i] = 0
        if done:
            break

    awt = sum(wt) / n
    return wt, awt

time_quantum = 2
wt_rr, awt_rr = round_robin(process, burst_time, arrival_time
                                , time_quantum)
print("RR Waiting Time:", wt_rr)
print("RR Average Waiting Time:", awt_rr)
```

Output:

- RR (Q = 2) WT: [12, 7, 11, 3] (WT dari P1–P4 berturut-turut)
- RR (Q = 2) AWT: 8.25

SJF:

```
def sjf(process, burst_time, arrival_time):
    n = len(process)
    wt = [0] * n
    bt = burst_time[:]
    at = arrival_time[:]

    completed = 0
    t = 0
    min_bt = 0
    shortest = 0
    check = False

    while completed != n:
        for i in range(n):
            if at[i] <= t and bt[i] > 0 and (not check or bt[
                                                i] < bt[shortest])
                                                :
                shortest = i
                min_bt = bt[i]
                check = True

        if not check:
            t += 1
            continue

        bt[shortest] -= 1
        min_bt -= 1
        if bt[shortest] == 0:
            completed += 1
            check = False
            finish_time = t + 1
            wt[shortest] = finish_time - burst_time[shortest]
                                               - at[shortest]
            if wt[shortest] < 0:
                wt[shortest] = 0
        t += 1

    awt = sum(wt) / n
    return wt, awt

wt_sjf, awt_sjf = sjf(process, burst_time, arrival_time)
print("SJF Waiting Time:", wt_sjf)
print("SJF Average Waiting Time:", awt_sjf)
```

Output:

- SJF WT: [12, 0, 4, 1] (WT dari P1–P4 berturut-turut)
- SJF AWT: 4.25

Dengan demikian, urutan algoritma berdasarkan rata-rata waktu tunggu (AWT) yang paling cepat adalah:

- SJF
- RR (Q = 2)
- FCFS

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.

4.3.1 Group 1

Soal

Pada sistem komputasi modern, penggunaan *multithreading* sering kali diterapkan untuk mempercepat eksekusi program dengan cara menjalankan beberapa bagian program secara paralel. Efektivitas dari pendekatan ini dapat diukur menggunakan Hukum Amdahl.

Diketahui persamaan Hukum Amdahl sebagai berikut:

$$S_{max} = \frac{1}{(1 - P) + \frac{P}{N}}$$

di mana:

- S_{max} adalah percepatan maksimum yang dapat dicapai,
- P adalah persentase bagian dari program yang dapat diparalelkan,
- N adalah jumlah *threads* atau prosesor yang digunakan.

Berikut ini adalah tiga skenario yang harus diselesaikan:

1. Sebuah program memiliki 70% bagian yang dapat diparalelkan ($P = 0.7$). Tentukan percepatan maksimum (S_{max}) jika program dijalankan dengan:

- 2 *threads*,
- 4 *threads*,
- 8 *threads*.

2. Jika suatu program dapat diparalelkan sebesar 85% ($P = 0.85$), berapa jumlah minimum *threads* yang diperlukan untuk mencapai percepatan 5 kali lipat dari waktu eksekusi serial ($S_{max} = 5$)?

3. Jelaskan secara singkat, apakah selalu lebih baik menggunakan lebih banyak *threads*? Berdasarkan Hukum Amdahl, berikan alasan mengapa penggunaan *multithreading* kadang tidak memberikan percepatan yang signifikan.

Jawaban

1. Menghitung percepatan maksimum (S_{max}):

```
P = 0.7

def amdahls_law(P, N):
    return 1 / ((1 - P) + (P / N))

threads = [2, 4, 8]
for N in threads:
    S_max = amdahls_law(P, N)
    print(f"Percepatan maksimum untuk {N} threads: {S_max:.2f
                                    }")
```

Output:

- Untuk 2 *threads*, $S_{max} \approx 1.75$

- Untuk 4 *threads*, $S_{max} \approx 2.61$
- Untuk 8 *threads*, $S_{max} \approx 3.43$

2. Mencari jumlah minimum *threads* untuk mencapai percepatan 5 kali lipat:

```
from math import ceil

P = 0.85
S_target = 5

def find_min_threads(P, S_target):
    N = 1
    while True:
        S_max = amdahls_law(P, N)
        if S_max >= S_target:
            return N
        N += 1

min_threads = find_min_threads(P, S_target)
print(f"Jumlah minimum threads untuk mencapai percepatan 5
                                kali lipat: {min_threads}")
```

Output: Jumlah minimum *threads* yang diperlukan adalah 10.

3. Mengapa penggunaan lebih banyak *threads* tidak selalu efektif?

Multithreading memiliki batas percepatan yang dapat dicapai. Berdasarkan Hukum Amdahl, jika persentase bagian program yang tidak dapat diparalelkan $(1 - P)$ cukup besar, maka menambahkan lebih banyak *threads* tidak akan memberikan percepatan yang signifikan. Bahkan dengan jumlah *threads* yang sangat besar, bagian program yang tidak bisa diparalelkan akan tetap menjadi penghalang utama untuk mencapai percepatan yang tinggi.

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 1

Soal

Mahasiswa diminta untuk membuat *Simple Command-Line Interface* (CLI) untuk manajemen antarmuka pengguna. CLI ini harus mendukung perintah-perintah dasar seperti:

- Manipulasi berkas (membuat, menampilkan daftar, dan menghapus berkas),
- Manajemen proses,
- Pelaporan status sistem.

Berikut ini adalah spesifikasi tugasnya:
1. **Manipulasi berkas:**

- `buat_berkas <nama_berkas>` - Membuat berkas baru dengan nama yang diberikan.
- `daftar_berkas` - Menampilkan daftar semua berkas di direktori saat ini.
- `hapus_berkas <nama_berkas>` - Menghapus berkas dengan nama yang diberikan.

2. **Manajemen proses:**

- `daftar_proses` - Menampilkan daftar semua proses yang sedang berjalan.
- `hentikan_proses <PID>` - Menghentikan proses berdasarkan *Process ID* (PID).

3. **Pelaporan status sistem:**

- `status_sistem` - Menampilkan informasi tentang penggunaan CPU, memori, dan ruang disk.

Mahasiswa harus mengimplementasikan fungsi-fungsi ini dalam bahasa pemrograman Python dan memastikan semua perintah dapat dijalankan dari *command-line*.

Tugas:

- Implementasikan semua fungsi di atas menggunakan Python.
- Jelaskan bagaimana setiap perintah dieksekusi dan berikan contoh keluaran dari setiap perintah.

Jawaban

Contoh implementasi Python untuk perintah-perintah dasar CLI:

```
import os # Operating System
import psutil # Process and System Utilities

# Fungsi untuk membuat berkas baru
def buat_berkas(nama_berkas):
    with open(nama_berkas, 'w') as f:
        f.write('Berkas berhasil dibuat.\n')
    print(f'Berkas "{nama_berkas}" telah dibuat.')

# Fungsi untuk menampilkan daftar berkas di direktori saat
                                ini
def daftar_berkas():
    files = os.listdir('.')
    for file in files:
        print(file)

# Fungsi untuk menghapus berkas
def hapus_berkas(nama_berkas):
    if os.path.exists(nama_berkas):
        os.remove(nama_berkas)
        print(f'Berkas "{nama_berkas}" telah dihapus.')
    else:
        print(f'Berkas "{nama_berkas}" tidak ditemukan.')

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

# Fungsi untuk menghentikan proses berdasarkan PID
def hentikan_proses(pid):
    try:
        p = psutil.Process(pid)
        p.terminate()
        print(f'Proses dengan PID {pid} telah dihentikan.')
```

```
    except psutil.NoSuchProcess:
        print(f'Tidak ada proses dengan PID {pid}.')

# Fungsi untuk menampilkan status sistem
def status_sistem():
    print(f'Penggunaan CPU: {psutil.cpu_percent()}%')
    print(f'Penggunaan Memori: {psutil.virtual_memory().
                                percent}%')
    print(f'Penggunaan Disk: {psutil.disk_usage("/").percent}
                                %')
```

Contoh Penggunaan:

- `buat_berkas("contoh.txt")` akan membuat berkas bernama `contoh.txt`.
- `daftar_berkas()` akan menampilkan semua berkas di direktori saat ini.
- `hapus_berkas("contoh.txt")` akan menghapus berkas bernama `contoh.txt`.
- `daftar_proses()` akan menampilkan semua proses yang sedang berjalan.
- `hentikan_proses(12345)` akan menghentikan proses dengan PID 12345.
- `status_sistem()` akan menampilkan penggunaan CPU, memori, dan ruang disk.

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

4.5.1 Group 1

Soal

Dalam tugas ini, mahasiswa diminta untuk mengimplementasikan beberapa fungsi akses sistem berkas, yang meliputi:

- Pembuatan dan penghapusan berkas,
- Membaca dari dan menulis ke dalam berkas,
- Menavigasi direktori dan mengelola izin berkas.

Spesifikasi tugas:
1. **Pembuatan dan penghapusan berkas:**

- Implementasikan fungsi untuk membuat berkas baru di direktori yang ditentukan oleh pengguna.
- Implementasikan fungsi untuk menghapus berkas yang ada.

2. **Membaca dan menulis berkas:**

- Buat fungsi untuk membaca isi berkas.
- Buat fungsi untuk menulis data baru ke dalam berkas yang ada.

3. **Navigasi direktori dan izin berkas:**

- Buat fungsi untuk berpindah antara direktori di sistem file.
- Buat fungsi untuk menampilkan dan mengubah izin berkas (*file permissions*).

Tugas:

- Implementasikan fungsi-fungsi di atas menggunakan Python.
- Sertakan contoh penggunaan dari setiap fungsi yang telah dibuat.

Jawaban

Contoh implementasi Python untuk fungsi-fungsi akses sistem berkas:

```
import os

# Fungsi untuk membuat berkas baru
def buat_berkas(nama_berkas):
    with open(nama_berkas, 'w') as f:
        f.write('Ini adalah berkas baru.\n')
    print(f'Berkas "{nama_berkas}" telah dibuat.')

# Fungsi untuk menghapus berkas
def hapus_berkas(nama_berkas):
    if os.path.exists(nama_berkas):
        os.remove(nama_berkas)
        print(f'Berkas "{nama_berkas}" telah dihapus.')
    else:
        print(f'Berkas "{nama_berkas}" tidak ditemukan.')

# Fungsi untuk membaca isi berkas
def baca_berkas(nama_berkas):
    if os.path.exists(nama_berkas):
        with open(nama_berkas, 'r') as f:
            isi = f.read()
            print(isi)
    else:
        print(f'Berkas "{nama_berkas}" tidak ditemukan.')

# Fungsi untuk menulis ke berkas
def tulis_berkas(nama_berkas, data):
    with open(nama_berkas, 'a') as f:
        f.write(data + '\n')
    print(f'Data telah ditulis ke berkas "{nama_berkas}".')

# Fungsi untuk menavigasi direktori
def pindah_direktori(direktori):
    try:
        os.chdir(direktori)
        print(f'Berpindah ke direktori: {os.getcwd()}')
    except FileNotFoundError:
        print(f'Direktori "{direktori}" tidak ditemukan.')

# Fungsi untuk mengubah izin berkas
def ubah_izin(nama_berkas, izin):
    if os.path.exists(nama_berkas):
```

```
        os.chmod(nama_berkas, izin)
        print(f'Izin berkas "{nama_berkas}" telah diubah.')
    else:
        print(f'Berkas "{nama_berkas}" tidak ditemukan.')
```

Contoh Penggunaan:

- `buat_berkas("file_contoh.txt")` akan membuat berkas bernama `file_contoh.txt`.
- `hapus_berkas("file_contoh.txt")` akan menghapus berkas bernama `file_contoh.txt`.
- `baca_berkas("file_contoh.txt")` akan menampilkan isi dari `file_contoh.txt`.
- `tulis_berkas("file_contoh.txt", "Data baru")` akan menulis `Data baru` ke dalam `file_contoh.txt`.
- `pindah_direktori("/home/user/direktori")` akan berpindah ke direktori `/home/user/direktori`.
- `ubah_izin("file_contoh.txt", 0o755)` akan mengubah izin berkas `file_contoh.txt` menjadi 755.

Arti 0o755:

- 0o menunjukkan bahwa angka berikutnya adalah dalam sistem oktal.
- 7 (untuk pemilik): 4 (baca) + 2 (tulis) + 1 (eksekusi) = 7. Jadi, pemilik dapat membaca, menulis, dan mengeksekusi berkas.
- 5 (untuk grup): 4 (baca) + 1 (eksekusi) = 5. Jadi, anggota grup dapat membaca dan mengeksekusi berkas, tetapi tidak dapat menulis.
- 5 (untuk lainnya): 4 (baca) + 1 (eksekusi) = 5. Jadi, pengguna lain dapat membaca dan mengeksekusi berkas, tetapi tidak dapat menulis.

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

Bestari, N. (2022). Bagaimana interaksi antara perangkat keras dan perangkat lunak? [Diakses pada tanggal 3 Oktober 2024]. https://bobo.grid.id/read/083475438/bagaimana-interaksi-antara-perangkat-keras-dan-perangkat-lunak?page=all

Fadillah, R. (2023). Brainware: Definition, functions, and types [Diakses pada tanggal 3 Oktober 2024]. https://cmlabs.co/en-id/seo-guidelines/brainware

Putri, V. K. M., & Gischa, S. (2021). Brainware: Pengertian, kategori hingga fungsinya dalam sistem komputer [Diakses pada tanggal 3 Oktober 2024]. https://www.kompas.com/skola/read/2021/11/02/160000569/definisi-brainware-fungsi-dan-contohnya