Operating System Course Report - First Half of the Semester

B class

October 8, 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*

3.6.1 Sumber Daya Proses

A. Alokasi Memori
Setiap proses memerlukan memori untuk menyimpan data dan instruksi yang akan dijalankannya. Sistem operasi bertanggung jawab untuk mengalokasikan memori ini kepada proses dan memastikan bahwa proses lain tidak mengakses memori yang dialokasikan secara ilegal.

- Jenis memori yang dialokasikan:
  1. *Stack:* Digunakan untuk menyimpan variabel lokal dan melacak fungsi yang dipanggil selama eksekusi program.
  2. *Heap:* Menyediakan memori dinamis untuk aplikasi, digunakan saat proses memerlukan memori yang dialokasikan secara dinamis selama *runtime*.
  3. *Segmen Kode (Text Segment):* Menyimpan instruksi yang akan dieksekusi oleh CPU. Ini adalah bagian dari memori yang bersifat read-only, sehingga mencegah modifikasi terhadap instruksi program.

B. Penjadwalan Waktu CPU
Sistem operasi mengatur bagaimana dan kapan setiap proses dapat menggunakan CPU. Karena jumlah CPU terbatas dan banyak proses yang berjalan secara bersamaan, sistem operasi menggunakan mekanisme penjadwalan untuk mengalokasikan waktu CPU secara adil dan efisien.

- Jenis Penjadwalan Waktu CPU:
  1. *Preemptive Scheduling:* Sistem operasi dapat menghentikan proses yang sedang berjalan dan memberikan CPU kepada proses lain yang lebih prioritas.
  2. *Non-Preemptive Scheduling:* Proses diberikan CPU sampai selesai atau secara sukarela melepaskan CPU, baru setelah itu sistem operasi memberikan CPU ke proses lain.

C. Pengaturan *File Descriptors*
File descriptor adalah integer unik yang diberikan oleh sistem operasi untuk merujuk ke *file*, *socket*, atau perangkat yang dibuka oleh proses. Proses menggunakan *file descriptor* untuk melakukan operasi I/O seperti membaca, menulis, atau mengelola *file*.

- Fungsi *File Descriptor*:
  1. Akses File: *File descriptors* digunakan oleh proses untuk membaca, menulis, atau melakukan operasi lainnya pada *file*.
  2. Komunikasi Antar-Proses (IPC): *File descriptors* digunakan dalam komunikasi antar-proses seperti socket atau pipe untuk mengirimkan data antar proses.
  3. Manajemen Sumber Daya: *File descriptors* membantu sistem operasi melacak dan mengelola sumber daya seperti *file* terbuka, socket, atau perangkat yang digunakan oleh suatu proses.

3.6.2 Terminasi Paksa oleh Proses Lain

A. *Sinyal Kill*
Sinyal Kill adalah perintah yang digunakan dalam sistem operasi untuk menghentikan proses yang sedang berjalan. Meskipun namanya

terdengar drastis, sinyal ini tidak selalu berarti menghentikan proses secara paksa, ada berbagai jenis sinyal yang dapat dikirim.

Contoh Sinyal:

1. *SIGTERM:* Sinyal ini memberi tahu proses untuk berhenti dengan cara yang teratur. Proses dapat menangani sinyal ini dengan melakukan pembersihan sebelum keluar.
2. *SIGKILL:* Sinyal ini menghentikan proses secara paksa dan tidak dapat diabaikan atau ditangani oleh proses. Ini digunakan ketika proses tidak merespons permintaan untuk berhenti.

Penggunaan *Sinyal Kill*:
Sinyal ini biasanya digunakan ketika proses tidak merespons, macet, atau berperilaku tidak sesuai. Misalnya, jika aplikasi grafis tidak merespons, pengguna dapat menggunakan perintah *kill* untuk menghentikannya.

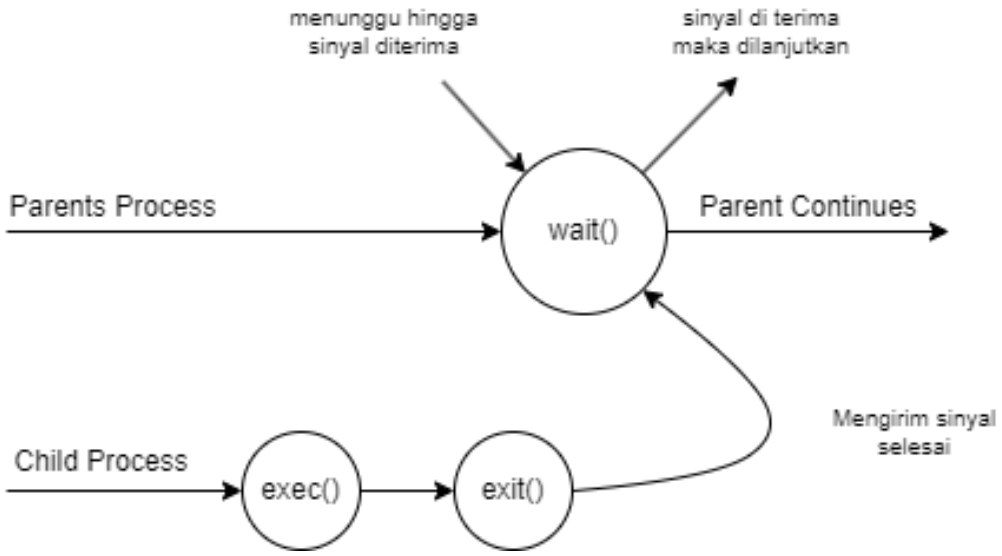
B. *Timeout*
Timeout adalah mekanisme di mana sistem operasi menetapkan batas waktu tertentu untuk proses. Jika proses tidak menyelesaikan tugasnya dalam jangka waktu yang ditentukan, sistem operasi akan menghentikannya.

Contoh Penggunaan: Dalam sistem waktu nyata atau dalam lingkungan penjadwalan pekerjaan, jika sebuah proses tidak merespons dalam waktu tertentu, sistem operasi dapat mengakhiri proses tersebut untuk mencegah masalah dalam pengelolaan sumber daya.

Pentingnya *Timeout:*
Timeout mencegah satu proses dari penggunaan sumber daya sistem yang berlebihan, yang bisa mengakibatkan penurunan kinerja untuk proses lain.

3.6.3 *Orphan Process* atau *Zombie Process*

A. *Orphan Process*
Orphan Process adalah proses anak yang tetap berjalan setelah proses



induknya berakhir. Ketika proses induk mati, proses anak menjadi yatim, tetapi sistem operasi tidak langsung menghapusnya.

Pengelolaan *Orphan Process*: Dalam sistem berbasis Unix, *Orphan Process* diadopsi oleh proses init (PID 1). Proses init bertugas untuk mengelola *orphan process* dan memastikan bahwa semua sumber daya yang dialokasikan untuk proses anak dibebaskan saat proses tersebut berakhir.

B. *Zombie Process*

Zombie Process adalah proses yang telah selesai eksekusi tetapi masih ada dalam tabel proses karena proses induknya belum mengambil status keluar (exit status)nya. Proses ini tidak aktif, tetapi masih ada dalam sistem.

Proses zombie muncul karena sistem operasi menyimpan informasi tentang proses yang telah selesai sehingga proses induk dapat mendapatkan status keluar untuk melakukan pembersihan yang diperlukan.

Menghapus Proses Zombie: Setelah proses induk menggunakan `wait()` atau `waitpid()` untuk mengambil status keluar dari proses anaknya, sistem operasi dapat menghapus entri proses zombie dari tabel proses.

3.6.4 Tahapan dalam Terminasi Proses

A. Sinyal untuk Terminasi
Proses dapat berakhir dengan cara alami atau menerima sinyal untuk terminasi. Jika menerima sinyal, proses akan melaksanakan handler (jika ada) atau menghentikan eksekusinya.

B. Pembersihan Sumber Daya
Setelah proses menerima sinyal terminasi atau selesai eksekusi, sistem operasi akan mulai mereklamasi semua sumber daya yang digunakan oleh proses tersebut. Ini termasuk:

1. *Memori:* Menghapus alokasi memori yang digunakan oleh stack, heap, dan segmen kode.
2. *File Deskriptor:* Menutup semua file yang dibuka oleh proses untuk mencegah kebocoran sumber daya.
3. Perangkat I/O: Mengembalikan kontrol atas perangkat I/O yang digunakan oleh proses.

C. Penghapusan *Process Control Block* (PCB)
Setiap proses memiliki PCB yang menyimpan informasi penting tentang proses, termasuk status, prioritas, dan informasi penggunaan sumber daya. Setelah proses dibersihkan, PCB dihapus dari tabel proses untuk membebaskan ruang dan menghilangkan informasi tentang proses tersebut.

D. Pemberitahuan kepada Proses Induk
Dalam sistem berbasis Unix, setelah proses anak selesai, proses induknya diberi tahu melalui sinyal `SIGCHLD`. Proses induk dapat menangani sinyal ini dan memanggil `wait()` atau `waitpid()` untuk mengambil status keluar dari proses anak. Ini memungkinkan proses induk untuk melakukan pembersihan yang diperlukan dan memastikan bahwa tidak ada proses zombie yang tertinggal.

Daftar Pustaka

– Silberschatz, A., Galvin, P. B., Gagne, G. (2018). Operating system concepts (10th ed.). Wiley.

– Almaunah. (n.d.). Mengidentifikasi Proses Sistem Operasi. Fakultas Komputer, Universitas Mitra Indonesia.

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 1: Ini adalah gambar contoh dari 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

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

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

4.3.2 group 2

4.3.3 group 3

4.3.4 group 4

4.3.5 group 5

4.3.6 group 6

Buatlah sebuah program Python untuk menghitung nilai Fibonacci ke-40 menggunakan multithreading. Setelah itu, implementasikan Amdahl's Law untuk menghitung speedup teoretis dengan berbagai jumlah thread. Jawaban:

```
import threading
import time

# Fungsi untuk menghitung Fibonacci secara rekursif
def hitung_fibonacci(angka):
    if angka <= 1:
        return angka
    else:
        return hitung_fibonacci(angka-1) + hitung_fibonacci(
                                        angka-2)

# Fungsi untuk mengukur waktu eksekusi
def jalankan_threads(jumlah_threads, angka_fibonacci):
    threads = []

    waktu_mulai = time.time()

    for i in range(jumlah_threads):
        thread = threading.Thread(target=hitung_fibonacci,
                                        args=(angka_fibonacci
                                        ,))
        threads.append(thread)
        thread.start()

    for thread in threads:
        thread.join()

    waktu_selesai = time.time()
    return waktu_selesai - waktu_mulai
```

```
# Implementasi Amdahl's Law
def hitung_speedup_amdahl(persentase_paralel, jumlah_threads)
                                    :
    return 1 / ((1 - persentase_paralel) + (
                                        persentase_paralel /
                                        jumlah_threads))

# Main program
if __name__ == "__main__":
    angka_fibonacci = 40  # Fibonacci ke-40
    persentase_paralel = 0.95  # Estimasi paralelisme
    daftar_threads = [1, 2, 4, 8]  # Daftar jumlah thread
                                        yang akan diuji

    for jumlah_threads in daftar_threads:
        waktu_eksekusi = jalankan_threads(jumlah_threads,
                                            angka_fibonacci)
        speedup_teoretis = hitung_speedup_amdahl(
                                            persentase_paralel,
                                            jumlah_threads)

        print(f"\nJumlah Threads: {jumlah_threads}")
        print(f"Waktu Eksekusi: {waktu_eksekusi:.4f} detik")
        print(f"Speedup Teoretis (Amdahl's Law): {
                                            speedup_teoretis:.4f}"
                                            )
```

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.