Operating System Course Report - First Half of the Semester

B class

October 1, 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:

3.6.1 Pembuatan dan Terminasi Proses

Bagian ini menjelaskan bagaimana proses diakhiri oleh sistem operasi, dengan fokus pada proses terminasi dan pembersihan yang terjadi setelahnya. Proses terminasi adalah tahapan akhir dari siklus hidup proses yang ditangani secara hati-hati oleh sistem operasi untuk mencegah masalah seperti kebocoran memori, penggunaan sumber daya yang tidak efisien, atau penguncian sumber daya. Setelah terminasi, langkah-langkah pembersihan (cleanup) menjadi penting dalam menjaga stabilitas dan kinerja sistem secara keseluruhan.

3.6.2 Proses Terminasi dan Kondisi Cleanup

Ketika sebuah proses berakhir, beberapa mekanisme sistem operasi akan melakukan langkah-langkah untuk melepaskan dan mendaur ulang sumber daya yang telah digunakan oleh proses. Proses cleanup ini bertujuan agar sumber daya tersebut dapat digunakan kembali oleh proses lain. Proses terminasi mencakup de-allocating sumber daya yang digunakan, menghapus informasi dari tabel proses, dan mengembalikan status eksekusi ke proses induk.

3.6.3 De-Alokasi Sumber Daya

Sumber daya yang digunakan oleh sebuah proses selama masa hidupnya meliputi memori, file, dan perangkat keras. Ketika proses selesai, sumber daya tersebut harus dilepaskan. Jika tidak, sistem akan kehabisan sumber daya yang menyebabkan kinerja sistem menurun. De-allocating atau pelepasan sumber daya ini sangat krusial karena setiap byte memori atau perangkat keras yang tidak dibebaskan dapat membuat sistem menjadi kurang efisien. Berikut penjelasan lebih detail tentang jenis sumber daya yang dilepaskan:

- **Memori (RAM)**: Memori yang dialokasikan ke suatu proses harus dikembalikan ke pool memori setelah proses berakhir. RAM yang tidak dibebaskan kembali ke sistem dapat menyebabkan kebocoran memori (*memory leak*), yaitu kondisi di mana memori yang tidak terpakai tetap terkunci oleh proses yang telah berakhir. Jika kebocoran memori ini dibiarkan, sistem akan mengalami penurunan kinerja karena kapasitas memori yang tersedia semakin berkurang seiring dengan berjalannya waktu, hingga akhirnya menyebabkan sistem melambat atau crash.

- **File yang Terbuka**: Ketika sebuah proses membuka file selama eksekusi, file tersebut harus ditutup setelah proses selesai. File descriptor yang tetap terbuka dapat mengunci file tersebut, sehingga tidak dapat diakses oleh proses lain yang memerlukannya. Selain itu, jika jumlah file descriptor yang tidak tertutup meningkat, sistem bisa mencapai batas maksimum file descriptor yang dapat dibuka. Ini menyebabkan aplikasi atau proses baru tidak dapat membuka file karena tidak ada lagi file descriptor yang tersedia, dan ini dapat mengakibatkan kegagalan dalam eksekusi proses lain.

- **Perangkat Keras yang Digunakan**: Beberapa proses membutuhkan akses eksklusif ke perangkat keras tertentu, misalnya port jaringan, printer, atau disk drive. Setelah proses berakhir, akses eksklusif ini harus dilepaskan, sehingga perangkat keras tersebut dapat digunakan oleh proses lain. Jika perangkat keras tidak dilepaskan dengan benar, perangkat tersebut bisa tetap terkunci oleh proses yang telah berakhir, menyebabkan masalah bagi proses lain yang mencoba menggunakan perangkat keras yang sama. Ini bisa menghambat operasi lain dalam sistem dan menyebabkan sistem macet atau tidak responsif.

3.6.4 Menghapus Entri dari Tabel Proses

Setiap proses di dalam sistem dilacak oleh tabel proses (*Process Table*), yang berisi informasi penting mengenai setiap proses yang berjalan. Setelah proses berakhir, entri dalam tabel proses harus dihapus untuk mencegah masalah yang bisa muncul jika PID yang sama digunakan oleh proses baru. Tabel proses berfungsi sebagai "peta" untuk sistem operasi, yang memungkinkan sistem memantau setiap proses yang berjalan dan sumber daya yang sedang digunakan. Penghapusan entri yang tepat sangat penting dalam menjaga efisiensi dan menghindari kebingungan dalam alokasi sumber daya.

- **Status Proses**: Setiap proses memiliki status, seperti sedang berjalan (*running*), ditunda (*suspended*), atau selesai (*terminated*). Ketika proses telah berakhir, statusnya harus diubah menjadi "terminated" dan entri ini perlu dihapus untuk menghindari konflik dengan proses baru yang mungkin menggunakan PID yang sama. Jika tabel proses tidak diperbarui, sistem mungkin akan menganggap bahwa proses yang sudah berakhir masih berjalan, sehingga dapat menyebabkan penggunaan PID yang tidak benar atau tumpang tindih sumber daya.

- **PID (Process ID)**: Setiap proses di sistem operasi memiliki identifikasi unik yang disebut PID. Setelah sebuah proses selesai, PID yang digunakan oleh proses tersebut dikembalikan dan dapat digunakan oleh proses lain. Namun, entri di tabel proses harus dihapus terlebih dahulu untuk memastikan bahwa tidak ada informasi yang salah atau usang terkait proses lama yang dapat mengganggu proses baru yang menggunakan PID tersebut. Proses penghapusan ini memastikan bahwa setiap PID yang digunakan tetap unik dan akurat.

- **Informasi Sumber Daya**: Selain status dan PID, tabel proses juga menyimpan informasi terkait sumber daya yang sedang digunakan oleh proses, seperti memori, file descriptor, dan perangkat keras. Setelah proses berakhir, informasi ini harus dihapus untuk memastikan bahwa sumber daya tersebut dapat digunakan oleh proses lain. Ini juga membantu menghindari masalah yang terkait dengan sumber daya yang tersisa terkunci atau tidak tersedia untuk proses baru.

3.6.5 Mengembalikan Status Eksekusi ke Parent Process

Ketika sebuah proses diciptakan, biasanya ada proses induk (*parent process*) yang bertanggung jawab atas proses tersebut. Setelah proses anak (*child process*) selesai, sistem operasi harus mengembalikan informasi mengenai status eksekusi proses anak ke proses induk. Informasi ini meliputi apakah proses anak berhasil atau gagal, dan alasan mengapa proses berakhir. Mekanisme pengembalian status ini penting agar proses induk dapat mengambil tindakan yang sesuai, seperti melanjutkan eksekusi, menangani kesalahan, atau membuat proses baru.

- `wait()`: Panggilan sistem ini digunakan oleh proses induk untuk menunggu hingga proses anak selesai. Saat proses anak berakhir, kode keluaran (*exit code*) dikembalikan ke proses induk. Kode ini menunjukkan hasil akhir dari proses anak, seperti apakah proses berhasil diselesaikan atau ada kesalahan yang terjadi. Berdasarkan kode keluaran ini, proses induk dapat mengambil tindakan yang sesuai, seperti melanjutkan operasi, melakukan rollback, atau mencoba menjalankan kembali proses anak dengan kondisi yang berbeda.

- `waitpid()`:

  Panggilan ini adalah variasi dari `wait()`, di mana proses induk dapat menunggu proses anak tertentu berdasarkan PID-nya. Ini sangat berguna dalam situasi di mana proses induk memiliki lebih dari satu proses anak yang berjalan secara bersamaan. Dengan `waitpid()`, proses induk dapat memilih proses anak mana yang ingin dia tunggu, memberikan kontrol yang lebih besar atas pengelolaan proses dan memastikan bahwa eksekusi berjalan lebih efisien dan terstruktur.

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

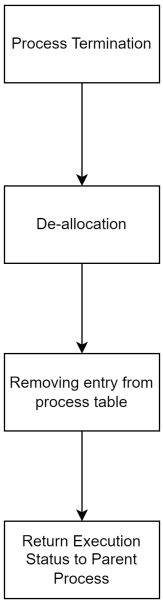

Figure 1: Diagram alur pembersihan setelah terminasi proses.

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:

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 $_{init}{}_{(self,pid,arrival_time,burst_time):self.pid=pidself.arrival_time=arrival_timeself.burst_time=burst_timeself.completion}$

4.2 Assignment 2: Deadlock Handling

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

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

4.3 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.

References

[1] Universitas Muhammadiyah Yogyakarta. (n.d.). *Creation and Termination of Processes*. Diambil dari: `https://oprek.um.ac.id/proses-spawning`

[2] University of Illinois Chicago. (n.d.). *Operating System Cleanup After Process Termination*. Diambil dari: `https://os-cleanup.uic.edu`