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:

- Konsep Threads
- Hubungan antara Threads dan Proses
- Manfaat Penggunaan Threads
- Multithreading
- Threads vs Process
- Model Multithreading
- Pengelolaan Threads
- Penerapan Threads pada Sistem Operasi

3.7.1 Konsep Threads

3.7.2 Hubungan antara Threads dan Proses

3.7.3 Manfaat Penggunaan Threads

3.7.4 Multithreading

3.7.5 *Threads vs Proses*

Dalam sistem operasi modern, konsep proses dan *thread* adalah dua mekanisme penting untuk melakukan eksekusi program secara paralel atau *multitasking*. Keduanya memberikan kemampuan bagi sistem untuk menjalankan berbagai tugas secara efisien, tetapi dengan cara yang berbeda dalam hal pengelolaan sumber daya, memori, dan *performa*. Makalah ini akan membahas perbedaan mendasar antara *thread* dan proses, cara kerja keduanya, serta kelebihan dan kekurangan masing-masing.

Proses adalah program yang sedang dieksekusi, yang terdiri dari instruksi yang dijalankan oleh CPU dan memiliki ruang memori sendiri, mencakup ruang alamat, data, dan variabel. Setiap proses yang berjalan di sistem operasi adalah entitas independen yang tidak berbagi memori atau sumber daya dengan proses lain kecuali menggunakan mekanisme komunikasi khusus seperti *Inter-Process Communication* (IPC).

Karakteristik proses:

1. Isolasi Memori: Setiap proses memiliki ruang memori sendiri, terisolasi dari proses lain.

2. Komunikasi: Menggunakan mekanisme IPC seperti *shared memory*, *pipes*, atau *message passing*.

3. *Overhead* Tinggi: Membuat proses baru membutuhkan waktu dan sumber daya lebih banyak karena setiap proses memiliki salinan terpisah dari memori, sumber daya, dan lingkungan eksekusi.

Thread

Thread adalah unit eksekusi yang lebih ringan di dalam proses. Sebuah proses dapat memiliki banyak *thread* yang berbagi ruang memori dan sumber daya. Semua *thread* dalam satu proses dapat mengakses data yang sama dan bekerja secara bersamaan, membuatnya lebih efisien untuk tugas-tugas yang memerlukan multitasking dalam ruang memori yang sama.

Karakteristik *thread*:

1. Berbagi Memori: Semua *thread* dalam satu proses berbagi ruang alamat yang sama, membuat komunikasi antar *thread* lebih cepat.

2. Komunikasi Lebih Efisien: Karena berbagi memori, *thread* tidak memerlukan mekanisme IPC yang kompleks untuk berkomunikasi.

3. *Overhead* Rendah: Membuat *thread* baru lebih cepat dan membutuhkan lebih sedikit sumber daya dibandingkan membuat proses baru.

Perbedaan utama antara Proses dan Thread

1. Isolasi Memori

   (a) Proses memiliki ruang memori sendiri yang terpisah dari proses lain. Ini berarti setiap proses memiliki wilayah alamat memori yang terisolasi, dan satu proses tidak bisa secara langsung mengakses data dari proses lain.

   (b) *Thread*, di sisi lain, berbagi ruang memori dengan *thread* lain dalam satu proses. Semua *thread* dalam satu proses memiliki akses ke data dan variabel yang sama, sehingga komunikasi antar-*thread* lebih mudah tetapi membutuhkan manajemen sinkronisasi.

2. Komunikasi

   (a) Proses menggunakan mekanisme *Inter-Process Communication* (IPC) untuk berkomunikasi satu sama lain, seperti *message passing*, *pipes*, atau *shared memory*. Mekanisme IPC sering kali lebih lambat dan kompleks karena memerlukan protokol tambahan untuk mentransfer data antara proses.

   (b) *Thread* dalam satu proses dapat berkomunikasi secara langsung melalui memori bersama tanpa perlu menggunakan IPC. Karena itu, komunikasi antar-*thread* jauh lebih cepat dibandingkan dengan antar-proses.

3. *Overhead*

(a) Membuat atau mengelola proses baru membutuhkan lebih banyak waktu dan sumber daya. Setiap kali sebuah proses baru dibuat, sistem harus menyiapkan ruang memori dan salinan sumber daya terpisah untuk proses tersebut, yang membutuhkan *overhead* yang besar.

(b) Membuat *thread* baru dalam sebuah proses jauh lebih efisien dan memerlukan lebih sedikit sumber daya karena *thread* berbagi sebagian besar lingkungan eksekusi dengan *thread* lain di proses yang sama.

4. Manajemen

(a) Proses dikelola secara independen oleh sistem operasi. Setiap proses memiliki siklus hidup sendiri yang tidak bergantung pada proses lain. Jika sebuah proses mengalami *crash*, tidak akan mempengaruhi proses lain.

(b) *Thread* dikelola di bawah proses yang sama dan berbagi sumber daya bersama. Jika satu *thread* mengalami *crash* atau kegagalan, bisa mempengaruhi *thread* lain dalam proses yang sama, sehingga meningkatkan risiko ketidakstabilan.

5. Kegagalan

(a) Jika sebuah proses mengalami kegagalan atau *crash*, proses tersebut tidak akan mempengaruhi proses lain karena setiap proses terisolasi satu sama lain. Hal ini memberikan keandalan yang lebih besar.

(b) Jika satu *thread* dalam sebuah proses mengalami kegagalan, hal itu dapat berdampak pada *thread* lain dalam proses yang sama karena mereka berbagi ruang memori dan sumber daya yang sama. Oleh karena itu, kegagalan satu *thread* bisa menyebabkan seluruh proses menjadi tidak stabil.

6. Penggunaan

(a) Proses lebih cocok untuk menjalankan aplikasi yang berbeda secara terpisah, terutama ketika keamanan dan isolasi sangat penting. Misalnya, menjalankan dua aplikasi seperti *browser* dan editor teks sebagai dua proses berbeda.

(b) *Thread* lebih cocok digunakan ketika sebuah aplikasi memerlukan multitasking internal. Contohnya, aplikasi seperti server web menggunakan *thread* untuk menangani banyak permintaan pengguna dalam satu proses yang sama.

Manfaat dan Kekurangan

1. Proses

   Manfaat:

   (a) Isolasi dan Keamanan: Proses yang berbeda terisolasi satu sama lain, sehingga kegagalan atau *crash* pada satu proses tidak akan memengaruhi proses lain. Ini memberikan stabilitas dan keamanan lebih baik, terutama untuk aplikasi yang berjalan secara independen.

   (b) Penggunaan untuk Aplikasi Terpisah: Proses cocok untuk menjalankan aplikasi yang berbeda, seperti menjalankan editor teks bersamaan dengan *browser*.

   Kekurangan:

   (a) *Overhead* Besar: Pembuatan proses baru memerlukan lebih banyak sumber daya dan waktu karena setiap proses memiliki salinan sendiri dari memori dan sumber daya.

   (b) Komunikasi yang Rumit: Komunikasi antar proses memerlukan mekanisme IPC, yang lebih lambat dan kompleks dibandingkan komunikasi antar-*thread*.

2. *Thread*

   Manfaat:

(a) Multitasking yang Cepat: *Thread* memungkinkan multitasking yang lebih efisien dalam satu proses, karena berbagi memori dan sumber daya dengan *thread* lain.

(b) Lebih Efisien dalam Sumber Daya: Membuat *thread* baru lebih cepat dan ringan dibandingkan membuat proses baru, sehingga cocok untuk aplikasi yang membutuhkan banyak tugas paralel seperti server web atau aplikasi *real-time*.

Kekurangan:

(a) Masalah Keamanan dan Kestabilan: Karena *thread* berbagi memori dan sumber daya, jika satu *thread* mengalami *crash* atau error, hal itu bisa berdampak pada *thread* lain dalam proses yang sama.

(b) Kompleksitas Pengelolaan Sinkronisasi: Dalam aplikasi *multi-threaded*, sinkronisasi antara *thread* harus dikelola dengan hati-hati untuk menghindari masalah seperti *race conditions* atau *deadlock*.

Implementasi

1. Proses

   Pada sistem operasi seperti Linux, menjalankan dua aplikasi berbeda seperti *Google Chrome* dan *LibreOffice* adalah contoh penggunaan proses. Masing-masing aplikasi berjalan dalam ruang memori yang terpisah dan tidak dapat berinteraksi secara langsung tanpa melalui mekanisme IPC.

2. *Thread*

   Aplikasi seperti *Google Chrome* juga menggunakan multithreading di dalam satu proses. Setiap tab atau plugin di *browser* mungkin dijalankan sebagai *thread* terpisah yang berbagi memori, tetapi berfungsi secara independen untuk meningkatkan performa dan responsivitas.

Kapan menggunakan Proses dan *Thread*?

1. Gunakan proses ketika isolasi dan keamanan antar tugas sangat penting, misalnya saat menjalankan aplikasi yang berbeda. Isolasi memori dan sumber daya membantu melindungi stabilitas sistem jika terjadi *crash*.

2. Gunakan *thread* ketika kamu memerlukan efisiensi dalam multitasking di dalam satu aplikasi. *Thread* cocok untuk aplikasi yang membutuhkan banyak tugas paralel yang saling bergantung, seperti server web atau aplikasi multimedia.

Tanenbaum, A. S., & Bos, H. (2015). *Modern operating systems* (4th ed.). Pearson.

3.7.6 Model Multithreading

3.7.7 Pengelolaan Threads

3.7.8 Penerapan Threads pada Sistem Operasi

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.

4.2 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

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.