Operating System Course Report - First Half of the Semester

B class

October 8, 2024

Contents

1 Introduction **4**

2 Course Overview **4**
2.1 Objectives 4
2.2 Course Structure 4

3 Topics Covered **5**
3.1 Basic Concepts and Components of Computer Systems 5
3.2 System Performance and Metrics 5
3.3 System Architecture of Computer Systems 5
3.4 Process Description and Control 5
3.5 Scheduling Algorithms 6
3.6 Process Creation and Termination 6
3.7 Introduction to Threads 6
3.8 File Systems 6
3.8.1 File System Structure 8
3.8.2 F 13
3.8.3 D 14
3.9 Input and Output Management 14
3.10 Deadlock Introduction and Prevention 14
3.11 User Interface Management 14
3.12 Virtualization in Operating Systems 14

4 Assignments and Practical Work **15**
4.1 Assignment 1: Process Scheduling 15
4.1.1 Group 1 15
4.1.2 15
4.1.3 15
4.1.4 15
4.1.5 15
4.1.6 15
4.1.7 15
4.1.8 Group 8 15
4.2 Assignment 2: Deadlock Handling 19
4.3 Assignment 3: Multithreading and Amdahl's Law 19
4.3.1 19

4.3.2 19
4.3.3 19
4.3.4 19
4.3.5 19
4.3.6 19
4.3.7 Group 8 19
4.4 Assignment 4: Simple Command-Line Interface (CLI) for User Interface Management 21
4.5 Assignment 5: File System Access 21

5 Conclusion **22**

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:

- Single-threaded vs. multi-threaded processes
- Benefits of multithreading

Seperti yang terlihat pada Gambar 1, inilah cara menambahkan gambar dengan keterangan.

3.8 File Systems

File adalah unit penyimpanan logika yang diabstraksi oleh sistem operasi dari perangkat penyimpanan. File berisi informasi yang disimpan pada penyimpanan sekunder (seperti *magnetic disk*, *magnetic tape*, dan *optical disk*). Informasi dalam file didefinisikan oleh pembuatnya. Setiap file memiliki struktur tertentu tergantung pada isinya. File dapat terdiri dari data (baik berupa data numerik, karakter, maupun biner) serta program seperti *source program*, *object program*, dan *executable program*.

Figure 1: Ini adalah gambar contoh dari multithreading.

a. Ruang Alamat Logis yang Berdekatan

File dalam sistem biasanya disimpan sebagai ruang alamat logis yang berdekatan. Hal ini berarti bahwa file menempati urutan alamat logis dalam memori. Namun, hal ini tidak selalu menyiratkan bahwa file juga menempati ruang fisik yang berdekatan pada disk. Sistem file berfungsi untuk menyederhanakan akses dan pengambilan file dengan membuatnya tampak berurutan, terlepas dari bagaimana penyimpanannya secara fisik.

b. Jenis File

File dapat dikategorikan menjadi dua jenis utama:

1. **File Data**: Berisi data terstruktur atau tidak terstruktur dan dapat dikategorikan lebih lanjut menjadi:
    - **Numerik**: Data yang direpresentasikan sebagai angka (misalnya, data statistik atau data keuangan).
    - **Character**: Data yang direpresentasikan sebagai teks yang dapat dibaca manusia.
    - **Biner**: File non-teks, seperti file media (audio, video), atau jenis data lainnya yang dikodekan.

2. **File Program**: File yang berisi kode yang dapat dieksekusi untuk menjalankan aplikasi atau layanan pada komputer.

c. Konten File yang Ditentukan oleh Pencipta

Isi file ditentukan oleh pembuatnya dan tujuan dari pembuatan file tersebut. File yang berbeda mungkin berisi instruksi (*kode sumber*) untuk program komputer, dokumentasi tertulis (file teks), atau kode yang sudah dikompilasi (file *executable*). Contoh jenis file:

- **File Teks**: File sederhana yang berisi karakter yang dapat dibaca oleh manusia.
- **File Sumber**: File yang berisi kode sumber yang ditulis dalam bahasa pemrograman.
- **File yang dapat dieksekusi**: File yang dapat dijalankan langsung oleh sistem operasi sebagai program. File ini berisi kode yang sudah dikompilasi atau ditafsirkan.

3.8.1 File System Structure

Struktur sistem file mengatur bagaimana file disimpan, diakses, dan diatur dalam perangkat penyimpanan. Struktur ini mencakup beberapa komponen, seperti direktori, tabel alokasi, dan *metadata* yang membantu mengelola file dengan efisien. Struktur file dapat dikategorikan ke dalam berbagai jenis berdasarkan kompleksitas organisasi data. Misalnya:

1. **Simple record Structure**: terdiri dari urutan baris atau catatan dengan panjang tetap atau variabel
2. **Complex structures**: lebih rumit mencakup dokumen yang diformat dan file beban yang dapat direlokasi, yang memerlukan mekanisme penanganan yang lebih canggih.

Struktur ini sangat penting dalam mengelola bagaimana data disimpan dan diakses, dan karakter kontrol dapat mensimulasikan struktur yang kompleks. Keputusan tentang organisasi struktur file dipengaruhi oleh sistem operasi dan program yang digunakan. Tipe file juga digunakan untuk menunjukkan struktur internal dari file. File tertentu harus konfirmasi ke struktur

yang dibutuhkan yang dimengerti oleh sistem operasi. Misalnya, sistem operasi membutuhkan file *executable* yang mempunyai struktur khusus sehingga dapat menentukan dimana letak memori dan lokasi dari instruksi pertama.

Beberapa sistem operasi menggunakan sekumpulan sistem pendukung struktur file dengan sejumlah operasi khusus untuk manipulasi file dengan struktur tersebut. Hal ini menjadi kelemahan pada sistem operasi yang mendukung struktur file lebih dari satu. Jika sistem operasi menentukan 10 struktur file berbeda, maka perlu menyertakan kode untuk mendukung struktur file tersebut. Setiap file perlu dapat didefinisikan sebagai satu dari tipe file yang didukung oleh sistem operasi. Beberapa sistem operasi seperti UNIX dan MS-DOS hanya mendukung sejumlah struktur file. UNIX menentukan setiap file merupakan deret 8 bit byte dan bit tersebut tidak diterjemahkan oleh sistem operasi. Skema ini mempunyai fleksibilitas maksimum, tetapi sedikit dukungan. Setiap program aplikasi harus menyertakan kode sendiri untuk menterjemahkan file input ke dalam struktur yang tepat. Setidaknya semua SO harus mendukung sedikitnya satu struktur file *executable* sehingga sistem dapat *load* dan menjalankan program.

a. File Attributes

File memiliki nama dan data. Selain itu, ia juga menyimpan informasi meta seperti tanggal dan waktu pembuatan file, ukuran saat ini, tanggal terakhir dimodifikasi, dll. Semua informasi ini disebut atribut sistem file. Berikut adalah beberapa atribut File penting yang digunakan di OS:.

1. **N**ama: Ini adalah satu-satunya informasi yang disimpan dalam bentuk yang dapat dibaca manusia.
2. **P**engidentifikasi: Setiap file diidentifikasi dengan nomor tag unik dalam sistem file yang dikenal sebagai pengidentifikasi.
3. **T**empat: Menunjuk ke lokasi file di perangkat.
4. **J**enis: Atribut ini diperlukan untuk sistem yang mendukung berbagai jenis file.
5. **U**kuran: Atribut yang digunakan untuk menampilkan ukuran file saat ini.

6. **P**erlindungan: Atribut ini menetapkan dan mengontrol hak akses membaca, menulis, dan mengeksekusi file.

7. **W**aktu, tanggal, dan keamanan: Ini digunakan untuk perlindungan, keamanan, dan juga digunakan untuk pemantauan

Atribut ini disimpan dalam struktur direktori sistem file, yang memungkinkan sistem operasi untuk mengelola file dengan lebih efisien.

b. File Operations

1. **File sebagai Tipe Data Abstrak**:
   File adalah konsep dalam komputasi yang mewakili kumpulan data yang disimpan secara terstruktur. Ini diperlakukan sebagai tipe data abstrak (ADT) karena cara data disimpan dan diakses disembunyikan dari pengguna. Sistem mengabstraksi kompleksitas mengakses dan mengelola data, menyediakan struktur logis bagi pengguna untuk berinteraksi dengan file.

2. **Buat (Create)**:
   Pembuatan file dalam sistem file melibatkan alokasi ruang dan menginisialisasi metadata file. Metadata mencakup nama file, jenis, ukuran, dan atribut lain yang diperlukan agar file dikelola dalam sistem. File yang baru dibuat mungkin kosong pada awalnya dan siap untuk ditulis.

3. **Tulis (Write)**:
   Menulis ke file berarti memasukkan data ke dalam file, dimulai dari lokasi penunjuk tulis saat ini. Proses ini menambahkan data baru atau mengganti data yang ada pada posisi yang ditentukan dalam file. Jika file tidak ada, beberapa sistem akan membuat file baru untuk menyimpan data.

4. **Baca (Read)**:
   Operasi baca mengambil data dari file mulai dari lokasi penunjuk baca saat ini. Penunjuk menunjukkan di mana dalam file sistem harus mulai membaca. Setelah data dibaca, data dimuat ke dalam memori untuk digunakan atau diproses lebih lanjut oleh program.

5. **Reposisi dalam File – Cari**:
   Operasi pencarian mengubah posisi penunjuk baca atau tulis dalam

file tanpa membaca atau menulis data apa pun. Ini memungkinkan sistem untuk menavigasi ke berbagai bagian file dengan cepat. Misalnya, pindah ke akhir file untuk menambahkan data atau pergi ke lokasi tertentu untuk mengubah informasi yang ada.

6. **Hapus (Delete)**:
   Operasi hapus menghapus file dari sistem file. Itu dapat menghapus seluruh file atau menandai ruang yang ditempatinya sebagai tersedia untuk data baru, tergantung pada sistem file yang digunakan. Dalam kebanyakan kasus, metadata dan referensi ke file juga dihapus.

7. **Truncate**:
   Pemotongan mengurangi ukuran file dengan memotong kontennya dari titik tertentu dan seterusnya. Operasi ini berguna ketika ukuran file perlu dikurangi, sambil tetap menjaga file itu sendiri dan metadatanya tetap utuh.

8. **Open (Buka)**:
   Saat file dibuka, sistem mengambil metadatanya dari struktur direktori dan memuatnya ke dalam memori. Deskriptor file dibuat, memungkinkan operasi berikutnya (seperti membaca atau menulis) dilakukan. Membuka file juga mengatur penunjuk baca/tulis ke posisi yang sesuai.

9. **Close (Tutup)**:
   Menutup file berarti menyelesaikan operasi yang tersisa (seperti membilas data yang di-buffer ke disk) dan menghapus informasi file dari memori. Operasi ini memastikan bahwa tidak ada operasi lebih lanjut yang dapat dilakukan pada file hingga file tersebut dibuka kembali.

c. Open Files

Manajemen *open files* adalah salah satu elemen penting dalam sistem operasi modern, karena pada dasarnya, setiap proses yang berjalan sering kali berinteraksi dengan file untuk membaca data, menulis informasi, atau mengakses sumber daya tertentu. Oleh karena itu, manajemen *open files* yang efektif akan berdampak langsung pada kinerja. Beberapa komponen utama yang dibutuhkan untuk mengelola file:

1. **Tabel file terbuka**:
   Struktur ini melacak semua file yang sedang dibuka oleh proses. Se-

tiap entri biasanya berisi informasi tentang file, seperti penunjuk file, deskriptor file, dan detail relevan lainnya. Tabel file terbuka sangat penting untuk manajemen file yang efisien dalam sistem operasi. Tanpa tabel ini, sistem operasi akan kesulitan melacak file mana yang terbuka dan oleh proses mana. Tabel ini menyediakan kontrol terpusat untuk penanganan file.

2. **Penunjuk berkas**:
   Ini merujuk pada penunjuk yang menyimpan posisi baca/tulis terakhir dalam berkas. Penunjuk ini biasanya disimpan per proses untuk melacak posisi tertentu tempat operasi akan dilanjutkan. Penunjuk berkas memungkinkan setiap proses mempertahankan posisinya di dalam berkas secara independen. Hal ini penting dalam lingkungan multitugas di mana beberapa proses dapat membaca atau menulis ke berkas yang sama.

3. **Jumlah file yang dibuka**:
   Penghitung digunakan untuk melacak berapa banyak proses yang telah membuka berkas tertentu. Hal ini berguna untuk memastikan bahwa entri berkas dalam tabel berkas yang dibuka hanya dihapus setelah proses terakhir menutup berkas tersebut. Penghitung berkas yang dibuka membantu memastikan bahwa berkas tidak dihapus sebelum waktunya dari tabel berkas yang masih digunakan oleh proses lain. Mekanisme ini menyediakan cara penghitungan referensi yang sederhana namun efektif.

4. **Lokasi disk berkas**:
   Cache yang menyimpan informasi mengenai lokasi fisik berkas pada disk, yang memungkinkan akses lebih cepat selama operasi. *Caching* lokasi disk mengurangi operasi I/O disk dengan melacak data akses file dalam memori, sehingga menghasilkan kecepatan akses yang lebih cepat.

5. **Hak akses**:
   Izin yang dimiliki setiap proses terkait berkas tertentu. Misalnya, suatu proses mungkin memiliki hak baca, tulis, atau eksekusi berdasarkan mode aksesnya. Hak akses menegakkan keamanan dan memastikan bahwa proses hanya dapat berinteraksi dengan berkas sesuai dengan cara yang diizinkan, mencegah akses atau modifikasi yang tidak sah.

d. Tipe File

Tipe File	Ekstensi biasa	Fungsi
Execute	exe, com, bin or none	Baca untuk menjalankan program bahasa mesin
Objek	obj, o	Dikompilasi, bahasa mesin tidak ditautkan
Source code	c, cc, java, pas, asm, a	Kode sumber dalam berbagai bahasa
Batch	bat, sh	Perintah ke penerjemah perintah
Teks	txt, doc	Data tekstual, dokumen
Pengolah	wp, tex, rrf, doc	Berbagai format pengolah kata
Library	lib, a, so	Berisi pustaka rutinitas untuk pemrograman
Cetak atau Lihat	arc, zip, tar	Format untuk mencetak atau melihat file ASCII atau biner.
Arsip	arc, zip, tar	File-file terkait dikelompokkan menjadi satu file terkompresi untuk arsip atau penyimpanan.
Multimedia	mpeg, mov, rm	Berisi informasi audio/video

Table 1: Tipe File

Salah satu pertimbangan penting dalam merancang sistem file dan keseluruhan sistem operasi adalah apakah sistem operasi mengenali dan mendukung sistem file. Bila sistem operasi mengenali tipe suatu file, maka dapat dilakukan operasi terhadap file dengan lebih baik.

UNIX menggunakan *magic number* yang disimpan pada awal file untuk mengindikasikan tipe file berupa program *executable*, *batch file* (shell script), file *postscript*, dan lain-lain. Tidak semua file mempunyai *magic number*, sehingga informasi tipe tidak dapat digambarkan. UNIX tidak menyimpan nama dari program pembuatnya. UNIX juga mengizinkan nama ekstensi dari file tersembunyi, sehingga pengguna dapat menentukan tipe file sendiri dan tidak tergantung pada sistem operasi.

3.8.2 F

ile access methods

3.8.3 D

irectory 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 2: Your table caption

4.1.2

4.1.3

4.1.4

4.1.5

4.1.6

4.1.7

4.1.8 Group 8

Implementasikan algoritma penjadwalan proses *First Come First Serve* (FCFS). Ada 4 proses dengan *arrival time* dan *burst time* sebagai berikut:

- P1: Arrival Time = 0, Burst Time = 5
- P2: Arrival Time = 1, Burst Time = 3
- P3: Arrival Time = 2, Burst Time = 8
- P4: Arrival Time = 3, Burst Time = 6

Hitung *waiting time* dan *turnaround time* untuk setiap proses.

Proses	Arrival Time	Burst Time
1	0	5
2	1	3
3	2	8
4	3	6

Table 3: Tabel Data Proses

```
# FCFS Scheduling
def fcfs_scheduling(processes, burst_time, arrival_time):
    n = len(processes)
    waiting_time = [0] * n
    turnaround_time = [0] * n

    # Service time calculation
    service_time = [0] * n
    service_time[0] = arrival_time[0]

    for i in range(1, n):
        service_time[i] = service_time[i - 1] + burst_time[i
                                                    - 1]
        waiting_time[i] = service_time[i] - arrival_time[i]
        waiting_time[i] = max(waiting_time[i], 0)

    # Turnaround time calculation
    for i in range(n):
        turnaround_time[i] = burst_time[i] + waiting_time[i]

    avg_waiting_time = sum(waiting_time) / n
    avg_turnaround_time = sum(turnaround_time) / n

    return waiting_time, turnaround_time, avg_waiting_time,
                                        avg_turnaround_time
```

```
# SJN Scheduling (non-preemptive)
def sjn_scheduling(processes, burst_time, arrival_time):
    n = len(processes)
    waiting_time = [0] * n
    turnaround_time = [0] * n
    completed = [False] * n
    time = 0
    completed_count = 0

    while completed_count < n:
        min_bt = float('inf')
        min_index = -1

        for i in range(n):
            if arrival_time[i] <= time and not completed[i]
                                        and burst_time[i]
                                        < min_bt:
                min_bt = burst_time[i]
                min_index = i

        if min_index == -1:
            time += 1
            continue

        waiting_time[min_index] = time - arrival_time[
                                        min_index]
        time += burst_time[min_index]
        turnaround_time[min_index] = time - arrival_time[
                                        min_index]
        completed[min_index] = True
        completed_count += 1

    avg_waiting_time = sum(waiting_time) / n
    avg_turnaround_time = sum(turnaround_time) / n

    return waiting_time, turnaround_time, avg_waiting_time,
                                    avg_turnaround_time

# Round Robin Scheduling
def rr_scheduling(processes, burst_time, arrival_time,
                                time_quantum):
    n = len(processes)
    waiting_time = [0] * n
    turnaround_time = [0] * n
```

```
    remaining_bt = burst_time[:]
    time = 0
    complete = 0

    while complete < n:
        for i in range(n):
            if remaining_bt[i] > 0:
                if remaining_bt[i] > time_quantum:
                    time += time_quantum
                    remaining_bt[i] -= time_quantum
                else:
                    time += remaining_bt[i]
                    waiting_time[i] = time - burst_time[i] -
                                                    arrival_time
                                                    [i]
                    turnaround_time[i] = time - arrival_time[
                                                    i]
                    remaining_bt[i] = 0
                    complete += 1

    avg_waiting_time = sum(waiting_time) / n
    avg_turnaround_time = sum(turnaround_time) / n

    return waiting_time, turnaround_time, avg_waiting_time,
                                    avg_turnaround_time

# Fungsi untuk membandingkan algoritma
def compare_algorithms(processes, burst_time, arrival_time,
                                time_quantum):
    print("== FCFS ==")
    wt_fcfs, tat_fcfs, avg_wt_fcfs, avg_tat_fcfs =
                                    fcfs_scheduling(processes,
                                     burst_time, arrival_time)
    print(f"Average Waiting Time: {avg_wt_fcfs:.2f}")
    print(f"Average Turnaround Time: {avg_tat_fcfs:.2f}\n")

    print("== SJN ==")
    wt_sjn, tat_sjn, avg_wt_sjn, avg_tat_sjn = sjn_scheduling
                                    (processes, burst_time,
                                    arrival_time)
    print(f"Average Waiting Time: {avg_wt_sjn:.2f}")
    print(f"Average Turnaround Time: {avg_tat_sjn:.2f}\n")

    print("== Round Robin ==")
    wt_rr, tat_rr, avg_wt_rr, avg_tat_rr = rr_scheduling(
```

```
                                    processes, burst_time,
                                    arrival_time, time_quantum
                                    )
    print(f"Average Waiting Time: {avg_wt_rr:.2f}")
    print(f"Average Turnaround Time: {avg_tat_rr:.2f}\n")

# Data proses
processes = ['P1', 'P2', 'P3', 'P4']
burst_time = [5, 3, 8, 6]
arrival_time = [0, 1, 2, 3]
time_quantum = 2

compare_algorithms(processes, burst_time, arrival_time,
                                time_quantum)
```

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

4.3.2

4.3.3

4.3.4

4.3.5

4.3.6

4.3.7 Group 8

Buat sebuah skenario *multithreading* untuk menyelesaikan masalah komputasi *intensif*, seperti menghitung elemen terbesar dalam sebuah *array* be-

sar. Implementasikan kode Python untuk menghitung elemen terbesar menggunakan *multithreading* dan hitung *speedup teoritis* menggunakan *Amdahl's Law* saat jumlah *thread* meningkat.

```
import threading
import random
import time


def find_max(arr, result, index):
    result[index] = max(arr)


def amdahls_law(serial_fraction, n_threads):
    parallel_fraction = 1 - serial_fraction
    speedup = 1 / (serial_fraction + (parallel_fraction /
                                      n_threads))
    return speedup


def multithreaded_max(arr, num_threads):
    chunk_size = len(arr) // num_threads
    threads = []
    results = [0] * num_threads

    for i in range(num_threads):
        chunk = arr[i*chunk_size : (i+1)*chunk_size] if i
                                        != num_threads -
                                       1 else arr[i*
                                       chunk_size:]
        t = threading.Thread(target=find_max, args=(chunk
                                       , results, i))
        threads.append(t)
        t.start()

    for t in threads:
        t.join()

    overall_max = max(results)
    return overall_max


arr_size = 1000000
arr = [random.randint(1, 1000000) for _ in range(arr_size
                                    )]
```

```
n_threads_list = [1, 2, 4, 8]
serial_fraction = 0.1  # Assuming 10% of the code is not
                                parallelizable

print("Finding the maximum element in the array...")
for n_threads in n_threads_list:
    start_time = time.time()
    max_element = multithreaded_max(arr, n_threads)
    end_time = time.time()

    elapsed_time = end_time - start_time
    print(f"Number of threads: {n_threads}, Maximum
                                element: {max_element}
                                , Time taken: {
                                elapsed_time:.4f}
                                seconds")


    theoretical_speedup = amdahls_law(serial_fraction,
                                n_threads)
    print(f"Theoretical Speedup with {n_threads} threads:
                                 {theoretical_speedup
                                :.2f}\n")
```

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.