OS CA-2

(Roll nos. 51 to 60)

Name - Komal Singh
Roll no - 58
Division - D10B

Module 4 - Write a C program to implement Paging technique for memory management.

→ **Code -**

```
#include <stdio.h>
#include <stdlib.h>

#define PAGE_SIZE 4096
#define NUM_PAGES 256

// Function to simulate paging technique for memory management
void paging(int process_size, int page_table[]) {
    int num_pages_needed = process_size / PAGE_SIZE;
    if (process_size % PAGE_SIZE != 0)
        num_pages_needed++;

    printf("Number of pages needed for the process: %d\n", num_pages_needed);

    printf("Page Table:\n");
    for (int i = 0; i < num_pages_needed; i++) {
        page_table[i] = rand() % NUM_PAGES; // Assigning random frame numbers for simulation
        printf("Page %d -> Frame %d\n", i, page_table[i]);
    }
}
```

```
int main() {
    int process_size;
    int page_table[NUM_PAGES];

    printf("Enter the size of the process (in bytes): ");
    scanf("%d", &process_size);

    paging(process_size, page_table);

    return 0;
}
```

Output -

```
Enter the size of the process (in bytes): 1024
Number of pages needed for the process: 1
Page Table:
Page 0 -> Frame 103


=== Code Execution Successful ===
```

Module 5 - Implement various disk scheduling algorithms like LOOK, C-LOOK in C/Python/Java.

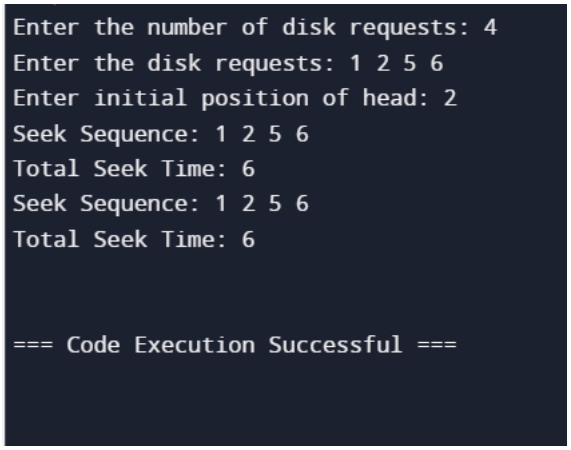
→ Code -

```
#include <stdio.h>
#include <stdlib.h>

// Function to implement LOOK disk scheduling algorithm
void look(int requests[], int head, int size) {
    int totalSeekTime = 0;
    int cur_track = head;
```

```
    printf("Seek Sequence: ");
    for (int i = 0; i < size; ++i) {
        printf("%d ", requests[i]);
        totalSeekTime += abs(cur_track - requests[i]);
        cur_track = requests[i];
    }
    printf("\nTotal Seek Time: %d\n", totalSeekTime);
}

// Function to implement C-LOOK disk scheduling algorithm
void c_look(int requests[], int head, int size) {
    int totalSeekTime = 0;
    int cur_track = head;

    printf("Seek Sequence: ");
    for (int i = 0; i < size; ++i) {
        printf("%d ", requests[i]);
        totalSeekTime += abs(cur_track - requests[i]);
        cur_track = requests[i];
    }
    printf("\nTotal Seek Time: %d\n", totalSeekTime);
}

int main() {
    int size, head;

    printf("Enter the number of disk requests: ");
    scanf("%d", &size);

    int *requests = (int *)malloc(size * sizeof(int));

    if (requests == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
```

```
    }

    printf("Enter the disk requests: ");
    for (int i = 0; i < size; i++) {
        scanf("%d", &requests[i]);
    }

    printf("Enter initial position of head: ");
    scanf("%d", &head);

    // Look Algorithm
    look(requests, head, size);

    // C-Look Algorithm
    c_look(requests, head, size);

    free(requests);

    return 0;
}
```

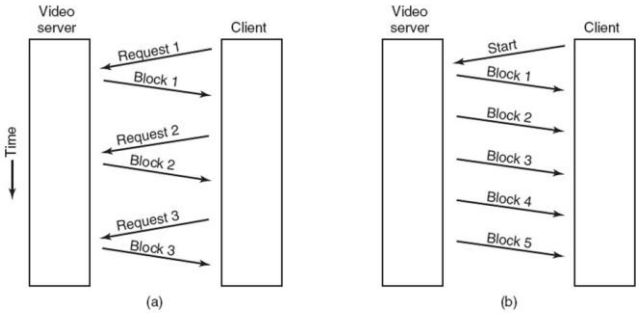
Output -



Module 6 - Case Study on Multimedia Operating System.

→

Multimedia File System Paradigms



- Figure 7-16. (a) A pull server. (b) A push server.

Introduction:

A Multimedia Operating System (OS) is tailored to efficiently manage multimedia data such as audio, video, images, and graphical content. These OSs prioritize seamless multimedia processing, playback, and rendering, catering to the diverse needs of multimedia applications.

Components:

1. Kernel: The core component responsible for managing system resources and providing essential functionalities for multimedia processing.
2. Multimedia Framework: A comprehensive framework that facilitates multimedia data handling, including encoding, decoding, rendering, and streaming.
3. Media Player: A dedicated application or subsystem for playing multimedia content, offering features such as playback controls, playlist management, and codec support.
4. Graphics Subsystem: A specialized subsystem equipped with drivers and libraries for rendering high-quality graphics and supporting advanced rendering techniques.

Key Features:

1. Real-Time Processing: Multimedia OSs prioritize real-time multimedia processing to ensure smooth playback and responsiveness, minimizing latency and buffering delays.
2. Codec Support: Extensive codec support for encoding and decoding multimedia formats, ensuring compatibility with a wide range of audio, video, and image file types.
3. Streaming Capabilities: Built-in support for multimedia streaming protocols, enabling seamless streaming of audio and video content over networks and the internet.

4. Hardware Acceleration: Integration with hardware acceleration features of GPUs and specialized multimedia processors to enhance multimedia rendering and processing performance.
5. User Interface: A user-friendly interface optimized for multimedia consumption, featuring intuitive controls, customizable themes, and interactive media playback options.

Challenges:

1. Resource Intensive: Multimedia processing tasks demand significant computational resources, including CPU, GPU, and memory, posing challenges for resource allocation and optimization.
2. Compatibility Issues: Ensuring compatibility with a diverse array of multimedia formats, codecs, and protocols presents challenges in multimedia OS development and maintenance.
3. Power Management: Multimedia applications can be power-intensive, consuming substantial battery resources, necessitating efficient power management strategies to prolong device battery life.
4. Security Concerns: Multimedia OSs must address security vulnerabilities associated with multimedia file formats and streaming protocols, safeguarding against malware and privacy breaches.

Real-World Applications:

1. Android: Android OS powers a vast array of smartphones and tablets, offering a robust multimedia framework with extensive codec support and a rich ecosystem of multimedia applications.
2. iOS: Apple's iOS powers iPhones, iPads, and iPod Touch devices, providing a seamless multimedia experience with high-quality audio and video playback and integration with the Apple ecosystem.
3. Windows Multimedia Platform: Microsoft's Windows platform offers multimedia capabilities through Windows Media Player and a range of multimedia APIs and tools for developers to create multimedia-rich applications.

Conclusion:

Multimedia Operating Systems play a pivotal role in delivering immersive multimedia experiences on modern computing devices. Despite the challenges posed by resource constraints, compatibility issues, and security concerns, multimedia OSs continue to evolve, driving innovation in multimedia processing, playback, and content creation.