OS CA-2

(Roll nos. 51 to 60)

NAME : Arjun Singh Silwal

DIVISION : D10B

ROLL NO. : 57

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("Pages required 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 process size in bytes: ");
    scanf("%d", &process_size);

    paging(process_size, page_table);

    return 0;
}
```

Output -

```
Enter process size in bytes: 1024
Pages required for the process: 1
Page Table:
Page 0 -> Frame 103
```

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 ", cur_track);
        totalSeekTime += abs(cur_track - requests[i]);
        cur_track = requests[i];
    }
    printf("\nTotal Seek Time: %d\n", totalSeekTime);
}

int main() {
    int size, head;

    printf("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("Initial head position : ");
    scanf("%d", &head);

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

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

    free(requests);

    return 0;
}
```

Output -

```
Number of disk requests: 5
Enter the disk requests: 7
4
9
8
2
Initial head position : 8
Seek Sequence: 7 4 9 8 2
Total Seek Time: 16
Seek Sequence: 8 7 4 9 8
Total Seek Time: 16
```

Module 6

Case Study: Case Study on Multimedia Operating System.

Title: Optimizing Power Management in Multimedia Operating Systems

Introduction:

➔ Multimedia Operating Systems (OS) play a crucial role in facilitating seamless multimedia processing and playback on modern computing devices. However, the resource-intensive nature of multimedia tasks poses significant challenges in managing power consumption efficiently. This case study explores the strategies and technologies employed in optimizing power management within multimedia operating systems to enhance device battery life while maintaining multimedia performance.

Challenges:

➔ Resource Intensive Tasks: Multimedia processing tasks such as audio/video decoding, image rendering, and graphics processing demand significant computational resources, leading to increased power consumption.
➔ Real-Time Requirements: Multimedia applications often require real-time processing to ensure smooth playback and responsiveness, limiting the scope for power-saving mechanisms.
➔ Heterogeneous Workloads: Multimedia operating systems must balance power consumption across diverse multimedia tasks with varying resource requirements and priorities.
➔ User Expectations: Users expect uninterrupted multimedia playback and performance, making it challenging to implement aggressive power-saving measures without compromising user experience.

Strategies and Technologies:

➔ Dynamic Voltage and Frequency Scaling (DVFS): Multimedia operating systems employ DVFS techniques to adjust CPU frequency and voltage dynamically based on workload requirements, reducing power consumption during periods of low multimedia activity.
➔ Adaptive Display Brightness: Automatic adjustment of display brightness based on ambient lighting conditions and multimedia content reduces power consumption without sacrificing visibility or image quality.
➔ Hardware Acceleration: Leveraging hardware acceleration features of GPUs and dedicated multimedia processors for tasks such as video decoding and image processing minimizes CPU usage and conserves power.

➔ Context-Aware Power Management: Implementing context-aware power management algorithms that consider user behavior, application usage patterns, and multimedia content characteristics enables more precise power optimization.

Real-World Applications:

➔ Android: Android OS incorporates advanced power management features such as Doze mode and App Standby to optimize power consumption during multimedia playback and idle periods.
➔ iOS: Apple's iOS employs sophisticated power management techniques, including low-power hardware acceleration for multimedia tasks and adaptive screen brightness adjustment based on ambient light sensors.
➔ Windows Multimedia Platform: Microsoft Windows integrates power-saving features such as Connected Standby mode and Adaptive Display Brightness to enhance battery life without compromising multimedia performance.

Conclusion:

➔ Optimizing power management in multimedia operating systems is essential for prolonging device battery life while maintaining optimal multimedia performance. By employing a combination of dynamic power-saving techniques, context-aware algorithms, and energy-efficient technologies, multimedia OSs can strike a balance between power efficiency and multimedia functionality, providing users with a seamless and energy-efficient multimedia experience on their computing devices.