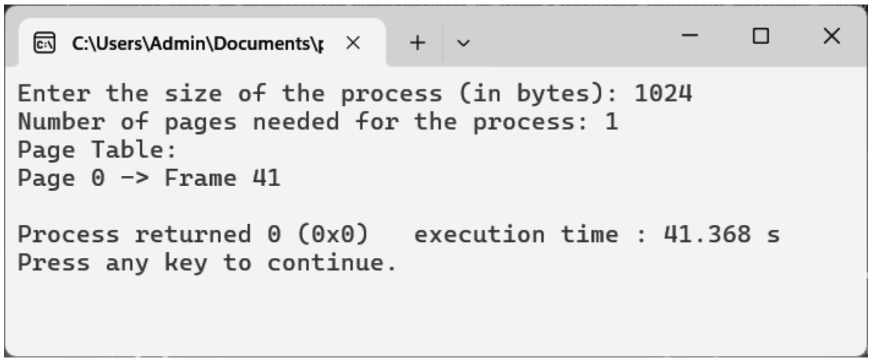
OPERATING SYSTEM
CONTINUOUS ASSESSMENT - 2

NAME: **Aryan Deeepak Saraf**
DIV: **D10B**
ROLL NO.: **51**

Q.1) 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;
}
```



```
Enter the size of the process (in bytes): 1024
Number of pages needed for the process: 1
Page Table:
Page 0 -> Frame 41

Process returned 0 (0x0)   execution time : 41.368 s
Press any key to continue.
```

Q.2) Implement various disk scheduling algorithms like LOOK, C-LOOK in C/Python/Java.

1. LOOK algorithm in python.

```
def look(arr, head, direction):
    seek_sequence = []

    Splitting requests into two parts:
    1. Requests below the current head position
    2. Requests above the current head position
    lower_requests = [req for req in arr if req < head]
    upper_requests = [req for req in arr if req > head]

    lower_requests.sort(reverse=True)
    upper_requests.sort()

    Adding head position as the initial point
    seek_sequence.append(head)

    Traversing in the chosen direction
    if direction == "left":
        for req in lower_requests:
            seek_sequence.append(req)

        for req in upper_requests:
            seek_sequence.append(req)
    else:
        for req in upper_requests:
            seek_sequence.append(req)

        for req in lower_requests:
            seek_sequence.append(req)

    return seek_sequence

def calculate_seek_operations(sequence):
    operations = 0
    for i in range(1, len(sequence)):
        operations += abs(sequence[i] - sequence[i-1])
    return operations

Example
```

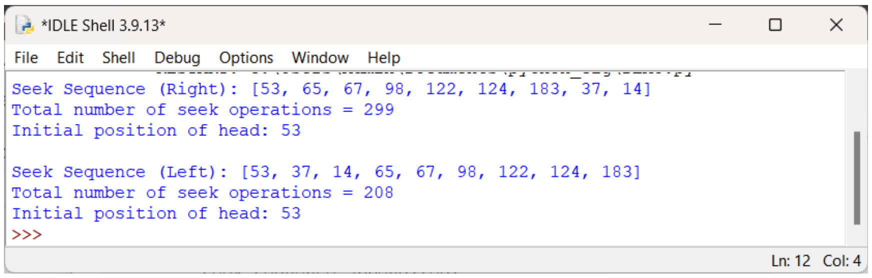
```
requests = [98, 183, 37, 122, 14, 124, 65, 67]
initial_head = 53

initial_direction = "right"

sequence = look(requests, initial_head, initial_direction)
print("Seek Sequence (Right):", sequence)
print("Total number of seek operations =", calculate_seek_operations(sequence))
print("Initial position of head:", initial_head)

initial_direction = "left"

sequence = look(requests, initial_head, initial_direction)
print("\nSeek Sequence (Left):", sequence)
print("Total number of seek operations =", calculate_seek_operations(sequence))
print("Initial position of head:", initial_head)
```



```
Seek Sequence (Right): [53, 65, 67, 98, 122, 124, 183, 37, 14]
Total number of seek operations = 299
Initial position of head: 53

Seek Sequence (Left): [53, 37, 14, 65, 67, 98, 122, 124, 183]
Total number of seek operations = 208
Initial position of head: 53
>>>
```

2. C-LOOK algorithm in python.

```
def c_look(arr, head, direction):
    seek_sequence = []

    Splitting requests into two parts:
    1. Requests below the current head position
    2. Requests above the current head position
    lower_requests = [req for req in arr if req < head]
    upper_requests = [req for req in arr if req > head]

    lower_requests.sort()
    upper_requests.sort()

    Adding head position as the initial point
    seek_sequence.append(head)

    Traversing in the chosen direction
    if direction == "left":
        for req in lower_requests:
            seek_sequence.append(req)
        for req in upper_requests:
            seek_sequence.append(req)
    else:
        for req in upper_requests:
            seek_sequence.append(req)

        for req in lower_requests:
            seek_sequence.append(req)

    return seek_sequence

def calculate_seek_operations(sequence):
    operations = 0
    for i in range(1, len(sequence)):
        operations += abs(sequence[i] - sequence[i-1])
    return operations
```

Example

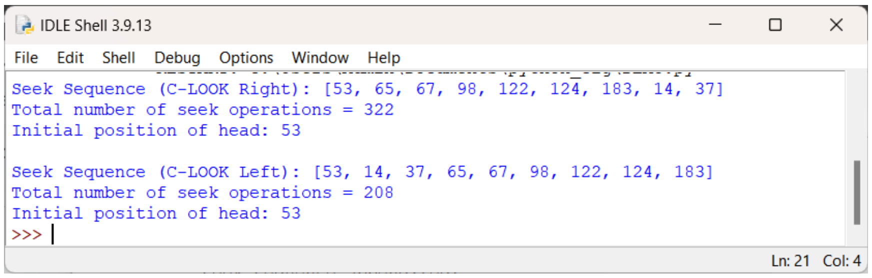
```
requests = [98, 183, 37, 122, 14, 124, 65, 67]
initial_head = 53

initial_direction = "right"

sequence = c_look(requests, initial_head, initial_direction)
print("Seek Sequence (C-LOOK Right):", sequence)
print("Total number of seek operations =", calculate_seek_operations(sequence))
print("Initial position of head:", initial_head)

initial_direction = "left"

sequence = c_look(requests, initial_head, initial_direction)
print("\nSeek Sequence (C-LOOK Left):", sequence)
print("Total number of seek operations =", calculate_seek_operations(sequence))
print("Initial position of head:", initial_head)
```



```
Seek Sequence (C-LOOK Right): [53, 65, 67, 98, 122, 124, 183, 14, 37]
Total number of seek operations = 322
Initial position of head: 53

Seek Sequence (C-LOOK Left): [53, 14, 37, 65, 67, 98, 122, 124, 183]
Total number of seek operations = 208
Initial position of head: 53
>>> 
```

Q.3) Case Study on Mobile Operating System.

Multimedia Operating System - Raspberry Pi OS (formerly Raspbian)

Introduction

Raspberry Pi OS, formerly known as Raspbian, is a popular operating system designed for the Raspberry Pi single-board computers. While it is a general-purpose OS, its lightweight nature and extensive software support make it suitable for multimedia applications, especially on the Raspberry Pi platform. This case study explores how Raspberry Pi OS caters to multimedia needs and its advantages and challenges in this context.

Background

Raspberry Pi OS is based on Debian Linux and optimized for the ARM architecture, which powers the Raspberry Pi boards. It comes pre-installed with a variety of multimedia software and tools, making it easy for users to create, edit, and play multimedia content right out of the box.

Features

1. Multimedia Software Bundle
Raspberry Pi OS includes a bundle of multimedia software such as VLC media player, Audacity audio editor, and GIMP image editor. These tools provide users with the necessary functionalities to handle audio, video, and image files effectively.

2. Hardware Acceleration
The OS supports hardware-accelerated graphics and video playback, leveraging the GPU capabilities of the Raspberry Pi. This ensures smooth and high-quality multimedia playback, even on the low-powered Raspberry Pi devices.

3. GPIO Support
For users interested in integrating multimedia projects with physical components, Raspberry Pi OS offers GPIO (General-Purpose Input/Output) support. This allows for interactive multimedia applications where sensors, buttons, and LEDs can be controlled programmatically.

4. Easy Installation and Updates
Raspberry Pi OS provides a user-friendly installer and a simple package management system, making it easy for users to install additional multimedia software and keep their system updated with the latest features and security patches.

5. Community Support
Being a popular OS for Raspberry Pi, Raspberry Pi OS benefits from a vibrant community of developers and enthusiasts. This community support includes forums, tutorials, and open-source contributions, making it easier for users to troubleshoot issues, find resources, and collaborate on multimedia projects.

Advantages

1. Cost-Effective Solution
Raspberry Pi OS offers a cost-effective solution for multimedia applications, especially when paired with Raspberry Pi hardware. This makes it accessible to hobbyists, educators, and small businesses looking to create multimedia projects on a budget.

2. Extensive Software Support
The availability of a wide range of multimedia software and tools, both pre-installed and through package repositories, makes Raspberry Pi OS a versatile platform for multimedia editing, playback, and streaming.

3. Community Engagement
The active community surrounding Raspberry Pi OS provides valuable support, resources, and collaborative opportunities for users interested in multimedia projects. This community-driven approach fosters innovation and knowledge sharing.

Challenges

1. Hardware Limitations
While Raspberry Pi OS is capable of handling multimedia tasks, the hardware limitations of Raspberry Pi boards, such as limited processing power and memory, can impact performance, especially for resource-intensive applications.

2. Learning Curve
For users new to Linux or Raspberry Pi, there can be a learning curve associated with setting up and configuring the OS for multimedia applications. This may require additional time and effort to become familiar with the platform and its ecosystem.

3. Customization and Scalability
While Raspberry Pi OS offers a range of multimedia software and tools, its general-purpose nature may not meet the specific requirements of advanced or specialized multimedia projects. Customization and scalability options may be limited compared to more specialized multimedia operating systems.

Conclusion

Raspberry Pi OS, with its lightweight nature, extensive software support, and community engagement, serves as a viable platform for multimedia applications, particularly on Raspberry Pi hardware. While it faces challenges related to hardware limitations, learning curve, and customization, its cost-effectiveness, versatility, and community-driven ecosystem make it a popular choice among hobbyists, educators, and small businesses looking to explore multimedia projects on a budget.