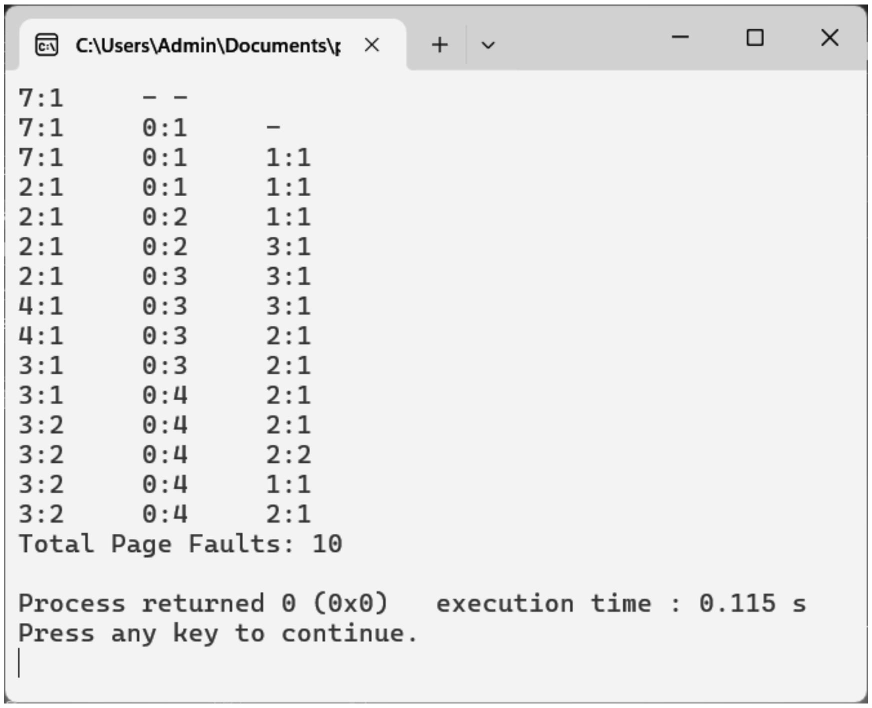
OPERATING SYSTEM
CONTINUOUS ASSESSMENT - 2

NAME: **Naikwadi Yash Shivdas**
DIV: **D10B**
ROLL NO.: **42**

Q.1) To write a c program to implement the LFU page replacement algorithm.

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

#define FRAME_SIZE 3
#define INVALID_PAGE -1

typedef struct {
   int page_number;
   int frequency;
   int timestamp;
} Page;

void initialize_frame(Page frame[], int n) {
   for (int i = 0; i < n; i++) {
      frame[i].page_number = INVALID_PAGE;
      frame[i].frequency = 0;
      frame[i].timestamp = -1;
   }
}

int find_least_frequent(Page frame[], int n) {
   int min_frequency = frame[0].frequency;
   int min_timestamp = frame[0].timestamp;
   int index = 0;

   for (int i = 1; i < n; i++) {
      if (frame[i].frequency < min_frequency ||
         (frame[i].frequency == min_frequency && frame[i].timestamp < min_timestamp)) {
          min_frequency = frame[i].frequency;
          min_timestamp = frame[i].timestamp;
          index = i;
      }
   }

   return index;
}
```

```
void print_frame(Page frame[], int n) {
   for (int i = 0; i < n; i++) {
      if (frame[i].page_number != INVALID_PAGE) {
         printf("%d:%d\t", frame[i].page_number, frame[i].frequency);
      } else {
         printf("- ");
      }
   }
   printf("\n");
}

int main() {
   int page_requests[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2};
   int n = sizeof(page_requests) / sizeof(page_requests[0]);

   Page frame[FRAME_SIZE];
   initialize_frame(frame, FRAME_SIZE);

   int page_faults = 0;
   int timestamp = 0;

   for (int i = 0; i < n; i++) {
      bool page_hit = false;

      // Check if page is already in frame
      for (int j = 0; j < FRAME_SIZE; j++) {
         if (frame[j].page_number == page_requests[i]) {
            frame[j].frequency++;
            page_hit = true;
            break;
         }
      }

      if (!page_hit) {
         // Page fault occurred
         int empty_frame = -1;
         for (int j = 0; j < FRAME_SIZE; j++) {
            if (frame[j].page_number == INVALID_PAGE) {
               empty_frame = j;
               break;
            }
         }

         if (empty_frame != -1) {
                frame[empty_frame].page_number = page_requests[i];
                frame[empty_frame].frequency = 1;
                frame[empty_frame].timestamp = timestamp++;
            } else {
                int least_freq_index = find_least_frequent(frame, FRAME_SIZE);
                frame[least_freq_index].page_number = page_requests[i];
                frame[least_freq_index].frequency = 1;
                frame[least_freq_index].timestamp = timestamp++;
            }

            page_faults++;
        }

        print_frame(frame, FRAME_SIZE);
    }

    printf("Total Page Faults: %d\n", page_faults);

    return 0;
}
```



C:\Users\Admin\Documents\p

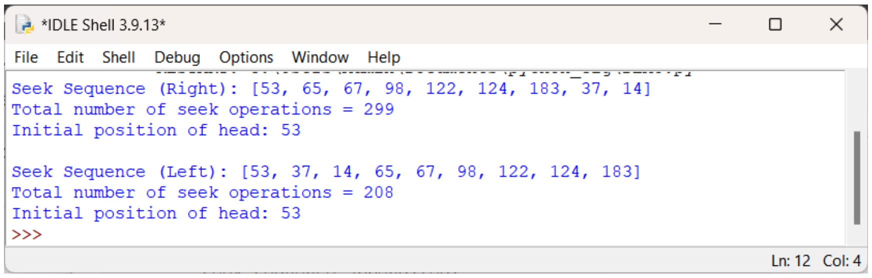
```
7:1     - -
7:1     0:1     -
7:1     0:1     1:1
2:1     0:1     1:1
2:1     0:2     1:1
2:1     0:2     3:1
2:1     0:3     3:1
4:1     0:3     3:1
4:1     0:3     2:1
3:1     0:3     2:1
3:1     0:4     2:1
3:2     0:4     2:1
3:2     0:4     2:2
3:2     0:4     1:1
3:2     0:4     2:1
Total Page Faults: 10

Process returned 0 (0x0)   execution time : 0.115 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
```

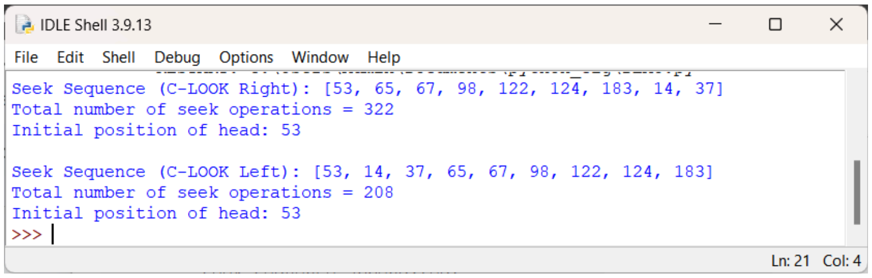
```
initial_direction = "right"

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.

Samsung Tizen Mobile Operating System

Introduction:

Tizen, developed by Samsung Electronics, is an open-source, Linux-based mobile operating system that serves as an alternative to Android and iOS. Initially launched in 2012, Tizen has evolved over the years to power a variety of Samsung devices, including smartphones, smartwatches, smart TVs, and IoT devices. This case study explores the history, market strategy, key features, app ecosystem, security, and challenges faced by the Tizen operating system.

History:

1. Developed by Samsung Electronics in collaboration with Intel and other industry partners.
2. Launched in 2012 with the goal of creating a flexible and scalable operating system for a wide range of connected devices.

Market Strategy:

1. Initially targeted at smart TVs, wearables, and IoT devices before expanding to smartphones.
2. Samsung's commitment to integrating Tizen across its product lineup, combined with strategic partnerships and collaborations, aims to establish Tizen as a viable alternative to Android and iOS.

Key Features:

1. Versatility: Designed to support a diverse range of devices, from smartphones and tablets to smart TVs, wearables, and IoT devices, offering a unified and seamless user experience across devices.
2. Performance and Efficiency: Optimized for performance, energy efficiency, and resource management, ensuring smooth and responsive operation even on lower-end hardware.
3. Customization and Flexibility: Offers a customizable user interface, support for native and web-based applications, and flexibility for manufacturers and developers to tailor and optimize the OS according to their specific requirements.

App Ecosystem:

1. Samsung Galaxy Store serves as the primary app store for Tizen-powered devices, offering a curated selection of apps and services tailored for Samsung devices.
2. Collaboration with developers and partners to expand the app ecosystem, improve app quality, and attract popular apps and services to Tizen platform.

Security:

1. Incorporates advanced security features and protocols, including secure boot, device encryption, and app sandboxing, to protect user data and ensure the integrity and confidentiality of the system.

Challenges:

1. Ecosystem Building: Building and maintaining a robust and diverse ecosystem of apps, services, and content is crucial for Tizen's success and competitiveness against established players like Android and iOS.
2. Market Adoption: Despite Samsung's efforts to promote and integrate Tizen across its product lineup, achieving widespread market adoption and convincing consumers to switch from familiar platforms like Android or iOS remains a significant challenge.
3. Developer Support: Attracting and retaining developers to create high-quality apps and services for Tizen platform, especially compared to the larger and more established ecosystems of Android and iOS, requires incentives, support, and collaboration from Samsung and partners.

Conclusion:

Tizen, with its versatility, performance, and Samsung's strong backing, presents a compelling alternative to traditional mobile operating systems like Android and iOS. While it may face challenges in terms of ecosystem building, market adoption, and developer support, Tizen's potential to power a wide range of connected devices, combined with Samsung's global reach, brand recognition, and strategic initiatives, positions it as a significant player in the competitive mobile operating system landscape. As Samsung continues to invest in Tizen's development, integration, and promotion across its product portfolio, the evolution, adoption, and success of Tizen will be instrumental in shaping the future of connected devices and ecosystems.