

OPERATING SYSTEM
CONTINUOUS ASSESSMENT - 2

NAME: **Yash Santosh Rahate**

DIV: **D10B**

ROLL NO.: **48**

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++;
    }

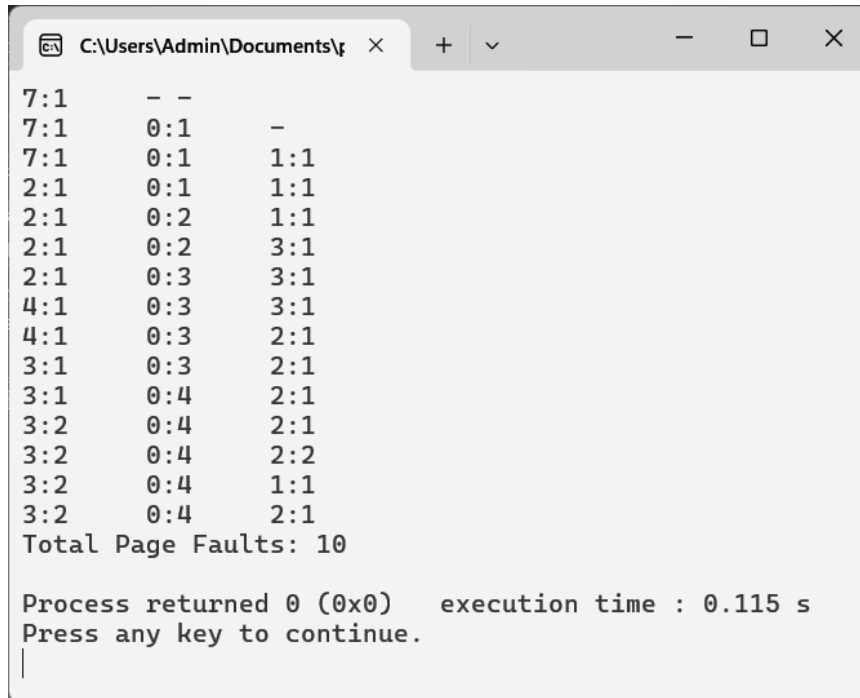
    page_faults++;
}

print_frame(frame, FRAME_SIZE);
}

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

return 0;
}

```



```

C:\Users\Admin\Documents\...  x  +  -  □  x
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

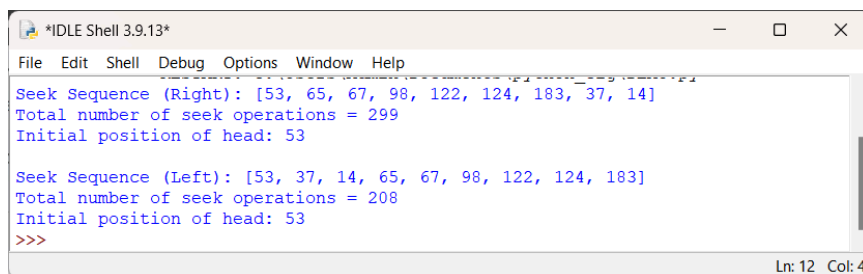
# Example
requests = [98, 183, 37, 122, 14, 124, 65, 67]
initial_head = 53
```

```
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)
```



```
*IDLE Shell 3.9.13*
File Edit Shell Debug Options Window Help
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

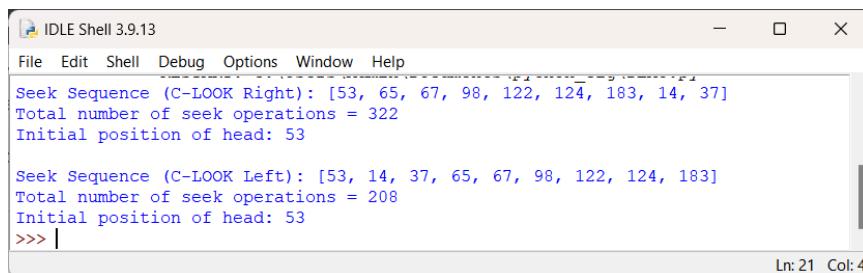
initial_direction = "right"

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)

```



```

IDLE Shell 3.9.13
File Edit Shell Debug Options Window Help
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
>>>
Ln: 21 Col: 4

```

Q.3) Case Study on Mobile Operating System.

Title: NexOS - Revolutionizing the Mobile Experience

Introduction:

In the ever-evolving landscape of mobile technology, the demand for innovative operating systems continues to rise. This case study delves into the development process and features of NexOS, a next-generation mobile operating system designed to provide users with an unparalleled experience in terms of performance, security, and versatility.

Background:

NexOS is being developed by a team of engineers and designers at NexCorp, a leading technology company known for its groundbreaking products. The inspiration behind NexOS stems from the need to address the shortcomings of existing mobile operating systems while introducing new features and functionalities that cater to the evolving needs of users worldwide.

Objective:

1. The primary objective of NexOS is to redefine the mobile experience by offering:
2. Enhanced Performance: NexOS aims to deliver lightning-fast performance, ensuring smooth multitasking, app launches, and overall system responsiveness.
3. Advanced Security: Security is a top priority for NexOS, with robust encryption, biometric authentication, and privacy features built into the core of the operating system to safeguard user data.
4. Seamless Integration: NexOS seamlessly integrates with other devices and platforms, allowing users to effortlessly transition between their mobile devices, computers, smart home devices, and more.
5. Personalization: NexOS offers extensive customization options, empowering users to tailor their devices to suit their preferences and lifestyle.

Development Process:

The development of NexOS follows an iterative process, involving several key stages:

1. Research and Planning: The development team conducts extensive research to identify market trends, user preferences, and technological advancements. Based on this research, a comprehensive plan is devised outlining the features, architecture, and timeline for the project.
2. Design and Prototyping: UX/UI designers collaborate to create intuitive interfaces and visually appealing designs for NexOS. Prototypes are developed and tested to gather feedback from users, which is then incorporated into the design iterations.

3. **Engineering and Development:** Software engineers leverage cutting-edge technologies and programming languages to build the core components of NexOS, including the kernel, drivers, system services, and APIs. Agile methodologies are employed to ensure flexibility and adaptability throughout the development process.
4. **Testing and Quality Assurance:** Rigorous testing procedures are implemented to identify and address any bugs, errors, or performance issues in NexOS. Automated testing, user testing, and beta testing are conducted to validate the stability, security, and functionality of the operating system.
5. **Deployment and Release:** Once NexOS has undergone thorough testing and received approval from stakeholders, it is deployed to manufacturing partners for integration into new devices. Regular updates and patches are released to continuously improve and optimize the user experience.

Key Features of NexOS:

1. **NexUI:** A sleek and intuitive user interface designed for seamless navigation and customization.
2. **NexSecure:** Advanced security features, including facial recognition, fingerprint authentication, and secure enclave technology.
3. **NexConnect:** Integration with NexCloud, NexHome, and other NexCorp products for seamless connectivity and synchronization.
4. **NexAI:** Built-in artificial intelligence capabilities for intelligent voice assistants, predictive analytics, and personalized recommendations.
5. **NexPerformance:** Optimized performance through intelligent resource management, cache optimization, and low-latency communication protocols.

Conclusion:

NexOS represents a significant leap forward in the world of mobile operating systems, setting new standards for performance, security, and innovation. With its advanced features, seamless integration, and unparalleled user experience, NexOS is poised to revolutionize the way people interact with their mobile devices, empowering them to do more, stay connected, and express their individuality in the digital age.