

OPERATING SYSTEM
CONTINUOUS ASSESSMENT - 2

NAME: **Harsh Pramod Padyal**

DIV: **D10B**

ROLL NO.: **43**

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

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

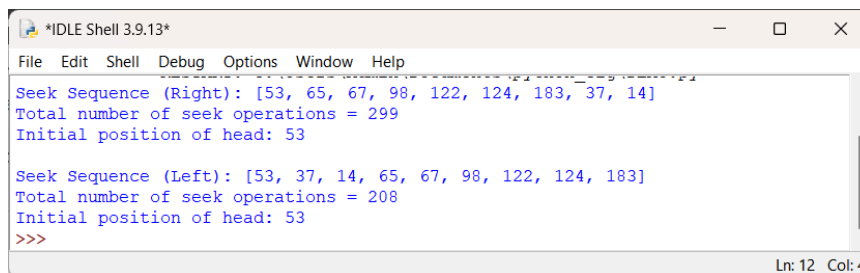
# Example
requests = [98, 183, 37, 122, 14, 124, 65, 67]
initial_head = 53
```

```
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

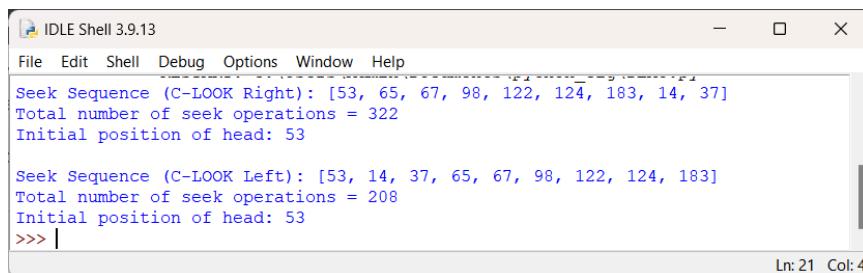
initial_direction = "right"

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)

```



The screenshot shows a Python IDLE Shell window with the following output:

```

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
>>>

```

The status bar at the bottom right indicates "Ln: 21 Col: 4".

Q.3) Case Study on Mobile Operating System.

KaiOS Mobile Operating System

Introduction:

KaiOS is a mobile operating system based on Linux, designed primarily for feature phones and smart feature phones (often referred to as "smart feature phones"). Launched in 2017, KaiOS aims to bridge the gap between traditional feature phones and smartphones by offering a simplified, affordable, and efficient mobile experience. This case study explores the history, market strategy, key features, app ecosystem, security, and challenges faced by the KaiOS operating system.

History:

1. Developed by KaiOS Technologies Inc., a U.S.-based company founded in 2016.
2. Launched in 2017, targeting emerging markets and consumers seeking affordable and accessible mobile connectivity.

Market Strategy:

1. Initially targeted at feature phones and smart feature phones, particularly in emerging markets like India, Africa, and Southeast Asia, where there is a significant demand for affordable mobile devices.
2. Partnership with mobile manufacturers, telecom operators, and content providers to integrate KaiOS into their devices, expand market reach, and offer tailored services and content to users.

Key Features:

1. **Simplicity and Efficiency:** Designed to offer a user-friendly interface, intuitive navigation, and efficient performance, catering to users who prioritize simplicity and basic mobile functionality.
2. **Affordability and Accessibility:** Optimized for low-cost hardware, ensuring affordability and accessibility for consumers in emerging markets and regions with limited access to smartphones and high-speed internet connectivity.
3. **Internet Connectivity and Apps:** Supports essential internet services, including web browsing, email, messaging, and social media apps, enabling users to stay connected, informed, and entertained.

App Ecosystem:

1. KaiStore serves as the official app store for KaiOS-powered devices, offering a selection of essential apps, games, and services tailored for feature phones and smart feature phones.
2. Collaboration with developers, content providers, and telecom operators to expand the app ecosystem, localize content, and enhance the user experience based on regional preferences and needs.

Security:

1. Incorporates basic security features and protocols to protect user data, ensure device integrity, and mitigate potential threats and vulnerabilities commonly associated with feature phones and emerging markets.

Challenges:

1. Market Education and Adoption: Educating consumers about the benefits of smart feature phones, overcoming misconceptions, and promoting KaiOS as a viable alternative to traditional feature phones and entry-level smartphones.
2. Content and App Localization: Tailoring content, apps, and services to meet the diverse needs, preferences, and languages of users in emerging markets and regions with distinct cultural, economic, and technological landscapes.
3. Competitive Landscape: Navigating the competitive mobile market, facing competition from established mobile operating systems, local brands, and entry-level smartphones offering similar features and functionalities at competitive price points.

Conclusion:

KaiOS, with its focus on affordability, simplicity, and connectivity, addresses a unique market segment and consumer demand for affordable mobile devices in emerging markets. While it may face challenges in terms of market education, content localization, and competitive positioning, KaiOS's potential to empower millions of users with basic internet connectivity, essential apps, and digital services positions it as a significant player in the global mobile ecosystem. As KaiOS Technologies continues to innovate, collaborate, and expand its footprint across new markets and regions, the growth, adoption, and impact of KaiOS on bridging the digital divide and enhancing mobile accessibility will be closely monitored and analyzed by industry stakeholders, policymakers, and consumers alike.