Name: Sanika Mehetre
Div: D10B
Roll no: 34

Q1. To write a c program to implement LRU page replacement algorithm.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_FRAMES 3

int pageFaults(int pages[], int n) {
  int faults = 0;
  int queue[MAX_FRAMES];
  int front = 0, rear = 0;
  int set[MAX_FRAMES] = {0};

  for (int i = 0; i < n; i++) {
    if (set[pages[i]] == 0) {
      faults++;
      if (rear - front == MAX_FRAMES) {
        set[queue[front]] = 0;
        front = (front + 1) % MAX_FRAMES;
      }
      queue[rear] = pages[i];
      set[pages[i]] = 1;
      rear = (rear + 1) % MAX_FRAMES;
    } else {
      int j;
      for (j = front; j < rear; j++) {
        if (queue[j] == pages[i])
          break;
      }
      for (; j < rear - 1; j++) {
        queue[j] = queue[j + 1];
      }
      queue[j] = pages[i];
    }
  }
  return faults;
}

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

```
    printf("Number of page faults: %d\n", pageFaults(pages, n));

    return 0;
}
```

```
/tmp/1LkRzYvEyU.o
Number of page faults: 5


=== Code Execution Successful ===
```

Q2. Implement various disk scheduling algorithms like LOOK,C-LOOK in C/Python/Java.

```
def look(arr, head, direction):
    seek_sequence = []
    size = len(arr)
    distance = 0

    while True:
        if direction == "left":
            for i in range(size):
                if arr[i] < head:
                    seek_sequence.append(arr[i])
            seek_sequence.sort(reverse=True)
        elif direction == "right":
            for i in range(size):
                if arr[i] > head:
                    seek_sequence.append(arr[i])
            seek_sequence.sort()

        if len(seek_sequence) == 0:
            break

        distance += abs(head - seek_sequence[0])
        head = seek_sequence[0]
        seek_sequence.pop(0)

    return distance

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

```
    size = len(arr)
    distance = 0

    while True:
        if direction == "left":
            for i in range(size):
                if arr[i] < head:
                    seek_sequence.append(arr[i])
            seek_sequence.sort()
        elif direction == "right":
            for i in range(size):
                if arr[i] > head:
                    seek_sequence.append(arr[i])
            seek_sequence.sort()

        if len(seek_sequence) == 0:
            break

        distance += abs(head - seek_sequence[0])
        head = seek_sequence[0]
        seek_sequence.pop(0)

    return distance

# Example usage
arr = [176, 79, 34, 60, 92, 11, 41, 114]
head = 50
direction = "left"

print("LOOK:", look(arr, head, direction))
print("C-LOOK:", clook(arr, head, direction))
```

```
LOOK: 39
C-LOOK: 191

=== Code Execution Successful ===
```

Q3. Case Study on Real-Time Operating System

A real-time operating system (RTOS) is designed to manage the resources of embedded systems and ensure that tasks with specific timing requirements are executed reliably and predictably. Let's consider a case study of an RTOS used in an automotive application.

Case Study: Real-Time Operating System in Automotive Systems

Background:

- Company: ABC Automotive Inc.
- Product: Smart Infotainment System for Cars
- RTOS Used: QNX Neutrino RTOS

Scenario:

ABC Automotive Inc. is developing a smart infotainment system for cars that includes features like navigation, multimedia playback, connectivity with mobile devices, and real-time monitoring of vehicle diagnostics. The infotainment system needs to be responsive, reliable, and able to handle multiple tasks simultaneously without compromising safety or performance.

Why QNX Neutrino RTOS?

- Real-Time Capabilities: QNX Neutrino RTOS is known for its real-time capabilities, providing deterministic behavior and predictable response times, which are critical for automotive applications.
- Reliability: The RTOS is highly reliable, with a microkernel architecture that isolates critical components, preventing system failures from affecting other parts of the system.
- Scalability: QNX Neutrino RTOS is scalable and can be tailored to meet the specific requirements of the infotainment system, ensuring optimal performance.
- Safety-Critical Features: The RTOS complies with safety standards such as ISO 26262, which is essential for automotive systems where safety is a top priority.

Key Features and Benefits:

Task Scheduling: The RTOS efficiently schedules tasks based on their priority and timing requirements, ensuring that critical tasks are executed on time.
Inter-Process Communication: QNX Neutrino RTOS provides mechanisms for efficient communication between processes, allowing different components of the infotainment system to exchange data seamlessly.
Resource Management: The RTOS manages system resources such as memory and CPU time effectively, preventing resource conflicts and ensuring optimal performance.
Fault Tolerance: QNX Neutrino RTOS includes features like process isolation and fault recovery mechanisms, which enhance the system's reliability and resilience to failures.

Implementation:

ABC Automotive Inc. integrates the QNX Neutrino RTOS into its smart infotainment system, leveraging its real-time capabilities to ensure that critical tasks like navigation and vehicle diagnostics are executed without delay. The RTOS also enables seamless connectivity with mobile devices and provides a responsive user interface for multimedia playback.

Conclusion:

By using QNX Neutrino RTOS in its smart infotainment system, ABC Automotive Inc. can deliver a reliable and high-performance solution that meets the stringent requirements of the automotive industry. The RTOS's real-time capabilities, reliability, and scalability make it an ideal choice for embedded systems where timing, responsiveness, and safety are paramount.