Operating System CA 2

Name:Aarya Gandhi
Class; D10B
Roll No. 66

Module 4 - Write a C program to implement LRU page replacement algorithm.

→ **Code -**

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_FRAMES 10

// Function to find the index of the least recently used page in the given array
int findLRUIndex(int pages[], int n, int frame[], int frameSize) {
   int res = -1, farthest = 0;
   for (int i = 0; i < frameSize; ++i) {
      int j;
      for (j = n - 1; j >= 0; --j) {
         if (frame[i] == pages[j]) {
            if (j > farthest) {
               farthest = j;
               res = i;
            }
            break;
         }
      }
      if (j == -1)
         return i;
   }
```

```
    return (res == -1) ? 0 : res;
}

// Function to implement LRU page replacement algorithm
void LRU(int pages[], int n, int frameSize) {
    int pageFaults = 0;
    int frame[frameSize];
    for (int i = 0; i < frameSize; ++i)
        frame[i] = -1;

    printf("Page Reference String: ");
    for (int i = 0; i < n; ++i) {
        printf("%d ", pages[i]);
        int flag = 0;
        for (int j = 0; j < frameSize; ++j) {
            if (frame[j] == pages[i]) {
                flag = 1;
                break;
            }
        }
        if (flag == 0) {
            int lruIndex = findLRUIndex(pages, n, frame, frameSize);
            frame[lruIndex] = pages[i];
            ++pageFaults;
        }
    }
    printf("\nNumber of Page Faults: %d\n", pageFaults);
}

int main() {
    int pages[MAX_FRAMES];
    int n, frameSize;

    printf("Enter the number of pages: ");
```

```
    scanf("%d", &n);

    printf("Enter the page reference string: ");
    for (int i = 0; i < n; ++i) {
        scanf("%d", &pages[i]);
    }

    printf("Enter the number of frames: ");
    scanf("%d", &frameSize);

    LRU(pages, n, frameSize);

    return 0;
}
```

Output -

```
Enter the number of pages: 6
Enter the page reference string: 1 2 3 4 7 8
Enter the number of frames: 5
Page Reference String: 1 2 3 4 7 8
Number of Page Faults: 6


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

Module 5 - Implement various disk scheduling algorithms like LOOK, C-LOOK in C/Python/Java.

→ Code -

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

// Function to implement LOOK disk scheduling algorithm
void look(int requests[], int head, int size) {
    int totalSeekTime = 0;
```

```
    int cur_track = head;

    printf("Seek Sequence: ");
    for (int i = 0; i < size; ++i) {
        printf("%d ", requests[i]);
        totalSeekTime += abs(cur_track - requests[i]);
        cur_track = requests[i];
    }
    printf("\nTotal Seek Time: %d\n", totalSeekTime);
}

// Function to implement C-LOOK disk scheduling algorithm
void c_look(int requests[], int head, int size) {
    int totalSeekTime = 0;
    int cur_track = head;

    printf("Seek Sequence: ");
    for (int i = 0; i < size; ++i) {
        printf("%d ", requests[i]);
        totalSeekTime += abs(cur_track - requests[i]);
        cur_track = requests[i];
    }
    printf("\nTotal Seek Time: %d\n", totalSeekTime);
}

int main() {
    int *requests, head, size;

    printf("Enter the number of disk requests: ");
    scanf("%d", &size);

    requests = (int *)malloc(size * sizeof(int));

    if (requests == NULL) {
```

```
        printf("Memory allocation failed.\n");
        return 1;
    }

    printf("Enter the disk requests: ");
    for (int i = 0; i < size; i++) {
        scanf("%d", &requests[i]);
    }

    printf("Enter initial position of head: ");
    scanf("%d", &head);

    // Look Algorithm
    look(requests, head, size);

    // C-Look Algorithm
    c_look(requests, head, size);

    free(requests);

    return 0;
}
```

Output -

```
Enter the number of disk requests: 4
Enter the disk requests: 2 3 4 5
Enter initial position of head: 4
Seek Sequence: 2 3 4 5
Total Seek Time: 5
Seek Sequence: 2 3 4 5
Total Seek Time: 5


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

Module 6 - Case Study on Comparison between functions of various Special-purpose Operating Systems.

→

Introduction:
Special-purpose operating systems play a crucial role in catering to the diverse needs of modern computing environments by offering tailored functionality and optimizations for specific tasks or domains. Unlike general-purpose operating systems like Windows, Linux, or macOS, which aim to provide broad compatibility and functionality for a wide range of applications, special-purpose operating systems are designed to excel in particular areas, such as real-time processing, embedded systems, or networking. These specialized operating systems are crafted to meet the unique requirements of their intended applications, often prioritizing factors like determinism, resource efficiency, or network performance over generic features. Understanding the functions and characteristics of various special-purpose operating systems is essential for developers, engineers, and decision-makers tasked with selecting the most suitable platform for their specific use cases.
In this comparative analysis, we explore three prominent categories of special-purpose operating systems: real-time operating systems (RTOS), embedded operating systems, and network operating systems (NOS

Real-Time Operating Systems (RTOS):

Definition: RTOS is designed to manage tasks with strict timing requirements. It guarantees a certain level of performance within a specified time frame, making it ideal for time-critical applications.

Key Features:

- **Determinism**: RTOS ensures that tasks are executed within predictable time constraints, providing deterministic behavior crucial for real-time applications.
- **Task Scheduling:** Utilizes scheduling algorithms like priority-based scheduling or rate monotonic scheduling to prioritize tasks based on their urgency..
- **Memory Management:** Provides optimized memory allocation strategies to minimize memory fragmentation and ensure predictable behavior.

  Use Cases:

- **Industrial Automation**: Control systems in manufacturing processes where precise timing is essential for

synchronization and coordination.

- **Automotive Systems:** Powertrain control, ABS (anti-lock braking system), and airbag deployment systems require instantaneous response to ensure passenger safety.
- **Aerospace and Defense:** Flight control systems, missile guidance systems, and avionics applications demand real-time responsiveness for navigation and mission-critical operations.

Embedded Operating Systems (EOS):
Definition: EOS is tailored for resource-constrained embedded devices, focusing on minimizing memory footprint, energy consumption, and hardware dependencies while providing essential operating system services.

Key Features:
- **Footprint Optimization:** Strives for minimal code size and memory usage to run efficiently on embedded hardware with limited resources.
- **Hardware Abstraction:** Provides abstraction layers to facilitate hardware interaction, enabling developers to write device-independent code.
- **Peripheral Support:** Offers drivers and APIs for interfacing with peripheral devices like sensors, actuators, and communication interfaces.
- **Energy Efficiency:** Implements power management techniques to optimize energy consumption and extend battery life in portable or battery-operated devices

.
Use Cases:
- **Consumer Electronics:** Smartphones, IoT devices, wearables, and home automation systems rely on embedded OS for seamless integration with hardware and efficient resource utilization.
- **Medical Devices:** Implantable medical devices, patient monitoring systems, and medical imaging equipment utilize embedded OS for reliable operation in healthcare settings.
- **Automotive Infotainment:** In-car entertainment systems, navigation systems, and telematics systems leverage embedded OS for multimedia playback, communication, and vehicle diagnostics.

Network Operating Systems (NOS):
Definition: NOS is specialized for managing and coordinating network resources, facilitating communication between devices, and ensuring efficient data transmission across networked environments.

Key Features:
- **Network Protocol Support:** Provides protocols like TCP/IP, UDP, ICMP, and others for communication between networked devices over Ethernet, Wi-Fi, or other networking technologies.
- **Resource Sharing:** Enables sharing of files, printers, and other resources across the network, allowing multiple users to access and utilize shared resources efficiently.
- **Scalability:** Scales to accommodate growing network infrastructures and increasing numbers of connected devices while maintaining performance and reliability.

Use Cases:

- **Server Environments:** File servers, print servers, email servers, and database servers rely on NOS for managing network resources and facilitating communication between clients and servers.
- **Networking Devices:** Routers, switches, firewalls, and network appliances use NOS to control network traffic, implement routing protocols, and ensure seamless connectivity within network infrastructures.
- **Network Attached Storage (NAS):** NAS systems provide centralized storage accessible over the network, leveraging NOS for efficient data storage, retrieval, and sharing among users and devices.

Aspect	**Real-Time Operating Systems (RTOS)**	**Embedded Operating Systems (EOS)**	**Network Operating Systems (NOS)**
Focus	Timely execution of tasks with strict timing	Resource optimization for embedded devices	Management of network resources and communication
Timing Constraints	Guarantees deterministic behavior	Emphasizes minimal footprint and energy efficiency	Facilitates efficient data transmission and networking
Use Cases	Industrial automation, automotive systems, aerospace applications	Consumer electronics, medical devices, automotive infotainment systems	Server environments, networking devices, NAS systems
Features	Deterministic task scheduling Efficient interrupt handling Optimized memory management	Hardware abstraction Peripheral support Energy efficiency	Network protocol support Resource sharing mechanisms- Security features Scalability capabilities

Conclusion:

Each special-purpose operating system brings unique strengths and capabilities to the table, making it suitable for specific application domains. The choice of operating system depends on factors such as real-time requirements, scalability, security, and ecosystem compatibility. By carefully evaluating the features and characteristics of each operating system, organizations can make informed decisions to meet their application requirements
effectively.