OS CA-2

Name : Teena Waishy
Div: D10B
Roll no: 63

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

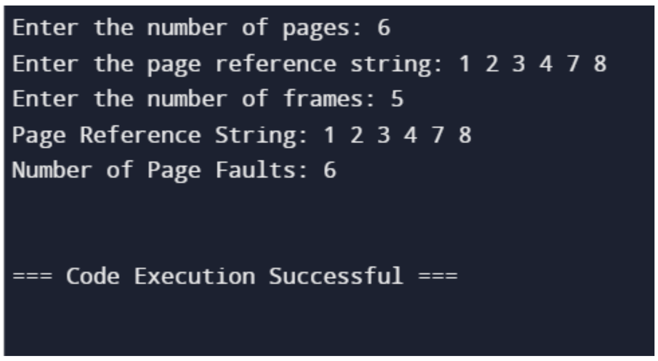
➔ **Code -**

```
#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 -



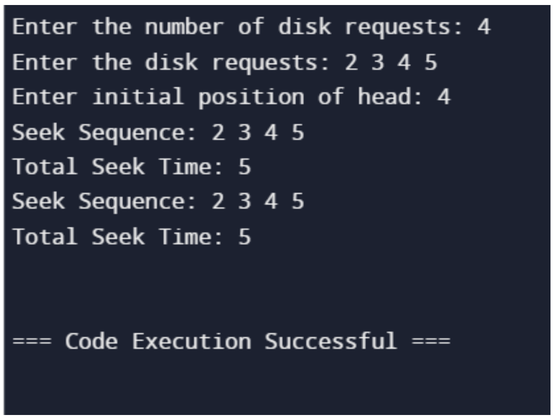
```
Enter the number of pages: 6
Enter the page reference string: 1 2 3 4 7 8
Enter the number of frames: 5
Page Reference String: 1 2 3 4 7 8
Number of Page Faults: 6


=== 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.

When comparing the functions of various special-purpose operating systems, it's important to consider the specific tasks or environments they are designed for. Let's take a look at three examples: real-time operating systems (RTOS), embedded operating systems, and network operating systems (NOS).

1. Real-Time Operating Systems (RTOS):
   - Functions:
     - Deterministic response times: RTOSs are designed to provide predictable and deterministic response times for critical tasks.
     - Task scheduling: RTOSs often use priority-based or time-sliced scheduling algorithms to ensure that critical tasks are executed on time.
     - Interrupt handling: RTOSs have efficient mechanisms for handling interrupts to minimize latency.
     - Resource management: RTOSs manage resources such as memory and peripherals to ensure efficient use by tasks.
   - Examples: FreeRTOS, VxWorks, QNX

2. Embedded Operating Systems:
   - Functions:
     - Low resource footprint: Embedded OSs are designed to operate on devices with limited resources (e.g., memory, processing power).
     - Hardware abstraction: Embedded OSs provide abstraction layers to simplify access to hardware components.
     - Real-time support: Some embedded OSs offer real-time capabilities similar to RTOSs.
     - Power management: Embedded OSs often include power management features to optimize energy consumption.
   - Examples: Embedded Linux, Windows Embedded, FreeRTOS

3. Network Operating Systems (NOS):
   - Functions:
     - Network resource management: NOSs manage network resources such as servers, routers, and switches.
     - Security: NOSs provide security features such as authentication, access control, and encryption.
     - Network services: NOSs offer services like file sharing, printing, and email over a network.
     - Performance monitoring: NOSs include tools for monitoring network performance and diagnosing issues.
   - Examples: Windows Server, Linux-based server distributions (e.g., Ubuntu Server, CentOS), Cisco IOS

In conclusion, while all three types of operating systems serve specific purposes, they have distinct functions tailored to their respective environments. RTOSs focus on deterministic response times, embedded OSs prioritize resource efficiency and hardware abstraction, and NOSs specialize in managing network resources and services. Understanding these differences is crucial when selecting an operating system for a particular application or system.

Q3. Case Study on Comparison between functions of various Special-purpose Operating Systems.

1. Introduction

Operating systems designed for specialized purposes are essential for satisfying the unique demands of different sectors and uses. Special-purpose operating systems are made to excel in certain fields, such real-time systems, embedded devices, networking equipment, and mobile devices, in contrast to general-purpose operating systems, like Windows or Linux, which can handle a wide range of activities. The goal of this case study is to present a thorough analysis of the characteristics, advantages, and disadvantages of several special-purpose operating systems.

2. Selection of Operating Systems

For this comparative analysis, we have selected a diverse set of special-purpose operating systems:

- Real-time operating systems (RTOS): VxWorks and QNX.
- Embedded operating systems: FreeRTOS and ThreadX.
- Network operating systems: Cisco IOS and Juniper Junos.
- Mobile operating systems: iOS and Android.

3. Comparison of Functions

3.1 Kernel Design

A modular microkernel architecture is used by VxWorks to deliver predictable real-time performance appropriate for mission-critical applications. Similar microkernel architecture is used by QNX, which prioritizes scalability and stability in embedded systems.

3.2 Task Scheduling

A priority-based preemptive scheduling method is used by VxWorks to guarantee that important activities are completed on time. Effective use of system resources is provided by QNX's dynamic priority-based scheduler with adaptive algorithms for resource management.

3.3 Memory Management

Virtual memory management is supported by both VxWorks and QNX, allowing for effective memory allocation and protection. Whereas QNX uses a nanokernel architecture for lightweight memory management, VxWorks uses a demand-paged memory management method.

3.4 File System

Optimized for embedded applications, VxWorks offers a configurable file system that supports a range of file types and storage devices. For mission-critical situations, QNX provides a scalable and dependable file system with journaling features that guarantees data integrity.

3.5 Networking

With a focus on routing and switching in business networks, Cisco IOS provides an extensive set of networking protocols and functionality. With its powerful routing and security features, Juniper Junos prioritizes scalability and dependability in high-performance network environments.

3.6 Device Drivers

Device drivers are heavily supported by VxWorks and QNX, which makes it easier to integrate with a variety of hardware accessories. Standardized APIs for driver creation and compatibility are provided by both operating systems.

3.7 Security Features

To defend network infrastructure against cyber attacks, Cisco IOS has strong security features like encryption, intrusion prevention systems, and access control lists (ACLs). To protect network assets, Juniper Junos deploys cutting-edge security measures like application-level security and unified threat management (UTM).

3.8 Development Tools

Debuggers, simulators, and profiling tools are just a few of the extensive development tools that VxWorks and QNX provide to make software development and testing easier. Industry-standard programming languages like C and C++ are supported by both operating systems, and integrated development environments (IDEs) facilitate the quick development of applications.

4. Case Studies

Real-time applications, such as aircraft systems and industrial automation, frequently depend on VxWorks due to its dependability and deterministic performance. QNX is commonly utilized in medical devices and car entertainment systems, where security and safety are top priorities.

5. Conclusion

To sum up, every special-purpose operating system has distinct characteristics and capabilities designed for particular application areas. In real-time and embedded systems, VxWorks and QNX are industry leaders, whereas in networking, Cisco IOS and Juniper Junos are market leaders. Leading the mobile operating system market with their intuitive interfaces and strong security are iOS and Android. Organizations choosing the best platform for their applications can make well-informed judgments by being aware of the advantages and disadvantages of each operating system.