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

Solution ->

Code:

```
#include <iostream>
#include<bits/stdc++.h>
using namespace std;
int main()
{
int capacity = 4;
int arr[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};
deque<int> q(capacity);
int count=0;
int page_faults=0;
deque<int>::iterator itr;
q.clear();
for(int i:arr)
{
   // Insert it into set if not present
   // already which represents page fault
   itr = find(q.begin(),q.end(),i);
   if(!(itr != q.end()))
   {
   ++page_faults;
   // Check if the set can hold equal pages
   if(q.size() == capacity)
   {
      q.erase(q.begin());
      q.push_back(i);
   }
   else{
      q.push_back(i);
   }  }
   else
   {
   // Remove the indexes page
   q.erase(itr);
   // insert the current page
   q.push_back(i);
   } }
cout<<page_faults;
}
```

Output:

6

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

LOOK =>

Code:

```
int size = 8;
#include <bits/stdc++.h>
using namespace std;
int disk_size = 200;
void LOOK(int arr[], int head, string direction)
{
    int seek_count = 0;
    int distance, cur_track;
    vector<int> left, right;
    vector<int> seek_sequence;
    // appending values which are
    // currently at left and right
    // direction from the head.
    for (int i = 0; i < size; i++) {
        if (arr[i] < head)
            left.push_back(arr[i]);
        if (arr[i] > head)
            right.push_back(arr[i]);
    }
    // sorting left and right vectors
    // for servicing tracks in the
    // correct sequence.
    std::sort(left.begin(), left.end());
    std::sort(right.begin(), right.end());
    // run the while loop two times.
    // one by one scanning right
    // and left side of the head
    int run = 2;
    while (run--) {
        if (direction == "left") {
            for (int i = left.size() - 1; i >= 0; i--) {
                cur_track = left[i];
                // appending current track to seek sequence
                seek_sequence.push_back(cur_track);
                // calculate absolute distance
                distance = abs(cur_track - head);
                // increase the total count
                seek_count += distance;
                // accessed track is now the new head
                head = cur_track;
            }
```

```
            // reversing the direction
            direction = "right";
        }
        else if (direction == "right") {
            for (int i = 0; i < right.size(); i++) {
                cur_track = right[i];
                // appending current track to seek sequence
                seek_sequence.push_back(cur_track);
                // calculate absolute distance
                distance = abs(cur_track - head);
                // increase the total count
                seek_count += distance;
                // accessed track is now new head
                head = cur_track;
            }
            // reversing the direction
            direction = "left";
        }
    }
    cout << "Total number of seek operations = "
        << seek_count << endl;
    cout << "Seek Sequence is" << endl;
    for (int i = 0; i < seek_sequence.size(); i++) {
        cout << seek_sequence[i] << endl;
    }
}
// Driver code
int main()
{
    // request array
    int arr[size] = { 176, 79, 34, 60,
                    92, 11, 41, 114 };
    int head = 50;
    string direction = "right";
    cout << "Initial position of head: "
        << head << endl;
    LOOK(arr, head, direction);
    return 0;
}
```

Output:

Initial position of head: 50
Total number of seek operations = 291
Seek Sequence: 60, 79, 92, 114, 176, 41, 34, 11

C-LOOK =>

Code:

```
#include <bits/stdc++.h>
using namespace std;
int size = 8;
int disk_size = 200;

// Function to perform C-LOOK on the request
// array starting from the given head
void CLOOK(int arr[], int head)
{
   int seek_count = 0;
   int distance, cur_track;
   vector<int> left, right;
   vector<int> seek_sequence;

   // Tracks on the left of the
   // head will be serviced when
   // once the head comes back
   // to the beginning (left end)
   for (int i = 0; i < size; i++) {
      if (arr[i] < head)
         left.push_back(arr[i]);
      if (arr[i] > head)
         right.push_back(arr[i]);
   }

   // Sorting left and right vectors
   std::sort(left.begin(), left.end());
   std::sort(right.begin(), right.end());

   // First service the requests
   // on the right side of the
   // head
   for (int i = 0; i < right.size(); i++) {
      cur_track = right[i];
      // Appending current track to seek sequence
      seek_sequence.push_back(cur_track);
      // Calculate absolute distance
      distance = abs(cur_track - head);
      // Increase the total count
      seek_count += distance;
      // Accessed track is now new head
      head = cur_track;
   }
```

```
    // Once reached the right end
    // jump to the last track that
    // is needed to be serviced in
    // left direction
    seek_count += abs(head - left[0]);
    head = left[0];
    // Now service the requests again
    // which are left
    for (int i = 0; i < left.size(); i++) {
        cur_track = left[i];
        // Appending current track to seek sequence
        seek_sequence.push_back(cur_track);
        // Calculate absolute distance
        distance = abs(cur_track - head);
        // Increase the total count
        seek_count += distance;
        // Accessed track is now the new head
        head = cur_track;
    }
    cout << "Total number of seek operations = "
        << seek_count << endl;
    cout << "Seek Sequence is" << endl;
    for (int i = 0; i < seek_sequence.size(); i++) {
        cout << seek_sequence[i] << endl;
    }
}
// Driver code
int main()
{
    // Request array
    int arr[size] = { 176, 79, 34, 60, 92, 11, 41, 114 };
    int head = 50;
    cout << "Initial position of head: " << head << endl;
    CLOOK(arr, head);
    return 0;
}
```

Output:

Initial Position of Head: 50
Total Number of Seek Operations: 321
Seek Sequence: 60, 79, 92, 114, 176, 11, 34, 41

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.