

Operating Systems Continuous Assessment – II

Name: Swayam Raut

Class: D10B

Roll No. 50

Q1) To write C program to implement LFU page replacement

```
#include <stdio.h>

#include <stdlib.h>

#define MAX_FRAMES 3 // Maximum number of frames

// Structure representing a page
typedef struct {
    int pageNumber;
    int frequency;
} Page;

// Function to find the index of the page with minimum frequency
int findLFUIndex(Page pages[], int n) {
    int minFreqIndex = 0;
    int minFreq = pages[0].frequency;
    for (int i = 1; i < n; i++) {
        if (pages[i].frequency < minFreq) {
            minFreq = pages[i].frequency;
            minFreqIndex = i;
        }
    }
    return minFreqIndex;
}
```

```

// Function to simulate LFU page replacement algorithm

void LFU(int pages[], int n, int frames) {
    Page frame[MAX_FRAMES];
    int pageFaults = 0;

    // Initialize frames
    for (int i = 0; i < frames; i++) {
        frame[i].pageNumber = -1; // Set page number to -1 indicating an empty frame
        frame[i].frequency = 0;
    }

    for (int i = 0; i < n; i++) {
        int pageFound = 0;

        // Check if the page is already in a frame
        for (int j = 0; j < frames; j++) {
            if (frame[j].pageNumber == pages[i]) {
                frame[j].frequency++;
                pageFound = 1;
                break;
            }
        }

        // If page is not found in any frame
        if (!pageFound) {
            int index = findLFUIndex(frame, frames);
            frame[index].pageNumber = pages[i];
            frame[index].frequency = 1;
            pageFaults++;
        }
    }
}

```

```

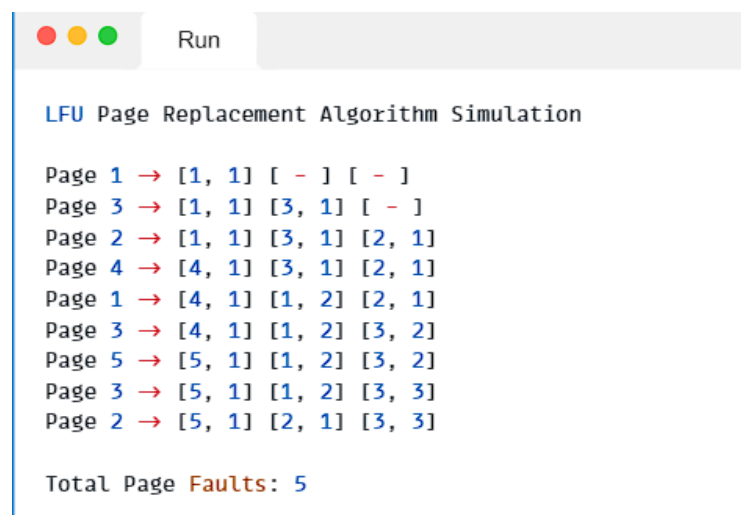
// Print current state of frames
printf("Page %d -> ", pages[i]);
for (int j = 0; j < frames; j++) {
    if (frame[j].pageNumber != -1)
        printf("[%d, %d] ", frame[j].pageNumber, frame[j].frequency);
    else
        printf("[ - ] ");
}
printf("\n");
}

printf("\nTotal Page Faults: %d\n", pageFaults);
}

int main() {
    int pages[] = {1, 3, 2, 4, 1, 3, 5, 3, 2};
    int n = sizeof(pages) / sizeof(pages[0]);
    int frames = 3;

    printf("LFU Page Replacement Algorithm Simulation\n\n");
    LFU(pages, n, frames);
    return 0;
}

```



```

LFU Page Replacement Algorithm Simulation

Page 1 → [1, 1] [ - ] [ - ]
Page 3 → [1, 1] [3, 1] [ - ]
Page 2 → [1, 1] [3, 1] [2, 1]
Page 4 → [4, 1] [3, 1] [2, 1]
Page 1 → [4, 1] [1, 2] [2, 1]
Page 3 → [4, 1] [1, 2] [3, 2]
Page 5 → [5, 1] [1, 2] [3, 2]
Page 3 → [5, 1] [1, 2] [3, 3]
Page 2 → [5, 1] [2, 1] [3, 3]

Total Page Faults: 5

```

Q2) LOOK ALGORITHM

```
def look(arr, head, direction):
    seek_sequence = []
    seek_count = 0

    if direction == "left":
        left = [request for request in arr if request < head]
        right = [request for request in arr if request >= head]
        left.reverse() # Reverse to start from outermost request
        left.append(0) # Add endpoint
        left.extend(right) # Concatenate with right side
        sequence = left
    else:
        right = [request for request in arr if request > head]
        left = [request for request in arr if request <= head]
        right.append(max(arr)) # Add endpoint
        right.extend(left) # Concatenate with left side
        sequence = right

    for i in range(len(sequence) - 1):
        seek_count += abs(sequence[i] - sequence[i+1])
        seek_sequence.append(sequence[i])

    seek_sequence.append(sequence[-1]) # Add the last request

    return seek_sequence, seek_count

# Example usage:
requests = [98, 183, 37, 122, 14, 124, 65, 67]
initial_head = 53
direction = "left"

sequence, count = look(requests, initial_head, direction)
print("Seek Sequence:", sequence)
print("Total Seek Count:", count)
```

```
For LOOK Algorithm:
Seek Sequence: [53, 37, 14, 13, 65, 67, 98, 122, 124, 183]
Total Seek Count: 235
```

LOOK Disk Scheduling Algorithm:

The LOOK algorithm is a disk scheduling algorithm that scans through the disk in a particular direction, stopping at the last request in that direction, then changes direction and scans in the opposite direction. It does not go all the way to the edge of the disk.

Explanation:

If the head is moving towards the left (lower cylinder numbers), it first services the requests located on the left side of the initial head position in decreasing order until the first cylinder. Then, it reverses direction and moves towards the right, servicing the requests in increasing order until the last request.

If the head is moving towards the right (higher cylinder numbers), it first services the requests located on the right side of the initial head position in increasing order until the last cylinder. Then, it reverses direction and moves towards the left, servicing the requests in decreasing order until the first request.

2) C LOOK Disk Scheduling Algorithm

```
def c_look(arr, head, direction):
    seek_sequence = []
    seek_count = 0

    if direction == "left":
        left = [request for request in arr if request < head]
        right = [request for request in arr if request >= head]
        left.sort()
        right.sort()
        left.append(max(arr)) # Add endpoint
        left.extend(right) # Concatenate with right side
        sequence = left
    else:
        right = [request for request in arr if request > head]
        left = [request for request in arr if request <= head]
        right.sort()
        left.sort()
        right.append(0) # Add endpoint
        right.extend(left) # Concatenate with left side
        sequence = right

    for i in range(len(sequence) - 1):
        seek_count += abs(sequence[i] - sequence[i+1])
        seek_sequence.append(sequence[i])

    seek_sequence.append(sequence[-1]) # Add the last request

    return seek_sequence, seek_count

# Example usage:
requests = [98, 183, 37, 122, 14, 124, 65, 67]
initial_head = 53
direction = "left"

sequence, count = c_look(requests, initial_head, direction)
print("Seek Sequence:", sequence)
print("Total Seek Count:", count)
```



For C-LOOK Algorithm:

Seek Sequence: [53, 65, 67, 98, 122, 124, 183, 14, 13, 37]

Total Seek Count: 196

C-LOOK Disk Scheduling Algorithm:

The C-LOOK algorithm is an improvement over LOOK that mitigates the inefficiency of LOOK by always scanning in the same direction (either towards higher cylinder numbers or lower ones) but only services requests along the way, not going all the way to the end of the disk.

Explanation:

C-LOOK starts by scanning towards either higher or lower cylinder numbers, servicing requests along the way until the last request in that direction.

Instead of going all the way to the end of the disk and then reversing direction like in LOOK, C-LOOK jumps back to the beginning of the disk (the first request in the same direction) and continues servicing requests along the way until it reaches the last request.

Case Study: Evolution of Mobile Operating Systems

Introduction

Mobile operating systems (OS) have become an integral part of everyday life, powering smartphones, tablets, wearables, and various other smart devices. This case study explores the evolution of mobile operating systems, from their inception to the present day, focusing on key players in the market, their features, and their impact on society.

1. Early Mobile Operating Systems

In the early days of mobile devices, operating systems were basic and primarily designed for making calls and sending text messages. One of the first widely adopted mobile OS was Symbian, developed by Symbian Ltd. It dominated the market in the early 2000s, powering Nokia phones and other mobile devices. Symbian provided basic functionalities such as contacts, calendar, and messaging.

Another notable player was Palm OS, developed by Palm, Inc. It gained popularity in the late 1990s and early 2000s, particularly with devices like PalmPilot and Palm Treo. Palm OS featured a stylus-based interface and supported third-party applications through its Palm OS Software Installation Tool (Palm OS Installer).

2. Rise of Smartphone Operating Systems

The advent of smartphones revolutionized the mobile industry, leading to the development of more advanced mobile operating systems. One of the most significant milestones was the introduction of Apple's iOS in 2007 with the launch of the first iPhone. iOS introduced a touch-based interface, multi-touch gestures, and a rich ecosystem of mobile apps through the App Store. It quickly gained popularity for its intuitive user experience and seamless integration with other Apple products.

Around the same time, Google introduced Android, an open-source mobile operating system. Initially developed by Android, Inc., which was later acquired by Google, Android offered versatility and customization, allowing device manufacturers to modify the OS according to their preferences. Android's open nature and extensive app ecosystem contributed to its widespread adoption across various device manufacturers.

3. Dominance of iOS and Android

In the following years, iOS and Android emerged as the dominant players in the mobile OS market, with a duopoly that continues to this day. Both platforms underwent significant advancements, introducing new features and improvements with each iteration.

iOS continued to evolve with regular updates, introducing features such as Siri (voice assistant), iCloud (cloud storage service), and Apple Pay (mobile payment system). iOS devices, including iPhones, iPads, and iPod Touch, maintained a loyal user base, known for their seamless integration with other Apple services and robust security measures.

Meanwhile, Android expanded its market share globally, powering a wide range of devices from budget smartphones to flagship models. Google introduced several major updates to Android, each named after a dessert (e.g., Cupcake, Donut, KitKat, etc.). Android's flexibility, extensive customization options, and support for a wide variety of hardware made it a favorite among both users and device manufacturers.

4. Emergence of New Players

While iOS and Android dominate the mobile OS market, several other players have emerged over the years, each targeting specific niches or regions. For example:

- Microsoft developed Windows Phone, a mobile OS aimed at providing a unified experience across smartphones and desktops. However, despite some innovative features like Live Tiles and Cortana (virtual assistant), Windows Phone struggled to gain significant traction and eventually ceased development.
- BlackBerry OS, developed by BlackBerry Limited, was known for its strong focus on security and productivity. However, with the rise of iOS and Android, BlackBerry's market share declined, leading the company to transition to Android-based devices.
- Ubuntu Touch, developed by Canonical Ltd., aimed to bring the Ubuntu operating system to smartphones and tablets. While it received praise for its convergence features (ability to run desktop applications), Ubuntu Touch failed to gain widespread adoption.

5. Current Landscape and Future Trends

As of the present day, iOS and Android continue to dominate the mobile OS market, with iOS maintaining a strong presence in developed markets like North America and Europe, while Android enjoys broader global adoption.

Looking ahead, several trends are shaping the future of mobile operating systems:

- 5G and IoT Integration: With the rollout of 5G networks and the proliferation of Internet of Things (IoT) devices, mobile operating systems are expected to prioritize connectivity and interoperability across a wide range of devices.
- Artificial Intelligence (AI) Integration: AI-driven features such as voice assistants, predictive analytics, and machine learning algorithms are becoming increasingly prevalent in mobile operating systems, enhancing user experiences and personalization.
- Privacy and Security: With growing concerns about data privacy and cybersecurity, mobile operating systems are focusing on strengthening security measures, implementing robust encryption, and providing users with more control over their personal data.
- Foldable and Dual-Screen Devices: The emergence of foldable smartphones and dual-screen devices presents new challenges and opportunities for mobile operating systems, requiring support for flexible displays and innovative user interfaces.

In conclusion, mobile operating systems have come a long way since their inception, evolving from basic platforms for making calls to sophisticated ecosystems powering a wide range of smart devices. While iOS and Android dominate the market, the landscape continues to evolve, driven by technological advancements, changing consumer preferences, and emerging trends in connectivity and device form factors.