Continuous Assessment - 3

Name: Sushant Tulasi
Class: D10B
Roll No. : 62

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

```
#include <stdio.h>
#define MAX_FRAMES 3
#define MAX_PAGES 10
int findLRU(int frames[], int n, int pages[], int m, int index) {
    int res = -1, farthest = index;
    for (int i = 0; i < n; ++i) {
        int j;
        for (j = index; j < m; ++j) {
            if (frames[i] == pages[j]) {
                if (j > farthest) {
                    farthest = j;
                    res = i;
                }
                break;
            }
        }
        if (j == m)
            return i;
    }
    return (res == -1) ? 0 : res;
}
void lruPageReplace(int pages[], int m) {
    int frames[MAX_FRAMES], frameCount = 0;
    for (int i = 0; i < MAX_FRAMES; ++i)
        frames[i] = -1;
    int pageFaults = 0;
    for (int i = 0; i < m; ++i) {
        if (frameCount < MAX_FRAMES) {
            int j;
            for (j = 0; j < frameCount; ++j)
                if (frames[j] == pages[i])
                    break;
            if (j == frameCount) {
                frames[frameCount++] = pages[i];
                ++pageFaults;
            }
        } else {
            int j = findLRU(frames, MAX_FRAMES, pages, m, i + 1);
```

```
            if (frames[j] != pages[i]) {
                frames[j] = pages[i];
                ++pageFaults;
            }
        }
        printf("Pages in frames after page %d: ", i + 1);
        for (int k = 0; k < frameCount; ++k)
            printf("%d ", frames[k]);
        printf("\n");
    }
    printf("Total Page Faults: %d\n", pageFaults);
}
int main() {
    int pages[MAX_PAGES], n;
    printf("Enter number of pages: ");
    scanf("%d", &n);
    printf("Enter the page sequence: ");
    for (int i = 0; i < n; ++i)
        scanf("%d", &pages[i]);
    lruPageReplace(pages, n);
    return 0;
}
```

OUTPUT:

```
/tmp/OiMyQR1L5T.o
Enter number of pages: 5
Enter the page sequence: 2
3
1
2
4
Pages in frames after page 1: 2
Pages in frames after page 2: 2 3
Pages in frames after page 3: 2 3 1
Pages in frames after page 4: 2 3 1
Pages in frames after page 5: 4 3 1
Total Page Faults: 4
```

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

```
#include <stdio.h>
#include <stdlib.h>
void look(int arr[], int head, int size) {
    int seek_count = 0;
    int distance = 0;
    int cur_track;
```

```
int left = 0, right = 0;
int diff = 0;
int max_track = 200;
int min_track = 0;
for (int i = 0; i < size; i++) {
    cur_track = arr[i];
if (cur_track >= head)
        right++;
    else
        left++;
}
// Sort the requests array
for (int i = 0; i < size - 1; i++) {
    for (int j = i + 1; j < size; j++) {
        if (arr[i] > arr[j]) {
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
}
int move_left = left < right;
if (move_left) {
    for (int i = size - 1; i >= 0; i--) {
        if (arr[i] <= head) {
            printf("Seek Sequence: %d\n", arr[i]);
            seek_count += abs(head - arr[i]);
            head = arr[i];
        }
    }
    head = min_track;
    for (int i = 0; i < size; i++) {
        if (arr[i] >= head) {
            printf("Seek Sequence: %d\n", arr[i]);
            seek_count += abs(head - arr[i]);
            head = arr[i];
        }
    }
} else {
    for (int i = 0; i < size; i++) {
        if (arr[i] >= head) {
            printf("Seek Sequence: %d\n", arr[i]);
            seek_count += abs(head - arr[i]);
            head = arr[i];
        }
    }
    head = max_track;
    for (int i = size - 1; i >= 0; i--) {
        if (arr[i] <= head) {
```

```
                printf("Seek Sequence: %d\n", arr[i]);
                seek_count += abs(head - arr[i]);
                head = arr[i];
            }
        }
    }
printf("Total Seek Count = %d\n", seek_count);}
void clook(int arr[], int head, int size) {
    int seek_count = 0;
    int distance = 0;
    int cur_track;
    int left = 0, right = 0;
    int diff = 0;
    int max_track = 200;
    int min_track = 0;
    for (int i = 0; i < size; i++) {
        cur_track = arr[i];
        if (cur_track >= head)
            right++;
        else
            left++;
    }
    for (int i = 0; i < size - 1; i++) {
        for (int j = i + 1; j < size; j++) {
            if (arr[i] > arr[j]) {
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
    int move_left = left < right;
    if (move_left) {
        for (int i = size - 1; i >= 0; i--) {
            if (arr[i] <= head) {
                printf("Seek Sequence: %d\n", arr[i]);
                seek_count += abs(head - arr[i]);
                head = arr[i];
            }
        }
        head = max_track;
        for (int i = size - 1; i >= 0; i--) {
            if (arr[i] <= head) {
                printf("Seek Sequence: %d\n", arr[i]);
                seek_count += abs(head - arr[i]);
                head = arr[i];
            }
        }
    } else {
```

```
        for (int i = 0; i < size; i++) {
            if (arr[i] >= head) {
                printf("Seek Sequence: %d\n", arr[i]);
                seek_count += abs(head - arr[i]);
                head = arr[i];
            }
        }
    head = min_track;
        for (int i = 0; i < size; i++) {
            if (arr[i] >= head) {
                printf("Seek Sequence: %d\n", arr[i]);
                seek_count += abs(head - arr[i]);
                head = arr[i];
            }
        }
    }
    printf("Total Seek Count = %d\n", seek_count);
}
int main() {
    int n, head, choice;
    printf("Enter the size of the queue: ");
    scanf("%d", &n);
    int *arr = (int *)malloc(n * sizeof(int));
    printf("Enter the requests: ");
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);
    printf("Enter the initial head position: ");
    scanf("%d", &head);
    printf("Enter the choice of disk scheduling algorithm (1 for LOOK, 2 for C-LOOK): ");
    scanf("%d", &choice);
    switch (choice) {
        case 1:
            look(arr, head, n);
            break;
        case 2:
            clook(arr, head, n);
            break;
        default:
            printf("Invalid choice!\n");
            break;
    }
    free(arr);
    return 0;
}
```

OUTPUT:

```
/tmp/CeqljIHTUH.o
Enter the size of the queue: 8
Enter the requests: 98
183
37
122
14
124
65
67
Enter the initial head position: 53
Enter the choice of disk scheduling algorithm (1 for LOOK, 2 for C-LOOK): 1
Seek Sequence: 37
Seek Sequence: 14
Seek Sequence: 14
Seek Sequence: 37
Seek Sequence: 65
Seek Sequence: 67
Seek Sequence: 98
Seek Sequence: 122
Seek Sequence: 124
Seek Sequence: 183
Total Seek Count = 222
```

```
/tmp/T7GcnWa1KN.o
Enter the size of the queue: 8
Enter the requests: 98
183
37
122
14
124
65
67
Enter the initial head position: 53
Enter the choice of disk scheduling algorithm (1 for LOOK, 2 for C-LOOK): 2
Seek Sequence: 37
Seek Sequence: 14
Seek Sequence: 183
Seek Sequence: 124
Seek Sequence: 122
Seek Sequence: 98
Seek Sequence: 67
Seek Sequence: 65
Seek Sequence: 37
Seek Sequence: 14
Total Seek Count = 225
```

Q3. Case Study on Comparison between functions of various Special-purpose Operating Systems.
Ans:
Comparing the functions of various special-purpose operating systems can provide valuable insights into their design, capabilities, and suitability for specific tasks. Let's consider three special-purpose operating systems: QNX, FreeRTOS, and Android Things, and compare their functions based on their intended use cases.

1. **QNX:**
   - **Real-time Capabilities:** QNX is known for its real-time capabilities, making it suitable for mission-critical applications where timing and reliability are essential, such as automotive systems, industrial automation, and medical devices.
   - **Microkernel Architecture:** QNX employs a microkernel architecture, which provides modularity, scalability, and robustness. It allows for efficient resource management and isolation of system components.
   - **Interprocess Communication (IPC):** QNX offers advanced IPC mechanisms, including message passing and shared memory, facilitating communication between processes while ensuring determinism and low latency.
   - **High Availability:** QNX supports fault tolerance and high availability features, allowing systems to recover from failures quickly and continue operating without disruption.

2. **FreeRTOS:**
   - **Embedded Systems Support:** FreeRTOS is designed for embedded systems with limited resources, such as microcontrollers and IoT devices. It provides a lightweight, real-time kernel tailored for low-power, constrained environments.
   - **Task Management:** FreeRTOS offers task scheduling and management capabilities, allowing developers to create and prioritize tasks based on their requirements. It supports preemptive and cooperative multitasking.
   - **Peripheral Support:** FreeRTOS includes drivers and libraries for various peripherals commonly found in embedded systems, such as UART, SPI, I2C, and GPIO, facilitating hardware interaction and device control.
   - **Portability:** FreeRTOS is highly portable and can run on a wide range of microcontroller architectures. It provides a consistent API across different platforms, simplifying development and deployment.

3. **Android Things:**
   - **IoT Platform:** Android Things is a special-purpose operating system focused on IoT development. It provides a framework for building connected devices, integrating with Google services, and leveraging the Android ecosystem.
   - **Application Framework:** Android Things includes a rich application framework based on Android, enabling developers to create IoT applications using familiar tools and APIs. It supports features such as user interfaces, multimedia, and networking.
   - **Security:** Android Things emphasizes security, with built-in mechanisms for secure boot, device authentication, and data encryption. It aims to protect IoT devices from threats such as malware, unauthorized access, and data breaches.

- **Cloud Integration:** Android Things integrates with cloud platforms such as Google Cloud IoT Core, enabling seamless communication between devices and cloud services. It supports features like device management, data analytics, and over-the-air updates.

Conclusion:

In summary, QNX excels in real-time performance and reliability for critical systems, FreeRTOS is tailored for resource-constrained embedded environments, and Android Things provides a comprehensive IoT development platform with cloud integration and security features. The choice of operating system depends on the specific requirements and constraints of the target application.