**WEEK 8**

**Y. Shamil Ahamed**

**1BM21CS248.**

**1)FCFS,SCAN,C-SCAN:**

**INPUT:**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_REQUESTS 100

void fcfs(int requests[], int n, int start) {

    int totalSeek = 0, current = start;

    printf("FCFS Disk Scheduling:\n");

    for (int i = 0; i < n; i++) {

        totalSeek += abs(current - requests[i]);

        printf("Move from %d to %d\n", current, requests[i]);

        current = requests[i];

    }

    printf("Total Seek Distance: %d\n", totalSeek);

}

void scan(int requests[], int n, int start, int maxCylinder) {

    int totalSeek = 0, current = start;

    printf("SCAN Disk Scheduling:\n");

    int direction = 1; // 1 for right, -1 for left

    int maxIndex = (direction == 1) ? maxCylinder : 0;

    for (int i = 0; i < n; i++) {

        totalSeek += abs(current - requests[i]);

        printf("Move from %d to %d\n", current, requests[i]);

        current = requests[i];

    }

    totalSeek += abs(current - maxIndex);

    printf("Move from %d to %d\n", current, maxIndex);

    for (int i = n - 1; i >= 0; i--) {

        totalSeek += abs(maxIndex - requests[i]);

        printf("Move from %d to %d\n", maxIndex, requests[i]);

        maxIndex = requests[i];

    }

    printf("Total Seek Distance: %d\n", totalSeek);

}

void cScan(int requests[], int n, int start, int maxCylinder) {

    int totalSeek = 0, current = start;

    printf("C-SCAN Disk Scheduling:\n");

    int maxIndex = maxCylinder;

    for (int i = 0; i < n; i++) {

        totalSeek += abs(current - requests[i]);

        printf("Move from %d to %d\n", current, requests[i]);

        current = requests[i];

    }

    totalSeek += abs(current - maxIndex);

    printf("Move from %d to %d\n", current, maxIndex);

    current = 0;

    for (int i = 0; i < n; i++) {

        totalSeek += abs(current - requests[i]);

        printf("Move from %d to %d\n", current, requests[i]);

        current = requests[i];

    }

    printf("Total Seek Distance: %d\n", totalSeek);

}

int main() {

    int requests[MAX\_REQUESTS], n, start, maxCylinder;

    printf("Enter the number of requests: ");

    scanf("%d", &n);

    if (n > MAX\_REQUESTS) {

        printf("Maximum number of requests exceeded.\n");

        return 1;

    }

    printf("Enter the requests: ");

    for (int i = 0; i < n; i++)

        scanf("%d", &requests[i]);

    printf("Enter the starting position: ");

    scanf("%d", &start);

    printf("Enter the maximum cylinder value: ");

    scanf("%d", &maxCylinder);

    fcfs(requests, n, start);

    scan(requests, n, start, maxCylinder);

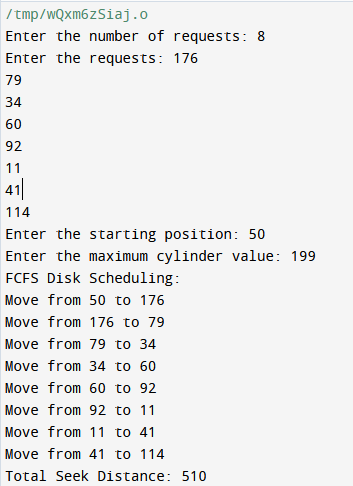
    cScan(requests, n, start, maxCylinder);

    return 0;

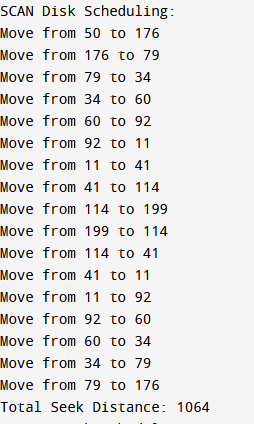
}

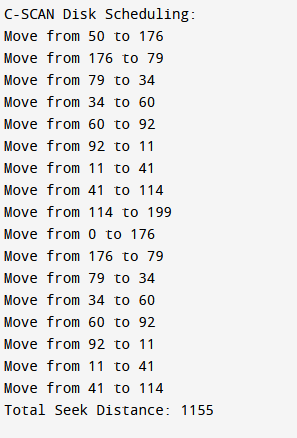
**OUTPUT:**

**FCFS:**

****

**SCAN:**

****

**C-SCAN:**

2)SSTF,LOOK,C-LOOK:

INPUT:

#include <stdio.h>

#include <stdlib.h>

#define MAX\_REQUESTS 100

void sort(int arr[], int n) {

    for (int i = 0; i < n - 1; i++)

        for (int j = 0; j < n - i - 1; j++)

            if (arr[j] > arr[j + 1])

                arr[j] ^= arr[j + 1] ^= arr[j] ^= arr[j + 1];

}

int absDiff(int a, int b) {

    return (a > b) ? a - b : b - a;

}

void sstf(int requests[], int n, int start) {

    int totalSeek = 0, current = start, visited[MAX\_REQUESTS] = {0};

    printf("SSTF Disk Scheduling:\n");

    for (int i = 0; i < n; i++) {

        int minDist = \_\_INT\_MAX\_\_, nextIndex = -1;

        for (int j = 0; j < n; j++) {

            if (!visited[j]) {

                int distance = absDiff(current, requests[j]);

                if (distance < minDist) {

                    minDist = distance;

                    nextIndex = j;

                }

            }

        }

        visited[nextIndex] = 1;

        totalSeek += minDist;

        printf("Move from %d to %d\n", current, requests[nextIndex]);

        current = requests[nextIndex];

    }

    printf("Total Seek Distance: %d\n", totalSeek);

}

void look(int requests[], int n, int start, int direction) {

    int totalSeek = 0, current = start;

    sort(requests, n);

    printf("LOOK Disk Scheduling:\n");

    int i = (direction == 1) ? 0 : n - 1;

    int end = (direction == 1) ? n : -1;

    int step = (direction == 1) ? 1 : -1;

    while (i != end) {

        if (direction == 1 && requests[i] < current)

            break;

        if (direction == 0 && requests[i] > current)

            break;

        totalSeek += absDiff(current, requests[i]);

        printf("Move from %d to %d\n", current, requests[i]);

        current = requests[i];

        i += step;

    }

    printf("Total Seek Distance: %d\n", totalSeek);

}

void cLook(int requests[], int n, int start) {

    int totalSeek = 0, current = start;

    sort(requests, n);

    printf("C-LOOK Disk Scheduling:\n");

    int index = 0;

    while (index < n && requests[index] <= current)

        index++;

    for (int i = index; i < n; i++) {

        totalSeek += absDiff(current, requests[i]);

        printf("Move from %d to %d\n", current, requests[i]);

        current = requests[i];

    }

    for (int i = 0; i < index; i++) {

        totalSeek += absDiff(current, requests[i]);

        printf("Move from %d to %d\n", current, requests[i]);

        current = requests[i];

    }

    printf("Total Seek Distance: %d\n", totalSeek);

}

int main() {

    int requests[MAX\_REQUESTS], n, start, direction, choice;

    printf("Enter the number of requests: ");

    scanf("%d", &n);

    if (n > MAX\_REQUESTS) {

        printf("Maximum number of requests exceeded.\n");

        return 1;

    }

    printf("Enter the requests: ");

    for (int i = 0; i < n; i++)

        scanf("%d", &requests[i]);

    printf("Enter the starting position: ");

    scanf("%d", &start);

    printf("Enter the direction (1 for upward, 0 for downward): ");

    scanf("%d", &direction);

    printf("Choose Disk Scheduling Algorithm:\n");

    printf("1. SSTF\n2. LOOK\n3. C-LOOK\n");

    scanf("%d", &choice);

    switch (choice) {

        case 1:

            sstf(requests, n, start);

            break;

        case 2:

            look(requests, n, start, direction);

            break;

        case 3:

            cLook(requests, n, start);

            break;

        default:

            printf("Invalid choice!\n");

            break;

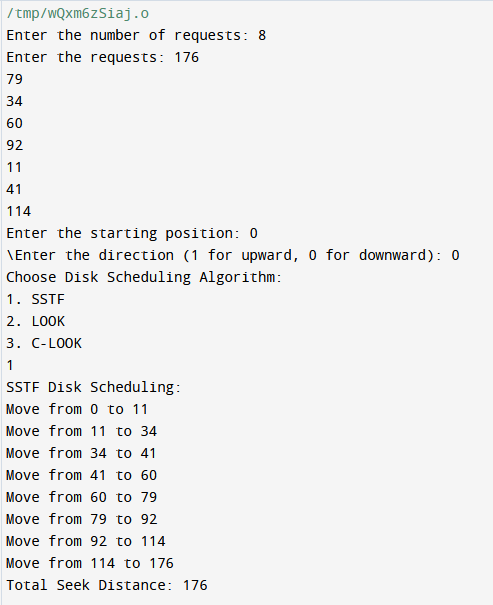
    }

    return 0;

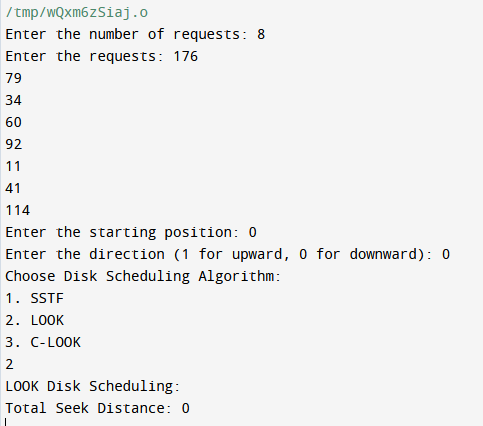
}

OUTPUT:

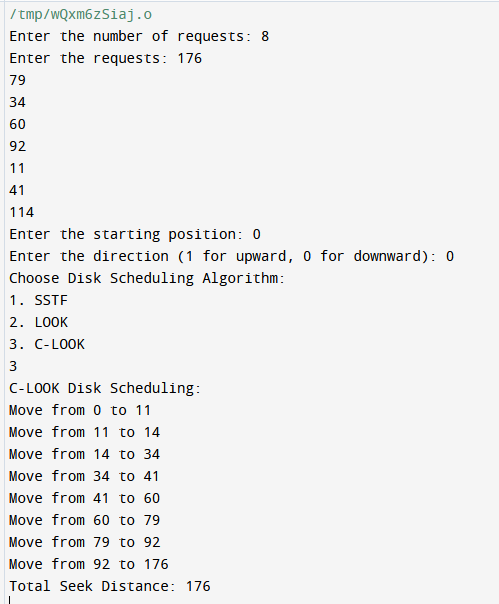
SSTF:



LOOK:



C-LOOK:



3)FIFO,LRU,Optimal:

INPUT:

#include <stdio.h>

#include <stdbool.h>

#define NUM\_FRAMES 3

#define NUM\_PAGES 10

void printFrames(int frames[], int n) {

    for (int i = 0; i < n; i++) {

        if (frames[i] == -1)

            printf(" - ");

        else

            printf(" %d ", frames[i]);

    }

    printf("\n");

}

int findIndex(int arr[], int n, int element) {

    for (int i = 0; i < n; i++) {

        if (arr[i] == element)

            return i;

    }

    return -1;

}

int findOptimal(int pages[], int n, int frames[], int start) {

    int res = -1, farthest = start;

    for (int i = 0; i < NUM\_FRAMES; i++) {

        int j;

        for (j = start; j < n; j++) {

            if (frames[i] == pages[j]) {

                if (j > farthest) {

                    farthest = j;

                    res = i;

                }

                break;

            }

        }

        if (j == n)

            return i;

    }

    return (res == -1) ? 0 : res;

}

void fifo(int pages[], int n) {

    int frames[NUM\_FRAMES];

    int frameIndex = 0;

    int pageFaults = 0;

    for (int i = 0; i < NUM\_FRAMES; i++)

        frames[i] = -1;

    printf("FIFO Page Replacement:\n");

    for (int i = 0; i < n; i++) {

        int page = pages[i];

        if (findIndex(frames, NUM\_FRAMES, page) == -1) {

            frames[frameIndex] = page;

            frameIndex = (frameIndex + 1) % NUM\_FRAMES;

            pageFaults++;

        }

        printf("Page %d -> ", page);

        printFrames(frames, NUM\_FRAMES);

    }

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

}

void lru(int pages[], int n) {

    int frames[NUM\_FRAMES];

    int pageFaults = 0;

    for (int i = 0; i < NUM\_FRAMES; i++)

        frames[i] = -1;

    printf("LRU Page Replacement:\n");

    for (int i = 0; i < n; i++) {

        int page = pages[i];

        if (findIndex(frames, NUM\_FRAMES, page) == -1) {

            int emptyIndex = findIndex(frames, NUM\_FRAMES, -1);

            if (emptyIndex != -1) {

                frames[emptyIndex] = page;

            } else {

                int lruIndex = i;

                for (int j = 0; j < NUM\_FRAMES; j++) {

                    if (findIndex(pages, n, frames[j]) < lruIndex) {

                        lruIndex = findIndex(pages, n, frames[j]);

                    }

                }

                frames[lruIndex] = page;

            }

            pageFaults++;

        }

        printf("Page %d -> ", page);

        printFrames(frames, NUM\_FRAMES);

    }

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

}

void optimal(int pages[], int n) {

    int frames[NUM\_FRAMES];

    int pageFaults = 0;

    for (int i = 0; i < NUM\_FRAMES; i++)

        frames[i] = -1;

    printf("Optimal Page Replacement:\n");

    for (int i = 0; i < n; i++) {

        int page = pages[i];

        if (findIndex(frames, NUM\_FRAMES, page) == -1) {

            int emptyIndex = findIndex(frames, NUM\_FRAMES, -1);

            if (emptyIndex != -1) {

                frames[emptyIndex] = page;

            } else {

                int optimalIndex = findOptimal(pages, n, frames, i + 1);

                frames[optimalIndex] = page;

            }

            pageFaults++;

        }

        printf("Page %d -> ", page);

        printFrames(frames, NUM\_FRAMES);

    }

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

}

int main() {

    int pages[NUM\_PAGES] = {2, 3, 2, 1, 5, 2, 4, 5, 3, 2};

    int choice;

    printf("Choose Page Replacement Algorithm:\n");

    printf("1. FIFO\n2. LRU\n3. Optimal\n");

    scanf("%d", &choice);

    switch (choice) {

        case 1:

            fifo(pages, NUM\_PAGES);

            break;

        case 2:

            lru(pages, NUM\_PAGES);

            break;

        case 3:

            optimal(pages, NUM\_PAGES);

            break;

        default:

            printf("Invalid choice!\n");

            break;

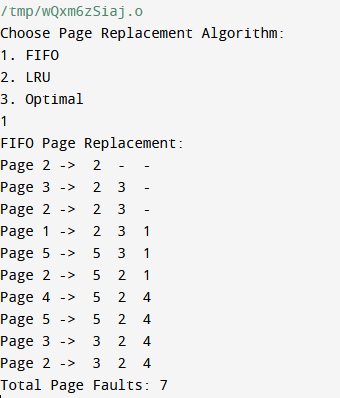
    }

    return 0;

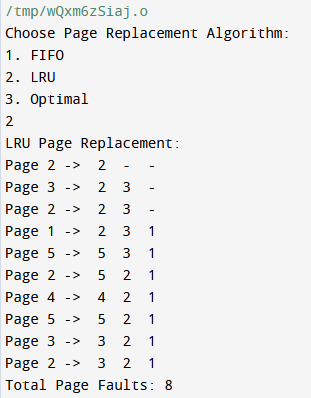
}

OUTPUT:

FIFO:



LRU:



Optimal:

