1.Multithreaading:

main.c

Share Run

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void* print_message(void* thread_id) {
    long tid = (long)thread_id;
    printf("Hello from thread %ld\n", tid);
    pthread_exit(NULL);
}

int main() {
    pthread_t threads[2];
    int rc;
    long t;

    for (t = 0; t < 2; t++) {
        printf("Creating thread %ld\n", t);
        rc = pthread_create(&threads[t], NULL, print_message, (void
            *)t);
        if (rc) {
            printf("Error:unable to create thread, %d\n", rc);
            exit(-1);
        }
    }

    for (t = 0; t < 2; t++) {
```

Output Clear

```
Creating thread 0
Creating thread 1
Hello from thread 0
Hello from thread 1
Main thread exiting.


=== Code Execution Successful ===
```

```c
    }

    for (t = 0; t < 2; t++) {
        pthread_join(threads[t], NULL);
    }

    printf("Main thread exiting.\n");
    return 0;
}
```

2. FIFO Paging:

main.c

```c
#include <stdio.h>

#define MAX 100

void fifoPageReplacement(int pages[], int n, int capacity) {
    int frames[capacity];
    int front = 0, rear = 0, count = 0;
    int pageFaults = 0;
    int i, j, found;

    for (i = 0; i < n; i++) {
        found = 0;

        // Check if page is already in a frame
        for (j = 0; j < count; j++) {
            if (frames[j] == pages[i]) {
                found = 1;
                break;
            }
        }

        if (!found) {
            // If frames not full, add page at rear
            if (count < capacity) {
                frames[rear] = pages[i];
                rear = (rear + 1) % capacity;
                count++;
            } else {
                // Replace the oldest page (FIFO)
                frames[front] = pages[i];
                front = (front + 1) % capacity;
                rear = (rear + 1) % capacity;
            }
            pageFaults++;
        }

        // Display current frame state
        printf("Step %d: ", i + 1);
        for (j = 0; j < count; j++) {
            printf("%d ", frames[(front + j) % capacity]);
        }
        if (found)
            printf(" (No Page Fault)");
        else
            printf(" (Page Fault)");
        printf("\n");
    }

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

int main() {
    int pages[MAX], n, capacity, i;

    printf("Enter number of pages: ");
    scanf("%d", &n);

    printf("Enter the page reference string:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &pages[i]);
    }

    printf("Enter number of frames: ");
    scanf("%d", &capacity);

    fifoPageReplacement(pages, n, capacity);

    return 0;
}
```

Output

```
Enter number of pages: 12
Enter the page reference string:
1 3 0 3 5 6 3 0 1 2 1 2
Enter number of frames: 3
Step 1: 1  (Page Fault)
Step 2: 1 3  (Page Fault)
Step 3: 1 3 0  (Page Fault)
Step 4: 1 3 0  (No Page Fault)
Step 5: 3 0 5  (Page Fault)
Step 6: 0 5 6  (Page Fault)
Step 7: 5 6 3  (Page Fault)
Step 8: 6 3 0  (Page Fault)
Step 9: 3 0 1  (Page Fault)
Step 10: 0 1 2  (Page Fault)
Step 11: 0 1 2  (No Page Fault)
Step 12: 0 1 2  (No Page Fault)

Total Page Faults: 9


=== Code Execution Successful ===
```

3. LRU Paging:

main.c

```c
#include <stdio.h>

#define MAX 100

// Function to find the least recently used page
int findLRU(int time[], int n) {
    int i, minimum = time[0], pos = 0;
    for (i = 1; i < n; i++) {
        if (time[i] < minimum) {
            minimum = time[i];
            pos = i;
        }
    }
    return pos;
}

void lruPageReplacement(int pages[], int n, int capacity) {
    int frames[capacity], time[capacity];
    int pageFaults = 0, counter = 0;
    int i, j, pos, flag1, flag2;

    for (i = 0; i < capacity; i++) {
        frames[i] = -1;
    }

    for (i = 0; i < n; i++) {
```

Output

```
Enter number of pages: 12
Enter the page reference string:
1 3 0 3 5 6 3 0 1 2 1 2
Enter number of frames: 3
Step 1: 1  (Page Fault)
Step 2: 1 3  (Page Fault)
Step 3: 1 3 0  (Page Fault)
Step 4: 1 3 0  (No Page Fault)
Step 5: 5 3 0  (Page Fault)
Step 6: 5 3 6  (Page Fault)
Step 7: 5 3 6  (No Page Fault)
Step 8: 0 3 6  (Page Fault)
Step 9: 0 3 1  (Page Fault)
Step 10: 0 2 1  (Page Fault)
Step 11: 0 2 1  (No Page Fault)
Step 12: 0 2 1  (No Page Fault)

Total Page Faults: 8


=== Code Execution Successful ===
```

```c
    for (i = 0; i < n; i++) {
        flag1 = flag2 = 0;

        // Check if page is already in frame
        for (j = 0; j < capacity; j++) {
            if (frames[j] == pages[i]) {
                counter++;
                time[j] = counter;
                flag1 = flag2 = 1;
                break;
            }
        }

        // If page not found, insert it
        if (flag1 == 0) {
            for (j = 0; j < capacity; j++) {
                if (frames[j] == -1) {
                    counter++;
                    pageFaults++;
                    frames[j] = pages[i];
                    time[j] = counter;
                    flag2 = 1;
                    break;
                }
            }
```


```
Enter number of pages: 12
Enter the page reference string:
1 3 0 3 5 6 3 0 1 2 1 2
Enter number of frames: 3
Step 1: 1  (Page Fault)
Step 2: 1 3  (Page Fault)
Step 3: 1 3 0  (Page Fault)
Step 4: 1 3 0  (No Page Fault)
Step 5: 5 3 0  (Page Fault)
Step 6: 5 3 6  (Page Fault)
Step 7: 5 3 6  (No Page Fault)
Step 8: 0 3 6  (Page Fault)
Step 9: 0 3 1  (Page Fault)
Step 10: 0 2 1  (Page Fault)
Step 11: 0 2 1  (No Page Fault)
Step 12: 0 2 1  (No Page Fault)

Total Page Faults: 8


=== Code Execution Successful ===
```


```c
            printf(" (Page Fault)");
        printf("\n");
    }

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

int main() {
    int pages[MAX], n, capacity, i;

    printf("Enter number of pages: ");
    scanf("%d", &n);

    printf("Enter the page reference string:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &pages[i]);
    }

    printf("Enter number of frames: ");
    scanf("%d", &capacity);

    lruPageReplacement(pages, n, capacity);

    return 0;
}
```


```
Enter number of pages: 12
Enter the page reference string:
1 3 0 3 5 6 3 0 1 2 1 2
Enter number of frames: 3
Step 1: 1  (Page Fault)
Step 2: 1 3  (Page Fault)
Step 3: 1 3 0  (Page Fault)
Step 4: 1 3 0  (No Page Fault)
Step 5: 5 3 0  (Page Fault)
Step 6: 5 3 6  (Page Fault)
Step 7: 5 3 6  (No Page Fault)
Step 8: 0 3 6  (Page Fault)
Step 9: 0 3 1  (Page Fault)
Step 10: 0 2 1  (Page Fault)
Step 11: 0 2 1  (No Page Fault)
Step 12: 0 2 1  (No Page Fault)

Total Page Faults: 8


=== Code Execution Successful ===
```

4. Optimal Paging:

main.c

```c
#include <stdio.h>
#define MAX 100
int predict(int pages[], int frames[], int n, int index, int
    capacity) {
    int res = -1, farthest = index;
    for (int i = 0; i < capacity; i++) {
        int j;
        for (j = index; 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 optimalPageReplacement(int pages[], int n, int capacity) {
    int frames[capacity];
    int count = 0, pageFaults = 0;
```

Output

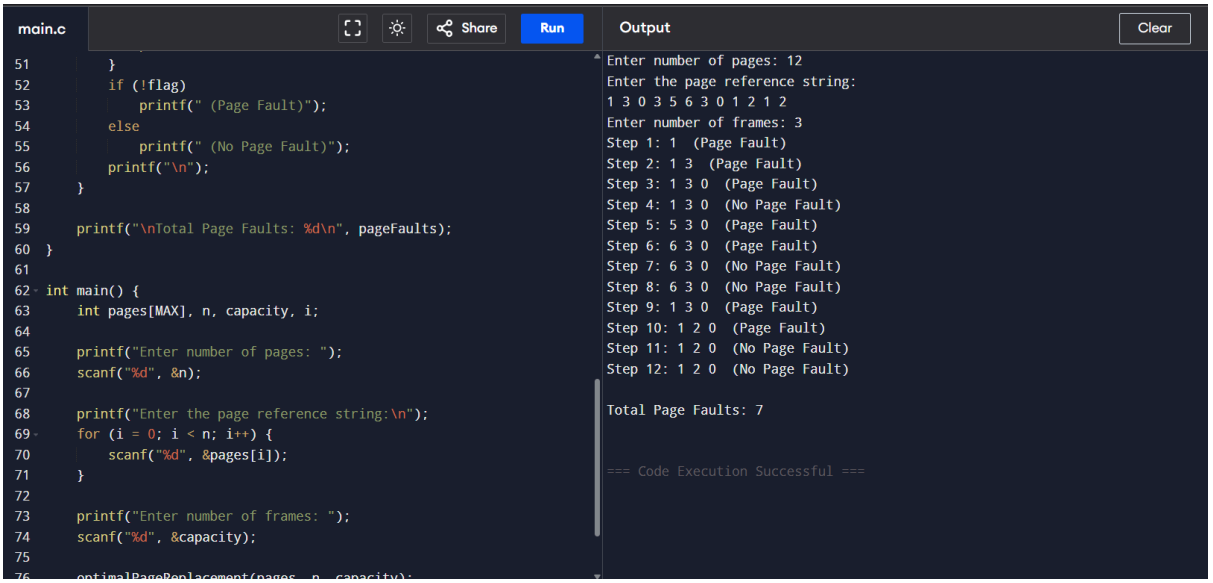
```
Enter number of pages: 12
Enter the page reference string:
1 3 0 3 5 6 3 0 1 2 1 2
Enter number of frames: 3
Step 1: 1  (Page Fault)
Step 2: 1 3  (Page Fault)
Step 3: 1 3 0  (Page Fault)
Step 4: 1 3 0  (No Page Fault)
Step 5: 5 3 0  (Page Fault)
Step 6: 6 3 0  (Page Fault)
Step 7: 6 3 0  (No Page Fault)
Step 8: 6 3 0  (No Page Fault)
Step 9: 1 3 0  (Page Fault)
Step 10: 1 2 0  (Page Fault)
Step 11: 1 2 0  (No Page Fault)
Step 12: 1 2 0  (No Page Fault)

Total Page Faults: 7


=== Code Execution Successful ===
```

main.c

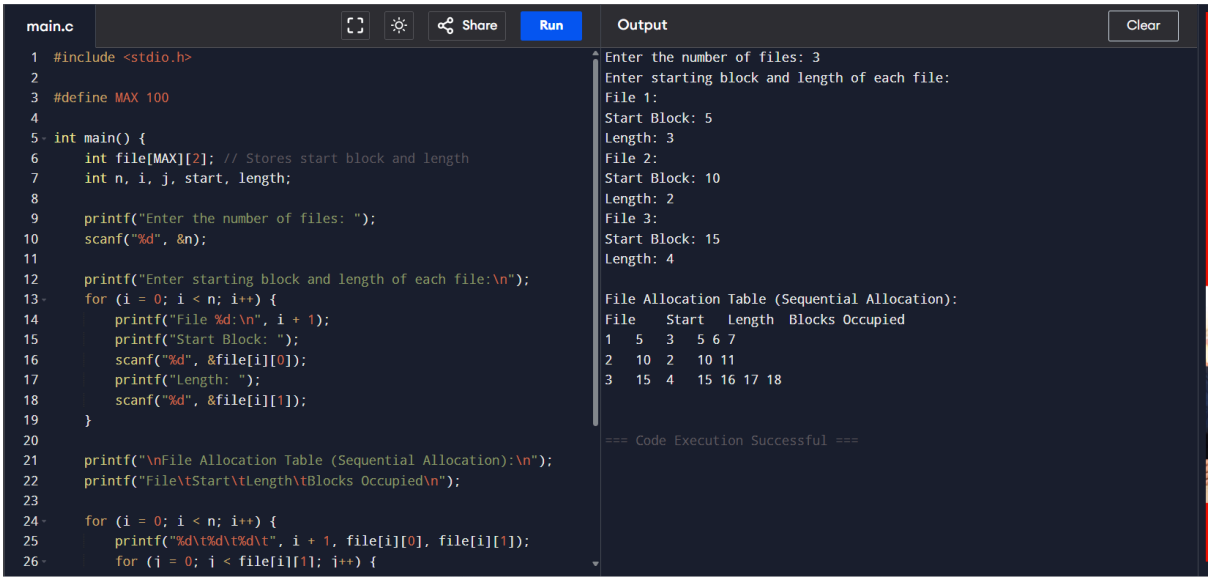
```c
    int count = 0, pageFaults = 0;
    int i, j, k, flag;
    for (i = 0; i < capacity; i++) {
        frames[i] = -1;
    }
    for (i = 0; i < n; i++) {
        flag = 0;
        for (j = 0; j < count; j++) {
            if (frames[j] == pages[i]) {
                flag = 1;
                break;
            }
        }

        if (!flag) {
            if (count < capacity) {
                frames[count++] = pages[i];
            } else {
                int pos = predict(pages, frames, n, i + 1, capacity
                    );
                frames[pos] = pages[i];
            }
            pageFaults++;
        }
        printf("Step %d: ", i + 1);
```

Output

```
Enter number of pages: 12
Enter the page reference string:
1 3 0 3 5 6 3 0 1 2 1 2
Enter number of frames: 3
Step 1: 1  (Page Fault)
Step 2: 1 3  (Page Fault)
Step 3: 1 3 0  (Page Fault)
Step 4: 1 3 0  (No Page Fault)
Step 5: 5 3 0  (Page Fault)
Step 6: 6 3 0  (Page Fault)
Step 7: 6 3 0  (No Page Fault)
Step 8: 6 3 0  (No Page Fault)
Step 9: 1 3 0  (Page Fault)
Step 10: 1 2 0  (Page Fault)
Step 11: 1 2 0  (No Page Fault)
Step 12: 1 2 0  (No Page Fault)

Total Page Faults: 7


=== Code Execution Successful ===
```



5. Sequential File Allocation:




```
            for (j = 0; j < file[i][1]; j++) {
                printf("%d ", file[i][0] + j);
            }
            printf("\n");
        }

        return 0;
    }
```