192311035

M.RAVI SAI VINAY

EXPERIMENT-33

Construct a C program to simulate the optimal paging technique of memory management

AIM:-

To simulate the Optimal Page Replacement algorithm and show how memory management works using this technique.

ALGORITHM:-

1. **Input**: Take the reference string (sequence of page requests) and the number of frames (available memory slots).
2. **Page Fault**: A page fault occurs when a requested page is not in memory.
3. **Optimal Replacement**: When memory is full and a page fault occurs, replace the page that is not used for the longest period of time in the future.
4. **Display**: Print the page frames and the number of page faults.

PROCEDURE:-

1. Initialize the page frame as empty.
2. For each page request in the reference string:
    - If the page is not in memory, cause a page fault and insert the page into memory.
    - If memory is full, replace the page that will not be used for the longest period of time.
3. Display the page frames after each page request and count the number of page faults.

CODE:-

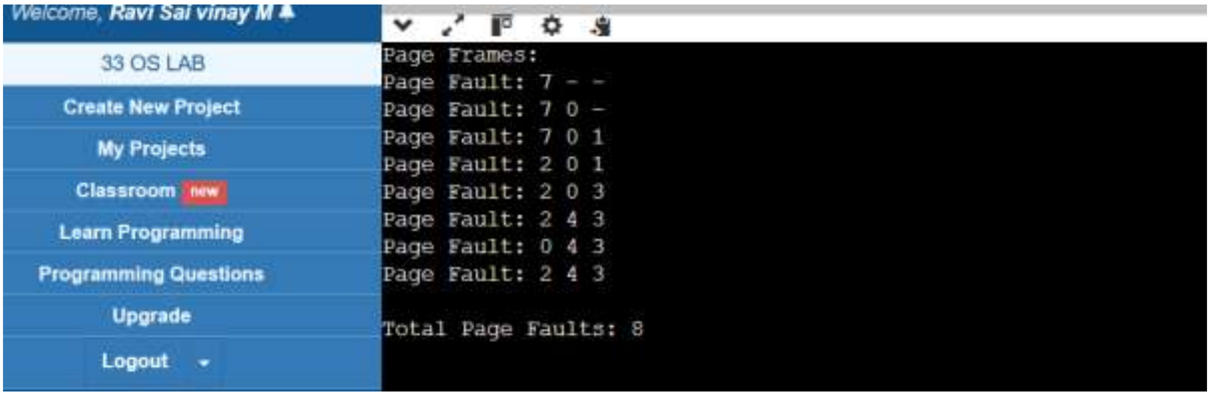
```
#include <stdio.h>
```

```
#define MAX_FRAMES 10

// Function to simulate Optimal page replacement
void optimalPageReplacement(int referenceString[], int numPages, int numFrames) {
    int frames[numFrames];  // Array to hold pages in memory
    int pageFaults = 0;      // Count of page faults

    // Initialize frames as empty (-1 means empty)
    for (int i = 0; i < numFrames; i++) {
        frames[i] = -1;
    }

    printf("Page Frames:\n");

    // Process each page in the reference string
    for (int i = 0; i < numPages; i++) {
        int page = referenceString[i];
        int pageFault = 1;

        // Check if the page is already in memory
        for (int j = 0; j < numFrames; j++) {
            if (frames[j] == page) {
                pageFault = 0;  // No page fault, page is already in memory
                break;
            }
        }

        // If page is not in memory, cause a page fault
        if (pageFault) {
            // If there is space in memory, add the new page
            int emptyIndex = -1;
            for (int j = 0; j < numFrames; j++) {
                if (frames[j] == -1) {
                    emptyIndex = j;
                    break;
                }
            }

            // If there is space in memory
            if (emptyIndex != -1) {
                frames[emptyIndex] = page;
            }
            // If memory is full, replace the page that will not be used for the longest period
            else {
                int farthestIndex = -1;
                int farthestDistance = -1;

                for (int j = 0; j < numFrames; j++) {
```

```
        int k;
        for (k = i + 1; k < numPages; k++) {
            if (referenceString[k] == frames[j]) {
                break;
            }
        }

        // If page is not used in future
        if (k == numPages) {
            farthestIndex = j;
            break;
        }
        // Otherwise, find the farthest used page
        else if (k > farthestDistance) {
            farthestDistance = k;
            farthestIndex = j;
        }
    }

    frames[farthestIndex] = page;
}

pageFaults++;
```

```
            // Print the current frame contents
            printf("Page Fault: ");
            for (int k = 0; k < numFrames; k++) {
                if (frames[k] == -1) {
                    printf("- ");
                } else {
                    printf("%d ", frames[k]);
                }
            }
            printf("\n");
        }
    }

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

int main() {
    // Reference string (sequence of page requests)
    int referenceString[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 4, 2};
    int numPages = sizeof(referenceString) / sizeof(referenceString[0]);

    int numFrames = 3;  // Number of frames in memory

    // Simulate Optimal page replacement
    optimalPageReplacement(referenceString, numPages, numFrames);



    return 0;

}
```

OUTPUT:-



```
Page Frames:
Page Fault: 7 - -
Page Fault: 7 0 -
Page Fault: 7 0 1
Page Fault: 2 0 1
Page Fault: 2 0 3
Page Fault: 2 4 3
Page Fault: 0 4 3
Page Fault: 2 4 3

Total Page Faults: 8
```

RESULT:-

Optimal Page Replacement: The program simulates the Optimal page replacement technique correctly.

Page Faults: It correctly identifies when a page fault occurs and replaces the page that will not be used for the longest period of time.

Output: The program outputs the content of the page frames and the total number of page faults.