Exp: - 12

**Problem Statement:**

Write a program to implement page replacement policy using a) LRU b) FIFO c) Optimal.

CODE: -

import java.util.\*;

public class PageReplacement {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Input number of frames and reference string

System.out.print("Enter the number of frames: ");

int framesCount = sc.nextInt();

System.out.print("Enter the number of pages in the reference string: ");

int pagesCount = sc.nextInt();

System.out.println("Enter the reference string:");

int[] referenceString = new int[pagesCount];

for (int i = 0; i < pagesCount; i++) {

referenceString[i] = sc.nextInt();

}

System.out.println("\nChoose Page Replacement Policy:");

System.out.println("1. FIFO");

System.out.println("2. LRU");

System.out.println("3. Optimal");

System.out.print("Enter your choice: ");

int choice = sc.nextInt();

switch (choice) {

case 1:

fifo(referenceString, framesCount);

break;

case 2:

lru(referenceString, framesCount);

break;

case 3:

optimal(referenceString, framesCount);

break;

default:

System.out.println("Invalid choice!");

}

sc.close();

}

// FIFO Algorithm

public static void fifo(int[] referenceString, int framesCount) {

Set<Integer> frames = new HashSet<>();

Queue<Integer> queue = new LinkedList<>();

int pageFaults = 0;

System.out.println("\nFIFO Page Replacement:");

for (int page : referenceString) {

if (!frames.contains(page)) {

pageFaults++;

if (frames.size() == framesCount) {

int removed = queue.poll();

frames.remove(removed);

}

frames.add(page);

queue.offer(page);

}

System.out.println("Frames: " + frames);

}

System.out.println("Total Page Faults: " + pageFaults);

}

// LRU Algorithm

public static void lru(int[] referenceString, int framesCount) {

List<Integer> frames = new ArrayList<>();

int pageFaults = 0;

System.out.println("\nLRU Page Replacement:");

for (int page : referenceString) {

if (!frames.contains(page)) {

pageFaults++;

if (frames.size() == framesCount) {

frames.remove(0); // Remove the least recently used page

}

frames.add(page);

} else {

frames.remove((Integer) page); // Move page to the most recently used position

frames.add(page);

}

System.out.println("Frames: " + frames);

}

System.out.println("Total Page Faults: " + pageFaults);

}

// Optimal Algorithm

public static void optimal(int[] referenceString, int framesCount) {

List<Integer> frames = new ArrayList<>();

int pageFaults = 0;

System.out.println("\nOptimal Page Replacement:");

for (int i = 0; i < referenceString.length; i++) {

int page = referenceString[i];

if (!frames.contains(page)) {

pageFaults++;

if (frames.size() == framesCount) {

int indexToReplace = -1;

int farthest = -1;

for (int j = 0; j < frames.size(); j++) {

int frame = frames.get(j);

int nextUse = Integer.MAX\_VALUE;

for (int k = i + 1; k < referenceString.length; k++) {

if (referenceString[k] == frame) {

nextUse = k;

break;

}

}

if (nextUse > farthest) {

farthest = nextUse;

indexToReplace = j;

}

}

frames.remove(indexToReplace);

}

frames.add(page);

}

System.out.println("Frames: " + frames);

}

System.out.println("Total Page Faults: " + pageFaults);

}

}

OUTPUT: -

