21BCE7371
RADHA KRISHNA GARG

OS LAB ASSIGNMENT
IMPLEMENTATION OF PAGE REPLACEMENT ALGORITHM

Page Replacement Algorithms

There are three types of Page Replacement Algorithms. They are

1. First In First Out Page Replacement Algorithm
2. Least Recently Used (LRU) Page Replacement Algorithm
3. Optimal Page Replacement Algorithm

1. First In First Out Page Replacement Algorithm

INPUT

CODE

```
#include < stdio.h >
int main()
{
    int incomingStream[] = {4 , 1 , 2 , 4 , 5};
    int pageFaults = 0;
    int frames = 3;
    int m, n, s, pages;
    pages = sizeof(incomingStream)/sizeof(incomingStream[0]);
    printf(" Incoming \ t Frame 1 \ t Frame 2 \ t Frame 3 ");
    int temp[ frames ];
    for(m = 0; m < frames; m++)
    {
        temp[m] = -1;
    }
    for(m = 0; m < pages; m++)
    {
        s = 0;
        for(n = 0; n < frames; n++)
        {
            if(incomingStream[m] == temp[n])
            {
                s++;
```

```
                pageFaults--;
            }
        }
        pageFaults++;
        if((pageFaults <= frames) && (s == 0))
        {
            temp[m] = incomingStream[m];
        }
        else if(s == 0)
        {
            temp[(pageFaults - 1) % frames] = incomingStream[m];
        }
        printf("\n");
        printf("%d\t\t\t",incomingStream[m]);
        for(n = 0; n < frames; n++)
        {
            if(temp[n] != -1)
                printf(" %d\t\t\t", temp[n]);
            else
                printf(" - \t\t\t");
        }
    }
    printf("\nTotal Page Faults:\t%d\n", pageFaults);
    return 0;
}
```

OUTPUT

```
Output
/tmp/PYwshoLPE8.o
Incoming  t Frame 1  t Frame 2  t Frame 3
4            4           -           -
1            4           1           -
2            4           1           2
4            4           1           2
5            5           1           2
Total Page Faults:  4
```

2. Least Recently Used (LRU) Page Replacement Algorithm

INPUT

CODE

```
// Java implementation of above algorithm

import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;

class Test
{
    // Method to find page faults using indexes
    static int pageFaults(int pages[], int n, int capacity)
    {
        // To represent set of current pages. We use
        // an unordered_set so that we quickly check
        // if a page is present in set or not
        HashSet<Integer> s = new HashSet<>(capacity);

        // To store least recently used indexes
        // of pages.
        HashMap<Integer, Integer> indexes = new HashMap<>();

        // Start from initial page
        int page_faults = 0;
        for (int i=0; i<n; i++)
        {
            // Check if the set can hold more pages
            if (s.size() < capacity)
            {
                // Insert it into set if not present
                // already which represents page fault
                if (!s.contains(pages[i]))
                {
                    s.add(pages[i]);

                    // increment page fault
                    page_faults++;
                }
```

```
            // Store the recently used index of
            // each page
            indexes.put(pages[i], i);
        }

        // If the set is full then need to perform lru
        // i.e. remove the least recently used page
        // and insert the current page
        else
        {
            // Check if current page is not already
            // present in the set
            if (!s.contains(pages[i]))
            {
                // Find the least recently used pages
                // that is present in the set
                int lru = Integer.MAX_VALUE, val=Integer.MIN_VALUE;

                Iterator<Integer> itr = s.iterator();

                while (itr.hasNext()) {
                    int temp = itr.next();
                    if (indexes.get(temp) < lru)
                    {
                        lru = indexes.get(temp);
                        val = temp;
                    }
                }

                // Remove the indexes page
                s.remove(val);
            //remove lru from hashmap
            indexes.remove(val);
                // insert the current page
                s.add(pages[i]);

                // Increment page faults
                page_faults++;
            }

            // Update the current page index
            indexes.put(pages[i], i);
```

```
            }
        }

        return page_faults;
    }

    // Driver method
    public static void main(String args[])
    {
        int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};

        int capacity = 4;

        System.out.println(pageFaults(pages, pages.length, capacity));
    }
}
// This code is contributed by Gaurav Miglani
```

OUTPUT

3. Optimal Page Replacement Algorithm

INPUT

CODE

```
// Java program to demonstrate optimal page
// replacement algorithm.
import java.io.*;
import java.util.*;
```

```java
class GFG {

    // Function to check whether a page exists
    // in a frame or not
    static boolean search(int key, int[] fr)
    {
        for (int i = 0; i < fr.length; i++)
            if (fr[i] == key)
                return true;
        return false;
    }

    // Function to find the frame that will not be used
    // recently in future after given index in pg[0..pn-1]
    static int predict(int pg[], int[] fr, int pn,
                       int index)
    {
        // Store the index of pages which are going
        // to be used recently in future
        int res = -1, farthest = index;
        for (int i = 0; i < fr.length; i++) {
            int j;
            for (j = index; j < pn; j++) {
                if (fr[i] == pg[j]) {
                    if (j > farthest) {
                        farthest = j;
                        res = i;
                    }
                    break;
                }
            }

            // If a page is never referenced in future,
            // return it.
            if (j == pn)
                return i;
        }

        // If all of the frames were not in future,
        // return any of them, we return 0. Otherwise
        // we return res.
        return (res == -1) ? 0 : res;
```

```
    }

    static void optimalPage(int pg[], int pn, int fn)
    {
        // Create an array for given number of
        // frames and initialize it as empty.
        int[] fr = new int[fn];

        // Traverse through page reference array
        // and check for miss and hit.
        int hit = 0;
        int index = 0;
        for (int i = 0; i < pn; i++) {

            // Page found in a frame : HIT
            if (search(pg[i], fr)) {
                hit++;
                continue;
            }

            // Page not found in a frame : MISS

            // If there is space available in frames.
            if (index < fn)
                fr[index++] = pg[i];

            // Find the page to be replaced.
            else {
                int j = predict(pg, fr, pn, i + 1);
                fr[j] = pg[i];
            }
        }
        System.out.println("No. of hits = " + hit);
        System.out.println("No. of misses = " + (pn - hit));
    }

    // driver function
    public static void main(String[] args)
    {

        int pg[]
            = { 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 };
        int pn = pg.length;
```

```
        int fn = 4;
        optimalPage(pg, pn, fn);
    }
}
```

OUTPUT