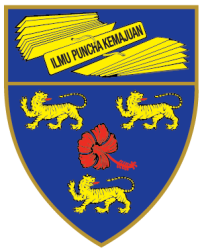


UNIVERSITI MALAYA

WIA2004 Operating Systems

Lab 8: Page Management

Group : F5U

Occ: 10

Lecturer: Dr. Fazidah Othman

GROUP MEMBERS	MATRICS NUMBER
ZHANG ZHIYANG	S2193685
MEILIN LI	S2174975
HUMYRA TASMIA	S2176677
HUSNA NADIAH BINTI ROSTHAM	22060027
HUSNA BINTI IHSANUDDIN	U2102305

Table of Content

1.0 Introduction

FIFO Page Replacement Algorithm

Page replacement algorithms are essential for managing a computer's memory. When memory is full, these algorithms determine which pages to swap out to make space for new ones.

FIFO (First-In-First-Out) is a simple page replacement strategy. The main idea is to replace the oldest page in memory first. Here's how it works:

1. Queue Management: Pages are kept in a queue based on their arrival order.
2. Page Access: If a page is already in memory, it's a hit; if not, it's a page fault.
3. Replacement: On a page fault and full memory, the page at the front of the queue (oldest) is removed and replaced with the new page.

Example: With memory frames of 3 and page requests: 1, 2, 3, 4, 1, 2, 5:

- Initially, load pages 1, 2, 3 → [1, 2, 3]
- Page 4 request replaces page 1 → [4, 2, 3]
- Page 1 request replaces page 2 → [4, 1, 3]
- And so on.

Pros:

- Simple and easy to implement.

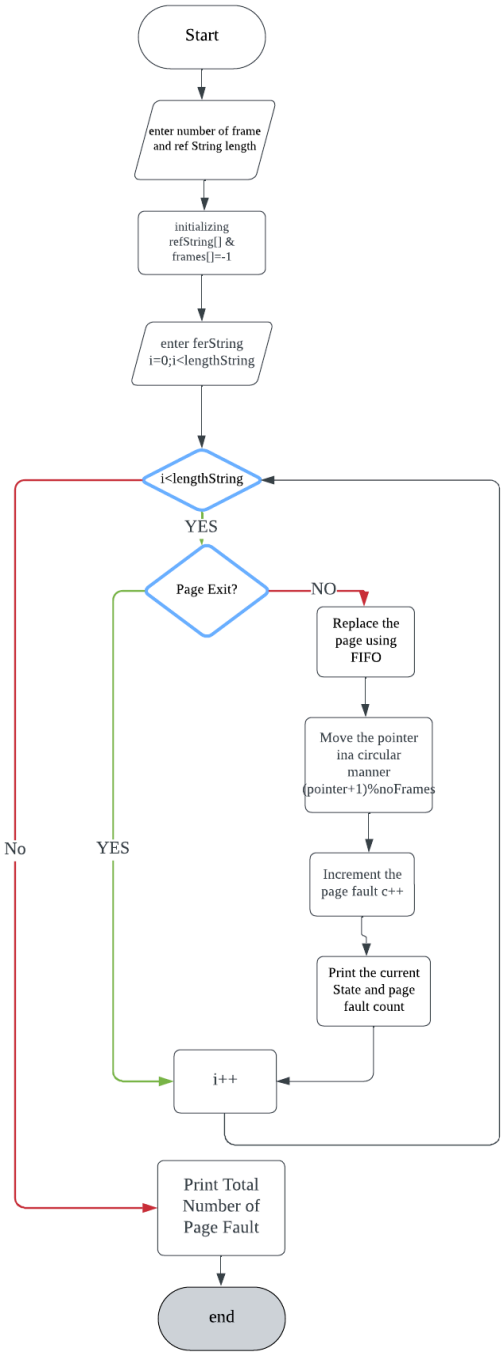
Cons:

- Can result in high page fault rates (Belady's anomaly).

FIFO is basic but often less efficient compared to other algorithms like LRU (Least Recently Used).

2.0 Methodology

Flowchart



Pseudocode

```
1. Initialize variables
   1.2 DECLARE lengthString, noFrames, count, pointer AS INTEGER
   1.3 SET count TO 0
   1.4 SET pointer TO 0

2. Get user input
   2.1 PRINT "Enter the number of frames: "
   2.3 PRINT "Enter the length of the reference string: "

3. Initialize arrays
   3.1 DECLARE referenceString[lengthString]
   3.2 DECLARE frames[noFrames]
   3.3 FOR i = 0;i< noFrames
      3.3.1 SET frames[i] = -1

4. Read the reference string
   4.1 PRINT "Enter the reference string: "
   4.2 FOR i = 0; i < lengthString
      4.2.1 READ referenceString[i]

5. Simulate the page replacement process
   5.1 PRINT "The page replacement process is:"
   5.2 FOR i= 0;i< lengthString
      5.2.1 SET pageExists= false

      5.2.2 FOR j = 0 ;j<noFrames
         5.2.2.1 IF frames[j] == referenceString[i] THEN
            5.2.2.2 SET pageExists = true
            5.2.2.3 BREAK
         5.2.2.4 END IF
      5.2.3 END FOR

      5.2.4 IF NOT pageExists THEN
         5.2.4.1 SET frames[pointer] = referenceString[i]
         5.2.4.2 SET pointer TO (pointer + 1) MOD noFrames
         5.2.4.3 INCREMENT count

         5.2.4.4 FOR j FROM 0 TO noFrames - 1
            5.2.4.4.1 IF frames[j] != -1 THEN
               5.2.4.4.2 PRINT frames[j] + " "
            5.2.4.4.3 ELSE
               5.2.4.4.4 PRINT "* "
            5.2.4.4.5 END IF
         5.2.4.5 END FOR
```

```
            5.2.4.6 PRINT "PF Count: " + count
        5.2.5 END IF
    5.3 END FOR

6. Print the total number of page faults
    6.1 PRINT "The number of page faults is: " + count
```

3.0 Implementation

3.1 Coding

```java

package oslab8;
import java.util.Scanner;

public class PageReplacement {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(source: System.in);
        int lengthString, noFrames, count = 0, previousCount = 0;

        // user input of the number of frames
        System.out.print(s: "Enter the number of frames: ");
        noFrames = scanner.nextInt();

        // user input of the reference string
        System.out.print(s: "Enter the length of the reference string: ");
        lengthString = scanner.nextInt();

        // Initialising
        int[] referenceString = new int[lengthString];
        int[] frames = new int[noFrames];


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

        // Read the reference string
        System.out.println(x: "Enter the reference string: ");
        for (int i = 0; i < lengthString; i++) {
            referenceString[i] = scanner.nextInt();
        }

        System.out.println(x: "The page replacement process is:");
        int pointer = 0;

        // Simulate the page replacement process
        for (int i = 0; i < lengthString; i++) {
            boolean pageExists = false;

            // Check if the current page is already in one of the frames
            for (int j = 0; j < noFrames; j++) {
                if (frames[j] == referenceString[i]) {
                    pageExists = true;
                    break;
                }
            }
```

```
47
48              if (!pageExists) {
49                  // Replace the page using FIFO
50                  frames[pointer] = referenceString[i];
51                  pointer = (pointer + 1) % noFrames;  // Move the pointer in a circular manner
52                  count++;  // Increment the page fault count
53
54                  // Print the current state of frame contents
55                  for (int j = 0; j < noFrames; j++) {
56                      if (frames[j] != -1) {
57                          System.out.print(frames[j] + "\t");
58                      } else {
59                          System.out.print(s: "*\t");
60                      }
61                  }
62                  System.out.println("PF Count: " + count); // Indicate a page fault occurred
63              }
64          }
65
66          // Print the total number of page faults
67          System.out.println("The number of page faults is: " + count);
68          scanner.close();
69      }
```

3.2 Sample output

```
run:
Enter the number of frames: 3
Enter the length of the reference string: 8
Enter the reference string:
1 3 0 3 5 6 3 1
The page replacement process is:
1       *       *       PF Count: 1
1       3       *       PF Count: 2
1       3       0       PF Count: 3
5       3       0       PF Count: 4
5       6       0       PF Count: 5
5       6       3       PF Count: 6
1       6       3       PF Count: 7
The number of page faults is: 7
BUILD SUCCESSFUL (total time: 4 seconds)
```

4.0 Conclusion

In conclusion, The FIFO algorithm is a fundamental approach used to manage memory in computer systems. Its simplicity and predictable behavior make it easy to understand and implement, but it also has limitations. FIFO has a high page fault rates, and can suffer from Belady's anomaly, where increasing the number of frames may lead to more page faults. Beside, it is also inefficient as it doesn't consider actual page usage patterns unlike more advanced algorithms like LRU. Hence, while FIFO serves as a basic approach, more sophisticated algorithms are often preferred for better memory management. When the number of incoming pages is less, and a user is looking for a simple approach, FIFO might be a reasonable choice.

5.0 References

- "A Study of Replacement Algorithms for a Virtual-Storage Computer" by Laszlo A. Belady: This is the original paper where Belady's anomaly is discussed. https://sci-hub.ru/10.1147/sj.52.0078#:~:text=URL%3A%20https%3A%2F%2Fsci,100