Experiment No. 9
Memory Management: Virtual Memory A. Write a program in C demonstrate the concept of page replacement policies for handling page faults eg: FIFO, LRU, Optimal
Name Of Student:-Bhagyashri Kaleni Sutar
Roll No:-75
Date of Performance:
Date of Submission:
Marks:
Sign:

Experiment No. 9

Aim: Memory Management: Virtual Memory

Objective:

To study and implement page replacement algorithm FIFO, LRU, OPTIMAL

Theory:

Demand Paging

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced.

Page Replacement Algorithm

Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated.

Reference String

The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference.

First In First Out (FIFO)

This is the simplest page replacement algorithm. In this algorithm, the OS maintains a queue that keeps track of all the pages in memory, with the oldest page at the front and the most recent page at the back.

When there is a need for page replacement, the FIFO algorithm, swaps out the page at the front of the queue, that is the page which has been in the memory for the longest time.

Least Recently Used (LRU)

Least Recently Used page replacement algorithm keeps track of page usage over a short period of time. It works on the idea that the pages that have been most heavily used in the past are most likely to be used heavily in the future too.

In LRU, whenever page replacement happens, the page which has not been used for the longest amount of time is replaced.

Optimal Page Replacement

Optimal Page Replacement algorithm is the best page replacement algorithm as it gives the least number of page faults. It is also known as OPT, clairvoyant replacement algorithm, or Belady's optimal page replacement policy.

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future, i.e., the pages in the memory which are going to be referred farthest in the future are replaced.

This algorithm was introduced long back and is difficult to implement because it requires future knowledge of the program behavior. However, it is possible to implement optimal page replacement on the second run by using the page reference information collected on the first run.

Program:

1.FIFO:-

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>


void FIFO(int referenceString[], int n, int frameCount) {

   int frames[frameCount];

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

      frames[i] = -1; // Initially, all frames are empty

   }
```

```
int pageFaults = 0;

int pageHits = 0;

int pageMisses = 0;

int index = 0;


printf("FIFO Page Replacement:\n");


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

    int page = referenceString[i];

    bool pageFound = false;


    // Check if page is already in the frames

    for (int j = 0; j < frameCount; j++) {

        if (frames[j] == page) {

            pageFound = true;

            break;

        }

    }


    if (pageFound) {

        pageHits++;

    } else {

        frames[index] = page;

        index = (index + 1) % frameCount; // Move to the next frame in a circular manner

        pageFaults++;

        pageMisses++;

    }
```

```
        // Display current frames
        printf("Frames: ");
        for (int j = 0; j < frameCount; j++) {
            if (frames[j] != -1) {
                printf("%d ", frames[j]);
            }
        }
        printf("\n");
    }

    printf("Total Page Hits: %d\n", pageHits);
    printf("Total Page Faults: %d\n", pageFaults);
    printf("Total Page Misses: %d\n", pageMisses);
}

int main() {
    int referenceString[] = {5,0,2,3,0,1,3,4,5,4,2,0,3,4,3};
    int n = sizeof(referenceString) / sizeof(referenceString[0]);
    int frameCount = 3;

    FIFO(referenceString, n, frameCount);
    return 0;
}
//Output
```

```
FIFO Page Replacement:
Frames: 5
```

```
Frames: 5 0

Frames: 5 0 2

Frames: 3 0 2

Frames: 3 0 2

Frames: 3 1 2

Frames: 3 1 2

Frames: 3 1 4

Frames: 5 1 4

Frames: 5 1 4

Frames: 5 2 4

Frames: 5 2 0

Frames: 3 2 0

Frames: 3 4 0

Frames: 3 4 0

Total Page Hits: 4

Total Page Faults: 11

Total Page Misses: 11
```

2.LRU:-

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>


void LRU(int referenceString[], int n, int frameCount) {

   int frames[frameCount];

   int recentUse[frameCount];

   for (int i = 0; i < frameCount; i++) {
       frames[i] = -1; // Initially, all frames are empty
       recentUse[i] = -1;
   }

   int pageFaults = 0;
   int pageHits = 0;
   int pageMisses = 0;

   printf("LRU Page Replacement:\n");

   for (int i = 0; i < n; i++) {
       int page = referenceString[i];
       bool pageFound = false;

       // Check if the page is already in the frames
       for (int j = 0; j < frameCount; j++) {
           if (frames[j] == page) {
               pageFound = true;
               pageHits++;
               recentUse[j] = i; // Update the recent use
               break;
           }
       }

       if (!pageFound) {
           // Page miss: Replace the least recently used page
           int lruIndex = 0;
           for (int j = 1; j < frameCount; j++) {
               if (recentUse[j] < recentUse[lruIndex]) {
                   lruIndex = j;
               }
           }

           frames[lruIndex] = page;
           recentUse[lruIndex] = i; // Update the recent use
           pageFaults++;
           pageMisses++;
       }

       // Display current frames
       printf("Frames: ");
       for (int j = 0; j < frameCount; j++) {
           if (frames[j] != -1) {
               printf("%d ", frames[j]);
           }
       }
       printf("\n");
   }

   printf("Total Page Hits: %d\n", pageHits);
   printf("Total Page Faults: %d\n", pageFaults);
   printf("Total Page Misses: %d\n", pageMisses);
}
```

```
int main() {
   int referenceString[] = {5,0,2,3,0,1,3,4,5,4,2,0,3,4,3};
   int n = sizeof(referenceString) / sizeof(referenceString[0]);
   int frameCount = 3;


   LRU(referenceString, n, frameCount);
   return 0;
}
//Output
LRU Page Replacement:
Frames: 5
Frames: 5 0
Frames: 5 0 2
Frames: 3 0 2
Frames: 3 0 2
Frames: 3 0 1
Frames: 3 0 1
Frames: 3 4 1
Frames: 3 4 5
Frames: 3 4 5
Frames: 2 4 5
Frames: 2 4 0
Frames: 2 3 0
Frames: 4 3 0
Frames: 4 3 0
Total Page Hits: 4
Total Page Faults: 11
```

Total Page Misses: 11

3.OPTIMAL:-

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>


int findNextUse(int referenceString[], int n, int page, int currentIndex) {

   for (int i = currentIndex + 1; i < n; i++) {

      if (referenceString[i] == page) {

         return i;

      }

   }

   return -1; // Return -1 if the page is not found in the future

}


void Optimal(int referenceString[], int n, int frameCount) {

   int frames[frameCount];

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

      frames[i] = -1; // Initially, all frames are empty

   }


   int pageFaults = 0;

   int pageHits = 0;

   int pageMisses = 0;


   printf("Optimal Page Replacement:\n");
```

```
for (int i = 0; i < n; i++) {
    int page = referenceString[i];
    bool pageFound = false;

    // Check if the page is already in the frames
    for (int j = 0; j < frameCount; j++) {
        if (frames[j] == page) {
            pageFound = true;
            pageHits++;
            break;
        }
    }

    if (!pageFound) {
        // If there is an empty frame, put the page in it
        bool placed = false;
        for (int j = 0; j < frameCount; j++) {
            if (frames[j] == -1) {
                frames[j] = page;
                pageFaults++;
                pageMisses++;
                placed = true;
                break;
            }
        }
        // If no empty frame, replace the page that will not be used for the longest time
        if (!placed) {
            int farthest = -1, indexToReplace = -1;
            for (int j = 0; j < frameCount; j++) {
                int nextUse = findNextUse(referenceString, n, frames[j], i);
                if (nextUse == -1) {
                    indexToReplace = j;
                    break;
                } else if (nextUse > farthest) {
                    farthest = nextUse;
                    indexToReplace = j;
                }
            }
            frames[indexToReplace] = page;
            pageFaults++;
            pageMisses++;
        }
    }

    // Display current frames
    printf("Frames: ");
    for (int j = 0; j < frameCount; j++) {
        if (frames[j] != -1) {
            printf("%d ", frames[j]);
        }
    }
    printf("\n");
    }

    printf("Total Page Hits: %d\n", pageHits);
    printf("Total Page Faults: %d\n", pageFaults);
    printf("Total Page Misses: %d\n", pageMisses);
}

int main() {
    int referenceString[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3};
    int n = sizeof(referenceString) / sizeof(referenceString[0]);
    int frameCount = 3;

    Optimal(referenceString, n, frameCount);
    return 0;
}
//output
```

```
Optimal Page Replacement:
Frames: 7
Frames: 7 0
Frames: 7 0 1
Frames: 2 0 1
Frames: 2 0 1
Frames: 2 0 3
Frames: 2 0 3
Frames: 2 4 3
Frames: 2 4 3
Frames: 2 4 3
```

Frames: 0 4 3

Frames: 0 4 3

Total Page Hits: 5

Total Page Faults: 7

Total Page Misses: 7

Conclusion:

Why do we need page replacement strategies ?

Ans:- age replacement strategies are essential in operating systems that utilize virtual memory. Here's why:

- **Virtual Memory and Limited Physical RAM:**
    - **Modern operating systems employ virtual memory, which allows programs to use more memory than is physically available in RAM.**
    -
    - **This is achieved by storing portions of programs (pages) on secondary storage (like a hard drive or SSD) and bringing them into RAM only when needed.**
    -
    - **However, RAM is a limited resource, so when it's full, the operating system must decide which page to remove to make room for a new one.**
    -
- **Managing Page Faults:**
    - **When a program tries to access a page that isn't in RAM, a "page fault" occurs.**
    -
    - **The operating system must then retrieve the required page from secondary storage, which is a relatively slow process.**
    - **Page replacement algorithms aim to minimize the number of page faults by selecting the "best" page to remove.**
    -
- **Optimizing Performance:**
    - **A well-designed page replacement strategy can significantly improve system performance by reducing the frequency of page faults.**
    -
    - **This leads to faster program execution and a more responsive user experience.**
- **Efficient Memory Utilization:**
    - **Page replacement algorithms help the operating system utilize RAM efficiently by keeping frequently used pages in memory and removing those that are less likely to be needed.**
    -

- **This allows the system to run more programs concurrently without experiencing excessive slowdowns.**

In essence, page replacement strategies are crucial for effectively managing virtual memory and ensuring that computer systems can handle large programs and multitask efficiently.