Tribhuvan University
Faculty of Humanities and Social Sciences

Lab report on:
Operating System Lab 8:
Page replacement algorithms

Submitted to:
Mr. Roshan Maharjan,
Er. Himal Chand Thapa ,
Department of Computer Application,
Himalaya College of Engineering,
Chyasal,Lalitpur

Submitted by:
Sujal Gurung
Roll no: 34
BCA II/IV
September 21, 2023

1 Objectives

- to implement FIFO & MRU page replacement algorithms

2 Introduction

The previously discussed partitioning system falls under continguous memory allocation i.e. adjacent memory addresses are allocated. There is possibility of internal fragmentation which causes under-utilization of resources. To combat this, operating systems can use non-contiguous allocation methods. One method is **paging**, where process is divided into equal size blocks(**pages**) & memory is divided into same sized **frames**. A process reqeuests required page to be loaded into memory & if it isn't there alread, a page fault occurs. OS may have to swap page residing in memory with one that is requested. Page replacement algorithms can help decide which to replace, so as to keep system efficient. The 2 algorithms covered in this lab are:

- **FIFO:** Oldest page in memory is replaced, hoping it won't be needed for a while.
- **Most Recently Used(MRU):** Page that was most recently requested is replaced.

3 Lab Work

3.1 Write a program to find the no of pagefault using FIFO page replacement algorithm

```c
#include <stdio.h>
#define MAX_SIZE 100
int main() {
  int n, pages[MAX_SIZE], frames[3] = {-1, -1, -1}, faults = 0, i, j, k, pos = 0,
  ↪  flag;
    printf("Enter the number of pages: ");
    scanf("%d", &n);
    printf("Enter the page reference string: ");
    for (i = 0; i < n; i++) {
        scanf("%d", &pages[i]);
    }
    for (i = 0; i < n; i++) {
        flag = 0;
        for (j = 0; j < 3; j++) {
            if (frames[j] == pages[i]) {
                flag = 1;
                break;
            }
        }
        if (flag == 0) {
            frames[pos] = pages[i];
            pos = (pos + 1) % 3;
            faults++;
        }
        printf("\nFrame: ");
        for (k = 0; k < 3; k++) {
            printf("%d ", frames[k]);
        }
    }
```

```
    printf("\n\nTotal Page Faults: %d\n", faults);
    return 0;
}
```

Output Note: A frame having the contents `-1` indicates that it is currently empty

```
Enter the number of pages: 10
Enter the page reference string: 4 7 6 1 7 6 1 2 7 2

Frame: 4 -1 -1
Frame: 4 7 -1
Frame: 4 7 6
Frame: 1 7 6
Frame: 1 7 6
Frame: 1 7 6
Frame: 1 7 6
Frame: 1 2 6
Frame: 1 2 7
Frame: 1 2 7

Total Page Faults: 6
```

3.2 Write a program to find the no of pagefault using MRU page replacement algorithm

```
#include <stdio.h>
#define MAX 30
int main() {
    int frames[MAX], pages[MAX], freq[MAX], index[MAX], num_frames, num_pages,
    ↪  page_faults = 0, i, j, k, max_index;

    printf("Enter the number of frames: ");
    scanf("%d", &num_frames);

    printf("Enter the number of pages: ");
    scanf("%d", &num_pages);

    printf("Enter the page reference string: ");
    for(i=0; i<num_pages; i++)
        scanf("%d", &pages[i]);

    // Initialize the frames with -1
    for(i=0; i<num_frames; i++)
        frames[i] = -1;

    for(i=0; i<num_pages; i++) {
        // Check if the page is already present in the frame
        int flag = 0;
        for(j=0; j<num_frames; j++) {
            if(frames[j] == pages[i]) {
                flag = 1;
                break;
            }
        }
        if(flag == 0) {
            // If the page is not present in the frame, find the page with the
            ↪  maximum frequency
            for(j=0; j<num_frames; j++) {
```

```
                int temp = frames[j];
                freq[j] = 0;
                for(k=0; k<num_pages; k++) {
                    if(temp == pages[k])
                        break;
                    freq[j]++;
                }
            }
            max_index = 0;
            for(j=1; j<num_frames; j++) {
                if(freq[j] > freq[max_index])
                    max_index = j;
            }

            // Replace the page with the maximum frequency
            frames[max_index] = pages[i];
            index[max_index] = i;
            page_faults++;
        }
        // Print the frames after each page replacement
        printf("\n");
        for(j=0; j<num_frames; j++)
        {
            printf("|%d|", frames[j]);
        }
        if(frames[2] == pages[i])
            printf("\tPage fault for frame no. 3");  }
    printf("\nTotal number of page faults: %d", page_faults);
    return 0;
}
```

Output

```
Enter the number of frames: 3
Enter the number of pages: 10
Enter the page reference string: 4 7 6 1 7 6 1 2 7 2

|4||-1||-1|
|4||7||-1|
|4||7||6|       Page fault for frame no. 3
|4||7||1|       Page fault for frame no. 3
|4||7||1|
|4||7||6|       Page fault for frame no. 3
|4||7||1|       Page fault for frame no. 3
|4||7||2|       Page fault for frame no. 3
|4||7||2|
|4||7||2|       Page fault for frame no. 3
Total number of page faults: 7
```

4 Conclusion

While FIFO generally produces less page faults than MRU, it all depends on factors such as number of pages, frams and the page reference string. Thus, we learned the differences between the 2 methods by implementing them.