Ex. No.: 11a)
Date: 16/4/25

FIFO PAGE REPLACEMENT

Aim:

To find out the number of page faults that occur using First-in First-out (FIFO) page replacement technique.

Algorithm:

1. Declare the size with respect to page length
2. Check the need of replacement from the page to memory
3. Check the need of replacement from old page to new page in memory 4.
Form a queue to hold all pages
5. Insert the page require memory into the queue
6. Check for bad replacement and page fault
7. Get the number of processes to be inserted
8. Display the values

Program Code:

```
#include <stdio.h>
#define MAX 100
int main(){
    int page[MAX], queue[MAX];
    int n, Capacity;
    int front=0, rear=0, Page faults=0;
    int i,j, found;
    printf("Enter the number of page: ");
    Scanf("%d", &n);
    Printf("Enter the reference string: ");
    for(i=0; i<n; i++){
        Scanf("%d", &page[i]);
    }
    printf("Enter page frame size: ");
    Scanf("%d", &Capacity);
```

```
int count = 0
for (i=0; i<n; i++){
    found = 0;
    for (j=0; j<count; j++){
      if (queue[j] == page[i]){
          found = 1;
          break;
      }
    }
    if(!found){
        page faults++;
        if (count < capacity) {
          queue [count++] = page [i];
        }
        else{
          queue [front] = page [i];
          front = (front +1) % capacity;
        }
    }
    printf("%d", page [i]);
    for (j=0; j<count; j++){
        printf("%d", queue[j]);
    }
    print("\n");
}
printf("Total Page faults: %d\n", Page faults);
return 0;
}
```

Sample Output:

```
[root@localhost student]# python fifo.py

Enter the size of reference string: 20
Enter [ 1] : 7
Enter [ 2] : 0
Enter [ 3] : 1
Enter [ 4] : 2
Enter [ 5] : 0
Enter [ 6] : 3
Enter [ 7] : 0
Enter [ 8] : 4
Enter [ 9] : 2
Enter [10] : 3
Enter [11] : 0
Enter [12] : 3
Enter [13] : 2
Enter [14] : 1
Enter [15] : 2
Enter [16] : 0
Enter [17] : 1
Enter [18] : 7
Enter [19] : 0
Enter [20] : 1

Enter page frame size : 3

7 -> 7 - -
0 -> 7 0 -
1 -> 7 0 1
2 -> 2 0 1
0 -> No Page Fault

3 -> 2 3 1
0 -> 2 3 0
4 -> 4 3 0
2 -> 4 2 0
3 -> 4 2 3
0 -> 0 2 3
3 -> No Page Fault
2 -> No Page Fault
1 -> 0 1 3
2 -> 0 1 2
0 -> No Page Fault
1 -> No Page Fault
7 -> 7 1 2
0 -> 7 0 2
```

Input:

Enter the number of pages: 12

Enter the reference string: 1 3 0 3 5 6 3 0 6 4 1 7

Enter the frame size: 3

Output:

1: 1

3: 1 3

0: 1 3 0

3: 1 3 0

5: 5 3 0

6: 5 6 0

3 0: 5 6 3

0: 0 6 3

6: 0 6 3

4: 4 6 3

1: 4 1 3

7: 4 1 7

Total page faults: 9

1 -> 7 0 1
Total page faults: 15.
[root@localhost student]#

Result :

Thus, number of pages faults using FIFO page replacement algorithm.

Ex. No.: 11b)
Date:

LRU

Aim:

To write a c program to implement LRU page replacement algorithm.

Algorithm:

1: Start the process
2: Declare the size
3: Get the number of pages to be inserted
4: Get the value
5: Declare counter and stack
6: Select the least recently used page by counter value
7: Stack them according the selection.
8: Display the values
9: Stop the process

Program Code:

```
# include <stdio.h>
# include <stdlib.h>
# define MAX 100
int find LRU(int time[], int n){
    int i, minimum = time[0], pos=0;
    for (i=1; i<n; i++){
        if (time[i] < minimum){
            minimum = time[i];
            pos=i;
        }
    }
    return pos;
}
int main(){
    int frame[MAX], page[MAX], time[MAX];
    int n, capacity, counter=0, fault=0;
    int i, j, pos, flag, flag2;
```

```
printf("Enter the number of pages : ");
scanf("%d", &n);
printf("Enter the pages reference string : ");
for (i=0; i<n; i++){
    scanf("%d", &pages[i])
}
printf("Enter the number of frames : ");
scanf("%d", &capacity);
for (i=0; i<capacity; i++){
        frames[i] = -1;
}
for(i=0; i<n; i++){
    flag1 = flag2 = 0;
    for (j=0; j<capacity; j++){
        if (frames[j] == pages[i]){
            counter++;
            time[j] = counter++;
            flag1 = flag2 = 1;
            break;
        }
    }
    if (flag==0){
        for (j=0; j<capacity; j++){
        if ( frames[j] == -1){
            counter++;
```

```
            frames++;
            frames[j] = pages[i];
            time[j] = Counter;
            flag2 = -1;
            break;
        }
    }
}
if (flag2 == 0){
    pos = findLU(time, Capacity);
    Counter++;
    faults++;
    frames[pos] = pages[i];
    time[pos] = Counter;
}
printf("Memory after inserting %d: ", pages[i]);
for (j = 0; j < Capacity; j++){
    if(frames[j] != -1)
        printf("%d", frames[j]);
    else
        printf("_");
}
printf("Total Page faults = %d\n", faults);
return 0;
}
```

Input:

Enter the number of pages : 12

Enter the reference string : 1 3 0 3 5 6 3 0 6 4 1 7

Enter the number of frame : 3

Output:

1 : 1 _

3 : 13 _

0 : 130

3 : 130

5 : 530

6 : 560

3 : 360

0 : 360

6 : 360

4 : 460

1 : 410

7 : 417

Total page faults = 9.

Sample Output :
Enter number of frames: 3
Enter number of pages: 6
Enter reference string: 5 7 5 6 7 3
5 -1 -1
5 7 -1
5 7 -1
5 7 6
5 7 6
3 7 6
Total Page Faults = 4

Result:

Thus. LRU page replacement algorithm is implemented and executed successfully.

Ex. No.: 11c)
Date: 23/4/25

Optimal

Aim:
To write a c program to implement Optimal page replacement algorithm.

ALGORITHM:

1. Start the process
2. Declare the size
3. Get the number of pages to be inserted
4. Get the value
5. Declare counter and stack
6. Select the least frequently used page by counter value
7. Stack them according the selection.
8. Display the values
9. Stop the process

PROGRAM:

```
# include <stdio.h>
# define MAX 100
int predict [int pages [], int frames [], int n, int index, int Capacity){
    int result =-1, farthest = index ;
    for (int i=0; i< Capacity; i++){
        int j;
        for (j = index ; j<n; j++){
        if ( frames [i] == pages [j] {
            if (j > farthest){
                farthest = j;
                result = i;
}
```

```
        break;
    }
  }
  if(j==n)
    return i;
  }
  return(result==-1)?0:result;
}
int main(){
  int pages[MAX], frames[MAX];
  int n, capacity, faults=0, hit=0;
  int i, j, k, filled=0;
  printf("Enter the Number of pages:");
  scanf("%d", &n);
  printf("Enter the reference string:");
  for(i=0; i<n; i++){
      scanf("%d", &pages[i]);
  }
  printf("Enter the number of frames:");
  scanf("%d", &capacity);
  for(i=0; i<capacity; i++){
        frames[i]=-1;
  }
  for(i=0; i<n; i++){
        int found=0;
        for(j=0; j<capacity; j++){
          if(frames[j]==pages[i]){
            found=1;
            hit++;
            break;
          }
```

```
    if (!found){
      if (filled < Capacity){
        frames[filled++] = page[i];
      }
      else{
        int pos = predict(page, frames, n, i+1, Capacity);
        frames[pos] = page[i];
      }
      faults++;
    }

    printf("%d", page[i]);
    for (k=0; k<Capacity; k++){
      if (frames[k] != -1){
          printf(" %d", frames[k]); }
      else{
        printf("-");
      }
      printf("\n");
    }

  printf("Total Page faults = %d\n", faults);
  printf("Total Page Hit = %d\n", hit);
  return 0;
}
```

Output:

Enter the number of pages : 12
Enter the reference string : 1 3 0 3 5 6 3 0 6 4 1 7
Enter the number of frame : 3

1 : 1 - -
3 : 1 3 -
0 : 1 3 0
3 : 1 3 0
5 : 5 3 0
6 : 5 6 0
3 : 5 6 3
0 : 5 0 3
6 : 6 0 3
4 : 6 0 4
1 : 1 0 4
7 : 1 7 4
Total Page faults = 9
Total page Hits = 3

Result:

Thus, optimal page replacement algorithms is implemented and executed successfully.