EX:10 PAGE REPLACEMENT TECHNIQUES

-S.Vishakan CSE-C 18 5001 196

Source Code:

```
#include <stdio.h>
#include <stdlib.h>

#define MAXSIZE 20 //max. size of reference string

struct Node
{ //node for linked list based queue
   int data;
   struct Node *next;
};

typedef struct Node node;

node *current = NULL; //pointers for queue
node *prev = NULL;

node   *enqueue(node *head, int data);
node   *dequeue(node *head);
node   *delete (node *head, int data);
int    search(node *head, int val);
int    getSize(node *head);
void   printList(node *head);
void   toArray(node *head, int arr[]);
int    min(int arr[], int len);
int    max(int arr[], int len);
int    linearSearch(int arr[], int start, int len, int elt);
int    occurenceCounter(int arr[], int start, int len, int elt);
void   printArray(int arr[], int len);
int    FIFO(int ref_str[], int len, int fsize);
int    LRU(int ref_str[], int len, int fsize);
int    OPT(int ref_str[], int len, int fsize);
int    LFU(int ref_str[], int len, int fsize);

int main(void)
{

   int ref_str[MAXSIZE] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1}; //default loaded ref.
string.
   int len = MAXSIZE, opt = -1, fsize = 3, pagefaults = 0, i = 0;
```

```
    while (opt != 0)
    {
        printf("\n\n\t\t\tMain Menu\n\n\t1. Enter the Reference String\n\t2. View the Reference String\
n\t3. Implement FIFO Algorithm\n\t4. Implement LRU Algorithm\n\t5. Implement Optimal
Algorithm\n\t6. Implement LFU Algorithm\n\t0. Exit\n\tYour Choice -> ");
        scanf("%d", &opt);

        switch (opt)
        {
        case 1:
            printf("\nEnter the length of the reference string(Maximum : 20): ");
            scanf("%d", &len);
            printf("\nEnter the reference string: ");
            for (i = 0; i < len; i++)
            {
                scanf("%d", &ref_str[i]);
            }
            printf("\nEnter the frame size: ");
            scanf("%d", &fsize);
            break;

        case 2:
            printf("\n\tReference String:\n\t");
            printArray(ref_str, len);
            printf("\n\n\tLength of Reference String: %d", len);
            printf("\n\n\tFrame Size: %d", fsize);
            break;

        case 3:
            pagefaults = FIFO(ref_str, len, fsize);
            printf("\nNo. of Page Faults on performing FIFO Algorithm: %d", pagefaults);
            break;

        case 4:
            pagefaults = LRU(ref_str, len, fsize);
            printf("\nNo. of Page Faults on performing LRU Algorithm: %d", pagefaults);
            break;

        case 5:
            pagefaults = OPT(ref_str, len, fsize);
            printf("\nNo. of Page Faults on performing Optimal Algorithm: %d", pagefaults);
            break;

        case 6:
            pagefaults = LFU(ref_str, len, fsize);
            printf("\nNo. of Page Faults on performing LFU Algorithm: %d", pagefaults);
            break;

        case 0:
            printf("\n\t\tThank You!\n");
            break;
```

```
        default:
            printf("\nInvalid Choice.");
            break;
        }
    };

    return 0;
}

node *enqueue(node *head, int data)
{ //enqueuing a new frame on the loaded frame queue
    node *new_node = (node *)malloc(sizeof(node));
    new_node->data = data;
    new_node->next = head;
    head = new_node;

    return head;
}

node *dequeue(node *head)
{ //dequeueing from the loaded frame queue
    node *temp = head;

    if (head == NULL)
    {
        return head;
    }

    else if (head->next == NULL)
    {
        free(head);
        head = NULL;

        return head;
    }

    for (temp = head; temp->next != NULL; temp = temp->next)
    {
        prev = temp;
    }

    free(temp);
    prev->next = NULL;

    return head;
}

node *delete (node *head, int data)
{ //deleting a particular frame from the frame queue
    if (head == NULL)
    {
        return head;
```

```
    }

    else if (head->data == data)
    {
        node *temp = head;
        head = head->next;
        free(temp);

        return head;
    }

    else
    {
        node *temp = NULL;
        node *t = NULL;

        for (temp = head; temp->next != NULL; temp = temp->next)
        {
            if ((temp->next)->data == data)
            {
                t = temp->next;
                temp->next = (temp->next)->next;
                free(t);
                break;
            }
        }
    }

    return head;
}

int search(node *head, int data)
{ //searching for a particular frame in the queue
    current = head;

    if (head == NULL)
    {
        return 0;
    }

    while (current != NULL)
    {
        if (current->data == data)
        {
            return 1;
        }
        current = current->next;
    }

    return 0;
}
```

```
int getSize(node *head)
{ //obtaining the size of the queue
   int size = 0;

   if (head == NULL)
   {
      return 0;
   }

   current = head;
   size = 1;

   while (current->next != NULL)
   {
      size++;
      current = current->next;
   }

   return size;
}

void printList(node *head)
{ //printing the loaded frame queue
   node *temp = head;
   printf("\n");
   while (temp != NULL)
   {
      printf("%d ", temp->data);
      temp = temp->next;
   }

   printf("\n");
}

void printArray(int arr[], int len)
{ //printing an array
   int i = 0;
   printf("\n");
   for (i = 0; i < len; i++)
   {
      printf("%d ", arr[i]);
   }
   printf("\n");
}

void toArray(node *head, int arr[])
{ //converting a linked list to an array
   node *temp = head;
   int i = 0;

   while (temp != NULL)
   {
```

```
        arr[i] = temp->data;
        temp = temp->next;
        i++;
    }
}

int min(int arr[], int len)
{ //finding index of min. element of an array
    int min = 0, i = 0;
    for (i = 0; i < len; i++)
    {
        if (arr[i] <= arr[min])
        {
            min = i;
        }
    }

    return min;
}

int max(int arr[], int len)
{ //finding index of max. element of an array
    int max = 0, i = 0;
    for (i = 0; i < len; i++)
    {
        if (arr[i] > arr[max])
        {
            max = i;
        }
    }

    return max;
}

int linearSearch(int arr[], int start, int len, int elt)
{ //searching for a specific element in an array
    int index = 9999, i = 0;

    for (i = start; i < len; i++)
    {
        if (arr[i] == elt)
        {
            index = i;
        }
    }

    return index;
}

int occurenceCounter(int arr[], int start, int len, int elt)
{ //finding number of occurrences of a frame in a reference string
    int count = 0, i = 0;
```

```
    for (i = start; i < len; i++)
    {
        if (arr[i] == elt)
        {
            count++;
        }
    }

    return count;
}

int FIFO(int ref_str[], int len, int fsize)
{ //performing the FIFO algorithm
    node *head = NULL;
    int i = 0, listsize = 0, found, pagefaults = 0;

    printf("\nImplementing FIFO Algorithm: \n");

    for (i = 0; i < len; i++)
    {
        listsize = getSize(head);

        if (listsize < fsize)
        { //initial loading to frame queue
            found = search(head, ref_str[i]);
            if (found == 0)
            {
                pagefaults++;
                head = enqueue(head, ref_str[i]);
            }
            printList(head);
        }
        else
        { //page replacement strategy : FIFO
            found = search(head, ref_str[i]);
            if (found == 0)
            {
                pagefaults++;
                head = dequeue(head); //FIFO dequeue-enqueue
                head = enqueue(head, ref_str[i]);
            }
            printList(head);
        }
    }

    return pagefaults;
}
```

```
int LRU(int ref_str[], int len, int fsize)
{ //performing the LRU algorithm
   node *head = NULL;
   int current_list[MAXSIZE];
   int current_found_index[fsize];
   int i = 0, j = 0, listsize = 0, found = 0, pagefaults = 0, min_used;

   printf("\nImplementing Least Recently Used Algorithm: \n");

   for (i = 0; i < len; i++)
   {
      listsize = getSize(head);
      if (listsize < fsize)
      { //initial loading of frame queue
         found = search(head, ref_str[i]);
         if (found == 0)
         {
            pagefaults++;
            head = enqueue(head, ref_str[i]);
         }
         printList(head);
      }
      else
      { //page replacement strategy : LRU
         found = search(head, ref_str[i]);
         if (found == 0)
         {
            pagefaults++;
            toArray(head, current_list);
            for (j = 0; j < fsize; j++)
            { //finding the least recently used page
               current_found_index[j] = linearSearch(ref_str, 0, i, current_list[j]);
            }
            min_used = min(current_found_index, fsize);
            if (min_used == fsize - 1)
            { //tie breaker : FIFO
               head = dequeue(head);
               head = enqueue(head, ref_str[i]);
            }
            else
            { //replace the least recently used page
               head = delete (head, current_list[min_used]);
               head = enqueue(head, ref_str[i]);
            }
         }
         printList(head);
      }
   }

   return pagefaults;
}
```

```
int OPT(int ref_str[], int len, int fsize)
{ //performing the Optimal algorithm
   node *head = NULL;
   int current_list[MAXSIZE];
   int current_found_index[fsize];
   int i = 0, j = 0, found = 0, pagefaults = 0, listsize = 0, max_used = 0;

   printf("\nPerforming Optimal Algorithm: \n");

   for (i = 0; i < len; i++)
   {
      listsize = getSize(head);
      if (listsize < fsize)
      { //initial loading of frame queue
         found = search(head, ref_str[i]);
         if (found == 0)
         {
            pagefaults++;
            head = enqueue(head, ref_str[i]);
         }
      }
      else
      { //page replacement strategy : Optimal
         found = search(head, ref_str[i]);
         if (found == 0)
         {
            pagefaults++;
            toArray(head, current_list);
            for (j = 0; j < fsize; j++)
            { //finding the next usage of each frame in the queue in the ref. string
               current_found_index[j] = linearSearch(ref_str, i + 1, len, current_list[j]);
            }
            max_used = max(current_found_index, fsize); //replacing the latest used frame with the
new frame
            head = delete (head, current_list[max_used]);
            head = enqueue(head, ref_str[i]);
         }
      }

      printList(head);
   }

   return pagefaults;
}

int LFU(int ref_str[], int len, int fsize)
{ //performing the LFU algorithm
   node *head = NULL;
   int current_list[MAXSIZE];
   int frame_counts[fsize];
   int i = 0, j = 0, listsize = 0, found = 0, pagefaults = 0, min_used;
```

```
    printf("\nImplementing Least Frequently Used Algorithm: \n");

    for (i = 0; i < len; i++)
    {
        listsize = getSize(head);
        if (listsize < fsize)
        { //initial loading of frame queue
            found = search(head, ref_str[i]);
            if (found == 0)
            {
                pagefaults++;
                head = enqueue(head, ref_str[i]);
            }
            printList(head);
        }
        else
        { //page replacement strategy : LRU
            found = search(head, ref_str[i]);
            if (found == 0)
            {
                pagefaults++;
                toArray(head, current_list);
                for (j = 0; j < fsize; j++)
                { //finding the least frequently used page
                    frame_counts[j] = occurenceCounter(ref_str, 0, i, current_list[j]);
                }
                min_used = min(frame_counts, fsize);
                if (min_used == fsize - 1)
                { //tie breaker : FIFO
                    head = dequeue(head);
                    head = enqueue(head, ref_str[i]);
                }
                else
                { //replace the least recently used page
                    head = delete (head, current_list[min_used]);
                    head = enqueue(head, ref_str[i]);
                }
            }
            printList(head);
        }
    }

    return pagefaults;
}
```

OUTPUT:

(base) vishakan@Legion:~/Desktop/Operating-Systems/Ex 10 Page Replacement Techniques$ gcc Replacement.c -o r
(base) vishakan@Legion:~/Desktop/Operating-Systems/Ex 10 Page Replacement Techniques$ ./r

Main Menu

1. Enter the Reference String
2. View the Reference String
3. Implement FIFO Algorithm
4. Implement LRU Algorithm
5. Implement Optimal Algorithm
6. Implement LFU Algorithm
0. Exit
Your Choice -> 1

Enter the length of the reference string(Maximum : 20): 20

Enter the reference string: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Enter the frame size: 3

Main Menu

1. Enter the Reference String
2. View the Reference String
3. Implement FIFO Algorithm
4. Implement LRU Algorithm
5. Implement Optimal Algorithm
6. Implement LFU Algorithm
0. Exit
Your Choice -> 3

Implementing FIFO Algorithm:

7

0 7

1 0 7

2 1 0

2 1 0

3 2 1

0 3 2

4 0 3

2 4 0

3 2 4

0 3 2

0 3 2

0 3 2

1 0 3

2 1 0

2 1 0

2 1 0

7 2 1

0 7 2

1 0 7

No. of Page Faults on performing FIFO Algorithm: 15

Main Menu

1. *Enter the Reference String*
2. *View the Reference String*
3. *Implement FIFO Algorithm*
4. *Implement LRU Algorithm*
5. *Implement Optimal Algorithm*
6. *Implement LFU Algorithm*
0. *Exit*

Your Choice -> 4

Implementing Least Recently Used Algorithm:

7

0 7

1 0 7

2 1 0

2 1 0

3 2 0

3 2 0

4 3 0

2 4 0

3 2 4

0 3 2

0 3 2

0 3 2

1 3 2

1 3 2

0 1 2

0 1 2

7 0 1

7 0 1

7 0 1

No. of Page Faults on performing LRU Algorithm: 12

Main Menu

1. *Enter the Reference String*
2. *View the Reference String*
3. *Implement FIFO Algorithm*
4. *Implement LRU Algorithm*
5. *Implement Optimal Algorithm*
6. *Implement LFU Algorithm*
0. *Exit*

Your Choice -> 5

Performing Optimal Algorithm:

7

0 7

1 0 7

2 0 7

2 0 7

3 2 7

0 3 2

4 3 2

4 3 2

4 3 2

0 3 2

0 3 2

0 3 2

1 0 2

1 0 2

1 0 2

1 0 2

7 1 0

7 1 0

7 1 0

No. of Page Faults on performing Optimal Algorithm: 10

Main Menu

1. Enter the Reference String
2. View the Reference String
3. Implement FIFO Algorithm
4. Implement LRU Algorithm
5. Implement Optimal Algorithm
6. Implement LFU Algorithm
0. Exit
Your Choice -> 6

Implementing Least Frequently Used Algorithm:

7

0 7

1 0 7

2 1 0

2 1 0

3 2 0

3 2 0

4 3 0

2 4 0

3 2 0

3 2 0

3 2 0

3 2 0

1 3 0

2 3 0

2 3 0

1 2 0

7 2 0

7 2 0

1 2 0

No. of Page Faults on performing LFU Algorithm: 13

Main Menu

1. Enter the Reference String
2. View the Reference String
3. Implement FIFO Algorithm
4. Implement LRU Algorithm
5. Implement Optimal Algorithm
6. Implement LFU Algorithm
0. Exit
Your Choice -> 0

Thank You!