SSN COLLEGE OF ENGINEERING, KALAVAKKAM
(An Autonomous Institution, Affiliated to Anna University, Chennai)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UCS1411 - OPERATING SYSTEMS LAB

LAB EXERCISE 10

Implementation of Page Replacement Algorithms

Submission Date:23-05-2022

Name: Jayannthan P T *Dept: CSE 'A'* *Roll No.: 205001049*

1. Develop a C program to implement the page replacement algorithms (FIFO, Optimal, LRU and LFU) using linked list.

Algorithm:

1. Start.
2. Get user input and reference string.
3. Get user choice for the page replacement algorithm.

FIFO:

1. Create an empty linked list.
2. Get frame from reference string.
3. Search and check if the frame is already present in the list of frames.
    a) If not found.
    b) Insert into list.
    c) Increase size.
    d) Check for the oldest frame.
    e) Replace the oldest frame with the current frame.
    f) Increment the oldest frame.
4. Insert into table.
5. Increment number of faults.
6. Display table.

Optimal:

1. Create an empty linked list.
2. Search if the frame from the reference string is in the current list.
3. If not
    - If size is lesser than list size then insert and increment size.
    - Iterate through the list.
    - For each frame in the list check the next occurrence in the reference string in the future.
    - Assign and find the max distance.

- Replace the frame with greater future distance.

4. Increment the no of page faults.
5. Display table.

LRU:

1. Create an empty list.
2. Search if the frame from the reference string is in the current list.
3. If not,
    - If size is less than no of frames then insert and increment size.
    - Iterate through the list.
    - Check the previous frames and assign distance.
    - Calculate max distance for each frame.
    - Replace the frame with max distance.
4. Increment the no of page faults.
5. Display table.

LFU:

1. Create an empty list.
2. Search if the frame from the reference string is in the current list.
3. If not
    - If size is lesser than list size then insert and increment size.
    - If not, iterate through the list and increment frequency.
    - Go backwards and check frequency.
    - CHeck the least frequency with the frame.
4. If found, increment the frequency.
5. Increment the number of faults.
6. Display table.

Code:

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

struct memory
{
    char page;
    struct memory * next;
};
void init_frame(struct memory *frame)
{
    struct memory *ptr = frame->next;
    while (ptr != NULL)
    {
        ptr->page = '-';
        ptr = ptr->next;
    }
}
void insert_frame(struct memory *frame, char page)
{
    struct memory *new_frame = (struct memory *) malloc(sizeof(struct memory));
    new_frame->page = page;
```

```
    new_frame->next = NULL;
    struct memory *ptr = frame;
    while (ptr->next != NULL)
    {
        ptr = ptr->next;
    }
    ptr->next = new_frame;
}
void delete_frame(struct memory *frame)
{
    while (frame->next != NULL)
    {
        struct memory *temp = frame->next;
        frame->next = frame->next->next;
        free(temp);
    }
}
void replace(struct memory *frame, int cur_fault, char page)
{
    int count = 0;
    struct memory *ptr = frame->next;
    while (count < cur_fault && ptr != NULL)
    {
        ptr = ptr->next;
        count++;
    }
    if (ptr != NULL)
    {
        ptr->page = page;
    }
}
int present(struct memory *frame, int start, int end, char page)
{
    int count = 0;
    struct memory *ptr = frame->next;
    while (count != start)
    {
        count++;
        ptr = ptr->next;
    }
    while (count < end && ptr != NULL)
    {
        if (page == ptr->page)
        {
            return count;
        }
        count++;
        ptr = ptr->next;
    }
    return 100;
}
int present_ref(char str_ref[], int start, int end, char page)
{
    for (int i = start; i < end; i++)
```

```
    {
        if (page == str_ref[i])
        {
            return i;
        }
    }
    return 100;
}
int present_last(struct memory *frame, int start, int end, char page)
{
    int pos;
    int count = 0;
    struct memory *ptr = frame->next;
    while (count != start)
    {
        count++;
        ptr = ptr->next;
    }
    while (count <= end && ptr != NULL)
    {
        if (page == ptr->page)
        {
            pos = count;
        }
        count++;
        ptr = ptr->next;
    }
    return pos;
}
int present_last_ref(char ref_str[], int start, int end, char page)
{
    int pos;
    for (int i = start; i <= end; i++)
    {
        if (page == ref_str[i])
        {
            pos = i;
        }
    }
    return pos;
}
int max(int duration[], int page_follow[], int no_frames)
{
    int first = page_follow[0], first_pos = 0;
    for (int i = 0; i < no_frames; i++)
    {
        if (page_follow[i] > first)
        {
            first = page_follow[i];
            first_pos = i;
        }
    }
    if (page_follow[first_pos] == 100)
    {
```

```
        first = duration[first_pos];
        for (int i = 0; i < no_frames; i++)
        {
            if ((duration[i] > first) && (page_follow[i] == 100))
            {
                first = duration[i];
                first_pos = i;
            }
        }
    }
    return first_pos;
}
int min(int page_past[], int no_frames)
{
    int first = page_past[0], first_pos = 0;
    for (int i = 0; i < no_frames; i++)
    {
        if (page_past[i] < first)
        {
            first = page_past[i];
            first_pos = i;
        }
    }
    return first_pos;
}
int frequency(char ref_str[], int start, int end, char page)
{
    int freq = 0;
    for (int i = start; i <= end; i++)
    {
        if (page == ref_str[i])
        {
            freq++;
        }
    }
    return freq;
}
int min_freq(int duration[], int page_past[], int no_frames)
{
    int first = page_past[0], first_pos = 0, count = 0;
    for (int i = 0; i < no_frames; i++)
    {
        if (page_past[i] < first)
        {
            first = page_past[i];
            first_pos = i;
        }
    }
    for (int i = 0; i < no_frames; i++)
    {
        if (page_past[i] == page_past[first_pos])
        {
            count++;
        }
```

```
    }
    if (count >= 2)
    {
        first = duration[first_pos];
        int pos = 0;
        for (int i = 0; i < no_frames; i++)
        {
            if ((duration[i] > first) && (page_past[i] == page_past[pos]))
            {
                first = duration[i];
                pos = i;
            }
        }
        return pos;
    }
    return first_pos;
}
void print_frame(char page, struct memory *frame, int page_fault)
{
    printf("\n%c\t--->\t", page);
    struct memory *ptr = frame->next;
    while (ptr != NULL)
    {
        printf("%c\t", ptr->page);
        ptr = ptr->next;
    }
    if (!page_fault) {}
    else
    {
        printf("\tPage fault : %d", page_fault);
    }
}
char frame_page(struct memory *frame, int index)
{
    struct memory *ptr = frame->next;
    int count = 0;
    while (ptr != NULL && count < index)
    {
        ptr = ptr->next;
        count++;
    }
    return ptr->page;
}
void fifo(struct memory *frame, int no_frames, char ref_str[])
{
    init_frame(frame);
    int page_fault = 0, cur_fault = 0;
    for (int i = 0; ref_str[i] != '\0'; i++)
    {
        if (present(frame, 0, no_frames, ref_str[i]) == 100)
        {
            replace(frame, cur_fault, ref_str[i]);
            page_fault++;
            cur_fault = (cur_fault + 1) % no_frames;
```

```
            print_frame(ref_str[i], frame, page_fault);
        }
        else
        {
            print_frame(ref_str[i], frame, 0);
        }
    }
    printf("\nTotal page faults : %d", page_fault);
}
void optimal(struct memory *frame, int no_frames, char ref_str[])
{
    init_frame(frame);
    int duration[100];
    for (int i = 0; i < no_frames; i++)
    {
        duration[i] = 0;
    }
    int page_fault = 0, cur_fault = 0;
    for (int i = 0; ref_str[i] != '\0'; i++)
    {
        if (present(frame, 0, no_frames, '-') != 100 && present(frame, 0,
                no_frames, ref_str[i]) == 100)
        {
            cur_fault = present(frame, 0, no_frames, '-');
            replace(frame, cur_fault, ref_str[i]);
            page_fault++;
            print_frame(ref_str[i], frame, page_fault);
        }
        else if (present(frame, 0, no_frames, ref_str[i]) == 100)
        {
            int page_follow[100];
            for (int j = 0; j < no_frames; j++)
            {
                char item = frame_page(frame, j);
                page_follow[j] = present_ref(ref_str, i + 1, strlen(ref_str),
                    item);
            }
            int cur_fault = max(duration, page_follow, no_frames);
            replace(frame, cur_fault, ref_str[i]);
            page_fault++;
            for (int j = 0; j < no_frames; j++)
            {
                if (j != cur_fault)
                {
                    duration[j]++;
                }
                else
                {
                    duration[j] = 0;
                }
            }
            print_frame(ref_str[i], frame, page_fault);
        }
        else
```

```
        {
            print_frame(ref_str[i], frame, 0);
        }
    }
    printf("\nTotal page faults = %d", page_fault);
}
void lru(struct memory *frame, int no_frames, char ref_str[])
{
    init_frame(frame);
    int page_fault = 0, cur_fault = 0;
    for (int i = 0; ref_str[i] != '\0'; i++)
    {
        if (present(frame, 0, no_frames, '-') != 100 && present(frame, 0,
                no_frames, ref_str[i]) == 100)
        {
            cur_fault = present(frame, 0, no_frames, '-');
            replace(frame, cur_fault, ref_str[i]);
            page_fault++;
            print_frame(ref_str[i], frame, page_fault);
        }
        else if (present(frame, 0, no_frames, ref_str[i]) == 100)
        {
            int page_follow[100];
            for (int j = 0; j < no_frames; j++)
            {
                char item = frame_page(frame, j);
                page_follow[j] = present_last_ref(ref_str, 0, i - 1, item);
            }
            int cur_fault = min(page_follow, no_frames);
            replace(frame, cur_fault, ref_str[i]);
            page_fault++;
            print_frame(ref_str[i], frame, page_fault);
        }
        else
        {
            print_frame(ref_str[i], frame, 0);
        }
    }
    printf("\nTotal page faults = %d", page_fault);
}
void lfu(struct memory *frame, int no_frames, char ref_str[])
{
    init_frame(frame);
    int duration[100];
    for (int i = 0; i < no_frames; i++)
    {
        duration[i] = 0;
    }
    int page_fault = 0, cur_fault = 0;
    for (int i = 0; ref_str[i] != '\0'; i++)
    {
        if (present(frame, 0, no_frames, '-') != 100 && present(frame, 0,
                no_frames, ref_str[i]) == 100)
        {
```

```
            cur_fault = present(frame, 0, no_frames, '-');
            replace(frame, cur_fault, ref_str[i]);
            page_fault++;
            print_frame(ref_str[i], frame, page_fault);
        }
        else if (present(frame, 0, no_frames, ref_str[i]) == 100)
        {
            int page_past[100]; //Finding frequency of page usage
            for (int j = 0; j < no_frames; j++)
            {
                char item = frame_page(frame, j);
                page_past[j] = frequency(ref_str, 0, i - 1, item);
            }
            int cur_fault = min_freq(duration, page_past, no_frames);
            replace(frame, cur_fault, ref_str[i]);
            page_fault++;
            for (int j = 0; j < no_frames; j++)
            {
                if (j != cur_fault)
                {
                    duration[j]++;
                }
                else
                {
                    duration[j] = 0;
                }
            }
            print_frame(ref_str[i], frame, page_fault);
        }
        else
        {
            print_frame(ref_str[i], frame, 0);
        }
    }
    printf("\nTotal page faults = %d", page_fault);
}
int main()
{
    struct memory *frame = (struct memory *) malloc(sizeof(struct memory));
    int no_frames, ref_len;
    char frames[100], ref_str[100];

    delete_frame(frame);
    printf("Number of frames: ");
    scanf(" %d", &no_frames);
    for (int i = 0; i < no_frames; i++)
    {
        insert_frame(frame, '-');
    }
    printf("\nReference string length: ");
    scanf(" %d", &ref_len);
    printf("\nReference string: ");
    for (int i = 0; i < ref_len; i++)
    {
```

```
        scanf(" %c", &ref_str[i]);
    }
    ref_str[ref_len] = '\0';

    while (1)
    {
        printf("\n1. FIFO\n2. Optimal\n3. LRU\n4. LFU\n5.Exit\nEnter your choice: ");
        int ch;
        scanf(" %d", &ch);
        switch (ch)
        {
            case 1:
                {
                    fifo(frame, no_frames, ref_str);
                    break;
                }
            case 2:
                {
                    optimal(frame, no_frames, ref_str);
                    break;
                }
            case 3:
                {
                    lru(frame, no_frames, ref_str);
                    break;
                }
            case 4:
                {
                    lfu(frame, no_frames, ref_str);
                    break;
                }
            case 5:
                {
                    exit(1);
                    break;
                }
            default:
                {
                    printf("\nInvalid choice!");
                    break;
                }
        }
    }
    printf("\n");
    return 0;
}
```

Output:

```
1. FIFO
2. Optimal
3. LRU
4. LFU
5.Exit
Enter your choice: 1

7       --->    7       -       -               Page fault : 1
0       --->    7       0       -               Page fault : 2
1       --->    7       0       1               Page fault : 3
2       --->    2       0       1               Page fault : 4
0       --->    2       0       1
3       --->    2       3       1               Page fault : 5
0       --->    2       3       0               Page fault : 6
4       --->    4       3       0               Page fault : 7
2       --->    4       2       0               Page fault : 8
3       --->    4       2       3               Page fault : 9
0       --->    0       2       3               Page fault : 10
3       --->    0       2       3
2       --->    0       2       3
1       --->    0       1       3               Page fault : 11
2       --->    0       1       2               Page fault : 12
0       --->    0       1       2
1       --->    0       1       2
7       --->    7       1       2               Page fault : 13
0       --->    7       0       2               Page fault : 14
1       --->    7       0       1               Page fault : 15
Total page faults : 15
```

```
Enter your choice: 2

7       --->    7       -       -               Page fault : 1
0       --->    7       0       -               Page fault : 2
1       --->    7       0       1               Page fault : 3
2       --->    2       0       1               Page fault : 4
0       --->    2       0       1
3       --->    2       0       3               Page fault : 5
0       --->    2       0       3
4       --->    2       4       3               Page fault : 6
2       --->    2       4       3
3       --->    2       4       3
0       --->    2       0       3               Page fault : 7
3       --->    2       0       3
2       --->    2       0       3
1       --->    2       0       1               Page fault : 8
2       --->    2       0       1
0       --->    2       0       1
1       --->    2       0       1
7       --->    7       0       1               Page fault : 9
0       --->    7       0       1
1       --->    7       0       1
Total page faults = 9
```

```
Enter your choice: 3

7       --->    7       -       -               Page fault : 1
0       --->    7       0       -               Page fault : 2
1       --->    7       0       1               Page fault : 3
2       --->    2       0       1               Page fault : 4
0       --->    2       0       1
3       --->    2       0       3               Page fault : 5
0       --->    2       0       3
4       --->    4       0       3               Page fault : 6
2       --->    4       0       2               Page fault : 7
3       --->    4       3       2               Page fault : 8
0       --->    0       3       2               Page fault : 9
3       --->    0       3       2
2       --->    0       3       2
1       --->    1       3       2               Page fault : 10
2       --->    1       3       2
0       --->    1       0       2               Page fault : 11
1       --->    1       0       2
7       --->    1       0       7               Page fault : 12
0       --->    1       0       7
1       --->    1       0       7
Total page faults = 12
```

```
Enter your choice: 4

7       --->    7       -       -               Page fault : 1
0       --->    7       0       -               Page fault : 2
1       --->    7       0       1               Page fault : 3
2       --->    2       0       1               Page fault : 4
0       --->    2       0       1
3       --->    2       0       3               Page fault : 5
0       --->    2       0       3
4       --->    4       0       3               Page fault : 6
2       --->    4       0       2               Page fault : 7
3       --->    3       0       2               Page fault : 8
0       --->    3       0       2
3       --->    3       0       2
2       --->    3       0       2
1       --->    3       0       1               Page fault : 9
2       --->    3       0       2               Page fault : 10
0       --->    3       0       2
1       --->    1       0       2               Page fault : 11
7       --->    7       0       2               Page fault : 12
0       --->    7       0       2
1       --->    1       0       2               Page fault : 13
Total page faults = 13
```

Learning Outcome:

- Learnt to implement page replacement techniques