****

**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.
   * + 1. If not found.
       2. Insert into list.
       3. Increase size.
       4. Check for the oldest frame.
       5. Replace the oldest frame with the current frame.
       6. 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:**

#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;

                }

        }

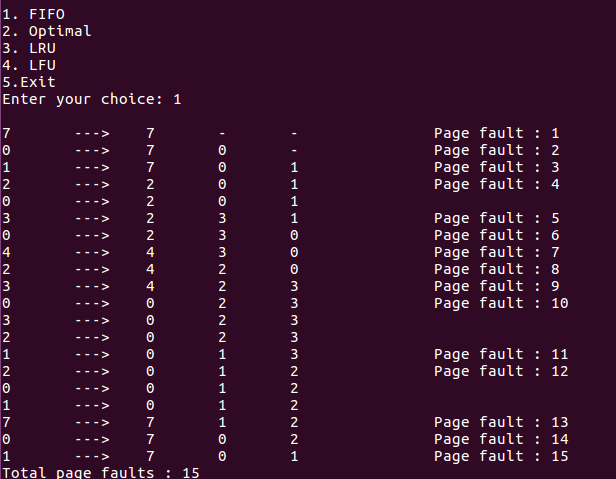
    }

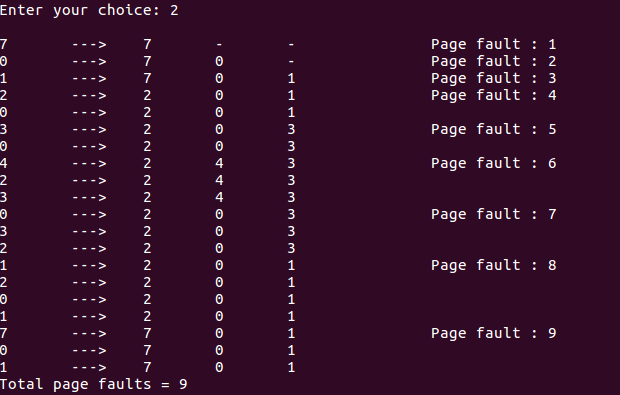
    printf("\n");

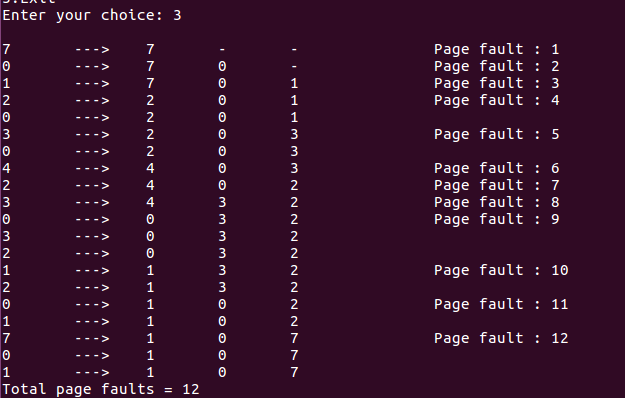
    return 0;

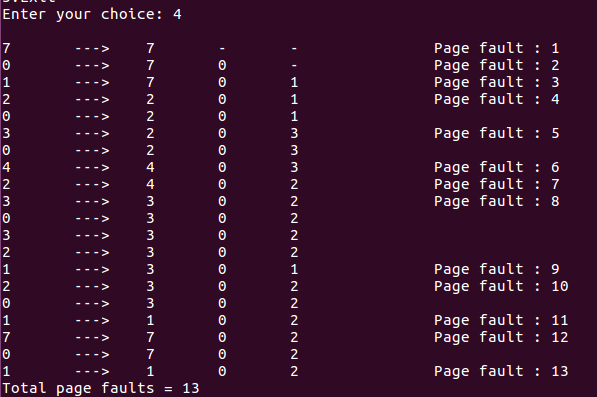
}

**Output:**









**Learning Outcome:**

* Learnt to implement page replacement techniques