East West University

Department of Computer Science and Engineering

Lab Experiment: Memory management

Objective: The objective of this lab is to implement various page replacement algorithms (FIFO, LRU, LFU) and simulate paging techniques for memory management. Through these experiments, students will understand the functionality, implementation, and comparison of different page replacement strategies and paging mechanisms in operating systems.

PAGE REPLACEMENT ALGORITHMS

A) FIRST IN FIRST OUT (FIFO):

AIM: To implement FIFO page replacement technique.

DESCRIPTION: ► The FIFO page-replacement algorithm is easy to understand and program. However, its performance is not always good. ► On the one hand, the page replaced may be an initialization module that was used a long time ago and is no longer needed. ► On the other hand, it could contain a heavily used variable that was initialized early and is in constant use.

ALGORITHM:

1. Start the process.
2. Read number of pages `n`.
3. Read number of frames no.
4. Read page numbers into an `array a[i]`.
5. Initialize `avail[i]=0` to check page hits.
6. Replace the page with a circular queue; while replacing, check page availability in the frame. Place `avail[i]=1` if the page is placed in the frame. Count page faults.
7. Print the results.
8. Stop the process.

SOURCE CODE:

```
#include<stdio.h>

#include<conio.h>

int fr[3];

void main()

{

    void display();

    int i, j, page[12] = {2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2};
int flag1 = 0, flag2 = 0, pf = 0, frsize = 3, top = 0;

clrscr();
for(i = 0; i < 3; i++)
{
    fr[i] = -1;
}
for(j = 0; j < 12; j++)
{
    flag1 = 0; flag2 = 0;
    for(i = 0; i < frsize; i++)
    {
        if(fr[i] == page[j])
        {
            flag1 = 1; flag2 = 1;
            break;
        }
    }
    if(flag1 == 0)
    {
        for(i = 0; i < frsize; i++)
        {
            if(fr[i] == -1)
            {
                fr[i] = page[j];
                flag2 = 1;
                break;
            }
        }
    }
    if(flag2 == 0)
    {
            fr[top] = page[j];
            top++;
            pf++;
            if(top >= frsize)
                top = 0;
        }
        display();
    }
    printf("Number of page faults: %d ", pf);
    getch();
}
void display()
{
    int i;
    printf("\n");
    for(i = 0; i < 3; i++)
        printf("%d\t", fr[i]);
}
```

OUTPUT:

```
2 -1 -1
2 3 -1
2 3 -1
2 3 1
5 3 1
5 2 1
5 2 4
5 2 4
3 2 4
3 2 4
3 5 4
3 5 2
Number of page faults: 6
```

B) LEAST RECENTLY USED (LRU):

AIM: To implement LRU page replacement technique.

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 to the selection.
8. Display the values.
9. Stop the process.

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
int fr[3];

void main()
{
    void display();
    int p[12] = {2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2}, i, j, fs[3];
    int index, k, l, flag1 = 0, flag2 = 0, pf = 0, frsize = 3;
    clrscr();
    for(i = 0; i < 3; i++)
    {
        fr[i] = -1;
    }
    for(j = 0; j < 12; j++)
    {
        flag1 = 0; flag2 = 0;
        for(i = 0; i < frsize; i++)
        {
```

```
    if(fr[i] == p[j])
    {
        flag1 = 1; flag2 = 1;
        break;
    }
}
if(flag1 == 0)
{
    for(i = 0; i < frsize; i++)
    {
        if(fr[i] == -1)
        {
            fr[i] = p[j];
            flag2 = 1;
            break;
        }
    }
}
if(flag2 == 0)
{
    for(i = 0; i < frsize; i++)
        fs[i] = 0;
    for(k = j - 1, l = 1; l <= frsize - 1; l++, k--)
    {
        for(i = 0; i < frsize; i++)
        {
            if(fr[i] == p[k])
                fs[i] = l;
        }
    }
    for(i = 0; i < frsize; i++)
    {
```

```
                if(fs[i] == 0)
                    index = i;
            }
            fr[index] = p[j];
            pf++;
        }
        display();
    }
    printf("\nNumber of page faults: %d", pf);
    getch();
}
void display()
{
    int i;
    printf("\n");
    for(i = 0; i < 3; i++)
        printf("%d\t", fr[i]);
}
```

OUTPUT:

```
2 -1 -1
2 3 -1
2 3 -1
2 3 1
2 5 1
2 5 1
2 5 4
2 5 4
3 5 4
3 5 2
3 5 2
3 5 2
Number of page faults: 4
```

C) LEAST FREQUENTLY USED (LFU):

AIM: To implement LFU page replacement technique.

ALGORITHM:

1. Start Program.
2. Read Number Of Pages And Frames.
3. Read Each Page Value.
4. Search For Page In The Frames.
5. If Not Available Allocate Free Frame.
6. If No Frames Is Free, Replace The Page With The Page That Is Least Used.
7. Print Number Of Page Faults.
8. Stop process.

SOURCE CODE:

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

int fr[10], n, m;

void display();

void main()
{
    int i, j, page[20], fs[10];
    int max, found = 0, index, k, l, flag1 = 0, flag2 = 0, pf = 0;

    clrscr();

    printf("Enter length of the reference string: ");
    scanf("%d", &n);

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

```
printf("Enter number of frames: ");
scanf("%d", &m);

// Initialize all frames to -1
for(i = 0; i < m; i++)
    fr[i] = -1;

for(j = 0; j < n; j++)
{
    flag1 = 0;
    flag2 = 0;

    // Check if the page is already in the frame
    for(i = 0; i < m; i++)
    {
        if(fr[i] == page[j])
        {
            flag1 = 1;
            flag2 = 1;
            break;
        }
    }

    // If the page is not found and there's an empty frame, insert it
    if(flag1 == 0)
    {
        for(i = 0; i < m; i++)
        {
            if(fr[i] == -1)
            {
                fr[i] = page[j];
                flag2 = 1;
                break;
            }
        }
    }

    // If no empty frame is found, apply the LFU replacement policy
    if(flag2 == 0)
    {
        // Reset frequency counter
        for(i = 0; i < m; i++)
            fs[i] = 0;

        // Count frequency of each page in the frame up to the current
        for(k = 0; k <= j; k++)
        {
            for(i = 0; i < m; i++)
            {
                if(fr[i] == page[k])
                    fs[i]++;
            }
        }

        // Find the page with the lowest frequency to replace
        max = fs[0];
        index = 0;
        for(i = 1; i < m; i++)
        {
            if(fs[i] < max)
            {
                max = fs[i];
                index = i;
                }
            }

            // Replace the least frequently used page
            fr[index] = page[j];
            pf++; // Increment the page fault count
        }

        // Display the frame content
        display();
    }

    printf("\nNumber of page faults: %d", pf);
    getch();
}

void display()
{
    int i;
    printf("\n");
    for(i = 0; i < m; i++)
    {
        if(fr[i] == -1)
            printf("-\t");
        else
            printf("%d\t", fr[i]);
    }
}  {
        flag1 = 0; flag2 = 0;
        for(i = 0; i < m; i++)
        {
            if(fr[i] == page[j])
        {
            flag1 = 1;
            flag2 = 1;
            break;
        }
    }
    if(flag1 == 0)
    {
        for(i = 0; i < m; i++)
        {
            if(fr[i] == -1)
            {
                fr[i] = page[j];
                flag2 = 1;
                break;
            }
        }
    }
    if(flag2 == 0)
    {
        for(i = 0; i < m; i++)
            fs[i] = 0;

        for(k = 0; k <= j; k++)
        {
            for(i = 0; i < m; i++)
            {
                if(fr[i] == page[k])
                    fs[i]++;
            }
        }
```

```
            max = fs[0];
            index = 0;
            for(i = 1; i < m; i++)
            {
                if(fs[i] < max)
                {
                    max = fs[i];
                    index = i;
                }
            }
            fr[index] = page[j];
            pf++;
        }
        display();
    }
    printf("\nNumber of page faults: %d", pf);
    getch();
}

void display()
{
    int i;
    printf("\n");
    for(i = 0; i < m; i++)
        printf("%d\t", fr[i]);
}
```

Output:

```
Enter length of the reference string: 12
Enter the reference string: 2 3 2 1 5 2 4 5 3 2 5 2
Enter number of frames: 3
2 -1 -1
```

```
2 3 -1

2 3 -1

2 3 1

5 3 1

5 3 1

5 2 1

5 2 4

5 2 4

3 2 4

3 5 4

3 5 2

Number of page faults: 8
```

PAGING

AIM: Simulate Paging technique for memory management.

ALGORITHM:

Step 1: Read all the necessary input from the keyboard. Step 2: Pages - Logical memory is broken into fixed-sized blocks. Step 3: Frames - Physical memory is broken into fixed-sized blocks. Step 4: Calculate the physical address using the following: `Physical address = ( Frame number * Frame size ) + offset`. Step 5: Display the physical address. Step 6: Stop the process.

SOURCE CODE:

/* Memory Allocation with Paging Technique */

```
#include <stdio.h>

#include <conio.h>


struct pstruct {

    int fno;

    int pbit;

} ptable[10];


int pmsize, lmsize, psize, frame, page, ftable[20], frameno;


void info() {

    printf("\n\nMEMORY MANAGEMENT USING PAGING\n\n");
    printf("\n\nEnter the Size of Physical memory: ");
    scanf("%d", &pmsize);
    printf("\n\nEnter the size of Logical memory: ");
    scanf("%d", &lmsize);
    printf("\n\nEnter the partition size: ");
    scanf("%d", &psize);

    frame = (int)pmsize / psize;
    page = (int)lmsize / psize;

    printf("\nThe physical memory is divided into %d no. of frames\n",
frame);
    printf("\nThe Logical memory is divided into %d no. of pages\n", page);
}

void assign() {
    int i;
    for (i = 0; i < page; i++) {
        ptable[i].fno = -1;
        ptable[i].pbit = -1;
    }
    for (i = 0; i < frame; i++)
        ftable[i] = 32555;
    for (i = 0; i < page; i++) {
        printf("\n\nEnter the Frame number where page %d must be placed: ",
i);
        scanf("%d", &frameno);
        ftable[frameno] = i;
        if (ptable[i].pbit == -1) {
            ptable[i].fno = frameno;
            ptable[i].pbit = 1;
        }
    }
    getch();

    clrscr();

    printf("\n\nPAGE TABLE\n\n");

    printf("PageAddress\tFrameNo.\tPresenceBit\n\n");

    for (i = 0; i < page; i++)

        printf("%d\t\t%d\t\t%d\n", i, ptable[i].fno, ptable[i].pbit);

    printf("\n\n\n\tFRAME TABLE\n\n");

    printf("FrameAddress\tPageNo\n\n");

    for (i = 0; i < frame; i++)

        printf("%d\t\t%d\n", i, ftable[i]);

}


void cphyaddr() {

    int laddr, paddr, disp, phyaddr, baddr;

    getch();

    clrscr();

    printf("\n\n\n\tProcess to create the Physical Address\n\n");

    printf("\nEnter the Base Address: ");

    scanf("%d", &baddr);

    printf("\nEnter the Logical Address: ");

    scanf("%d", &laddr);

    paddr = laddr / psize;

    disp = laddr % psize;

    if (ptable[paddr].pbit == 1)

        phyaddr = baddr + (ptable[paddr].fno * psize) + disp;

    printf("\nThe Physical Address where the instruction is present: %d",
phyaddr);

}


void main() {

    clrscr();

    info();
    assign();
    cphyaddr();
    getch();
}
```

OUTPUT:

```
MEMORY MANAGEMENT USING PAGING

Enter the Size of Physical memory: 16
Enter the size of Logical memory: 8
Enter the partition size: 2

The physical memory is divided into 8 no. of frames
The Logical memory is divided into 4 no. of pages

Enter the Frame number where page 0 must be placed: 5
Enter the Frame number where page 1 must be placed: 6
Enter the Frame number where page 2 must be placed: 7
Enter the Frame number where page 3 must be placed: 2

PAGE TABLE
PageAddress     FrameNo.        PresenceBit
0               5               1
1               6               1
2               7               1
3               2               1

FRAME TABLE
FrameAddress    PageNo
0               32555
1               32555
2               3
```

```
3                       32555

4                       32555

5                       0

6                       1

7                       2


Process to create the Physical Address

Enter the Base Address: 1000

Enter the Logical Address: 3


The Physical Address where the instruction is present: 1013
```