**Name : Ansari Aatif Akhlaque**

**Roll No : 25**

**Subject : OSL**

**Assignment No. 6:**

Implement the C program for Page Replacement Algorithms: FCFS, LRU, and Optimal for frame size as minimum three.

**Assignment Statement:**

Write a menu driven program to simulate the following page replacement algorithms.

· First In First Out (FIFO).

· Least Recently Used (LRU).

**Objectives:**

To understand various concepts such as

▪Memory management

▪Paging

▪Segmentation

▪Virtual memory

▪Page Replacement Strategies

**Theory:**

One of the most important and complex tasks of an operating system is memory management. Memory management involves treating main memory as a resource to be allocated to and shared among a number of active processes. To use the processor and the I/O facilities efficiently, it is desirable to maintain as many processes in main memory as possible. In addition, it is desirable to free users from size restrictions in program development.

Relocation, Protection, Sharing, Logical Organization and physical Organization are the requirements that memory management is intended to satisfy.

**Description of Memory management Techniques**

**Fixed Partitioning**

**Description:** Main memory is divided into a no .of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.

**Strengths:** Simple to implement, little operating system overhead

**Weakness**: Inefficient use of memory due to internal fragmentation, Maximum number of active processes is fixed.

**Dynamic Partitioning:**

**Description:** Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as that process**.**

**Strengths:** No internal Fragmentation, more efficient use of main memory

**Weaknesses:** Inefficient use of processor due to the need for compaction to counter external fragmentation

**Simple Paging**

**Description:** Main memory is divided into a no. of equal size frames. Each process is divided into a no. of equal size pages of the same length of frames. A process is loaded by loading all of its pages into available, not necessarily contiguous, frames

**Strengths:** No external fragmentation

**Weaknesses:** A small amount of internal fragmentation

**Simple Segmentation:**

**Description:** Each process is divided into no. of segments. A process is loaded by loading all of its segments into dynamic partitions that need not be contiguous.

**Strengths:** No internal fragmentation

**Weaknesses:** Improved memory utilization and reduced overhead compared to dynamic partitioning.

**Virtual Memory Paging**

**Description:** As with simple paging, except that it is not necessary to load all of the pages of a process. Nonresident pages that are needed are brought in later automatically.

**Strengths**: No external fragmentation, higher degree of multiprogramming, large virtual address space.

**Weaknesses:** Overhead of complex memory management

**Virtual Memory Segmentation**

**Description**: As with simple segmentation except that it is not necessary to load all of the segments of a process. Nonresident segments that are needed are brought in later automatically.

**Strengths:** No internal fragmentation, higher degree of multiprogramming, large virtual address space, protection and sharing support

**Weaknesses**: Overhead of complex memory management

**Key Terms**

**Internal Fragmentation**: In case of fixed partition technique, there is wasted space internal to a partition due to the fact that the block of data loaded is smaller than the partition. This is known as internal fragmentation.

**External Fragmentation**: In case of dynamic partitions, method leads to a situation in which there are a lot of small holes in a memory. As time goes on, memory becomes more and more fragmented, and memory utilization declines. This phenomenon is referred to as external fragmentation. This is due to the fact that memory that is external to all partitions becomes increasingly fragmented.

**Compaction:** This is a technique used to overcome external fragmentation. From time to time, the operating system shifts the processes so that they are contiguous so that all of the free memory is together in one block. This is a time consuming procedure and wasteful of processor time. It also implies that need for dynamic relocation capability should exist.

**Swapping**: At intervals determined by the operating system, usually dictated by CPU scheduling policies, processes are copied from main memory to a backing store and later are copied back to main memory. This allows more processes to be run than can be fit into memory at one time.

**Logical Address**: It is a reference to a memory location independent of current assignment of data to memory.

**Relative Address**: It is a particular example of logical address, in which the address is expressed as a location relative to some known point, usually the beginning of the program.

**Physical Address**: It is an actual address in main mempry.

**Frame**:: The main memory is divided into equal fixed size chunks that are relatively small. Chunks of main memory are called as frames.

**Page**: Each process is divided into small size chunks of the same size. Chunks of process are known as pages.

**Segmentation**: A process is divided into a number of segments which need not be of equal size. With simple segmentation a process is brought in means all segments are loaded into available regions of memory and segment table is set up.

**Virtual Memory**: It is a technique that allows the execution of processes that may not be completely in memory. The programs can be larger than physical memory. Virtual memory

abstracts main memory into extremely large, uniform array of storage, separating logical memory as viewed by the user from physical memory. This technique frees programmers from the concerns of memory storage limitations. This also allows user processes to easily share files and address spaces, and it provides efficient mechanism for process creation.

**Page fault**: With virtual memory only few pages of the entire process are present in the main memory. If the program branches to an instruction or references a data item on a piece not in the main memory, page fault is triggered. This tells the operating system to bring the desired page in the memory.

**Thrashing**: If all of the main memory is occupied with pages of processes, and if a page fault occurs, then operating system must throw one page out main memory and bring the desired page in. Sometimes it is required to bring in the page which was just thrown out. Thrashing is the process of bringing in the pages just thrown out again and again.

**Principle of locality**: It states that program and data references within a process tend to cluster.

**Demand Paging**: It is a fetch policy in which page is brought into memory only when a reference is made to a location on that page.

**Replacement Policy**: This deals with the selection pf a page in memory to be replaced when a new page is to be brought in.

**Frame Locking:** Some of the frames in main memory may be locked. When a frame is locked, the page currently residing in that frame may not be used for replacement. Key control structures, i/o buffers or time critical areas are examples of locked frames.

**Page Replacement Algorithms**: There are certain basic algorithms that are used for the selection of a page to replace. These are **Optimal**, Least Recently Used(**LRU**), First In First Out(**FIFO**), **Clock**

**Brief description of algorithms.**

•**Optimal Page replacement:** This policy selects for replacement that page for which the time to next reference is the longest. This algorithm results into fewest number of page faults.

Limitations: Impossible to implement, since operating system must have a prior knowledge of future events, that is which page will be or will not be referenced in future.

•**Least Recently Used (LRU)**: This policy replaces the page in memory that has not been referenced for the longest time.

It is difficult to implement. One can have a tag with each page with the time of its last reference and it should be done at each memory reference

•**First In first Out(FIFO):** It treats the page frames allocated to a process as a circular buffer, and pages are removed in round robin manner. A pointer is required that circles through page frames of the process.

This is the simplest algorithm to implement and one is replacing that page which is there in the memory for the longest period. Number of page faults could be more as compared to Optimal and LRU.

**Belady’s Anomaly**: For some page replacement algorithms, the page fault rate may increase as the number of allocated frames increases. Giving more memory to a process would not improve the performance. This is known as Belady’s Anomaly.

**Platform: Linux**

**Programming Language** : C/C++

**Code :**

**1-FIFO**

#include<stdio.h>

int main()

{

int i,j,n,a[50],frame[10],no,k,avail,count=0;

printf("\n ENTER THE NUMBER OF PAGES:\n");

scanf("%d",&n);

printf("\n ENTER THE PAGE NUMBER :\n");

for(i=1;i<=n;i++)

scanf("%d",&a[i]);

printf("\n ENTER THE NUMBER OF FRAMES :");

scanf("%d",&no);

for(i=0;i<no;i++)

frame[i]= -1;

j=0;

printf("\tref string\t page frames\n");

for(i=1;i<=n;i++)

{

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

avail=0;

for(k=0;k<no;k++)

if(frame[k]==a[i])

avail=1;

if (avail==0)

{

frame[j]=a[i];

j=(j+1)%no;

count++;

for(k=0;k<no;k++)

printf("%d\t",frame[k]);

}

printf("\n");

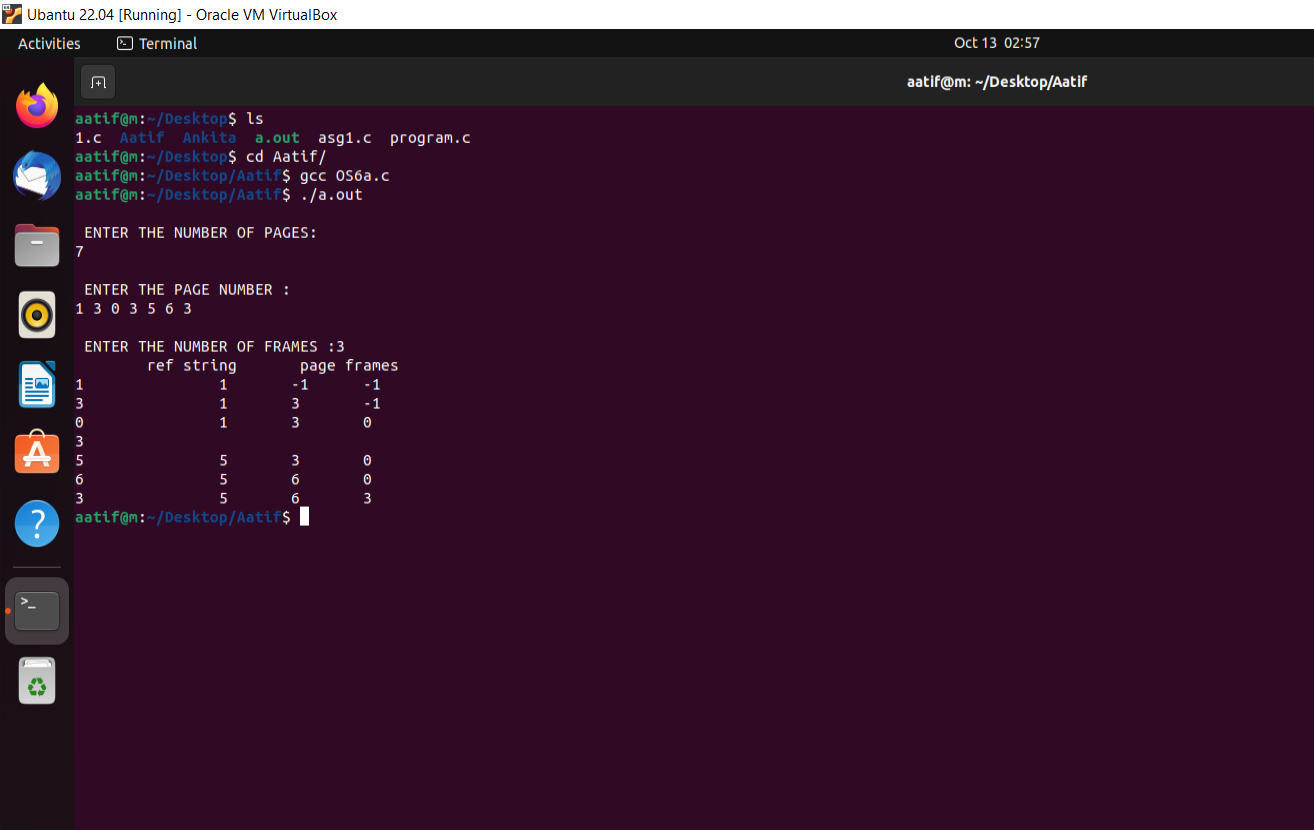
}

printf("Page Fault Is %d",count);

return 0;

}

**OUTPUT:**



**2-LRU**

#include<stdio.h>

**int** findLRU(**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** no\_of\_frames, no\_of\_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j, pos, faults = 0;

**printf**("Enter number of frames: ");

scanf("%d", &no\_of\_frames);

**printf**("Enter number of pages: ");

scanf("%d", &no\_of\_pages);

**printf**("Enter reference string: ");

**for**(i = 0; i < no\_of\_pages; ++i){

scanf("%d", &pages[i]);

}

**for**(i = 0; i < no\_of\_frames; ++i){

frames[i] = -1;

}

**for**(i = 0; i < no\_of\_pages; ++i){

flag1 = flag2 = 0;

**for**(j = 0; j < no\_of\_frames; ++j){

**if**(frames[j] == pages[i]){

counter++;

time[j] = counter;

flag1 = flag2 = 1;

**break**;

}

}

**if**(flag1 == 0){

**for**(j = 0; j < no\_of\_frames; ++j){

**if**(frames[j] == -1){

counter++;

faults++;

frames[j] = pages[i];

time[j] = counter;

flag2 = 1;

**break**;

}

}

}

**if**(flag2 == 0){

pos = findLRU(time, no\_of\_frames);

counter++;

faults++;

frames[pos] = pages[i];

time[pos] = counter;

}

**printf**("\n");

**for**(j = 0; j < no\_of\_frames; ++j){

**printf**("%d\t", frames[j]);

}

}

**printf**("\n\nTotal Page Faults = %d", faults);

**return** 0;

}

**OUTPUT :**

