**PAGE REPLACEMENT ALGORITHM**

**FIFO**

**EXPT NO: 6 DATE: 5/12/2022**

**AIM:** To implement page replacement algorithm using FIFO

As studied in Demand Paging, only certain pages of a process are loaded initially into the memory. This allows us to get more processes into memory at the same time. but what happens when a process requests for more pages and no free memory is available to bring them in. Following steps can be taken to deal with this problem:

Put the process in the wait queue, until any other process finishes its execution thereby freeing frames. Or, remove some other process completely from the memory to free frames. Or, find some pages that are not being used right now, move them to the disk to get free frames. This technique is called Page replacement and is most commonly used.

In Virtual Memory Management, Page Replacement Algorithms play an important role. The main objective of all the Page replacement policies is to decrease the maximum number of page faults.

Page Fault – It is basically a memory error, and it occurs when the current programs attempt to access the memory page for mapping into virtual address space, but it is unable to load into the physical memory then this is referred to as Page fault.

FIFO which is also known as First In First Out is one of the types of page replacement algorithm. The FIFO algorithm is used in the paging method for memory management in an operating system that decides which existing page needs to be replaced in the queue. FIFO algorithm replaces the oldest (First) page which has been present for the longest time in the main memory bringing a new page from the secondary memory.

In simple words, When a new page comes in from secondary memory to main memory, It selects the front of the queue which is the oldest page present, and removes it. FIFO is the simplest among all algorithms which are responsible for maintaining all the pages in a queue for an operating system and also keeping track of all the pages in a queue.

The older pages are kept in the front and the newer ones are kept at the end of the queue. Pages that are in the front are removed first and the pages which are demanded are added. Why do we need to swap the pages?: Since we have a fixed number of frames and all the processes cannot be stored in the main memory at a single time hence we use page replacement algorithms to store pages of a process instead of the whole process.

**FIFO Page Replacement Algorithm:**

1. Start traversing the pages.
2. Now declare the size w.r.t length of the Page.
3. Check need of the replacement from the page to memory.
4. Similarly, Check the need of the replacement from the old page to new page in memory.
5. Now form the queue to hold all pages.
6. Insert Require page memory into the queue.
7. Check bad replacemets and page faults.
8. Get no of processes to be inserted.
9. Show the values.
10. Stop

**Advantage**:

1. FIFO page replacement algorithm is commonly known for its simplicity.
2. FIFO algorithm is much easy to implement as well as understand.
3. Small systems can use the FIFO algorithm efficiently.

**Disadvantage**:

1. FIFO algorithm in operating system uses an additional Queue data structure.
2. It suffers from Belady's anomaly problem i.e when the number of page frames increases, more memory is given to processes, but instead of decreasing, the number of page faults increases (Belady's anomaly can be prevented using stackbased algorithms like LRU).

**CODE**

#include<iostream>

using namespace std;

#define MAX\_PROCESSES 20

#define HIT H++

#define FAULT F++

int FRAMES,N,P[MAX\_PROCESSES];

int FRAME\_box[MAX\_PROCESSES];

int H=0,F=0;

void RESET\_ALLO()

{

for(int i=0; i<FRAMES;i++)

{FRAME\_box[i]=-1;}

}

void PRINT\_FRAME()

{

for(int i=0; i<FRAMES;i++)

{cout.width(3);cout<<FRAME\_box[i]<<" ";}

cout<<endl;

}

void FIFO()

{

RESET\_ALLO();

int INC=0;

for(int i=0;i<N;i++)

{

int FLAG=0;

for(int j=0;j<FRAMES;j++)

{

if(P[i]==FRAME\_box[j])

{

HIT;

FLAG=1;

}

}

if(FLAG!=1)

{

FRAME\_box[INC]=P[i];

INC=(INC+1)%FRAMES;

FAULT;

}

PRINT\_FRAME();

}

cout<<"\nNUMBER OF FAULTS: "<<F;

cout<<"\nNUMBER OF HITS: "<<H;

}

int main()

{

cout<<"ENTER NUMBER OF FRAMES: ";

cin>>FRAMES;

cout<<"ENTER NUMBER OF PROCESSES:";

cin>>N;

cout<<"ENTER PROCESSES: ";

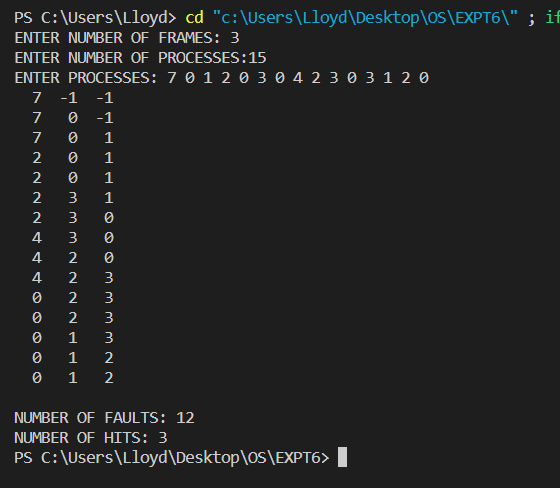
for(int i=0;i<N;i++)

{cin>>P[i];}

FIFO();

}

**OUTPUT**

****

**CONCLUSION**

The problem statement was successfully compiled and executed.