Experiment-7

Page Replacement Algorithm

Arm (1)

Deepesh Tank.
18-7-5017

DATE / /
PAGE

Aim:-

Write a C program to stimulate the following page replacement algorithm

a) FIFO
b) Optimal Page Replacement
c) LRU.

Theory:-

a) FIFO:-

This is the simplest page replacement technique in this OS keep tracks of all pages in the memory in a queue oldest page is in the front of queue. when a page is needed to be replaced page in front of queue is selected for removal

algorithm for Fifo:-

1) - Start traveling the pages
   i) if set hold less pages than capacity
      a) insert page into set one by one until size of set reaches capacity or all page request are proceed
      b) simultaneously maintain all pages in the queue to perform fifo
      c) increment page fault
   ii) else
      if current page is present in set, do nothing
      Else
      a) Remove the first page from the queue as it was first to be entered in the memory
      b) Replace the first page in the queue with current page in the string
      c) store current page in queue
      d) increment page fault
2) return page fault

(b) Optimal Page replacement :-

in OS, whenever this algorithm OS will replace the page that will not be used for the longest period of time in future.

algorithm :-

for every reference we do following :-

1) if refered page is already present, increment hit count
2) if not present find if page that is never reference in future if such a page exsist replace this page with new page if no such page exsist find the page that is reference farthest in future replace this page with new page.

c) LRU :-

Least Recently used (LRU) Algorithm is a greedy algorithm ~~exceeded~~ where the page to be replaced is least recently used The idea is based on locality of reference.

algorithm :-

let capacity be the number of pages that memory can hold. let set be the current set of pages in memory.

1) start traversing the pages
   i) if set holds less pages than capacity.
      a) insert pages into the set, ~~do nothing~~ one by one
   ~~Else~~ until the size of set reaches capacity or all page request are fulfilled.
      b) simultaneously maintain the recent occurred index of each page in a map called indexes.
      c) increment page fault.
   ii) Else.
      if current page is present in set, do nothing

Else

a) Find the page in the set that was least recently used we find it using index array. we basically need to replace the page with minimum index.

b) Replace the found page with current page.

c) Increment page faults.

d) Update index of current page.

Program code:-

A) FIFO:-

```
#include<bits/stdc++.h>
using namespace std;
int pagefaults(int pages[], int n, int capacity){
unordered_set<int> s;
queue<int> indexes; int pagefaults = 0;
for(int i=0;i<n;i++){
if(s.size()<capacity){ if(s.find(pages[i])==s.end()){
s.insert(pages[i]);
page_faults++;
indexes.push(pages[i]); }}
else{ if(s.find(pages[i])==s.end()){
int val=indexes.front();
indexes.pop();
s.erase(val);
s.insert(pages[i]);
indexes.push(pages[i]);
page_faults++; }}
return page_faults;
}
int main(){
int pages[]={7,0,1,2,0,3,0,4,2,3,0,3,2};
int n = sizeof(pages)/sizeof(pages[0]);
int capacity = 4;
cout<<pagefaults(pages,n,capacity);
return 0;
}
```

Output-7

b) Optimal page replacement algorithm

```
#include<bits/stdc++.h>
using namespace std;
using namespace std:
bool search(int key, vector<int>& fr){
for(int i=0;i<fr.size();i++) if(fr[i]==key) return true;
return false; }.
int predict(int pg[], vector<int>& fr, int pn, int index){
int res=-1, farthest=index;
for(int i=0; i<fr.size(); i++){ int j;
for (j=index; j<pn; j++){
if (fr[i]==pg[j]){ if (j>farthest){
farthest=j; res=j; break; } }
if (j==pn) return i; }
return (res==-1)? 0: res;
}
void optimalpage(int pg[], int pn, int fn){
vector<int> fr;
int hit=0;
for(int i=0; i<pn; i++){
if(search(pg[i], fr)){
hit++;
continue;
}
if(fr.size()<fn) fr.push_back(pg[i]);
else{
int j= predict(pg, fr, pn, i+1);
fr[j]=pg[i]; }}
cout<<"No of hits = "<<hit<<endl;
Cout<<" No of misses = "<<pn-hit<<endl;
}
```

```
int main(){
 int pg[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};
 int pn = sizeof(pg) / sizeof(pg[0]);
 int fn = 4;
optimalpage(pg, pn, fn);
 return 0; }
```

output :-

No of hits = 7

No of misses = 6

c) LFU:-

```
#include<bits/stdc++.h>
using namespace std;
int pagefaults(int pages[],int n,int capacity){
unordered_set<int>s;
unordered_map<int,int>indexes;
int page_faults=0;
for(int i=0;i<n;i++){
if(s.size()<capacity){
if(s.find(pages[i])==s.end())
{ s.insert(pages[i]);
page_faults++;}
indexes[pages[i]]=i;}
else{
if(s.find(pages[i])==s.end())
{ int lru=INT_MAX,val;
for(auto it=s.begin();it!=s.end();it++){
if(indexes[*it]<lru)
{ lru=indexes[*it]
val=*it;}}
s.erase(val);
s.insert(pages[i]);
page_faults++;
}
indexes[pages[i]]=i;}}
return page_faults;}
```

```
int main() {
int pages[] = { 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 };
int n = sizeof(pages)/sizeof(pages[0]);
int capacity = 4;
cout << pagefaults(pages, n, capacity);
return 0;
}
```

output - 6

DATE / /
PAGE

Aim:-

write a C program to show belady's ~~algorithm~~ anomaly in FIFO algorithm.

Theory:-

it is the name given to the phenomenon where increasing the number of page frames results in an increase in the number of pagefaults for a given memory access pattern.

Belady's Anomaly in FIFO:-

Assuming a system that has no pages loaded in the memory and uses the fifo page replacement algorithm Consider the following reference string:

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Case 1:-

if the system has 3 frames, the given reference string on using fifo. yield a total of 9 pagefaults.

1	1	1	2	3	4	1	1	1	2	5	5
	2	2	3	4	1	2	2	2	5	3	3
		3	4	1	2	5	5	5	3	4	4
PF	PF	PF	PF	PF	PF	PF	X	X	PF	PF	X

Case 2:-

if the system has 4 frames, the given reference string on using fifo page replacement algorithm yield a total of 10 pagefaults

1	1	1	1	1	1	2	3	4	5	1	2
	2	2	2	2	2	3	4	5	1	2	3
		3	3	3	3	4	5	1	2	3	4
			4	4	4	5	1	2	3	4	5
PF	PF	PF	PF	X	X	PF	PF	PF	PF	PF	PF

it can be seen that on increasing the number of frames while using the FIFO page replacement algorithm, the number of page faults increased from 9 to 10.

Program code

```
#include<stdio.h>
~~int main(){~~
void pagefaults (int frame_size, int* ref, int len)
{ int* arr = new int[frame_size];
for(int i=0; i<frame_size; i++)
{ arr[i]=-1; }
int cnt=0; int start=0; int flag; int elm;
for (int i=0; i<len; i++) { elm=ref[i]; flag=0;
for (int j=0; j<frame_size; j++) { if (elm==arr[j]) { flag=1; break; } }
if (flag==0) {
  if (start<frame_size) {
arr[start]=elm;
start++; }
else if (start==frame_size) {
arr[0]=elm; start=1; } cnt++; }
printf("when the number of frames are :%d ", frame_size);
printf("the number of pagefaults is %d \n", cnt);
}
int main()
{ int ref[] = {1,2,3,4,1,2,5,1,2,3,4,5};
int len = sizeof(ref)/sizeof(ref[0]);
int frame_size=3;
pagefaults(frame_size, ref, len);
frame_size++;
pagefaults(framesize, ref, len);
}
```

output:-

when the number of frames are 3, the number of pagefaults is :9

when the number of fromes are 4, the number of pagefault is :10