**REPORT**

**CSN-232**

**OPERATING SYSTEMS**

**SUBMITTED BY:**

**GHETIA SIDDHARTH (17114038) HEMANT SINGH (17114038)**

**GAGAN KUMRE (17114028) DIVYANSHU SALVE (17114027)**

**ABHISHEK KUMAR (17114005) DEEPANSH NAGARIA (17114024)**

**SIDDHANT NAYAK (17114071) VED TAMBAT (17114075)**

**Introduction**

A good approximation to the optimal algorithm is based on the observation that pages that have been heavily used in the last few instructions will probably be heavily used again in the next few. Conversely, pages that have not been used for ages will probably remain unused for a long time. This idea suggests a realizable algorithm: when a page fault occurs, throw out the page that has been unused for the longest time. This strategy is called **LRU**(**Least Recently Used**) paging.

We have implemented the following algorithms in C or C++ without using existing/predefined classes. 

i. LRU - Counter Method  
ii. LRU - Stack Method  
iii. LRU - Aging Register Method  
iv. Approx. LRU - Clock Method

**Description**

* **LRU Counter implementation:**
  + Every page entry has a counter.
  + Every time a page is referenced, increment a global counter and store it in the page counter.
  + For page replacement, select the page with the lowest counter (search for it).
* **LRU Stack implementation:**
  + Maintain a stack of page number in a double link list.
  + Move a referenced page to the top (locate and change pointers).
  + For page replacement, take the bottom page.
* **LRU Aging Register Implementation:**
  + Maintain an n-bit aging register on a page reference, set R n-1 to 1, for each page frame.
  + Every T units of time shift the aging vector right by one bit.
  + Value of R decreases periodically unless the page is referenced.
  + On a page fault replace the page with the smallest value of R.
* **Approx. LRU Clock Implementation:**
  + Maintain a circular list of pages resident in memory by using a clock bit to track how often a page is accessed.
  + The bit is set whenever a page is referenced.
  + Clock hand sweeps over pages looking for one with used bit = 0.
  + Replaces pages that haven’t been referenced for one complete revolution of clock.

**How to run:**

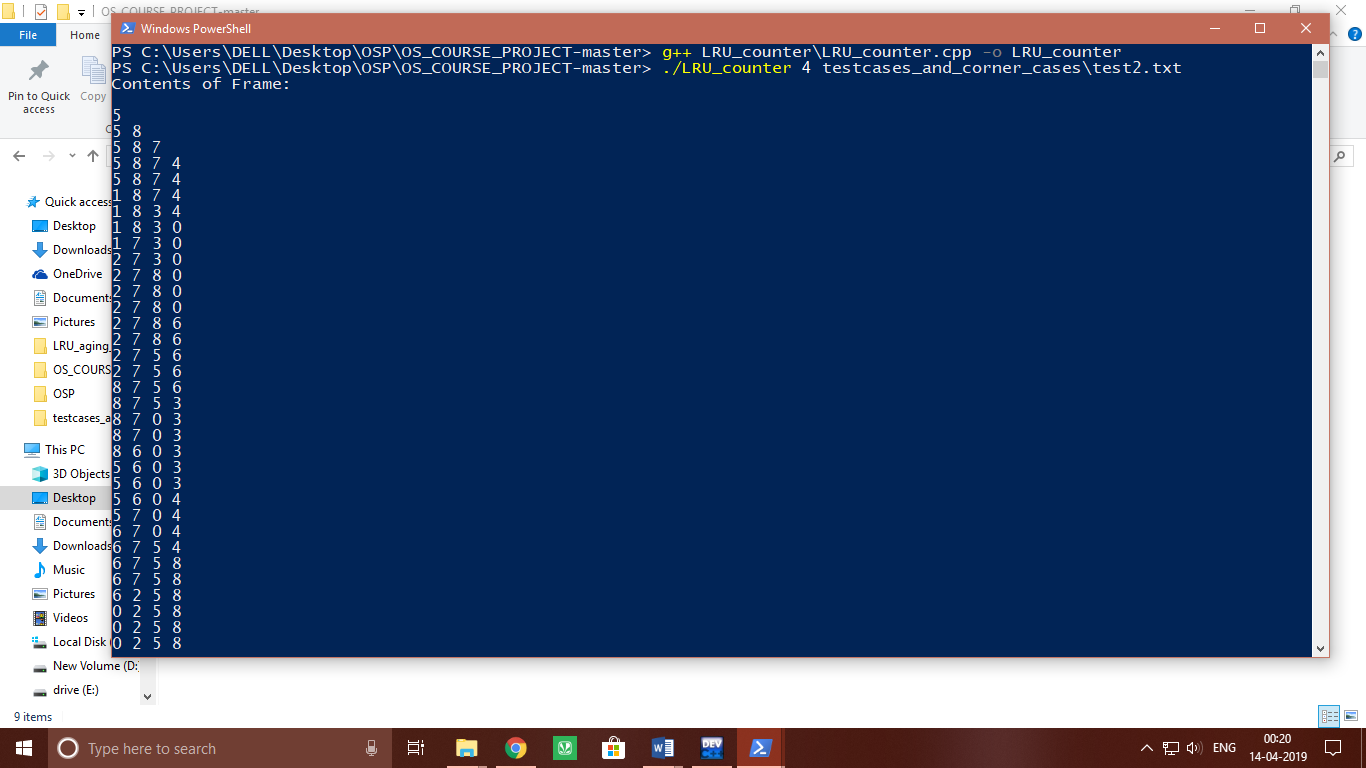
1. First, compile the file and create its object file

**g++ directory\<filename.cpp> -o <object\_file\_name>**

.

1. Second, to run the object file and give the input

**.\<object\_file\_name> <no.\_of\_frames> <test\_case\_file>**

****

**Graphs:**

Test1:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. of frames | LRU\_counter | LRU\_stack | LRU\_clock | LRU\_aging |
| 4 | 16 | 16 | 16 | 16 |
| 5 | 13 | 13 | 14 | 13 |
| 6 | 12 | 12 | 13 | 11 |
| 7 | 11 | 11 | 12 | 11 |
| 8 | 8 | 8 | 8 | 8 |
| 9 | 8 | 8 | 8 | 8 |

Test2:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. of frames | LRU\_counter | LRU\_stack | LRU\_clock | LRU\_aging |
| 4 | 33 | 33 | 31 | 32 |
| 5 | 28 | 28 | 25 | 26 |
| 6 | 23 | 23 | 22 | 19 |
| 7 | 16 | 16 | 14 | 16 |
| 8 | 13 | 13 | 14 | 13 |
| 9 | 9 | 9 | 9 | 9 |

Test3:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. of frames | LRU\_counter | LRU\_stack | LRU\_clock | LRU\_aging |
| 4 | 25 | 25 | 26 | 20 |
| 5 | 18 | 18 | 21 | 20 |
| 6 | 13 | 13 | 11 | 18 |
| 7 | 7 | 7 | 7 | 11 |
| 8 | 7 | 7 | 7 | 8 |
| 9 | 7 | 7 | 7 | 8 |

Test4:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. of frames | LRU\_counter | LRU\_stack | LRU\_clock | LRU\_aging |
| 4 | 24 | 24 | 22 | 28 |
| 5 | 14 | 14 | 14 | 13 |
| 6 | 6 | 6 | 6 | 6 |
| 7 | 6 | 6 | 6 | 6 |
| 8 | 6 | 6 | 6 | 6 |
| 9 | 6 | 6 | 6 | 6 |

Test5:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. of frames | LRU\_counter | LRU\_stack | LRU\_clock | LRU\_aging |
| 4 | 48 | 48 | 50 | 49 |
| 5 | 41 | 41 | 41 | 41 |
| 6 | 28 | 28 | 28 | 29 |
| 7 | 19 | 19 | 16 | 18 |
| 8 | 8 | 8 | 8 | 8 |
| 9 | 8 | 8 | 8 | 8 |

Corner Case1:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. of frames | LRU\_counter | LRU\_stack | LRU\_clock | LRU\_aging |
| 4 | 60 | 60 | 60 | 60 |
| 5 | 60 | 60 | 60 | 43 |
| 6 | 6 | 6 | 6 | 6 |
| 7 | 6 | 6 | 6 | 6 |
| 8 | 6 | 6 | 6 | 6 |
| 9 | 6 | 6 | 6 | 6 |

Corner Case2:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. of frames | LRU\_counter | LRU\_stack | LRU\_clock | LRU\_aging |
| 4 | 1 | 1 | 1 | 1 |
| 5 | 1 | 1 | 1 | 1 |
| 6 | 1 | 1 | 1 | 1 |
| 7 | 1 | 1 | 1 | 1 |
| 8 | 1 | 1 | 1 | 1 |
| 9 | 1 | 1 | 1 | 1 |

Corner Case3:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. of frames | LRU\_counter | LRU\_stack | LRU\_clock | LRU\_aging |
| 4 | 10 | 10 | 10 | 10 |
| 5 | 10 | 10 | 10 | 10 |
| 6 | 10 | 10 | 10 | 10 |
| 7 | 10 | 10 | 10 | 10 |
| 8 | 10 | 10 | 10 | 10 |
| 9 | 10 | 10 | 10 | 10 |

**Average Time to execute for different Algorithm:**

**For normal test case:**

**For corner cases:**

**Complexity Analysis:**

**LRU\_counter.cpp :**

time complexity= O(f\*p)

Explanation:

We have called the LRU\_Policy method in the main function. In the LRU\_Policy method we have nested loops each running f times i.e. number of frames and p times i.e. size of page stream.

**LRU\_Stack.cpp :**

time complexity= O(f\*p\_f)

Explanation:

We have a while loop in our main function which is calling the policy method. Policy method is having the constraints like page\_faults and the stac which will fill up to the number of frames. So we can write the complexity of our program as O(f\*p\_f) where f is number of frames and p\_f is number of page faults.

\*We have considered the unordered map performance to be O(1).

**App\_LRU\_Clock\_Algo.cpp** :

time complexity= O((f+s\_c)\*p)

Explanation:

We are running a for loop which runs for p times i.e. page size times. Inside the loop LRU\_Policy is running. It contains while loop as well as for loop. It is contributing O(f) complexity. There is also an additional cost of maintaining the clock and giving 2nd chance. So net complexity is O((f+s\_c)\*p) where f is number of frames, s\_c is the additional cost and p is size of page stream.

**Aging\_register.cpp** :

time complexity= O(r\*p)

Explanation:

We are running a LRU\_Policy in our main function. In the LRU\_Policy method we have nested loops each running reg\_number times i.e. number of frames and p times i.e. size of page stream. (In comparison to reg\_number the f i.e. frame size is small and thus is neglected). Its running time also depends on the fact that we are shifting registers only once in every clock cycle not in all instances of the for loop thus it decreases its complexity to some extent.