**Cache Memory and Analysis on Least Recently Used Replacement Algorithm**

Cache Memory and Analysis on Least Recently Used (LRU) Replacement Algorithm

BY

TAYYBA SALAMAT\_032

SANA AKBAR\_026

UBAIDA WAHEED\_034

Page | 1

**Cache Memory and Analysis on Least Recently Used Replacement Algorithm**

**Abstract:**

Cache memory is the less costly memory which contains the currently used data. As it is more time taking to access data from main memory, it reduces the processing time and boost up the performance. Another concept is the Cache block replacement algorithms. With the passage of time and evolution of technologies, new methodologies and algorithms are introduced to improve the performance of cache. In this Research paper, Cache memory and analysis on one of the most common Cache block replacement algorithm i.e. least recently used cache block replacement Algorithm is done. Moreover, implementation of least recently used algorithm in C++, advantages, disadvantages and the variation of miss and hit rate with the replacement of each block is discussed. Analysis shows that least recently used algorithm is easy and common amongst all other replacement algorithms.

**Keywords:**

Cache memory, replacement techniques, performance of cache, least recently used algorithm. **Introduction:**

Cache memory act as a bridge between main memory and CPU which is high speed memory, less expensive than register. Main purpose of the cache is to reduce the fetching time between slow main memory and fast processor by reducing cost. The overall performance of the system is somehow dependent on the cache which is directly proportional to the size of cache. When a processor requests for a particular block of information, firstly it is checked in cache by system. In case, if desired information is present cache then processor uses it and it will increment the hit rate otherwise it must carried from main memory to cache for future use. In case, block is not present in cache, then it will increment the miss rate or page fault.

The cache is restricted in size i.e. number of blocks, so there is a requirement of an algorithm to fetch blocks of data from the main memory into the cache and to throw out already present data from to cache to the main memory. There are number of algorithms introduced e.g. Recency Based Algorithms, frequency Based Algorithms, Recency/Frequency Based Algorithms, Function based algorithms, Randomized algorithms etc. The present-day implementation is based on Least Recently Used (LRU) algorithm belongs to Recency based Algorithm’s category, which has been around since the start of computer science. It basically removes the least recently used block from cache when it becomes full according to its capacity. It is used in numerous applications of web solutions (proxies, web-servers, etc.), Virtual memory and general improvements of CPU performance time.

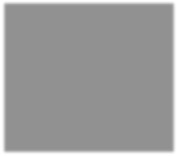
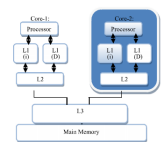
**Literature Review:**

Generally cache memory is classified into 3 levels.

**3 Levels of cache memory:**

Page | 2

**Cache Memory and Analysis on Least Recently Used Replacement Algorithm**

Level L1/Chip memory: it is the Smallest and fastest 

as lies with in processor.

Level L2: It is the larger but slower than L1 cache and

stores recently accessed information.

Level L3: It lies in motherboard, faster than main

memory but largest and slowest among other levels.

In asymmetric clusters, every processor has its L1 and

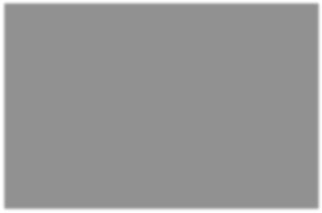
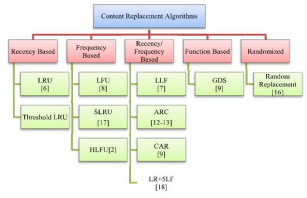
L2 cache memory but L3 is shared as shown:

Due to the shortage of cache memory size, different

algorithms and replacement techniques are

presented to make decision that which block of data is going to be replaced when cache becomes full.

**Replacement Algorithms:**

****

**(i) Recency Based Algorithms**

∙ **HLFU (History Least Recently Used) Algorithm):** Removes block which have less hist value ∙ **LFU (Least Frequently Used) Algorithm:** Eliminates which is used less frequently used if have same frequency then removes randomly

∙ **SLRU (Segmented LRU) Algorithm:** Divided into protected(mostly used data) and unprotected parts (when received first request) and both are managed by LRU **(ii) frequency Based Algorithms**

∙ **LRU (Least Recently Used) Algorithm):** Removes least recently used data from cache ∙ **FIFO Algorithm:** first in first out

**(iii) Regency/Frequency Based Algorithms**

∙ **CAR (Clock with Adaptive Replacement) Algorithm:** It is scan resistant, best and easy to implement. Provides facility of self-tuning and

Page | 3

**Cache Memory and Analysis on Least Recently Used Replacement Algorithm**

∙ **ARC (Adaptive Replacement Cache):** cache is separated in two queues and each handled by LRU and a ghost cache which has data that would be used In coming stages

∙ **LLF (Lowest Latency First):** eliminating the block with the lowest download potential ∙ **LR+5LF Algorithm:** combination of **LFU** and **LRU** and also reduces cache miss (process of shifting data from main memory to cache)

**(iv) Function based algorithms**

∙ **GDS (Greedy Dual Size):**if the block has larger size then correspondingly less index of file and file that have less index will be removed

**(v) Randomized algorithms**

∙ **RR (Random Replacement) Algorithm:** Removes any random file , don’t keep any record

It can be seen from above list that the LRU algorithm is also implemented in mutual form with other algorithms. At present, cache replacement algorithms mainly pays attention on the following strategies: Least Recently Used Algorithm, Least Frequently Used Algorithm, Function based Algorithm, Randomized-based Algorithm and Weighting-based Algorithm.

Least Recently Used (LRU) algorithm is one of the most useful technique in web cache. In this algorithm measure of time period for which block is present in cache has very importance and helps in replacement. The advantage of this algorithm is that it is easy to be implemented and

has a good performance in client-side. But it only considers the last referenced time while ignoring the frequency of references for a certain web object.

Every algorithm has its associated advantages and drawbacks. In least recently used algorithm advantages are that it is easy to understand, implement and also it has less chance of ***Belady’s Anomaly.*** Belady’s anomaly is actually a phenomena in which when we increase the page frames then the miss rate or page faults increases. On the other hand there are some problems with the LRU replacement algorithm i.e. for understanding of its implementation best data structure concepts are required and it requires considerable hardware support.

We should also focus on some optimization techniques for LRU. These are: **1. Eliminate shifting in LRU:**

We can implement it via queue as well as via linked list. Problem with queue is that when we replace the least recently used block and inserts the new one then it requires lot of shifting and also it is time consuming:

| **4** | **7** | **9** | **1** | **3** |
| --- | --- | --- | --- | --- |

8

This problem can easily be overcome by using linked list that is simply by deleting the first node and inserting a node at last.

Page | 4

**Cache Memory and Analysis on Least Recently Used Replacement Algorithm**

**NULL Null** 4 7 9 1 3

**Null** 8

**2. Optimize searching:**

This can be done by using hash i.e. each of node having an associated a key and value where key is the content of node and value is the address of the node e.g. x, y and z addresses. When blocks are fetched into the hash they have content along with their corresponding addresses. When new block is arrived in hash we will check its address, if its address value is empty then we easily conclude that it is not present in cache. Then removes a node from front end along with its address and creates new one assigning its address. In this respect we can reduce the process of searching again and again in all nodes.

**Methodology:**

**Concepts:** *Programming Fundamental, Object Oriented Programming and Data Structure* **Tool, Language:** visual studio, C++

**Description:**

As we said that cache memory has limited number of blocks to hold the currently used data. In order to understand the concept of least recently used replacement algorithm let us consider the cache has capacity of three block, initially our cache is empty and number of hits and misses are zero. **Cache**

**Main Memory**

|  |  |  |
| --- | --- | --- |

Number of hit=0 Number of miss=0

**Process** or

If processor request for 7 block, it is not present in cache so carried from main memory by increasing miss rate by one.

| **7** |  |  |
| --- | --- | --- |

Requests for 7 **Main Memory Process**

or

Number of hit=0 Number of miss=1

Same for blocks 3 and 8, firstly system will check them in the cache as these are not found in cache then again these are fetched one by one from main memory and increments miss rate respectively.

| **7** | **3** |  |
| --- | --- | --- |

Requests for block **3 Main Memory Processor**

Page | 5

**Cache Memory and Analysis on Least Recently Used Replacement Algorithm**

Number of hit=0 Number of miss=2

| **7** | **3** | **8** |
| --- | --- | --- |

Requests for block 8 **Main Memory Processor** Number of hit=0 Number of miss=3

Now the cache did not have any capacity of block. If processor request for 7 again as it is present in the cache so 7 will come in rear end and rearranged.

Request for 7

**Main Memory**

| **7** | **3** | **8** |
| --- | --- | --- |

Number of hit=1 Number of miss=3

| **3** | **8** | **7** |
| --- | --- | --- |

**Processor** or

**Main Memory Processor**

So when processor request for another block that did not exist in cache then there is need to replace the existing block in order to accommodate new one. LRU replacement algorithm states that the least recently used block should be removed. Let processor requests for the block 9 and least recently block is at front end i.e. 3 will be removed from cache to accommodate 9.And it will be again miss.

Removes 3 Request for 9

| **8** | **7** | **9** |
| --- | --- | --- |

**Main Memory Processor** Place 9

Number of hit=1 Number of miss=4

So in this way Least recently used (LRU) replacement algorithm of cache block works. **Implementation/source Code:**

#include "stdafx.h"

#include<iostream>

#include<string>

using namespace std;

int hit=0,miss=0,count=1;;

class cache

{private:

struct node

{

int data;

struct node\* next; // Pointer to next node in DLL

struct node\* prev; // Pointer to previous node in DLL

}\*head;

public:

Page | 6

**Cache Memory and Analysis on Least Recently Used Replacement Algorithm**

cache()

{head=NULL;}

bool empty() //when tack is empty

{

if(head==NULL)

{return true;}

else

return false;}

void shift\_mf(int x)

{ int repeat;

node \*ptr;

ptr=head;

if(!empty())

{

if(head->data==x) //if first block is required

{repeat=head->data;

head->next->prev=NULL;

head=head->next;

cout<<"You Shifted '"<<ptr->data<<"' Block"<<endl; delete ptr;

}

else //if not first

{node \*ptr=head;

node \*preptr=NULL;

while(ptr->data!=x)

{preptr=ptr;

ptr=ptr->next;}

repeat=ptr->data;

preptr->next=ptr->next;

if(ptr->next!=NULL) //if not the last block

{ptr->next->prev=preptr;}

cout<<"You Shifted '"<<ptr->data<<"' Block"<<endl; delete ptr;}

node \*mr=new node;

mr->next=NULL;

node \*last;

last=head;

while(last->next!=NULL)

{last=last->next;}

mr->data=repeat;

last->next=mr;

mr->prev=last;

}

}

void insert(int capacity) //function to insert block in cache { //to keep record of Blocks and where cache full node \*ptr;

Page | 7

**Cache Memory and Analysis on Least Recently Used Replacement Algorithm**

int repeat,value; //repeat to hold frequent block value,value is to take value from user ptr=new node;

ptr->next=NULL;

string ch;

cout<<"Please Enter the Value of Cache Block"<<endl;

cin>>value;

if(empty()) //if empty cache

{

ptr->data=value;

head=ptr;

ptr->prev=NULL;

cout<<"You Inserted : "<<ptr->data<<endl<<endl;//insertion

count++;

miss++; //miss when not present in cache

}

else

{//if not epmty,then check for the same/repeated block

node \*check;

check=head;

bool present=false;

if(empty())

{present=false;}

else

{ while(check!=NULL) //check till end if not empty

{

if(check->data==value)

{present=true;

repeat=check->data;}

check=check->next;

}

}

if(present==true)

{if(head->next==NULL)

{hit++;}

else if(count<=capacity || count>=capacity)

{

shift\_mf(repeat);

hit++; //hit when present in cache

}

}

if(present==false)

{ if(capacity<count) //if greater then capacity,removes first

{node \*first;

first=head;

head=first->next;

head->prev=NULL;

Page | 8

**Cache Memory and Analysis on Least Recently Used Replacement Algorithm**

first->next=NULL;

delete first;

}

node \*nw=new node; //insert block at end and miss increments

nw->next=NULL;

node \*last;

last=head;

while(last->next!=NULL)

{last=last->next;}

last->next=nw;

nw->prev=last;

nw->data=value; //insertion

count++;

cout<<"You Inserted : "<<nw->data<<endl<<endl;

miss++;

}

}

cout<<"\t\tNO OF Misses : \t"<<miss<<"\n\t\tNO OF Hits : \t"<<hit<<endl;

}

void display() //function to display cache blocks

{

if(empty())

{cout<<"Your Cache has no Block"<<endl;}

else

{node \*ptr;

ptr=head;

cout<<"Your Cache has :"<<endl;

while(ptr!=NULL)

{

cout<<ptr->data<<"\t";

ptr=ptr->next;

}

}

cout<<endl<<endl;

}

};

int \_tmain(int argc, \_TCHAR\* argv[])

{cache c;

int cap,ch;

cout<<" LEAST RECENTLY USED(LRU) CACHE REPLACEMENT ALGORITHM"<<endl; cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_"<<endl;

cout<<"Enter the Capacity of Blocks in Your Cache"<<endl;

cin>>cap;

do

{

Page | 9

**Cache Memory and Analysis on Least Recently Used Replacement Algorithm**

cout<<"1 \tPress '1' to get Required Block \n2. \tPress '2' to Display Cache Content \n3. \tPress '3' to Exit\n"<<endl;

cout<<"Please Enter Your Choice in Number 1-3"<<endl;

cin>>ch;

if(ch==1)

{system("CLS");

c.insert(cap);}

if(ch==2)

{system("CLS");

c.display();}

if(ch==3)

{system("CLS");

cout<<"\n\n\n\n\n\n\n......................\*THANK YOU\*......................\n\n\n\n\n\n"<<endl;} if(ch!=1 && ch!=2 && ch!=3)

{system("CLS");

cout<<"ERROR! Please Enter Valid Number"<<endl;

}

}while(ch!=3);

cout<<endl;

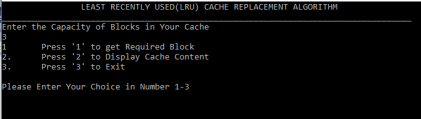
system("pause");

return 0;

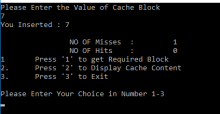
}

**Results and Discussion:**

When we executes the source code of LRU replacement algorithm, first ask to enter the capacity of the cache. After that, the following main menu appears.



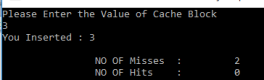
When we pressed 1, system request to enter the block value as assuming initially our cache is empty.

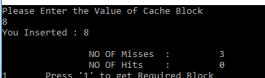


Page | 10

**Cache Memory and Analysis on Least Recently Used Replacement Algorithm**

Then we inserted 3 and 8 one by one.

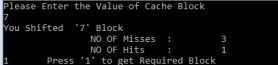




Pressed ‘2’ in order to see the cache contents.



Requested 7 again.



7 is shifted in rear end.



Then requested for 9 and 3 removed.





Thus block that is least recently used was kept at front end that is removed and the new one i.e. 9 is inserted at the rear end at it is most recently used block.

Page | 11

**Cache Memory and Analysis on Least Recently Used Replacement Algorithm**

**Conclusion:**

Cache is the vital part of processing and its performance can be improved using different policies and algorithms. There are many algorithms and each of them have their respective pros and cons. LRU algorithm is the algorithm which is being used since the introduction of cache replacement techniques and still it is being used in integration with other algorithms. Our approach should be in favor of increasing the performance of the system which is greatly be influenced by cache performance. Performance of the cache memory can be improved by reducing the number of miss rate, increasing the hit rates, by increasing cache size and using an appropriate cache block replacement algorithm.

**Reference:**

1. Ma, T., Hao, Y., Shen, W., Tian, Y., & Al-Rodhaan, M. (2018). An improved web cache replacement algorithm based on weighting and cost. *IEEE Access*, *6*, 27010-27017.

https://ieeexplore.ieee.org/abstract/document/8344541

2. Friedlander, E., & Aggarwal, V. (2019). Generalization of LRU cache replacement policy with applications to video streaming. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, *4*(3), 1-22.

https://dl.acm.org/doi/abs/10.1145/3345022

3. Monniaux, D., & Touzeau, V. (2019). On the complexity of cache analysis for different replacement policies. *Journal of the ACM (JACM)*, *66*(6), 1-22.

https://dl.acm.org/doi/abs/10.1145/3366018

4. Meizhen, W., Yanlei, S., & Yue, T. (2013, August). The design and implementation of LRU-based web cache. In *2013 8th International Conference on Communications and Networking in China (CHINACOM)* (pp. 400-404). IEEE.

https://ieeexplore.ieee.org/abstract/document/6694629

5. Javaid, Q., Zafar, A., Awais, M., & Shah, M. A. (2017). Cache memory: An analysis on replacement algorithms and optimization techniques. *Mehran University Research Journal of Engineering & Technology*, *36*(4), 831.

https://search.informit.com.au/documentSummary;dn=238122130715258;res=IELENG

6. Morales, K., & Lee, B. K. (2012, December). Fixed segmented LRU cache replacement scheme with selective caching. In *2012 IEEE 31st International Performance Computing and Communications Conference (IPCCC)* (pp. 199-200). IEEE.

https://ieeexplore.ieee.org/abstract/document/6407712

7. Eytan, O., Harnik, D., Ofer, E., Friedman, R., & Kat, R. (2020). It's Time to Revisit {LRU} vs.{FIFO}. In *12th {USENIX} Workshop on Hot Topics in Storage and File Systems (HotStorage 20)*.

Page | 12

**Cache Memory and Analysis on Least Recently Used Replacement Algorithm**

https://www.usenix.org/conference/hotstorage20/presentation/eytan

8. Cioranu, C., Cioca, M., & Cioca, L. I. (2012). Improving Image Processing Systems by Using Software Simulated LRU Cache Algorithms. *Informatica Economica*, *16*(4), 68-73.

http://www.revistaie.ase.ro/content/64/08%20-%20Cioranu,%20Cioca,%20Cioca.pdf

9. Godavarthy, A., Lakshminarasimhachar, S., & Gopinathan, S. (2010). Simulation and analysis of cache replacement algorithms. In *Proc. International Conference on Computer Design* (pp. 1-41).

https://www.cse.scu.edu/~twang1/studentProjects/Cache\_replacement\_10s.pdf

10. Ghasemzadeh, H., Mazrouee, S., & Kakoee, M. R. (2006, March). Modified pseudo LRU replacement algorithm. In *13th Annual IEEE International Symposium and Workshop on Engineering of Computer-Based Systems (ECBS'06)* (pp. 6-pp). IEEE.

https://ieeexplore.ieee.org/abstract/document/1607387

Page | 13