California State University, Fresno
Lyles College of Engineering
Electrical and Computer Engineering Department

TECHNICAL REPORT

Assignment:	**Project 2**
Experiment Title:	**Cache Replacement – Hit Rate**
Course Title:	**ECE 144 (Embedded Operating Systems)**
Instructor:	**Dr. Nan Wang**

Prepared by: Omer Al Sumeri

Student ID # 110513461

Date Submitted: 11/11/2023

INSTRUCTOR SECTION

Comments: ____

____

____

____

____

____

____

Final Grade: ____________________

Table of Contents

1. Objective

The objective of this project is to demonstrate the three different page replacement schemes: FIFO or oldest page replacement, random replacement, and least recently used. The programming C++ will be used.

2. Hardware Requirements

- Personal Computer that supports Visual Studio 2022 or comparable IDE that will compile C++

3. Software Requirements

- Visual Studio 2022 or comparable IDE that will compile C++

4. Introduction

The following knowledge is required in order to complete the assignment.

The cache is one of the higher storage devices in the memory hierarchy. The cache is used to store temporary data that the CPU needs. This requires the cache to be a high-speed data storage since it interacts the register on top of the CPU. The cache is also smaller in size to minimize the time for search.

There are three different placement methods for the cache. The three are Direct mapping, Set Associate mapping, and Associate mapping. Direct mapping relates one spot in the cache for each page in memory. Set Associate mapping splits the cache into different sets and allows a page to enter any spot in the set. Associate mapping allows for any page to be placed anywhere in the cache. See **Figure 1** for a visual figure. For this project, the cache will have Associate Mapping placement method. [1]

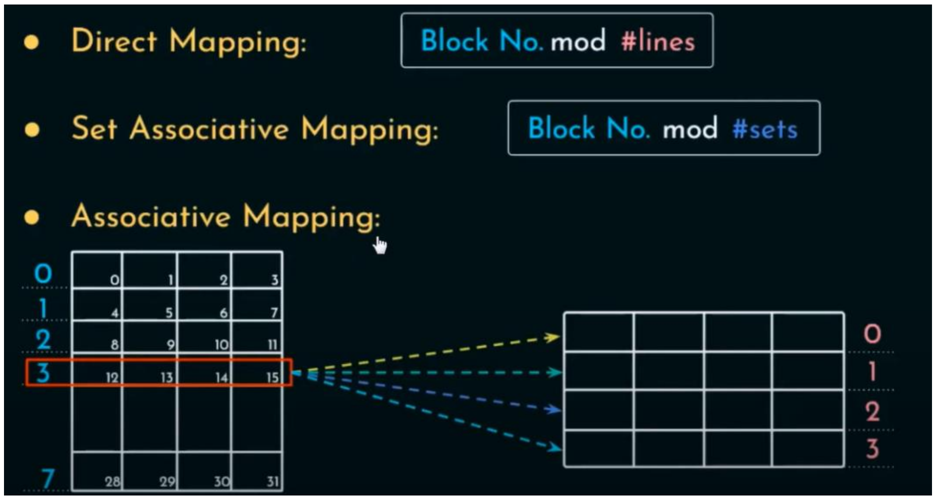


Figure 1. Visual Demonstration of the different placement methods for the cache.[1]

Once the cache is full after using the placement method, the question arises which page should be replaced for the new page to enter? If the cache is searched and the currently requested page is not found in the cache, block replacement methods need to be deployed. The three page replacement schemes that will be researched and experimented are FIFO or oldest page replacement, random replacement, and least recently used. [1]

For the page replacement method FIFO or oldest page replacement, a queue is used to keep track of the pages in the cache. Once the cache is full, pages are replaced depending on when they enter the cache. **Figure 2** contain an example.

The page reference log is 1,3,0,3,5,6,3 and there are 3 spots in the cache. Since initially the empty, there are three page faults. The fourth page call is 3, which is already in the cache, so it is a hit. The next page call, 5, is not in the cache so it replaces the spot of page that was entered first which is 1. The same idea happens for the next page call, 6, which replaces 3. The last page call is for 3 which replaces 0. [2]

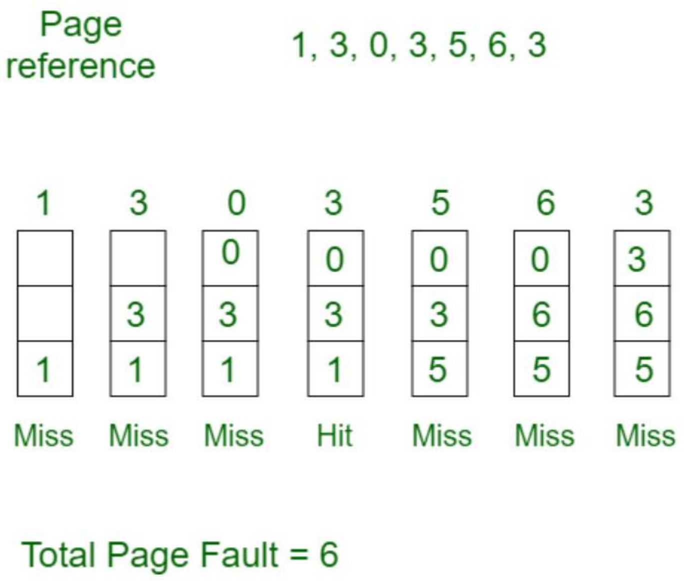


Figure 2. Visual example of the page replacement method FIFO or oldest page replacement.[2]

For the page replacement method least recently used, the page that is replaced in the cache is the least recently used page. If a page is requested and it is already in the cache, then it will be marked as a hit and that page will be the most recently used page. **Figure 3** contains an example.

The page reference log is 7,0,1,2,0,3,0,4,2,3,0,3,2,3 and there are four spots in the cache. Since the cache is initially empty, there are four page faults. The next miss is when there is a page call for 3. The LRU page was 7, so 3 takes the spot of 7. The same pattern of updating the recently used number continues as there is a hit and when there is a miss, the currently requested page takes the place of the last recently used.[2]

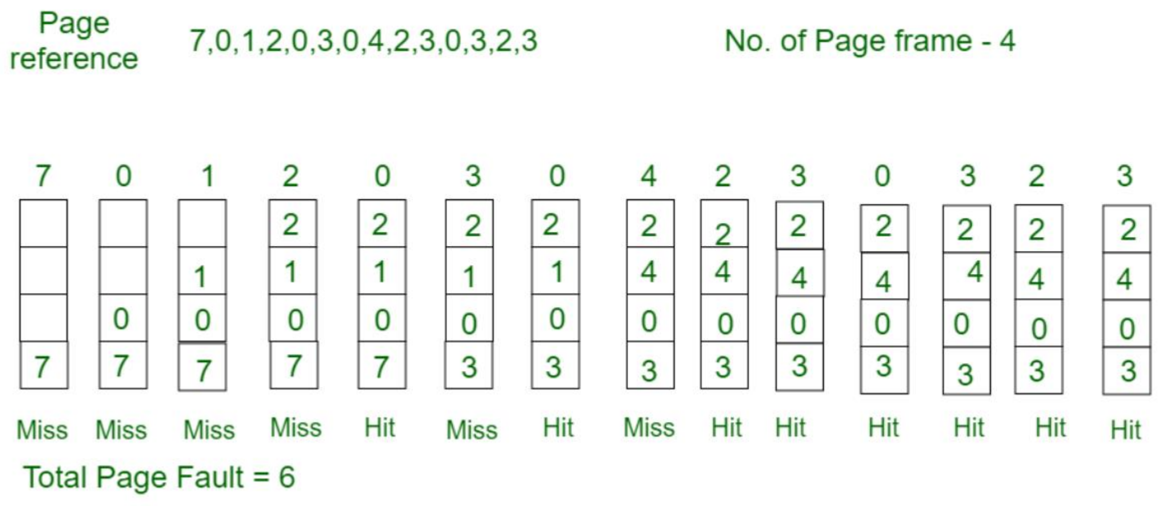


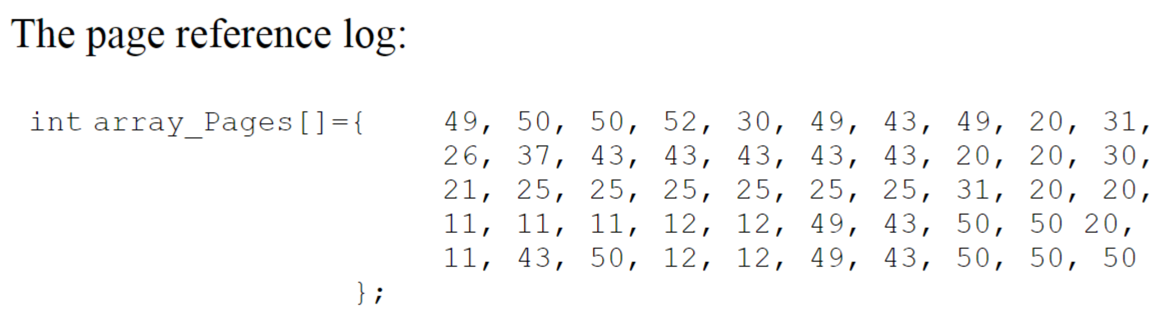
Figure 3. Visual example of the page replacement method Least Recently Used.[2]

The last page replacement method is random placement. Once the cache is full, the currently selected page is randomly placed into the cache. Random page replacement reduces the search time to find the least recently used or added memory needed since no data structure is used to keep track of the contents such as oldest page replacement.

5. Problem Statement

Given the page reference log in **Figure 4** that is provided in the assignment handout, compare the three page replacement methods in programming language of the student's choice. The three page replacement methods are FIFO or oldest page replacement, random replacement, and least recently used replacement.

The cache will be Associate mapping so any page can be placed anywhere in the cache. Compare the efficiency between the three different placement algorithms.



The page reference log:

```
int array_Pages[]={     49, 50, 50, 52, 30, 49, 43, 49, 20, 31,
                        26, 37, 43, 43, 43, 43, 43, 20, 20, 30,
                        21, 25, 25, 25, 25, 25, 25, 31, 20, 20,
                        11, 11, 11, 12, 12, 49, 43, 50, 50 20,
                        11, 43, 50, 12, 12, 49, 43, 50, 50, 50
                  };
```

Figure 4. Page Reference Log provided by the Professor.

6. Implementation

The project was implemented in the programming language C++ using the Visual Studio IDE. A class was created called Page. The class has two attributes, content and last recently used. The content attribute is an integer that will hold the value of the page and the last recently used attribute is an integer also that will contain the time the content was last recently used.

In the default constructor of the program, the content is set to 0 and the last recently used attribute is set to 100. The last recently used attribute is set to a large number to signify the page is old.

There are functions in the class such as seters and getters for the content and last recently used attribute. There is a function that increments the last recently used value by 1. This will be used in the last recently used replacement method function as that attribute will be incremented of all the pages in the cache.

In the main program, two array of data type Page were created. One for the cache and another for the page log. The page log has to be manually inserted into the array since it was provided beforehand by the professor. Three functions separate function were created for each replacement method and then called in the main function.

Oldest Page Replacement Function

For the implementation of oldest page replacement method, six arguments are passed into the function. The arguments are the array that hold the cache, the cache size, the array that hold the page log, page log size, and the address of the hit and miss counter.

There are two for loops. The outer for loop went through each number in the page log and the inner for loop went through each number in the cache. In the nested for loops, there are two if statements. If the current content in the page log is equal to the current content in the cache, then the hit counter is incremented and the inner for loop breaks.

If the end of the cache is reach and content in the page log is not found, then the miss counter is incremented. All content in the cache is shifted to the left once, stopping right before position 0. The 0 position in the cache array is then equal to the current content of the page log. This continues to happen until all numbers in the page log are searched.

Random Page Replacement Function

For the implementation of random page replacement method, six arguments are passed into the function. The arguments are the array that hold the cache, the cache size, the array that hold the page log, page log size, and the address of the hit and miss counter.

There are two for loops. The outer for loop went through each number in the page log and the inner for loop went through each number in the cache. In the nested for loops, there are two if statements. If the current content in the page log is equal to the current content in the cache, then the hit counter is incremented and the inner for loop breaks.

If the end of the cache is reach and content in the page log is not found, then the miss counter is incremented. A random number is generated using the rand() function and that number is then modular 6. The modular 6 ensures that the random location is less than 6. The content of the current page log is then placed in the random location that was generated. This continues to happen until all numbers in the page log are searched.

Least Recently Used Page Replacement Function

For the implementation of least recently used page replacement method, six arguments are passed into the function. The arguments are the array that hold the cache, the cache size, the array that hold the page log, page log size, and the address of the hit and miss counter.

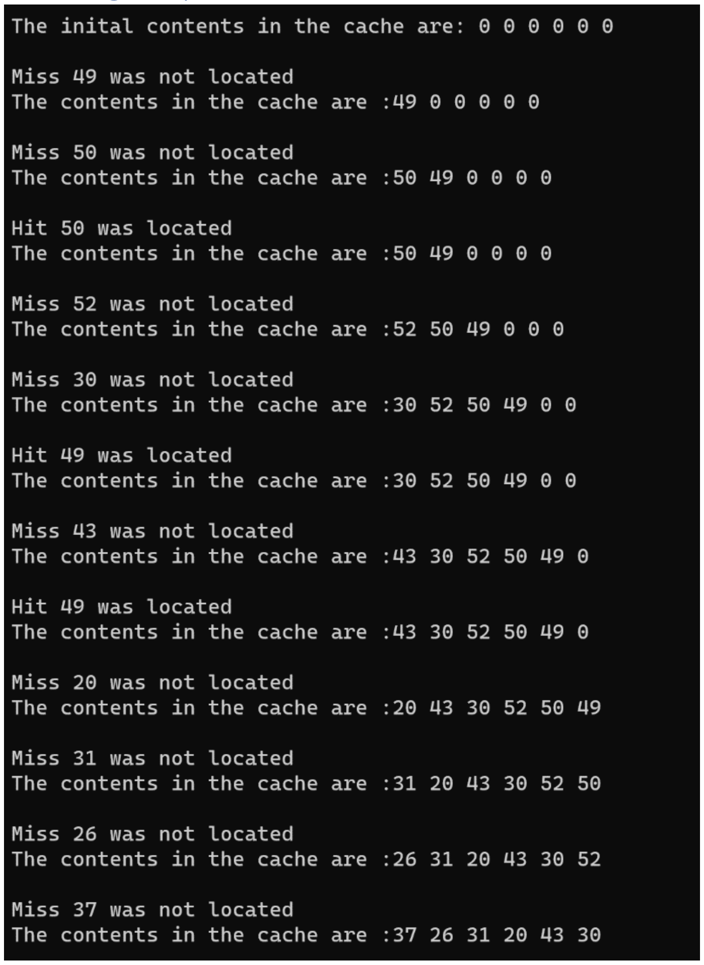
There are two for loops. The outer for loop went through each number in the page log and the inner for loop went through each number in the cache. In the nested for loops, there are two if statements. If the current content in the page log is equal to the current content in the cache, then the hit counter is incremented, the last recently used attribute of all the pages in the cache are incremented the inner for loop breaks.

If the end of the cache is reach and content in the page log is not found, then the miss counter is incremented. A variable that holds the maximum number of last recently used is set equal to the first number in the cache. The 0 is also saved in another variable. The whole cache is then searched to look for which page in the cache has the highest number of last recently used. After that number is found, the currently requested page from the page log then takes the place of that page in the cache. This continues to happen until all numbers in the page log are searched.

7. Result and Comparison

To test the functionality of all three replacement methods, the cache will be printed out to the terminal for the first 10 page log entries. The terminal will display if it is a hit or miss and the contents of the cache after the current page log entry has been entered.

Oldest Page Replacement



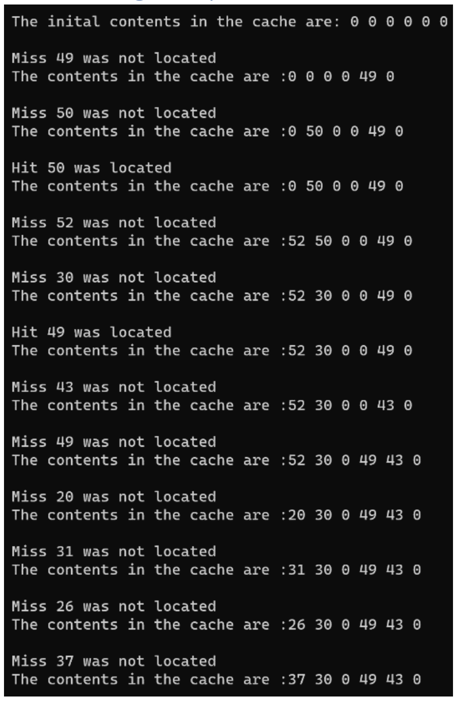
```
The inital contents in the cache are: 0 0 0 0 0 0

Miss 49 was not located
The contents in the cache are :49 0 0 0 0 0

Miss 50 was not located
The contents in the cache are :50 49 0 0 0 0

Hit 50 was located
The contents in the cache are :50 49 0 0 0 0

Miss 52 was not located
The contents in the cache are :52 50 49 0 0 0

Miss 30 was not located
The contents in the cache are :30 52 50 49 0 0

Hit 49 was located
The contents in the cache are :30 52 50 49 0 0

Miss 43 was not located
The contents in the cache are :43 30 52 50 49 0

Hit 49 was located
The contents in the cache are :43 30 52 50 49 0

Miss 20 was not located
The contents in the cache are :20 43 30 52 50 49

Miss 31 was not located
The contents in the cache are :31 20 43 30 52 50

Miss 26 was not located
The contents in the cache are :26 31 20 43 30 52

Miss 37 was not located
The contents in the cache are :37 26 31 20 43 30
```

Figure 5. Simulation screenshot of the terminal after the Oldest Page Replacement has run.

To the left is **Figure 5**. **Figure 5** shows the cache contents during the execution of the function Oldest Page replacement. For the first two page searches, 49 and 50, were misses so they were placed in the cache. 50 was then searched for so it was a hit. Pages 52 and 30 were not in cache so they were labeled as misses and then placed in the cache.

It can be seen that all the pages that are placed in the cache in a FIFO order. The order is not changed even if the Page is found in the cache. When page 31 is searched for, the cache is full. 49 was taken out of the cache and all other pages are incremented forward. 31 is now the first spot in the cache. This order continues for the whole page log.

Random Page Replacement



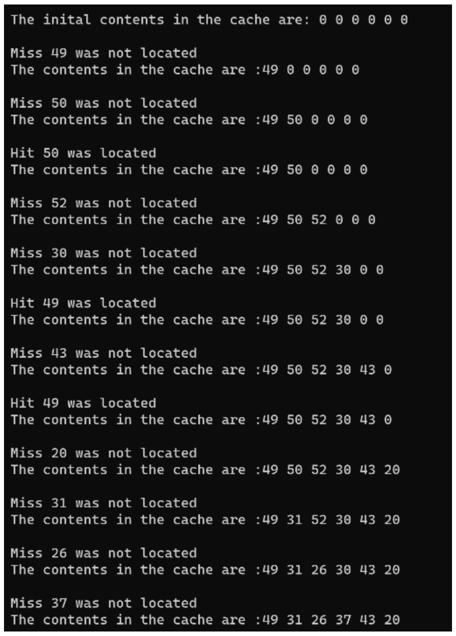
```
The inital contents in the cache are: 0 0 0 0 0 0

Miss 49 was not located
The contents in the cache are :0 0 0 0 49 0

Miss 50 was not located
The contents in the cache are :0 50 0 0 49 0

Hit 50 was located
The contents in the cache are :0 50 0 0 49 0

Miss 52 was not located
The contents in the cache are :52 50 0 0 49 0

Miss 30 was not located
The contents in the cache are :52 30 0 0 49 0

Hit 49 was located
The contents in the cache are :52 30 0 0 49 0

Miss 43 was not located
The contents in the cache are :52 30 0 0 43 0

Miss 49 was not located
The contents in the cache are :52 30 0 49 43 0

Miss 20 was not located
The contents in the cache are :20 30 0 49 43 0

Miss 31 was not located
The contents in the cache are :31 30 0 49 43 0

Miss 26 was not located
The contents in the cache are :26 30 0 49 43 0

Miss 37 was not located
The contents in the cache are :37 30 0 49 43 0
```

Figure 6. Simulation screenshot of the terminal after the Random Page Replacement has run.

To the left is **Figure 6**. **Figure 6** shows the cache contents during the execution of the function Random Page replacement. The contents are initially all zero. The two first two searches, 49 and 50, were misses, so they were randomly placed in the cache. When 30 was searched for and it was a miss, it overwrote the spot of 50 although there were other empty spots. Random replacement does not search the cache at all so this is an example of when an unnecessary overwrite happened.

Continuing from that point shows misses of the numbers 43,49,20,31, and 26. These numbers were randomly placed in the cache and pages were overwritten even though there was space in the cache for them. This order continues for the whole page log.

Least Recently Used Page Replacement



```
The inital contents in the cache are: 0 0 0 0 0 0

Miss 49 was not located
The contents in the cache are :49 0 0 0 0 0

Miss 50 was not located
The contents in the cache are :49 50 0 0 0 0

Hit 50 was located
The contents in the cache are :49 50 0 0 0 0

Miss 52 was not located
The contents in the cache are :49 50 52 0 0 0

Miss 30 was not located
The contents in the cache are :49 50 52 30 0 0

Hit 49 was located
The contents in the cache are :49 50 52 30 0 0

Miss 43 was not located
The contents in the cache are :49 50 52 30 43 0

Hit 49 was located
The contents in the cache are :49 50 52 30 43 0

Miss 20 was not located
The contents in the cache are :49 50 52 30 43 20

Miss 31 was not located
The contents in the cache are :49 31 52 30 43 20

Miss 26 was not located
The contents in the cache are :49 31 26 30 43 20

Miss 37 was not located
The contents in the cache are :49 31 26 37 43 20
```

Figure 7.Simulation screenshot of the terminal after the Least Recently Used Page Replacement has run.

To the left is **Figure 6**. **Figure 7** shows the cache contents during the execution of the function Least Recently Used Page replacement. The two first two searches, 49 and 50, were misses, so they were placed in the cache. 50 was searched for, and it was hit labeled as a hit since it was in the cache. For every hit, the cache resets its least recently used attribute and increments the others. The trend of miss and the pages being placed in the cache continues for 52,30,43,49 and 20.

When page 31 is searched for, the cache is full. Since it is full, it needs replace a page. Page 31 replaces Page 50 since it was the Page that was last recently used. This order continues for the whole page log

To compare the results of the three different replacement methods, the least recently used and oldest page replacement methods were run once since the results are not going to change regardless of the number of times they are run. For the random replacement method, the function was ran 50 times and the average was taken.

Figure 8 contains the number of hits and misses for the different replacement methods. The replacement method with the lowest number of hits is the random replacement with an average of 27.82. The next best replacement method is least recently used with 32 hits. The replacement method that had the largest number of hits is the oldest page replacement, with 33 hits.

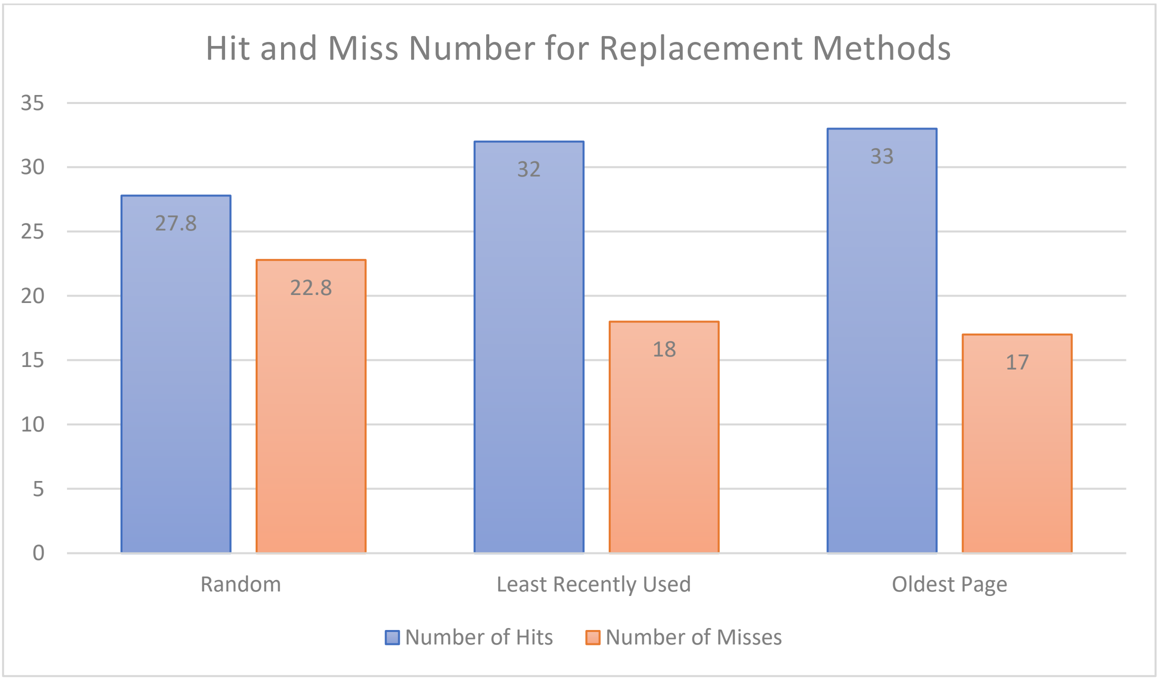


Figure 8. Figure of the number of hits and misses for the different replacement methods.

Figure 9 shows hit efficiency rate for the three different replacement methods. The average hit efficiency rate for random replacement method is 55.64%. The hit efficiency rate for least recently used and oldest page replacement are 64% and 66% respectively.

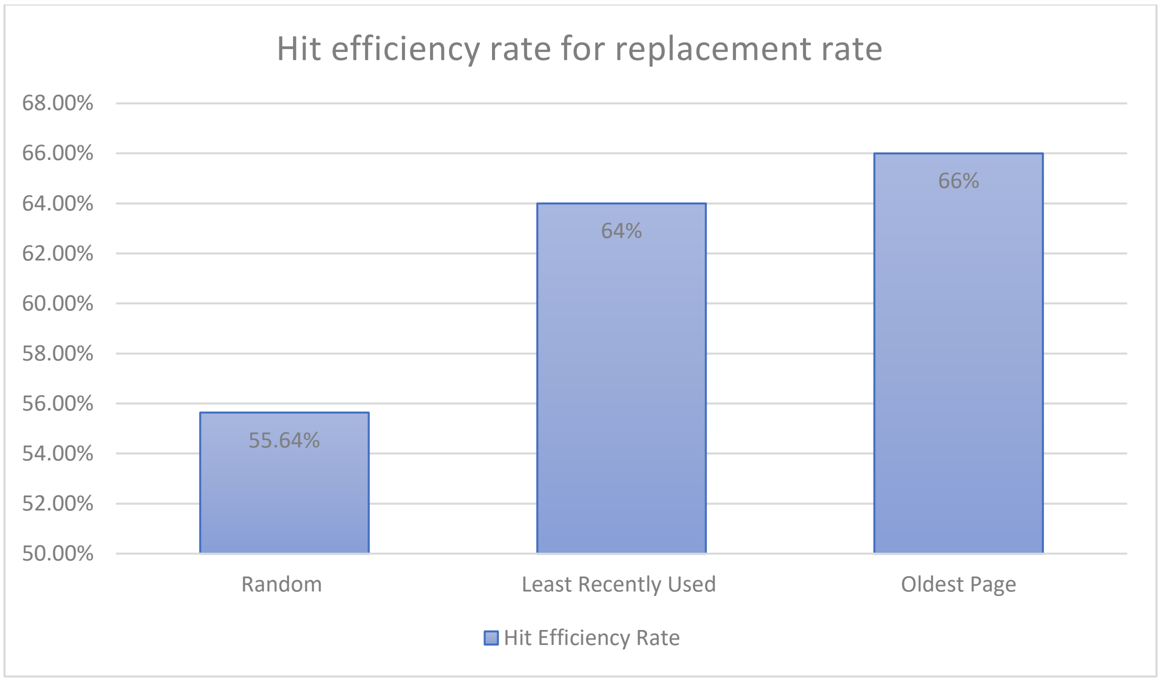


Figure 9. Hit efficiency rate for the different replacement methods.

From the findings after running the experiments, The oldest page replacement is best replacement method for the page log given from the professor. Least recently used was close with only having one less hit. The random page replacement did not do terrible as it was only 10% less efficient then the other two replacement methods.

8. Conclusion

The cache is one of the higher storage devices in the memory hierarchy. The cache is used to store temporary data that the CPU needs. Once the cache is full, pages in the cache need to be replaced to allow access to the currently requested page. The three replacement methods that were investigated were FIFO or oldest page replacement, random, and least recently accessed.

After creating the three different page replacements in the programming C++ and tested the different page replacement methods on page log provided by the professor, it was found that oldest page replacement had the best results with a 66% efficiency rate. A close second was least recently accessed with 64% and the random has the worst efficiency rate with 55.64%. It was concluded that

References

[1] Neso Academy. Cache Design – An Overview(Aug. 26, 2021). Accessed: Nov. 10 2023. [Online Video] https://www.youtube.com/watch?v=1tvW8kzOpSA&list=PLBlnK6fEyqRjdT1xkkBZSXKwFKqQoYhwy&index=22

[2] "Page Replacement Algorithms in Operating Systems" geeksforgeeks.org. https://www.geeksforgeeks.org/page-replacement-algorithms-in-operating-systems/ (Accessed Nov. 10 2023).

Appendix A

A.1 Project .cpp file :

Hit_Miss_Cache_Project.cpp

```
#include <iostream>
#include <time.h>


using namespace std;


class Page
{
   int content;
   int last_recently_used;
public:
   Page()                              // Default Constructor
   {
      content = 0;
      last_recently_used = 100;              // set to a really high value to say the data is really old


   }
   Page(int acontent)                        // Constructor
   {
      content = acontent;
      last_recently_used = 100;
   }

   int get_content()
   {
      return content;
   }

   void set_content(int acontent)
   {
      content = acontent;
   }

   void show_content()
   {
      cout << content;
   }

   void set_last_recently_used(int alast_recently_used)
```

```
    {
        last_recently_used = alast_recently_used;
    }

    int get_last_recently_used()
    {
        return last_recently_used;
    }

    void increase_LRU_Value()
    {
        last_recently_used++;
    }


};

void Random_Replacement(Page Cache[], int cache_size, Page Pagelog[], int pagelog_size,
double& hit_counter, double& miss_counter);
void LRU_Replacement(Page Cache[], int cache_size, Page Pagelog[], int pagelog_size,
double& hit_counter, double& miss_counter);
void Oldest_Page_Replacement(Page Cache[], int cache_size, Page Pagelog[], int pagelog_size,
double& hit_counter, double& miss_counter);
void print_Page_array(Page arr[], int size);
void increase_LRU_Arr(Page arr[], int size);
void reset_Cache(Page Cache[], int cache_size);

void Random_Replacement(Page Cache[], int cache_size, Page Pagelog[], int pagelog_size,
double& hit_counter, double& miss_counter)
{
    int random_location;
    int j;                          // defined it outside so i can check if the whole cache is checked


    for (int i = 0; i < pagelog_size; i++)              // for loop for the page log
    {
        for (j = 0; j < cache_size; j++)              // for loop for the cache
        {



            if (Pagelog[i].get_content() == Cache[j].get_content())
            {
                hit_counter++;
                break;
```

```
          }



          if (j == cache_size - 1)                         // if the whole cache is checked, then it is a miss
          {
             miss_counter++;
             random_location = rand() % 6;
             Cache[random_location] = Pagelog[i].get_content();

          }
       }

    }
}

void LRU_Replacement(Page Cache[], int cache_size, Page Pagelog[], int pagelog_size,
double& hit_counter, double& miss_counter)
{
   int max_LRU;                              // variable will be used to check for the max LRU, and then
it will be swapped
   int max_index;
   int j;
   for (int i = 0; i < pagelog_size; i++)                  // for loop for the page log
   {
      for (j = 0; j < cache_size; j++)                  // for loop for the cache
      {



          if (Pagelog[i].get_content() == Cache[j].get_content())
          {
             hit_counter++;
             increase_LRU_Arr(Cache, cache_size);
             Cache[j].set_last_recently_used(1);
             break;
          }



          if (j == cache_size - 1)                         // if the whole cache is checked, then it is a miss
          {
             miss_counter++;
             max_LRU = Cache[0].get_last_recently_used();    // For LRU, I look at the last
recently used
```

```
            max_index = 0;                              // value of all the pages and replace the largest
            for (int k = 1; k < cache_size; k++)            // one
            {
               if (Cache[k].get_last_recently_used() > max_LRU)
               {
                  max_LRU = Cache[k].get_last_recently_used();
                  max_index = k;
               }

            }
            increase_LRU_Arr(Cache, cache_size);
            Cache[max_index] = Pagelog[i].get_content();
            Cache[max_index].set_last_recently_used(1);



         }
      }

   }
}

void Oldest_Page_Replacement(Page Cache[], int cache_size, Page Pagelog[], int pagelog_size,
double& hit_counter, double& miss_counter)
{
   int j;
   for (int i = 0; i < pagelog_size; i++)                 // for loop for the page log
   {
      for (j = 0; j < cache_size; j++)                 // for loop for the cache
      {



         if (Pagelog[i].get_content() == Cache[j].get_content())
         {
            hit_counter++;
            break;
         }



         if (j == cache_size - 1)                        // if the whole cache is checked, then it is a miss
         {
            miss_counter++;
            for (int k = cache_size - 1; k > 0; k = k - 1)
            {
               Cache[k] = Cache[k - 1];                   // pushed all of the values one ahead
            }
            Cache[0] = Pagelog[i].get_content();



         }
      }

   }

}
void print_Page_array(Page arr[], int size)
{
   for (int i = 0; i < size; i++)
   {
      arr[i].show_content();
      cout << " ";
   }
   cout << endl << endl;
}

void increase_LRU_Arr(Page arr[], int size)
{
   for (int i = 0; i < size; i++)
   {
      arr[i].increase_LRU_Value();
   }

}

void reset_Cache(Page Cache[], int cache_size)
{
   for (int i = 0; i < cache_size; i++)
   {
      Cache[i].set_content(0);
   }
}

int main()
{
                                             // miss and hit counter rate variables
   double miss_counter_random = 0;
   double hit_counter_random = 0;
   double hit_count_random_sum = 0;
   double miss_count_random_sum = 0;
   double hit_count_random_avg;
   double miss_count_random_avg;

   double miss_counter_LRU = 0;
   double hit_counter_LRU = 0;

   double miss_counter_oldest = 0;
   double hit_counter_oldest = 0;

                                           // Percentage of hit

   double random_percentage = 0;
   double LRU_percentage = 0;
   double oldest_percentage = 0;

   const int cache_size = 6;
   const int pagelog_size = 50;


   Page Cache[cache_size];
   Page Pagelog[pagelog_size];

   srand(time(nullptr));

                                           // Page log provided by the professor in the assignemnt
   Pagelog[0].set_content(49);
   Pagelog[1].set_content(50);
   Pagelog[2].set_content(50);
   Pagelog[3].set_content(52);
   Pagelog[4].set_content(30);
   Pagelog[5].set_content(49);
   Pagelog[6].set_content(43);
   Pagelog[7].set_content(49);
   Pagelog[8].set_content(20);
   Pagelog[9].set_content(31);

   Pagelog[10].set_content(26);
   Pagelog[11].set_content(37);
   Pagelog[12].set_content(43);
   Pagelog[13].set_content(43);
   Pagelog[14].set_content(43);
   Pagelog[15].set_content(43);
```

```
Pagelog[16].set_content(43);
Pagelog[17].set_content(20);
Pagelog[18].set_content(20);
Pagelog[19].set_content(30);

Pagelog[20].set_content(21);
Pagelog[21].set_content(25);
Pagelog[22].set_content(25);
Pagelog[23].set_content(25);
Pagelog[24].set_content(25);
Pagelog[25].set_content(25);
Pagelog[26].set_content(25);
Pagelog[27].set_content(31);
Pagelog[28].set_content(20);
Pagelog[29].set_content(20);

Pagelog[30].set_content(11);
Pagelog[31].set_content(11);
Pagelog[32].set_content(11);
Pagelog[33].set_content(12);
Pagelog[34].set_content(12);
Pagelog[35].set_content(49);
Pagelog[36].set_content(43);
Pagelog[37].set_content(50);
Pagelog[38].set_content(50);
Pagelog[39].set_content(20);

Pagelog[40].set_content(11);
Pagelog[41].set_content(43);
Pagelog[42].set_content(50);
Pagelog[43].set_content(12);
Pagelog[44].set_content(12);
Pagelog[45].set_content(49);
Pagelog[46].set_content(43);
Pagelog[47].set_content(50);
Pagelog[48].set_content(50);
Pagelog[49].set_content(50);



// Tested random replacement 10 different times and took the average

for (int i = 0; i < 50; i++)
{
        Random_Replacement(Cache, cache_size, Pagelog, pagelog_size, hit_counter_random,
miss_counter_random);
        hit_count_random_sum = hit_counter_random + hit_count_random_sum;
        miss_count_random_sum = miss_counter_random + miss_count_random_sum;
        hit_counter_random = 0;
        miss_counter_random = 0;
        reset_Cache(Cache, cache_size);

    }

    hit_count_random_avg = hit_count_random_sum / 50;
    miss_count_random_avg = miss_count_random_sum / 50;

    cout << "The amount of hits for random replacement are " << hit_count_random_avg << endl;
    cout << "The amount of misses for random replacement are " << miss_count_random_avg <<
endl;

    random_percentage = (hit_count_random_avg / 50) * 100;

    cout << "The percentage for a hit using random replacement is " << random_percentage <<
"%." << endl;


    reset_Cache(Cache, cache_size);

    cout << endl;

    LRU_Replacement(Cache, cache_size, Pagelog, pagelog_size, hit_counter_LRU,
miss_counter_LRU);

    cout << "The amount of hits for LRU replacement are: " << hit_counter_LRU << endl;
    cout << "The amount of misses for LRU replacement are: " << miss_counter_LRU << endl;

    LRU_percentage = (hit_counter_LRU / 50) * 100;

    cout << "The percentage for a hit using random replacement is " << LRU_percentage << "%."
<< endl;

    cout << "The final contents inside the cache are after LRU replacement: ";
    print_Page_array(Cache, cache_size);


    reset_Cache(Cache, cache_size);

    cout << endl;
    Oldest_Page_Replacement(Cache, cache_size, Pagelog, pagelog_size, hit_counter_oldest,
miss_counter_oldest);

    cout << "The amount of hits for oldest replacement are: " << hit_counter_oldest << endl;
    cout << "The amount of misses for oldest replacement are: " << miss_counter_oldest << endl;

    oldest_percentage = (hit_counter_oldest / 50) * 100;

    cout << "The percentage for a hit using random replacement is " << oldest_percentage <<
"%." << endl;

    cout << "The final contents inside the cache are after Oldest Page replacement: ";
    print_Page_array(Cache, cache_size);

    return 0;
}
```