2016

Operating Systems

CACHE MANAGEMENT

Professor

Sanjay Chaudhary

Mentor

Mayank Jobanputra

Group 17

Group Members

Amee Bhuva (1401009)

Subhashi Dobariya (1401012)

Twinkle Vaghela (1401106)

Himani Patel (1401111)

Table Of Contents

BRIEF DESCRIPTION

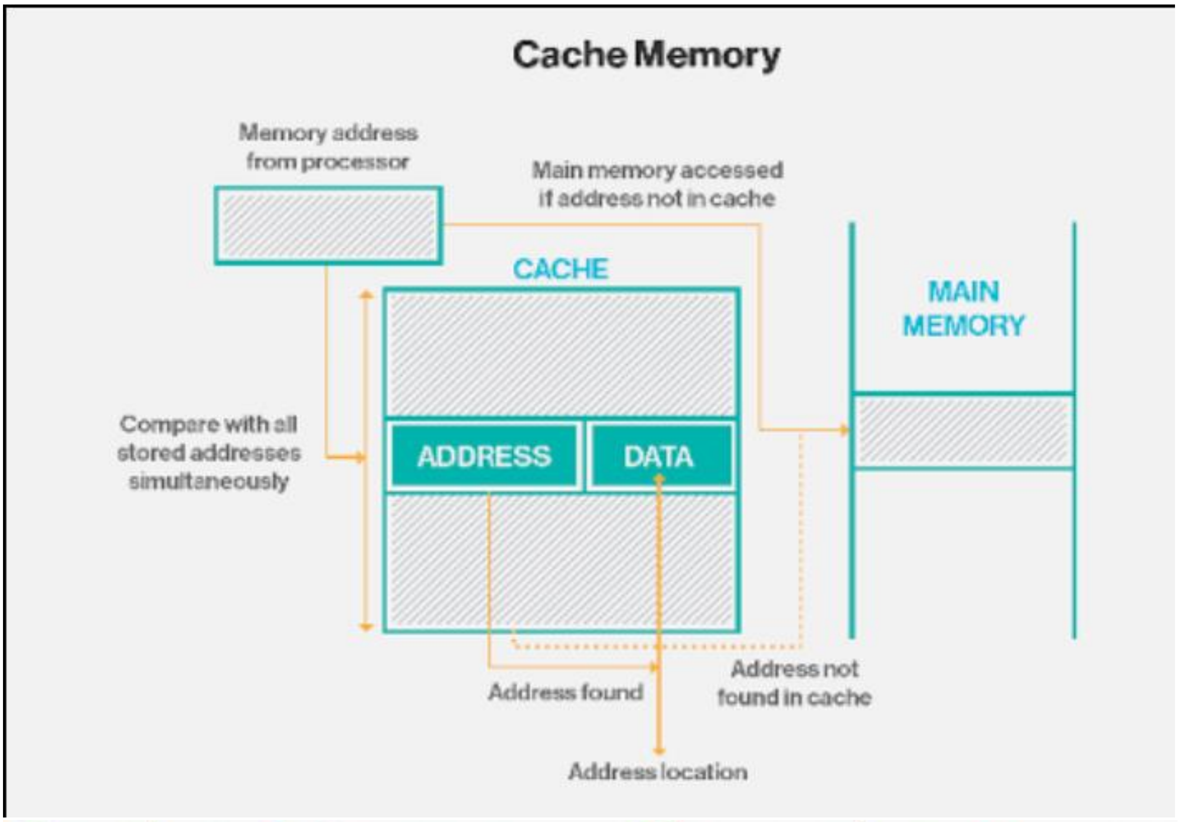
- Memory Management in operating system includes management of files, database, recently visited files etc. One of its parts is cache management. Cache is the file having history of recently used files, links, software etc. There are different types of cache like web cache, database cache etc. Web cache includes recently used files in web browser while database cache includes recently used and opened files and software history in the operating system.

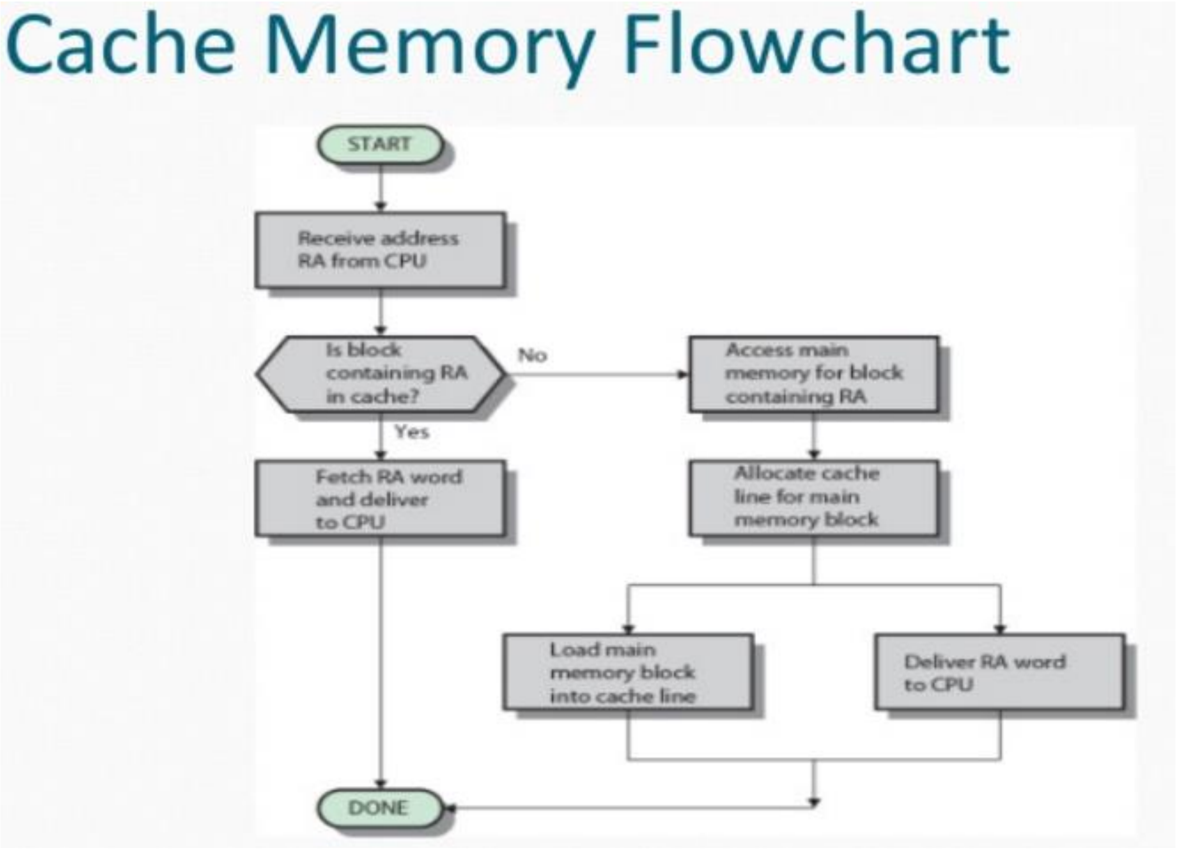
- Cache files should be managed as the cache memory is very limited and less than the main memory. Whenever it gets full, the cache files that are in the cache memory should be removed. There are many types of algorithm to do so. In this project we are going to manage those cache files according to the frequency of the same. When the cache memory gets full the least recently used file will be deleted from the cache memory by calculating age bits. Age bits decide the age of the file that is how older the file is. Also the file that is running infinitely which is unwanted and of no use that will also be deleted.

- This kind of algorithm will work because concept of cache files is useful for most frequently used files and links. The files which are not frequently used or the file that is running infinitely does not make any sense in the cache memory. It unnecessarily occupies the memory. To store the data in cache memory there are different types of mapping scheme. Here direct mapping as well as Associative mapping is useful.

- This project is useful for cache management in operating system as well as web as cache files can be managed and we get rid from cache memory getting overloaded.

ARCHITECTURE


Cache Memory
Memory address from processor
Main memory accessed if address not in cache
CACHE
MAIN MEMORY
Compare with all stored addresses simultaneously
ADDRESS
DATA
Address found
Address not found in cache
Address location


Cache Memory Flowchart
START
Receive address RA from CPU
Is block containing RA in cache?
No
Yes
Access main memory for block containing RA
Fetch RA word and deliver to CPU
Allocate cache line for main memory block
Load main memory block into cache line
Deliver RA word to CPU
DONE

TECHNICAL SPECIFICATIONS

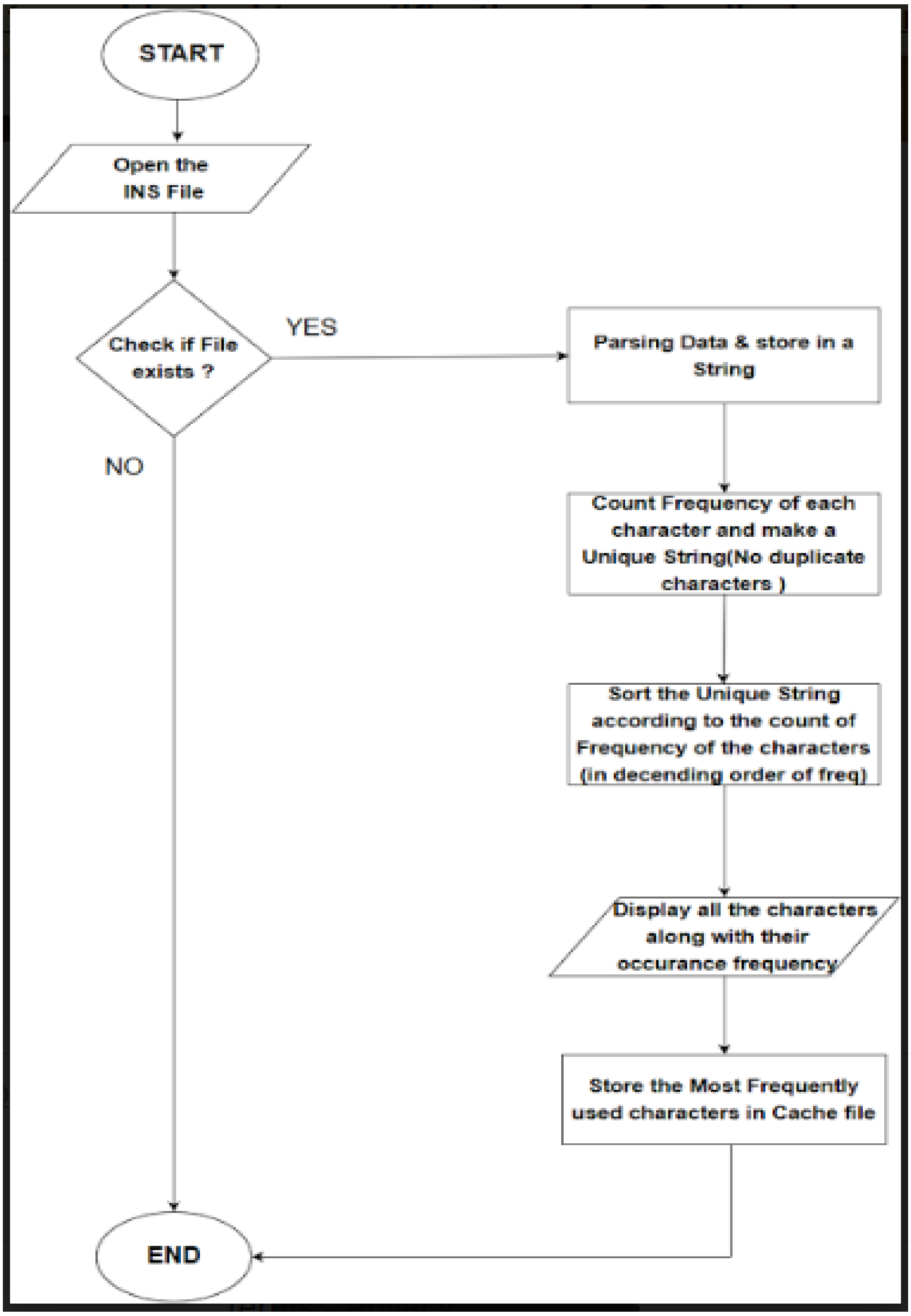
- **Hardware requirements :**
  1. Processor : Intel Core i3/i5
  2. RAM : 2 GB*
  3. HDD : 5 GB*

  (* : May change dynamically)
- **Software requirements :**
  Windows 7/8/8.1/10 , Ubuntu
- We have limited our cache size to 10 bytes, that is, only five variables can be stored at first, after that, the variables will be replaced as per the algorithm implemented.

ALGORITHMS/FLOWCHARTS

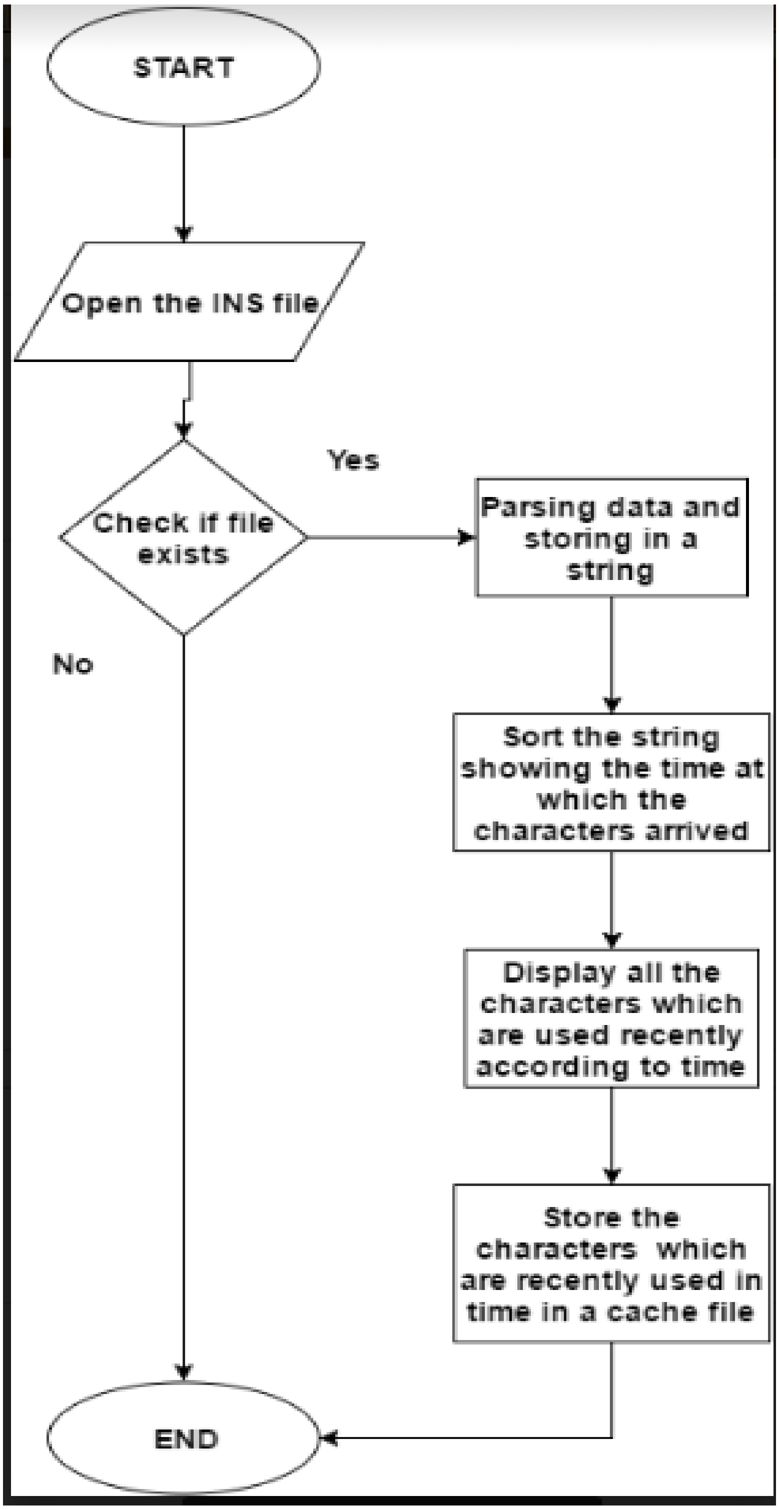
1. MFU

- The system keeps track of number of times a block is referenced in main memory
- Discards the most frequently used items first
- The element which has the largest count would be deleted first


START
Open the INS File
Check if File exists ?
YES
NO
Parsing Data & store in a String
Count Frequency of each character and make a Unique String(No duplicate characters)
Sort the Unique String according to the count of Frequency of the characters (in decending order of freq)
Display all the characters along with their occurance frequency
Store the Most Frequently used characters in Cache file
END

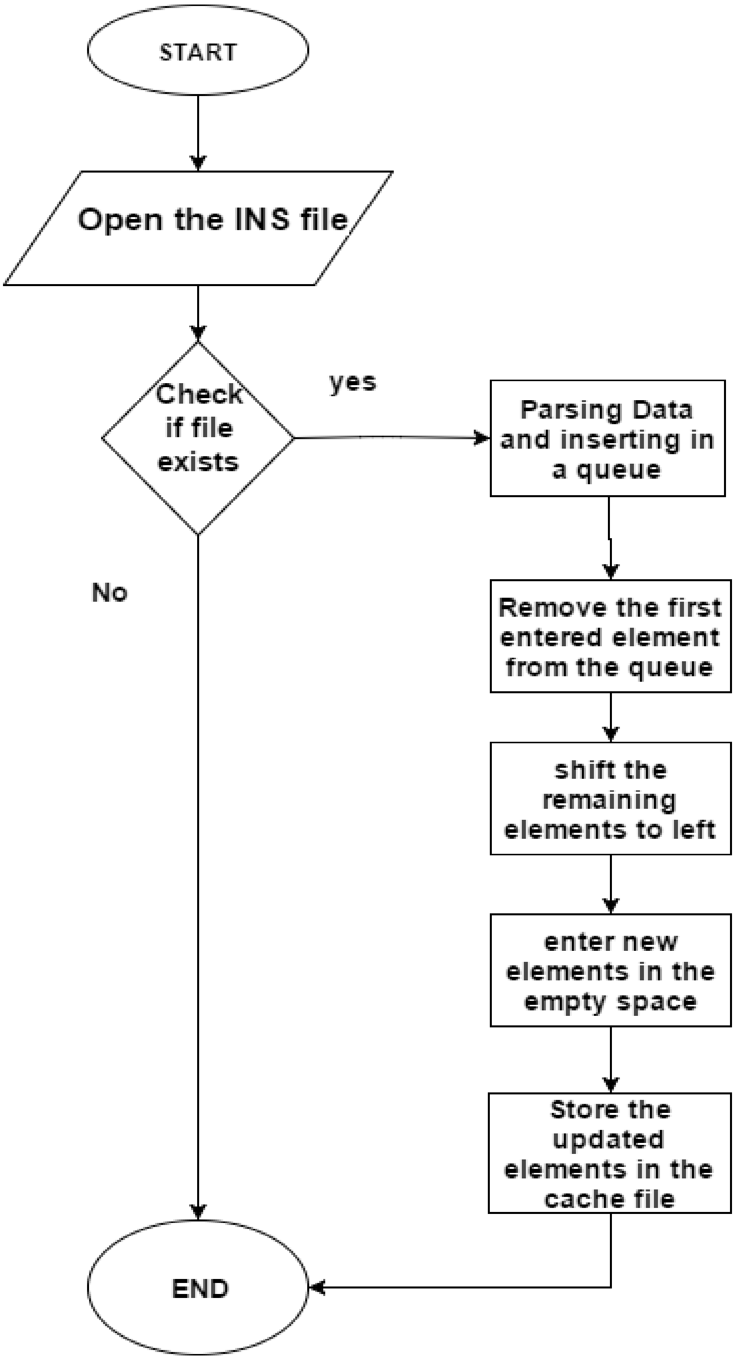
2. MRU

- MRU algorithms are most useful in situations where the older an item is, the more likely it is to be accessed.
- This algorithm discards the most recently used items first.


START
Open the INS file
Check if file exists
Yes
No
Parsing data and storing in a string
Sort the string showing the time at which the characters arrived
Display all the characters which are used recently according to time
Store the characters which are recently used in time in a cache file
END

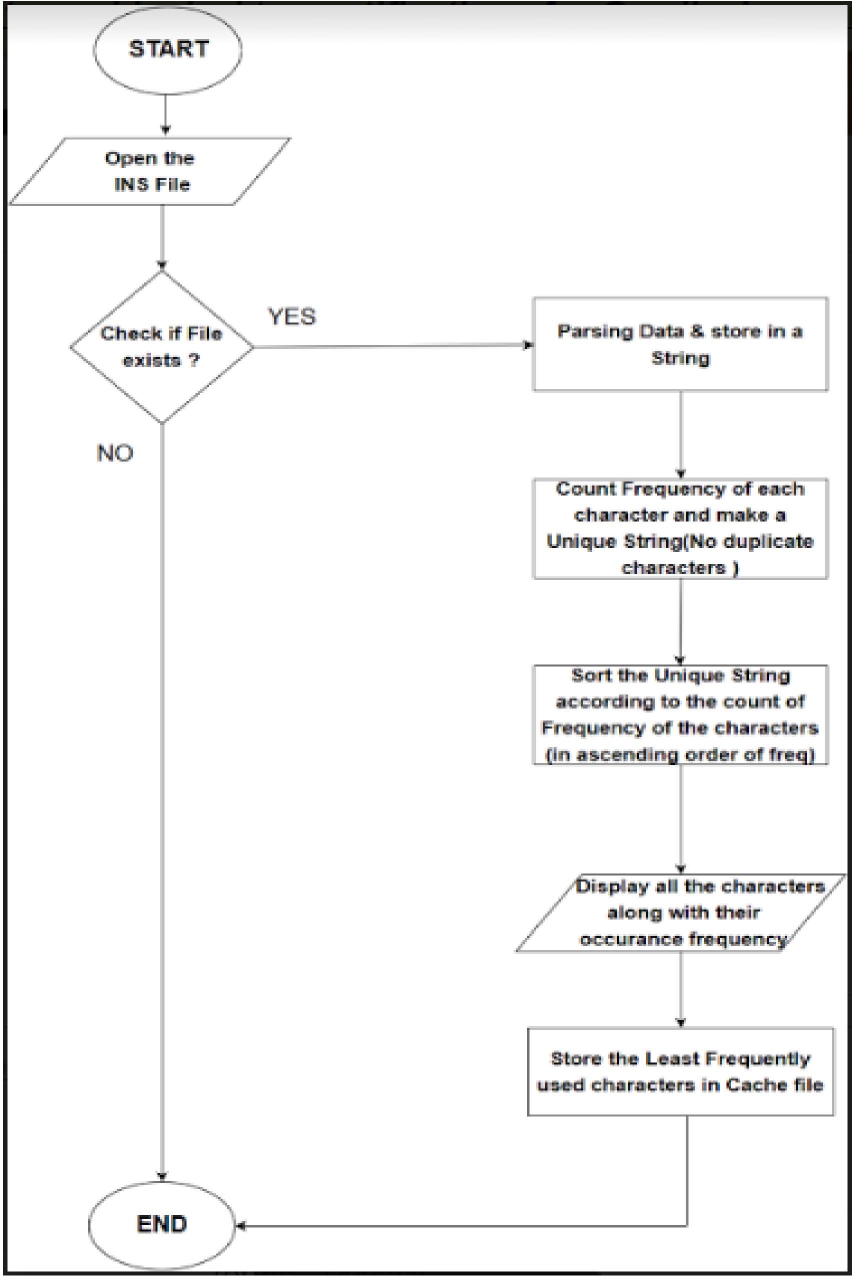
3. FIFO

- Using this algorithm the cache behaves in the same way as a FIFO queue.
- The idea is obvious from the name – the operating system keeps track of all the elements in the cache memory in a queue, with the most recent arrival at the back, and the oldest arrival in front
- The cache evicts the first block accessed first without any regard to how often or how many times it was accessed before.


START
Open the INS file
Check if file exists
yes
No
Parsing Data and inserting in a queue
Remove the first entered element from the queue
shift the remaining elements to left
enter new elements in the empty space
Store the updated elements in the cache file
END

4. LFU

- LFU is a type of cache algorithm used to manage memory within a computer. The standard characteristics of this method involve the system keeping track of the number of times a block is referenced in memory. When the cache is full and requires more room the system will remove the unwanted items with the lowest reference frequency
- This basically counts how frequently an item is needed
- Those that are used least often are discarded first
- E.g., if A was used (accessed) 5 times and B was used 3 times and others C and D were used 10 times each, we will replace B


START
Open the INS File
Check if File exists ?
YES
NO
Parsing Data & store in a String
Count Frequency of each character and make a Unique String(No duplicate characters)
Sort the Unique String according to the count of Frequency of the characters (in ascending order of freq)
Display all the characters along with their occurance frequency
Store the Least Frequently used characters in Cache file
END

PROGRAM

```
/*****************************************************************
*********/
// Project : Cache Managemment
// Group   : 17
// Date    : 7th December,2016
/*****************************************************************
*********/

/*****************************************************************
*********/
// GROUP MEMBERS
// 1. Amee Bhuva (1401009)
// 2. Subhashi Dobariya (1401012)
// 3. Twinkle Vaghela (1401106)
// 4. Himani Patel (1401111)
/*****************************************************************
*********/

/*****************************************************************
*********/
// ALGORITHMS
// 1. Most Recently Used (MRU)
// 2. Most Frequently Used (MFU)
// 3. First In First Out (FIFO)
// 4. Least Frequently Used (LFU)
/*****************************************************************
*********/

/*==================================*/
// HEADER FILES
/*==================================*/
#define _REENTRANT
#include <stdio.h>
#include <pthread.h>
#include <math.h>
#include <stdlib.h>
#define MAX 10
```

```
#define NUM_THREADS 4
#include <string.h>
#include <time.h>
#include <signal.h>

/*=================================*/
// GLOBAL VARIABLES
/*=================================*/
char str[100];
int number[50];
int numbers[50];
int gvar2=0;
time_t sec;
int gvar1=0;
char array[100];
long curtime[100];
char cacheStr[10];
char newStr[10];

/*=================================*/
// FUNCTIONS
/*=================================*/
void *main1(void *);
void *main2(void *);
void *main3(void *);
void *main4(void *);
void sigproc(void);
void quitproc(void);
int Parcer(char a[50]);
int sorting(int number[50]);
int findUnique(char arr[]);
int inverse(char str[]);

/*=================================*/
// STRUCTURE
/*=================================*/
typedef struct _thread_data_t {
  int tid;
  double stuff;
```

```
} thread_data_t;

/*===================================*/
// MAIN FUNCTION
/*===================================*/
int main(int argc, char *argv[])
{

  //Call sigproc and quitproc
  signal(SIGINT,sigproc);
  signal(SIGQUIT,quitproc);


printf("\n***************************************************
*****");
  printf("\nCAUTION ! Ctrl-C is disabled..!! Use Ctrl-\ to quit..\n");

printf("****************************************************
***\n");

  int i,rc, status, *status_ptr = &status;
  pthread_t thread[NUM_THREADS];

  //Create a thread_data_t argument array
  thread_data_t thr_data[NUM_THREADS];

  //This loop runs four times which is the number of theads in the program
  for (i = 0; i < NUM_THREADS; ++i)
  {
    thr_data[i].tid = i;

    //Error statement if creation of THREAD 1 fails
    if(i==0)
    {
          if ((rc = pthread_create(&thread[i], NULL, main1, &thr_data[i])))
          {
                    fprintf(stderr, "error: pthread_create, rc: %d\n", rc);
                    return EXIT_FAILURE;
          }
    }

    //Error statement if creation of THREAD 2 fails
    else if(i==1)
    {
          if ((rc = pthread_create(&thread[i], NULL, main2, &thr_data[i])))
          {
                    fprintf(stderr, "error: pthread_create, rc: %d\n", rc);
                    return EXIT_FAILURE;
          }
    }

    //Error statement if creation of THREAD 3 fails
    else if(i==2)
    {
          if ((rc = pthread_create(&thread[i], NULL, main3, &thr_data[i])))
          {
                    fprintf(stderr, "error: pthread_create, rc: %d\n", rc);
                    return EXIT_FAILURE;
          }
    }

    //Error statement if creation of THREAD 4 fails
    else if(i==3)
    {
          if ((rc = pthread_create(&thread[i], NULL, main4, &thr_data[i])))
          {
                    fprintf(stderr, "error: pthread_create, rc: %d\n", rc);
                    return EXIT_FAILURE;
          }
    }
  }

  //Block until all threads complete
  for (i = 0; i < NUM_THREADS; ++i) {
    pthread_join(thread[i], NULL);
  }

  //Infinite for loop
```

```
  for(;;);

  return EXIT_SUCCESS;
}

//THREAD 1 : Implements MFU
void *main1(void *arg)
{
        //To get tid for itself
        pthread_t tid = pthread_self();

        int i=0;
        int as =1;
        int aa=1;
        int k=0;
        char ch,a[100];
        FILE *fp,*fn;
        int j=0, n=0, count=0;
        int frequency[50];
        int count_i[26]={0},mn=0,l=0;
      int timer[100]={0},char_count=0;
      clock_t start,stop;

        //INS.txt : Opened for fetching variables
        //Cache_MFU.txt : Opened for storing cached variables
        fp = fopen("INS.txt","r");
        fn = fopen("Cache_MFU.txt","w");

        if ( fp != NULL )
        {
       //Starting the timer for keeping track of arrival of variables
       start = clock();
       ch = fgetc(fp);
         while (ch!=EOF)
        {

        if(ch >= 'a' && ch <= 'z')
        {
                        sec = time(NULL);
```

```
                    curtime[gvar1] = sec;
                    gvar1++;

                              //For printing the character and the time at which the
character has arrived
                         printf ("Character : %c\tTime : %.8f
sec\n",ch,((double)(stop - start)/CLOCKS_PER_SEC));
                              a[l]=ch;
                              l++;
                    curtime[gvar1] = sec;
                    gvar1++;
                              timer[mn]=sec;
                              count_i[ch - 'a']++;
                              mn++;
                              char_count++;
        }

        //Sleep is added so that a bit difference in the arrival of the
characters can be noticed
          usleep(50000);

        //Stop the timer
        stop = clock();

        //Get next character from the file
          ch = fgetc(fp);
    }
    printf("\nString : %s \n",a);
    findUnique(a);
    int tt;
    printf("\n-----------------------");
    printf("\nVariable     Frequency\n");
    printf("-----------------------\n");
    for(int i = 0;i < 26;i++)
    {
          if(count_i[i]>0)
          {
           for (int k = 0; k < char_count+2; ++k)
           {
            for (j = k + 1; j < char_count+2; ++j)
                    {
                     if (count_i[k] < count_i[j])
                       {
                    tt = count_i[k];
                         count_i[k] = count_i[j];
                         count_i[j] = tt;
                       }
                    }
            }
            if(i>=0 && i<5)
            {
                    fprintf(fn,"%c\n",newStr[i]);
            }
            printf("%c\t\t%d\n",newStr[i],count_i[i]);
            }
       }


            }
            else
            {
                    //Why didn't the file open?
                    perror ("INS.txt");
            }

            //Closing the previously opened files
            fclose (fp);
            fclose (fn);

            //Prints ID of the thread
            printf("\nThread ID (MRU) : %u\n",tid);
            return (void *)NULL;
            pthread_exit(NULL);
    }

    //THREAD 2 : Implements MRU
    void *main2(void *arg)
    {
```

```
  //To get tid for itself
  pthread_t tid = pthread_self();

  int i=0;
  int as =1;
  int aa=1;
  int k=0;
  char line [20],ch,a[100];
  FILE *fp,*fn;
  int j=0, n=0, count=0;
  int frequency[50];
  char txt[256], var[100][256], temp[256];
  int count_i[26]={0},mn=0,l=0;
int timer[100]={0},char_count=0;


  //INS.txt : Opened for fetching variables
  //Cache_MRU.txt : Opened for storing cached variables
  fp = fopen("INS.txt","r");
  fn = fopen("Cache_MRU.txt","w");

  if ( fp != NULL )
  {
 ch = fgetc(fp);
   while (ch!=EOF)
  {
  if(ch >= 'a' && ch <= 'z')
  {
            sec = time(NULL);
       curtime[gvar1] = sec;
       gvar1++;
            a[l]=ch;
            l++;
       curtime[gvar1] = sec;
       gvar1++;
            timer[mn]=sec;
            count_i[ch - 'a']++;
            mn++;
            char_count++;
            }
              //For proper synchronisation between threads
              usleep(60000);

              //Get next character from the file
              ch = fgetc(fp);
        }
        findUnique(a);
        int tt;
        for(int i = 0;i < 26;i++)
        {
              if(count_i[i]>0)
              {
               for (int k = 0; k < char_count+2; ++k)
               {
              for (j = k + 1; j < char_count+2; ++j)
                        {
                         if (count_i[k] < count_i[j])
                           {
                        tt = count_i[k];
                             count_i[k] = count_i[j];
                             count_i[j] = tt;
                           }
                        }
              }
              }
        }
              }
              else
              {
                        //Why didn't the file open?
                        perror ("INS.txt");
              }

              inverse(newStr);
              for(int i=0;i<6;i++)
              {
                        if(return_size(fn)<10)
                        {
```

```
                        fprintf(fn,"%c\n",cacheStr[i]);
                    }
                }

                //Closing the previously opened files
                fclose (fp);
                fclose (fn);

                //Prints ID of the thread
                printf("\nThread ID (MFU): %u\n",tid);
                return (void *)NULL;
                pthread_exit(NULL);
        }

        //THREAD 3 : Implements FIFO
        void *main3(void *arg)
        {
                pthread_t tid = pthread_self();     /* to get tid for itself */
                char ch,a[5];
                FILE *fp,*fn;
                char txt[256], var[100][256];
                int i=0,j;
                char temp;

                //INS.txt : Opened for fetching variables
                //Cache_FIFO.txt : Opened for storing cached variables
                fp = fopen("INS.txt","r");
                fn = fopen("Cache_FIFO.txt","w");

                //For proper synchronisation between the threads
                sleep(3);

                if ( fp != NULL )
                {
                 //Get next character from the file
               ch = fgetc(fp);
                 while (ch!=EOF)
                {
                if(ch >= 'a' && ch <= 'z')
```

```
    {
        if(i<5)
        {
            a[i]=ch;
            i++;
        }
        else if(i==5)
        {
                //Checks whether the arrived character exists in
the cached data
            if(a[0]!=ch && a[1]!=ch && a[2]!=ch && a[3]!=ch
&& a[4]!=ch)
            {
                for(j=0;j<5;j++)
                {
                    temp=a[0];
                    a[j]=a[j+1];
                }
                a[4]=ch;
            }
        }
     }
            //For proper synchronisation between threads
            usleep(50000);
            ch = fgetc(fp);
    }
    printf("This is FIFO : %s",a);
  }
     else
    {
        //Why didn't the file open?
        perror ("INS.txt");
    }
    for(i=0;i<5;i++)
    {
        fprintf(fn,"%c\n",a[i]);
    }

    //Closing the previously opened files
```

```
        fclose (fp);
        fclose (fn);

        //Prints ID of the thread
        printf("\nThread ID (FIFO): %u\n",tid);

        return (void *)NULL;
        pthread_exit(NULL);
}

//THREAD 4 : Implements LFU
void *main4(void *arg)
{
         //To get tid for itself
        pthread_t tid = pthread_self();

        int i=0;

        char ch,a[100];
        FILE *fp,*fn,*fh;
        int j=0, count=0;

        int count_i[26]={0};
        int k=0,mn=0,l=0;
      int timer[100]={0},char_count=0;


        //INS.txt : Opened for fetching variables
        //Cache_LFU.txt : Opened for storing cached variables
        fp = fopen("INS.txt","r");
        fn = fopen("Cache_LFU.txt","w");
        fh = fopen("Cache.txt","w");

        //For proper synchronisation between threads
        sleep(4);

        if ( fp != NULL )
        {
                while ((ch = fgetc(fp))!=EOF)
```

```
          {
                  //for parsing the line of the file
                  if((ch >= 'a' && ch <= 'z') )
                  {
                          sec = time(NULL);
                          curtime[gvar1++] = sec;
                          a[char_count++]=(char)ch;
                          curtime[gvar1++] = sec;
                          timer[mn++]=sec;
                          count_i[ch - 'a']++;
                  }

          }
          a[char_count]='\0';
          //Printing the characters
       printf("\nString : %s \n",a);
       findUnique(a);
       //Printing the characters without duplicate
       printf("\nString : %s \n",newStr);
       int tt;

       printf("\n----------------------");
       printf("\nVariable     Frequency\n");
       printf("----------------------");

          for ( k = 0; k < strlen(newStr); k++)
          {
                  for (j = k+1; j < strlen(newStr); j++)
                  {
                          if(count_i[newStr[k]-97]>count_i[newStr[j]-
97])
                          {
                                  tt=newStr[k];
                                  newStr[k]=newStr[j];
                                  newStr[j]=(char)tt;
                          }
                  }
          }
```

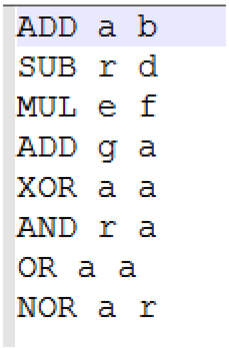
```
                //printing the first five least frequency words in the cache
file.
                for( i = 0;i < strlen(newStr);i++)
                {
                        if(i>=0 && i<5)
                        {
                                fprintf(fn,"%c\n",newStr[i]);
                        }
                        printf("\n %c%15d ",newStr[i],count_i[newStr[i]-97]);
                }

                //printing the words other than the least frequency words in
the cache file.
                for( i = 0;i < strlen(newStr);i++)
                {
                        if(i>=5)
                        {
                                fprintf(fh,"%c\n",newStr[i]);
                        }

                }

        }
        else
        {
                //Why didn't the file open?
                perror ("INS.txt");
        }

        //Closing the previously opened files
        fclose (fp);
        fclose (fn);
        fclose (fh);

        //Prints ID of the thread
        printf("\nThread ID (LFU): %u\n",tid);

        return (void *)NULL;
        pthread_exit(NULL);
```

```
}

/*====================================*/
// FUNCTION FOR INVERTING STRING
/*====================================*/
int inverse(char str[25])
{
          //length of the string
          int a = strlen(str);
          int i=0,k=0;

          for(i=a-1;i>=0;i--)
          {
                    //inverting array
                    cacheStr[k]=str[i];
                    k++;
          }
          printf("Inversed String: %s\n",cacheStr);

return 0;
}

/*====================================*/
// FUNCTION THAT RETIRNS FILE SIZE
/*====================================*/
int return_size(FILE *f)
{
          int size=0;
          size=ftell(f);
          return size;
}

/*====================================*/
// FUNCTION TO REMOVE DUPLICATES
/*====================================*/
int findUnique(char str[50])
{
          int i=0,j=0;
          int k=0;
```

```
    int count=0;
    int a = strlen(str);

    //removing all repeated variables
    for(i=0;i<a;i++)
    {
        if(str[i]==' ')
        {
            i++;
        }

        //including non repeated variables
        for(j=0;j<i;j++)
        {
            if(str[i]==str[j])
            {
                count++;
            }
        }
        if(count==0)
        {
            newStr[k] = str[i];
            k++;
        }
        count=0;
    }

    //printing unique variables (characters in string)
    printf("Unique String is: %s\n",newStr);
    return 0;
}

/*==================================*/
// FUNCTIONS FOR BLOCKING Ctrl+C
/*==================================*/
void sigproc()
{           signal(SIGINT, sigproc); /*  */
            /* NOTE some versions of UNIX will reset signal to default
            after each call. So for portability reset signal each time */
```

```
                printf("Oops ! You've pressed Ctrl-c..! PRESS Cntrl-\ to
EXIT...!!! \n");
}

void quitproc()
{               //printf("ctrl-\\ pressed to quit\n");
                exit(0); /* normal exit status */
}
```

TEST DATA SET



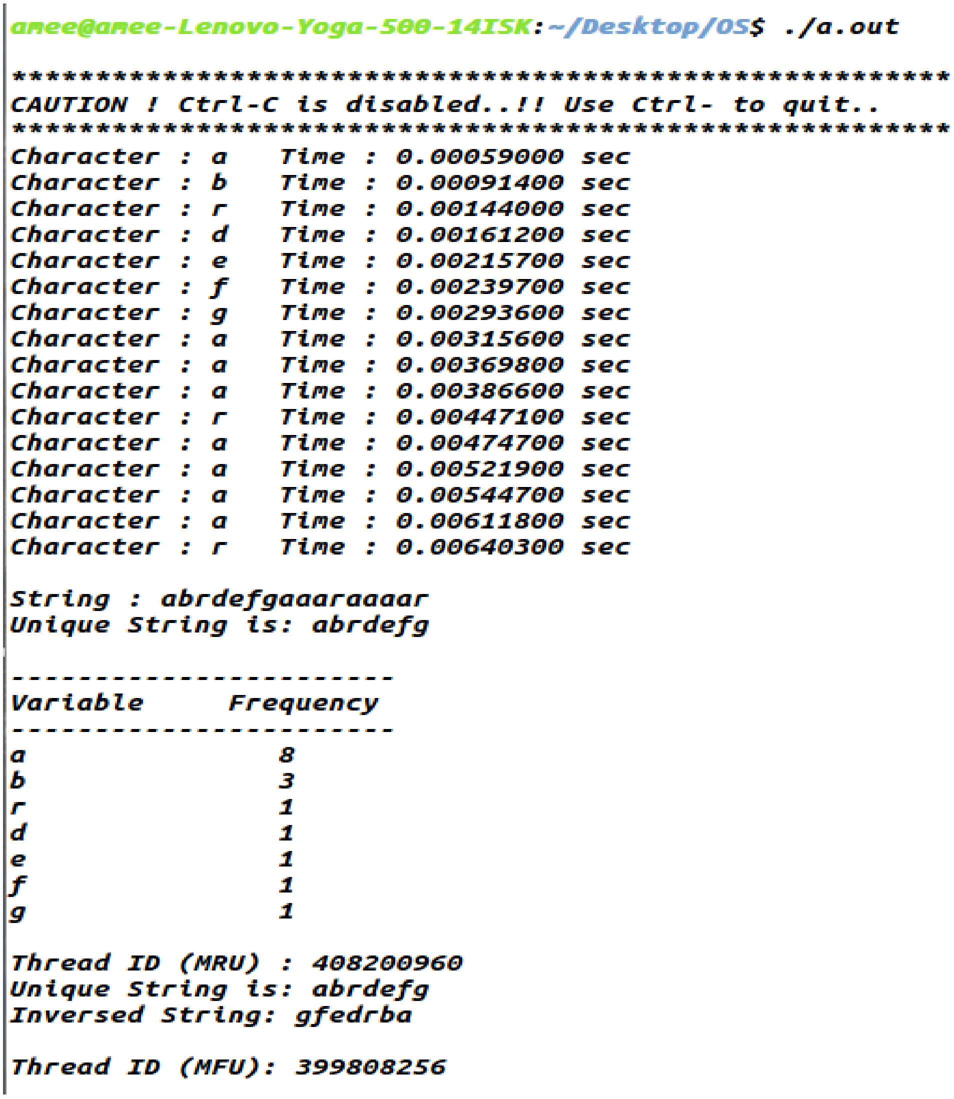
```
ADD a b
SUB r d
MUL e f
ADD g a
XOR a a
AND r a
OR a a
NOR a r
```

IMPLEMENTATION

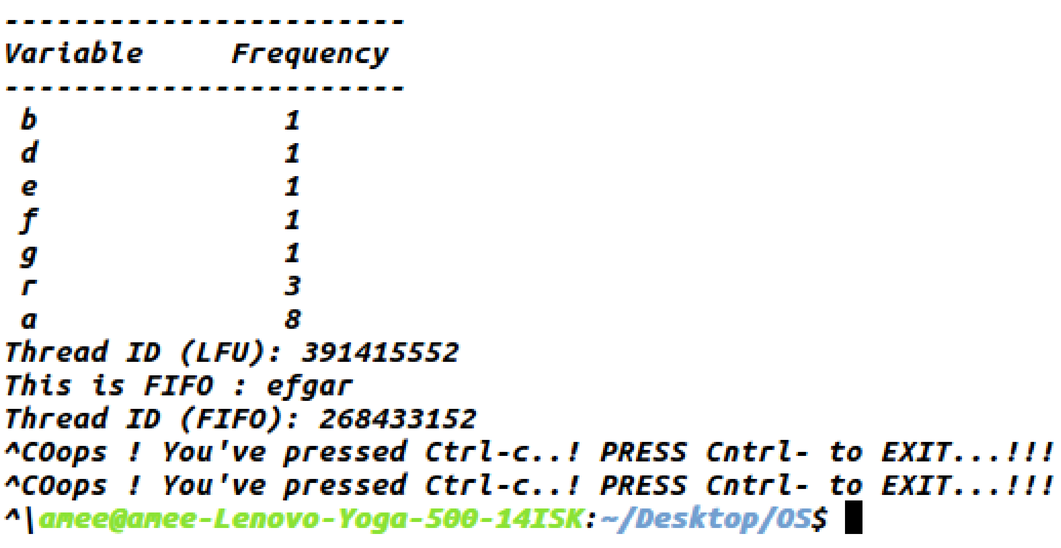
- Here in this project, we have implemented cache memory management using four different algorithms, that are : MRU. MFU, FIFO and LFU.
- All of the four algorithms run parallel in four different threads and generate four different cache files, each for one algorithm.
- Apart from this we have managed to implement graceful exit for the program as well as disabled Ctrl+C as a concept of signalling.

TEST RESULTS

- **Output Screen (Terminal)**

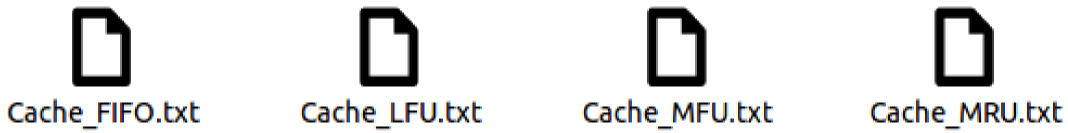


```
amee@amee-Lenovo-Yoga-500-14ISK:~/Desktop/OS$ ./a.out

*********************************************************
CAUTION ! Ctrl-C is disabled..!! Use Ctrl- to quit..
*********************************************************
Character : a   Time : 0.00059000 sec
Character : b   Time : 0.00091400 sec
Character : r   Time : 0.00144000 sec
Character : d   Time : 0.00161200 sec
Character : e   Time : 0.00215700 sec
Character : f   Time : 0.00239700 sec
Character : g   Time : 0.00293600 sec
Character : a   Time : 0.00315600 sec
Character : a   Time : 0.00369800 sec
Character : a   Time : 0.00386600 sec
Character : r   Time : 0.00447100 sec
Character : a   Time : 0.00474700 sec
Character : a   Time : 0.00521900 sec
Character : a   Time : 0.00544700 sec
Character : a   Time : 0.00611800 sec
Character : r   Time : 0.00640300 sec

String : abrdefgaaaraaaar
Unique String is: abrdefg

-----------------------
Variable     Frequency
-----------------------
a               8
b               3
r               1
d               1
e               1
f               1
g               1

Thread ID (MRU) : 408200960
Unique String is: abrdefg
Inversed String: gfedrba

Thread ID (MFU): 399808256
```

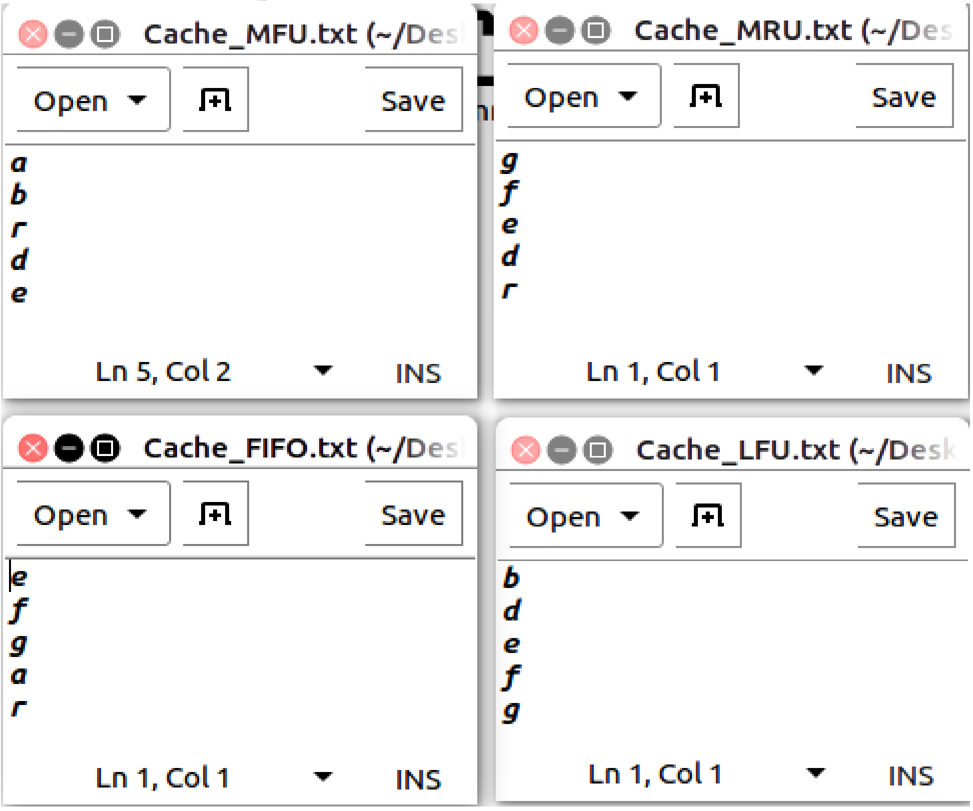


```
-----------------------
Variable     Frequency
-----------------------
 b               1
 d               1
 e               1
 f               1
 g               1
 r               3
 a               8
Thread ID (LFU): 391415552
This is FIFO : efgar
Thread ID (FIFO): 268433152
^COops ! You've pressed Ctrl-c..! PRESS Cntrl- to EXIT...!!!
^COops ! You've pressed Ctrl-c..! PRESS Cntrl- to EXIT...!!!
^\amee@amee-Lenovo-Yoga-500-14ISK:~/Desktop/OS$ █
```

- **Files Created**



- **Cache Files Output**



Cache_MFU.txt:

```
a
b
r
d
e
```

Cache_MRU.txt:

```
g
f
e
d
r
```

Cache_FIFO.txt:

```
e
f
g
a
r
```

Cache_LFU.txt:

```
b
d
e
f
g
```

REFERENCES

1. N. Ahmed, N. Mateev, and K. Pingali. Tiling Imperfectly{Nested Loop Nests. In Proc. of the ACM/IEEE Supercomputing Conference, Dallas, Texas, USA, 2000.
2. R. Allen and K. Kennedy. Optimizing Compilers for Modern Architectures. Morgan Kaufmann Publishers, San Francisco, California, USA, 2001.
3. M. Altieri, C. Becker, and S. Turek. On the Realistic Performance of Linear Algebra Components in Iterative Solvers. In H.-J. Bungartz, F. Durst, and C. Zenger, editors, High Performance Scientific and Engineering Computing, Proc. of the Int. FORTWIHR Conference on HPSEC, volume 8 of LNCSE, pages 3{12. Springer, 1998.
4. 4. B.S. Andersen, J.A. Gunnels, F. Gustavson, and J. Wa_sniewski. A Recursive Formulation of the Inversion of Symmetric Positive Definite Matrices in Packed Storage Data Format. In Proc. of the 6th Int. Conference on Applied Parallel Computing, volume 2367 of LNCS, pages 287{296, Espoo, Finland, 2002. Springer.
5. E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz,.
6. An Overview of Cache Optimization Techniques and Cache Aware Numerical Algorithms.
7. Wikipedia