JUDICIAL CASE MANAGER SYSTEM USING VIRTUAL MEMORY

A Project Report by -

Kulvir Singh (19BCE2074)

Madhur Goel (19BCE2100)

Course Code : CSE2005
Course Title : Operating Systems

Under the guidance of
Prof. Deepa K
VIT, Vellore

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

JUNE, 2021

Table of Contents

1.Abstract

Segregating Judicial Cases on the basis of their importance and gives a storage location for these cases which are not yet active in the court.

Writing a program that translates file number(logical) to storage location(physical addresses) for a virtual address space of size 2^16 = 65,536 bytes. Program will read from a file containing file numbers(logical addresses) and using a TLB as well as a page table, will translate these address to its corresponding physical address which are not yet assigned to the court.

The files will be read from file addresses.txt, which contains 1000 logical addresses ranging from 0 to 65535. Apart from translating each logical address to a physical address, it determines the contents of the signed byte stored at the correct physical address.

The program is to output the following values:

1. The logical address being translated (the integer value being rd. from addresses.txt).

2. The corresponding physical address (what your program translates the logical address to).

3. The signed byte value stored at the translated physical address.

2.Introduction

Virtual memory is a memory management ability of an OS that uses hardware and software to let a computer to compensate for physical memory shortages by temporarily transferring data from RAM to disk storage. Physical memory is the only memory that is directly accessible to the Central Processing Unit. Central Processing Unit reads the instructions stored in the physical memory and executes them constantly. The data that is operated will also be stored in physical memory in even manner. Random access memory is physical memory that grips the applications, document and procedures on a computer whereas Virtual Memory is a storage area that gives place to the files on hard drive for the retrieval when a computer runs out of RAM. Virtual memory manager maps logical addresses to physical addresses. The TLB keeps record of the recent transactions of virtual memory to physical memory. A page table is the data structure used by virtual memory system to record the mapping between virtual addresses and physical addresses.

3.Literature Survey

3.1 Virtual Memory

Virtual memory was developed to automate the movement of program code and data between main memory and secondary storage to give the appearance of a single large store. This technique greatly simplified the programmer's job, particularly when program code and data exceeded the main memory's size. Virtual memory has now become widely used, and most modern processors have hardware to support it. Unfortunately, there has not been much agreement on the form that this support should take. The result of this lack of agreement is that hardware mechanisms are often completely incompatible. Thus, designers and porters of system-level software have two somewhat unattractive choices: they can write software to fit many different architectures or they can insert layers of software to emulate a particular hardware interface.

The first version of the Microsoft Windows operating system introduced a method of managing dynamic memory based on a single global heap, which all applications and the system share, and multiple, private local heaps, one for each application. Local and global memory management functions were also provided, offering extended features for this new memory management system. More recently, the Microsoft C run-time (CRT) libraries were modified to include capabilities for managing these heaps in Windows using native CRT functions such as malloc and free. Consequently, developers are now left with a choice— learn the new application programming interface (API) provided as part of Windows or stick to the portable, and typically familiar, CRT functions for managing memory in applications written for Windows.

The virtual memory management functions provide capabilities for applications to alter the state of pages in the virtual address space. An application can change the type of memory from committed to reserved or change the protection from PAGE_READWRITE to PAGE_READONLY to prevent access to a region of addresses. An application can lock a page into the working set for a process to minimize paging for a critical page of memory. The virtual memory functions are considered low-level functions, meaning they are relatively fast but they lack many high-level features.

3.2 Translation Lookaside Buffer (TLB)

High-speed cache is about up for page table entries called a Translation Lookaside Buffer (TLB). Translation Lookaside Buffer (TLB) is nothing but a special cache used to keep track of recently used transactions. TLB contains page table entries that are last used. Given a virtual address, the processor examines the TLB if a page table entry is present (TLB hit), the frame number is retrieved and thus the 000 address is made. If a page table entry isn't found within the TLB (TLB miss), the paging is used as index while processing page table. TLB first checks if the page is already in main memory, if not in main memory a page fault is issued then the TLB is updated to include the new page entry.

3.3 Judicial System in India

Every one is very well aware of the kind of mess that India’s judicial system has become. Cases are pending from tens of years, and many of them don’t even have their files/record anymore.

Management of such files seems to be very unorganized. Ranging from police stations to courts at all levels, the types of primitive methods used to store files is very traditional and therefore, outdated.

The program aims to curb this technological gap by introducing the concept of virtual memory to generate storage locations for files, based on their activity status in the courts.

4.Requirements

SOFTWARE REQUIREMENTS:

• Ubuntu OS

• GCC Compiler (Terminal or Third-party C++ IDE)

HARDWARE REQUIREMENTS:

• A system running Ubuntu OS

• Physical RAM of at least 2GB

MODULES OF THE PROJECT

• Architecture of Memory Translation

• Logical Diagram representing the Architecture of Memory Translation

• Algorithm of the C++ code

• C++ code and Test File for Judicial Case Management

5.System Architecture

Architecture of Memory Translations

The program will read a file containing several 32-bit integer numbers that represent logical addresses. However, we were only concerned with 16-bit addresses, so we masked the rightmost 16 bits of each logical address. These 16 bits were divided into:

1. An 8-bit page number and

2. 8-bit page offset.

Other specifics include the following:

• 2^8 entries in the page table

• Page size of 2^8 bytes

• 16 entries in the TLB

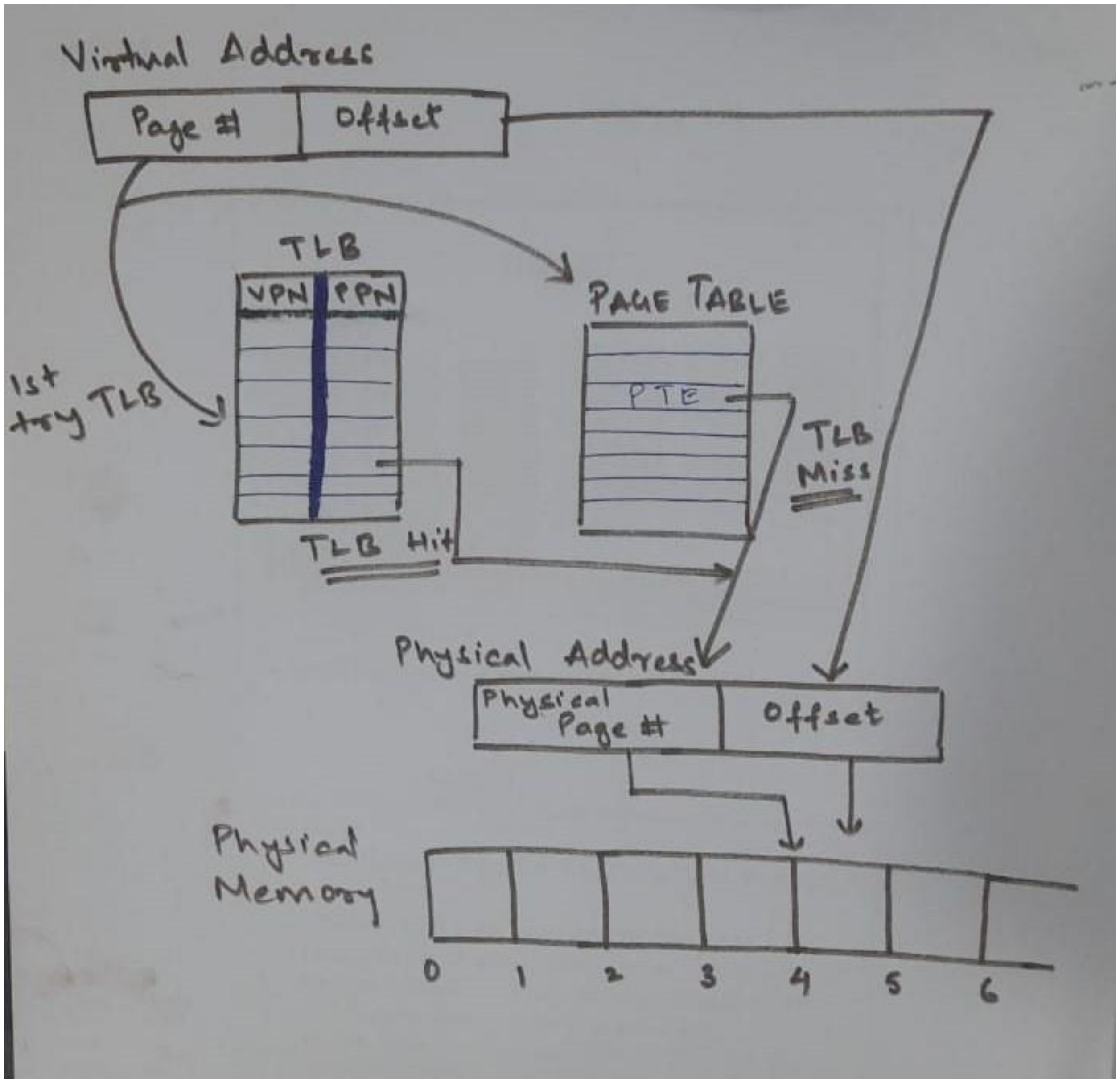
• Frame size of 2^8 bytes

• 256 frames

• Physical memory of 65,536 bytes (256 frames * 256-byte frame size)

Additionally, the program was concerned with only reading logical addresses and translating them to their corresponding physical addresses.

Fig. 1: Logical Diagram Representing Virtual Memory to Physical Memory Translation



5.1 Address Translation

The program translates logical to physical addresses using a TLB and page table. First, the page number is extracted from the logical address, and the TLB is consulted. In the case of a TLB-hit, the frame number is obtained from the TLB. In the case of a TLB-miss, the page table must be consulted. In the latter case, either the frame number is obtained from the page table or a page fault occurs.

5.2. Handling Page Faults

The program implements demand paging. The backing store is represented by the file BACKING_STORE.bin, a binary file of size 65,536 bytes. When a page fault occurs, we read in a 256-byte page from the file BACKING_STORE and store it in an available page frame in physical memory. For example, if a logical address with page number 15 resulted in a page fault, our program would read in page 15 from BACKING_STORE (pages begin at 0 and are 256 bytes in size) and store it in a page frame in physical memory. Once this frame is stored (and the page table and TLB are updated), subsequent accesses to page 15 is resolved by either the TLB or the page table.

BACKING_STORE.bin is treated as a random-access file so that the program can randomly seek to certain positions of the file for reading. The standard C library functions to be used for performing I/O are fopen(), fread(), fseek(), and fclose().

The size of physical memory is the same as the size of the virtual address space-- 65,536 bytes-- so the program is made to not concern about page replacements during a page fault. Later on, a modification to this project is described using a smaller amount of physical memory; at that point, a page-replacement strategy will be required.

5.3. Blending with the Judicial Backdrop

Cases have been divided into segments and allotted priorities. CRITICAL cases are those under categories of Rape/Murder and Human Trafficking. IMPORTANT cases deal with appeals under any kind of Harassment. Damage to Property cases are considered of NORMAL priority and cases dealing with Violence/Theft are marked as LOW.

The idea is to immediately assign cases with Critical priority to the court, and keep other cases on hold- thereby giving them a storage location where the respective case's file can be stored. Later on, as the processing of the higher priority case finishes, the next greatest priority case is made active in the court. This way, the cases will be sequentially executed in the court, and those which are pending will be stored securely

5.4. Test File

The file addresses.txt contains integer values representing logical addresses(case file number) ranging from 0-65536 (the size of the virtual address space). The program opens this file, reads each logical address and translates it to its corresponding physical address(case file storage location), and outputs the value of the signed byte at the physical address.

6.Code

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>

FILE * backStore;
FILE * addressFile;
FILE * searchfile;

#define LINELENGTH 10
#define PAGESIZE 256

int pageTable[PAGESIZE];
int pageFrame[PAGESIZE];

#define TLB_LENGTH 16

int TLBPage[TLB_LENGTH];
int TLBFrame[TLB_LENGTH];
int TLBNum = 0;
int TLBCounter = 0;

#define FRAMELENGTH 256

char readBacker[FRAMELENGTH];

#define physicalMemoryBytes 65536
```

```
int physicalMemory[physicalMemoryBytes];
int pageFault = 0;


void initializeInfo(int *arr, int n) {
int i = 0;
for(i=0;i<n;i++) {
arr[i] = -1;
}
}

int readBackStore(int page) {
int i=0,j=0,availableFrame=0,startFrameIndex=0;

if(fseek(backStore, page * PAGESIZE,SEEK_SET)!=0){
printf("ERROR\n");
}
if(fread(readBacker, sizeof(signed char), PAGESIZE,backStore)==0) {
printf("ERROR\n");
}

for(i=0;i<PAGESIZE;i++) {
if(pageFrame[i]==-1) {
pageFrame[i] = 0;
availableFrame = i;
break;
}
```

```
}

startFrameIndex = PAGESIZE * availableFrame;
for(j=0;j<PAGESIZE;j++) {
physicalMemory[startFrameIndex] = readBacker[j];
startFrameIndex++;
}
pageTable[page] = availableFrame;
pageFault++;
return availableFrame;
}

int changeAddress(int logAddress) {
int page=0,i=0,frameNum = -1,offset = 0;

double oriPage,decPage,intPage,offsetDub=0.0;

page = logAddress/PAGESIZE;
oriPage = (double)logAddress/PAGESIZE;
decPage = modf(oriPage, &intPage);
offsetDub = decPage*PAGESIZE;
offset = (int)offsetDub;

for(i=0;i<TLB_LENGTH;i++) {
if(TLBPage[i] == page) {
frameNum = TLBFrame[i];
TLBNum++;
}
```

```
}

if(frameNum == -1) {

if(pageTable[page] == -1) {
frameNum = readBackStore(page);
} else {

frameNum = pageTable[page];
}
TLBPage[TLBCounter%TLB_LENGTH] = page;
TLBFrame[TLBCounter%TLB_LENGTH] = frameNum;
TLBCounter++;
}
return (frameNum*PAGESIZE) + offset;
}

int main(int argc, char *argv[]) {

int translations = 0, logAddress = 0, address = 0;
char line[LINELENGTH];
char filenm[20];
if(argc!=2) {
printf("Please enter the case files.\nEx: addresses.txt\n");
scanf("%s", filenm);
}
```

```
backStore = fopen("BACKING_STORE.bin", "r");
if(backStore == NULL) {
printf("1 Null ,,,,,,, Backing Store file is NULL or corrupted. Re-Download the file and run
again\n");
return -1;
}

addressFile = fopen(filenm,"r");
if(addressFile==NULL) {
printf("2 Null ,,,,,, addresses file is NULL or corrupted. Re-enter the file and run again\n");
return -1;
}

initializeInfo(pageTable, PAGESIZE);
initializeInfo(pageFrame, PAGESIZE);
initializeInfo(TLBPage, TLB_LENGTH);
initializeInfo(TLBFrame, TLB_LENGTH);

printf("\n\n\n\t\t\t\t\t\t======================================\n" );
printf("\t\t\t\t\t\t || JUDICIAL CASE MANAGER using VMM ||\n" );
printf("\t\t\t\t\t\t======================================\n\n\n" );

while(fgets(line, LINELENGTH, addressFile) != NULL) {
logAddress = atoi(line);

address = changeAddress(logAddress);

if(logAddress<100){
```

```
printf("File Number: %d\t\t Category: RAPE/MURDER\t\t Priority: CRITICAL\t Status:
ASSIGNED\n", logAddress);

}

else if(logAddress<=5000){

printf("File Number: %d\t Category: RAPE/MURDER\t\t Priority: CRITICAL\t Status:
ASSIGNED\n", logAddress);

}

else if(logAddress>5000 && logAddress<=10000){

printf("File Number: %d\t Category: HUMAN TRAFFICKING\t Priority: CRITICAL\t Status:
ASSIGNED\n", logAddress);

}

else if(logAddress>10000 && logAddress<=20000){

printf("File Number: %d\t Category: HARASSMENT\t\t Priority: IMPORTANT\t Status:
PENDING\t File Location: %d\n", logAddress,address);

translations++;

}

else if(logAddress>20000 && logAddress<=40000){

printf("File Number: %d\t Category: DAMAGE TO PROPERTY\t Priority: NORMAL\t Status:
PENDING\t File Location: %d\n", logAddress,address);

translations++;

}

else if(logAddress>40000){

printf("File Number: %d\t Category: VIOLENCE/THEFT\t Priority: LOW\t\t Status:
PENDING\t File Location: %d\n", logAddress,address);

translations++;

}


}


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

printf("Number of translations/mapping: %d\n", translations);
printf("Number of Page Faults: %d\n", pageFault);
printf("Page Fault Rate: %f\n",(float)(pageFault*100)/(float)translations);
printf("Number of TLB Hits: %d\n", TLBNum);
printf("TLB Rate: %f\n", (float)(TLBNum*100)/(float)translations);
char flloc[10];
int chch=0;
printf("\nEnter 1 to search for a file\nEnter 0 to exit the program\n ");
scanf("%d",&chch);
while(chch)
{
printf("Enter the file location you want to search: ");
printf("\n");
scanf("%s",&flloc);
strcat(flloc,".txt");
searchfile = fopen(flloc,"r");
char ch;
ch = fgetc(searchfile);
while(ch!=EOF)
{
if(ch!="/")
printf ("%c", ch);
else
printf("\n\n");
ch = fgetc(searchfile);
}
fclose(searchfile);
printf("\nEnter 1 to search for a file\nEnter 0 to exit the program \n");
scanf("%d",&chch);
}

fclose(addressFile);
fclose(backStore);
return 0;
}
```

7.Expected Output

After completion, the program was to report the following statistics and it successfully reported the following:

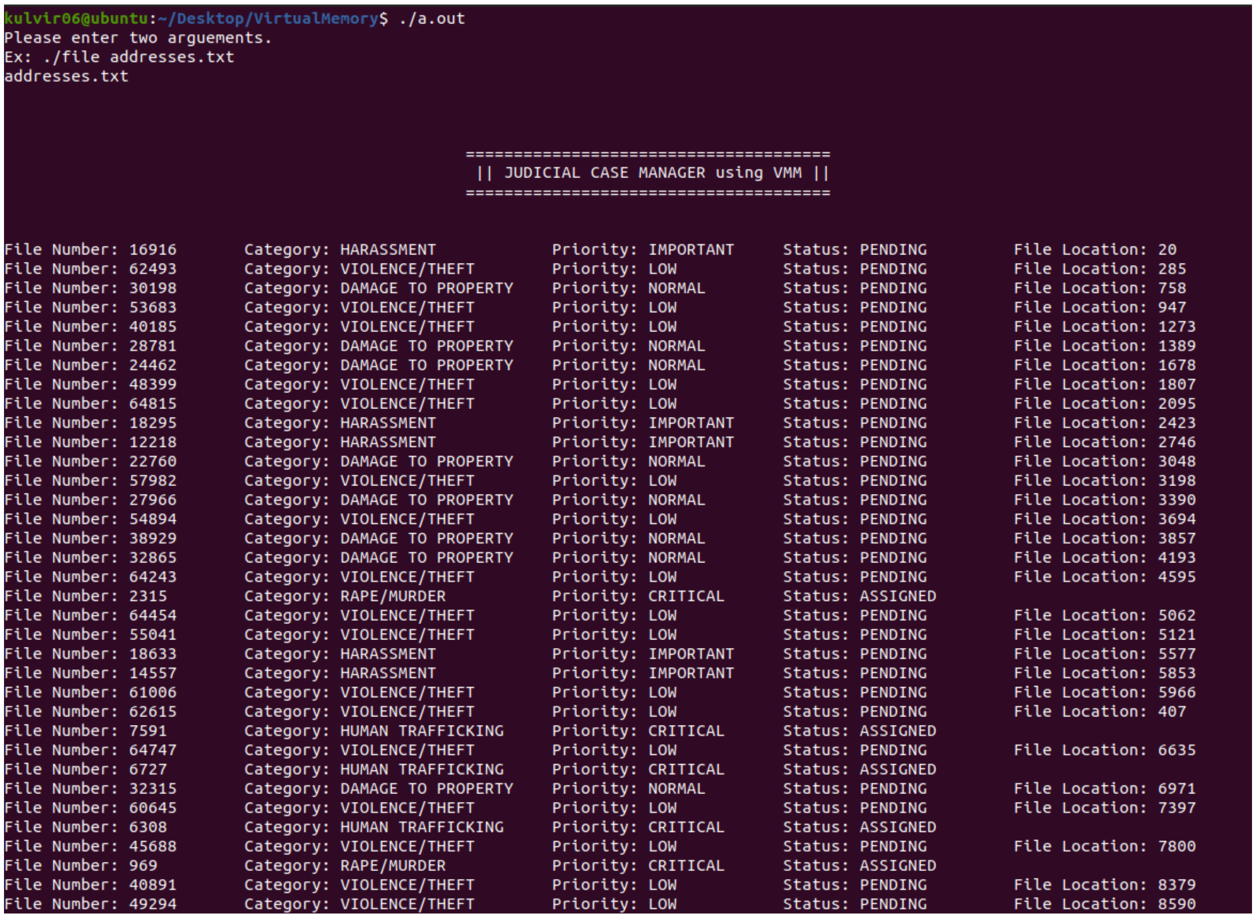
1. File Number, Category, Priority Level and Status of each entry case

2. Number of Translations: Total number of cases which are in pending state and have been allotted a storage location

3. Page-fault/Page-fault Rate-- The percentage of address references that resulted in page faults

4. TLB hit/ TLB hit Rate-- The percentage of address references that were resolved in the TLB

8.Program Output Screenshots

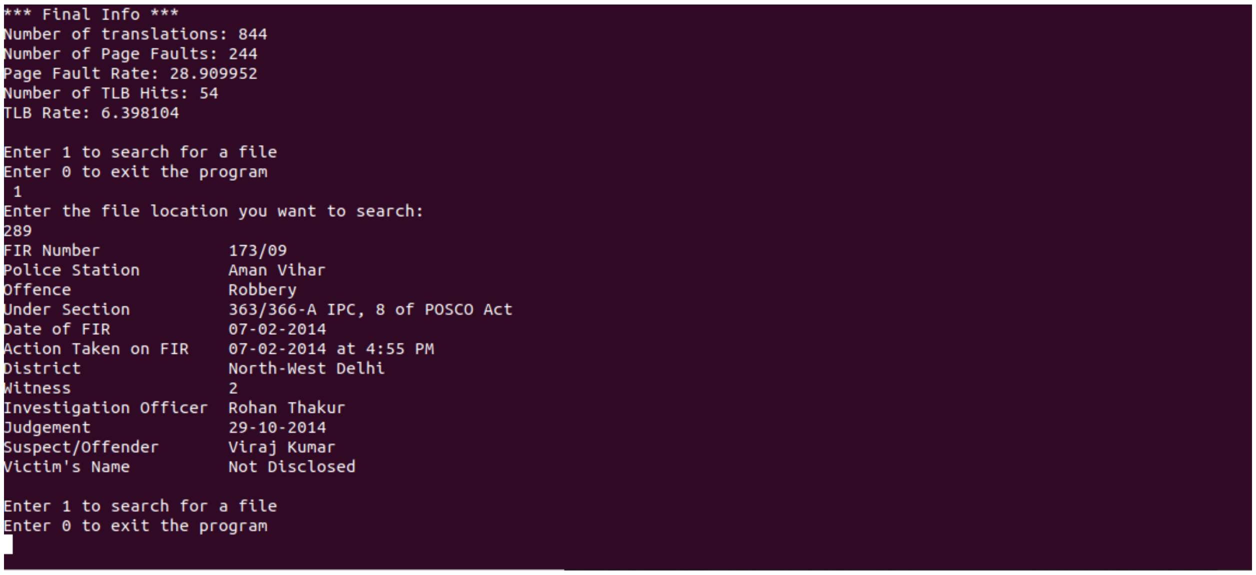


```
kulvir06@ubuntu:~/Desktop/VirtualMemory$ ./a.out
Please enter two arguements.
Ex: ./file addresses.txt
addresses.txt



                                                ======================================
                                                 || JUDICIAL CASE MANAGER using VMM ||
                                                ======================================


File Number: 16916       Category: HARASSMENT            Priority: IMPORTANT     Status: PENDING         File Location: 20
File Number: 62493       Category: VIOLENCE/THEFT        Priority: LOW           Status: PENDING         File Location: 285
File Number: 30198       Category: DAMAGE TO PROPERTY    Priority: NORMAL        Status: PENDING         File Location: 758
File Number: 53683       Category: VIOLENCE/THEFT        Priority: LOW           Status: PENDING         File Location: 947
File Number: 40185       Category: VIOLENCE/THEFT        Priority: LOW           Status: PENDING         File Location: 1273
File Number: 28781       Category: DAMAGE TO PROPERTY    Priority: NORMAL        Status: PENDING         File Location: 1389
File Number: 24462       Category: DAMAGE TO PROPERTY    Priority: NORMAL        Status: PENDING         File Location: 1678
File Number: 48399       Category: VIOLENCE/THEFT        Priority: LOW           Status: PENDING         File Location: 1807
File Number: 64815       Category: VIOLENCE/THEFT        Priority: LOW           Status: PENDING         File Location: 2095
File Number: 18295       Category: HARASSMENT            Priority: IMPORTANT     Status: PENDING         File Location: 2423
File Number: 12218       Category: HARASSMENT            Priority: IMPORTANT     Status: PENDING         File Location: 2746
File Number: 22760       Category: DAMAGE TO PROPERTY    Priority: NORMAL        Status: PENDING         File Location: 3048
File Number: 57982       Category: VIOLENCE/THEFT        Priority: LOW           Status: PENDING         File Location: 3198
File Number: 27966       Category: DAMAGE TO PROPERTY    Priority: NORMAL        Status: PENDING         File Location: 3390
File Number: 54894       Category: VIOLENCE/THEFT        Priority: LOW           Status: PENDING         File Location: 3694
File Number: 38929       Category: DAMAGE TO PROPERTY    Priority: NORMAL        Status: PENDING         File Location: 3857
File Number: 32865       Category: DAMAGE TO PROPERTY    Priority: NORMAL        Status: PENDING         File Location: 4193
File Number: 64243       Category: VIOLENCE/THEFT        Priority: LOW           Status: PENDING         File Location: 4595
File Number: 2315        Category: RAPE/MURDER           Priority: CRITICAL      Status: ASSIGNED
File Number: 64454       Category: VIOLENCE/THEFT        Priority: LOW           Status: PENDING         File Location: 5062
File Number: 55041       Category: VIOLENCE/THEFT        Priority: LOW           Status: PENDING         File Location: 5121
File Number: 18633       Category: HARASSMENT            Priority: IMPORTANT     Status: PENDING         File Location: 5577
File Number: 14557       Category: HARASSMENT            Priority: IMPORTANT     Status: PENDING         File Location: 5853
File Number: 61006       Category: VIOLENCE/THEFT        Priority: LOW           Status: PENDING         File Location: 5966
File Number: 62615       Category: VIOLENCE/THEFT        Priority: LOW           Status: PENDING         File Location: 407
File Number: 7591        Category: HUMAN TRAFFICKING     Priority: CRITICAL      Status: ASSIGNED
File Number: 64747       Category: VIOLENCE/THEFT        Priority: LOW           Status: PENDING         File Location: 6635
File Number: 6727        Category: HUMAN TRAFFICKING     Priority: CRITICAL      Status: ASSIGNED
File Number: 32315       Category: DAMAGE TO PROPERTY    Priority: NORMAL        Status: PENDING         File Location: 6971
File Number: 60645       Category: VIOLENCE/THEFT        Priority: LOW           Status: PENDING         File Location: 7397
File Number: 6308        Category: HUMAN TRAFFICKING     Priority: CRITICAL      Status: ASSIGNED
File Number: 45688       Category: VIOLENCE/THEFT        Priority: LOW           Status: PENDING         File Location: 7800
File Number: 969         Category: RAPE/MURDER           Priority: CRITICAL      Status: ASSIGNED
File Number: 40891       Category: VIOLENCE/THEFT        Priority: LOW           Status: PENDING         File Location: 8379
File Number: 49294       Category: VIOLENCE/THEFT        Priority: LOW           Status: PENDING         File Location: 8590
```



```
*** Final Info ***
Number of translations: 844
Number of Page Faults: 244
Page Fault Rate: 28.909952
Number of TLB Hits: 54
TLB Rate: 6.398104

Enter 1 to search for a file
Enter 0 to exit the program
 1
Enter the file location you want to search:
289
FIR Number             173/09
Police Station         Aman Vihar
Offence                Robbery
Under Section          363/366-A IPC, 8 of POSCO Act
Date of FIR            07-02-2014
Action Taken on FIR    07-02-2014 at 4:55 PM
District               North-West Delhi
Witness                2
Investigation Officer  Rohan Thakur
Judgement              29-10-2014
Suspect/Offender       Viraj Kumar
Victim's Name          Not Disclosed

Enter 1 to search for a file
Enter 0 to exit the program
```

9.Conclusion

Through this project, we have successfully created a Judicial Case Manager using Virtual Memory Management (VMM) which converts virtual memory addresses(file number) to physical memory addresses(file storage) using a Transation Lookaside Buffer (TLB). Thus the following values were observed:

1. The logical address being translated (the integer value being read from addresses.txt).

2. The corresponding physical address (what your program translates the logical address to).

3. The signed byte value stored at the translated physical address.

The drawback of this system is that to search for the file details, the user has to search for the file location manually and type it in.

10.References

Virtual Memory, by Peter J. Denning, published in ACM Computing Surveys Volume 2, Issue 3

Efficient Virtual Memory for Big Memory Servers, by Arkaprava Basu, Jayneel Gandhi, Jichuan Chang, Mark D. Hill, Michael M. Swift, published in ACM SIGARCH Computer Architecture News,

Understanding The Linux Virtual Memory Manager, by Mel Gorman,

A Banked-promotion translational lookaside buffer system, by Jung-Hoon Lee, Seh-Woong Jeong, Shin-Dug Kim, Charles Weems, published in Journal of Systems Architecture

Low-synchronization translation lookaside buffer consistency in large-scale shared-memory multiprocessors, by Bryan S. Rosenberg, published in 12th ACM symposium on Operating System Principles

Online resources such as-

Virtual Memory Manager IBM Knowledge Support
The Design of PARAS Microkernel Rajkumar Buyya
Virtual Memory Wikipedia
Operating System – Virtual Memory Tutorials Point