MEMBERS: ISHAAN OHRI-18BCE0265, ANMOL PANT-18BCE0283, SHIVAM ANAND-18BCE0490, RAUNAQ NIJHAWAN-18BCE0633

Implementing Memory Management for Hypervisor

Abstract

Virtual machines have become an increasingly used feature of modern computing. This project focuses on learning, understanding and implementing the virtual machine specific code for memory management module. We are required to write code that creates and manages inverted, hypervisor-level page tables. We start by having a small, working implementation of guest-level page tables, so that we have a related reference implementation. The kernel data structures to manage allocated and free memory are unchanged.

Keywords

VMM (virtual machine monitor/manager)

Page table (PT)

Mapping

Inverted PT

Physical page number

Offset

Virtual page number

Introduction

Just like many technological achievements in use today, virtualization is not a new concept, and its roots reach back to 1960s and 1970s. Virtualization is the process of creating a software-based, or virtual, representation of something, such as virtual applications, servers, storage, and networks, while boosting efficiency and agility for all size businesses.

Hypervisors -allocate resources between one or more virtual machines (instances; guests) that operate on top of it by continually monitoring guests and their resource requirements and active utilization rates in order to redistribute resources and boost system's overall efficiency. Without hypervisors, it would be difficult to take full advantage of the modern processing power and memory resources simply because most applications only use a fraction of what is available, leaving a significant percentage of resources unevolved.

Proposed Work

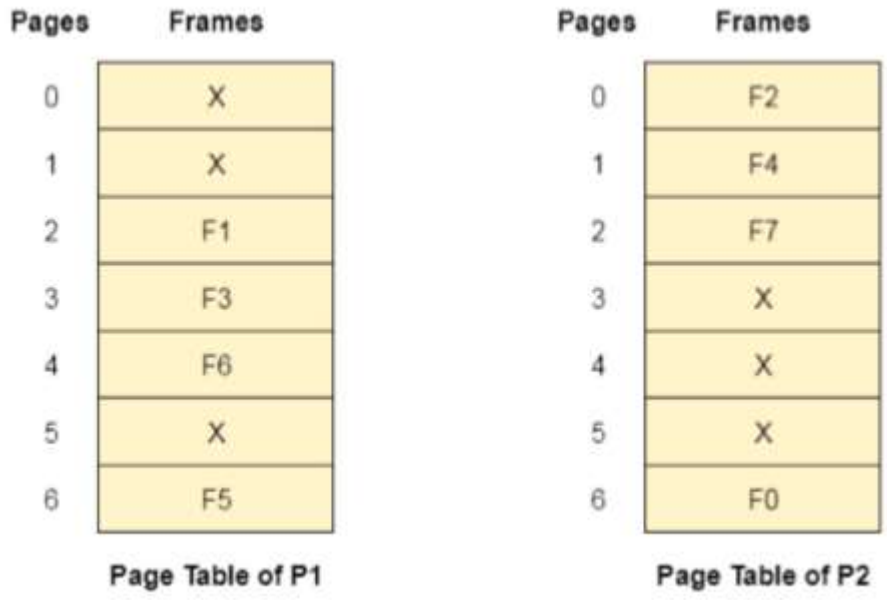
In the project, we will be implementing memory management in hypervisor using a code that creates and manages inverted, hypervisor-level page tables. We start by having a small, working implementation of guest-level page tables, so that we have a related reference implementation.

Working methodology:

We implement the concept of Inverted page tables.

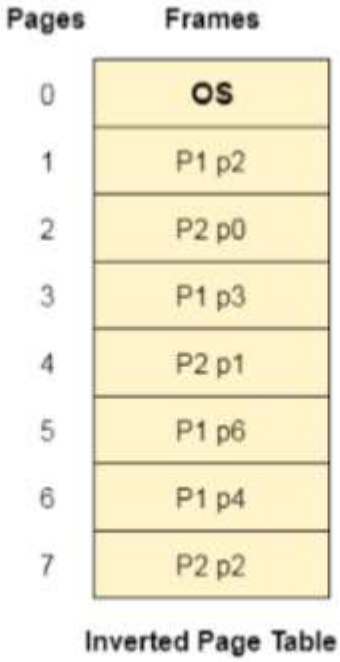
Inverted Page Table is the global page table which is maintained by the Operating System for all the processes. In inverted page table, the number of entries is equal to the number of frames in the main memory. It can be used to overcome the drawbacks of page table.

There is always a space reserved for the page regardless of the fact that whether it is present in the main memory or not. However, this is simply the wastage of the memory if the page is not present.



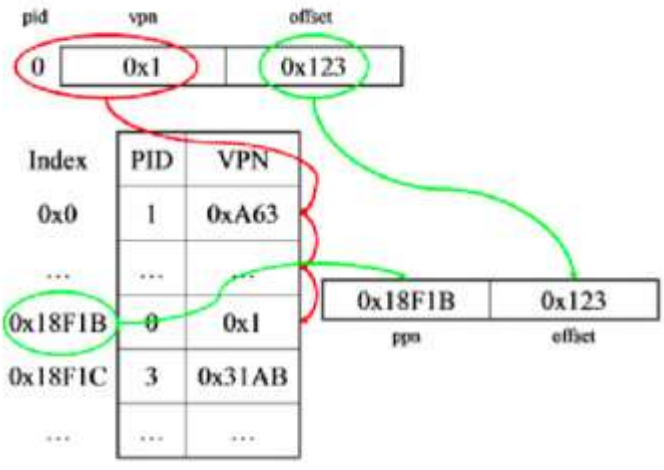
Consider these two page-tables, if we implement normal page tables then the memory used will be equivalent to the memory of both these page tables and thus a lot of memory will be wasted because frames are not allocated.

We can save this wastage by just inverting the page table. We can save the details only for the pages which are present in the main memory. Frames are the indices and the information saved inside the block will be Process ID and page number.



The simplest form of an inverted page table contains one entry per physical page in a linear array. Since the table is shared, each entry must contain the process ID of the page owner. And since physical pages are now mapped to virtual, each entry contains a virtual page number instead of a physical. The physical page number is not stored, since the index in the table corresponds to it.

In order to translate a virtual address, the virtual page number and current process ID are compared against each entry, traversing the array sequentially. When a match is found, the index of the match replaces the virtual page number in the address to obtain a physical address. If no match is found, a page fault occurs.



Shadow Paging

Shadow page tables are used by the hypervisor to keep track of the state in which the guest "thinks" its page tables should be. The guest can't be allowed access to the hardware page tables because then it would essentially have control of the machine. So, the hypervisor keeps the "real" mappings (guest virtual -> host physical) in the hardware when the relevant guest is executing, and keeps a representation of the page tables that the guest thinks it's using "in the shadows," or at least that's how he like to think about it.

The hypervisor must build up these shadow page tables as it sees page hit generated by the guest. When the guest writes a mapping into one of its page tables, the hypervisor won't know right away, so the shadow page tables won't instantly "be in sync" with what the guest intends.

Another alternative to shadow table is the much-improved nested paging, the hardware provides another level of indirection when translating linear to physical addresses. Page tables function as before, but linear addresses are now translated to "guest physical" addresses first and not physical addresses directly. A new set of paging registers now exists under the traditional paging mechanism and translates from guest physical addresses to host physical addresses, which are used to access memory. Nested paging eliminates the overhead caused by Virtual Machine exits and page table accesses. In essence, with nested page tables the guest can handle paging without intervention from the hypervisor. Nested paging thus significantly improves virtualization performance.

Code

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


char pagetable[16][10];

char frametable[16][10];

char shadowtable[16][10];

char nestedtable[16][10];

int frame = 0;

int shadowframe=5;

int nestedframe=7;

int NestI = 0 ;

int ShadowI = 0;

int p;

int I =0;
```

```
char converttohex(int a)
{
    char b;
    switch(a)
    {
        case 10 :
            b = 'A';
            break;
        case 11 :
            b = 'B';
            break;
        case 12 :
            b = 'C';
            break;
        case 13 :
            b = 'D';
            break;
        case 14 :
            b = 'E';
            break;
        case 15 :
            b = 'F';
            break;
        default:
            b = a + '0';
    }
    return (b);
}
void map_address()
{
    printf("Enter the process id (Less than 10) - \n");
    char pageid[10];
    scanf("%s",pageid);
    char nestedid[10];
```

```
    char frameid[10];
    strcpy(frameid,pageid);
    strcpy(nestedid,pageid);
    printf("Enter the virtual address\n");
    char virtual1[7];
    scanf("%s",virtual1);
    strcat(pageid,virtual1);
    char Frameno = converttohex(frame);
    virtual1[2]=Frameno;
    strcat(frameid,virtual1);
    strcpy(pagetable[I],pageid);
    strcpy(frametable[I],frameid);
    char NestedFrameno = converttohex(nestedframe);
    virtual1[2]=NestedFrameno;
    strcat(nestedid,virtual1);
    strcpy(nestedtable[NestI],nestedid);
    nestedframe++;
    I++;
    frame++;
    NestI++;
}

void print_pagetable()
{
    printf("\nPid \t Virtual Page Number\n");
    for(int i = 0 ; i < I; i++)
    {
        char pagenumber[7];
        for(int j = 0; j<=6;j++)
        {
            pagenumber[j]=pagetable[i][j+1];
        }
        printf("%c \t %s\n",pagetable[i][0],pagenumber);
    }
```

```
}

void print_frametable()
{
    printf("\nPid \t Frame Number\n");
    for(int i = 0 ; i < I; i++)
    {
        char framenumber[7];
        for(int j =0; j<=6;j++)
        {
            framenumber[j]=frametable[i][j+1];
        }
        printf("%c \t %s\n",frametable[i][0],framenumber);
    }
}

void find_page()
{
    printf("Enter the process id - \n");
    char pageid[10];
    scanf("%s",pageid);
    char frameid[10];
    char shadowid[10] ;
    strcpy(shadowid,pageid);
    strcpy(frameid,pageid);
    printf("Enter the virtual address\n");
    char virtual1[7];
    scanf("%s",virtual1);
    strcat(pageid,virtual1);
    int flag = 0;
    int i;
    for(i = 0 ; i < I ; i++)
    {
        if(strcmp(pageid,pagetable[i])==0)
```

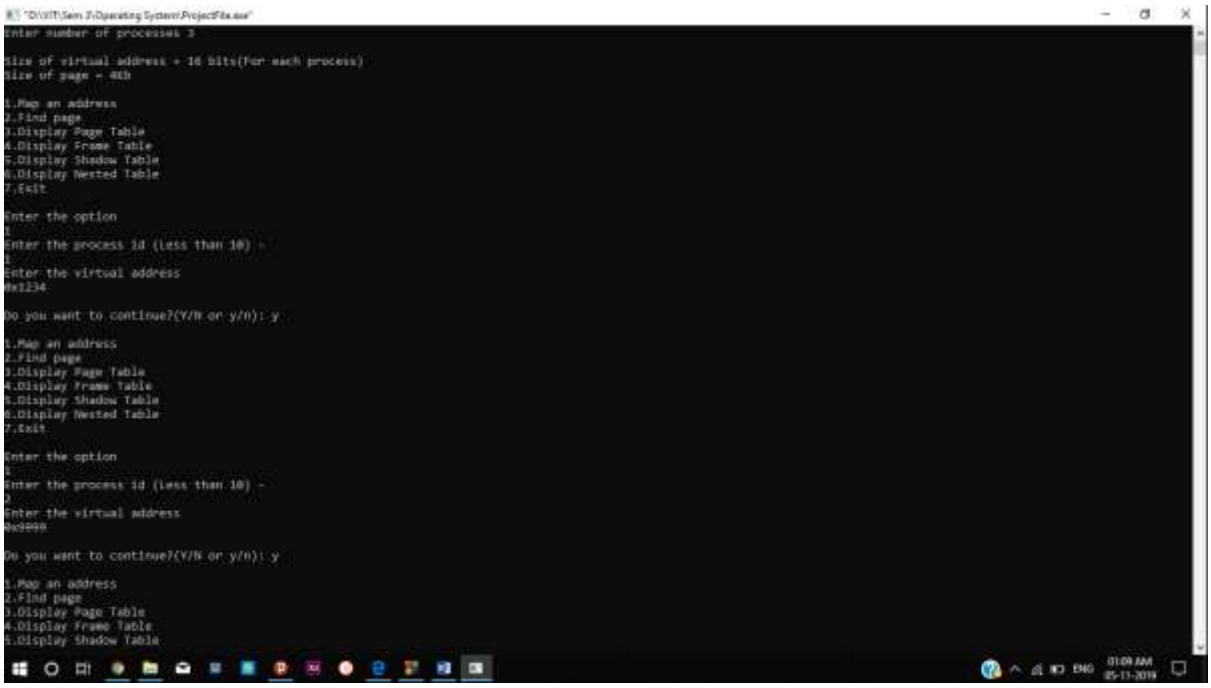
```
    {
        flag = 1;
        break;
    }
}
if(flag==1)
{
    printf("Page Hit\n");
    printf("Shadow frame will be made\n");
    printf("\nPhysical Address is \n");
    char framenumber[7];
    for(int j =0; j<=6;j++)
    {
        framenumber[j]=frametable[i][j+1];
    }
    printf("%s\n",framenumber);
    char ShadowFrameno = converttohex(shadowframe);
    virtual1[2]=ShadowFrameno;
    strcat(shadowid,virtual1);
    strcpy(shadowtable[ShadowI],shadowid);
    ShadowI++;
    shadowframe++;
}
else
{
    char Frameno = converttohex(frame);
    virtual1[2]=Frameno;
    printf("Page Fault, page will be mapped\n");
    strcat(frameid,virtual1);
    strcpy(pagetable[I],pageid);
    strcpy(frametable[I],frameid);
    frame++;
    I++;
}
```

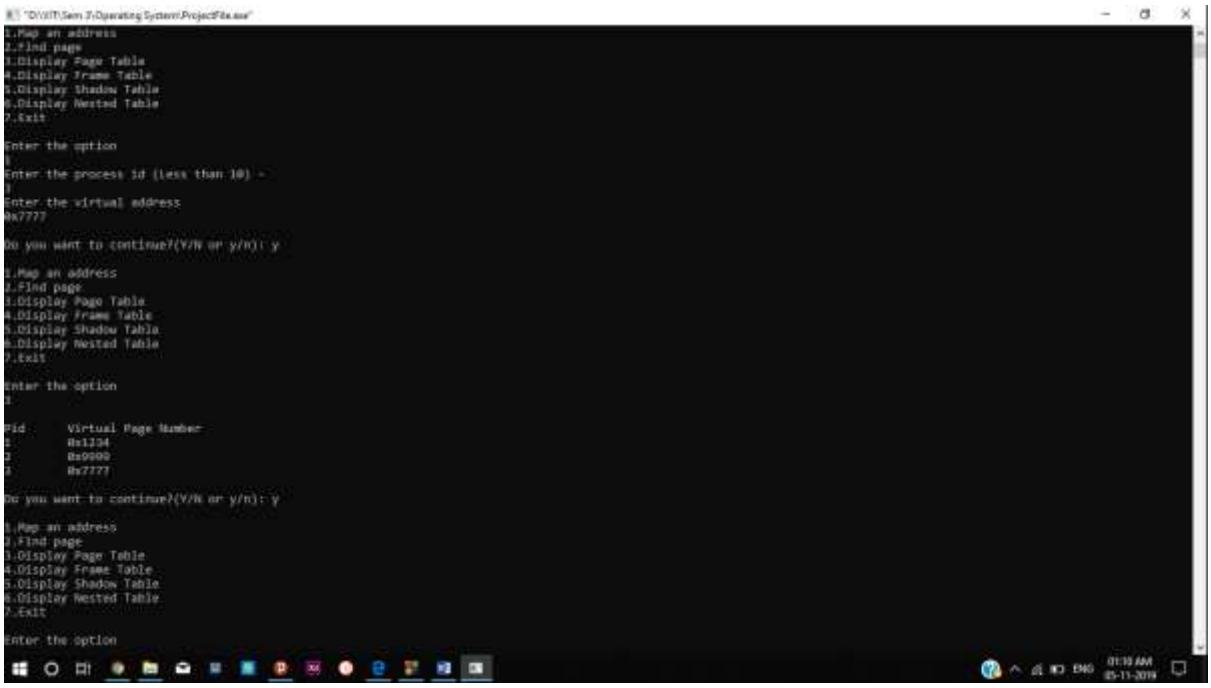
```
}

void shadow_page()
{
    printf("\nShadow Table");
    printf("\nPid \t Frame Number\n");
    for(int i = 0 ; i < ShadowI; i++)
    {
        char framenumber[7];
        for(int j =0; j<=6;j++)
        {
            framenumber[j]=shadowtable[i][j+1];
        }
        printf("%c \t %s\n",shadowtable[i][0],framenumber);
    }
}

void nested_page()
{
    printf("\nNested Table");
    printf("\nPid \t Frame Number\n");
    for(int i = 0 ; i < NestI; i++)
    {
        char framenumber[7];
        for(int j =0; j<=6;j++)
        {
            framenumber[j]=nestedtable[i][j+1];
        }
        printf("%c \t %s\n",nestedtable[i][0],framenumber);
    }
}

int main()
{
```

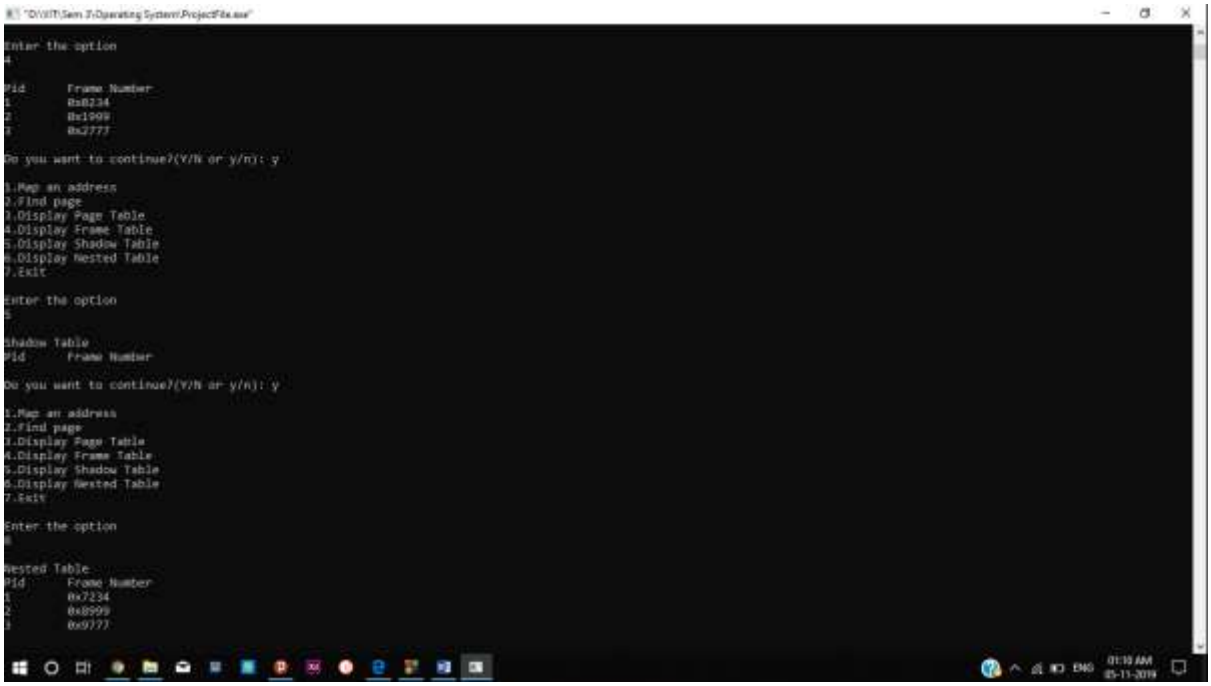
```
    printf("Enter number of processes ");
    scanf("%d",&p);
    printf("\nSize of virtual address = 16 bits(For each process)\n");
    printf("Size of page = 4Kb\n");
    char choice = 'Y';
    int option;
    do
    {
        printf("\n1.Map an address\n2.Find page\n3.Display Page Table\n4.Display
Frame Table\n5.Display Shadow Table\n6.Display Nested Table\n7.Exit\n");
        printf("\nEnter the option\n");
        scanf("%d",&option);
        switch (option)
        {
            case 1 :
                map_address();
                break;
            case 2 :
                find_page();
                break;
            case 3 :
                print_pagetable();
                break;
            case 4 :
                print_frametable();
                break;
            case 5 :
                shadow_page();
                break;
            case 6 :
                nested_page();
                break;
            case 7 :
                exit(0);
```

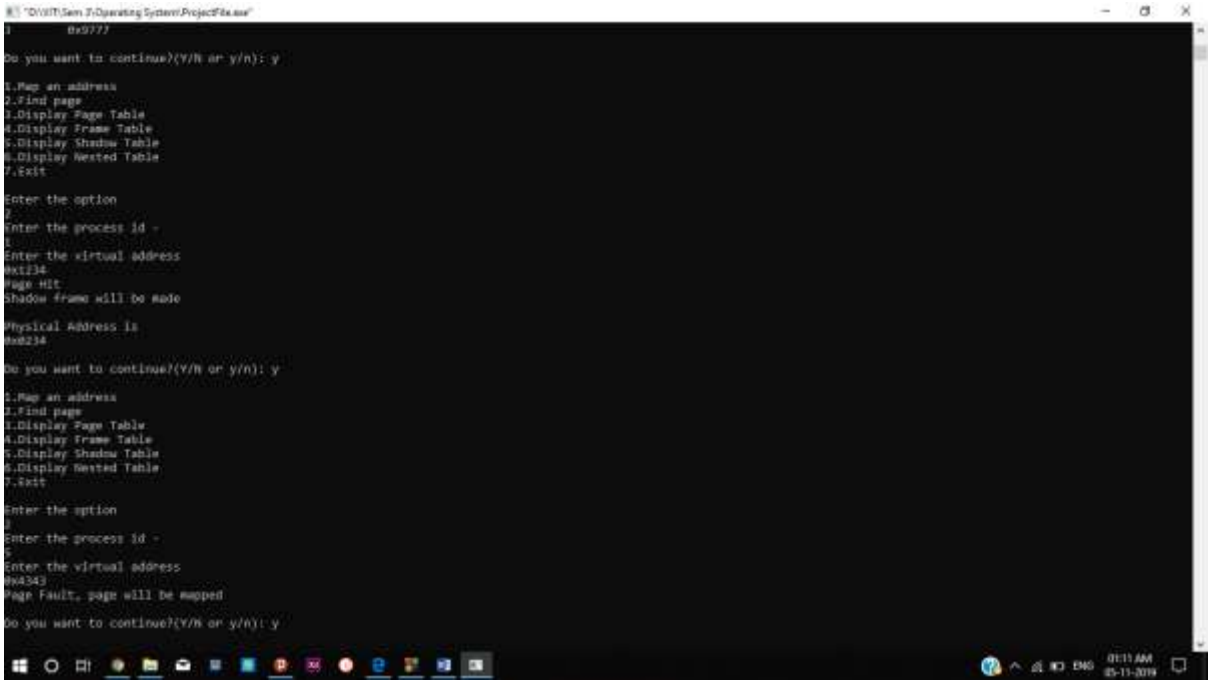
```
        }
        printf("\nDo you want to continue?(Y/N or y/n): ");
        scanf(" %c",&choice);
    }
    while(choice == 'Y' || choice == 'y');
    return 0;
}
```

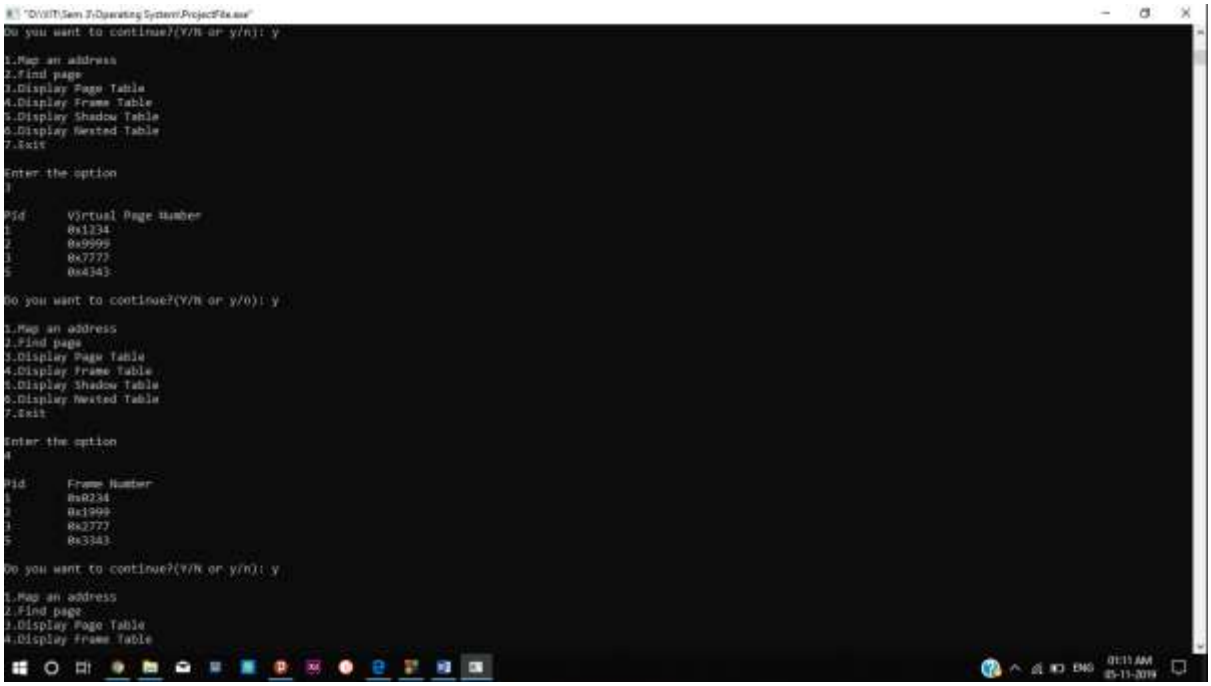
Output

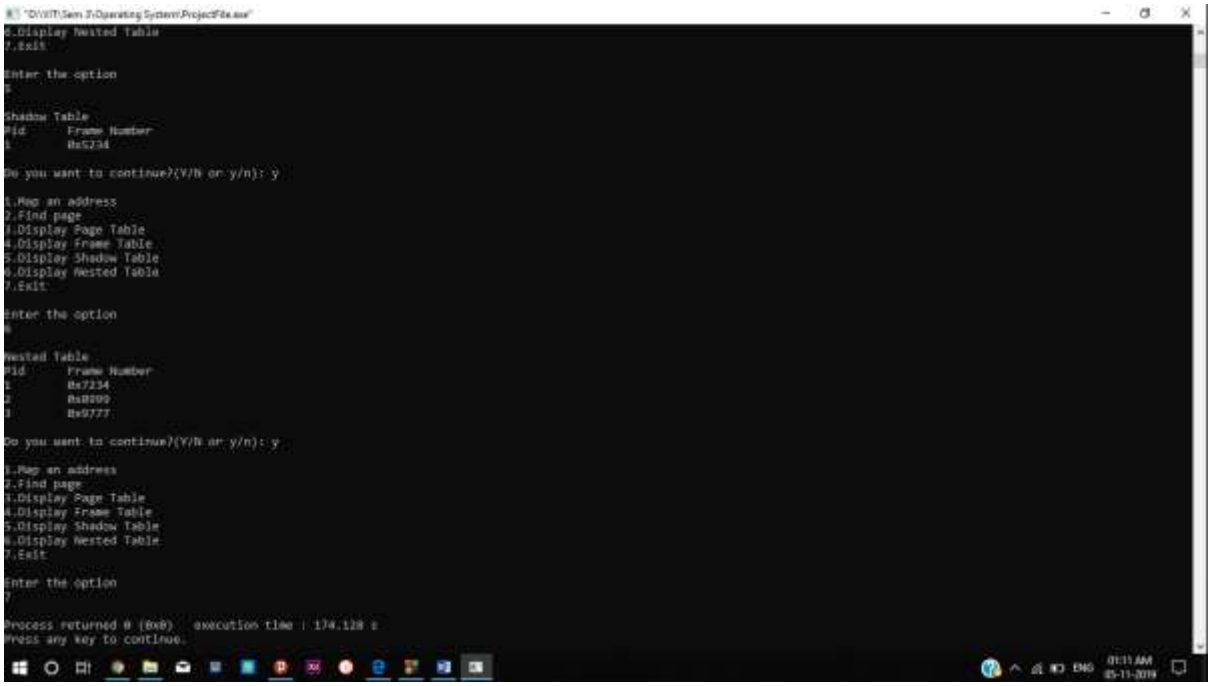





Enter the option
4
Pid Frame Number
1 0x8234
2 0x1999
3 0x2777
Do you want to continue?(Y/N or y/n): y
1.Map an address
2.Find page
3.Display Page Table
4.Display Frame Table
5.Display Shadow Table
6.Display Nested Table
7.Exit
Enter the option
5
Shadow Table
Pid Frame Number
Do you want to continue?(Y/N or y/n): y
1.Map an address
2.Find page
3.Display Page Table
4.Display Frame Table
5.Display Shadow Table
6.Display Nested Table
7.Exit
Enter the option
6
Nested Table
Pid Frame Number
1 0x7234
2 0x8999
3 0x9777


3 0x9777
Do you want to continue?(Y/N or y/n): y
1.Map an address
2.Find page
3.Display Page Table
4.Display Frame Table
5.Display Shadow Table
6.Display Nested Table
7.Exit
Enter the option
2
Enter the process id -
1
Enter the virtual address
0x1234
Page Hit
Shadow frame will be made
Physical Address is
0x8234
Do you want to continue?(Y/N or y/n): y
1.Map an address
2.Find page
3.Display Page Table
4.Display Frame Table
5.Display Shadow Table
6.Display Nested Table
7.Exit
Enter the option
2
Enter the process id -
1
Enter the virtual address
0x4343
Page Fault, page will be mapped
Do you want to continue?(Y/N or y/n): y


Do you want to continue?(Y/N or y/n): y
1.Map an address
2.Find page
3.Display Page Table
4.Display Frame Table
5.Display Shadow Table
6.Display Nested Table
7.Exit
Enter the option
3
Pid Virtual Page Number
1 0x1234
2 0x9999
3 0x7777
5 0x4343
Do you want to continue?(Y/N or y/n): y
1.Map an address
2.Find page
3.Display Page Table
4.Display Frame Table
5.Display Shadow Table
6.Display Nested Table
7.Exit
Enter the option
4
Pid Frame Number
1 0x8234
2 0x1999
3 0x2777
5 0x3343
Do you want to continue?(Y/N or y/n): y
1.Map an address
2.Find page
3.Display Page Table
4.Display Frame Table


6.Display Nested Table
7.Exit
Enter the option
5
Shadow Table
Pid Frame Number
1 0x5234
Do you want to continue?(Y/N or y/n): y
1.Map an address
2.Find page
3.Display Page Table
4.Display Frame Table
5.Display Shadow Table
6.Display Nested Table
7.Exit
Enter the option
6
Nested Table
Pid Frame Number
1 0x7234
2 0x8899
3 0x9777
Do you want to continue?(Y/N or y/n): y
1.Map an address
2.Find page
3.Display Page Table
4.Display Frame Table
5.Display Shadow Table
6.Display Nested Table
7.Exit
Enter the option
7
Process returned 0 (0x0) execution time : 174.128 s
Press any key to continue.

Related Work

‘Inverted Page Tables, TLBs’ - Amir Kamil

Inverted Page Table is the global page table which is maintained by the Operating System for all the processes. In inverted page table, the number of entries is equal to the number of frames in the main memory. It can be used to overcome the drawbacks of page table.

‘Teaching Virtualization by Building a Hypervisor’ - Abhinand Palicherla, Tao Zhang, Donald E. Porter.

A set of JOS and HOSS exercises related to implementing different modules of a hypervisor is being given to students after a course and elaborate description of hypervisors and its functions are provided.

‘Stephen J. Bigelow, Senior Technology Editor, Tech Target Network

Virtualization often puts memory at a premium, but hypervisors employ a set of memory paging features to enhance performance and do more with less.

‘Paging’ - Michael O’Boyle

It discusses about paging, page tables and implementation of page tables. It also talks about diff types of pages and its uses.

‘Virtual memory management techniques: A beginner's guide’ - Greg Shields
Concentrated Technology, Tech Target Network

There are lots of virtual memory management techniques, including ballooning and memory compression. You can use these methods to control resource sharing among collocated VMs.

‘A survey of memory management techniques in virtualized systems’ – Debadatta Mishra , Purushottam Kulkarni

It focuses on virtualization and management of memory on a single host, where a hypervisor controls and manages the memory sub-system of the machine. Memory virtualization solutions, implemented both in software and with hardware assistance, are discussed and evaluated against virtualization requirements

‘Hypervisor’ - Wikipedia

There are lots of virtual memory management techniques, including ballooning and memory compression. You can use these methods to control resource sharing among co located VMs.

‘What is a hypervisor?’ - Keith Shaw

A hypervisor is a process that separates a computer’s operating system and applications from the underlying physical hardware. Usually done as software although embedded hypervisors can be created for things like mobile devices.

‘Shadow Paging’ - Wikipedia

Shadow paging is a copy-on-write technique for avoiding in-place updates of pages. Instead, when a page is to be modified, a shadow page is allocated.

‘Virtual Machine’ - Margaret Rouse, Tech Target Network

A virtual machine (VM) is an operating system (OS) or application environment that is installed on software, which imitates dedicated hardware. The end user has the same experience on a virtual machine as they would have on dedicated hardware.

Result

We implement a simple simulation of the inverted page table in the hypervisor along with shadow and nested paging. The inverted page table helps in reducing the memory used at the cost of increased time to access the page. So if we have a memory shortage and don’t want to waste chunks of memory inverted page table is suggested over other page tables. The concept of nested and shadow table helps as well in trying to get out from the hypervisor as less as possible. Clearly it can be seen that nested tables will be much more efficient in the longer run.

Conclusion

The project has focused on implementing memory management module in a hypervisor using inverted page table, which we have considered out of the many other types of page tables. We have also implemented nested and shadow paging, which are two techniques of indexing the page table. Out of the scope of the project, we can consider the implementation of the same techniques with other types of tables for the comparison of the performances and memory consumptions.

References

1. https://web.eecs.umich.edu/~akamil/teaching/sp04/040104.pdf
2. http://www.cs.unc.edu/~porter/pubs/jos-vm.pdf
3. https://searchservervirtualization.techtarget.com/feature/Memory-paging-techniques-hypervisors-use-to-improve-VM-performance
4. https://www.inf.ed.ac.uk/teaching/courses/os/slides/10-paging16.pdf
5. https://searchservervirtualization.techtarget.com/tip/Virtual-memory-management-techniques-A-beginners-guide
6. https://www.cse.iitb.ac.in/~puru/courses/spring19/cs695/downloads/memvirt-survey.pdf
7. https://en.m.wikipedia.org/wiki/Hypervisor
8. https://www.networkworld.com/article/3243262/what-is-a-hypervisor.html
9. https://en.m.wikipedia.org/wiki/Shadow_paging
10. https://searchservervirtualization.techtarget.com/definition/virtual-machine