OSD Lab Exam 190031187
InSem Radhakrishna.

1. Write a system program to assign and remove execute permission to a file using chmod() system call

2. Write a program that translates logical to physical addresses. convert logical address 21 to physical address using the given page table page size is 16 bytes. So offset is 4 bits. page table will have 4 entries.

program

1.
```
#ifdef CS333_P5
#include "types.h"
#include "user.h"
//sets the mode or permission bits for the target
 specified by pathname
int main (int argc, char *argv[])
{   if (argc > 2)
    {
      if ((atoo(argv[1]) >0 && atoo(argv[1])<1778)||
          (atoi(argv[1])==0))
      {
        if (chmod(argv[2], atoo(argv[1]))
          printf(1,"chmod %s failed\n",argv[2]);
        else
          printf(1,"updating permissions..\n");
      else
          printf(1,"Invalid arguments for
                                  chmod \n");
      }
      exit();
}}
```

2 19003187

```
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>

#define pages 4
#define page_mask 255
#define page_size 16
#define offset_bits 4
#define offset_mask 255
#define memo_size pages * page_size
#define buf_size 10

int pagetable[pages];
signed char main_Memo[memo_size];
signed char * backing_ptr;

int main (int argc, const char * argv[])
{   if (argc != 3) {
        printf("please enter 3 args: </file_exe_name>
    <backing store><input file>\n");
    exit(0);
    }
    for (int i=0; i<pages; i++)
    { pagetable[i] = -1;
    }
    const char *file_name argv[1];
```

```
const char *input_file = argv[2]
const char * output_file = "output.txt"
int backing_ptr_fd = open(file_name, O_RDONLY);
backing_ptr = mmap( 0, memo_size, prot_read,
                map_private, backing_ptr_fd, 0);
File *input_fp = fopen(input_file, "r");
FILE * output_fp = fopen( output_file, "w");
char buf [BUF_SIZE];
unsigned char freepage = 0;
while ( fgets (buf, BUF_SIZE, input_fp) != NULL)
{ int logical_addr = atoi (buf);
  int offset = (logical_addr >> offset_Bits) &
                PAGE_MASK;
  int logical = (logical_addr >> offset_Bits) &
                PAGE_MASK;
  int physical = pagetable [logical];
  total_addr++;
  if (physical == -1){
     pagefault ++;
     physical = freepage;
  freepage++
  memcpy (main_MEMO + physical * page_size,
  backing_ptr + logical * page_size, page_size);
     page table [logical] = physical;
  }
  int physical_addr = (physical << offset_bits) | offset;
  signed char value = main_Memo [physical * page_size
   + offset];
```

```
printf("Number of Translated Addresses = %d\n",
  total_addr);
printf("page-faults = %d\n", pagefault);
(pageFaults/total)
printf("page fault rate = %.1f %\n",
    (pageFault/ total_addr * 1)) * 100);
return 0;
}
```

Address Translation

The program will translate logical to physical addresses using a TLB and page table ~~outlined in section 5~~. First, the page number is ~~encoded and~~ extracted from the logical address and the TLB is consulted, In the case of a TLB-miss, the page table must have consulted. In the latter case, either the frame number is obtained from the page table or a page fault occurs. A visual representation of the address translation process ~~appears in~~ appears.

Compile with

```
gcc virtual_mem.c -o virtual_memo -std=c99 -lm
```