**OSD Practical-8**

**Pre-lab:**

1.Problems on address translation using paging, segmentation, and hybrid(both).

**Ans:**

Problem-01:

A certain computer system has the segmented paging architecture for virtual memory. The memory is byte addressable. Both virtual and physical address spaces contain 216 bytes each. The virtual address space is divided into 8 non-overlapping equal size segments. The memory management unit (MMU) has a hardware segment table, each entry of which contains the physical address of the page table for the segment. Page tables are stored in the main memory and consists of 2 byte page table entries. What is the minimum page size in bytes so that the page table for a segment requires at most one page to store it?

Solution-

Given-

* Virtual Address Space = Process size = 216 bytes
* Physical Address Space = Main Memory size = 216 bytes
* Process is divided into 8 equal size segments
* Page table entry size = 2 bytes

Let page size = n bytes.

Now, since page table has to be stored into a single page, so we must have-

Size of page table <= Page size

## Size of Each Segment-

Size of each segment

= Process size / Number of segments

= 216 bytes / 8

= 216 bytes / 23

= 213 bytes

= 8 KB

## Number of Pages Of Each Segment-

Number of pages each segment is divided

= Size of segment / Page size

= 8 KB / n bytes

= (8K / n) pages

## Size of Each Page Table-

Size of each page table

= Number of entries in page table x Page table entry size

= Number of pages the segment is divided x 2 bytes

= (8K / n) x 2 bytes

= (16K / n) bytes

**Page Size-**

Substituting values in the above condition, we get-

(16K / n) bytes <= n bytes

(16K / n) <= n

n2 >= 16K

n2 >=214

n >= 27

Thus, minimum page size possible = 27 bytes = 128 bytes.

**In-lab**

1.Write a program to demonstrate Accessing Memory with Paging - linear translates.

**Ans:**

**Vm.c code:**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <sys/types.h>**

**#include <sys/stat.h>**

**#include <fcntl.h>**

**#include <unistd.h>**

**#define PSIZE 4096**

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

**{**

**int addressFile,backingStore;**

**char \*file= argv[1];**

**char ch,ct, input[1000], output;**

**int logicalAddress, physicalAddress;**

**int i=0, j=0;**

**int p,d;**

**int f;**

**char frames[PSIZE\*16];**

**int frametable[16];**

**int start, current;**

**int offset, pagefault=0;**

**int freeFrame=-1;**

**int pagetable[16];**

**for (j=0;j<16;j++)**

**{**

**pagetable[j] = -1;**

**}**

**pagetable[0]=0x2;**

**pagetable[1]=0x1;**

**pagetable[2]=0x6;**

**pagetable[3]=0x0;**

**pagetable[4]=0x4;**

**pagetable[5]=0x3;**

**pagetable[9]=0x5;**

**pagetable[11]=0x7;**

**for (j=0;j<16;j++)**

**{**

**frametable[j] = -1;**

**}**

**frametable[0] = 1;**

**frametable[1] = 1;**

**frametable[2] = 1;**

**frametable[3] = 1;**

**frametable[4] = 1;**

**frametable[5] = 1;**

**frametable[9] = 1;**

**frametable[11] = 1;**

**addressFile = open("address.txt",O\_RDONLY);**

**backingStore = open("BACKING\_STORE.bin",O\_RDONLY);**

**if(addressFile != -1)**

**{**

**while(read(addressFile, &ch, sizeof(char)) != 0)**

**{**

**if(ch != '\n')**

**{**

**input[i] = ch;**

**i++;**

**}**

**else**

**{**

**logicalAddress =atoi(input);**

**p = (logicalAddress & 0x000000000000f000UL) >> 12;**

**d = (logicalAddress & 0x0000000000000fffUL);**

**printf("\nlogicalAddress: %d, p: %d, d: %d", logicalAddress,p,d);**

**if(pagetable[p]!= -1){**

**f = pagetable[p];**

**physicalAddress = (f \* PSIZE) + d;**

**printf("\nphysicalAddress: %d, f: %d", physicalAddress,f);**

**}**

**// pagetable-miss, page-fault**

**else**

**{**

**pagefault++;**

**// locate free frame (-1) in physical memory**

**for (j=0;j<8;j++)**

**{**

**if(frametable[j]==-1)**

**{**

**freeFrame = j;**

**break;**

**}**

**}**

**if(backingStore != -1)**

**{**

**offset=0;**

**start = PSIZE \* p;**

**current=lseek(backingStore, start, SEEK\_SET);**

**while((offset < PSIZE)&&(current))**

**{**

**current = read(backingStore, &ct, sizeof(char));**

**frames[freeFrame\*offset] = ct;**

**offset++;**

**}**

**}**

**else**

**{**

**printf("Backing-Store Does not exist!");**

**close(backingStore);**

**close(addressFile);**

**return 0;**

**}**

**// update pagetable, frametable**

**pagetable[p] = freeFrame;**

**frametable[freeFrame] = 0;**

**physicalAddress = (freeFrame \* PSIZE) + d;**

**printf("\nphysicalAddress: %d, freeFrame: %d", physicalAddress, freeFrame);**

**}**

**output = frames[physicalAddress];**

**printf("\nByte value stored at physicalAddress %d: %c\n",physicalAddress, output);**

**memset(input,0,sizeof(input));**

**i=0;**

**}**

**}**

**printf("\nTotal Page Faults: %d",pagefault);**

**}**

**else**

**printf("Addresses File Does not exist!");**

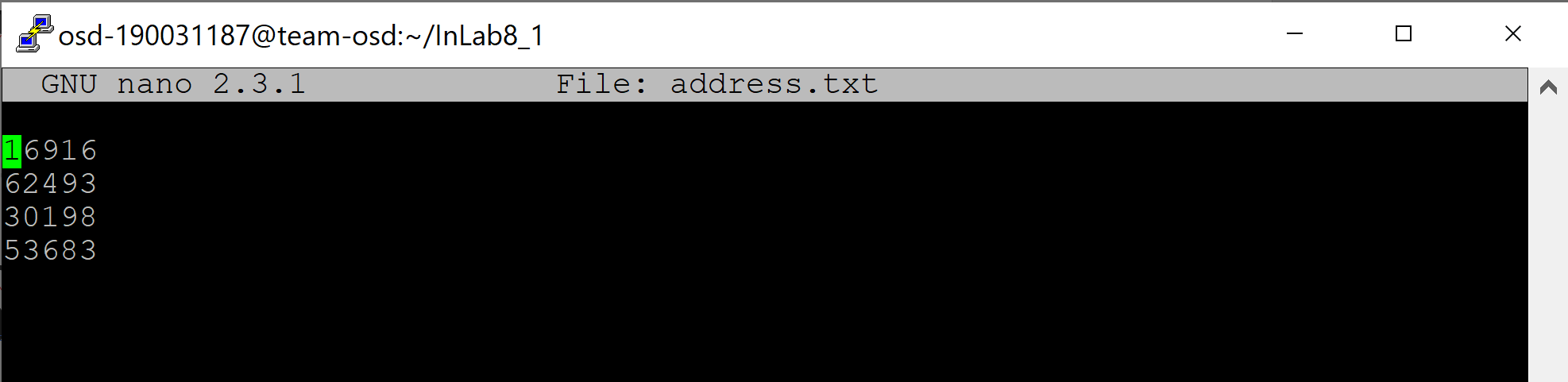
**close(backingStore);**

**close(addressFile);**

**return 0;**

**}**

**address.txt**

****

**pagetable.c**

**#include <stdio.h>**

**#include <stdbool.h>**

**#include <stdlib.h>**

**struct page{**

**int page\_no;**

**int frame;**

**};**

**int main()**

**{**

**int size\_logical\_address,size\_physical\_address,i,size\_of\_page,j;**

**printf("Enter size of logical address space: ");**

**scanf("%d",&size\_logical\_address);**

**printf("Enter size of physical address space: ");**

**scanf("%d",&size\_physical\_address);**

**printf("Enter size of page: ");**

**scanf("%d",&size\_of\_page);**

**int number\_of\_frames = size\_physical\_address/size\_of\_page;**

**int number\_of\_pages = size\_logical\_address/size\_of\_page;**

**struct page pageTable[number\_of\_pages];**

**printf("Enter page table: \n");**

**for(i=0;i<number\_of\_pages;i++)**

**{**

**pageTable[i].frame = -1;**

**}**

**for(i=0;i<number\_of\_pages;i++)**

**{**

**int frame;**

**bool replica = false;**

**pageTable[i].page\_no = i;**

**printf("Enter frame for %d page number(-1 if frame doesn't exist): ",i);**

**scanf("%d",&frame);**

**for(j=0;j<number\_of\_pages;j++)**

**{**

**if(frame!= -1 && pageTable[j].frame == frame)**

**{**

**replica = true;**

**printf("Frame number already stored\n");**

**}**

**}**

**if(frame > number\_of\_frames)**

**{**

**replica = true;**

**printf("Cannot exceed frame size\n");**

**}**

**if(replica == false)**

**{**

**pageTable[i].frame = frame;**

**}**

**}**

**int logical\_address;**

**printf("Enter -1 to exit\n");**

**while(1)**

**{**

**printf("Enter logical address: ");**

**scanf("%d",&logical\_address);**

**if(logical\_address == -1)**

**return 0;**

**int page\_no = logical\_address/size\_of\_page;**

**int offset = logical\_address%size\_of\_page;**

**if(pageTable[page\_no].frame == -1)**

**{**

**printf("No such logical address exist\n");**

**}**

**else**

**{**

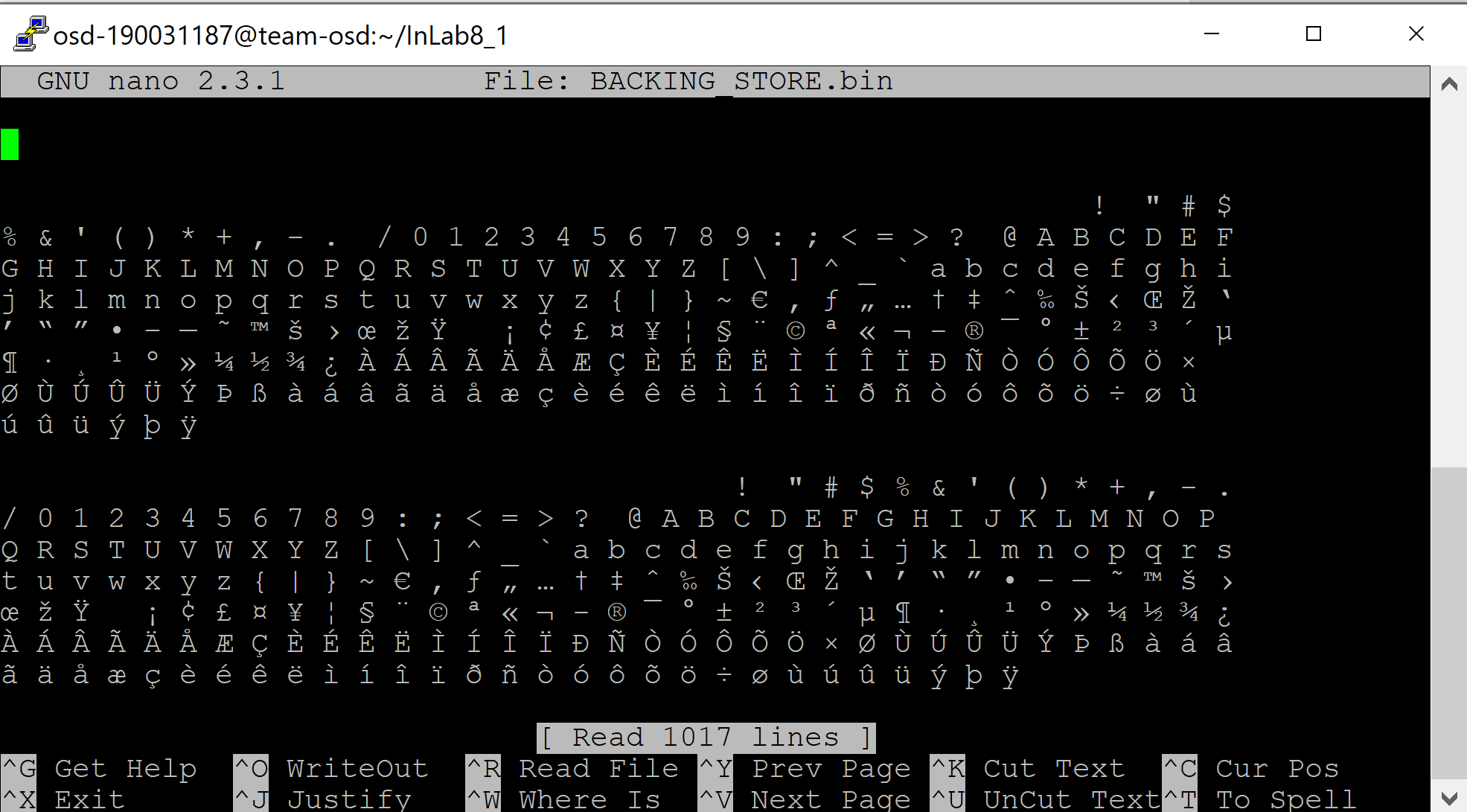
**printf("Page no: %d \nOffset: %d\nFrame no: %d\nPhysical address: %d\n",page\_no, offset, pageTable[page\_no].frame,pageTable[page\_no].frame\*size\_of\_page + offset );**

**}**

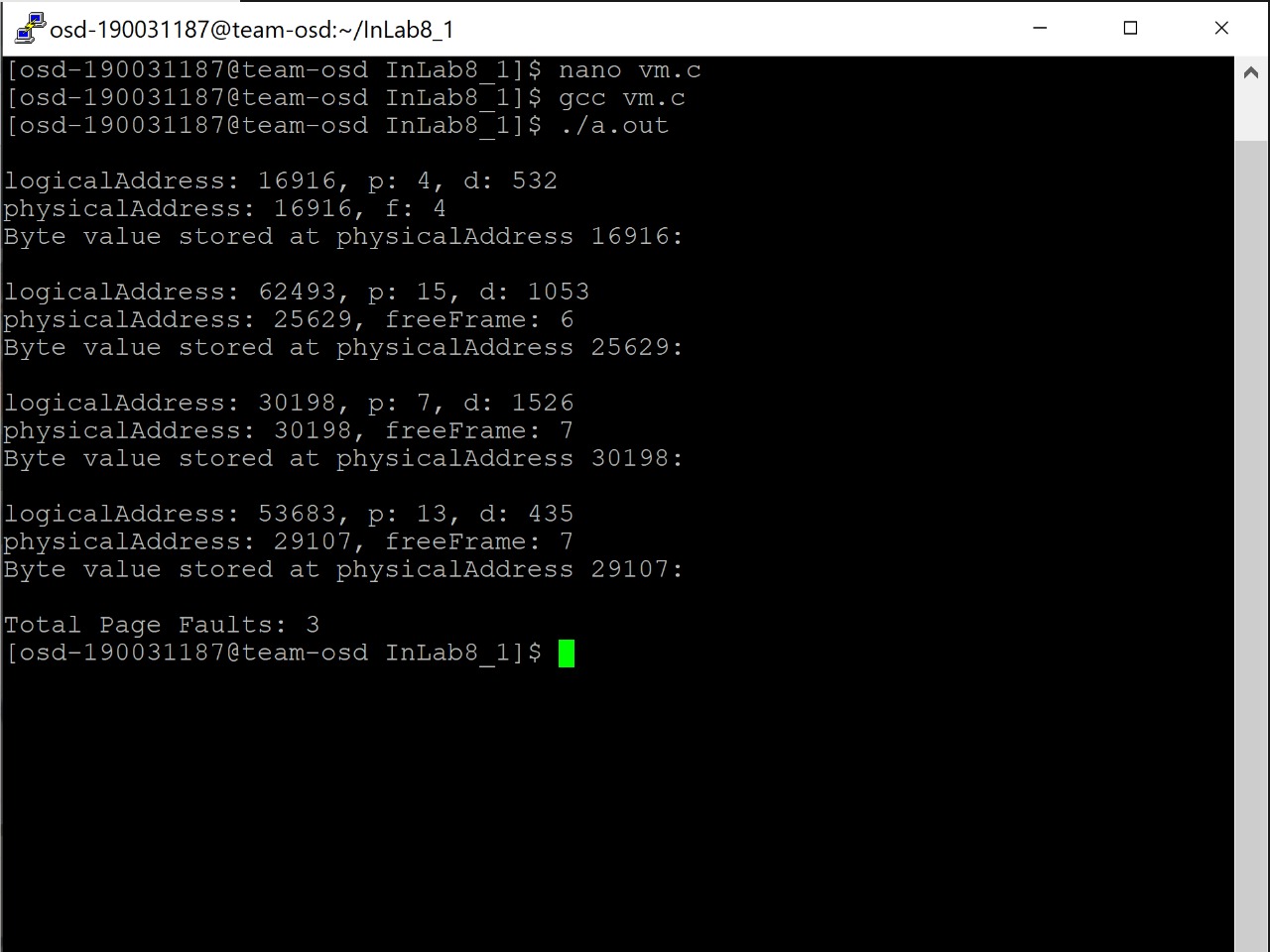
**}**

**}**

**BACKING\_STORE.bin**

****

**Output**

****

2.Write a program that translates logical to physical addresses for a virtual address space of size 2^16 = 65,536 bytes. Your program will read from a file containing logical addresses and, using a TLB as well as a page table

**Ans:**

Virtual\_mem.c code:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <errno.h>

#define PAGE\_SIZE 256

#define PAGE\_ENTRIES 256

#define PAGE\_NUM\_BITS 8

#define FRAME\_SIZE 256

#define FRAME\_ENTRIES 256

#define MEM\_SIZE (FRAME\_SIZE \* FRAME\_ENTRIES)

#define TLB\_ENTRIES 16

int virtual,page\_number,offset,physical,frame\_number,value,page\_table[PAGE\_ENTRIES],tlb[TLB\_ENTRIES][2],tlb\_front = -1,tlb\_back = -1,mem\_index = 0,fault\_counter = 0,tlb\_counter = 0;

char memory[MEM\_SIZE];

int address\_counter = 0;

float fault\_rate,tlb\_rate;

int get\_physical(int virtual),get\_offset(int virtual);

int get\_page\_number(int virtual);

void initialize\_page\_table(int n);

void initialize\_tlb(int n);

int consult\_page\_table(int page\_number);

int consult\_tlb(int page\_number);

void update\_tlb(int page\_number, int frame\_number);

int get\_frame();

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

char\* in\_file;

char\* out\_file;

char\* store\_file;

char\* store\_data;

int store\_fd;

char line[8];

FILE\* in\_ptr;

FILE\* out\_ptr;

initialize\_page\_table(-1);

initialize\_tlb(-1);

if (argc != 4) {

printf("Enter input, output, and store file names!");

exit(EXIT\_FAILURE);

}

else {

in\_file = argv[1];

out\_file = argv[2];

store\_file = argv[3];

if ((in\_ptr = fopen(in\_file, "r")) == NULL) {

printf("Input file could not be opened.\n");

exit(EXIT\_FAILURE);

}

if ((out\_ptr = fopen(out\_file, "a")) == NULL) {

printf("Output file could not be opened.\n");

exit(EXIT\_FAILURE);

}

store\_fd = open(store\_file, O\_RDONLY);

store\_data = mmap(0, MEM\_SIZE, PROT\_READ, MAP\_SHARED, store\_fd, 0);

if (store\_data == MAP\_FAILED) {

close(store\_fd);

printf("Error mmapping the backing store file!");

exit(EXIT\_FAILURE);

}

while (fgets(line, sizeof(line), in\_ptr)) {

virtual = atoi(line);

address\_counter++;

page\_number = get\_page\_number(virtual);

offset = get\_offset(virtual);

frame\_number = consult\_tlb(page\_number);

if (frame\_number != -1) {

physical = frame\_number + offset;

value = memory[physical];

}

else {

frame\_number = consult\_page\_table(page\_number);

if (frame\_number != -1) {

physical = frame\_number + offset;

update\_tlb(page\_number, frame\_number);

value = memory[physical];

}

else {

int page\_address = page\_number \* PAGE\_SIZE;

if (mem\_index != -1) {

memcpy(memory + mem\_index,

store\_data + page\_address, PAGE\_SIZE);

frame\_number = mem\_index;

physical = frame\_number + offset;

value = memory[physical];

page\_table[page\_number] = mem\_index;

update\_tlb(page\_number, frame\_number);

if (mem\_index < MEM\_SIZE - FRAME\_SIZE) {

mem\_index += FRAME\_SIZE;

}

else {

mem\_index = -1;

}

}

else {

}

}

}

fprintf(out\_ptr, "Virtual address: %d ", virtual);

fprintf(out\_ptr, "Physical address: %d ", physical);

fprintf(out\_ptr, "Value: %d\n", value);

}

fault\_rate = (float) fault\_counter / (float) address\_counter;

tlb\_rate = (float) tlb\_counter / (float) address\_counter;

fprintf(out\_ptr, "Number of Translated Addresses = %d\n", address\_counter);

fprintf(out\_ptr, "Page Faults = %d\n", fault\_counter);

fprintf(out\_ptr, "Page Fault Rate = %.3f\n", fault\_rate);

fprintf(out\_ptr, "TLB Hits = %d\n", tlb\_counter);

fprintf(out\_ptr, "TLB Hit Rate = %.3f\n", tlb\_rate);

fclose(in\_ptr);

fclose(out\_ptr);

close(store\_fd);

}

return EXIT\_SUCCESS;

}

int get\_physical(int virtual) {

physical = get\_page\_number(virtual) + get\_offset(virtual);

return physical;

}

int get\_page\_number(int virtual) {

return (virtual >> PAGE\_NUM\_BITS);

}

int get\_offset(int virtual) {

int mask = 255;

return virtual & mask;

}

void initialize\_page\_table(int n) {

for (int i = 0; i < PAGE\_ENTRIES; i++) {

page\_table[i] = n;

}

}

void initialize\_tlb(int n) {

for (int i = 0; i < TLB\_ENTRIES; i++) {

tlb[i][0] = -1;

tlb[i][1] = -1;

}

}

int consult\_page\_table(int page\_number) {

if (page\_table[page\_number] == -1) {

fault\_counter++;

}

return page\_table[page\_number];

}

int consult\_tlb(int page\_number) {

for (int i = 0; i < TLB\_ENTRIES; i++) {

if (tlb[i][0] == page\_number) {

tlb\_counter++;

return tlb[i][1];

}

}

return -1;

}

void update\_tlb(int page\_number, int frame\_number) {

if (tlb\_front == -1) {

tlb\_front = 0;

tlb\_back = 0;

tlb[tlb\_back][0] = page\_number;

tlb[tlb\_back][1] = frame\_number;

}

else {

tlb\_front = (tlb\_front + 1) % TLB\_ENTRIES;

tlb\_back = (tlb\_back + 1) % TLB\_ENTRIES;

tlb[tlb\_back][0] = page\_number;

tlb[tlb\_back][1] = frame\_number;

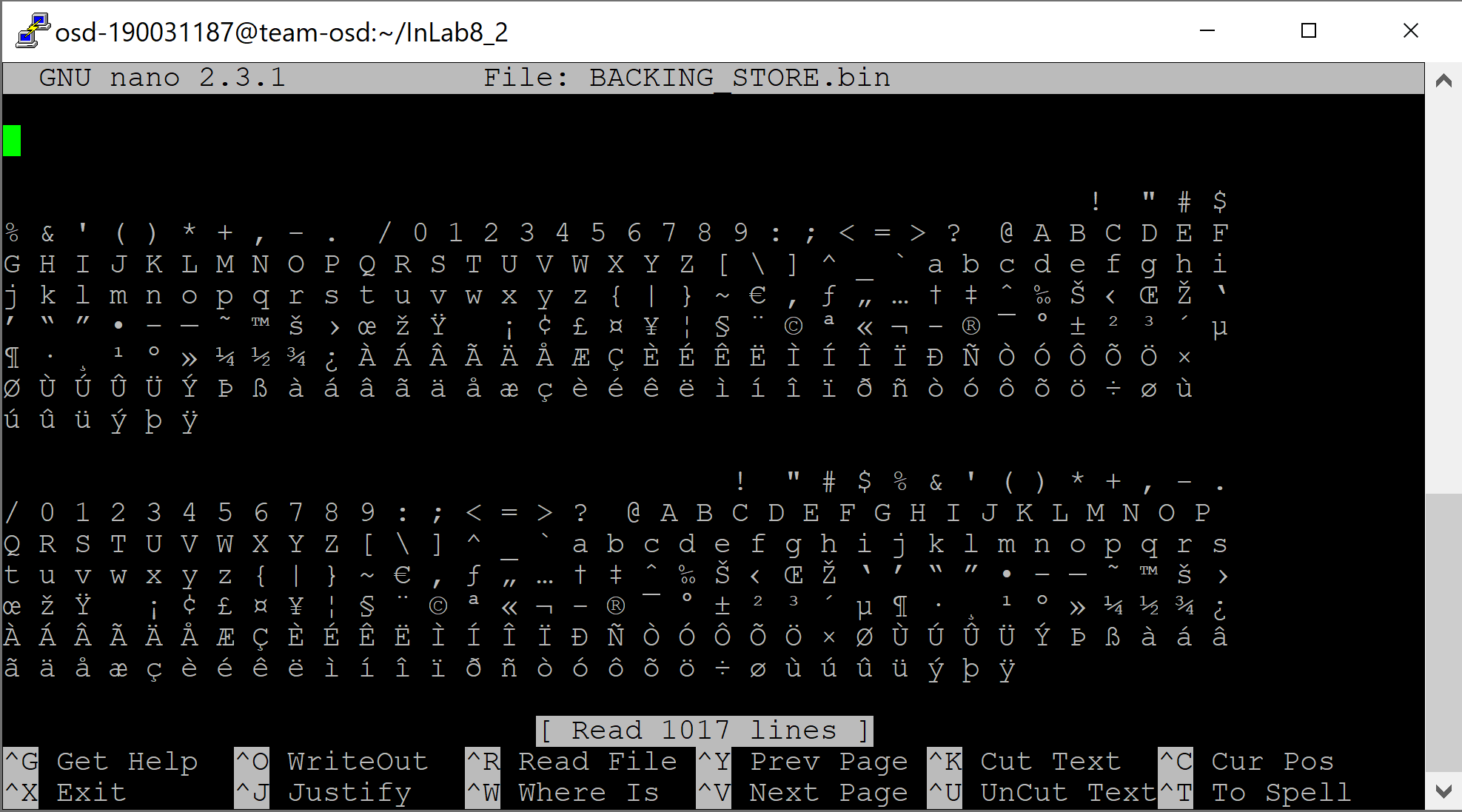
}

return;

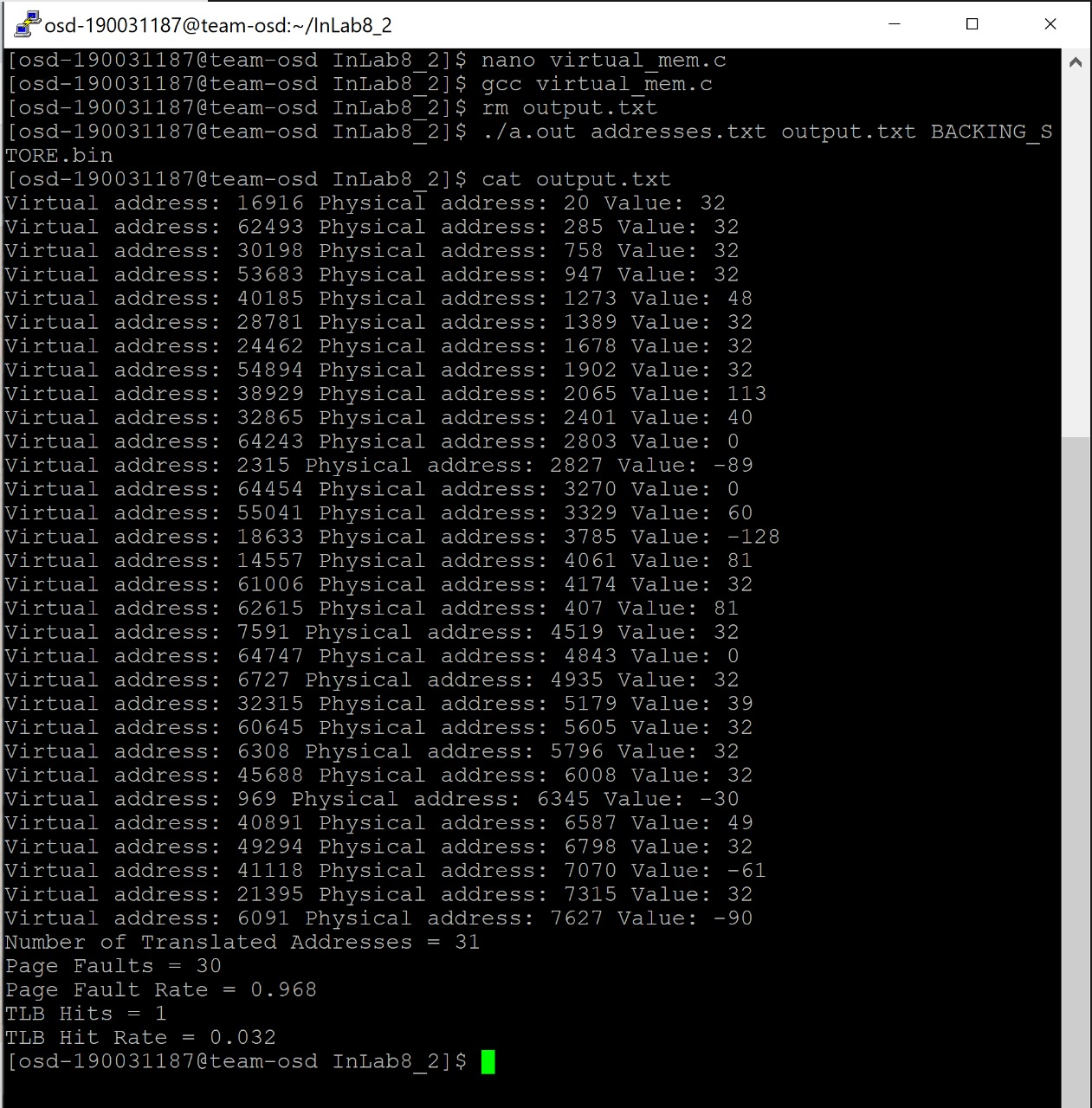
}

addresses.txt



BANKING\_STORE.bin

**OUTPUT**

****

**POSTLAB**

1.Write a program to demonstrate Accessing Memory with segmentation - linear translates.

**Ans:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

intmain() {

char name[100];

char \*description;

strcpy(name, "Zara Ali");

/\* allocate memory dynamically \*/

description = malloc( 200 \* sizeof(char) );

if( description == NULL ) {

fprintf(stderr, "Error - unable to allocate required memory\n");

}

else {

strcpy( description, "Zara ali a DPS student in class 10th");

}

printf("Name = %s\n", name ); printf("Description: %s\n", description );

}