**OSD Practical-9**

**Pre-Lab:**

1. Problems on page replacement and multilevel paging

Ans:

Problem:

Consider a system using multilevel paging scheme. The page size is 1 MB. The memory is byte addressable and virtual address is 64 bits long. The page table entry size is 4 bytes.

Find-

1. How many levels of page table will be required?
2. Give the divided physical address and virtual address.

Solution:

Given-

* Virtual Address = 64 bits
* Page size = 1 MB
* Page table entry size = 4 bytes

# Number of Bits in Frame Number-

We have, Page table entry size = 4 bytes = 32 bits

Thus, Number of bits in frame number = 32 bits

# Number of Frames in Main Memory-

We have, Number of bits in frame number = 32 bits

Thus,Number of frames in main memory = 232 frames

Size of Main Memory-

Size of main memory = Total number of frames x Frame size

= 232 x 1 MB

= 252 B

Thus, Number of bits in physical address = 52 bits

# Number of Bits in Page Offset-

We have,Page size = 1 MB = 220 B

Thus, Number of bits in page offset = 20 bits

Alternatively,Number of bits in page offset

= Number of bits in physical address – Number of bits in frame number

= 52 bits – 32 bits

= 20 bits

Process Size-

Number of bits in virtual address = 64 bits

Thus,Process size = 264 bytes

Number of Pages of Process-

Number of pages the process is divided = Process size / Page size

= 264 B / 1 MB

= 264 B / 220 B

= 244 pages

# Inner Page Table Size-

Inner page table keeps track of the frames storing the pages of process.

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

= Number of pages the process is divided x Page table entry size

= 244 x 4 bytes

= 246 bytes

Now, we can observe-

* The size of inner page table is greater than the frame size (1 MB).
* Thus, inner page table can not be stored in a single frame.
* So, inner page table has to be divided into pages.

# Number of Pages of Inner Page Table-

Number of pages the inner page table is divided = Inner page table size / Page size

= 246 B / 1 MB

= 246 B / 220 B

= 226 pages

Now, these 226 pages of inner page table are stored in different frames of the main memory.

# Number of Page Table Entries in One Page of Inner Page Table-

Number of page table entries in one page of inner page table = Page size / Page table entry size

= 1 MB / 4 B

= 220 B / 22 B

= 218 entries

# Number of Bits Required to Search an Entry in One Page of Inner Page Table-

One page of inner page table contains 218 entries.

Thus,

Number of bits required to search a particular entry in one page of inner page table = 18 bits

# Outer Page Table-1 Size-

Outer page table-1 is required to keep track of the frames storing the pages of inner page table.

Outer page table-1 size

= Number of entries in outer page table-1 x Page table entry size

= Number of pages the inner page table is divided x Page table entry size

= 226 x 4 bytes

= 228 bytes

= 256 MB

Now, we can observe-

* The size of outer page table-1 is greater than the frame size (1 MB).
* Thus, outer page table-1 can not be stored in a single frame.
* So, outer page table-1 has to be divided into pages.

# Number of Pages of Outer Page Table-1 -

Number of pages the outer page table-1 is divided = Outer page table-1 size / Page size

= 256 MB / 1 MB

= 256 pages

Now, these 256 pages of outer page table-1 are stored in different frames of the main memory.

# Number of Page Table Entries in One Page of Outer Page Table-1

Number of page table entries in one page of outer page table-1 = Page size / Page table entry size

= 1 MB / 4 B

= 220 B / 22 B

= 218 entries

Number of Bits Required to Search an Entry in One Page of Outer Page Table-1-

One page of outer page table-1 contains 218 entries.

Thus,

Number of bits required to search a particular entry in one page of outer page table-1 = 18 bits

# Outer Page Table-2 Size-

Outer page table-2 is required to keep track of the frames storing the pages of outer page table-1.

Outer page table-2 size = Number of entries in outer page table-2 x Page table entry size

= Number of pages the outer page table-1 is divided x Page table entry size

= 256 x 4 bytes

= 1 KB

Now, we can observe-

* The size of outer page table-2 is less than the frame size (16 KB).
* Thus, outer page table-2 can be stored in a single frame.
* In fact, outer page table-2 will not completely occupy one frame and some space will remain vacant.
* So, for given system, we will have three levels of page table.
* Page Table Base Register (PTBR) will store the base address of the outer page table-2.

Number of Bits Required to Search an Entry in Outer Page Table-2 Outer page table-2 contains 256 = 28 entries.

Thus,Number of bits required to search a particular entry in outer page table-2 = 8 bits

**In-Lab**

1. Write a program for the simulation of following paging algorithms FIFO LRU and MRU NFU.

Ans:

**Fifo.c code**

/\*

FIFO Page Replacement Algorithm

\*/

#include "stdio.h"

#include "stdlib.h"

#include "stdbool.h"

int pointer; int faults ,hits;

void print(int frame\_size,int frame[]) {

int i;

//printf("Printing the Frames: ");

for(i=0;i<frame\_size;i++) {

if(frame[i]==-1)

printf("- ");

else

printf("%d ",frame[i]);

}

printf("\n");

}

void add\_reference(int frame\_size,int frame[], int reference) {

int i;

bool alloted = false;

for(i=0;i<frame\_size;i++) {

if(frame[i]==reference) {

alloted = true;

printf(" Hit for %d | ", reference);

hits++; break;

}

else if(frame[i]==-1) {

alloted = true;

frame[i] = reference;

printf("Fault for %d | ", reference);

faults++; break;

}

}

if(alloted == false) {

faults++;

printf("Fault for %d | ", reference);

frame[pointer] = reference;

pointer = (pointer+1)%frame\_size;

}

print(frame\_size, frame);

}

int main() {

int frame\_size,i,number\_of\_references; printf("Enter frame size: ");

scanf("%d",&frame\_size);

int frame[frame\_size];

for(i=0;i<frame\_size;i++) {

frame[i] = -1;

}

print(frame\_size,frame);

printf("Enter the number of references: ");

scanf("%d",&number\_of\_references);

int reference[number\_of\_references];

for(i=0;i<number\_of\_references;i++) {

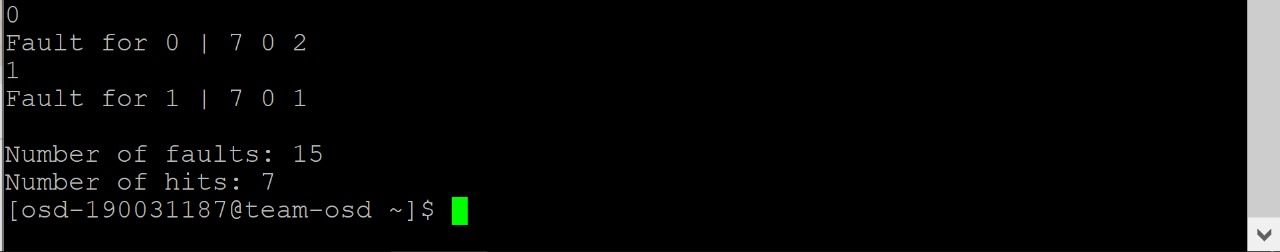
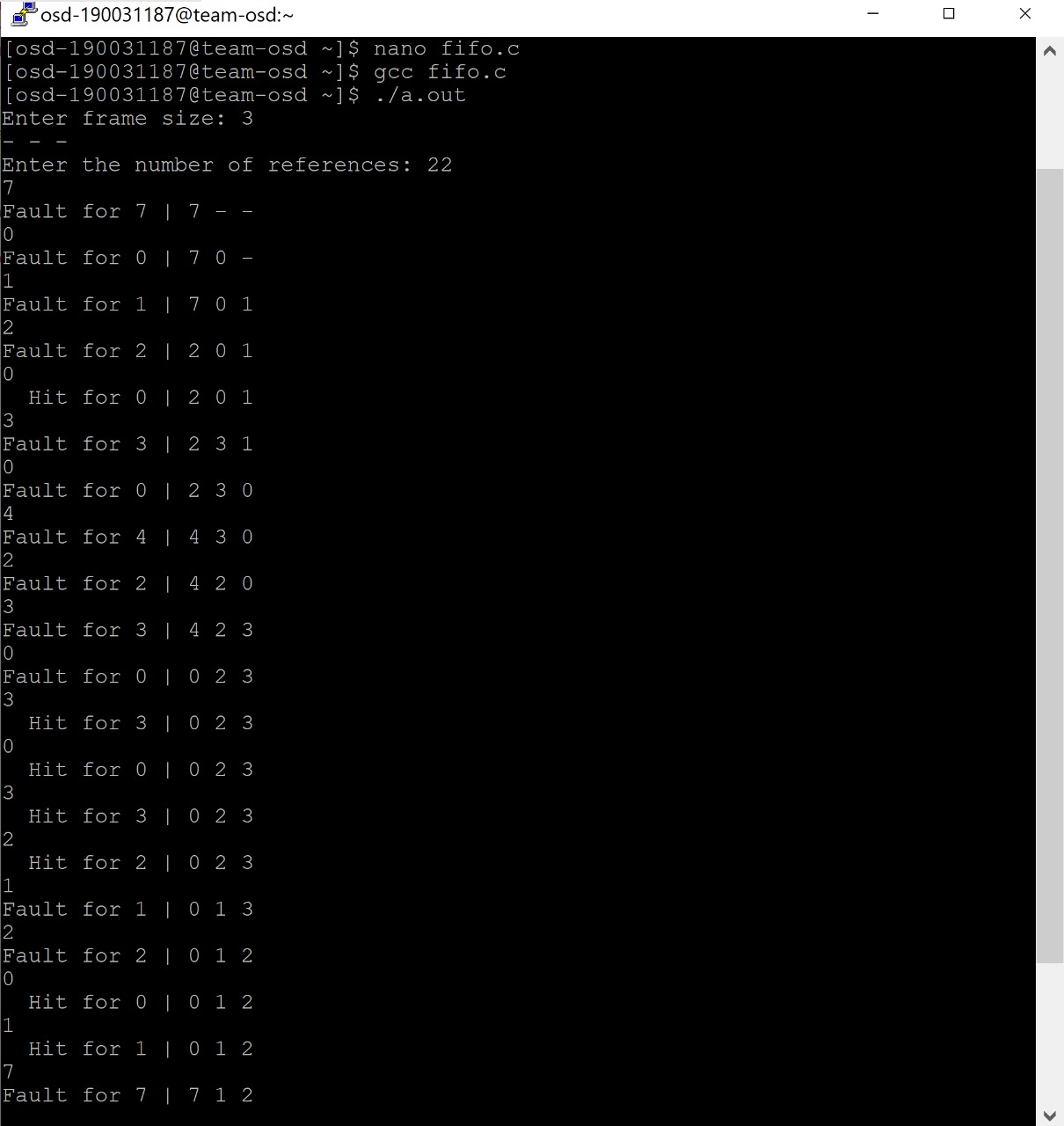
scanf("%d",&reference[i]);

add\_reference(frame\_size,frame,reference[i]);

}

printf("\nNumber of faults: %d \nNumber of hits: %d\n",faults,hits );

}



**lru.c code:**

/\* Least Recently Used Page Replacement Algorithm \*/

#include "stdio.h"

#include "stdlib.h"

#include "stdbool.h"

int pointer;

int faults, hits;

void print(int frame\_size,int frame[]) {

int i;

//printf("Printing the Frames: ");

for(i=0;i<frame\_size;i++) {

if(frame[i]==-1)

printf("- ");

else

printf("%d ",frame[i]);

}

printf("\n");

}

int predict(int reference\_length, int references[], int page\_no ,int frame\_size,int frame[], int start)

{

int pos = -1, farthest = start, i;

for(i=0;i<frame\_size;i++) {

int j;

for(j=start-1;j>=0;j--) {

if(frame[i]==references[j]) {

if(j<farthest) {

farthest=j; pos=i;

}

break;

}

}

if(j==page\_no)

return i;

}

if(pos == -1) return 0;

else return pos;

}

void add\_reference(int frame\_size,int frame[], int reference, int current\_position,int reference\_length, int references[])

{

int i;

bool allocated=false;

for(i=0;i<frame\_size;i++) {

if(frame[i]==reference) {

printf(" Hit for %d | ", reference);

hits++;

allocated = true;

break;

}

else if(frame[i]==-1) {

frame[i] = reference;

printf("Fault for %d | ", reference);

faults++;

allocated = true;

break;

}

}

if(allocated==false) {

int j =predict(reference\_length,references,current\_position,frame\_size,frame,current \_position+1);

frame[j] = reference;

printf("Fault for %d | ", reference);

faults++;

}

print(frame\_size, frame);

}

int main() {

int frame\_size,i,number\_of\_references; printf("Enter frame size: ");

scanf("%d",&frame\_size);

int frame[frame\_size];

for(i=0;i<frame\_size;i++)

frame[i] = -1;

print(frame\_size,frame);

printf("Enter the number of references: ");

scanf("%d",&number\_of\_references);

int reference[number\_of\_references];

for(i=0;i<number\_of\_references;i++) {

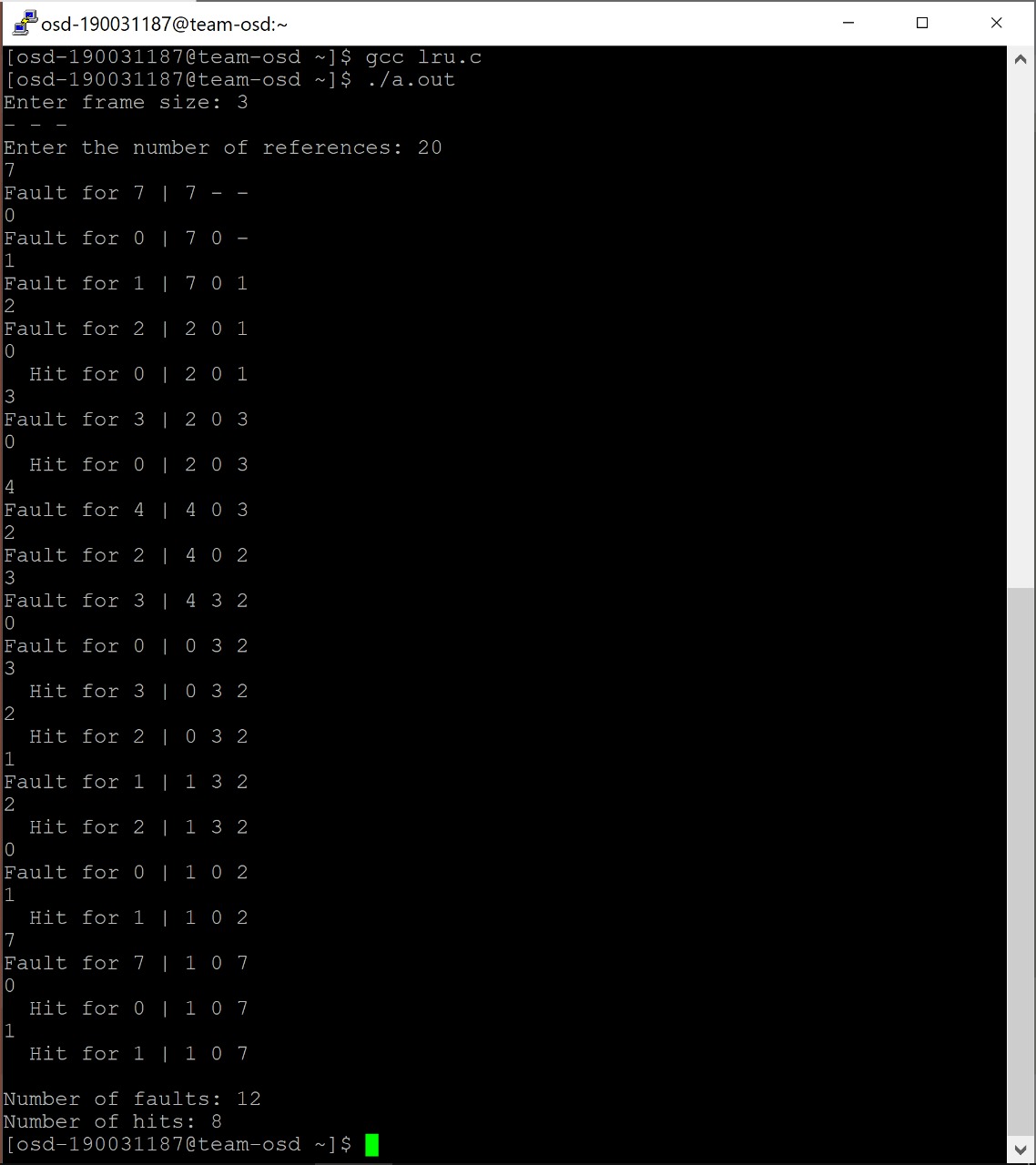
scanf("%d",&reference[i]);

add\_reference(frame\_size,frame,reference[i],i,number\_of\_references,ref erence);

}

printf("\nNumber of faults: %d \nNumber of hits: %d\n",faults,hits );

}



**Post-Lab:**

1.Program to demonstrate Multi-level Page Table Control Flow.

