Aisha Muhammad Nawaz
20L-0921
BSCS Section 4A1

National University of Computer and Emerging Sciences, Lahore Campus

Course: Operating Systems Course Code: CS 2006
Program: BS(Computer Science) Semester: Spring 2022 Due Date 3-June-2022 at 11:30 Total Marks: 35
Type: Assignment 3 Page(s): 2

SOLUTION:

Important Instructions:

1. Submit Soft copy of your solution using MS Word.
2. Late submission of your solution is not allowed.

Question 1: [5 marks]
Assume that the following three functions have already been implemented:

1. int getPageSize() //returns page size in bytes.
2. int* getPageTable(int processId) //returns the page table of the process whose id is processId 3. int loadFrame(int processId, int pageNo) //loads the given pager number of the given process in memory and returns the frame number where the page has been loaded.

When the process, whose id is processId, tries to access a virtual address (the address is stored in virtualAddr) belonging to a page that is not in memory, then a page fault exception will occur which will be handled by a page fault handler routine named as "handlePageFault". The purpose of this routine will be to load the page in memory. The signature of this routine is as follows:

```
void handlePageFault(int processId, int virtualAddr)
{
 //implement the routine
}
```

Your task is to implement this above routine with the help of routines already implemented.

ANSWER:

```
void handlePageFault(int processId, int virtualAddr)
{
 //implement the routine
int p=getPageSize();
int pageNo=virtualAddr/p;
int *pageTable=getPageTable(processId);
int frameNumber=loadFrame(processId,pageNo);
pageTable[pageNo]=frameNumber;
}
```

Aisha Muhammad Nawaz
20L-0921
BSCS Section 4A1

Question 2: [10 marks]

Show execution of the LRU page replacement algorithm on the following page reference string:

[10 marks]

2 3 1 4 1 5 3 2 1 4

Assume there are only three frames in the RAM. Show contents of memory frames after each page access from the reference string. (Please note that the number of boxes below maybe less or more depending upon the question. It, certainly, does not mean you have to utilize exactly the given number of boxes.)

2	2	2	4	4	3	3	3	4															
	3	3	3	5	5	5	1	1															
		1	1	1	1	2	2	2															

Department of Computer Science, FAST School of Computing FAST-NU, Lahore

Aisha Muhammad Nawaz
20L-0921
BSCS Section 4A1

Question 3: [10 marks]

Suppose that the page size is 128 bytes. Now consider process p1 whose page table is as follows:

0	5
1	10
2	9
3	6
4	Hard disk
5	1

Now assume that process p1 accesses the following virtual addresses:

a) 0 **b)** 139 **c)** 650 **d)** 257

1. Convert the above virtual addresses into physical addresses.

2. What is the range of virtual addresses of p1 that will cause page fault exception?

ANSWER:

PART 1

Page Size = 128 bytes
128 bytes = 128
Ceiling[log2(128)]= 7
so, 2^7

Virtual Address = Logical Address
Total Number of Pages= 6 (From Table)
Logical Address Space Size = Total Number of pages*Size of each page
= 6*128 = 768 bytes
Ceiling[log2(768)]= 10
So, 2^10

Bits needed to represent each page = 2^10/2^7 = 3 bits.
Total bits in Logical Address = 10 bits
Bits needed for offset = 10-3 = 7 bits.

Bits needed for physical memory address = ?
Frame Size = 128 bytes
Total Number of frames = 10 (From table)
Physical Address Space Size = Total Number of frames*Size of each frame
= 10 * 128 = 1280 bytes
Ceiling[log2(1280)]= 11
So, 11 bits required for physical address.

Aisha Muhammad Nawaz
20L-0921
BSCS Section 4A1

a) 0

0 in binary is: 0

7 bits for offset

0000000

Logical Address : 000 0000000

0 Page Number, Frame Number 5 in binary is 101

Physical Address : 0101 0000000 or (640)

b) 139

```
2| 139
2| 69    1
2| 34    1
2| 17    0
2|8      1
2| 4     0
2| 2     0
2| 1     0
```

139 in binary is : 10001011

7 bits for offset

0001011

Logical Address: 001 0001011

001 is 1 Page Number, Frame Number 10 in binary is 1010

Physical Address : 1010 0001011 or (1291)

c) 650

```
2| 650
2| 325  0
2| 162  1
2| 81   0
2| 40   1
2| 20   0
2| 10   0
2| 5    0
2| 2    1
2| 1    0
```

650 in binary is : 1010001010

7 bits for offset

0001010

Logical Address : 101 0001010

101 is 5 Page Number, Frame Number 1 in binary is 0001

Physical Address : 0001 0001010 or (138)

d) 257

```
2| 257
2| 128  1
2| 64    0
2| 32    0
2| 16    0
2|  8     0
2|  4     0
2|  2     0
2| 1      0
```

257 in binary is : 100000001

7 bits for offset
0000001

Logical Address : 010 0000001
010 is 2 Page Number, Frame Number 9 in binary is 1001
Physical Address : 1001 0000001 or (1153)

PART 2

Since Page Number 4 is Still on Hard disk accessing it will cause page fault exception.
Total bits required to represent page number Is = 3 bits

Since,
Page Number 4 in binary is 100

The range of virtual Addresses is
100 0000000
till
100 1111111

Which in decimal is 512 till 639

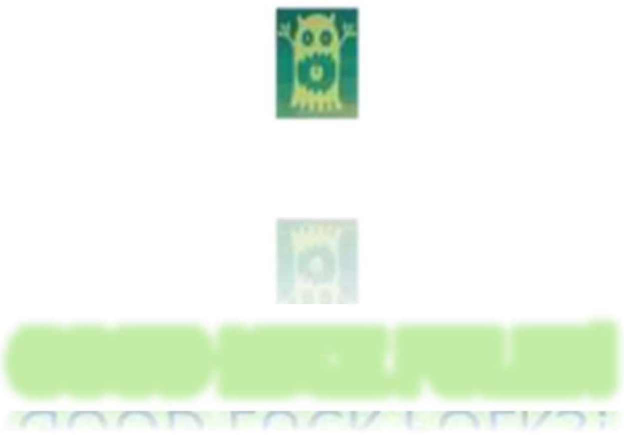
Aisha Muhammad Nawaz
20L-0921
BSCS Section 4A1

Question 4: [10 marks]

Consider the following code for a simple Stack:

```
class Stack {
private:
 int* a; // array for stack
 int max; // max size of array
 int top; // stack top
public:
 Stack(int m) {
  a = new int[m]; max = m; top = 0;
 }
 void push(int x) {
 while (top == max); // if stack is full then wait
  a[top] = x;
  ++top;
 }
 int pop() {
 while (top == 0); // if stack is empty then wait
 int tmp = top;
 --top;
 return a[tmp];
 }
};
```

Assuming the functions push and pop can execute concurrently, synchronize the code using semaphores. Also, replace the busy waiting with proper waiting.



Aisha Muhammad Nawaz
20L-0921
BSCS Section 4A1

ANSWER:

```
class Stack
{
 private:
 int* a; // array for stack
 int max; // max size of array
 int top; // stack top
Semaphore Mutex=1;    //To ensure mutual exclusion.
Semaphore Full=max;       // To indicate that stack is full.
Semaphore Empty=0;   // To indicate that stack is empty.

public:
 Stack(int m)
 {
  a = new int[m];
  max = m;
  top = 0;
 }
 void push(int x)
                        {
Wait(Full); // if stack is full then wait
Wait(Mutex);


  a[top] = x;
  ++top;

  Signal(Mutex);
  Signal(Empty);
                        }

 int pop()
             {
 Wait(Empty);  // if stack is empty then wait
 Wait(Mutex);

int tmp = top;
 --top;

Signal(Mutex);
Signal(Full);
return a[tmp];

           }


};
```