ALPARSLAN TURKES SCIENCE AND TECHNOLOGY UNIVERSITY

OPERATING SYSTEM

ASSIGNMENT 4

FURKAN SAY

170101019

1. Suppose we have 210 bytes of virtual memory and 28 bytes of physical main memory. Suppose the page size is 24 bytes.

(a) How many pages are there in virtual memory?

64

(b) How many pages frames are there in main memory?

16

(c) How many entries are in the page table for a process that uses all of virtual memory?

64

2. Consider a computer system with a virtual-memory space of 232 bytes and 218 bytes of physical memory. The virtual memory is implemented by paging, and the page size is 2048 bytes. Consider a user process generates a virtual address 12345678 (in hexadecimal). If the page table for the process contains frame number 34(in hexadecimal) at entry 0001 0010 0011 0100 0101 0(in binary), what is the corresponding physical address of the virtual address 12345678 (in hexadecimal)? Explain your answer.

12345678 = 0001 0010 0011 0100 0101 0110 0111 1000

table = VM/PS 2^32 / 2^11 = 21

Thus, 21 bits are used for page table entry and its frame number, whereas, last 11 bits are used from displacement into page.

Page table entry = 00010010 00110100 01010

Page size = 11001111000

34 = 00110100

I don't understant process contains frame number but I think it can be

if it must be 34 hex (binary 00110100) entry it can be

yes it contains . It is the physical address of the lower binary with the physical address specified as 2 to the 18th

physical address : 0000 0100 1000 1101 0001

3. What is a translation look-aside buffer (TLB)? Do you need a TLB for correct execution of programs?

TLB is a cache used to reduce the time the user spends accessing their memory location. If the processor uses virtual memory, we can give an example of virtualization from real life. It speeds up the conversion of virtual addresses to physical addresses. We can run an android software we wrote quickly using virtualization. TLB operates between CPU and main memory and page table is stored in ram. The difference between TLB and CPU cache is that TLB speeds up address translation for virtual memory so that the page table does not need to be accessed for each address. CPU cache, on the other hand, cache temporarily stores the requested information and the next access to the same information will be faster Uses the CPU cache to store data

4. Given the following sequence for page references: 1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9, 5, 4, 5, 4, 2 and the frequently of use (for LFU and MFU):

FIFO:

Windows PowerShell

```
PS C:\Users\admin\eclipse-workspace\Assingment4\src> java assingment4.java 3 FIFO data4.txt
1 --> [1, null, null]
2 --> [1, 2, null]
3 --> [1, 2, 3]
4 --> [4, 2, 3] page fault
5 --> [4, 5, 3] page fault
3 --> [4, 5, 3]
4 --> [4, 5, 3]
1 --> [4, 5, 1] page fault
6 --> [6, 5, 1] page fault
7 --> [6, 7, 1] page fault
8 --> [6, 7, 8] page fault
7 --> [6, 7, 8]
8 --> [6, 7, 8]
9 --> [9, 7, 8] page fault
7 --> [9, 7, 8]
8 --> [9, 7, 8]
9 --> [9, 7, 8]
5 --> [9, 5, 8] page fault
4 --> [9, 5, 4] page fault
5 --> [9, 5, 4]
4 --> [9, 5, 4]
2 --> [2, 5, 4] page fault

Total number of page faults is 10.
PS C:\Users\admin\eclipse-workspace\Assingment4\src>
```

LRU:

```
PS C:\Users\admin\eclipse-workspace\Assingment4\src> java assingment4.java 3 LRU data4.txt
1 --> [1, null, null]
2 --> [1, 2, null]
3 --> [1, 2, 3]
4 --> [4, 2, 3] page fault
5 --> [4, 5, 3] page fault
3 --> [4, 5, 3]
4 --> [4, 5, 3]
1 --> [4, 1, 3] page fault
6 --> [4, 1, 6] page fault
7 --> [7, 1, 6] page fault
8 --> [7, 8, 6] page fault
7 --> [7, 8, 6]
8 --> [7, 8, 6]
9 --> [7, 8, 9] page fault
7 --> [7, 8, 9]
8 --> [7, 8, 9]
9 --> [7, 8, 9]
5 --> [5, 8, 9] page fault
4 --> [5, 4, 9] page fault
5 --> [5, 4, 9]
4 --> [5, 4, 9]
2 --> [5, 4, 2] page fault

Total number of page faults is 10.
```

LFU:

1:0 , 2:2 , 3:0 , 4:4 , 5:3 , 6:0 , 7:0 , 8:0 , 9:0 Least Frequently

MFU:

```
1 -> 1
2 -> 1 2
3 -> 1 2 3
4 -> 1 4 3 page fault
5 -> 5 4 3 page fault
3 -> 5 4 3
4 -> 5 4 3
1 -> 5 4 1 page fault
6 -> 5 4 6 page fault
7 -> 5 4 7 page fault
8 -> 5 8 7
7 -> 5 8 7
8 -> 5 8 7 page fault
9 -> 9 8 7
7 -> 9 8 7
8 -> 9 8 7
9 -> 9 8 7 page fault
5 -> 5 8 7 page fault
4 -> 5 4 7
5 -> 5 4 7
4 -> 5 4 7

Total number of page faults is 8
```

1:0 , 2:2 , 3:0 , 4:4 , 5:3 , 6:0 , 7:0 , 8:0 , 9:0 Most Frequently

CODE:

```
import java.io.BufferedReader;

import java.io.File;

import java.io.FileInputStream;

import java.io.InputStreamReader;

import java.util.Arrays;

import java.util.Scanner;
```

```
public class assingment4 {

	static class twovalue {
	   public int path;
	   public String pagefault;
	   public int count;

	   public twovalue(int path, String pagefault,int count) {
	      this.path = path;
	      this.pagefault = pagefault;
	      this.count = count;
	   }

	   public int getpath() {
	      return path;
	   }

	   public int getcount() {
	      return count;
	   }
	   public String getpagefault() {
	      return pagefault;
	   }
	}

	public static void main(String[] args) {
```

```
		try {


			String filepath = args[2];//key.nextLine();

			String all = "";

			String method = args[1];

			Scanner key = new Scanner(System.in);

			String memory =args[0];

			File file = new File(filepath);

		FileInputStream fileInputStream;

			fileInputStream = new FileInputStream(file);

			InputStreamReader inputStreamReader = new
InputStreamReader(fileInputStream);


			BufferedReader bufferedReader = new BufferedReader(inputStreamReader);

			while(bufferedReader.readLine() != null) {

				String temp = bufferedReader.readLine();

				if(!temp.contains("#")) {

					all +=temp;

				}

			}

			String[] numberstr = all.split(" ");

			int[] numbers =
Arrays.stream(numberstr).mapToInt(Integer::parseInt).toArray();

			//System.out.println(Arrays.toString(numbers));

			String[] memoryarray = new String[Integer.valueOf(memory)];

			if(method.equals("FIFO")) {

				fifo(numberstr,memoryarray, Integer.valueOf(memory),0);

			}if(method.equals("LRU")) {

				lru(numberstr,memoryarray, Integer.valueOf(memory),0);
```

```
                }

            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }


    }


    public static void lru(String[] number,String[] memoryarray,int memory,int count)
    {
        String[] page = new String[memory];

        for (int i = 0; i < number.length-1; i++) {
            if (page[i % memory] == null && memoryarray[i % memory] == null) {
                page[i % memory] = number[i+1];
                memoryarray[i % memory] = String.valueOf(i % memory);
                System.out.println(number[i+1]+" --> " + Arrays.toString(page) );
                }else {


    //if(diff(page,Integer.valueOf(findold(memoryarray)),Integer.valueOf(number[i+1]))) {
                        //System.out.println("diff");}
                        //System.out.println(findold(memoryarray) + " olds");
                        int old =
findold(memoryarray,page,Integer.valueOf(number[i+1]),count).path;
                        String pagefault =
findold(memoryarray,page,Integer.valueOf(number[i+1]),count).pagefault;
```

```
                                        count =
findold(memoryarray,page,Integer.valueOf(number[i+1]),count).count;
                                        //System.out.println(count);
                                        page[old] = number[i+1];


                                        memoryarray[old] = String.valueOf(2);
                                        //System.out.println(old + " old");
                                        //System.out.println(Arrays.toString(memoryarray) + "
memoryarray");
                                        for (int j = 0; j < memory; j++) {


                                                if (old != j && Integer.valueOf(memoryarray[j]) != 0 ) {
                                                        //System.out.println(j + " dd");
                                                        memoryarray[j] =
String.valueOf(Integer.valueOf(memoryarray[j]) - 1);
                                                }
                                        }
                                        System.out.println(number[i+1]+" --> " + Arrays.toString(page)
+" "+ pagefault);




                                }



                }
                System.out.println();
                System.out.println("Total number of page faults is " + count+".");
```

```
	}
	public static void fifo(String[] number,String[] memoryarray,int memory,int count)
	{
		String[] page = new String[memory];

		for (int i = 0; i < number.length-1; i++) {
			if (page[i % memory] == null && memoryarray[i % memory] == null) {
				page[i % memory] = number[i+1];
				memoryarray[i % memory] = String.valueOf(i % memory);
				System.out.println(number[i+1]+" --> " + Arrays.toString(page) );
				}else {

	//if(diff(page,Integer.valueOf(findold(memoryarray)),Integer.valueOf(number[i+1]))) {
					//System.out.println("diff");}
					//System.out.println(findold(memoryarray) + " olds");
					int old =
findoldfifo(memoryarray,page,Integer.valueOf(number[i+1]),count).path;
					String pagefault =
findoldfifo(memoryarray,page,Integer.valueOf(number[i+1]),count).pagefault;
					count =
findoldfifo(memoryarray,page,Integer.valueOf(number[i+1]),count).count;
					//System.out.println(count);
					if(!issame(page, old,Integer.valueOf(number[i+1]) )){page[old] =
number[i+1];

					memoryarray[old] = String.valueOf(2);
					//System.out.println(old + " old");
					//System.out.println(Arrays.toString(memoryarray) + "
memoryarray");
					for (int j = 0; j < memory; j++) {
```

```
                                        if (old != j && Integer.valueOf(memoryarray[j]) != 0 ) {
                                                //System.out.println(j + " dd");
                                                memoryarray[j] =
String.valueOf(Integer.valueOf(memoryarray[j]) - 1);
                                        }
                                }}
                                System.out.println(number[i+1]+" --> " + Arrays.toString(page)
+" "+ pagefault);




                        }


                }
                System.out.println();
                System.out.println("Total number of page faults is " + count+".");
        }

        public static boolean issame(String[] page,int path,int value) {
                if(Integer.valueOf(page[path]) == value ) {
                        return true;
                }
                return false;
        }
```

```
public static String diff(String[] page,int t)
{
	String path = "";
	for (int i = 0; i < page.length; i++) {
		if (Integer.valueOf(page[i]) == t) {
			path = String.valueOf(i);
		}
	}

	return path;

}

public static twovalue findold(String[] memoryarray,String[] page, int t,int count)
{	String path=diff(page,t);
	String pagefault="";

	if(path.equals("")) {
		for (int i = 0; i < memoryarray.length; i++) {
			if(Integer.valueOf(memoryarray[i]) == 0) {
				path = String.valueOf(i);
			}
		}
		pagefault = "page fault";
		count+=1;
	}
```

```
            twovalue twovalue = new twovalue(Integer.valueOf(path), pagefault,count);
            return twovalue;
        }

        public static twovalue findoldfifo(String[] memoryarray,String[] page, int t,int count)
        {       String path=diff(page,t);
            String pagefault="";

            if(path.equals("")) {
            for (int i = 0; i < memoryarray.length; i++) {
                if(Integer.valueOf(memoryarray[i]) == 0) {
                    path = String.valueOf(i);
                }
            }
            pagefault = "page fault";
            count+=1;
        }


        twovalue twovalue = new twovalue(Integer.valueOf(path), pagefault,count);
        return twovalue;
}




}
```

In my "LRU" method, I first added the numbers I got from the text file to the array, when there was no space in the memory, I added the new number instead of the oldest added number and updated my memory array.

In my "fifo" method, the only difference from "lru" is that if the number needed is in the array, I did not update my memory array, so the number added to the first system became the changed number.

"'issame" returning boolean method is the method that checks if the number needed to use the page fault structure is in the array.

The "findold" method gives the path where the oldest loaded number is in the array after checking it, which is added in the array as it can be seen from its name, and I protected its path with another memory array while it was being added.

"findoldfifo", this method is the same as the method above, but while updating the memory location, that is, the priority of the number needed in the LRU, I removed it here, if the incoming number is in the array, it goes to the next number without updating the memory array.

In my "diff" method, the desired number in the LRU is set if it is in the page array, if it is old in its memory array, it is entered in the new system.

You can run it any way you want from the command section using the args structure.