COEN 346

OPERATING SYSTEMS

PROGRAMMING PROJECT #4

Virtual Memory Manager

Section: YK-X

Name: Jeffrey Tang ID:40037656

Name: Hong Yi Wang ID: 40034150

Course Mentor:

Lab Instructor:

Date Performed: 30 March 2018

Date Submitted: 13 April 2018

***"We certify that this submission is my original work and meets the Faculty's Expectations of Originality*.**

1. OBJECTIVE

The goal of this project is to implement and simulate a virtual memory manager. It will translate logical memory addresses into physical memory addresses. The program will read through a file that will contain logical addresses then use a TLB and a Page Table to translate the logical addresses into physical addresses.

2. PROCEDURE/DISCUSSION (MAY INCLUDE BELOW POINTS)

FOR ALL PARTS

- TECHNICAL/FUNCTIONAL DESCRIPTION
    - We have to create the structures of the page table, the logical addresses and the physical addresses. For the physical and logical addresses, we have to decide how we should store them. The logical address will be stored in an array. The page table will also be stored in an array. The physical memory is a 2D array. TLB will a structure containing the page number and frame number.
    - We have to write a method to mask the extra bits off of the logical address because depending on the information we need, we have to retrieve the either the offset or the page number.
    - We need a function to get the next frame within the physical memory in order to transfer the translated address from logical memory.
    - We need a method to retrieve data from the given file called the bin file. It will act as the backing store for processes. Data stored in the backing store are values that will be outputted according to the physical address.
    - We need a method to concatenate the new physical address value translated from the page number and offset of the logical address.
    - We need methods to calculate the hit rate as well as the page fault rates.
- PROGRAM FLOW
    - Initialise every value in the TLB to 0.
    - Initialise the page table to -1 because the values in the page table cannot be equal to 0. It will be counted as a value if doing so.
    - Open the “addresses” file and read through the value inside. They are all logical addresses and they are stored in the array. Once it has finished reading, close the file.
    - Then we create the output file “pbs_output.txt”. In a for loop of the size of the logical address array, we will call the method of bit masking for page number and offset.
    - Then first we will check with the TLB to see if it is already store there, if it is, it will directly output the corresponding physical address an increment the hit rate.

- If it doesn't, it will check whether it is in the page table. If it isn't, it will increase the page fault count. Then it will retrieve the data according to the page number in the "bin file" (backing store). Then call the method to concatenate into a physical address. Then update the TLB with the page number.
- If it is in the page table, call the concatenate method to get the physical address. Then update the TLB with the page number.
- Open the "pbs_output.txt" and write the according to the given format.

- IMPORTANT FUNCTIONS / TEMINOLOGY

getDataFromBackingStore(int pageNumber) is a method to retrieve the value that will be outputted according to the physical address. First we create a pointer of type FILE. Then we create a char array of size 256. Then we open the "bin file" and read in it. We create an Int variable locationOfPage that is the 256*pageNumber as well as free_frame that calls the method of getNextFreeFrame(). Then we seek the FILE f using fseek() and store the values in "storage". Then in a for loop until the size the 256, store the value at storage[i] in physicalMemory. Finally return free_frame.

```
52  int getDataFromBackingStore(int pageNumber) {
53          FILE *f;
54          char storage[256];
55
56          //open file in read mode
57          f = fopen("BACKING_STORE.bin", "r");
58
59          //a size of page is 256 bytes, so if we multiply by page number, we will get to the right page.
60          int locationOfPage = 256 * pageNumber;
61
62          //Looking for the location of the page
63          fseek(f, locationOfPage, SEEK_SET);
64
65          //Store data in temporary storage in memory
66          fread(storage, 1, 256, f);
67
68          int free_frame = getNextFreeFrame();
69          for (int i = 0; i < 256; i++)
70          {
71                  physicalMemory[(free_frame*256) + i] = (int) storage[i]; //store the page in a free frame
72          }
73
74          fclose(f);
75
76          return free_frame;
77  }
```

Figure 1: getDataFromBackingStore method

3. RESULTS (MAY INCLUDE PICTURES)

THE CODE WORKS PARTIALLY

The first few outputs correspond to the given output file, however the rest of the values do not correspond to the example output file.

```
1  Virtual address: 16916 Physical address: 20 Value: 0
2  Virtual address: 62493 Physical address: 285 Value: 0
3  Virtual address: 30198 Physical address: 758 Value: 29
4  Virtual address: 53683 Physical address: 947 Value: 108
5  Virtual address: 40185 Physical address: 1273 Value: 0
6  Virtual address: 28781 Physical address: 1389 Value: 0
7  Virtual address: 24462 Physical address: 1678 Value: 23
8  Virtual address: 48399 Physical address: 1807 Value: 67
9  Virtual address: 64815 Physical address: 2095 Value: 75
10 Virtual address: 18295 Physical address: 2423 Value: -35
11 Virtual address: 12218 Physical address: 2746 Value: 11
12 Virtual address: 22760 Physical address: 3048 Value: 0
13 Virtual address: 57982 Physical address: 3198 Value: 56
14 Virtual address: 27966 Physical address: 3390 Value: 27
15 Virtual address: 54894 Physical address: 3694 Value: 53
16 Virtual address: 38929 Physical address: 3857 Value: 0
```

Figure 2: Correct output

```
1  Virtual address: 16916 Physical address: 20 Value: 0
2  Virtual address: 62493 Physical address: 29 Value: 0
3  Virtual address: 30198 Physical address: 246 Value: 29
4  Virtual address: 53683 Physical address: 179 Value: 108
5  Virtual address: 40185 Physical address: 249 Value: -52
6  Virtual address: 28781 Physical address: 109 Value: -52
7  Virtual address: 24462 Physical address: 142 Value: 23
8  Virtual address: 48399 Physical address: 15 Value: 67
9  Virtual address: 64815 Physical address: 47 Value: 75
10 Virtual address: 18295 Physical address: 119 Value: -35
11 Virtual address: 12218 Physical address: 186 Value: 11
12 Virtual address: 22760 Physical address: 232 Value: -52
13 Virtual address: 57982 Physical address: 126 Value: 56
14 Virtual address: 27966 Physical address: 62 Value: 27
15 Virtual address: 54894 Physical address: 110 Value: 53
16 Virtual address: 38929 Physical address: 17 Value: 0
17 Virtual address: 32865 Physical address: 97 Value: 0
18 Virtual address: 64243 Physical address: 243 Value: -68
19 Virtual address: 2315 Physical address: 11 Value: 66
```

Figure 3: Test Output

4. CONCLUSION (MAY INCLUDE BELOW POINTS)

- LEARNING ASPECT
  - We learned about how we transform logical memory addresses to physical memory addresses. This version of the transformation uses the TLB. Using the TLB accelerates the fetching and matches of processes.
- CHALLENGES FACED
  - The configuration of the backing store was not easy. We had to understand clearly what it does in order to implement it. Moreover, we had to use methods that we have never seen before in order to implement it.