**Student name:** Federico Watkins

**CSIS 3810:** Operating Systems

**Chapter:** 9 Virtual Reality

**Assignment:** 7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1. 🡪 9.18 | 1. 🡪 9.19 | 1. 🡪 9.21 | 1. 🡪 9.22 | 1. 9.31 |

1. A certain computer provides its users with a virtual memory space of 232 bytes. The computer has 222 bytes of physical memory. The virtual memory is implemented by paging, and the page size is 4,096 bytes. A user process generates the virtual address 11123456. Explain how the system establishes the corresponding physical location. Distinguish between software and hardware operations.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Page Table Size** | | | | | **Page Size** | | |
| 1 | 1 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0001 | 0001 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 |

**Hardware and Software Operations**

When a physical location is established the system usually performs DAT and faulty page a set of operations is to service a faulty page if it occurs. DAT operations are handled by the hardware part of the system because it divides the two sets of operations depending on their needs towards resources, for better results. Servicing a faulty page, reading the resulting page and repeating the process whenever required is handled by the software operations.

1. Assume that we have a demand-paged memory. The page table is held in registers. It takes 8 milliseconds to service a page fault if an empty frame is available or if the replaced page is not modified and 20 milliseconds if the replaced page is modified. Memory-access time is 100 nanoseconds. Assume that the page to be replaced is modified 70 percent of the time. What is the maximum acceptable page-fault rate for an effective access time of no more than 200 nanoseconds?

0.2 **µ**sec = (1 − *P*) × 0.1 **µ**sec + (0.3*P*) × 8 millisecond + (0.7*P*) × 20 millisecond

0.1 = −0*.*1*P* + 2400 *P* + 14000 *P*

0.1 ≈ 16,400 *P*

*P* ≈ 0.000006

*P* = 6.0 × 10-7

1. Consider the following page reference string:

7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0, 1.

Assuming demand paging with three frames, how many page faults would occur for the following replacement algorithms?

* LRU replacement

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **2** | **3** | **1** | **2** | **5** | **3** | **4** | **6** | **7** | **7** | **1** | **0** | **5** | **4** | **6** | **2** | **3** | **0** | **1** |
| 7 | 7 | 7 | 1 |  | 1 | 3 | 3 | 3 | 7 |  | 7 | 7 | 5 | 5 | 5 | 2 | 2 | 2 | 1 |
|  | 2 | 2 | 2 |  | 2 | 2 | 4 | 4 | 4 |  | 1 | 1 | 1 | 4 | 4 | 4 | 3 | 3 | 3 |
|  |  | 3 | 3 |  | 5 | 5 | 5 | 6 | 6 |  | 6 | 0 | 0 | 0 | 6 | 6 | 6 | 0 | 0 |

|  |
| --- |
| **Number of faults =** 18 |

* FIFO replacement

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **2** | **3** | **1** | **2** | **5** | **3** | **4** | **6** | **7** | **7** | **1** | **0** | **5** | **4** | **6** | **2** | **3** | **0** | **1** |
| 7 | 7 | 7 | 1 |  | 1 |  | 1 | 6 | 6 |  | 6 | 0 | 0 | 0 | 6 | 6 | 6 | 0 | 0 |
|  | 2 | 2 | 2 |  | 5 |  | 5 | 5 | 7 |  | 7 | 7 | 5 | 5 | 5 | 2 | 2 | 2 | 1 |
|  |  | 3 | 3 |  | 3 |  | 4 | 4 | 4 |  | 1 | 1 | 1 | 4 | 4 | 4 | 3 | 3 | 3 |

|  |
| --- |
| **Number of faults =** 17 |

* Optimal replacement

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **2** | **3** | **1** | **2** | **5** | **3** | **4** | **6** | **7** | **7** | **1** | **0** | **5** | **4** | **6** | **2** | **3** | **0** | **1** |
| 7 | 7 | 7 | 1 |  | 1 |  | 1 | 1 | 1 |  |  | 1 |  | 1 | 1 | 1 | 1 |  |  |
|  | 2 | 2 | 2 |  | 5 |  | 5 | 5 | 5 |  |  | 5 |  | 4 | 6 | 2 | 3 |  |  |
|  |  | 3 | 3 |  | 3 |  | 4 | 6 | 7 |  |  | 0 |  | 0 | 0 | 0 | 0 |  |  |

|  |
| --- |
| **Number of faults =** 13 |

1. The page table shown in Figure 9.32 is for a system with 16-bit virtual and physical addresses and with 4,096-byte pages. The reference bit is set to 1 when the page has been referenced. Periodically, a thread zeroes out all values of the reference bit. A dash for a page frame indicates the page is not in memory. The page-replacement algorithm is localized LRU, and all numbers are provided in decimal.

|  |  |  |
| --- | --- | --- |
| **Page** | **Page Frame** | **Reference Bit** |
| 0 | 9 | 0 |
| 1 | 1 | 0 |
| 2 | 14 | 0 |
| 3 | 10 | 0 |
| 4 | – | 0 |
| 5 | 13 | 0 |
| 6 | 8 | 0 |
| 7 | 15 | 0 |
| 8 | – | 0 |
| 9 | 0 | 0 |
| 10 | 5 | 0 |
| 11 | 4 | 0 |
| 12 | – | 0 |
| 13 | – | 0 |
| 14 | 3 | 0 |
| 15 | 2 | 0 |

1. Convert the following virtual addresses (in hexadecimal) to the equivalent physical addresses. You may provide answers in either hexadecimal or decimal. Also set the reference bit for the appropriate entry in the page table.

• 0xE12C

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Page Table Index** | **Byte Index** | | |  |  |  |
| 1110 | 0001 | 0010 | 1110 |  |  |  |
| 11102 | = | E16 | 🡪 | E16 | = | 1410 |
| **Result** | = | 0x312C |  |  |  |  |

• 0x3A9D

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Page Table Index** | **Byte Index** | | |  |  |  |
| 0011 | 1010 | 1001 | 1101 |  |  |  |
| 00112 | = | 316 | 🡪 | 316 | = | 310 |
| **Result** | = | 0xAA9D |  |  |  |  |

• 0xA9D9

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Page Table Index** | **Byte Index** | | |  |  |  |
| 1010 | 1001 | 1101 | 1001 |  |  |  |
| 10102 | = | A16 | 🡪 | A16 | = | 1010 |
| **Result** | = | 0x59D9 |  |  |  |  |

• 0x7001

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Page Table Index** | **Byte Index** | | |  |  |  |
| 0111 | 0000 | 0000 | 0001 |  |  |  |
| 01112 | = | 716 | 🡪 | 716 | = | 710 |
| **Result** | = | 0xF001 |  |  |  |  |

• 0xACA1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Page Table Index** | **Byte Index** | | |  |  |  |
| 1010 | 1100 | 1010 | 0001 |  |  |  |
| 10102 | = | A16 | 🡪 | A16 | = | 1010 |
| **Result** | = | 0x5CA1 |  |  |  |  |

1. Using the above addresses as a guide, provide an example of a logical address (in hexadecimal) that results in a page fault.

Choices are pages 4, 8, 12, and 13. Hence, anything that begins with the following hexadecimal sequence:

|  |  |  |  |
| --- | --- | --- | --- |
| 0x4... | 0x8... | 0xC... | 0xD.... |

1. From what set of page frames will the LRU page-replacement algorithm choose in resolving a page fault?

Page table entries that have a reference bit of zero as do the following frames

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 9 | 1 | 14 | 13 | 8 | 0 | 4 |

1. Consider a demand-paging system with a paging disk that has an average access and transfer time of 20 milliseconds. Addresses are translated through a page table in main memory, with an access time of 1 microsecond per memory access. Thus, each memory reference through the page table takes two accesses. To improve this time, we have added an associative memory that reduces access time to one memory reference if the page-table entry is in the associative memory. Assume that 80 percent of the accesses are in the associative memory and that, of those remaining, 10 percent (or 2 percent of the total) causes page faults. What is the effective memory access time?

Effective access time = (0.8) × (1 **µ**sec) + (0.1) × (2 **µ**sec) + (0.1) × (5002 **µ**sec)

= 501.2 **µ**sec

= 0.5 millisecond

References:

1. Operating System Concepts, Ninth Edition, Abraham Silberschartz, Peter bear Galvin, Greg Gagne. Entire Chapter 9 for each question in its respective section.
2. http://www.ualberta.ca/CNS/RESEARCH/LinuxClusters/mem.html
3. http://csserver.evansville.edu/~amr63/linux/unix\_programming/unix-memory/unix-memory.html
4. <http://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Memory/virtual.html>
5. Tutorials: Virtual\_Memory.ppt, Dr. Szabo
6. Memory\_mapping.pdf Dr. Szabo

I learned that Virtual memory is a technique that allows the execution of processes that are not completely in memory. Its main advantage is that programs can be larger than physical memory. Virtual memory abstracts main memory into an extremely large, uniform array of storage, separating logical memory as viewed by the user from physical memory. Now I understand how it frees programmers from the concerns of memory storage limitations. Also, how it allows processes to share files easily and to implement shared memory. The assignments exposed me to the complexity involved with implementing virtual memory. I see how it my substantially decrease performance if it is used carelessly. I am better prepared to explain the concepts of demand paging, page-replacement algorithms, and allocation of page frames, as well as discuss the principles of the working-set model. The relationship between shared memory and memory-mapped files, and how kernel memory is managed was also explored.