|  |
| --- |
| **Q 1. How memory is protected in a paged environment?** |
| * Memory protection implemented by associating protection bit with each frame to indicate if access is allowed * **Valid-invalid** bit attached to each entry in the page table:   + “valid” indicates that the associated page is in the process’ logical address space, and is thus a legal page   + “invalid” indicates that the page is not in the process’ logical address space   + Or use **page-table length register** (**PTLR**) * Any violations result in a trap to the kernel   Can also add more bits to indicate if read-only, read-write, execute-only is allowed |
| **Q2. Consider the following segmentation table**  **Segment Base Length**  **0 219 600**  **1 2300 14**  **2 90 100**  **3 1327 580**  **4 1952 96**  **What are the physical addresses for the following logical addresses?**   1. **0, 430** 2. **1, 10** 3. **3, 400** 4. **4, 112** |
| (a) 219 + 430 = 649  (b) 2300 + 10 = 2310  (c) 1327 + 400 = 1727  (d) illegal reference; traps to operating system |
| **Q3. Compare the main memory organization schemes of contagious memory allocation, paging and segmentation with respect to the following issues.**   1. **Internal fragmentation** 2. **External fragmentation** 3. **Ability to share code across processes** |
| External fragmentation **Contiguous Allocation with Fixed-Size Partitions:** does not suffer from external fragmentation. Contiguous Allocation with Variable-Size Partitions: suffers from external fragmentation.  **Pure Segmentation:** suffers from external fragmentation.  **Paging:** does not suffer from external fragmentation. Internal fragmentation **Contiguous Allocation with Fixed-Size Partitions:** suffers from internal fragmentation. Contiguous Allocation with Variable-Size Partitions: does not suffer from internal fragmentation.  **Pure Segmentation:** does not suffer from internal fragmentation.  **Paging:** suffers from internal fragmentation. Ability to share code across processes **Contiguous Allocation with Fixed-Size Partitions:** no support for code sharing across processes.  **Contiguous Allocation with Variable-Size Partitions:** no support for code sharing across processes  **Pure Segmentation:** supports code sharing across processes. However, we must be careful to make sure that processes do not mix code and data in the same segment.  **Pure Paging:** supports code sharing across processes. As in pure segmentation, we must make sure that processes do not mix code and data in the same page   |  |  |  |  | | --- | --- | --- | --- | |  | External fragmentation | Internal fragmentation | Ability to share code | | Contiguous Allocation with Fixed-Size Partitions | No | Yes | No | | Contiguous Allocation with Variable-Size Partitions | Yes | No | No | | Pure Segmentation | Yes | No | make sure that processes do not mix code and data in the same segment | | Paging | No | Yes | Same as pure segmentation | |
| **Q4. How can you say that paging itself is a form of dynamic relocation?** |
| * Paging is a form of dynamic relocation, where each virtual address is bound by the paging hardware to a physical address. * Think of the page table as a set of relocation registers, one for each frame. * Mapping is invisible to the process; the OS maintains the mapping and the hardware does the translation. * Protection is provided with the same mechanisms as used in dynamic relocation. |
| **Q5. How to manage the page table in memory? What about paging the page table?** |
| * Memory structures for paging can get huge using straight-forward methods * One simple solution is to divide the page table into smaller units   + Hierarchical Paging   + Hashed Page Tables   + Inverted Page Tables * Hierarchical Page Tables   + Break up the logical address space into multiple page tables   + A simple technique is a two-level page table   + We then page the page table   + Since the page table is paged, the page number is further divided into:     - Outer page table     - Inner page table |
| **Q6. What is the limitation of using Page-table base register (PTBR)? How it is solved?** |
| The problem with this approach is the time required to access a user memory location. If we want to access location i, we must first index into the page table, using the value in the PTBR offset by the page number for i. This task requires a memory access. It provides us with the frame number, which is combined with the page offset to produce the actual address. We can then access the desired place in memory. With this scheme, two memory accesses are needed to access a byte (one for the page-table entry, one for the byte). Thus, memory access is slowed by a factor of 2. |
| **Q7. What is the difference between a physical address and a virtual address? In the following figure page/frame size is 4K (1KB = 1024Bytes). Page table shows corresponding frame numbers. Calculate the following**   1. **The physical address, if the virtual address is 0** 2. **The physical address, if the virtual address is 4000** 3. **The physical address, if the virtual address is (5, 20)** 4. **The virtual address if the physical address is 24576** 5. **The virtual address if the physical address is 12308**  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **Virtual**  **Address**  **Space**   |  |  | | --- | --- | | **60K – 64K** | **X** | | **56K – 60K** | **X** | | **52K – 56K** | **X** | | **48K – 52K** | **X** | | **44K – 48K** | **7** | | **40K – 44K** | **X** | | **36K – 40K** | **5** | | **32K – 36K** | **X** | | **28K – 32K** | **X** | | **24K – 28K** | **X** | | **20K – 24K** | **3** | | **16K – 20K** | **4** | | **12K – 16K** | **0** | | **8K – 12K** | **6** | | **4k – 8K** | **1** | | **0K – 4K** | **2** |   **} Virtual page** | **Physical**  **memory**  **address**   |  |  | | --- | --- | |  | **28K-32K** | |  | **24K-28K** | |  | **20K-24K** | |  | **16K-20K** | |  | **12K-16K** | |  | **8K-12K** | |  | **4K-8K** | |  | **0K-4K** |   **} Page frame** | |
| * **Logical address or Virtual address** – seen by user program; generated by the CPU * **Physical address** –address seen by the memory unit  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **Virtual**  **Address**  **Space**   |  |  |  | | --- | --- | --- | |  | **Frame** | Page | | **60K – 64K** | **X** | 15 | | **56K – 60K** | **X** | 14 | | **52K – 56K** | **X** | 13 | | **48K – 52K** | **X** | 12 | | **44K – 48K** | **7** | 11 | | **40K – 44K** | **X** | 10 | | **36K – 40K** | **5** | 9 | | **32K – 36K** | **X** | 8 | | **28K – 32K** | **X** | 7 | | **24K – 28K** | **X** | 6 | | **20K – 24K** | **3** | 5 | | **16K – 20K** | **4** | 4 | | **12K – 16K** | **0** | 3 | | **8K – 12K** | **6** | 2 | | **4k – 8K** | **1** | 1 | | **0K – 4K** | **2** | 0 | | **Physical**  **memory**  **address**   |  |  |  |  | | --- | --- | --- | --- | | **Page** | **Frame** |  | | | 11 | 7 |  | **28K-32K** | | 2 | 6 |  | **24K-28K** | | 9 | 5 |  | **20K-24K** | | 4 | 4 |  | **16K-20K** | | 5 | 3 |  | **12K-16K** | | 0 | 2 |  | **8K-12K** | | 1 | 1 |  | **4K-8K** | | 3 | 0 |  | **0K-4K** | |  1. **The physical address, if the virtual address is 0**   Frame no corresponding to Page no 0 = 2;  Physical address = Starting address of frame + offset of logical address  = 8K + 0 = 8K   1. **The physical address, if the virtual address is 4000**   Standard virtual address = 4000/1024 = 3.9K which falls in page 0  Frame no corresponding to Page no 0 = 2;  Physical address = Starting address of frame + offset of logical address  = 8K + 4000  = 8192 + 4000  = 12192   1. **The physical address, if the virtual address is (5, 20)**   Page no = 5  Page offset = 20  Frame no corresponding to Page no 5 = 3;  Physical address = Starting address of frame + offset of logical address  = 12K + 20  = 12288 + 20  = 12308   1. **The virtual address if the physical address is 24576**   Standard address = 24579/1024 = 24K  Frame no = 6  Page no corresponding to frame no = 2  Offset = 0  Virtual address = Starting address of page + offset  = 24K + 0  = 24K   1. **The virtual address if the physical address is 12308**   Standard physical address = 12308/1024 = 12.01K  Frame no = 3  Page no corresponding to frame no = 5  Offset = 12308 - 12K  = 12308 - (12x1024)  = 20  Virtual address = Starting address of page + offset  = 20K + 20  = 20480 + 20  = 20500 |
| **Q 8. Compare paging with segmentation with respect to the amount of memory required by the address translation structures in order to convert virtual address to physical address.** |
| Paging requires more memory overhead to maintain the translation structures. Segmentation requires just two registers per segment: one to maintain the base of the segment and the other to maintain the extent of the segment. Paging on the other hand requires one entry per page, and this entry provides the physical address in which the page is located |
| **Q9. Consider the following page table below. All numbers are decimal, everything is numbered starting from zero and all addresses are memory byte addresses. The page size is 1024 bytes and the processor architecture is 32-bit.**   |  |  |  |  | | --- | --- | --- | --- | | **Virtual Page #** | **Valid Bit** | **Dirty Bit** | **Page frame #** | | **0** | **1** | **0** | **4** | | **1** | **1** | **1** | **7** | | **2** | **0** | **0** | **-** | | **3** | **1** | **0** | **2** | | **4** | **0** | **0** | **-** | | **5** | **1** | **1** | **0** |   **Describe exactly how a virtual address generated by the CPU is translated into a physical address?**  **What physical address, if any, would each of the following virtual addresses correspond to? (Do not try to handle any page faults if any). Note that the addresses are decimal address, not hexadecimal addresses:**   1. **1** 2. **1052: 0000000000000000000001 | 0000011100** 3. **2221: 100010101101** 4. **5499: 1010101111011** 5. **32, 765**   **Hint: The system supports 32-bits addressing, where first 22 bits are used for page number and last 10 bits are used for off-set. 1052 = 0000000000000000000001 | 0000011100 (virtual page number 1, offset 28) in the table virtual page number 1 is valid (valid bit is 1), and the page frame is 7, so the physical address is 0000000000000000000111 | 0000011100 = 7052** |
| **a) Virtual Memory Address: 1**  *1 = 0000000000000000000000 | 0000000001 (virtual page number 0, offset 1), in the table virtual page number 0 is valid (valid bit 1), and the page frame is 4, so the physical address is 0000000000000000000100 | 0000000001 = 4097*  **b) Virtual Memory Address: 1052 (decimal)** *1052 = 0000000000000000000001 | 0000011100 (virtual page number 1, offset 28), in the table virtual page number 1 is valid (valid bit 1), and the page frame is 7, so the physical address is 0000000000000000000111 | 0000011100 = 7196*  **c) Virtual Memory Address: 2221 (decimal)**  *2221 -> 2221/1024 = 2 pages and 173 offset - this page is invalid and it is not in main memory so it can’t be translated into physical memory address*  **d) Virtual Memory Address: 5499 (decimal)**  *5499 -> 5499/1024 = 5 pages and 379 offset – this page maps to 0 page frame hence physical address is 379 bytes* |
| **Q 10. Differentiate between.**   1. **Logical and physical addresses** 2. **External fragmentation and Internal fragmentation** 3. **Best-fit and Worst-fit memory allocation strategies** 4. **Segmentation and Paging** |
| **a) Logical and physical addresses**   * **Logical address or Virtual address** – seen by user program; generated by the CPU * **Physical address** –address seen by the memory unit   **b) External fragmentation and Internal fragmentation**   * **External Fragmentation –** total memory space exists to satisfy a request, but it is not contiguous * **Internal Fragmentation –** allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used   **c) Best-fit and Worst-fit memory allocation strategies**   * **Best-fit:** Allocate the *smallest* hole that is big enough; must search entire list, unless ordered by size. Produces the smallest leftover hole * **Worst-fit:** Allocate the *largest* hole; must also search entire list. Produces the largest leftover hole   **d) Segmentation and Paging**   * **Segmentation:** Memory-management scheme that supports user view of memory. A program is a collection of segments. A segment is a logical unit such as: Subroutine, symbol table, stack, heap etc.   8   * **Paging**: Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available   + Avoids external fragmentation   + Avoids problem of varying sized memory chunks   + Set up a **page table** to translate logical to physical addresses   + Backing store likewise split into pages   + Still have Internal fragmentation   C:\Users\as668\Desktop\9_08.jpg |
| **Q11. Consider the following page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 How many page faults would occur with:**   1. **FIFO Page Replacement** 2. **LRU Page Replacement**   **Assuming four frames? Remember that all frames are initially empty, so your first unique pages will cost one fault each.** |
| 1. **FIFO Page Replacement**  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **7** | **7** | **7** | **7** | **7** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **2** | **2** | **2** | **2** | **2** | **2** | |  | **0** | **0** | **0** | **0** | **0** | **0** | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **7** | **7** | **7** | |  |  | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | |  |  |  | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | | **\*** | **\*** | **\*** | **\*** | **Hit** | **\*** | **Hit** | **\*** | **Hit** | **Hit** | **\*** | **Hit** | **Hit** | **\*** | **\*** | **Hit** | **Hit** | **\*** | **Hit** | **Hit** | | **7** | **0** | **1** | **2** | **0** | **3** | **0** | **4** | **2** | **3** | **0** | **3** | **2** | **1** | **2** | **0** | **1** | **7** | **0** | **1** |   10 page-faults occurred  **b) LRU Page Replacement**   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **7** | **7** | **7** | **7** | **7** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **7** | **7** | **7** | |  | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | |  |  | **1** | **1** | **1** | **1** | **1** | **4** | **4** | **4** | **4** | **4** | **4** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | |  |  |  | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | | **\*** | **\*** | **\*** | **\*** | **Hit** | **\*** | **Hit** | **\*** | **Hit** | **Hit** | **Hit** | **Hit** | **Hit** | **\*** | **Hit** | **Hit** | **Hit** | **\*** | **Hit** | **Hit** | | **7** | **0** | **1** | **2** | **0** | **3** | **0** | **4** | **2** | **3** | **0** | **3** | **2** | **1** | **2** | **0** | **1** | **7** | **0** | **1** |   8 page-faults occurred |
| **Q 12. Consider the following page reference string: 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6. How many page faults would occur for Optimal Page Replacement with**   1. **Four frames** 2. **Six frames**   **Remember that all frames are initially empty, so your first unique pages will cost one fault each.** |
| **a) Four frames**   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **7** | **7** | **7** | **7** | **1** | **1** | **1** | **1** | |  | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | |  |  | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **6** | |  |  |  | **4** | **4** | **4** | **5** | **6** | **6** | **6** | **6** | **6** | **6** | **6** | **6** | **6** | **6** | **6** | **6** | **6** | | **\*** | **\*** | **\*** | **\*** | **Hit** | **Hit** | **\*** | **\*** | **Hit** | **Hit** | **Hit** | **Hit** | **\*** | **Hit** | **Hit** | **Hit** | **\*** | **hit** | **hit** | **Hit** | | **1** | **2** | **3** | **4** | **2** | **1** | **5** | **6** | **2** | **1** | **2** | **3** | **7** | **6** | **3** | **2** | **1** | **2** | **3** | **6** |   8 page-faults occurred  **b) Six frames**   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | |  | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | |  |  | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | |  |  |  | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **7** | **7** | **7** | **7** | **7** | **7** | **7** | **7** | |  |  |  |  |  |  | **5** | **5** | **5** | **5** | **5** | **5** | **5** | **5** | **5** | **5** | **5** | **5** | **5** | **5** | |  |  |  |  |  |  |  | **6** | **6** | **6** | **6** | **6** | **6** | **6** | **6** | **6** | **6** | **6** | **6** | **6** | | **\*** | **\*** | **\*** | **\*** | **hit** | **hit** | **\*** | **\*** | **hit** | **hit** | **hit** | **Hit** | **\*** | **Hit** | **hit** | **hit** | **hit** | **hit** | **hit** | **Hit** | | **1** | **2** | **3** | **4** | **2** | **1** | **5** | **6** | **2** | **1** | **2** | **3** | **7** | **6** | **3** | **2** | **1** | **2** | **3** | **6** |   7 page-faults occurred |
| **Q 13. Enlist the design issues for paging systems and also for segmentation systems?**   1. Local v. Global    1. Global: The replacement algorithm considers all pages in memory.    2. Local: Replacement only considers pages owned by the faulting process.    3. With global, processes which need more pages naturally obtain them. Local requires a policy to modify the number of pages a process has.    4. Measuring the Page Fault Frequency (PFF) for each process can reveal which is suffering from its allocation size. 2. Load Control    1. Measure the overall PFF to see if the system is overloaded.    2. May want to swap a process out to reduce load on the VM system.    3. The process with the largest PFF is the obvious choice, but need to consider priorities, and keep enough processes running to keep the CPU busy. 3. Page Size    1. Basic page size is determined by hardware.    2. An OS can allocate pages in (small) groups, simulating larger pages for some purposes. 4. Starting a program    1. When a program is run, it isn't loaded into memory.    2. A page table is created, with all entries invalid.    3. Execution begins and needed pages are brought in on demand.    4. Only the parts of the program actually run are ever loaded into memory. 5. Shared pages    1. Each process has its own page table. The same page may be entered into more than one PT to allow sharing.    2. Creates extra record-keeping so shared pages aren't immediately removed when one process exits. |
| **Q 14. Consider a system with 80% hit ratio, 50 nano-seconds time to search the associative registers, 750 nano-seconds time to access memory. Find the time to access a page.**   1. **When the page number is in associative memory** 2. **When the time to access a page when not in associative memory** 3. **Find the effective memory access time** |
| a) The time required is  50 nano seconds to get the page number from associative memory  and 750 nano-seconds to read the desired word from memory.  Time = 50+750= 800 nano seconds.  b) Now the time when not in associative memory is  Time = 50+750+750= 1550 nano seconds  One memory access extra is required to read the page table from  memory.  c) Effective access time = Page number in associative memory +  Page number not in associated memory.  Page number in associative memory = 0.8 \* 800.  Page number not in associated memory = 0.2 \* 1550.  Effective access time = 0.8 \* 800 + 0.2 \* 1550 = 950 nano seconds |
| **Q 15. The Intel Pentium uses segmentation with paging for memory management. How the virtual address is translated into physical address in this system?**  Memory management in IA-32 systems is divided into two components— segmentation and paging—and works as follows:   * The CPU generates logical addresses, which are given to the segmentation unit. The segmentation unit produces a linear address for each logical address * The linear address is then given to the paging unit, which in turn generates the physical address in main memory. * Thus, the segmentation and paging units form the equivalent of the memory-management unit (MMU). |
| **Q 16. What advantages could be obtained by combining segmentation with paging?**   * Segmentation is visible but paging is transparent to the programmer * Reduces memory usage as opposed to pure paging   + - Page table size limited by segment size     - Segment table has only one entry per actual segment * Share individual pages by copying page table entries. * Share whole segments by sharing segment table entries, which is the same as sharing the page table for that segment. * Most advantages of paging still hold   + - Simplifies memory allocation     - Eliminates external fragmentation. * In general this system combines the efficiency in paging with the protection and sharing capabilities of the segmentation. |
| **Q 17. In LRU page replacement algorithm, how the time is measured to determine an order for the frame by the time last use?**   * Time-counter implementation   + - Every page entry has a time-counter variable; every time a page is referenced through this entry, copy the value of the clock into the time-counter     - When a page needs to be changed, look at the time-counters to find smallest value       * Search through a table is needed * Stack implementation   + - Keep a stack of page numbers in a double link form:     - Page referenced:     - Move it to the top     - Requires 6 pointers to be changed     - But each update more expensive     - No search for replacement |
| **Q 18. Compare Optimal and LRU page replacement algorithms for the following reference strings: 1,4,1,6,1,6,1,6,1,6,1** |
|  |
| **Optimal page replacement**   |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | |  | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **4** | |  |  |  | **6** | **6** | **6** | **6** | **6** | **6** | **6** | **6** | | **\*** | **\*** | **Hit** | **\*** | **hit** | **hit** | **hit** | **hit** | **hit** | **hit** | **Hit** | | **1** | **4** | **1** | **6** | **1** | **6** | **1** | **6** | **1** | **6** | **1** |   Page faults occurred = 3  **LRU**   |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | |  | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **4** | |  |  |  | **6** | **6** | **6** | **6** | **6** | **6** | **6** | **6** | | **\*** | **\*** | **Hit** | **\*** | **hit** | **hit** | **hit** | **hit** | **hit** | **hit** | **Hit** | | **1** | **4** | **1** | **6** | **1** | **6** | **1** | **6** | **1** | **6** | **1** |   Page faults = 3 |
| **Q 19. Consider a user program of logical address of size 6 pages and page size is 4 bytes. The physical address contains 300 frames. The user program consists of 22 instructions a, b, c, . . . u, v . Each instruction takes 1 byte. Assume at that time the free frames are 7, 26, 52, 20, 55, 6, 18, 21, 70, and 90. Find the following**   1. **Draw the logical and physical maps and page tables?** 2. **Allocate each page in the corresponding frame?** 3. **Find the physical addresses for the instructions m, d, v, r?** 4. **Calculate the fragmentation if exist?** |
| **a) Draw the logical and physical maps and page tables?**    **b) Allocate each page in the corresponding frame?**  In the diagram of **a)**  **c) Find the physical addresses for the instructions m, d, v, r?**  The physical address = page size \* frame number + offset  The physical address of m = 4\*20 +0 = 80  The physical address of d = 4\*7+3 = 31  The physical address of v = 4\* 6 +1 = 25  The physical address of r = 4\*55+ 1= 221  **d)** **Calculate the fragmentation if exist?**  The external fragmentation = 0  The internal fragmentation = 2 |
| **Q 20. How the page faults are handled in Windows XP?**  Copy-on-write is a common technique used by several operating systems, including Windows XP, Linux, and Solaris   * Copy-on-Write (COW) allows both parent and child processes to initially share the same pages in memory   + If either process modifies a shared page, only then is the page copied * COW allows more efficient process creation as only modified pages are copied * In general, free pages are allocated from a pool of zero-fill-on-demand pages   + Pool should always have free frames for fast demand page execution     - Don’t want to have to free a frame as well as other processing on page fault   + Why zero-out a page before allocating it? * vfork() variation on fork() system call has parent suspend and child using copy-on-write address space of parent   + Designed to have child call exec()   + Very efficient |
| **Q 21. Commonly available computer systems have RAM around 2-4 GB. Yet, when we look at addresses used by a program running on these systems, the programs are using higher range of addresses. Why is it so?**  This is due to implementation of virtual memory. Virtual memory allow processes’ virtual address much larger than total physical address space of main memory.   * **Virtual memory** – separation of user logical memory from physical memory   + - Only part of the program needs to be in memory for execution     - Logical address space can therefore be much larger than physical address space     - Allows address spaces to be shared by several processes     - Allows for more efficient process creation     - More programs running concurrently     - Less I/O needed to load or swap processes |
| **Q 22. What is the difference between inverted and hashed page tables? In what type of process each of this page table is used?**  **Inverted Page Table**   * Rather than having each process keep a page table and track of all possible logical pages, track all physical pages * One entry for each real page of memory * Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page * Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs * Use hash table to limit the search to one (or at most a few) page-table entries   + TLB can accelerate access * But how to implement shared memory?   + One mapping of a virtual address to the shared physical address     **Hashed Page Table**   * Used in architecture with address spaces > 32 bits * The virtual page number is hashed into a page table   + This page table contains a chain of elements hashing to the same location * Each element contains   + The virtual page number   + The value of the mapped page frame   + A pointer to the next element * Virtual page numbers are compared in this chain searching for a match   + If a match is found, the corresponding physical frame is extracted |