Matthew Kaiser

Problem Set #4

4/14/17

1. (+12) Describe at least 3 general approaches in memory management than can help solve the external fragmentation problem.

* **Garbage Collection: Collects unused and inaccessible memory, returning them as free memory.**
* **Full Compaction: Moves all processes that are allocated memory to one side of the physical memory, releases the old allocated memory.**
* **Partial Compaction: Similar to full compaction, but for partial you only move as much allocated memory as necessary to honor the incoming request.**
* **Paging: Divides system memory into fixed size pages while allocating pages selectively to frames, manages pages in memory.**

1. (+12) A memory manager for a variable-sized region strategy has a free list of blocks of size 600, 400, 1000, 2200, 1600, and 1050 bytes. What block will be selected to honor a request for:

a. 1603 bytes using a best-fit policy? **2200**

b. 949 bytes using a best-fit policy? **1000**

c. 1603 bytes using a worst-fit policy? **2200**

d. 349 bytes using a worst-fit policy? **2200**

e. 1603 bytes using a first-fit policy? (assume the free list is ordered as listed above) **2200**

f. 1049 using a first-fit policy? **2200**

1. (+20) Suppose two processes need to be mapped into main memory using pages. Process P1 consists of 7 pages, and process P2 consists of 4 pages. Assume main memory consists of 16 frames, a logical page is the same size as a physical frame, and that 4 entries in a page tale fills up a frame of memory. Assume also that within the process' allocated address spaces, there are two pages of shared code 'X' and 'Y' common to both address spaces. Design a memory management system that can store these two processes and their page tables in RAM. Identify which frames you have chosen to assign to which process pages and page tables in main memory/RAM. Also show possible page tables for P1 and P2 (e.g. page table for P1 should have 7 entries).

**Page Table P1:**

|  |  |
| --- | --- |
| **Page Number** | **Frames** |
| **0** | **1** |
| **1** | **7** |
| **2** | **13** |
| **3** | **10** |
| **4** | **3** |
| **5** | **4** |
| **6** | **8** |

**Page Table P2:**

|  |  |
| --- | --- |
| **Page Number** | **Frames** |
| **0** | **1** |
| **1** | **7** |
| **2** | **15** |
| **3** | **2** |

**Frames in Memory:**

|  |  |  |  |
| --- | --- | --- | --- |
| **F0** |  | **F8** | **P1 @ #6** |
| **F1** | **P1&P2 @ #0** | **F9** |  |
| **F2** | **P2 @ #3** | **F10** | **P1 @ #3** |
| **F3** | **P1 @ #4** | **F11** |  |
| **F4** | **P1 @ #5** | **F12** |  |
| **F5** |  | **F13** | **P1 @ #2** |
| **F6** |  | **F14** |  |
| **F7** | **P1&P2 @ #1** | **F15** | **P2 @ #3** |

* **It takes four entries to fill out a frame, however contiguous paging is not required. In my example all of the page numbers have unique frames except for pages 0&1 which similarly reside in F1&F7 respectively.**

1. (+12) Suppose on-demand paging is employed in addition to TLB caching. The time for a TLB hit is T = 1 ns, a memory read M = 10 ns, and a disk read D = 10 ms. Let p\_TLB = the probability of a TLB hit, and p = the probability of a page fault given a TLB miss. What is a general formula for the average memory access time expressed as a function of T, M, D, p, and p\_TLB? Once parameter values are substituted, and assuming p = .001 and p TLB = 90%, what is the calculated average memory access time?

**Formulas:**

|  |  |  |
| --- | --- | --- |
| **TLB hit** | **TLB hit & page hit** | **TLB miss & page fault** |
| **T \* p\_TLB** | **(1-p\_TLB)(1-p)\*M** | **(1-p\_TLB)(D)(p)** |

**Values:**

* **T = 1 ns**
* **p\_TLB = 0.9**
* **p = 0.001**
* **M = 10 ns**
* **D = 10^7 ns**

**Plug it in:**

* **average memory access time = TLB hit + TLB miss & page hit + TLB miss & page fault**
* **average memory access time = (1 \* 0.9) + (0.1 \* 0.999 \* 10) + (0.1 \* 0.001 \* 10^7)**
* **average memory access time = 1001.899 ns**

1. (+12) The Least Recently Used (LRU) page replacement policy does not suffer from Belady's Anomaly. Explain intuitively why this is the case. Construct an example page fault sequence to illustrate your point.

**The LRU page replacement policy does not suffer from Belady’s anomaly due to its FIFO properties. If a page has been called frequently it’s more likely to get called again before a new page. Due to this frequency is easier to calculate than the age of page loading.**

**Example: When calculating a page fault of three with LRU using a frame size of 3, the correct page fault result is six.**

1. (+20) Given a frame allocation of 3, and the following sequence of page references 3 2 4 3 4 2 2 3 4 5 6 7 7 6 5 4 5 6 7 2 1, and assuming main memory is initially unloaded, show the page faulting behavior using the following page replacement policies. How many page faults are generated by each page replacement algorithm? Which generates the fewest page faults?

a. FIFO

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

Page fault = 124

b. OPT

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 3 2 4 | 3 4 2 | 2 3 4 | 5 6 7 | 7 6 5 | 4 5 6 | 7 2 1 |
| **3** 3 3 | 33 3 | 33 3 | **5** 5 5 | **5** 5 5 | **5** 5 5 | **7** 7 7 |
| **2** 2 2 | 2 2 2 | 2 2 2 | **6** 6 6 | 6 6 6 | **6** 6 6 | **2** 2 |
| **4** 4 4 | 4 4 4 | 4 4 4 | **7** 7 7 | 7 **4** 4 | 4 4 4 | **1** |

**Page fault = 10------Most Effective!!-----fewest page faults**

c. LRU

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

Page fault = 12

**OPT was most effective**

1. (+12) Assume the same sequence of page references as in problem #6, and assume memory is initially unloaded, but now assume that a dynamic paging working-set algorithm is applied to the same sequence of page references, with a window size of 6. Draw the page faulting behavior. Your solution chart should show the frame allocation at any given time to the process.

* Work set = change(6)
* W(t1, 2) = {2,3,4}; W(t3, 2) = {4, 5, 6, 7}
* |----------------| |---------------- |
* 3 2 4 3 4 2 2 3 4 5 6 7 7 6 5 4 5 6 7 2 1
* |-----------------| |----------------|