Bronson Edralin

EE468 Operating Systems

HW #7

11/5/14

**Problem 8.11 (1 pt)**

Given six memory partitions of 300 KB, 600 KB, 350 KB, 200 KB, 750 KB, and 125 KB (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of size 115 KB, 500 KB, 358 KB, 200 KB, and 375 KB (in order)? Rank the algorithms in terms of how efficiently they use memory.

*First fit:*

115 KB -> 300KB

500 KB -> 600KB

358 KB -> 750KB

200 KB -> 350KB

375 KB -> 750KB

*Best fit:*

115 KB -> 125KB

500 KB -> 600KB

358 KB -> 750KB

200 KB -> 200KB

375 KB -> 750KB

*Worst fit:*

115 KB -> 750KB

500 KB -> 750KB

358 KB -> 600KB

200 KB -> 350KB

375 KB -> Internal fragmentation waits for 600KB or 750KB memory to get free.

Efficiency

Best fit > First fit > Worst fit

**Problem 8.20 do only pars (a, c, e) (1 pt). Hint: 1KB = 1024 Bytes**

Assuming a 1-KB page size, what are the page numbers and offsets for the following address references (provided as decimal numbers):

a) 3085

c) 215201

e) 2000001

1. 3085:

c) 215201:

e) 2000001:

**Problem 8.23 (1 pt)**

Consider a logical address space of 256 pages with a 4-KB page size,  mapped onto a physical memory of 64 frames.

1. How many bits are required in the logical address?
2. How many bits are required in the physical address?

256 is maximum pages for each process

Therefore,

Logical address in paging consists of: Page number + Offset

64 frames…

**Problem 8.25 (1 pt)**

Consider a paging system with the page table stored in memory.

1. If a memory reference takes 50 nanoseconds, how long does a  paged memory reference take?
2. If we add TLBs, and 75 percent of all page-table references are found in the TLBs, what is the effective memory reference time? (Assume that finding a page-table entry in the TLBs takes 2 nanoseconds, if the entry is present.)
3. It will take **100 nanoseconds**, because it will take 50ns to access page table and 50ns to access word in memory.
4. With a TLB the effective access time = 0.75\*(TLBhit) + 0.25\*(TLBmiss)=0.75\*(50ns)+0.25\*(100ns)=**62.5ns**

**Problem 8.28 do only parts (a, c, e) (1 pt)**

Consider the following segment 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?

a. 0,430

c. 2,500

e. 4,112

a) Segment 0, 430+219=**649**

c) Segment 2, 500+90=590 but Length is 100 so this can’t happen. Reference to physical address of logical address 2,500 will result in illegal reference and a trap to operating system will be generated.

e) Segment 4, 112+1952=2064 but Length is 96 so this is illegal and a trap will be generated for this situation as well.

**Problem A (1 pt) Consider a computer system with a 32-bit logical address and 4KB page size. The system supports up to 512MB of physical memory.**

**(a) How many bytes are there in the page table in a single-level page table system?**

**(b) Consider a hierarchical page table system with two levels. How many bytes are there in the page tables, i.e., the sum of both outer and inner page tables?**

1. bytes in the page table in a single-level page table system:

32-bit logical address

4KB page size

* 4KB=4096 bytes=
* 12 bits for offset
* 32-12
* 20
* page table

bytes per page

512MB physical memory = bytes

* 17 bits needed for physical page number

Page table entry = ~4 bytes

* 17 bit physical page number = ~3 bytes
* Access control bits = ~1 byte

Page table size

* ***are basically 4.19MB***

1. Hierarchical page table system with 2 levels. Bytes there in page tables:

approximately 10 bits of address space needed.

* **4 bytes per page table entry**

**Problem 9.14 (1 pt)**

Assume that a program has just referenced an address in virtual memory. Describe a scenario in which each of the following can occur. (If no such scenario can occur, explain why.)

* TLB miss with no page fault
* TLB miss and page fault
* TLB hit and no page fault
* TLB hit and page fault
* It is **possible** for a TLB miss with no page fault. TLB did not contain this page table entry but it did exist in page table and entry was valid
* It is **possible** for a TLB miss and page fault. TLB did not contain this page table entry so it asked the page table but the entry was either invalid or out on disk
* It is **possible** for a TLB hit and no page fault, because TLB contained that entry and it was valid and in memory
* It is **impossible** for a TLB hit and page fault because it cannot have a TLB entry if page is not in memory. Operating system updates page tables and TLB when a page in memory has been flushed out to disk.

**Problem 9.15 all three parts (1 pt)**

A simplified view of thread states is Ready, Running, and Blocked, where a thread is either ready and waiting to be scheduled, is running on the processor, or is blocked (for example, waiting for I/O). This is illustrated in Figure 9.31. Assuming a thread is in the Running state, answer the following questions, and explain your answer:

1. Will the thread change state if it incurs a page fault? If so, to what state will it change?
2. Will the thread change state if it generates a TLB miss that is resolved in the page table? If so, to what state will it change?
3. Will the thread change state if an address reference is resolved in the page table? If so, to what state will it change?
4. Yes, a thread changes its state from Running state to Blocked state when a page fault occurs because when a page fault occurs, the process starts to wait for an I/O operation to finish. OS does several things while the process is waiting. It checks whether the page is really invalid or just on disk, finds a free memory frame, schedules a disk access to load the page into the frame, updates the page table with the new logical physical mapping, updates the valid bit of that entry, and eventually restarts the process by change its state from Blocked to Ready.
5. Not really, because if a page table entry is not found in the TLB or a TLB miss occurs, the page number is used to index and process the page table. If the page is already in memory, then TLB is updated to include new page entry, while process execution continues since there is no I/O operation needed. If page is not in main memory, a page fault is generated. In this case, process needs to change to Blocked state and wait for I/O to access the disk.
6. No, because no I/O operation is needed if address reference is resolved in page table, which indicates the page required is loaded in main memory already.

**Problem 9.20 (1 pt)**

When a page fault occurs, the process requesting the page must block while waiting for the page to be brought from disk into physical memory. Assume that there exists a process with five user-level threads and that the mapping of user threads to kernel threads is one to one. If one user thread incurs a page fault while accessing its stack, would the other user threads belonging to the same process also be affected by the page fault—that is, would they also have to wait for the faulting page to be brought into memory? Explain.

Yes, if a user’s user thread incurs a page fault while accessing its stack, this would result in blocking the other user’s user thread belonging to same process. This is because in many-to-one thread mapping entire process will block if a thread gets blocked and only one thread can access the kernel at a time in many-to-one model. Multiple threads are not able to run in parallel.

**Problem 9.21 (1 pt)**

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
* FIFO replacement
* Optimal replacement
* This will result in **18 page faults** for LRU replacement. LRU means Least Recently Used so whenever all three frames are full and we want to access new page, we replace the least recently used page with required page. By doing this for the given page reference string, hits occurred at 2 positions.

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

* This will result in **17 page faults** for FIFO. FIFO means First In First Out. Hits occurred at 3 positions.

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

* This will result in **13 page faults** for Optimal replacement. According to this algorithm, whenever all three frames are full and we want to access a new page, we replace the page which we are going to access last with the required page. Hits occurred at 7 positions.

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

**Problem 9.22 all parts (1 pt)**

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.

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
   * 0x3A9D
   * 0xA9D9
   * 0x7001
   * 0xACA1
2. Using the above addresses as a guide, provide an example of a logical address (in hexadecimal) that results in a page fault.
3. From what set of page frames will the LRU page-replacement algorithm choose in resolving a page fault?

|  |  |  |
| --- | --- | --- |
| *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 virtual addresses (in hex) to equivalent physical addresses.
   1. 0xE12C: Page 14, Therefore Page Frame = 3

**0x312C is physical address**

* 1. 0x3A9D: Page 3, Therefore Page Frame = 10

**0xAA9D is physical address**

* 1. 0xA9D9: Page 10, Therefore Page Frame = 5

**0x59D9 is physical address**

* 1. 0x7001: Page 7, Therefore Page Frame = 15

**0xF001 is physical address**

* 1. 0xACA1: Page 10, Therefore Page Frame = 5

**0x5CA1 is physical address**

1. An example of a virtual address that results in a page fault would be 0x49D9, 0x812C, 0xC001, or 0xD9D9. We see from the table that Page 4, 8, 12, and 13 are not available so any 4 virtual addresses that started with those numbers in most significant byte will lead to a page fault.
2. Pages 3, 7, 10, and 14 with Page Frames 10, 15, 5, and 3 would be chosen to resolve the page fault.

**Problem 9.26 (1 pt)**

The VAX/VMS system uses a FIFO replacement algorithm for resident pages and a free-frame pool of recently used pages. Assume that the free-frame pool is managed using the LRU replacement policy. Answer the following questions:

1. If a page fault occurs and the page does not exist in the free-frame pool, how is free space generated for the newly requested page?
2. If a page fault occurs and the page exists in the free-frame pool, how is the resident page set and the free-frame pool managed to make space for the requested page?
3. What does the system degenerate to if the number of resident pages is set to one?
4. What does the system degenerate to if the number of pages in the free-frame pool is zero?
5. If this occurs, one of the pages in free frame pool will need to be evicted to the disk, which creates space for one of the resident pages to be moved to the free frame pool. Then, accessed page is moved to the resident set.
6. If page exists in free-frame pool, it will be moved into the set of resident pages. One of the resident pages will be moved to the free-frame pool.
7. System degenerates into the page replacement algorithm used in free frame pool. Mostly into a LRU managed system.
8. System degenerates into a FIFO managed system.

**Problem 9.32 (1 pt)**

What is the cause of thrashing? How does the system detect thrashing? Once it detects thrashing, what can the system do to eliminate this problem?

Paging is when faults occur again and again, replacing pages that it must bring back in right away. A process is called thrashing if it is spend more time paging than executing.

A new process tries to get started by taking frames from running processes, causing more page faults and paging occurs more. This results in CPU utilization dropping even more and the CPU scheduler will try to increase degree of multiprogramming even more. Thrashing occurs and system throughput plunges with the page fault rate increasing by a lot. Effective memory-access time increases and no execution is being done.

Thrashing can be eliminated by reducing the level of multiprogramming. To prevent thrashing we can provide a process with as many frames as it needs. This approach is called locality model of process execution.