CISC 324, Winter 2018, Assignment 5

This assignment is due Monday March 19, in lecture

Readings

Course reader — Pages 55-63. Details about address translation when using segmentation with paging, including TLB use (pp 56-57). Multi-level paging (pp 58-61). Coding considerations related to page faults (pp 61-63).

Textbook — Here is a summary of the textbook readings in chapters 8 and 9. Some of these readings were already covered in assignment 4.

8.1.1 and 8.1.3 Introduction to registers, main memory, cache. Logical versus physical address. Base and limit registers (for simple memory management using partitioning; this pre-dates paging)

8.1.2 and 8.1.4-8.1.5 Compile time, load time, execution time. Dynamic loading and dynamic linking.

8.2 Swapping. (Skim this. Swapping is not used in modern operating systems, as noted in the last paragraph of 8.2.1)

8.3 More details about base and limit registers (memory management using partitioning)

8.4 Segmentation

8.5 Paging. Intro 8.5.1. 8.5.2 hardware support and TLB. 8.5.3 protection. 8.5.4 shared pages.

8.6.1 Hierarchical paging: multilevel page table. (skip 8.6.2 to 8.6.4: hashed page table and inverted page table are advanced methods of dealing with large page tables)

8.7 Intel 32 and 64 bit architectures. (Skip 8.8 Arm architecture)

8.9 Summary

9.1 Advantages of virtual memory.

9.2 Demand paging; handling page faults. 9.2.2 effective memory access time (Skip 9.3)

9.4 Page replacement. Intro 9.4.1. FIFO, Optimal, LRU 9.4.2 to 9.4.4. (Skip 9.4.5 to 9.4.8)

9.5 Frame allocation. Global versus local allocation. (skip 9.5.4)

9.6 Thrashing. Working set model. Page-fault frequency algorithm.

(Skip 9.7, 9.8, 9.10. Optionally, look at Windows and Solaris examples in 9.10.1 and 9.10.2)

In section 9.9: only read 9.9.2 (page size) and 9.9.5 (program structure; array traversal with row major order)

Questions

1) This question gives you practice in computing the effective memory access time for virtual memory.

Find the maximum allowable page-fault rate so that effective virtual memory access time is ≤ 200 nanoseconds on a computer system with the following characteristics.

The time to access main memory is 100 nanoseconds; this is sometimes referred to as the *raw memory access time*. Paging is used, with the page table stored in main memory. The address translation cache (TLB, Translation Lookaside Buffer) has a 95% hit rate. TLB access time is negligible compared to the 100 nanosecond memory access time, so:

- When address translation encounters a TLB hit, the virtual memory access time is 100 nanoseconds. (Virtual memory access time is the same as raw memory access time, because we assume that address translation time is negligible.)
- When there is a TLB miss with no page fault, the virtual memory access time is 200 nanoseconds. (Address translation must access memory to read an entry in the page table, thus adding 100 nanoseconds of overhead.)

The time to handle a page fault is 8 milliseconds when a clean page is replaced, and 16 milliseconds when a dirty page is replaced. The replaced page is dirty 65% of the time; this means that 35% of the time the replaced page is clean. (A *dirty* page is one that has been written to since it was loaded into main memory. Replacing a dirty page takes longer because the operating system has to write the contents of the dirty page back to disk before it can reuse that page frame.)

Extensive discussion to help you with question 1

The *page-fault rate* is the fraction of virtual memory accesses that result in a page fault. For example if 10,000 accesses to virtual memory result in 40 page faults, then the page fault rate is 40/10000 or .4%. I use R for the page-fault rate, since P is already used for *probability*.

To find a bound on R, write an equation for the effective memory access time as a weighted sum of slow and fast memory access times. Two examples of this type of weighted sum are given in the textbook.

(a) Near the end of section 8.5.2 Fast access 80% of the time: 120 nanoseconds because TLB hit adds 20 nanoseconds to the raw memory access time of 100 nanoseconds. Slow access 20% of the time: 220 nanoseconds because address translation overhead is 20ns for TLB miss plus 100ns to read the page table. The weighted sum shows that the effective memory access time is 140 nanoseconds, a 40% slowdown compared to raw memory access. If the TLB hit rate goes up to 98%, the effective access time drops to 122 nanoseconds, so only a 22% slowdown.

(b) Near the end of section 9.2.2 The fast access time is 200 nanoseconds (no page fault) and the slow access time is 8 milliseconds (with page fault). For performance degradation to be less than 10%, the page fault rate must be less than 0.0000025.

For this homework problem your calculation is a weighted sum that combines four types of memory access: (1) TLB hit, (2) TLB miss and no page fault, (3) TLB miss and a clean page fault, and (4) TLB miss and a dirty page fault. You need to figure out the probabilities A, B, C, D in this calculation:

A*TLB_hit_time + B*TLB_miss_time + C*CleanPageFaultTime + D*DirtyPageFaultTime

You are told A=.95, and you know that the sum A+B+C+D must total to one. Your goal is to find a bound on the page fault rate, R. To calculate this, figure out how to write B, C, and D in terms of R. For example, B = .05-R.

To make sure you understand the problem, begin by analyzing the two extreme cases:

- Largest possible value for R=0.05. In this case, there is a TLB hit 95% of time, 0% of time there is a TLB miss with no page fault, and 5% of the time there is a TLB miss with a page fault.
- Smallest possible value for R=0. In this case, there is a TLB hit 95% of time, 5% of time there is a TLB miss with no page fault, and 0% of time there is a TLB miss with a page fault.

After you have computed a value for R, check your work by making sure that R*8ms is less than 200ns. For sure the effective memory access time must be larger than R*8ms, because processing a page fault takes at least 8ms. This reasoning demonstrates that R must be less than .000025. This is a rough upper bound on R – the correct answer for R is quite a bit smaller than .000025.

Some students get confused about conditional probabilities. (Skip this paragraph if you don't worry about conditional probabilities.) Let A denote the event "TLB hit", B denote the event "TLB miss", and F denote the event "page fault". Events A and B partition the space: always exactly one of A or B occurs. Write this as $P(A \cup B)=1$ and $P(A \cap B)=\varnothing$. Because of this partitioning, the probability of any event X can be written as P(X)=P(X|A)P(A)+P(X|B)P(B). Thus, the probability of a page fault is P(F)=P(F|A)P(A)+P(F|B)P(B). We know that P(F|A)=0 because a page fault cannot occur when there is a TLB hit, so this simplifies to P(F)=P(F|B)P(B). If you decide to solve this problem by finding the value of P(F|B), then remember to multiply by P(B) to get the final answer.

2) A paging system uses 32 bit virtual addresses. Each page table entry occupies 4 bytes. (See "Paging with a Large Virtual Memory" on pages 59-61 of the course reader, and the discussion of hierarchical paging in textbook section 8.6.1.)

(a) What is the smallest page size that allows the page table to fit into one page?

Hint: Since the page size is always a power of two, the answer has the form "2^k bytes" and you have to find the value of k. For example, let's guess k=20 and see what happens. The lowest 20 bits of the virtual address are the page offset, leaving the top 12 bits as the page number. Virtual memory is divided into 2^{12} pages. The page table size is 2^{12} * 4 bytes = 2^{14} bytes. This is quite a bit smaller than the page size of 2^{20} bytes, so our guess k=20 was too big.

(b) A page size of 2^{10} bytes is chosen. This means that the page table has to be stored in multiple levels. Describe how the 32 bit virtual address is divided into fields: how many fields are there, and how many bits are in each field?

3) The following sequence of virtual memory references is generated when a program is executed. Each memory reference is a 12 bit number written as three hexadecimal digits.

019, 01A, 1E4, 170, 073, 30E, 185, 24B, 24C, 430, 458, 364

(a) What is the reference string, assuming a page size of 100_{16} bytes? The definition of reference strings is given at the end of textbook section 9.4.1.

(b) Find the page fault rate for the reference string in part (a): assume that 2 frames of main memory are available to the program and the FIFO page replacement algorithm is used. Note that the page fault rate is calculated as "number of page faults" divided by "number of virtual memory references used to form the reference string".

(c) Repeat (b) using the LRU page replacement algorithm.

(d) Repeat (b) using the optimal page replacement algorithm.

4) Assume you have a reference string for a process with *m* frames, and initially all *m* frames are empty. The reference string has length *p*, with *n* distinct page numbers occurring in it. Give an upper bound and a lower bound on the number of page faults. Your bounds should hold for any page-replacement algorithm. Briefly justify your answers.

5) This program swaps the elements in the first half of array *A* with the elements in the second half of the array:

```
var A: array[1..1000] of integer;  // A is an array indexed from 1 to 1000
    temp1, temp2, i: integer;
// Code to initialize A has been omitted.  Here is the loop to swap elements in A.
for (i := 1 to 500) do {   // i is an index for the first half of the array.
       temp1 := A[i];      // This code swaps A[i] and A[500+i].
       temp2 := A[500+i];
       A[i] := temp2;
       A[500+i] := temp1;
       }
```

Assume that 100 integers fit into a page, so array *A* occupies 10 pages. Initially all of *A* is on disk. How many page faults occur during program execution in cases (a) and (b)? Use Least Recently Used (LRU) page replacement. [In this problem we only consider page faults that result from accessing *A*: assume that there are no page faults from fetching instructions.]

(a) Five page frames of size 100 integers are allocated to array *A*

(b) One page frame of size 100 integers is allocated to array *A*

6) Course reader pages 63 compares the use of linear search, binary search and hash table when searching for a set of 30 items in a table of 10,000 items. Repeat this analysis for the situation where we are searching for 50 items instead of 30 items.

7) Briefly describe what *thrashing* is. How can the operating system detect when thrashing occurs? What should the operating system do when thrashing occurs? [Refer to textbook Section 9.6.1 for a discussion of thrashing.]

8) Textbook section 9.5.3 discusses local versus global page replacement. Local page replacement algorithms are presented in sections 9.4.2 to 9.4.4: FIFO, Optimal, LRU. Global page replacement algorithms are presented in sections 9.6.2 and 9.6.3: Working Set and Page-fault Frequency.
(a) What are the potential advantages of using a global – rather than a local – page replacement algorithm?
(b) Briefly summarize the Page-fault frequency algorithm.