CSE 120: Final Solution

September 7, 2007

Name: _____

Student ID:_____

Write your test number on *all* pages because the pages may be separated for grading.

No books are allowed. Two double-sided 3x5 cards of handwritten notes are allowed, as is a calculator

If you think something about the question is open to interpretation, ask!.

Leaving a question blank will give you 25% partial credit. An answer that we can't read, can't understand, or is true, but not directly relevant to the question will receive a 0.

Problem	Score
1	/8
2	/4
3	/8
4	/6
5	/8
6	/4
7	/4
8	/8
9	/8
10	/12
11	/12
12	/10
13	/10
14	/10
Extra Credit	/5
Extra Credit	/8
Total	/112

1. *8 pts.* In Nachos project 3, there is a method, `Processor.writeTLBEntry` that will write an entry to the TLB. Fill in the missing class name in the declaration of that method:

   ```
   public void writeTLBEntry(int number, _____________________________ entry);
   ```

   Solution: `TranslationEntry` 2 points

   In Nachos, what are the fields that are stored in a TLB entry?

   Solution:

   - vpn
   - ppn
   - valid
   - used
   - dirty
   - readOnly

   1 point each

2. *4 pts.* Which of the following best describes the difference between global and local page replacement algorithms?

   (a) If a process *p* page faults, a local page replacement algorithm will page out a page belonging to *p*, while a global replacement algorithm can page out any page.

   (b) If a process *p* page faults, a local page replacement algorithm will consider only the recent history of page references (local), while a global page replacement algorithm will consider the entire history of all page references (global).

   (c) Global page replacement algorithms can't exhibit *Belady's anomaly*, while local page replacement algorithms can.

   (d) Global and local page replacement algorithms both try to reduce the frequency of page faults, but global algorithms are easier to implement

   Solution: a) is correct

3. *8 pts.* What is saved and restored (and where is it saved from and restored to) on a context switch between two threads in the same process? Between two threads in a different process? Be explicit.

   Solution: In the same process, must save register, sp, and pc into tcb of old thread (2 points) must restore registers, sp, pc from tcb of new thread. (2 points)

   In a different process, must save and restore as above (2 points), but must also restore the Page Table Base Register from the pcb of new thread (2 points).

4. *6 pts.* The Bankers algorithm is said to keep the system in a *safe* state. Describe what a *safe* state is and explain how the Bankers algorithm keeps the system in a safe state.

   Keep your answer short.

   Solution: A safe state is one where there some ordering of resource requests exists (up to a process's predefined maximum) that the system can satisfy without deadlock.

5. *8 pts.* Assume that it requires reading D disk blocks to read the first block of a file. Give a formula for the average time it takes to read the first block of the file in terms of D and the following constants:

   Cy CPU cycle time (in nanoseconds)

   Sk Average disk seek time (in seconds)

   La Average rotational latency in seconds

   Ov CPU overhead to initialize a one-block disk read (in seconds)

   Bw Disk transfer time (in bytes/second)

   Bs Disk block size (in bytes)

   Fs Size of the file (in bytes)

   Fg Percentage of files on the disk that are fragmented (non-contiguous)

   If the formula is wrong, but with an explanation you may receive partial credit.

   Solution: There are D disk blocks, so if we figure the time to read one block, we can multiply that by D. To read a block, the following occur (in order):

   (a) Initiate the read (Ov)
   (b) Seek to appropriate track (Sk)
   (c) Wait for the correct sector to rotate under the disk head (La)
   (d) Transfer one block worth of data (Bs/Bw)

   The units of each of those is (correctly) in seconds. Final answer:

$$D(Ov + Sk + La + \frac{Bs}{Bw})$$

6. *4 pts.* A non-maskable interrupt (NMI):

   (a) Can be generated on some machines by pushing a halt button.
   (b) Causes the hardware to immediately jump to the interrupt service routine assigned to that NMI.
   (c) Can't be disabled by disabling interrupts.
   (d) All of the above
   (e) None of the above

   Solution: d) is correct. They're all examples of NMIs

7. *4 pts.* A working set is best described as:

   (a) The set of pages that that CPU will reference in the near future.

   (b) The set of pages that the CPU has referenced in the recent history

   (c) The set of pages that a process will reference in its near future.

   (d) The set of pages that a process has referenced in its recent history

   Solution: d) is correct. The working set is based on looking back in time for a given process

8. *8 pts.* Consider the parameter Δ used to define the working-set window in the working-set model. What is the effect of setting Δ to a small value on the page fault frequency and the number of active (non-suspended) processes currently executing in the system? What is the effect when Δ is set to a very high value?

   Solution: A small value of Δ means that there will be fewer pages in the working set. This will lead to a higher page-fault frequency. Since there are fewer pages in the working set of each process, more processes can be actively executing.

   The effect of a large Δ is the opposite: lower page-fault frequency and fewer processes actively executing.

   One way to look at it is to imagine the extremes: A Δ of one instruction would mean only the last-accessed page is in memory. An infinite Δ would mean that no page ever accessed by a process would be evicted.

9. *8 pts.* Using the reference string <1 2 1 3 4 2 1 5 3 5 2>, fill in the table below (representing three page frames) using the LRU page replacement policy. How many page faults occur?

   Solution:

1*	1	1	1	1	2*	2	2	3*	3	3
	2*	2	1	4*	4	4	5*	5	5	5
			3*	3	3	1*	1	1	1*	2*

   There are 10 total page faults.

10. *12 pts.* Assume we have a demand-paging system with the following characteristics

    - There is a single-level page table.
    - It takes 3 milliseconds to service a page fault if there is a free frame available or if the frame to be evicted is not modified. Fifty percent of page faults fall into this category.
    - It takes 7 milliseconds to service a page fault if the frame to be evicted is modified. Fifty percent of page faults fall into this category.
    - Main-memory access time is 80 nanoseconds.
    - The page-fault frequency is 1 fault/1,000,000 virtual memory accesses.
    - The TLB caches page table entries. If a page table entry is in the TLB, the associated frame number is returned with an overhead of 10 ns. The TLB-hit rate is 99.9%.
    - The TLB lookup is parallel (starting a lookup of the page table while looking in the TLB).

    What is the effective memory access time? Is it more or less than 100 nanoseconds?

    Solution: The effective memory access time is the weighted average.

    - The page table entry is in the TLB .999 of the time. The cost is 80+10 nanoseconds for a memory access in this case.
    - The frame is not in memory 10^{-6} of the time. The cost is 3 millisecond (3,000,000 nanoseconds) thirty percent of that time, and 7 milliseconds (7,000,000 nanoseconds) seventy percent of that time.
    - The PTE is not in TLB, but the frame is in memory the remainder of the time (.001 - 10^{-6}). The cost is two memory accesses, or 160 nanoseconds.

    Total weighed time:

$$.999 \times (80+10) + 10^{-6}(0.5 \times 3 \times 10^6 + 0.5 \times 7 \times 10^6) + (.001 - 10^{-6}) \times 160 =$$
$$.999 \times 90 + (1.5+3.5) + .16 - 1.6 \times 10^{-4} \approx$$
$$95.07 \text{n}\textit{anoseconds}$$

    This is, of course, slightly less than 100 nanoseconds.

11. *12 pts.* A disk services a request for cylinder 20, and then while it is servicing another request for cylinder 40 (with an empty queue), the requests for the following cylinders appear in its queue in this order: 15 65 83 37 22 53. Assume no new requests come in. In what order are the requests in the queue satisfied using the following disk scheduling algorithms?

    (a) Shortest Seek First

    Solution: 37, 22, 15, 53, 65, 83. Always to the closest of the remaining cylinders.

    (b) Look (Scan or elevator but doesn't move all the way to the outer or inner cylinder if not necessary)

    Solution: The head is moving up (since it went from 20 to 40), so will continue increasing: 53, 65, 83. At this point, it reverses, moving down: 37, 22, 15.

    (c) C-Look (no servicing requests as the disk head moves from higher cylinders to lower cylinders).

    Solution: The head is moving up (since it went from 20 to 40), so will continue increasing: 53, 65, 83. At this point, it reverses direction , moving down until it gets to the lowest-numbered request. Then it reverses direction, servicing: 15, 22, 37.

12. *10 pts.* In this problem, all addresses and offsets are given in *hexadecimal*. We have an architecture that uses both segmentation and paging. An 18-bit virtual address consists of:

2-bit segment number	4-bit page number	12-bit offset

The frame number is an 8-bit value.

Here are the relevant tables for a specific process:

Segment Table

Segment #	Page Table
0	Page Table A
1	Page Table B
...	(Remainder invalid)

Page Table A (length: 6 PTEs)

Page #	Frame number
0	AA
1	FA
2	12
3	66
4	*INVALID*
5	33

Page Table B (length: 5 PTEs)

Page #	Frame number
0	31
1	62
2	15
3	*INVALID*
4	13

Find the physical address corresponding to each of these virtual addresses (write "INVALID" if the virtual address is invalid):

13456 _____

00000 _____

23456 _____

06FED _____

12122 _____

Solution:

(a) `13456`

Solution: We can break this into 1 (section number) 3 (page number) and 456 (offset) The section number of 1 leads us to page table B. The page number of 3 leads us to INVALID. Thus the physical address is INVALID.

(b) `000000`

Solution: We can break this into 0 (section number) 0 (page number) and 0000 (offset) The section number of 0 leads us to page table A. The page number of 0 leads us to frame number AA. Thus the physical address is AA000.

(c) `23456`

Solution: We can break this into 2 (section number) 3 (page number) and 456 (offset) The section number of 2 is invalid. Thus the physical address is INVALID.

(d) `06FED`

Solution: We can break this into 0 (section number) 6 (page number) and FED (offset) The section number of 0 leads us to page table A. The page number of 6 is larger than the page table size. Thus the physical address is INVALID.

(e) `12122`

Solution: We can break this into 1 (section number) 2 (page number) and 122 (offset) The section number of 1 leads us to page table B. The page number of 2 leads us to frame number 15. Thus the physical address is 15122.

13. *10 pts.* Consider a file system with blocks of size 2048. Block numbers are represented as 32-bit values. Each file has an inode structure containing 16 direct pointers, one singly-indirect pointer, one double-indirect pointer, and one triply-indirect pointer.

    (a) How big a disk (in bytes) can this file system support? (Give your final answer in standard terms of Kibibytes, Mebibytes, ...)

    Solution: A 32-bit block number can reference 2^{32} different blocks. Each block stores 2^{11} bytes. We can support a disk with at most 2^{43}, or 8 Tebibytes.

    (b) How big a file(in bytes) can this file system support? (Give your final answer in standard terms of Kibibytes, Mebibytes, ...)

    Solution: The file can contain 16 blocks via the direct pointers. A block number takes 4 bytes to store. Thus, each singly, doubly, or triply-indirect block has enough room for 2048/4 or 512 block pointers.

    The singly-indirect pointer can reference 512 blocks, the doubly: 512^2, and the triply: 512^3. The maximum number of bytes in the file, then, is:

$$\begin{aligned} 2048(16+512+512^2+512^3) = \\ 2^{11}(16+2^9+2^{18}+2^{27}) = \\ 2^{15}+2^{20}+2^{29}+2^{38} \end{aligned}$$

    This is 256Gibibytes + 513 Mebibytes + 32 Kibibytes.

14. *8 pts.* We have two threads which loop, modifying a shared variable:

```
shared Integer total = 0;
```

Process A

```
for i = 1 to 3 do
  A1: register=total
  A2: register=register+1
  A3: total=register
```

Process B

```
for j = 1 to 3 do
    B1: register=total
    B2: register=register+1
    B3: total = register
```

Come up with a sequence of instruction interleavings such that the final value ends up at 2, rather than the desired 6.

Solution: A1, A2, B1, B2, B3, B1, B2, B3, A3, B1 (B's register now holds 1), A1, A2, A3, A1, A2, A3, B2, B3 (Total is now 2).

Sho

How, exactly, could you fix the problem?

Solution: Acquire and release a lock before A1 and after A3 and before B1 and after B3:

shared Lock lock

lock.acquire; A1, A2, A3, lock.release

lock.acquire: B1, B2, B3, lock.release

15. *5 pts. Extra credit.*

(a) What are three things you liked most about this class?

(b) What are three things you liked least about this class?

(c) What would you like to see changed?

16. *8 pts. Extra credit.* Suppose that we have a memory system with 32-bit virtual addresses, 4KB pages, and a 20-level page table.

    (a) If the page table is full (with 2^{20} pages), show that a 20-level page table consumes approximately twice the space of a single level page table. *Hint: try drawing it out and summing a series.*

    Solution: The top-level page table will have two entries , pointing to two level-two page tables, each with two entries, for a total of 4 entries at level 2. We'll double at each level, so there will be 2^3 level-three page table entries, all the way to 2^{20} level-20 page tables entries. Thus, the total number of entries will be $\sum_{i=1}^{20} 2^i = (2^{21} - 1)$ page table entries.

    In the single-level page table, there are 2^{20} total page table entries. Thus, there are approximately twice as many in the 20-level page table case.

    (b) Show that for sparse tables (ones in which not all entries are in use), a 20-level page table can consume less space that a single-level page table.

    Solution: The most extreme example would be to compare a single-level page table with two pages in it: the first addressable page, and the last addressable page. This would consume 2^{20} page table entries, of which all but two would be invalid

    In the 20-level page table case, we'd have 2 page entries at the top level, and then 4 at each succeeding level, for a total of 19*4 + 2 = 78 page table entries, many orders of magnitude fewer entries than the single-level case.