Self-Test (Fix Partition, Virtual Partition and Paging)

1. True or false, every process has a page table and it can directly modify the page table?

2. True or false, every thread has a page table?

3. True or false, page table cannot be swapped out into disk because it is needed for virtual address to physical address look up

4. True or false, for a 32-bit architecture, if a process has data in every page, the two level page table takes more space than a single level page table?

5. For a 32-bit architecture, if the page size is 8KB, each PTE (page table entry) needs 4 bytes, and each second-level page table needs to fit in one physical page, how many PTE does a second-level page table have? How many entries do you have in the first-level page table (aka master page table)?

6. Explain why process context switch is slower than thread context switch.

7. Describe one con and one pro of hardware handling TLB miss over OS handling TLB miss

8. Describe the worst case execution time for a memory instruction

9. Why does TLB work in providing fast translation for memory instruction?

Answers are in the next page.
1. False. Yes, every process has a page table, but the process itself cannot directly access the page table.

2. False. Multiple threads of the same process shares the same page table.

3. False. Page tables can be swapped out to disks.

4. True. If every page has some data, you need to have an PTE for every single virtual page. All second-level page tables add together take the same amount space as one-level page table. But the two-level page table has an additional table—-the master page table. As a result, the two level page table actually takes more space.

5. A second-level page table has $8K/4 = 2048$ entries (since it needs to fit into one physical page). So you need 11 bits to index the second-level page table. The page offset is 13 bits since the page size is 8KB. So remaining bits for indexing the master page table is 32-11-13 = 32-24 = 8 bits. So the master page table has $2^8 = 256$ entries.

6. When the OS switches from process A to process B, it needs to invalidate all TLB entries so that process B will not wrongly use the virtual page to physical page translations stored in the TLB for process A. As a result, after the CPU switches to process B, process B needs to suffer many TLB misses to load its virtual to physical page translation into TLB. Each TLB miss takes time to handle. But when the OS switches from one thread to another thread of the same process, it doesn’t need to invalidate the translations stored in TLB because both threads belong to the same virtual address space.

7. **Pro:** Hardware handling TLB miss is faster than OS handling miss because the later needs to trigger an exception, saves the current process states, invokes the TLB exception handler to handle the TLB miss.

   **Con:** Hardware handling TLB miss has less flexibility in page table format because the OS needs to conform to the hardware-defined page table format. If OS handles the TLB miss, the OS can change the page table format to use more efficient data structure such as a hash table, invert table, etc.

8. 1 TLB miss, 2 page faults (one page fault for the page table, and one for the page that contains the data), 1 cache miss

9. Because most programs have good spatial and temporal locality so more than 99% of memory accesses, their virtual to physical translation lookups can hit in the TLB.