Sample Exam Answers

Q1

1. A smaller page size leads to smaller page tables – False – need more entries because we have more pages
2. A smaller page size leads to more TLB misses – True – the fixed number of TLB entries cover less virtual memory.
3. A smaller page size leads to fewer page faults – True - In practice, initial faulting-in is usually less important than page faults generated once the application is up and running. Once the app is up and running, reducing the page size results in a more accurate determination of work set, which can be smaller, which requires less memory, which in-turn leads to a lower page fault rate.
4. A smaller page size reduces paging I/O throughput – True
5. Threads are cheaper to create than processes – True
6. Kernel-scheduled threads are cheaper to create than user-level threads – False
7. A blocking kernel-scheduled thread blocks all threads in the process – False. This is true for user-level threads
8. Threads are cheaper to context switch than processes – True – don't have to save the address space
9. A blocking user-level thread blocks the process - True
10. Different user-level threads of the same process can have different scheduling priorities in kernel – False. User level threads appear as a single kernel thread.
11. All kernel-scheduled threads of a process share the same virtual address space – True
12. The optimal page replacement algorithm is the best choice in practice – False - it's impossible to implement
13. The operating system is not responsible for resource allocation between competing processes – False – it is responsible for this
14. System calls do not change to privilege mode of the processor – False – we trap into the kernel so we do change the privilege mode
15. A scheduler favouring I/O-bound processes usually does not significantly delay the completion of CPU-bound processes – True – the I/O processes get the I/O and then yield the CPU again.

Q2

a)

.99 TLB hits, which needs only a single memory access (the actual one required)

.0095 TLB miss, one read from memory to get the page table entry to load TLB, then one read to read actual memory.

.0005 pagefault plus eventual read from memory plus something like one of the following (I'd take a few variations here)

1. A memory reference to update the page table

2. A memory reference to update the page table and a memory reference for the hardware to re-read the same entry to refill the TLB
3. Any reasonable scenario that results in valid page table and refilled TLB.

I'll assume option 2 for final answer

thus

.99 * 100ns + .0095 *2 * 100ns + .0005 (3 * 100ns + 5ms)

would be one answer I would accept.

b)

//Someone should check this is correct...

Many applications just read the first few bytes of a file, for instance to determine the file's type. If these bytes were not in the same disk block as the inode, then when an application asks to read the start of the file the disk would need to read the block with the inode sending this back to the file system code, which would find out the address of the block with the data, then send another request to the disk. Whereas if the first few bytes were in the block with the inode, then only one request to the disk is needed.

// Kevin: valid reason, but also:

A significant fraction of files are small, thus a significant fraction of files can be stored in the inode entirely - thus reducing/avoiding the need to seek to the inode and then seek to the data - it is all in the inode.

Q3

a)

From 20T1, no longer in course

i) FCFS order traversed: 119,58,114,28,111,55,103,30,75

tracks traversed: 547

ii) SSTF order traversed: 75,58,55,30,28,103,111,114,119

tracks traversed: 143

iii) SCAN/elevator order traversed: 103,111,114,119,75,58,55,30,28

tracks traversed: 130

iv) C-SCAN/modified-elevator order traversed: 103,111,114,119,28,30,55,58,75

tracks traversed: 177.

b)

Cooperative scheduling is a system of scheduling where the threads manually yield to allow other threads to have a go. It is commonly used because user-level code normally has no access to timers. That means there is no way to interrupt the thread when it runs too long, so it relies on cooperation from other threads to switch. Sometimes user level code has access to timers, but the units of time are not small enough to give the illusion of parallelism.

//These are advantages of user level threads in general, not of cooperative threading for user level threads The scheduler can be optimized per application and context switches are faster than kernel level context switches.

Q4

- The first 10 bits (left most, 31:22) of the vaddr is the index into first-level page table. The index is used to load the corresponding entry in the first-level table which contains a pointer to the second level table.
- The next 10 bits (21:12) of the vaddr is combined with the pointer to index into the second level table to obtain the page table entry.
- The PTE contains a frame number which is combined with the final 12 bits of the vaddr (11:0) to gain the paddr.
- Assuming 10-bit/10-bit indexing of page table levels.

A diagram would make this answer even clearer.

Q5

a)

Internal fragmentation is when there is the space inside an allocated region is wasted. External fragmentation is when the space outside an allocated region is wasted.

Internal fragmentation is more likely with static partitioning because the memory allocated is often rounded up to something larger than needed, or where the minimum unit of allocation is larger than required - e.g. disks are allocated in whole blocks. With dynamic partitioning, external fragmentation is more likely, because dynamic partitioning often leaves areas of the memory that are too small to place anything.

b)

It is in the multi process lecture notes.
http://cgi.cse.unsw.edu.au/~cs3231/06s1/lectures/lect21.pdf

Starting at page 30.

Just some notes that may be good to point out

Test-and-Set Hardware locks the bus during the TSL instruction to prevent memory accesses by any other CPU Issue: Spinning on a lock requires bus locking which slows all other CPUs down

- Independent of whether other CPUs need a lock or not
- Causes bus contention

Caching does not help this test and set.

Hence to reduce bus contention.

- Read before TSL
  - Spin reading the lock variable waiting for it to change
  - When it does, use TSL to acquire the lock
- Allows lock to be shared read-only in all caches until its released
  - no bus traffic until actual release
- No race conditions, as acquisition is still with TSL.

Q6

1. Mutual exclusion
   - Each resource is either assigned to only one process or available.
2. Hold and wait
   - A process holding resources can request more.
3. No preemption
   - Requests granted cannot be taken away by the OS.
4. Circular wait
   - Each process in a set of at least 2 cannot proceed until it acquires a resource currently held by another process in the set.

One way to solve circular wait would be to set a requirement where they can only request resources in a pre-determined fixed order. All processes can only get and hold resources from a that order, this way, it's not possible for two processes to hold resources that the other needs.

Exam/Sample Exam Answers (last edited 2021-05-03 14:18:45 by KevinElphinstone)