Operating Systems Questions

GATE And Tech

November 14, 2024

Straight-forward True/False

1. An operating system is defined as hardware that converts software into a useful form for applications.

   False. It is software that converts hardware.

2. Examples of resources that the OS must manage include CPU, memory, and disk.

   True.

3. The abstraction that the OS provides for the CPU is a virtual address space.

   False. The CPU abstraction is a process.

4. A process is defined as an execution stream (or thread of control) in the context of a process state.

   True.

5. The address space of a process is part of its process state.

   True.

6. A process is identical to a program.

   False; the process is dynamic, the program is static.

7. A process is identical to a thread.

   False; can have multiple threads in a single process.

8. Two processes reading from the same virtual address will access the same contents.

   False; each process has its own address space.

9. A modern OS virtualizes a single CPU with time-sharing.

   True.

10. Entering a system call involves changing from user mode to kernel mode.

    True.

11. When a user-level process wishes to call a function inside the kernel, it directly jumps to the desired function.

    False; must use system call; after generating a trap which is handled by the OS with a trap handler, the desired system call is invoked through the system call table.

12. An example of a mechanism inside the OS is the process dispatcher.

    True.

13. Cooperative multi-tasking requires hardware support for a timer interrupt.

    False; cooperative assumes process will voluntarily relinquish CPU or enter OS.

14. A timer tick is identical to a time slice.

    False; a time slice (how long a process is scheduled with RR) may be multiple timer ticks long.

15. PCB stands for process control base.

    False; Process Control Block.

16. On a uniprocessor system, there may only be one ready process at any point in time.

    False; just one RUNNING process, but many could be ready and want to be scheduled.

17. A FIFO scheduler schedules ready processes according to their arrival time.

    True.

18. The convoy effect occurs when high priority jobs must wait for lower priority jobs.

    True/**False** both accepted; convoy effect usually implies short jobs must wait for long jobs, but one could consider a short job to be high priority and a long job to be low priority.

19. A SJF scheduler uses the past run time (i.e., cpu burst) of a job to predict future run time (i.e., cpu burst).

    False; SJF assumes an oracle with perfect knowledge of future behavior.

20. A STCF scheduler guarantees that it will schedule the ready job with the smallest remaining cpu burst.

    True.

21. A RR scheduler may preempt a previously running job.

    True.

22. An RR scheduler tends to decrease average response time as the time-slice is decreased.

    True; jobs will be scheduled sooner (decreasing response time) with a smaller time slice.

23. The shorter the time slice, the more a RR scheduler gives similar results to a FIFO scheduler.

False; with a longer time slice (imagine the time slice > cpu burst), RR schedules the jobs without rotating between jobs.

24. If all jobs arrive at the same point in time, a SJF and an STCF scheduler will behave the same.

    True.

25. If all jobs have identical run lengths, a FIFO and a SJF scheduler will behave the same.

    True.

26. If all jobs have identical run lengths, a RR scheduler provides better average turnaround time than FIFO.

    False; if all jobs are identical, RR is horrible for turnaround time because all jobs will complete at nearly the same time.

27. A RR scheduler is guaranteed to provide the optimal average turnaround time for a workload.

    False; RR can provide horrible turnaround time.

28. A SJF scheduler requires an oracle to predict how long each job will perform I/O in the future.

    False; needs a oracle, but it is to predict the next CPU burst of each job.

29. With a MLFQ scheduler, compute-bound jobs are given higher priority.

    False; jobs that do a lot of computation (long CPU burst) are given low priority.

30. With a MLFQ scheduler, high priority jobs have longer time-slices than low priority jobs.

    False; high priorities have short time-slices so that interactive jobs can run promptly, but for just a short time.

31. With a MLFQ scheduler, jobs run to completion as long as there is not a higher priority job.

    False; at the lowest priority level, MLFQ will still RR across jobs of equal priority.

32. The OS provides the illusion to each process that it has its own address space.

    True.

33. The static portion of an address space cannot contain any data.

    False; read-only data could be in a static portion (along with code).

34. Stacks are used for procedure call frames, which include local variables and parameters.

    True.

35. Pointers should not reference the heap.

    False; it can be bad practice to return pointers to data allocated on the stack.

36. An instruction pointer register is identical to a program counter.

    True.

37. A modern OS virtualizes memory with time-sharing.

    False, use space-sharing.

38. A virtual address is identical to a logical address.

    True.

39. With dynamic relocation, hardware dynamically translates an address on every memory access.

    True.

40. An MMU is identical to a Memory Management Unit.

    True.

41. The OS may not manipulate the contents of an MMU.

    False; OS sets up contents of MMU when switch to new process.

42. A disadvantage of segmentation is that different portions of an address space cannot grow independently.

    False; segmentation lets each segment grow independently.

43. A disadvantage of segmentation is that segment tables require a significant amount of space in memory.

    False; segment tables are usually small (just a base and bounds for each segment and not many segments).

44. With pure segmentation (and no other support), fetching and executing an instruction that performs a store from a register to memory will involve exactly one memory reference.

    False; fetching the instruction requires one memory reference and executing the store requires a second memory reference.

45. Paging approaches suffer from internal fragmentation, which grows as the size of a page grows.

    True.

46. A physical page is identical to a frame.

    True.

47. The size of a virtual page is always identical to the size of a physical page.

    True.

48. The number of virtual pages is always identical to the number of physical pages.

    False; the size of the virtual address space can be different than the amount of physical memory.

49. If **8** bits are used in a virtual address to designate an offset within a page, each page must be exactly **256** bytes.

    True; $2^8 = 256$ bytes.

50. If a physical address is 24 bits and each page is 4 KB, the top 10 bits exactly designate the physical page number.

    False; $\lg 4\text{ K} = 12 \rightarrow 12$ bits for page offset; $24 - 12 = 12$ bits for physical page number.

51. If a virtual address is 16 bits and each page is 128 B, then each address space can contain up to 512 pages.

    True. $\text{Lg}\,128 = 7$ bits. $16 - 7 = 9$ bits. $2\text{^}9 = 512$ pages.

52. A linear page table efficiently maps physical page numbers to virtual page numbers.

    False; efficiently maps VPN to PPN (would be expensive to search other way through linear structure).

53. Given a fixed page size, the size of a linear page table increases with a larger address space.

    True; larger address space $\rightarrow$ more pages $\rightarrow$ more page table entries in linear table.

54. Given a constant number of bits in a virtual address, the size of a linear page table increases with larger pages.

    False; larger pages $\rightarrow$ fewer pages $\rightarrow$ fewer entries in linear page table.

55. Given a 28-bit virtual address and 1 KB pages, each linear page table will consume 2^{18} bytes.

    False. 1 KB pages $\rightarrow$ 10 bit offsets. $28 - 10 = 18$ bits for VPN. Each PTE is 4 bytes (coversheet assumption). $2^{18} * 4$ bytes.

56. Page table entries are stored in the PCB of a process when a context switch occurs.

    False; keep base of page table in PCB, but not individual page table entries.

57. Compared to pure segmentation, a linear page table doubles the required number of memory references.

    True; need to look up VPN $\rightarrow$ PPN in page table on every memory access.

58. A disadvantage of paging is that it is difficult to track free memory.

    False; pages are all same size so just use bitmap to track state (free vs. allocated) of each page.

59. A disadvantage of paging is that all pages within an address space must be allocated.

    False; each page table entry must be allocated (with a linear table), but if the page table entries shows the page isn't valid, then the page doesn't have to be allocated.

60. A TLB is identical to a Translation Lookahead Buffer.

    True or **False**; In class, we only used term Translation Lookaside Buffer, but both terms are sometimes used.

61. A TLB caches translations from full virtual addresses to full physical addresses.

    False; TLB translates virtual page numbers to physical page numbers (no offset portion of address in TLB).

62. If a workload sequentially accesses 4096 4-byte integers stored on 256 byte pages, the TLB is likely to have a miss rate around 2^{-8} (ignore any other memory references).

    False. Access 16 KB of data sequentially;

    with 256 byte pages (and perfect alignment), this data fits on $2^{14}/2^8 = 2^6$ pages.

    Assume the first access to each page misses in TLB, while remaining accesses to that page hit in TLB.

    Miss rate = # misses /# accesses = # pages /# accesses = $2^6/2^{12} = 2^{(-6)}$.

63. If a workload sequentially accesses data, the TLB miss rate will decrease as the page size increases.

    True; with large pages, more accesses to same page, whose mapping will already be in TLB.

64. A workload that sequentially accesses data is likely to have good temporal locality, but not necessarily good spatial locality.

    False; good spatial locality but not necessarily temporal.

65. TLB reach is defined as the number of TLB entries multiplied by the size of each TLB entry.

    False; TLB entries multiplied by the size of each page.

66. On a context switch, the TLB must be flushed to ensure that one process cannot access the memory of another process.

    False; can avoid flushing TLB if have ASIDs.

67. A longer scheduling time slice is likely to decrease the overall TLB miss rate in the system.

    True; if a process is scheduled for a longer period of time, it will amortize the cost of the cold start misses to load up TLB with needed translations over a longer period of time.

68. On a TLB miss, the desired page must be fetched from disk.

    False; TLB miss means the page tables must be accessed; page fault will need to access disk.

69. With a TLB, only the outermost page table of each process needs to be accessed.

    False; if a TLB miss, still need to walk entire page table.

70. If the valid bit is clear (equals 0) in a PTE needed for a memory access, the running process is likely to be killed by the OS.

    True; if process tries to access page not valid in its address space, it is a segmentation fault.

71. There is a separate page table for every active process in the system.

    True.

72. An inverted page table is efficiently implemented in hardware.

    False; inverted page tables are implemented in software.

73. One advantage of adding segmentation to paging is that it potentially reduces the size of the page table.

    True; only need page table entries for valid pages in each separate segment.

74. One advantage of adding paging to segmentation is that it reduces the amount of internal fragmentation.

    False; Paging has internal fragmentation.

75. A single page can be shared across two address spaces by having each process use the same page table.

    False; if two processes use the same page table, all of their pages will be shared.

76. An advantage of a multi-level page table (compared to a linear page table) is that it potentially reduces the number of required memory accesses to translate an address.

    False; multiple levels increase the number of memory accesses needed for translation.

77. A page directory is identical to the outermost level of the page table.

    True.

78. With a multi-level page table, the complete VPN is used as an index into the page directory.

    False only outer bits of VPN are used.

79. TLBs are more beneficial with multi-level page tables than with linear (single-level) page tables.

    True; if a TLB hit, able to avoid more memory lookups for page translation (higher cost of a miss).

80. Given a 2-level page table (and no TLB), exactly 2 memory accesses are needed to fetch an instruction.

    False; 2 accesses for 2 levels of address translation +1 for fetch = 3 accesses.

81. With a multi-level page table, hardware must understand the format of PTEs.

    False; not necessary to have hardware support for multi-level page tables.

82. In the memory hierarchy, a backing store is faster than the memory layer above that uses that backing store.

    False; backing stores are larger and slower than the layers above that use it.

83. If the present bit is clear in a needed PTE, then the running process is likely to be killed by the OS.

    False; if present bit is clear, page must be brought in from disk.

84. A page fault is identical to a page miss.

    True.

85. When the dirty bit is set in a PTE, the contents of the TLB entry do not match the contents in the page table.

    False; dirty bit means contents of page in memory do not match contents on disk.

86. The OS can run a single process whose allocated address space exceeds the amount of physical memory available in the system.

    True.

87. The OS can run multiple processes whose total allocated address space exceeds the amount of physical memory available in the system.

    True.

88. When a page fault occurs, it is less expensive to replace a clean page than a dirty page.

    True; clean page can be simply discarded since it matches what is on disk; dirty page must be written to disk to update that (only) copy.

89. When a page fault occurs, the present bit of the victim page (i.e., the page chosen for replacement) will be cleared by the OS.

    True; the victim page will no longer be present in main memory.

90. A TLB miss is usually faster to handle than a page miss.

    True; TLB miss just requires accessing main memory; page miss requires accessing much slower disk.

91. Demand paging is identical to anticipatory paging.

    False; demand paging brings in page only when needed; anticipatory brings in pages beforehand.

92. Prefetching of pages helps sequential workloads to avoid page misses.

    True; prefetching works well with sequential accesses since can predict next page to be accessed.

93. LRU is an example of a mechanism for determining which page should be replaced from memory.

    False; LRU is a policy.

94. The OPT replacement policy replaces the page that is used the least often in the future.

    False; OPT replaces the page that will be used the furthest away in the future (not least often).

95. LRU always performs as well or better than FIFO.

    False; not necessarily.

96. OPT always performs as well or better than FIFO.

    True.

97. LRU with N + 1 pages of memory always performs as well or better than LRU with N pages of memory.

    True, due to stack property.

98. FIFO with N + 1 pages of memory always performs as well or better than FIFO with N pages of memory.

    False, see Belady's anomaly.

99. LRU-K and 2Q use both how recently and how frequently a page has been accessed to determine which page should be replaced.

    True.

100. The clock policy replaces the least-recently-used page belonging to any process in the system.

False; clock approximates LRU, but it doesn't necessarily replace the single least-recently-used page.

Reference: `https://pages.cs.wisc.edu/~dusseau/Classes/CS537/Fall2015/e1-soln.pdf`