OS1-EXCEPTION: 1-30

1 [T / F] The CPU is just a "passive" hardware (in terms of managing the entire computer). The OS (software) must tell the CPU what to do and how it interacts with the other hardware components (memory, disk, fan, power management, etc.).

2 [T / F] OS is essentially just a software, a program. To run, it also needs the CPU.

3 [T / F] If you only have one CPU core, a user code and the OS code can simultaneously run at the same time.

4 [T / F] Every computer motherboard must have a firmware that can execute a boot loader in order to load the OS binary file (e.g. vmlinux for Linux) from a specific disk location to the memory during boot up.

5 [T / F] If you run an infinite-loop program, your computer will freeze and you cannot do other activities (e.g. cannot run other programs, your keyboard/mouse is unresponsive).

6 [T / F] If you only have one CPU, and you run a long running program, another shorter program must wait until the long program finishes.

7 [T / F] If you have a divide-by-zero error in your program, that line will be skipped and the next instruction line will be executed.

8 [T / F] The CPU can directly read your file data from the disk just like a regular load/store instruction, *e.g.*, load Register1 [diskAddr] (instead of load Register1 [memAddr]).

9 [T / F] Because CPU is also shared with user processes, there must be some mechanism to "jump to the OS" (*e.g.*, the timer interrupt hardware and the interrupt/exception table) so that the OS can regain control of the CPU.

10 [T / F] To handle exceptional flow, hardware support is needed (*e.g.*, exception table).

11 [T / F] In this class, "interrupt table" and "exception table" are the same.

12 [T / F] The OS stores the interrupt table in its heap area in DRAM.

13 [T / F] You can configure the number of entries available in the interrupt table.

14 [T / F] The exception table contains a list of function names (the exception handlers) in strings.

15 [T / F] The CPU knows the locations of the exception handlers in the memory because they are *fixed* addresses that both the CPU and the OS must agree on before the computer starts.

16 [T / F] Divide-by-zero errors are first detected by the OS.

17 [T / F] Regardless of how many times your process is interrupted (by the timer interrupt), the correctness of your process will not be altered (one side effect is reduced performance).

18 [T / F] When the CPU is in the middle of executing an instruction (*e.g.*, add, sub, load, store) and the timer oscillator generates a timer interrupt, the CPU will stop and cancel the current instruction and jump to the OS.

19 [T / F] A "context switch" is when a CPU switches from running one process to another process (e.g. run process A then run process B).

20 [T / F] User-kernel switches is when a CPU switches from running a user program to running the OS code.

21 [T / F] A simple way to think about synchronous vs. asynchronous interrupt is that synchronous interrupts come from your processes at expected time (you need it to happen) but asynchronous interrupts come "asynchronously" outside the control of your process (*e.g.*, timer interrupt).

22 [T / F] Too much context switching is good and too much love will kill you.

23 [T / F] ...

24 [T / F] ...

25 [T / F] ...

26 [T / F] ...

27 [T / F] ...

28 [T / F] ...

29 [T / F] ...

30 [T / F] ... 30