OS2-SYSCALL+USER/KERNEL/MODE+PROCESSES: 31-60

os2-syscall:

31 [[✓ T] / F] For shared hardware peripherals such as disks, network cards, and monitors that user-mode applications cannot access directly, system calls are provided to allow the applications to interact with shared hardware.

32 [[✓ T] / F] Library calls are typically more user friendly (easier to use) than system calls.

33 [T / [✓ F]] The OS code and user program share the same stack area, so a program can put function arguments in the stack before making a system call.

34 [T / [✓ F]] If your process "jumps to the OS," it means your program executes a jump instruction that directly jumps to a specific function address in the kernel space (e.g. "jmp 0xc0012000" where 0xc0012000 is for example the system call handler physical address).

35 [[✓ T] / F] To pass arguments to the OS' syscall handler, the C library puts the arguments in CPU registers (ebx, ecx, etc.) and puts the syscall number in the eax register.

36 [T / [✓ F]] In any OS and on any hardware, "int 0x80" always represents a system call jump to the OS code.

37 [T / [✓ F]] How many system call functions your OS can provide (*e.g.*, read, write, open) depends on the size of the interrupt table.

38 [T / [✓ F]] When the OS boots up, it registers the function address of every system call function (*e.g.*, read, write, open) to the interrupt table.

39 [T / [✓ F]] If "0x80" is the syscall entry in the interrupt table, then "0x81" should be the entry for the first system call function address (e.g. sys_exit, which is syscall #1).

40 [T / [✓ F]] All operating systems must use the same system call numbers for the same type of operation (e.g. exit() must be syscall #1).

41 [[✓ T] / F] In our lecture, write() is compiled to x86 instructions where eax is set to 4. This is because the compiler (gcc) knows the OS and the hardware you're currently using.

42 [[✓ T] / F] If you have root access in a Linux computer, you can downlaod and modify Linux kernel code, add new specific system calls (*e.g.*, hideMyFilesFor1Hour()), compile and install your new kernel.

os2-user/kernel mode:

43 [T / [✓ F]] In my activity monitor, if I see a lot of "red" area (e.g. 90% of the time the CPU is spending in the system/kernel mode), I should be happy because it means my OS is working out its muscles.

44 [[✓ T] / F] Specific x86 instructions that communicate with the disks, network cards, and the MMU, are categorized as "privileged" instructions as they can only be run when the CPU is in kernel mode. When a hacker running in user mode runs these instructions, the CPU will throw an illegal instruction exception that jumps to the OS kernel, and the OS will kill this illegal process.

45 [T / [✓ F]] System calls (user-kernel crossing) incur the same cost as library calls.

46 [[✓ T] / F] You can write to your file via write() or fwrite(), but fwrite() is just a C library call that wraps the write() system call. Library functions usually have more features (*e.g.*, buffering, caching).

47 [[✓ T] / F] In general, it is a good practice to use library calls (that wrap system calls) than using system calls. This is because C libraries have a lot of optimization that can benefit your applications.

48 [T / [✓ F]] The library function strcpy() eventually makes a system call.

49 [T / [✓ F]] The only way CPU enters kernel mode is when a program makes "int 0x80".

os2-processes:

50 [T / [✓ F]] Abstractions such as "processes", "heap", etc. are hardware concepts.

51 [[✓ T] / F] The concept of "processes" (including the code/stack/heap segments) *abstracts* out the underlying processor and memory, i.e., as a programmer you just need to deal with global/local variables, pointers, functions, etc. (which are all encapsulated in the concept of process address space).

52 [T / [✓ F]] A program can be run as multiple processes, but a process cannot spawn multiple programs (in new processes).

Make → gcc

53 [T / [✓ F]] Every program has a current working directory (CWD) context stored in the program binary file.

✱ Parent -thesu

54 [[✓ T] / F] Every process has a current working directory (CWD) context stored in its process control block.

55 [T / [✓ F]] If you're a browser architect, it's a good practice to run a 3rd-party plugin programs within the browser process.

56 [[✓ T] / F] Process is a unit of fault containment.

57 [T / [✓ F]] If a browser employs a design that every tab is a separate process, then if a tab crashes the whole browser window will crash.

58 [[✓ T] / F] Every process has its own process state that carries information such as the process ID, program counter, stack/heap locations, etc.

59 [T / F] ...

60 [T / F] ... 60

Segmentation fault.