Due at BEGINNING of class, Monday, February 10. Where indicated, you must show your work to receive credit. You must do your work on THESE sheets and submit. Please write your name at the top of this sheet and on the back of the last sheet.

1. (8 pts) A Linux process can enter kernel mode via either a software interrupt or a hardware interrupt.
   
   (a) (2pts) What does it mean, specifically, for the processor to switch to kernel mode?

   (b) (3 pts) Give three specific examples of software interrupts.

   (c) (3 pts) Give three specific examples of hardware interrupts.

2. (6 pts) Briefly explain the difference between a segmentation fault and a page fault.
3. (5 pts) The interrupt vector table is initialized during the boot process. What does this table contain and how does it affect the execution of the operating system?

4. (5 pts) Every Linux process (actually every thread) has both a user stack and a kernel stack. Why does a process need a separate kernel stack?

5. (8 pts) Note: review from CSE 325. Suppose a 32-bit processor uses a 32KB 8-way set associative cache for its level-1 cache. If each cache line (block) contains 64 bytes, then:

   (a) (4 pts) How long is the tag for a cache entry? Show your work.

   (b) (4 pts) Draw the layout for an address, showing the number of bits and location of the: (i) tag; (ii) index; (iii) block offset.
6. (5 pts) Consider the class discussion of signals for a 32-bit architecture. Suppose a process wakes up and finds two signals, number 2 and number 9, have been posted against it. Can this process survive and return to user level? Explain why or why not.

7. (6 pts) Computer programs tend to exhibit high locality of reference. Briefly explain what this means and give two implications related to memory management.

8. (5 pts) A multiprocessor kernel needs locks around critical sections of code that might otherwise be executed in parallel on two or more processors. However, even on a uniprocessor system, one will find critical sections in the kernel that are protected by locks. Why are locks needed on a system with only one processor?

9. (6 pts) Consider the implementation of Linux system calls shown in class. All system calls trap to the same location in the kernel.
   
   (a) (3 pts) How does the kernel know which system call was invoked?
   
   (b) (3 pts) Where does the kernel find the parameters to the system call?
10. (9 pts) Consider the 32-bit virtual-to-physical address translation using two-level page tables, as shown below and as discussed in class. (Traditionally, the top level page table was referred to as the page directory.) Pages are 4KB long and each page table entry is 4 bytes long, so a page (and hence a page table) can contain 1024 page table entries (PTEs). Suppose the addresses labeled in the figure are as follows:
- virtual address: 0x021aff2b
- page directory address: 0x30200000
- page table address (top 20 bits): 0xa39a6
- page frame address (top 20 bits): 0x330e2

(a) (3 pts) What is the value of \( PA_1 \), the physical address of the relevant entry in the page directory? Show your work.

(b) (3 pts) What is the value of \( PA_2 \), the physical address of the relevant entry in the page table? Show your work.

(c) (3 pts) What is the value of \( PA_3 \), the physical address to which the virtual address is mapped? Show your work.
11. (6 pts) Suppose that when the processor references virtual addresses 0x003F0A30, it turns out that the TLB contains the mapping (a TLB hit), saving the processor from walking the page tables.

(a) (3 pts) What other virtual addresses will be mapped by the same TLB entry? Briefly explain your answer.

(b) (3 pts) Suppose the next address referenced by the process is 0x003F0A34. Describe a scenario where the mapping for this address is not present in the TLB.

12. (6 pts) Modern computing systems use 48-bit addresses and 4KB pages. Page table entries are 8 bytes long. Draw a picture, analogous to that in problem 10, showing the virtual-to-physical address translation via page tables.
13. (6 pts) When a process catches a signal (for example, using the sigaction() system call, the signal handler is executed at user level, not kernel level. Briefly explain how the kernel “gaslights” the process in order to accomplish this task.

14. (5 pts) For the early version of Linux discussed in class, bottom half handling (BHH) refers to the fact that some processing related to an interrupt is delayed until a later time, such as immediately before returning to user level. Briefly explain how BHH is implemented.

15. (5 pts) When one process spawns another process, for example using fork(), pages containing text (code) are almost always shared between the parent and the child. Even pages containing writeable data are shared until one of the processes writes to the page, at which time each process gets its own copy. Briefly explain how this “copy-on-write” functionality is implemented.