CS 219 Spring 2024: Quiz 1

(20 questions, 30 marks, 15% weightage)

Name:_____ **Roll number:**_____

1. **[4 marks]** Consider the output of the program shown below. Assume it compiles and runs successfully.

```
int a = 4;
while(a > 0) {
  int ret = fork();
  if(ret == 0) {
    printf("a=%d\n", a);
  }
  else wait(NULL);
  a--;
}
```

   (a) How many times is the statement `a=4` printed?

   (b) How many times is the statement `a=3` printed?

   (c) How many times is the statement `a=2` printed?

   (d) How many times is the statement `a=1` printed?

2. **[2 marks]** What is the output of the below program? Assume it compiles and runs successfully.

```
int a = 4;
while(a > 0) {
  int ret = fork();
  if(ret == 0) {
    execl("/bin/echo", "/bin/echo", "hello", NULL);
  }
  else wait(NULL);
  printf("a=%d\n", a);
  a--;
}
```

3. **[4 marks]** Consider a system running xv6, in which a parent process P makes the "fork" system call to spawn child C. Shown below is the superset of actions that can occur during the execution of this system call, in a jumbled chronological order.

   (A) User context in general purpose registers is saved on to the kernel stack of P.

   (B) The scheduler thread is invoked, which finds runnable process C and switches to it.

   (C) Process C calls trapret, and the trap frame is popped from the kernel stack of C.

   (D) The `int n` instruction is executed by the CPU.

   (E) The context structure is popped from the kernel stack of P.

   (F) The function named "fork" in the xv6 user library is invoked.

   (G) Process P calls trapret, and the trap frame is popped from the kernel stack of P.

   (H) EIP and ESP corresponding to usermode context are pushed on to the kernel stack of P.

   (I) The context structure is popped from the kernel stack of C.

   (J) A new `struct proc` entry and a new kernel stack are allocated for C.

   (K) The context structure is pushed onto the kernel stack of P.

   (L) The C trap handling function is called to handle the system call.

   (a) Arrange the above statements in chronological order (earliest to latest) to describe the sequence of steps that occur when the parent process P goes into kernel mode to execute the fork system call, and comes back into user mode again, without any context switch. Note that some of the steps above which do not occur can be left out. List your answer as a sequence of steps, e.g., A, B, C, D, ...

   (b) Repeat the above question in the alternate scenario where the xv6 scheduler is invoked after P finishes the forking, but before it returns to user mode. Assume the scheduler picks process C to run next, instead of P. Describe the sequence of steps beginning from P making the system call to C returning back in user mode.

4. **[4 marks]** Consider an xv6 system. For each of the processes described below, draw a figure showing the contents of the kernel stack, from top to bottom. You must indicate all the important structures like the context structure, trap frame, and anything else we have discussed in class. You must also describe the values of the EIP register stored in these structures.

   (a) A newly forked process that is waiting to be scheduled, but has not yet run.

   (b) A process that has made a system call, has gone into kernel mode, has just finished executing the system call, but has not yet returned from the trap to usermode.

The True/False questions given below carry 1 mark each. You must answer True or False correctly, along with a short explanation justifying your answer. For example, if you answer False, you can provide a counter-example where the statement is false, or explain what the actual true statement is. If you answer True, you can explain why the statement is true.

5. Process P1 spawns P2, which in turn spawns process P3. Processes P1 and P3 are still running, but P2 has terminated. Process P2 will be reaped by the init process. [T/F]

6. Consider a process P in an xv6 system. It is possible sometimes that the context structure is pushed onto the kernel stack of P, but is never popped off it. [T/F]

7. Consider a user process running in the foreground in a Linux system. When the user hits Ctrl+C on the keyboard to terminate this foreground process, the IDT entry to handle the resulting interrupt points to the process signal handling code. [T/F]

8. A user program written on one system running a POSIX-compliant OS never needs to be recompiled to run correctly on another system which also runs a POSIX-compliant OS. [T/F]

9. A CPU in privileged mode executes only privileged instructions. [T/F]

10. When a parent process forks a child C in xv6, all fields from the `struct proc` of P are copied into the `struct proc` of C, with the exception of the PID field. [T/F]

11. When a hardware device raises an interrupt, the trap instruction is invoked by the OS device driver code that handles the interrupt. [T/F]

12. When a system call is made, the trap instruction is invoked by userspace code and not by kernel code. [T/F]

13. In xv6, only a process in the sleeping/blocked state or in the terminated state will invoke the `sched()` function to give up the CPU to the scheduler thread, but a process in the ready/runnable state will never do so. [T/F]

14. Some system calls cause a process to transition from user mode to kernel mode and back to user mode again, without resulting in a context switch to another process. [T/F]

15. A process P running in an xv6 system has two child processes, one of which is a zombie process that has terminated, while the other child is still running. When process P invokes the wait system call in this scenario, the system call does not block. [T/F]

16. A process P running in an xv6 system invokes the exit system call. This system call blocks P until all the child processes of P have been terminated and reaped. [T/F]

17. A process that invokes the exec system call always overwrites its old memory image, and never runs code from the old memory image again. [T/F]

18. In xv6, the PID that is allocated to a newly created process during fork is available for reuse once again after the process terminates with exit. [T/F]

19. The `ptable` array in xv6 contains the list of only the ready/runnable processes in the system. [T/F]

20. In xv6, a process whose parent has already terminated never becomes a zombie upon invoking the exit system call. Instead, the process is reaped and its memory image is cleaned up during the exit system call itself. [T/F]

ROUGHWORK