Concept Check

The following are conceptual questions meant to be answered in your design document. You won't need to write any code for these questions, but you may need to read and reference some.

1. When a kernel thread in Pintos calls `thread_exit`, when and where is the page containing its stack and TCB (i.e. struct thread) freed? Why can't we just free this memory by calling `palloc_free_page` inside the `thread_exit` function?

2. When the `thread_tick` function is called by the timer interrupt handler, in which stack does it execute?

3. Suppose there are two kernel threads in the system, thread A running `functionA` and thread B running `functionB`. Give a scheduler ordering in which the following code can lead to deadlock.

```
struct lock lockA; // Global lock
struct lock lockB; // Global lock

void functionA() {
    lock_acquire(&lockA);
    lock_acquire(&lockB);
    lock_release(&lockB);
    lock_release(&lockA);
}

void functionB() {
    lock_acquire(&lockB);
    lock_acquire(&lockA);
    lock_release(&lockA);
    lock_release(&lockB);
}
```

4. Consider the following scenario: there are two kernel threads in the system, Thread A and Thread B. Thread A is running in the kernel, which means Thread B must be on the ready queue, waiting patiently in `threads/switch.S`. Currently in Pintos, threads cannot forcibly kill each other. However,

suppose that Thread A decides to kill Thread B by taking it off the ready queue and freeing its thread stack. This will prevent Thread B from running, but what issues could arise later from this action?

5 Consider a fully-functional and correct implementation of this project, except for a single bug, which exists in the kernel's `sema_up` function. According to the project requirements, semaphores (and other synchronization variables) must prefer higher-priority threads over lower-priority threads. However, the implementation chooses the highest-priority thread based on the *base priority* rather than the *effective priority*. Essentially, priority donations are **not taken into account** when the semaphore decides which thread to unblock. **Please design a test case that can prove the existence of this bug.** Pintos test cases contain regular kernel-level code (variables, function calls, if statements, etc) and can print out text. We can compare the expected output with the actual output. If they do not match, then it proves that the implementation contains a bug. **You should provide a description of how the test works, as well as the expected output and the actual output.**
