CSCI 1670/CSCI 2670
Homework Assignment 1

Due 11:59pm, Friday, February 14, 2025.

Please be concise!

1. The textbook, on page 101, describes how to switch from one thread to another (this also appears in slide II-23). The implicit assumption is that the thread being switched to in **thread_switch** has sometime earlier yielded the processor by calling **thread_switch** itself. Suppose, however, that this thread is newly created and is being run for the first time. Thus when its creator calls **thread_switch** to enter its context, it should start execution as if its first function had just been called.

    a. [40%] Show what the initial contents of its stack should be to make this happen (i.e., please draw a diagram showing the stack). Also, show what the stack looks like immediately after **thread_switch** returns in the context of that stack. (I.e., draw pictures of the stack before and after the first call to **thread_switch**.) Assume an x86-like assembler language, as used in Chapter 3 of the text. Also, assume the IA-32 calling convention, in which arguments are on the stack and the base pointer (ebp) is used. For this part of the problem you may assume the thread terminates by explicitly calling **uthread_exit**, i.e., you don't need to handle what happens when the thread returns from its first function. There should be a return address on the stack, but it's not important what its value is.

    b. [10%] When a thread returns from its first function, it should call **uthread_exit**, passing it the value returned by the first function. Explain how to modify your answer to part a so that this is done. If your solution requires some additional executable code, be sure to indicate what this code does. (Hint: it's not necessary to have an additional stack frame in the initial stack, but it may be necessary to employ an additional function.)

2. An operating system is said to be kernel-preemptible if threads may be preempted, even while running in kernel mode, and their processor switched to another thread. There may, of course, be certain code sequences during which such preemption is disabled. An operating system is said to be SMP-safe (symmetric multiprocessor safe) if it can run on a multiprocessor system in which any thread can run on any processor, whether in user mode or privileged mode, and any interrupt handler can run on any processor.

    a. [35%] If an operating system is kernel-preemptible, is it necessarily SMP-safe? Explain why not.

    b. [15%] If an operating system is SMP-safe, is it necessarily kernel-preemptible? Explain why not. [Hint: assume that processors don't share page tables with one another, but each has a private set of page tables, and that page tables are never manipulated via interrupt handlers.]