Homework 3: Interrupt handlers

Answer the following questions.

1.
```
int schedule_disabled = 0; // global
struct list *ready_list; // global
struct thread *cur_thread; // global

void schedule1 () {
    if (schedule_disabled) {
        return;
    }
    schedule_disabled = 1;
    push_list(ready_list, cur_thread);
    schedule();
    schedule_disabled = 0;
}
```

   Consider the above implementation of the schedule1 routine. Is it possible that the ready list can go to an inconsistent state? Does this prevent deadlock? Justify your answer. [0.5]

2. Suppose we don't have hardware support for disabling interrupts (e.g., cli, EFLAGS, etc.). How do you emulate the same behavior using the software? [0.5]

   A global variable `interrupt_disabled` can be used to store the status of the interrupt. Therads can set this variable to disable the interrupt. The interrupt handlers can simply return if this flag is set. We cad add the following instructions at the start of every interrupt handler.

3.
```
bar:
1. push %ebp
2. mov %esp, %ebp
3. mov $100, %eax
4. mov %ebp, %esp
5. pop %ebp
6. ret
```

```
foo:
1. push %ebp
2. mov %esp, %ebp
3. mov $101, %eax
4. mov %ebp, %esp
5. pop %ebp
6. ret

interrupt_handler:
1. push %eax
2. push %edx
3. push %ecx
4. call schedule1
5. pop %ecx
6. pop %edx
7. pop %eax
8. iret
```

Let us consider that we have two threads `foo` and `bar`. `bar` is the current thread, and `foo` is the only thread in the ready list. `foo` was preempted earlier after instruction-2 (in `foo`). `bar` had received an interrupt after instruction-3 (in `bar`) due to which `interrupt_handler` was called. `interrupt_handler` was interrupted again after instruction-2 (in `interrupt_handler`), and `interrupt_handler` was called again. Right not the CPU is executing the first instruction of the interrupt handler (after receiving the second interrupt). Let us assume that the CPU will not receive an interrupt until one of the routines `bar` or `foo` starts executing again. Under this assumption, the `context_switch` routine will be called twice before executing `foo` or `bar`. You can find the implementation of `schedule1`, `schedule`, and `context_switch` in the lecture slides.

- What will be the call-stack at the start of the `context_switch` when it is called the first time. [0.25]
- What will be the call-stack at the end of the `context_switch` when it is called the first time. [0.25]
- What will be the call-stack at the start of the `context_switch` when it is called the second time. [0.25]
- What will be the call-stack at the end of the `context_switch` when it is called the second time. [0.25]

By call-stack, we mean the stack of return addresses. You can use the function names corresponding to a return address to represent the return address.

How to submit

Submit your handwritten homework in the submission box placed at the old academic building (2nd floor). The box will be placed on days when the homework is due.