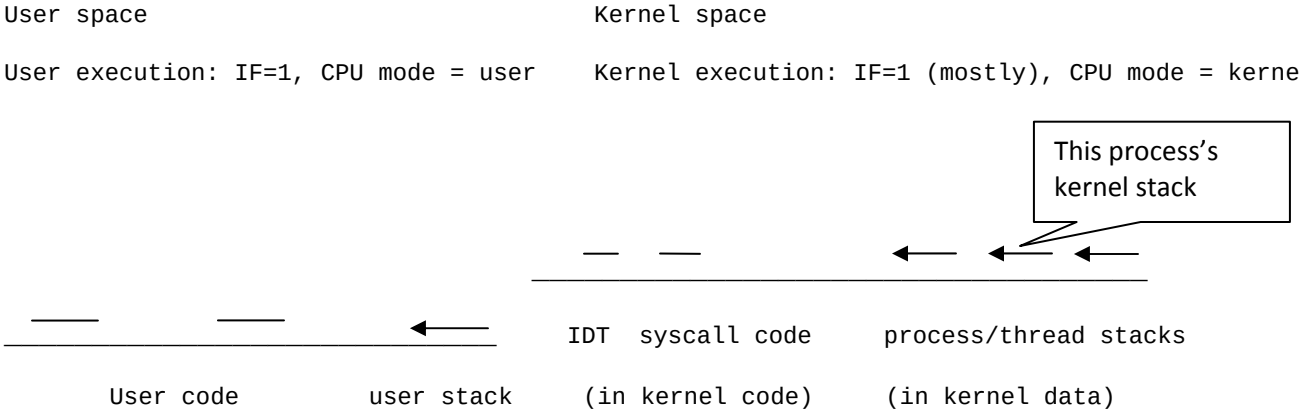
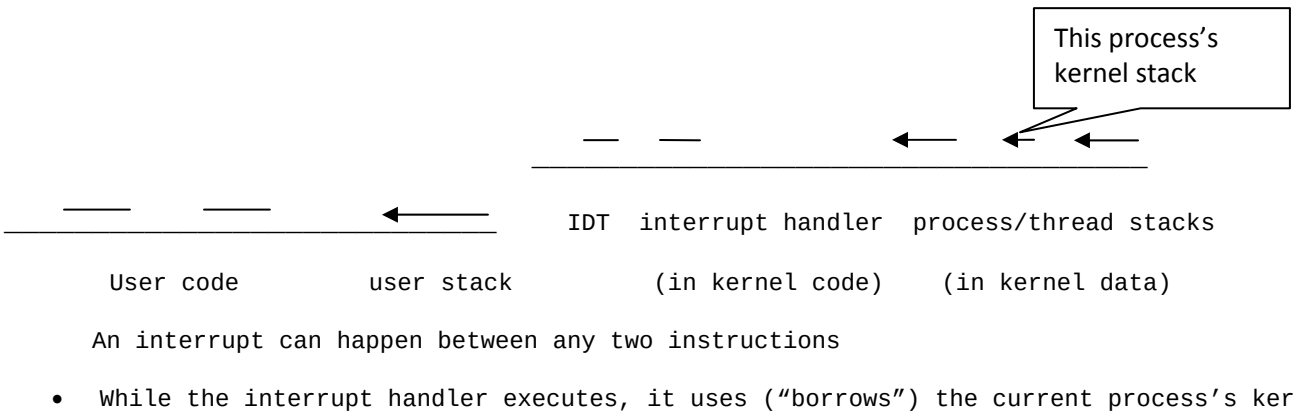
CS444 User-Kernel Transitions, showing stacks (uniprocessor case)

System call from user code (single-threaded process)



- While the syscall executes in the kernel, it uses ("legitimately") the current process's kernel stack.
- A single-threaded process has one kernel stack, for its one thread, so we can say it's the process's kernel stack. A multi-threaded process has multiple user thread stacks and multiple kernel stacks.

Interrupt in user code (single-threaded process, no interrupt stack in use by kernel)

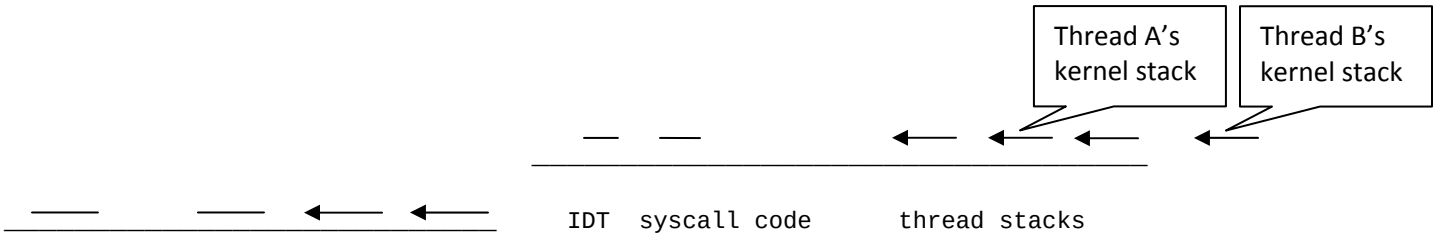


- While the interrupt handler executes, it uses ("borrows") the current process's kernel stack. I've drawn it smaller to indicate that it has just started growing from being empty during user execution.

System call from user code (multi-threaded process)

User space Kernel space

User execution: IF=1, CPU mode = user cycle Kernel execution: IF=1, mode = kernel after trap

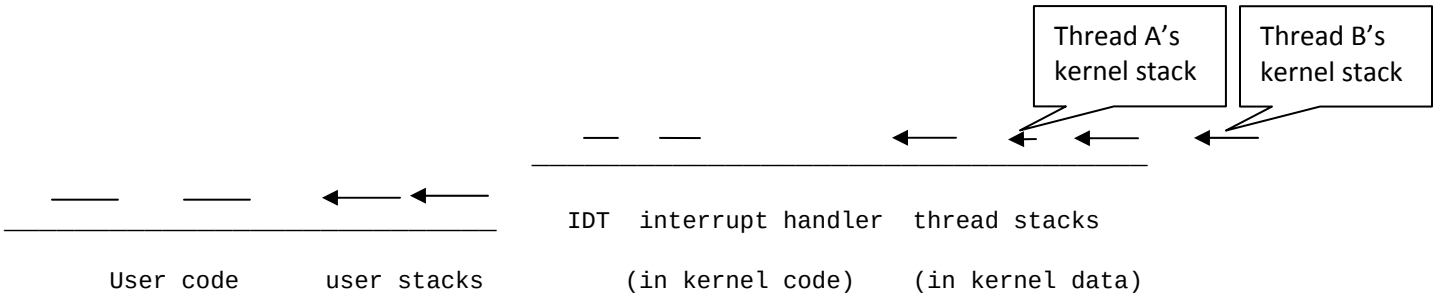


- While the syscall executes in the kernel, it uses ("legitimately") the current thread's kernel stack.
- The other thread is preempted or blocked, so also has a sizable kernel stack

Interrupt in user code (multi-threaded process, 2 threads, one interrupted)

User space Kernel space

User execution: IF=1, CPU mode = user Kernel execution: IF=0, CPU mode = kernel



- While the interrupt handler executes, it uses ("borrows") the current thread's kernel stack.
- Here thread A is interrupted from user execution, so has a small kernel stack
- Thread B has been preempted or blocked, so has a bigger kernel stack

```
while (TRUE) {
    get_next_request(&buf);
    handoff_work(&buf);
}
```

(a)

```
while (TRUE) {
    wait_for_work(&buf)
    look_for_page_in_cache(&buf, &page);
    if (page_not_in_cache(&page))
        read_page_from_disk(&buf, &page);
    return_page(&page);
}
```

(b)

Figure 2-9, pg. 99 Web server code (a)Dispatcher (b) Worker