1. In our experience, we've found that there are several advantages of using interrupts and ISRs over blocking code. For example, by using ISRs, our program can continue executing other tasks while waiting for an event, making it more efficient. Blocking code, like `delay()`, would halt the execution of our program, which would waste CPU cycles. Additionally, ISRs provide quick responsiveness to events, which is not the case with blocking code. However, there are also disadvantages. Implementing interrupts and ISRs makes our program more complex to debug and understand, and it requires us to set aside some resources for their use, which could limit what's available for the rest of our program. And no, ISR-driven tasks are not always faster. While ISRs can respond quickly to an event, the actual execution of the ISR code might take longer, especially if the ISR has to do a lot of work or if there are many interrupts occurring.

2. We keep our ISR functions simple and small for several reasons. First, while an ISR is running, most systems, including the ATmega2560 on our Arduino Mega 2560, will not respond to other interrupts. So, keeping ISRs short reduces the chance of missing an interrupt. Second, long ISRs can cause delays in the execution of the main program. By keeping ISRs short, we help prevent this. Lastly, ISRs usually have a limited stack, so long or complex ISRs might cause a stack overflow.

3. For this lab, we used the Timer 3 Compare Match A interrupt, which is indicated by the line `ISR(TIMER3_COMPA_vect)` in our code 【7†source】. The relative priorities for the other interrupts on the ATmega2560 are generally as follows:

a. SPI Serial Transfer Complete: This interrupt generally has a lower priority than the Timer interrupts.
b. RESET: The RESET interrupt is a non-maskable interrupt and typically has the highest priority on most systems, including the ATmega2560.
c. Pin Change Interrupt 2: This interrupt has a lower priority than the Timer interrupts.
d. External Interrupt 7 (INT7): This interrupt has a higher priority than Timer interrupts.
e. Timer/Counter(n) overflow: If n is the same as the Timer we used (in this case Timer 3), then the priority might be the same or depends on the specific order within the interrupt vector table. For the ATmega2560, these interrupts generally have a higher priority than the Timer Compare Match interrupts.

4. We chose the SSRI (Simple Round Robin with Interrupts) scheduler over the RR (Round Robin) scheduler because the SSRI scheduler can handle interrupts. This allows us to immediately respond to events and execute high-priority tasks, whereas the RR scheduler, without this feature, would have to wait until the currently executing task has finished its time slice.

5. We use a Task Control Block (TCB) in our multitasking system to store information about each task, including the task's priority, state (running, ready, blocked, etc.), and other task-specific data. TCBs are essential for context switching, where the system switches from running one task to another. By using a TCB, our scheduler can save the current state of a task when it is

interrupted and then restore that state when the task is resumed. This allows tasks to be effectively paused and resumed without losing their progress.