Details of what the lowest level of the operating system does when an interrupt occurs:

1. **Hardware stacks program counter, etc.**: When an interrupt occurs, the CPU saves its current state, including the program counter and other relevant information, onto the stack. This is done to preserve the context of the interrupted program.
2. **Hardware loads new program counter from interrupt vector.**: The CPU loads the program counter with the address of the interrupt handler routine from the interrupt vector table. This is the address where the execution will continue after handling the interrupt.
3. **Assembly-language procedure saves registers.**: The interrupt handler, typically written in assembly language, saves any registers it intends to modify during its execution. This is necessary to ensure that the state of these registers is preserved.
4. **Assembly-language procedure sets up new stack.**: In some cases, especially in systems with multiple stacks (e.g., kernel stack, user stack), the interrupt handler may set up a new stack or switch to a different stack, depending on the context of the interrupt.

   ensures that the handler's operations do not accidentally modify or corrupt the user program's stack.
5. **C interrupt service runs (typically reads and buffers input).**: The interrupt service routine (ISR), which may be written in C or another high-level language, runs and performs specific tasks related to the interrupt. For example, in the case of input/output (I/O) interrupts, the ISR might read and buffer input data.
6. **Scheduler decides which process is to run next.**: After the ISR completes its work, the scheduler determines which process should run next in a multitasking environment. It may decide to continue running the interrupted process or switch to another process.
7. **C procedure returns to the assembly code.**: Once the ISR has completed its work and the scheduler has made a decision, control is returned to the assembly-language code.
8. **Assembly-language procedure starts up new current process.**: The assembly-language code sets up the CPU registers and state to start executing the selected process, whether it's resuming the interrupted process or starting a new one.

This sequence represents a simplified view of interrupt handling in a computer system. In reality, the details can be more complex, and the specific steps can vary depending on the architecture of the CPU and the operating system's design. Additionally, modern systems often involve more layers of abstraction and hardware/software interactions.