Basics on Interrupt Handler

A handler is in fact the routine that is called by an interrupt, or in other words, it's the interrupt service subroutine (ISR) itself. Why a different name? Well, the word handler is normally used for ISRs created by you, the programmer, as opposed to those that are pre-built either in the OS or BIOS.

The next question is why to create a handler if the ISRs are already present? The answer is simple: To have more control and flexibility. Without handlers, your programs would have to abide by strict and rigid rules, which would limit their usefulness. Handlers are indispensable in several situations.

Creating Handlers

Writing an interrupt handler in it self is quite straightforward. A program preparing to handle interrupts must do the following:

1. Disable interrupts, if they were previously enabled, to prevent them from occurring while interrupt vectors are being modified.
2. Initialize the vector for the interrupt of interest to point to the program's interrupt handler.
3. Ensure that, if interrupts were previously disabled, all other vectors point to some valid handler routine.
4. Enable interrupts once again.

The interrupt handler must observe the following sequence of steps:

1. Save the system context (registers, flags and anything else that the handler is suitable of modifying and that wasn't saved

automatically by the CPU), normally by pushing the desired elements into the stack or saving them into variables.
2. Block any interrupts that might cause interference if they are allowed to occur during this handler's processing. Please note that sometimes you'll have to disable all interrupts.
3. Enable any interrupts that should still be allowed to occur during this handler's execution.
4. Determine the cause of the interrupt.
5. Take the appropriate action(s) for the interrupt.
6. Restore the system context, usually by popping the elements from the stack or by reading the variables.
7. Re-enable all interrupts blocked.
8. Resume execution of the interrupted process.

When writing an interrupt handler, take it easy and try to cover all the bases. The main reason interrupt handlers have acquired such a mystical reputation is that they are so difficult to debug when they contain obscure errors. Because interrupts can occur asynchronously - that is, because they can be caused by external events without regard to the state of the currently executing process - bugs can be a real problem to detect and correct. This means that an error can manifest its presence in the program much later than it actually occurs

References: 1.
http://www.delorie.com/djgpp/doc/ug/interrupts/inthandlers1.html

2.
http://www.delorie.com/djgpp/doc/ug/interrupts/inthandlers2.html