Using CTRL-C to stop our web server can be problematic. Not only is it inelegant but it will leave any sockets we are using open which in turn means that we can't reuse the port number until the operating system cleans up the mess we left behind. You will find it much simpler to debug your program if you don't have to keep changing port numbers. We can solve this problem with signals.

Signals are a mechanism used to communicate asynchronous extraordinary events to a process. Even if you are unaware of it, you probably use signals all the time. The key-combination CTRL-C, CTRL-Z, CTRL-S, CTRL-D all send a signal to the current process. When something sends a signal to a process that process must stop whatever it is doing and respond to the signal. Signals are identified by number. Pressing CTRL-C sends signal #2 to the process CTRL-Z sends signal 18.

Each signal has a default behavior. Signal #2's default behavior is to terminate the process. Signal #18's default behavior is to pause but not exit (you can resume by sending signal #19).
There is a full list of all the signals used by Linux and what those signals mean on Wikipedia (https://en.wikipedia.org/wiki/Signal_(IPC)).

To avoid issues with leftover sockets we can change the behavior of signal #2. Rather than immediately terminating the process we can change signal #2 so that it will close any open sockets or files and then terminate gracefully.

The term signal handler refers to a function that is called when a process receives a signal. We can write a signal handler that will close all the open file descriptors (sockets, files etc.) and terminate the program. Note that since our function will not know what file descriptors are open, we simply close everything.

```
#include <signal.h>

void sig_handler(int signo) {
    DEBUG << "Caught signal #" << signo << ENDL;
    DEBUG << "Closing file descriptors 3-31." << ENDL;
    closefrom(3);
    exit(1);
}
```

With our signal handler defined, we can add the following to main() to replace the default behavior of signal #2[1] with our signal handler.

```
// Catch SIGINT and send it to sig_handler
signal(SIGINT,sig_handler);
```

[1] The signal numbers are not always the same on all systems, so it is better if we use the system defined constant as a name for the signal rather than the number. The name for the signal sent by CTRL-C is SIGINT