Signals

- It is extremely useful if the OS is able to interrupt a process in a similar way that hardware is able to interrupt the OS.
- Sometimes we may wish to catch an execution error in the program (*e.g.* divide by zero, segmentation fault) and ensure data is saved before termination.
- Sometimes we require periodic interruption by setting a timer, or may wish to terminate another process.
- Such asynchronous processing is enabled in UNIX OS by the kernel's signalling service, which allow simple signals to be sent between processes (or from a process to itself). These are sometimes referred to as *software interrupts*.
  - To handle a particular signal, a process must explicitly register a signal handler routine (much like an interrupt handler in the OS), otherwise the default action is usually for the process to be terminated.

Pointers to Functions

- At the machine level a function is simply an address of a piece of code that does something.
- And we know already that a pointer is a type that stores the value of an address (*i.e.* it points to an address that holds some data, usually of a specific type)
- So it follows that we can create pointers to address, and this can be very useful, particularly for decoupling pieces of code and enabling run-time linkage (*i.e.* binding function calls to their addresses after compilation)
- The kernel uses such techniques for enabling demand-loadable modules, which you will soon learn about.

Mighty Macros

- So far we have seen how the preprocessor can be used to give labels to constant values.
- But the preprocessor is much more powerful than that, and is used extensibly in the writing of systems programs and operating systems, primarily to keep the code as flexible as possible.