25.3 Configuration In Xinu

Because it runs as an embedded system, Xinu follows a static configuration approach, with the majority of configuration occurring when the system is compiled and linked. Of course, even in some embedded systems, part of the configuration must be postponed until system startup. For example, some versions of Xinu calculate the size of memory during system initialization. Others use dynamic configuration to detect the presence of a real-time clock. As we have seen, some bus hardware chooses IRQs and device addresses when the bus is powered on. On such hardware, Xinu must wait until it runs to find the interrupt vector addresses and device CSR addresses.

To help manage configuration and to automate the selection of device driver modules, Xinu uses a separate configuration program. Named *config*, the program is not part of the operating system, and we do not need to examine the source code. Instead, we will look at how *config* operates: it takes an input file that contains specifications, and produces output files that become part of the operating system code. The next sections explain the configuration program and show examples.

25.4 Contents Of The Xinu Configuration File

The *config* program takes as input a text file named *Configuration*. It parses the input file, and generates two output files: *conf.h* and *conf.c*. We have already seen the output files, which contain defined constants for devices and a definition of the device switch table.†

The Xinu configuration file is a text file, divided into three sections. The sections are separated by lines that contain two percent characters (%%). The three sections are:

- Section 1: *Type declarations* for device types
- Section 2: *Device specifications* for specific devices
- Section 3: *Symbolic constants*

25.4.1 Section 1: Type Declarations

The motivation for a type declaration arises because a system may contain more than one copy of a particular hardware device. For example, a system may contain two UART devices that both use the tty abstraction. In such cases, a set of functions that comprise a tty driver must be specified for each UART. Entering the specification many times manually is error-prone, and can lead to inconsistencies. Thus, the type section allows the specification to be entered once and assigned a name that is used with both devices in the device specification section.

Each type declaration defines a name for the type, and lists a set of default device driver functions for the type. The declaration also allows one to specify the type of hardware with which the device is associated. For example, the type declaration:

†File *conf.h* can be found on page 267, and *conf.c* can be found on page 279.