Next Previous Contents

# 2. Using I/O ports in C programs

## 2.1 The normal method

Routines for accessing I/O ports are in `/usr/include/asm/io.h` (or `linux/include/asm-i386/io.h` in the kernel source distribution). The routines there are inline macros, so it is enough to `#include <asm/io.h>`; you do not need any additional libraries.

Because of a limitation in gcc (present in all versions I know of, including egcs), you *have to* compile any source code that uses these routines with optimisation turned on (`gcc -O1` or higher), or alternatively use `#define extern static` before you `#include <asm/io.h>` (remember to `#undef extern`afterwards).

For debugging, you can use `gcc -g -O` (at least with modern versions of gcc), though optimisation can sometimes make the debugger behave a bit strangely. If this bothers you, put the routines that use I/O port access in a separate source file and compile only that with optimisation turned on.

### Permissions

Before you access any ports, you must give your program permission to do so. This is done by calling the `ioperm()` function (declared in `unistd.h`, and defined in the kernel) somewhere near the start of your program (before any I/O port accesses). The syntax is `ioperm(from, num, turn_on)`, where `from` is the first port number to give access to, and `num` the number of consecutive ports to give access to. For example, `ioperm(0x300, 5, 1)` would give access to ports 0x300 through 0x304 (a total of 5 ports). The last argument is a Boolean value specifying whether to give access to the program to the ports (true (1)) or to remove access (false (0)). You can call `ioperm()` multiple times to enable multiple non-consecutive ports. See the `ioperm(2)` manual page for details on the syntax.

The `ioperm()` call requires your program to have root privileges; thus you need to either run it as the root user, or make it setuid root. You can drop the root privileges after you have called `ioperm()` to enable the ports you want to use. You are not required to explicitly drop your port access privileges with `ioperm(..., 0)` at the end of your program; this is done automatically as the process exits.

A `setuid()` to a non-root user does not disable the port access granted by `ioperm()`, but a `fork()` does (the child process does not get access, but the parent retains it).

`ioperm()` can only give access to ports 0x000 through 0x3ff; for higher ports, you need to use `iopl()` (which gives you access to all ports at once). Use the level argument 3 (i.e., `iopl(3)`) to give your program access to *all* I/O ports (so be careful --- accessing the wrong ports can do all sorts of nasty things to your computer). Again, you need root privileges to call `iopl()`. See the `iopl(2)` manual page for details.

### Accessing the ports

To input a byte (8 bits) from a port, call `inb(port)`, it returns the byte it got. To output a byte, call `outb(value, port)` (please note the order of the parameters). To input a word (16 bits) from ports `x` and `x+1` (one byte from each to form the word, using the assembler instruction `inw`), call `inw(x)`. To output a word to the two ports, use `outw(value, x)`. If you're unsure of which port instructions (byte or word) to use, you probably want `inb()` and `outb()` --- most devices are designed for bytewise port access. Note that all port access instructions take at least about a microsecond to execute.

The `inb_p()`, `outb_p()`, `inw_p()`, and `outw_p()` macros work otherwise identically to the ones above, but they do an additional short (about one microsecond) delay after the port access; you can make the delay about four microseconds with `#define REALLY_SLOW_IO` before you `#include <asm/io.h>`. These macros normally (unless you `#define SLOW_IO_BY_JUMPING`, which is probably less accurate) use a port output to port 0x80 for their delay, so you need to give access to port 0x80 with `ioperm()` first (outputs to port 0x80 should not affect any part of the system). For more versatile methods of delaying, read on.

There are manual pages for `ioperm(2)`, `iopl(2)`, and the above macros in reasonably recent releases of the Linux manual page collection.

## 2.2 An alternate method: `/dev/port`

Another way to access I/O ports is to `open()` `/dev/port` (a character device, major number 1, minor 4) for reading and/or writing (the stdio `f*()` functions have internal buffering, so avoid them). Then `lseek()` to the appropriate byte in the file (file position 0 = port 0x00, file position 1 = port 0x01, and so on), and `read()` or `write()` a byte or word from or to it.

Naturally, for this to work your program needs read/write access to `/dev/port`. This method is probably slower than the normal method above, but does not need compiler optimisation nor `ioperm()`. It doesn't need root access either, if you give a non-root user or group access to `/dev/port` --- but this is a very bad thing to do in terms of system security, since it is possible to hurt the system, perhaps even gain root access, by using `/dev/port` to access hard disks, network cards, etc. directly.

You cannot use `select(2)` or `poll(2)` to read /dev/port, because the hardware does not have a facility for notifying the CPU when a value in an input port changes.

Next Previous Contents