CS431 Operating Systems
Programming Assignment 6 - I/O Devices

1 Overview

A significant part of an operating system involves managing I/O devices. Part of the challenge in doing so is providing a standard and consistent programming interface to I/O devices despite the vast differences in their types and capabilities. In Unix systems, application programs can send commands and receive status information via device drivers by using the `ioctl()` (pronounced "I/O control") system call. This programming assignment will give practice using `ioctl()`.

2 Requirements

Write a program that uses `ioctl()` to first modify, then retrieve information about a virtual terminal device (tty). The terminal device is used in this assignment because it is the one device that is readily available on every Unix system and that can be accessed in a consistent way across POSIX-compliant operating systems. To use `ioctl()`, you must include the header:

```
#include <sys/ioctl.h>
```

2.1 Open the Device

The first step is to determine what the name of the terminal device is on your system. This will differ across Linux systems as well as on Mac OS and other Unix systems. To get the device name, at the terminal prompt enter the command:

```
tty
```

which will give an output such as `/dev/pts/0`, `/dev/ttys000`, etc. The output of the `tty` program is the name of the virtual terminal device we will work with.

In your program, you will open the device using `open()`, just as we've done with files. This will give us an `int` file descriptor. Open the device read-only. You can check to see if the device opened correctly and is in fact a virtual terminal by passing the file descriptor to the function `isatty()`, which returns 1 if the file descriptor references a tty device and 0 otherwise.

2.2 Read Device Attributes

For reading and modifying device attributes, use the `ioctl()` system call. The function is a variable parameter function, which means the number of parameters passed to it can differ depending on the type of device. For the virtual terminal, the function takes three arguments:

- The file descriptor returned after opening the tty device.
- A constant representing a request to the device
- A pointer to a data structure that will either be populated with device information as a result of the request or contains information needed to carry out the request

Of course, `ioctl()` is a very general function that can issue requests and pass information to a large number of different types of devices. For this assignment we need both the request information and data structure specific to a virtual terminal. One attribute of a tty we can easily obtain is the window size. To get window size, we pass the constant `TIOCGWINSZ`, the name of which should be reasonably intuitive. This constant is defined in the header file:

- `/usr/include/asm-generic/ioctls.h` (Ubuntu Linux)
- `/Libraries/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/ioctl.h` (Mac OS X)

The third argument is the data structure that will be populated with window size information. For this we use a struct named `winsize`, which is defined in the header file:

- `/usr/include/asm-generic/termios.h` (Ubuntu Linux)
- `/Libraries/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/ttycom.h` (Mac OS X)

The `winsize` struct contains the following fields:

- `ws_row`, which represents the number of rows in the terminal window
- `ws_col`, which represents the number of columns in the terminal window
- `ws_xpixel`, which represents the horizontal size (in pixels) of the terminal window
- `ws_ypixel`, which represents the vertical size (in pixels) of the terminal window

The last two attributes may or may not be used by default. For the third argument to `ioctl()`, declare a variable of the `winsize` struct and pass it **by reference**. After the call to `ioctl()` output the values of this struct.

2.3 Set Device Attributes

Setting attributes of the virtual terminal will work similarly to retrieving them, but the request argument will reflect setting versus getting the window size. For this part, change the attributes of the `winsize` struct and call `ioctl()` again. Afterwards, to check to see if the new attributes were correctly set, either clear the fields of the `winsize` struct or declare a new variable of that type. Then call `ioctl()` a **third** time, this time retrieving and outputting the attributes as described in section 2.2.

3 Extra Credit

Once you have a program that correctly reads and modifies the attributes of your virtual terminal, you can earn up to 10 percent extra credit on this assignment. Either add to the program or write a second program that uses `ioctl()` to retrieve attributes of a device other than the virtual terminal (hard drive might not be a bad choice, although it can be somewhat tricky on Mac OS X because of permissions). Also note that some devices might require root privilege to access, so your program might have to be run accordingly using `sudo`.

If you pursue extra credit, **include a screen shot of your program output**. Your extra credit code may not run correctly or even at all on the platform used for evaluation.

Deliverable. Submit the C code source file(s) and screen shot (if applicable) for your program through the WorldClass assignment dropbox no later than the due date specified in the weekly assignment.

Academic Integrity. I recognize that there are only so many ways to write code that meets the requirements of this assignment, and you may consult with other students in this class on the general approach to a solution. Having said that, sharing code, pseudocode, or other implementation details is not permitted, and I expect all work to be your own as evidenced by:

- Program structure, control flow, variable and function names (aside from system calls and C library functions, of course), etc. should be manifestly distinct from others' work.
- Your program should be commented in a manner sufficient for me to ascertain that you understand what the code you submit does.