ECE 222 System Programming Concepts
Lecture notes – Devices

In the following discussion, we use the term “device” to refer to a piece of hardware, or a peripheral, attached to a computer. We will expand on that definition as we go.

When we talked about streams, we mentioned that unix/linux treats devices like files. This makes sense for a lot of devices. For example, a keyboard produces a 1D stream of bytes, just like a file opened for reading. A printer accepts a 1D stream of bytes, just like a file opened for writing. The O/S can use the same principles (and code) to access devices that it can to access files.

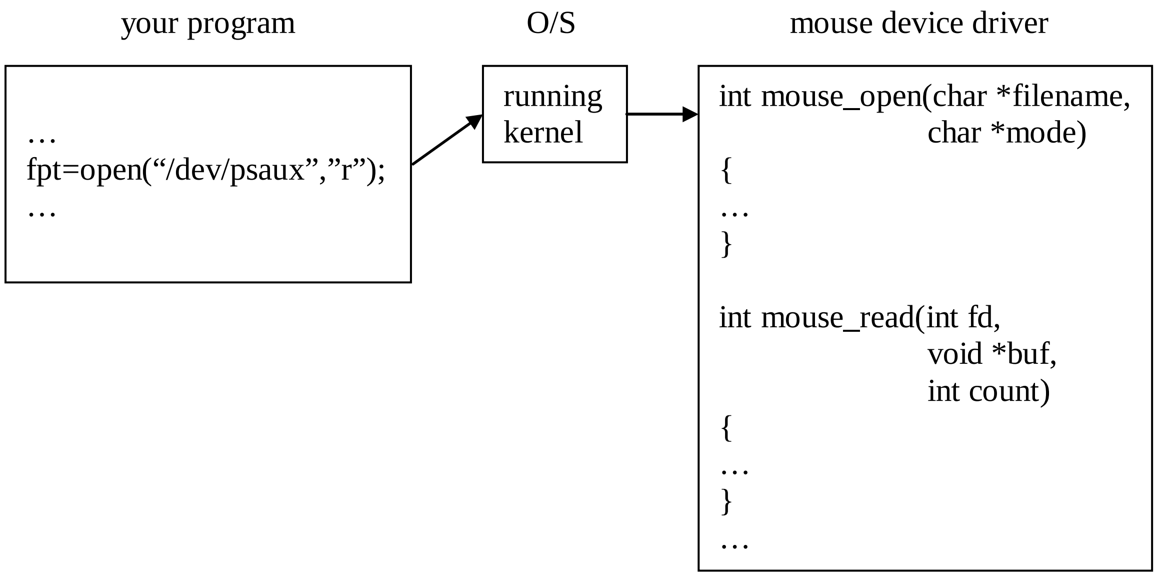
Three streams are opened automatically whenever a program is started: stdin, stdout, and stderr. But how are they opened? If devices are treated like files, where are the “filenames” that correspond to the devices?

/dev	the directory that contains filenames corresponding to devices

For example, the mouse is usually listed as /dev/psaux. We can “open” that file, and read bytes from it, observing what the mouse device is sending to the O/S.

[show example devmouse1.c]

These bytes are not meant for us to interpret. They are handled by a device driver. A device driver is a library of functions created for a specific device. It typically includes at least the following functions: open, close, read, and write. When we run devmouse1, and “open” the file /dev/psaux, we are actually calling a specific open() function in the device driver code for the mouse.



```
…
fpt=open(“/dev/psaux”,”r”);
…
```

```
int mouse_open(char *filename,
               char *mode)
{
…
}

int mouse_read(int fd,
               void *buf,
               int count)
{
…
}
…
```

There is a separate device driver, meaning a separate library of functions, for every device. For example, there is a separate open function for the printer, for the keyboard, and for the mouse. How does the O/S know which device driver (library) to use when we call open? The answer is major and minor device numbers. Associated with each filename that corresponds to a device is a major and minor number.

```
ahoover@login0> ls -al /dev/psaux
crw-------    1 root     root      10,   1 Jan 23 09:14 /dev/psaux
ahoover@login0>
```

Note that in place of the filesize, we see the numbers "10,1". These are the major and minor device numbers. The major number identifies the type of device, and therefore which device driver code to use when accessing that "file". The minor number identifies the specific instance of that device. It is possible to have multiple copies of the same device. Each would have the same major device number, but a different minor device number. This way, the O/S knows which device is being accessed.

Where is the code for a device driver? The kernel is a program, just like the ones you write. When compiled, the kernel can link to libraries statically or dynamically. Some device drivers are linked statically, meaning the code is in the running kernel. Other device drivers are linked dynamically, meaning the code is loaded from a file whenever the device is accessed. [Ask class: what type of code are we talking about? Answer: object code, aka machine code.]

Not all device files connect to physical pieces of hardware. Some hardware has multiple device files, each connecting to a different part of the hardware. For example, a hard drive can be "partitioned" into multiple separate storage areas. Each partition will have its own device file.

```
ls –al /dev/hda*
```

A "terminal" is a virtual device that embodies a keyboard/screen combination. Modern O/S's support multiple terminals being active on a single computer. It's like being able to have multiple computers, but only one real (physical) keyboard and monitor.

```
ls –al /dev/pts
```

We can run the "tty" command in a shell to determine which device filename corresponds to the terminal that our shell is running in. Once we know that, we can open the associated device filename and do interesting things.

[show devterm.c – open two terminals/shells, and run in opposite]

Almost all device drivers have the open, close, read and write functions in them. However, these functions do not always supply all the required access to a device. For example, suppose we want to change the settings on a printer, so that it prints sideways?

Or suppose we want to change the settings on a sound card, so that it plays louder? We need additional functions that allow us to do these things. One way to do this is to supply non-traditionally-named functions.

[show echostate.c example]

In this example, the tcgetattr() and tcsetattr() functions are part of the terminal device driver. The prototypes are defined in termios.h, where the template for a “struct termios” is also defined.

This example shows how each terminal has its own connection to the keyboard and monitor, so each terminal has its own stdin and stdout streams. In the example, we are changing how the stdin stream is handled by the device driver. We are changing the echo state. With echo on, stdin is copied to stdout. With echo off, it is not.

We can make similar changes to how the device driver handles stdin using a system program called ‘stty”:

```
stty erase 1
```

Now the “1” key becomes the erase key.

```
stty sane
```

This is a good command to know, to reset all terminal device driver settings to reasonable (sane) values.

Because a lot of devices need extra functions for access, there is a specially named function that works on most devices: ioctl, which stands for I/O control. This function will often be found in device drivers, implementing a cornucopia of actions.

[show screendim.c example]

In summary:
How is a device accessed? Through a device filename.
What is a device driver? A set of functions, called through the device filename.
How does the O/S know which device driver to use? Major and minor number.
What functions are inside a device driver? Open, read, write, close, ioctl, and perhaps others specific to the given device.
What sorts of things have device filenames associated with them? Hardware, subparts of hardware, virtual hardware.