/proc/bus/usb filesystem output

(version 2003.05.30)

The usbfs filesystem for USB devices is traditionally mounted at /proc/bus/usb. It provides the /proc/bus/usb/devices file, as well as the /proc/bus/usb/BBB/DDD files.

NOTE: If /proc/bus/usb appears empty, and a host controller driver has been linked, then you need to mount the filesystem. Issue the command (as root):

```
mount -t usbfs none /proc/bus/usb
```

An alternative and more permanent method would be to add

```
none  /proc/bus/usb  usbfs  defaults  0  0
```

to /etc/fstab. This will mount usbfs at each reboot. You can then issue `cat /proc/bus/usb/devices` to extract USB device information, and user mode drivers can use usbfs to interact with USB devices.

There are a number of mount options supported by usbfs. Consult the source code (linux/drivers/usb/core/inode.c) for information about those options.

NOTE: The filesystem has been renamed from "usbdevfs" to "usbfs", to reduce confusion with "devfs". You may still see references to the older "usbdevfs" name.

For more information on mounting the usbfs file system, see the "USB Device Filesystem" section of the USB Guide. The latest copy of the USB Guide can be found at http://www.linux-usb.org/

THE /proc/bus/usb/BBB/DDD FILES:

Each connected USB device has one file. The BBB indicates the bus number. The DDD indicates the device address on that bus. Both of these numbers are assigned sequentially, and can be reused, so you can't rely on them for stable access to devices. For example, it's relatively common for devices to re-enumerate while they are still connected (perhaps someone jostled their power supply, hub, or USB cable), so a device might be 002/027 when you first connect it and 002/048 sometime later.

These files can be read as binary data. The binary data consists of first the device descriptor, then the descriptors for each configuration of the device. Multi-byte fields in the device and configuration descriptors, but not other descriptors, are converted to host endianness by the kernel. This information is also shown in text form by the /proc/bus/usb/devices file, described later.

These files may also be used to write user-level drivers for the USB devices. You would open the /proc/bus/usb/BBB/DDD file read/write, read its descriptors to make sure it's the device you expect, and then bind to an interface (or perhaps several) using an ioctl call. You would issue more ioctls to the device to communicate to it using control, bulk, or other kinds of USB transfers. The IOCTLs are listed in the <linux/usbdevice_fs.h> file, and at this writing the source code (linux/drivers/usb/core/devio.c) is the primary reference for how to access devices through those files.

Note that since by default these BBB/DDD files are writable only by root, only root can write such user mode drivers. You can selectively grant read/write permissions to other users by using "chmod". Also, usbfs mount options such as "devmode=0666" may be helpful.

THE /proc/bus/usb/devices FILE:

In /proc/bus/usb/devices, each device's output has multiple lines of ASCII output.
I made it ASCII instead of binary on purpose, so that someone can obtain some useful data from it without the use of an auxiliary program. However, with an auxiliary program, the numbers in the first 4 columns of each "T:" line (topology info: Lev, Prnt, Port, Cnt) can be used to build a USB topology diagram.

Each line is tagged with a one-character ID for that line:

T = Topology (etc.)
B = Bandwidth (applies only to USB host controllers, which are virtualized as root hubs)
D = Device descriptor info.
P = Product ID info. (from Device descriptor, but they won't fit together on one line)
S = String descriptors.
C = Configuration descriptor info. (* = active configuration)
I = Interface descriptor info.
E = Endpoint descriptor info.

==========

/proc/bus/usb/devices output format:

Legend:
  d = decimal number (may have leading spaces or 0's)
  x = hexadecimal number (may have leading spaces or 0's)
  s = string

Topology info:

```
T:  Bus=dd Lev=dd Prnt=dd Port=dd Cnt=dd Dev#=ddd Spd=ddd MxCh=dd
|   |       |       |       |       |       |       |       |__MaxChildren
|   |       |       |       |       |       |       |__Device Speed in Mbps
|   |       |       |       |       |       |__DeviceNumber
```

```
|   |       |       |       |       |__Count of devices at this level
|   |       |       |       |__Connector/Port on Parent for this device
|   |       |       |__Parent DeviceNumber
|   |       |__Level in topology for this bus
|   |__Bus number
|__Topology info tag
```

Speed may be:

1.5	Mbit/s for low speed USB
12	Mbit/s for full speed USB
480	Mbit/s for high speed USB (added for USB 2.0)

Bandwidth info:

```
B:  Alloc=ddd/ddd us (xx%), #Int=ddd, #Iso=ddd
|   |                       |         |__Number of isochronous requests
|   |                       |__Number of interrupt requests
|   |__Total Bandwidth allocated to this bus
|__Bandwidth info tag
```

Bandwidth allocation is an approximation of how much of one frame (millisecond) is in use. It reflects only periodic transfers, which are the only transfers that reserve bandwidth. Control and bulk transfers use all other bandwidth, including reserved bandwidth that is not used for transfers (such as for short packets).

The percentage is how much of the "reserved" bandwidth is scheduled by those transfers. For a low or full speed bus (loosely, "USB 1.1"), 90% of the bus bandwidth is reserved. For a high speed bus (loosely, "USB 2.0") 80% is reserved.

Device descriptor info & Product ID info:

```
D:  Ver=x.xx Cls=xx(s) Sub=xx Prot=xx MxPS=dd #Cfgs=dd
P:  Vendor=xxxx ProdID=xxxx Rev=xx.xx
```

where

```
D:  Ver=x.xx Cls=xx(sssss) Sub=xx Prot=xx MxPS=dd #Cfgs=dd
|   |        |             |      |       |       |__NumberConfigurations
|   |        |             |      |       |__MaxPacketSize of Default Endpoint
|   |        |             |      |__DeviceProtocol
|   |        |             |__DeviceSubClass
|   |        |__DeviceClass
|   |__Device USB version
|__Device info tag #1
```

where

```
P:  Vendor=xxxx ProdID=xxxx Rev=xx.xx
|   |           |           |__Product revision number
|   |           |__Product ID code
|   |__Vendor ID code
|__Device info tag #2
```

String descriptor info:

```
S:  Manufacturer=ssss
|    |__Manufacturer of this device as read from the device.
|       For USB host controller drivers (virtual root hubs) this may
|       be omitted, or (for newer drivers) will identify the kernel
|       version and the driver which provides this hub emulation.
|__String info tag

S:  Product=ssss
|    |__Product description of this device as read from the device.
|       For older USB host controller drivers (virtual root hubs) this
|       indicates the driver; for newer ones, it's a product (and vendor)
|       description that often comes from the kernel's PCI ID database.
|__String info tag

S:  SerialNumber=ssss
|    |__Serial Number of this device as read from the device.
|       For USB host controller drivers (virtual root hubs) this is
|       some unique ID, normally a bus ID (address or slot name) that
|       can't be shared with any other device.
|__String info tag



Configuration descriptor info:

C:* #Ifs=dd Cfg#=dd Atr=xx MPwr=dddmA
|   |   |         |       |       |__MaxPower in mA
|   |   |         |       |__Attributes
|   |   |         |__ConfigurationNumber
|   |   |__NumberOfInterfaces
|   |__ "*" indicates the active configuration (others are " ")
|__Config info tag

    USB devices may have multiple configurations, each of which act
    rather differently.  For example, a bus-powered configuration
    might be much less capable than one that is self-powered.  Only
    one device configuration can be active at a time; most devices
    have only one configuration.

    Each configuration consists of one or more interfaces.  Each
    interface serves a distinct "function", which is typically bound
    to a different USB device driver.  One common example is a USB
    speaker with an audio interface for playback, and a HID interface
    for use with software volume control.


Interface descriptor info (can be multiple per Config):

I:* If#=dd Alt=dd #EPs=dd Cls=xx(sssss) Sub=xx Prot=xx Driver=ssss
|  |  |       |       |       |              |       |__Driver name
|  |  |       |       |       |              |          or "(none)"
|  |  |       |       |       |              |__InterfaceProtocol
|  |  |       |       |       |__InterfaceSubClass
|  |  |       |       |__InterfaceClass
|  |  |       |__NumberOfEndpoints
```

```
|  |  |      |__AlternateSettingNumber
|  |  |__InterfaceNumber
|  |__ "*" indicates the active altsetting (others are " ")
|__Interface info tag
```

A given interface may have one or more "alternate" settings.
For example, default settings may not use more than a small
amount of periodic bandwidth. To use significant fractions
of bus bandwidth, drivers must select a non-default altsetting.

Only one setting for an interface may be active at a time, and
only one driver may bind to an interface at a time. Most devices
have only one alternate setting per interface.

Endpoint descriptor info (can be multiple per Interface):

```
E:  Ad=xx(s) Atr=xx(ssss) MxPS=dddd Ivl=dddss
|    |         |             |         |__Interval (max) between transfers
|    |         |             |__EndpointMaxPacketSize
|    |         |__Attributes(EndpointType)
|    |__EndpointAddress(I=In,O=Out)
|__Endpoint info tag
```

The interval is nonzero for all periodic (interrupt or isochronous)
endpoints. For high speed endpoints the transfer interval may be
measured in microseconds rather than milliseconds.

For high speed periodic endpoints, the "MaxPacketSize" reflects
the per-microframe data transfer size. For "high bandwidth"
endpoints, that can reflect two or three packets (for up to
3KBytes every 125 usec) per endpoint.

With the Linux-USB stack, periodic bandwidth reservations use the
transfer intervals and sizes provided by URBs, which can be less
than those found in endpoint descriptor.

==========

If a user or script is interested only in Topology info, for
example, use something like "grep ^T: /proc/bus/usb/devices"
for only the Topology lines. A command like
"grep -i ^[tdp]: /proc/bus/usb/devices" can be used to list
only the lines that begin with the characters in square brackets,
where the valid characters are TDPCIE. With a slightly more able
script, it can display any selected lines (for example, only T, D,
and P lines) and change their output format. (The "procusb"
Perl script is the beginning of this idea. It will list only
selected lines [selected from TBDPSCIE] or "All" lines from
/proc/bus/usb/devices.)

The Topology lines can be used to generate a graphic/pictorial
of the USB devices on a system's root hub. (See more below
on how to do this.)

The Interface lines can be used to determine what driver is
being used for each device, and which altsetting it activated.

The Configuration lines could be used to list maximum power
(in milliamps) that a system's USB devices are using.
For example, "grep ^C: /proc/bus/usb/devices".

Here's an example, from a system which has a UHCI root hub,
an external hub connected to the root hub, and a mouse and
a serial converter connected to the external hub.

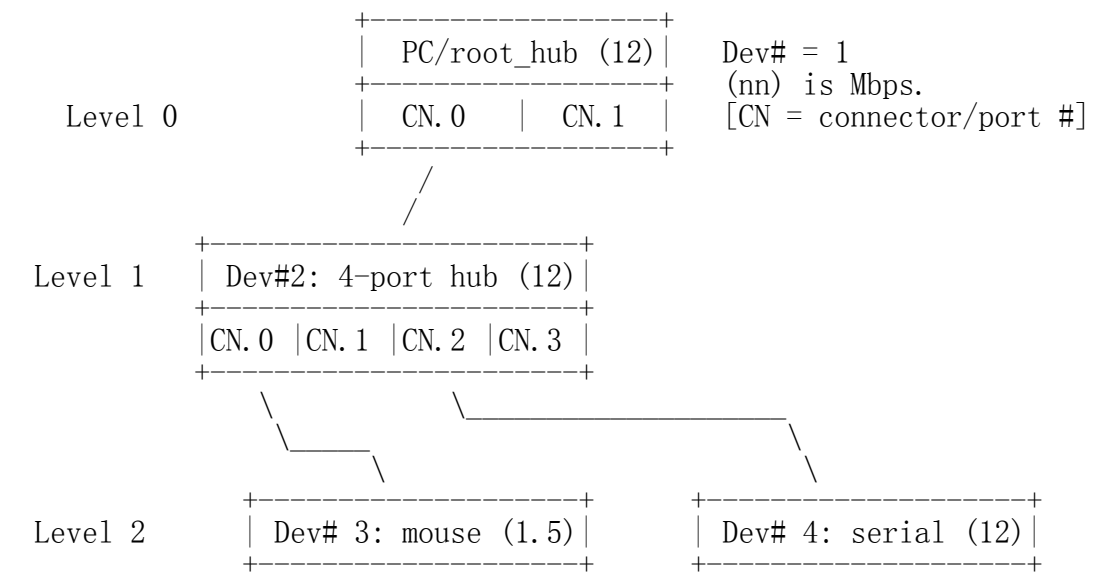
```
T:  Bus=00 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#=  1 Spd=12  MxCh= 2
B:  Alloc= 28/900 us ( 3%), #Int=  2, #Iso=  0
D:  Ver= 1.00 Cls=09(hub  ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=0000 ProdID=0000 Rev= 0.00
S:  Product=USB UHCI Root Hub
S:  SerialNumber=dce0
C:* #Ifs= 1 Cfg#= 1 Atr=40 MxPwr=  0mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub
E:  Ad=81(I) Atr=03(Int.) MxPS=   8 Ivl=255ms

T:  Bus=00 Lev=01 Prnt=01 Port=00 Cnt=01 Dev#=  2 Spd=12  MxCh= 4
D:  Ver= 1.00 Cls=09(hub  ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=0451 ProdID=1446 Rev= 1.00
C:* #Ifs= 1 Cfg#= 1 Atr=e0 MxPwr=100mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub
E:  Ad=81(I) Atr=03(Int.) MxPS=   1 Ivl=255ms

T:  Bus=00 Lev=02 Prnt=02 Port=00 Cnt=01 Dev#=  3 Spd=1.5 MxCh= 0
D:  Ver= 1.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=04b4 ProdID=0001 Rev= 0.00
C:* #Ifs= 1 Cfg#= 1 Atr=80 MxPwr=100mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=03(HID  ) Sub=01 Prot=02 Driver=mouse
E:  Ad=81(I) Atr=03(Int.) MxPS=   3 Ivl= 10ms

T:  Bus=00 Lev=02 Prnt=02 Port=02 Cnt=02 Dev#=  4 Spd=12  MxCh= 0
D:  Ver= 1.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=0565 ProdID=0001 Rev= 1.08
S:  Manufacturer=Peracom Networks, Inc.
S:  Product=Peracom USB to Serial Converter
C:* #Ifs= 1 Cfg#= 1 Atr=a0 MxPwr=100mA
I:  If#= 0 Alt= 0 #EPs= 3 Cls=00(>ifc ) Sub=00 Prot=00 Driver=serial
E:  Ad=81(I) Atr=02(Bulk) MxPS=  64 Ivl= 16ms
E:  Ad=01(O) Atr=02(Bulk) MxPS=  16 Ivl= 16ms
E:  Ad=82(I) Atr=03(Int.) MxPS=   8 Ivl=  8ms
```

Selecting only the "T:" and "I:" lines from this (for example, by using
"procusb ti"), we have:

```
T:  Bus=00 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#=  1 Spd=12  MxCh= 2
T:  Bus=00 Lev=01 Prnt=01 Port=00 Cnt=01 Dev#=  2 Spd=12  MxCh= 4
I:  If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub
T:  Bus=00 Lev=02 Prnt=02 Port=00 Cnt=01 Dev#=  3 Spd=1.5 MxCh= 0
```

```
I:  If#= 0 Alt= 0 #EPs= 1 Cls=03(HID  ) Sub=01 Prot=02 Driver=mouse
T:  Bus=00 Lev=02 Prnt=02 Port=02 Cnt=02 Dev#=  4 Spd=12  MxCh= 0
I:  If#= 0 Alt= 0 #EPs= 3 Cls=00(>ifc ) Sub=00 Prot=00 Driver=serial
```

Physically this looks like (or could be converted to):



Or, in a more tree-like structure (ports [Connectors] without connections could be omitted):

```
PC:  Dev# 1, root hub, 2 ports, 12 Mbps
|_ CN.0:  Dev# 2, hub, 4 ports, 12 Mbps
      |_ CN.0:  Dev #3, mouse, 1.5 Mbps
      |_ CN.1:
      |_ CN.2:  Dev #4, serial, 12 Mbps
      |_ CN.3:
|_ CN.1:
```

END