Authorizing (or not) your USB devices to connect to the system

This feature allows you to control if a USB device can be used (or not) in a system. This feature will allow you to implement a lock-down of USB devices, fully controlled by user space.

As of now, when a USB device is connected it is configured and its interfaces are immediately made available to the users. With this modification, only if root authorizes the device to be configured will then it be possible to use it.

Usage

Authorize a device to connect:

```
$ echo 1 > /sys/bus/usb/devices/DEVICE/authorized
```

De-authorize a device:

```
$ echo 0 > /sys/bus/usb/devices/DEVICE/authorized
```

Set new devices connected to hostX to be deauthorized by default (ie: lock down):

```
$ echo 0 > /sys/bus/usb/devices/usbX/authorized_default
```

Remove the lock down:

```
$ echo 1 > /sys/bus/usb/devices/usbX/authorized_default
```

By default, Wired USB devices are authorized by default to connect. Wireless USB hosts deauthorize by default all new connected devices (this is so because we need to do an authentication phase before authorizing). Writing "2" to the authorized_default attribute causes kernel to only authorize by default devices connected to internal USB ports.

Example system lockdown (lame)

Imagine you want to implement a lockdown so only devices of type XYZ can be connected (for example, it is a kiosk machine with a visible USB port):

```
boot up
rc.local ->

 for host in /sys/bus/usb/devices/usb*
 do
    echo 0 > $host/authorized_default
 done
```

Hookup an script to udev, for new USB devices:

```
if device_is_my_type $DEV
then
  echo 1 > $device_path/authorized
done
```

Now, device_is_my_type() is where the juice for a lockdown is. Just checking if the class, type and protocol match something is the worse security verification you can make (or the best, for someone willing to break it). If you need something secure, use crypto and Certificate Authentication or stuff like that. Something simple for an storage key could be:

```
function device_is_my_type()
{
  echo 1 > authorized          # temporarily authorize it
                               # FIXME: make sure none can mount it
  mount DEVICENODE /mntpoint
  sum=$(md5sum /mntpoint/.signature)
  if [ $sum = $(cat /etc/lockdown/keysum) ]
  then
       echo "We are good, connected"
       umount /mntpoint
       # Other stuff so others can use it
  else
       echo 0 > authorized
  fi
}
```

Of course, this is lame, you'd want to do a real certificate verification stuff with PKI, so you don't depend on a shared secret, etc, but

you get the idea. Anybody with access to a device gadget kit can fake descriptors and device info. Don't trust that. You are welcome.

Interface authorization

There is a similar approach to allow or deny specific USB interfaces. That allows to block only a subset of an USB device.

Authorize an interface:

```
$ echo 1 > /sys/bus/usb/devices/INTERFACE/authorized
```

Deauthorize an interface:

```
$ echo 0 > /sys/bus/usb/devices/INTERFACE/authorized
```

The default value for new interfaces on a particular USB bus can be changed, too.

Allow interfaces per default:

```
$ echo 1 > /sys/bus/usb/devices/usbX/interface_authorized_default
```

Deny interfaces per default:

```
$ echo 0 > /sys/bus/usb/devices/usbX/interface_authorized_default
```

Per default the interface_authorized_default bit is 1. So all interfaces would authorized per default.

Note:

> If a deauthorized interface will be authorized so the driver probing must be triggered manually by writing INTERFACE to /sys/bus/usb/drivers_probe

For drivers that need multiple interfaces all needed interfaces should be authorized first. After that the drivers should be probed. This avoids side effects.