How FunctionFS works

From kernel point of view it is just a composite function with some unique behaviour. It may be added to an USB configuration only after the user space driver has registered by writing descriptors and strings (the user space program has to provide the same information that kernel level composite functions provide when they are added to the configuration).

This in particular means that the composite initialisation functions may not be in init section (ie. may not use the __init tag).

From user space point of view it is a file system which when mounted provides an "ep0" file. User space driver need to write descriptors and strings to that file. It does not need to worry about endpoints, interfaces or strings numbers but simply provide descriptors such as if the function was the only one (endpoints and strings numbers starting from one and interface numbers starting from zero). The FunctionFS changes them as needed also handling situation when numbers differ in different configurations.

When descriptors and strings are written "ep#" files appear (one for each declared endpoint) which handle communication on a single endpoint. Again, FunctionFS takes care of the real numbers and changing of the configuration (which means that "ep1" file may be really mapped to (say) endpoint 3 (and when configuration changes to (say) endpoint 2)). "ep0" is used for receiving events and handling setup requests.

When all files are closed the function disables itself.

What I also want to mention is that the FunctionFS is designed in such a way that it is possible to mount it several times so in the end a gadget could use several FunctionFS functions. The idea is that each FunctionFS instance is identified by the device name used when mounting.

One can imagine a gadget that has an Ethernet, MTP and HID interfaces where the last two are implemented via FunctionFS. On user space level it would look like this:

```
$ insmod g_ffs.ko idVendor=<ID> iSerialNumber=<string> functions=mtp,hid
$ mkdir /dev/ffs-mtp && mount -t functionfs mtp /dev/ffs-mtp
$ ( cd /dev/ffs-mtp && mtp-daemon ) &
$ mkdir /dev/ffs-hid && mount -t functionfs hid /dev/ffs-hid
$ ( cd /dev/ffs-hid && hid-daemon ) &
```

On kernel level the gadget checks ffs_data->dev_name to identify whether it's FunctionFS designed for MTP ("mtp") or HID ("hid").

If no "functions" module parameters is supplied, the driver accepts just one function with any name.

When "functions" module parameter is supplied, only functions with listed names are accepted. In particular, if the "functions" parameter's value is just a one-element list, then the behaviour is similar to when there is no "functions" at all; however, only a function with the specified name is accepted.

The gadget is registered only after all the declared function filesystems have been mounted and USB descriptors of all functions have been written to their ep0's.

Conversely, the gadget is unregistered after the first USB function closes its endpoints.