sysfs : special virtual filesystem in Linux

- Used to boot Cortex M4 (remote processor) from Linux User Space.

Linux expects firmware for Cortex M4 to be located at **/lib/firmware/**

▶️

▶️ How will you start firmware

🔧 What if your firmware is stored elsewhere?

- You can tell Linux to look somewhere else, like **/lib/firmware/**
- **echo -n /lib/firmware/ > sys/module/firmware_class/parameters/path**

🔹 What Firmware Format is Supported?

- cat /sys/class/remoteproc/**remoteprocX**/fw_format
- Outputs either ELF (non-secure M4) / TEE (secure boot)

🔹 Telling Linux Which Firmware to Use

- By default, Linux looks for a file named : **rproc-nameOfFirm-fw**
- **echo -n your_m4_firmware.elf > /sys/class/remoteproc/remoteprocX/firmware**
- The **.elf** file should be in **/lib/firmware/** or the path you specified earlier

🔹 Booting the Remote Processor

- Actually start (boot) the M4 core.
- **echo start > /sys/class/remoteproc/remoteprocX/state**

1. **Allocates memory for the M4 firmware.**
2. **Loads the .elf file into the designated memory region.**
3. **Powers on and resets the M4.**
4. **Starts executing from the M4 firmware's entry point.**

🔹 Stopping the Remote Processor

- **echo stop > /sys/class/remoteproc/remoteprocX/state**

▶️

Extra 1:

🔍 How do you choose X in the remoteprocX ?

- X refers to the instance number of the remote processor, assigned by Linux kernel.
- Changes based on how many remote processors are registered during boot.

🔍 Step 1: List Available Remote Processors

- List available remote processors.
- **ls /sys/class/remoteproc/**
  - **remoteproc0 remoteproc1** // available remote processor instances

📄 Step 2: Check Which One is Your Cortex-M4

- Check below names for each processor to find which one corresponds to desired processor. (Look for output of A7/A9/M0 based on case)

- cat /sys/remoteproc/remoteprocX/name
- cat /sys/remoteproc/remoteproc0/name
- cat /sys/remoteproc/remoteproc1/name
    - m4@10000000 or stm32_m4 // possible outputs

▶️

Extra 2:

🔷 **Remote Processor 'Early' Boot**

📌 **What is it?**

- **Early boot is a process of starting the M4 firmware before Linux boots.**
- Typically done by the bootloader. (**U-Boot**)

🧠 **Why use this mode?**

- To run M4 tasks as early as possible (e.g., sensor init, motor control, secure key handling).
- Especially useful if your application has **tight real-time or startup deadlines**.
- Linux may disable unused peripherals during boot to save power.
- If Linux doesn't know M4 is already using those, it might **break M4** by shutting things down.
- So this tells Linux: **"Hey! M4 is running, leave its resources alone and attach instead."**

▶️

Extra 3:

🔷 **Automatic Attach on Linux Boot**

✅ **What does "auto attach" mean?**

- If the **M4 core** is already **running** (from **U-boot** or **early boot**), **Linux** can automatically detect and connect firmware during the booting phase.

🔷 **Manual Attach on Linux Boot**

✅ **What does "manual attach" mean?**

- If you booted the **M4 core** from **U-Boot** but **Linux didn't** automatically **load** the **firmware**, you need to attach it manually.

▶️

Extra 4:

🔍 **How do you display the state of the firmware?**

- The remoteproc firmware state can be monitored using following command:
- **cat /sys/class/remoteproc/remoteprocX/state**

📌 **What does this show?**

- Tells the current state of the remote processor (of Cortex-M4).
- **offline** : not running
- **running** : firmware is loaded and active
- **suspended** : paused state
- **crashed** : error occurred

▶️

Extra 5:

🔹 **Dynamic Debug for Kernel Logs**

- **This part is about enabling debug-level messages from the Linux kernel that are related to remoteproc.**

🔧 **Linux-side command**

- **echo -n 'file stm32_rproc.c +p' > /sys/kernel/debug/dynamic_debug/control**
- ***echo -n 'file remoteproc*.c +p' > /sys/kernel/debug/dynamic_debug/control**

📌 **What it does:**

- **stm32_rproc.c** is the STM32-specific remoteproc driver source file.
- **remoteproc*.c** refers to all remoteproc core files (**remoteproc_core.c**, etc.)
- **+p** tells the kernel to print all **pr_debug()** statements from those files.

✅ **Result:**

You'll now see detailed logs in the kernel ring buffer (dmesg) when:

- **Firmware is being loaded**
- **Resources are being parsed**
- **The processor is started or stopped**
- **Errors occur in the remoteproc lifecycle**

🔹 **Enable Trace Buffer from Firmware**

- **About your Cortex-M firmware logs. Different from kernel logs above.**
- **M4/M33 firmware can include a log buffer (like a printf()-style trace).**
- **A log buffer can be defined in the remoteproc firmware and declared in the resource table.**
- **If the feature is activated on the remote firmware, log traces can be dumped from the trace buffer using the following command**

🔧 **Linux-side command: (trace0** name was defined in resource table**)**

- **cat /sys/kernel/debug/remoteproc/remoteprocX/trace0**
- **cat :** concatenate **/** read and display content of a file .