Modify, rebuild and reload the Linux® kernel

Stable: 09.10.2019 - 17:03 / Revision: 09.10.2019 - 16:43

1 Overview

This stage explains how modify, rebuild and reload the Linux® kernel.
You will first be guided to install the Linux® kernel source code in the Developer Package directory. Then step by step you will execute procedures to modify, rebuild and reload the Linux® kernel.

2 Download the the Linux® kernel source code

- Download the STM32MP15-Ecosystem-v1.0.0 Developer Package Sources to the following directory: **$HOME/STM32MPU_workspace/STM32MP15-Ecosystem-v1.0.0/Developer-Package**
- Uncompress the tarball file to get the Linux® kernel tarball, the ST patches and the ST configuration fragments

```
PC $> cd $HOME/STM32MPU_workspace/STM32MP15-Ecosystem-v1.0.0/Developer-Package
PC $> tar xvf en.SOURCES-kernel-stm32mp1-openstlinux-4.19-thud-mp1-19-02-20.tar.xz
```

3 Prepare the Linux® kernel source code

- Extract the Linux® kernel source

```
PC $> cd stm32mp1-openstlinux-4.19-thud-mp1-19-02-20/sources/arm-openstlinux_weston-linux-
PC $> tar xvf linux-4.19.9.tar.xz
```

- Apply the ST patches

```
PC $> cd linux-4.19.9/
PC $> for p in `ls -1 ../*.patch`; do patch -p1 < $p; done
```

- Apply fragments

```
PC $> make multi_v7_defconfig fragment*.config
PC $> for f in `ls -1 ../fragment*.config`; do scripts/kconfig/merge_config.sh -m -r .conf
PC $> yes '' | make oldconfig
```

4 Build the Linux® kernel source code for the first time

The first time the kernel is build it could take several minutes.

- Build kernel images (uImage and vmlinux) and device tree (dtbs)

```
PC $> make uImage vmlinux dtbs LOADADDR=0xC2000040
```

- Build kernel module

```
PC $> make modules
```

- Generate output build artifacts

```
PC $> mkdir -p $PWD/install_artifact/
PC $> make INSTALL_MOD_PATH="$PWD/install_artifact" modules_install
```

5 Deploy the Linux® kernel on the board

5.1 Push the Linux® kernel into the board

```
PC $> scp arch/arm/boot/uImage root@<board ip address>:/boot
```

5.2 Push the devicetree into the board

```
PC $> scp arch/arm/boot/dts/stm32mp157*.dtb root@<board ip address>:/boot
```

5.3 Push the kernel modules into the board

- Remove the link created inside the **install_artifact/lib/modules/4.19.9** directory

```
PC $> rm install_artifact/lib/modules/4.19.9/build install_artifact/lib/modules/4.19.9/sou
```

- Optionally, strip kernel modules (to reduce the size of each kernel modules)

```
PC $> find install_artifact/ -name "*.ko" | xargs $STRIP --strip-debug --remove-section=.c
```

- Copy Kernel modules

```
PC $> scp -r install_artifact/lib/modules/* root@<ip of board>:/lib/modules
```

- Using the Linux console, re-generate the list of module dependencies (modules.dep) and the list of symbols provided by modules (modules.symbols)

```
Board $> /sbin/depmod -a
```

- Synchronize data on disk with memory

```
Board $> sync
```

5.4 Reboot the board

```
Board $> reboot
```

6 Modifying a built-in Linux kernel device driver

This simple example adds unconditional log information when the display driver is probed.

- Using the Linux console, check that there is no log information when the display driver is probed

```
Board $> dmesg | grep -i stm_drm_platform_probe
Board $>
```

- Go to the Linux® kernel source directory

```
PC $> cd $HOME/STM32MPU_workspace/STM32MP15-Ecosystem-v1.0.0/Developer-Package/stm32mp1-op
```

- Edit the *./drivers/gpu/drm/stm/drv.c* source file
- Add a log information in the *stm_drm_platform_probe* function as follow

```
static int stm_drm_platform_probe(struct platform_device *pdev)
{
        struct device *dev = &pdev->dev;
        struct drm_device *ddev;
        int ret;
        [...]

        DRM_INFO("Simple example - %s\n", __func__);

        return 0;
        [...]
}
```

- Save the file
- Rebuitd the Linux® kernel

```
PC $> make uImage LOADADDR=0xC2000040
```

- Update the Linux kernel image into board

```
PC $> scp arch/arm/boot/uImage root@<board ip address>:/boot
```

- Reboot the board

```
Board $> reboot
```

- Check that there is now log information when the display driver is probed

```
Board $> dmesg | grep -i stm_drm_platform_probe
[    2.764080] [drm] Simple example - stm_drm_platform_probe
```

Direct Rendering Manager