Cross Compilation of Linux Kernel and Device Driver for Raspberry Pi

Overview

This document outlines the steps taken to cross-compile the Linux kernel and a device driver for the Raspberry Pi from a host machine. The process includes compiling the kernel, generating necessary files, and transferring the compiled driver to the Raspberry Pi.

Steps

1. Setting Up the Host Environment

1. **Install Necessary Packages**: Ensure that you have the required packages for cross-compilation. Run the following command:

```
sudo apt install git build-essential gcc-aarch64-linux-gnu
```

2. Download the Raspberry Pi Kernel Source: Clone the kernel source code from the Raspberry Pi Git repository:

```
git clone --depth=1 https://github.com/raspberrypi/linux.git
/home/tselvan/Project/raspi/kernel/linux
```

2. Cross-Compiling the Kernel

1. **Navigate to the Kernel Source Directory**:

   ```
   cd /home/tselvan/Project/raspi/kernel/linux
   ```

2. **Configure the Kernel**: Use a default configuration suitable for your Raspberry Pi version:

```
make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- bcm2711_defconfig
```

3. **Compile the Kernel**: Start the kernel compilation process:

```
make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- -j$(nproc)
```

4. **Generate Required Files**: After the compilation is complete, you will have several files. The important ones include:

- **Image**: The compiled kernel image located at `arch/arm64/boot/Image`

- **Device Tree Blob (DTB)**: Files in the `arch/arm64/boot/dts/` directory, such as `bcm2711-rpi-4-b.dtb`
- **Modules**: Kernel modules located in `drivers/` and compiled to `.ko` files.

5. **Install Modules** (Optional): If you want to install modules in a specific directory for easy access:

```
make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu-
INSTALL_MOD_PATH=/path/to/destination modules_install
```

3. Cross-Compiling the Device Driver

1. **Navigate to Your Driver Directory**: Go to the directory containing your driver code:

   ```
   cd /home/tselvan/Project/raspi/matrix
   ```

2. **Create a Makefile**: Ensure you have a `Makefile` with the following content:

```
obj-m += led_matrix_drv.o

all:
	make -C /home/tselvan/Project/raspi/kernel/linux M=$(PWD) modules

clean:
	make -C /home/tselvan/Project/raspi/kernel/linux M=$(PWD) clean
```

3. **Compile the Driver**: Run the following command to compile the driver:

```
make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- modules
```

4. Transfer Compiled Files to Raspberry Pi

1. **Use SCP or USB Drive**: You can transfer the files using SCP or a USB drive. For example, using SCP:

```
scp arch/arm64/boot/Image pi@<raspberry_pi_ip>:/boot/
            scp arch/arm64/boot/dts/bcm2711-rpi-4-b.dtb
pi@<raspberry_pi_ip>:/boot/
scp *.ko pi@<raspberry_pi_ip>:~/
```

5. Install the Kernel on Raspberry Pi

1. **Log into Raspberry Pi**:

```
ssh pi@<raspberry_pi_ip>
```

2. **Install the Kernel**: Update the bootloader configuration to use the new kernel:

```
sudo cp /boot/Image /boot/kernel.img
sudo cp /boot/bcm2711-rpi-4-b.dtb /boot/
```

3. **Load the Driver**: Use `insmod` to insert the compiled module:

```
sudo insmod led_matrix_drv.ko
```

Issues Faced and Resolutions

Issue 1: Kernel Version Mismatch

Problem: Initially, there was a mismatch between the kernel version of the Raspberry Pi and the one being compiled on the host. The compiled module failed to load due to an "Invalid module format" error.

Resolution: Checked the kernel version on the Raspberry Pi using `uname -r` and ensured that the kernel source on the host matched this version. Recompiled the kernel and the driver after making sure of the correct configuration.

Issue 2: Missing Kernel Makefile

Problem: Encountered "No rule to make target 'modules'" error during the driver compilation.

Resolution: Verified the existence of the kernel source and its `Makefile`. Ensured the path to the kernel source was correct and that the necessary subdirectories and files were present.

Issue 3: Incompatible Pointer Type Error

Problem: During driver compilation, errors related to incompatible pointer types were encountered.

Resolution: Corrected the function signatures in the driver code to match the expected types, based on the latest kernel headers. This required adjusting the parameters passed to `class_create()` and ensuring correct data types were used.

Issue 4: Cross-Compiler Not Found

Problem: The cross-compiler was not found on the host system.

Resolution: Installed the necessary cross-compilation tools using the package manager to ensure the compiler was available.