Beaglebone Black Exercises

Training Material

All the training material is maintained at Github. Clone the repository with:

$ git clone https://github.com/embitude/training

Session-1 Exercises

Configure & Build the Kernel for BBB

1. Navigate to the OS directory of Builds
   $ cd <path_to_training_repo>/Builds/OS
2. Get the kernel source code
   $ wget https://mirrors.edge.kernel.org/pub/linux/kernel/v4.x/linux-4.19.103.tar.gz
3. Unpack the kernel source code
   $ tar -xvf linux-4.19.103.tar.gz
4. Get into the kernel source code
   $ cd linux-4.19.103/
5. Configure the kernel with already available configuration file for Beaglebone Black
   $ cp Builds/OS/Configs/config.4.19.103.default .config
6. Update the Kernel Makefile to cross compile for arm architecture
   Add following in Kernel Makefile (Search for ARCH and update as below)
   CROSS_COMPILE=arm-linux-gnueabihf-
   ARCH=arm
7. Finally, compile the kernel
   $ make zImage
8. Transfer the newly build kernel to the target board
   $ mount /dev/mmcblk0p1 /mnt (On board)
   $ scp arch/arm/boot/zImage root@<board_ip>:/mnt/ (On the system)
9. Unmount & Reboot the board to boot up with updated kernel
   $ umount /mnt (On board)
   $ reboot (On board)
10. Verify if the kernel is updated
    $ uname -a (Should show the latest kernel build time)

Write a simple Linux Kernel Module

1. Navigate to the FirstDriver directory under Exercises folder
   $ cd <path_to_training_repo>/Exercises/FirstDriver
2. Update the KERNEL_SOURCE variable in the Makefile to point to the compiled kernel source (Compiled in the above assignment)
3. Build the kernel module with make. This would generate the kernel module by name first_driver.ko
4. Transfer the kernel module to the board.
   $ scp first_driver.ko root@<board_ip>:
   The board_ip for bbb is 192.168.7.2
5. Load the kernel module
   $ insmod first_driver.ko
   This should display “driver registered” on the console

6. Unload the kenel module
   $ rmmod first_driver
   This should display “driver deregistered” on the console

Statically building the driver into the Kernel

This activity intends to describe the steps required to integrate any driver into the Kernel build system.

1. Copy the file *first_driver.c* from the Exercises/FirstDriver/ to Builds/OS/linux-4.19.103/drivers/char
   $ cp Exercises/FirstDriver/first_driver.c Builds/OS/linux-4.19.103/drivers/char
2. Edit the Kconfig file under linux-4.19.103/drivers/char and add the following:
   config MY_DRIVER
     tristate “My Driver”
     help
     Adding this small driver to the kernel
3. Edit the makefile under linux-4.19.103/drivers/char to add the following entry:
   obj-$(CONFIG_MY_DRIVER) += first_driver.o
4. Once the modifications are done, configure the kernel:
   $ make menuconfig
   Under the Driver Driver->Character Devices, there will be a menu option *MY Driver*. Just select it, exit and save the config.
5. This will add the *CONFIG_MY_DRIVER=y* entry in the .config, which in turn would be used by Makefile
6. Compile the kernel
   $ make zImage
7. Transfer the newly build kernel to the target board
   $ mount /dev/mmcblk0p1 /mnt (On board)
   $ scp arch/arm/boot/zImage root@<board_ip>:/mnt/
8. Unmount & reboot the board to boot up with updated kernel
   $ umount /mnt
   $ reboot (On board)
9. Verify if the driver is registered during the boot up
   $ dmesg | grep “driver registered”

Reference: https://www.linuxjournal.com/content/kbuild-linux-kernel-build-system

Session-2 Exercises

Character driver registration

The idea over here is to write a basic character driver to demonstrate the character driver registration and de-registration. Below are the steps:

1. Navigate to the CharDriver directory under Exercises folder
   $ cd Exercises/CharDriver
2. Make sure that the KERNEL_SOURCE variable in the Makefile is updated to point to the compiled kernel source
3. Complete all the TODOs (1 and 2) in first_driver.c (Refer apis.txt file in CharDriver for the corresponding APIs)
4. Build the kernel module with make. This would generate the kernel module by name first_char_driver.ko

5. Transfer the kernel module to the board.
   $ scp first_char_driver.ko root@<board_ip>:
   The board_ip for bbb is 192.168.7.2
6. Load the kernel module
   $ insmod first_char_driver.ko
7. This should display "<Major, Minor>: <240, 0>"
   NB: The major/minor number may vary in your case
8. Verify if the driver is registered for corresponding major number
   $ cat /proc/devices (first_char_driver should be displayed against Major number 240)

Register the file operations

The idea over is to register the call back handlers with VFS. Below are the steps:

1. Complete all the TODOs (1, 2 & 3) in null.c (Refer apis.txt file in CharDriver for the corresponding APIs)
2. Build the kernel module with make. This would generate the kernel module by name null.ko
3. Transfer the kernel module to the board.
   $ scp null.ko root@<board_ip>:
   The board_ip for bbb is 192.168.7.2
4. Load the kernel module
   $ insmod null.ko
5. This should display "<Major, Minor>: <240, 0>"
   NB: The major/minor number may vary in your case
6. Create the device file as per the major number allocated above
   $ mknod /dev/abc c 240 0
7. Test the write operation
   $ echo "Timmins" > /dev/abc
   (Should display the ascii value of the above string in user space)
8. Test the read operation
   $ cat /dev/abc

Enhancing the driver to send data to the user space

In the previous example, invocation of read had no effect except printing the messages from the my_open and my_read. Below are the steps to enhance read operation:

1. Complete TODO 4 in null.c (Refer apis.txt file in CharDriver for the corresponding APIs)
2. Build the kernel module with make. This would generate the kernel module by name null.ko
3. Transfer the kernel module to the board.
   $ scp null.ko root@<board_ip>:
   The board_ip for bbb is 192.168.7.2
4. Load the kernel module
   $ insmod null.ko
5. This should display "<Major, Minor>: <240, 0>"
   NB: The major/minor number may vary in your case
6. Create the device file as per the major number allocated above
   $ mknod /dev/abc c 240 0
7. Test the write operation

$ echo “Timmins” > /dev/abc
(Should display the ascii value of the above string)

8. Test the read operation
   $ cat /dev/abc (Should display ‘s’, which is last character of string provided during write operation)