

嵌入式系統學習筆記

環境建置

[安裝 GCC Toolchain for ARM](#)

[編譯 Linux Kernel](#)

[根檔案系統 rootfs](#)

[BusyBox](#)

[製作 rootfs](#)

[QEMU](#)

[建置](#)

[啟動](#)

[退出](#)

[Buildroot](#)

[版本: buildroot-2022.02.1.tar.gz](#)

[啟動](#)

[工具鍊 \(toolchain\) 路徑](#)

[設定 toolchain 路徑到環境變數](#)

[編譯命令參數](#)

[Linux header 路徑](#)

[PC](#)

[Embedded toolchain](#)

Kernel 模組

[新增模組](#)

[編譯 Module](#)

[複製模組到 rootfs](#)

[載入與移除模組](#)

實作 Character Driver

[註冊 character device](#)

[註銷 character device](#)

[file_operations](#)

[識別開啟的裝置](#)

[Read 操作](#)

[Write 操作](#)

[User space 測試程式](#)

[編譯 user space 測試程式](#)

[測試結果](#)

Reference

環境建置

安裝 GCC Toolchain for ARM

```
sudo apt-get install gcc-arm-linux-gnueabi
```

編譯 Linux Kernel

1. 下載位置 <https://www.kernel.org/> (下載版本: linux-5.15.37)
2. config 配置 (使用 vexpress 預設)
make CROSS_COMPILE=arm-linux-gnueabi- ARCH=arm vexpress_defconfig
3. make 編譯
make CROSS_COMPILE=arm-linux-gnueabi- ARCH=arm
4. Image 位置
arch/arm/boot/zImage

根檔案系統 rootfs

BusyBox

- 下載 busybox-1.35.0.tar.bz2
- 編譯 & 安裝

```
cd busybox-1.35.0.tar.bz2
make defconfig
make CROSS_COMPILE=arm-linux-gnueabi-
make install CROSS_COMPILE=arm-linux-gnueabi-
```

製作 rootfs

1. 創建根目錄rootfs

```
mkdir rootfs
```

2. 複製 busybox 命令到根目錄下, 注意 busybox 路徑

```
cp -r busybox-1.35.0/_install/* rootfs/
```

3. 從工具鏈中複製運行庫到lib目錄下

```
mkdir rootfs/lib
cp -P /usr/arm-linux-gnueabi/lib/* rootfs/lib
```

4. 創建 4 個 tty 終端設備 (c代表字符設備, 4是主設備號, 1 2 3 4分別是次設備號)

```
mkdir -p rootfs/dev
mknod rootfs/dev/tty1 c 4 1
mknod rootfs/dev/tty2 c 4 2
mknod rootfs/dev/tty3 c 4 3
mknod rootfs/dev/tty4 c 4 4
```

5. 創建 proc, etc... 等目錄

```
cd rootfs/  
mkdir -pv {bin,sbin,etc,proc,sys,usr/{bin,sbin}}
```

6. 生成 512M 大小的磁碟映象

```
qemu-img create -f raw disk.img 512M
```

7. 把磁碟映象格式化成ext4檔案系統

```
mkfs -t ext4 ./disk.img
```

8. 將文件複製到鏡像中

```
mkdir tmpfs  
sudo mount -o loop ./disk.img tmpfs/  
sudo cp -r rootfs/* tmpfs/  
sudo umount tmpfs
```

9. 建立 initial script

此步驟把 rootfs 全部打包進 initramfs-busybox-arm.cpio.gz, qemu 啟動時只要參數加上 -initrd initramfs-busybox-arm.cpio.gz 就不需要指定 -sd disk.img

```
cd rootfs  
vim init  
  
-----  
#!/bin/sh  
  
mount -t proc none /proc  
mount -t sysfs none /sys  
  
echo -e "\nBoot took $(cut -d' ' -f1 /proc/uptime) seconds\n"  
  
exec /bin/sh  
  
-----  
  
chmod +x init  
find . -print0 | cpio --null -ov --format=newc | gzip -9 > ./initramfs-busybox-arm.cpio.gz  
mv initramfs-busybox-arm.cpio.gz ../
```

QEMU

建置

To download and build QEMU 7.0.0:

```
// install essential tools  
sudo apt-get install git libglib2.0-dev libfdt-dev libpixman-1-dev zlib1g-dev  
// download source code  
wget https://download.qemu.org/qemu-7.0.0.tar.xz  
tar xvJf qemu-7.0.0.tar.xz  
cd qemu-7.0.0  
// config for arm/arm64  
./configure --target-list=arm-softmmu,aarch64-softmmu  
// build  
make -j2
```

```
// install to /usr/local/share/qemu  
make install
```

啟動

```
qemu-system-arm -M vexpress-a9 -m 512M -kernel zImage -dtb vexpress-v2p-ca9.dtb -nographic -append  
"root=/dev/mmcbk0 console=ttyAMA0 rw init=/linuxrc" -sd disk.img [-initrd initramfs-busybox-arm.cpio.gz]
```

轉存至 bash file

```
#!/bin/bash  
  
qemu-system-arm \  
-M vexpress-a9 \  
-m 512M \  
-kernel zImage \  
-dtb vexpress-v2p-ca9.dtb \  
-nographic \  
-append "root=/dev/mmcbk0 console=ttyAMA0 rw init=/linuxrc" \  
-sd disk.img \  
-initrd initramfs-busybox-arm.cpio.gz
```

“-sd disk.img”與“-initrd initramfs-busybox-arm.cpio.gz”兩者擇一即可，但使用“-sd disk.img”因為沒有 init 程式，啟動時會出現下列錯誤訊息：

```
can't run '/etc/init.d/rcS': No such file or directory
```

退出

```
ps -A | grep qemu-system-arm | awk '{print $1}' | xargs sudo kill -9
```

Buildroot

除了使用上述步驟建立自己的嵌入式系統模擬環境，也可以使用 Buildroot 快速建置。

版本 : buildroot-2022.02.1.tar.gz

```
cd buildroot  
make qemu_arm_vexpress_defconfig  
make -j2
```

啟動

```
qemu-system-arm -M vexpress-a9 -smp 1 -m 256 -kernel output/images/zImage -dtb  
output/images/vexpress-v2p-ca9.dtb -drive file=output/images/rootfs.ext2,if=sd,format=raw -append  
"console=ttyAMA0,115200 root=/dev/mmcbk0" -serial stdio -net nic,model=lan9118 -net user
```

工具鍊 (toolchain) 路徑

buildroot/output/host/bin

設定 toolchain 路徑到環境變數

```
export PATH=$PATH:/home/hank/Workspace/embedded/buildroot/output/host/bin
```

編譯命令參數

```
make ARCH=arm CROSS_COMPILE=arm-buildroot-linux-uclibcgnueabihf-
```

Linux header 路徑

PC

```
/usr/src/linux-headers-5.3.0-64
```

Embedded toolchain

```
buildroot/output/build/linux-headers-5.15.18
```

Kernel 模組

新增模組

vim hank.c

```
#include <linux/init.h>
#include <linux/module.h>

MODULE_DESCRIPTION("Hank");
MODULE_LICENSE("GPL");

static int hank_init(void) {
    printk(KERN_INFO "Hello Hank !\n");
    return 0;
}

static void hank_exit(void) {
    printk(KERN_INFO "ByeBye Hank !\n");
}

// Will be invoked when insmod
module_init(hank_init);
// Will be invoked when rmmod
module_exit(hank_exit);
```

vim Makefile

```
PWD := $(shell pwd)
# 指定 linux kernel source 路徑
KERNEL_DIR = /home/hank/Workspace/embedded/kernel/linux-5.15.37
# buildroot kernel source 路徑
# /home/hank/Workspace/embedded/buildroot/output/build/linux-5.15.18

MODULE_NAME = hank
obj-m := $(MODULE_NAME).o

all:
    make -C $(KERNEL_DIR) M=$(PWD) modules
```

```
# ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-
clean:
    make -C $(KERNEL_DIR) M=$(PWD) clean
```

編譯 Module

用下面指令編譯會得到 hank.ko 與其他中間檔。

```
# 使用 sudo apt-get install gcc-arm-linux-gnueabi 工具鍊
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-
# 使用 booldroot 工具鍊
make ARCH=arm CROSS_COMPILE=arm-buildroot-linux-uclibcgnueabihf-
```

複製模組到 rootfs

複製 hank.ko 到 rootfs 裡在重新啟動 qemu 即可在機器裡看到此模組。
複製方法可參考 “[製作 rootfs](#)” 小節的指令: [8. 將文件複製到鏡像中](#)”。

如果 qemu 使用 -initrd 啟動，則需要重新執行一次該節的指令 “[9. 建立 initial script](#)”，產生新的 initramfs-busybox-arm.cpio.gz，才會出現在重啟的 rootfs 裡。

載入與移除模組

載入

```
insmod <module_name>.ko
```

移除

```
rmmod <module_name>.ko
```

實作 Character Driver

註冊 character device

模組初始化函式中，調用 register_chrdev() 註冊字元設備，第一個參數為 major number，若帶 0 表示由 kernel 動態分配。
hankdev_fops 為 struct file_operations 的實體變數，負責定義 driver 支援的操作函式。
最後 device_create() 負責建立裝置並掛載到 /dev 底下。如下圖：

```
# ls -l /dev/hank0
crw----- 1 root    root      248,   0 May 10 02:34 /dev/hank0
```

實作程式碼：

```
static int __init hank_init(void) {
    printk(KERN_INFO "Init Hank Module !\n");

    // register device driver with dynamic major number assigned by kernel
    major_num = register_chrdev(0, DEVICE_NAME, &hankdev_fops);

    if (major_num < 0) {
        pr_alert("[Module][%d] Registering char device failed with %d\n", current->pid, major_num);
    }
}
```

```

        return major_num;
    }
    // A wrapper macro for printk (linux/printk.h)
    pr_info("[Module][%d] Assign major number %d to Hank device driver.\n", current->pid, major_num);
    cls = class_create(THIS_MODULE, DEVICE_NAME);

    // create single device
    device_create(cls, NULL, MKDEV(major_num, 0), NULL, "hank%d", 0);
    pr_info("[Module][%d] Devices created on /dev/hank%d\n", current->pid, 0);

    return 0;
}

```

註銷 character device

模組移除函式中，調用 `device_destroy()` 移除模組初始化時建立的裝置，調用 `unregister_chrdev()` 註銷字元設備。

```

static void __exit hank_exit(void) {
    printk(KERN_INFO "Exit Hank Module !\n");

    // destroy single device
    device_destroy(cls, MKDEV(major_num, 0));
    pr_info("[Module][%d] Devices /dev/hank%d destroyed\n", current->pid, 0);

    class_destroy(cls);

    // unregister character device driver
    unregister_chrdev(major_num, DEVICE_NAME);
}

```

file_operations

將 `file_operations` 結構體的函式指標指向對應的處理函式。

```

#include <linux/fs.h>

static struct file_operations hankdev_fops = {
    .owner = THIS_MODULE,
    .read = hank_read,
    .write = hank_write,
    .open = hank_open,
    .release = hank_release,
};

```

識別開啟的裝置

調用 `iminor(inode)` 可以得知裝置的編號，藉此識別裝置。

```

static int hank_open(struct inode *inode, struct file *file) {
    pr_info("[Module][%d] Hank device with minor %d was opened\n", current->pid, iminor(inode));
    return 0;
}

```

Read 操作

宣告一個靜態 buffer 作為裝置資料緩衝。

```
static char msg[BUF_LEN]; /* The msg the device will give when asked */
```

調用 copy_to_user() 將 msg 複製到 user pasce 的緩衝區，讓 user 得以讀取資料。

```
static ssize_t hank_read(struct file *file, char __user *buf /* buffer from user space */,
    size_t length, loff_t *offset) {
    size_t size = 0;
    int unread = 0;

    if (length < BUF_LEN) {
        size = length;
    } else {
        size = BUF_LEN;
    }

    unread = copy_to_user(buf, msg, size);
    return size - unread;
}
```

Write 操作

調用 copy_from_user() 將 user space 的資料複製到 driver 的緩衝區儲存起來。

```
static ssize_t hank_write(struct file *file, const char __user *buf /* buffer from user space */,
    size_t length, loff_t *offset) {
    int unwrite = 0;
    // clear device buffer
    memset(msg, 0x0, sizeof(msg));

    if (length > BUF_LEN) {
        length = BUF_LEN;
    }

    unwrite = copy_from_user(msg, buf, length);

    pr_info("[Module][%d] User write %s, unwrite: %d\n", current->pid, msg, unwrite);
    return length - unwrite;
}
```

User space 測試程式

調用 fopen() 開啟裝置，將裝置當作檔案來操作。fread() 從裝置讀取緩衝資料；fwrite() 將資料寫入裝置緩衝區。

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>

int main() {
    FILE *file = NULL;
    char buf[128];
```



```

int i = 0, sum = 0;

file = fopen("/dev/hank0", "w+");

if (file == NULL) {
    printf("[Test][%d] Failed to open /dev/hank0\n", getpid());
    return -1;
}

fread(buf, sizeof(buf), 1, file);
printf("[Test][%d] Read /dev/hank0: %s\n", getpid(), buf);

strncpy(buf, "Hello, Hank!\n", 12);
printf("[Test][%d] Write /dev/hank0: %s\n", getpid(), buf);
fwrite(buf, sizeof(buf), 1, file);

fread(buf, sizeof(buf), 1, file);
printf("[Test][%d] Read /dev/hank0: %s\n", getpid(), buf);

fclose(file);

return 0;
}

```

編譯 user space 測試程式

```
arm-buildroot-linux-uclibcgnueabi-hf-gcc -o test test.c
```

測試結果

```

Welcome to Buildroot
buildroot login: root
# cd /hank
# ls
hank.ko  test
# insmod hank.ko
hank: loading out-of-tree module taints kernel.
Init Hank Module !
[Module][124] Assign major number 248 to Hank device driver.
[Module][124] Devices created on /dev/hank0
# ls -l /dev/hank0
crw----- 1 root    root      248,   0 May 10 02:34 /dev/hank0
# echo "Hi, hank. How are you going?" > /dev/hank0
[Module][121] Hank device with minor 0 was opened
[Module][121] User write Hi, hank. How are you going?
, unwrite: 0
[Module][121] Hank device with minor 0 was released
# ./test
[Module][126] Hank device with minor 0 was opened
[Test][126] Read /dev/hank0: Hi, hank. How are you going?

[Test][126] Write /dev/hank0: Hello, Hank!w are you going?

[Module][126] User write Hello, Hank!w are you going?
, unwrite: 0
[Test][126] Read /dev/hank0: Hello, Hank!w are you going?

[Module][126] Hank device with minor 0 was released
# ls -l /dev/hank0
crw----- 1 root    root      248,   0 May 10 02:34 /dev/hank0
# rmmod hank.ko
Exit Hank Module !
[Module][128] Devices /dev/hank0 destroyed
#

```

插入模組

裝置已建立

用 echo 測試寫入裝置

執行測試程式

移除模組

Reference

1. [QEMU Documentation](#)
2. [QEMU Documentation/Platforms/ARM](#)
3. [Kernel Development on ARM](#)
4. [一步步教你: 如何用Qemu來模擬ARM系統](#)
5. [QEMU模擬vexpress-a9 搭建Linux kernel運行環境](#)
6. [使用qemu建立一個linux核心和busybox根檔案系統](#)
7. [利用 buildroot 與 Qemu 建構簡易 Embedded Linux 環境](#)
8. [QEMU搭建Linux實驗環境](#)
9. [根檔案系統製作詳解](#)
10. [歡迎來到 Linux 讀書會](#)
11. [Jason note](#)
12. [製作 ARM9 的 Bootstrap Root Filesystem](#)
13. [Yocto專案介紹及入門 -- 嵌入式工程師必備利器](#)
14. [Yocto Project Overview and Concepts Manual](#)
15. [寫一個簡單的 Linux Kernel Module](#)
16. [動手寫 Linux Driver](#)
17. [The Linux Kernel Module Programming Guide](#)
18. [Linux 核心模組運作原理](#)
- 19.