Крос-компілювання ядра Linux та модулів ядра на Ubuntu для Raspberry Pi

(Cross-Compiling Linux Kernel and Kernel Modules on Ubuntu for Raspberry Pi)

Вадим Мінзюк
vadym.minziuk@gmail.com

Версія документа: 2023-05-24

Host-система: Ubuntu (перевірено на версії 20.04.5)

Цільова ситема (target): Raspberry Pi (перевірено для Zero W та 3 Model B)

Table of Contents

Brief

Для наведеного нижче коду прийнято такі припущення:

Операційна система host-комп’ютера: Ubuntu 20

Ім’я користувача на host-комп’ютері: user

Цільова платформа: Raspberry Pi Zero W

Реліз ядра Linux на цільовій платформі: 5.15.84+

Ім’я користувача на цільовій платформі: user

Мережеве ім’я цільової платформи: zero

IP-адреса цільової платформи в локальній мережі: 192.168.1.142

Ім’я файлу з вихідним кодом модуля: my_module.c

Каталог з вихідним кодом модуля: /home/user/my_module

Каталог з вихідним кодом модуля містить такі файли:

- my_module.c
- Makefile
- checkpatch.pl

Модуль my_module створює в Proc_FS каталог “my_module”, а в цьому каталозі файл “status”.

```
/*Installing of build dependencies and toolchain*/
sudo apt install git bc bison flex libssl-dev make libc6-dev libncurses5-dev
sudo apt install crossbuild-essential-armhf
/*Downloading of kernel suorce code*/
cd /home/user
git clone --single-branch -b rpi-5.15.y https://github.com/raspberrypi/linux
cd ./linux
git checkout 3f4092766eaf -b rpi-5.15.84
/*defconfig для RPi Zero W*/
KERNEL=kernel
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcmrpi_defconfig
/*Kernel Cross-Compiling*/
make -j4 ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage modules dtbs
/*Module Cross-Compiling*/
export BUILD_KERNEL=/home/user/linux
cd ~/my_module
./checkpatch.pl --no-tree -f $( find . -name "*.c" -type f)
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- KDIR=${BUILD_KERNEL}
modinfo ./my_module.ko
/Sending my_module.ko to Raspberry Pi via SCP/
scp ~/my_module/my_module.ko user@192.168.1.142:/home/user/
/**********************/
/*Connecting to Raspberry Pi via SSH*/
ssh user@zero.local -p 22
/*or*/
ssh user@192.168.1.142 -p 22
/**********************/
/*If user “root” is not activated on Raspberry Pi*/
sudo passwd root
/**********************/
/*Copy module with root rights*/
su -l
cp -f /home/user/my_module.ko /lib/modules/my_module.ko
exit
```

```
/*Work with module*/
sudo dmesg -n 8
sudo insmod /lib/modules/my_module.ko
cat /proc/modules
lsmod
cat /var/log/kern.log
cat /proc/my_module/status
cat /var/log/kern.log
sudo rmmod  my_module
cat /var/log/kern.log
/*Shutdown Raspberry Pi remotely or “exit” instead*/
sudo shutdown -P 0
```

1 Крос-компіляція ядра (Kernel Cross-Compiling)

1.1 Встановлення додаткових інструментів та бібліотек (Tools and Build Dependencies Installing)

```
sudo apt install git bc bison flex libssl-dev make libc6-dev libncurses5-dev
```

1.2 Встановлення інструментарію крос-компілятора (Toolchain Installing)

Встановлення інструментарію крос-компілювання для 32-бітного цільового ядра

```
sudo apt install crossbuild-essential-armhf
```

Встановлення інструментарію крос-компілювання для 64-бітного цільового ядра

```
sudo apt install crossbuild-essential-arm64
```

1.3 Визначення номеру релізу ядра на цільовій платформі Raspberry Pi

Підключаємось до цільової машини Raspberry Pi за протоколом SSH командою:

```
ssh user@zero.local -p 22
```

В наведеному прикладі:

user – ім’я користувача на цільовій машині Raspberry Pi;

zero – ім’я цільової машини Raspberry Pi в мережі;

22 – номер порта SSH (22 по замовчуванню).

Якщо не вдасться підключитись, тоді потрібно повторно виконати цю команду, інакше в наведеній вище команді замінити “zero.local” на IP-адресу цільової машини Raspberry Pi в локальній мережі.

Далі керуємо віддалено цільовою машиною Raspberry Pi з host-комп’ютера через створене SSH-з’єднання.

Визначаємо реліз ядра Raspberry Pi OS цільової машини командою:

```
uname -r
```

Наприклад, одержали: **5.15.84+**

Вимикаємо цільову машину Raspberry Pi командою:

```
sudo shutdown -P 0
```

SHH-з’єднання розірветься автоматично. Подальші команди будуть виконуватися локально для host-комп’ютера.

1.4 Завантаження вихідного коду ядра (Getting the Kernel Sources)

На host-комп’ютері перейти в робочий каталог, наприклад, в домашній каталог користувача:

```
cd ~
```

Залежно від того, ядро якої версії нам потрібно, обираємо команду для завантаження з репозиторію https://github.com/raspberrypi/linux.

1.4.1 Завантаження найновішої версії без історії змін

```
git clone --depth=1 https://github.com/raspberrypi/linux
```

В робочому каталозі з’явиться каталог з іменем “linux”, який містить вихідний код ядра.

1.4.2 Завантаження одної із стабільних попередніх версії без історії змін

Якщо хочемо завантажити якусь попередню стабільну версію ядра, то дивимося її назву на https://github.com/raspberrypi/linux в закладці “Code” кнопка з випадаючим списком для перемикання гілок і тегів “Switch branches/tags” (рисунок 1).

Наприклад, нам потрібно rpi-5.15.y. Це версія 5.15.92 (фінальна версія в цій гілці). Завантажимо тільки цю гілку без її історії (фінальний коміт):

```
git clone --depth=1 --single-branch -b rpi-5.15.y https://github.com/raspberrypi/linux
```

В робочому каталозі з’явиться каталог з іменем “linux”, який містить вихідний код ядра.

Модулі, скомпільовані на основі цього ядра, працюватимуть в Raspberri Pi OS із версією ядра 5.15 (в тому числі 5.15 .84+).

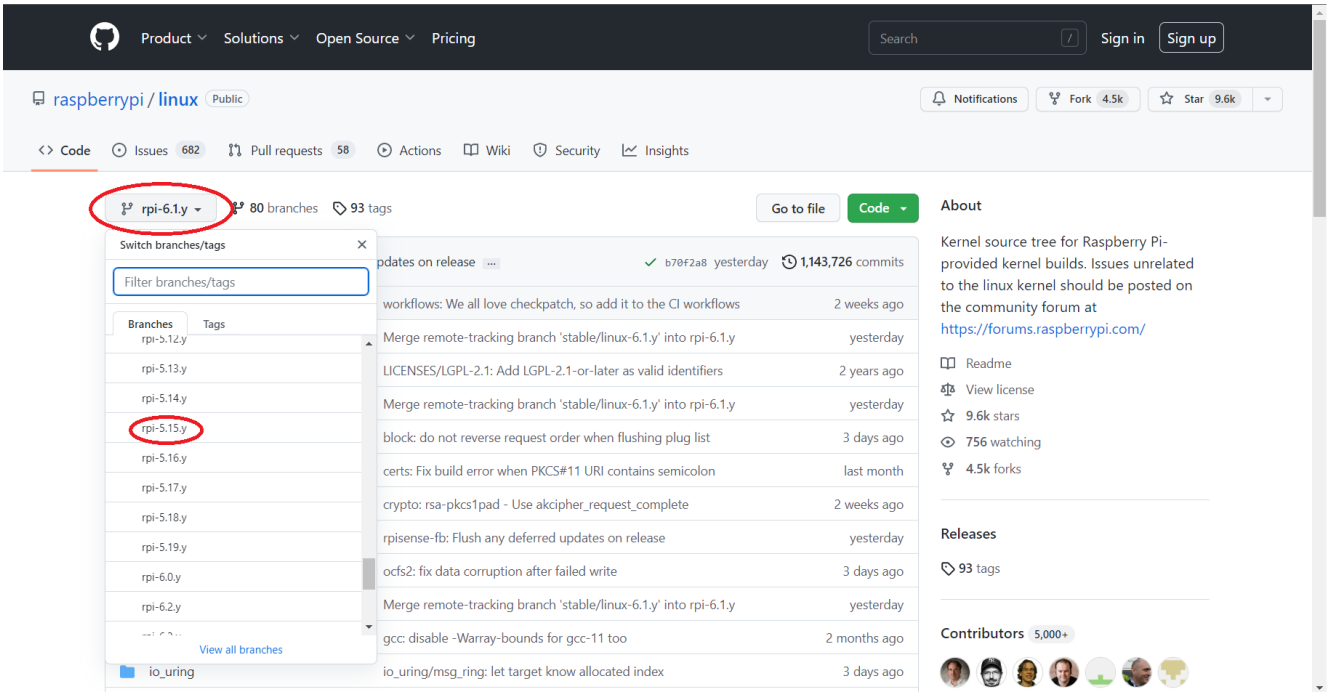

Рисунок 1

1.4.3 Завантаження версії ядра 5.15.84

Якщо є потреба компілювати модуль на основі саме 5.15.84, то завантажимо гілку rpi-5.15.y з її історією:

```
git clone --single-branch -b rpi-5.15.y https://github.com/raspberrypi/linux
```

Перейдемо в завантажений попередньою командою каталог з іменем “linux”.

```
cd ./linux
```

І виведемо список комітів:

```
git log --oneline
```

Шукаємо рядок з текстом “5.15.84”. І знаходимо:

```
3f4092766eaf Merge remote-tracking branch 'stable/linux-5.15.y' into rpi-5.15.y
d68f50bfb00f Linux 5.15.84
```

Але відгалужуємо гілку від наступного коміту, що на один рядок вище у виведеному списку, бо в ньому відбулося злиття стабільної гілки 5.15 із релізом 5.15.84. Тому в коміті 3f4092766eaf присутній файл bcmrpi_defconfig, якого не було в коміті d68f50bfb00f. Цей файл потрібний для конфігурування під Raspberry Pi Zero / Zero W.

```
git checkout 3f4092766eaf -b rpi-5.15.84
```

або переводимо каталог до стану цього коміту:

```
git checkout 3f4092766eaf
```

1.5 Конфігурування ядра

1.5.1 Застосування дефолтної конфігурації (Applying the Default Configuration)

Переходимо в каталог з вихідним кодом ядра (якщо не зробили цього на попередньому кроці). В нашому прикладі це:

cd ~/linux

Далі діємо залежно від цільової платформи.

1.5.1.1 defconfig для Raspberry Pi Series 1

Raspberry Pi 1

Raspberry Pi Zero

Raspberry Pi Zero W

Raspberry Pi Compute Module 1

Raspberry Pi Compute Module 3KERNEL=kernel
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcmrpi_defconfig

1.5.1.2 defconfig для 32-bit Raspberry Pi Series 2 and 3

Raspberry Pi 2

Raspberry Pi 3

Raspberry Pi 3+

Raspberry Pi Zero 2 W

Raspberry Pi Compute Module 3

Raspberry Pi Compute Module 3+

Raspberry Pi Compute Module 4KERNEL=kernel7
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcm2709_defconfig

1.5.1.3 defconfig для 32-bit Raspberry Pi Series 4

Raspberry Pi 4

Raspberry Pi 400

Raspberry Pi Compute Module 4

KERNEL=kernel7l
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcm2711_defconfig

1.5.1.4 defconfig для 64-bit Raspberry Pi

Raspberry Pi 3

Raspberry Pi 3+

Raspberry Pi 4

Raspberry Pi 400

Raspberry Pi Zero 2 W

Raspberry Pi Compute Module 3

Raspberry Pi Compute Module 3+

Raspberry Pi Compute Module 4

```
KERNEL=kernel8
make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- bcm2711_defconfig
```

1.5.2 Встановлення нового ідентифікатора версії ядра

Якщо ми модифікуємо ядро, версія нового ядра має відрізнятись від версії вихідного. Для цього необхідно змінити параметр LOCALVERSION у файлі .config, наприклад, так:

```
CONFIG_LOCALVERSION="-v7l-MY_CUSTOM_KERNEL"
```

Тепер ми можемо переконатись в тому, що запущено нове ядро, командою uname. А також бути впевнені в тому, що існуючі модулі в /lib/modules не перезаписані новими.

1.5.3 За потреби конфігуруємо ядро за допомогою Menuconfig

1.5.3.1 Menuconfig для 32-bit Raspberry Pi:

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
```

1.5.3.2 Menuconfig для 64-bit Raspberry Pi:

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

1.6 Компілювання ядра, модулів і дерева пристроїв (Building the kernel, modules, and Device Tree blobs)

Щоб скоротити час компілювання, в параметрі -j необхідно встановити максимально доступну кількість ядер процесора host-комп’ютера (чи потоків як віртуально доступних ядер). В наведеному прикладі встановлено -j4, оскільки використовувався двоядерний процесор із гіпертрейдингом.

1.6.1 Команда компіляції для 32-бітної цільової платформи:

```
make -j4 ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage modules dtbs
```

1.6.2 Команда компіляції для 64- бітної цільової платформи:

```
make -j4 ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- Image modules dtbs
```

У В А Г А ! Імена цільових образів різні для платформ 32 -bit і 64-bit.

2 Кроскомпіляція модулів (Modules Cross-Compiling)

На host-комп’ютері створюємо змінну оточення командного рядка BUILD_KERNEL та записуємо в неї шлях до каталога з вихідним кодом ядра. (В наведеному прикладі “user” — це ім’я поточного користувача на host-комп’ютері.)

```
export BUILD_KERNEL=/home/user/linux
```

Переходимо в каталог з вихідним кодом модуля my_module.c (там же має бути для нього Makefile і checkpatch.pl).

```
cd ~/my_module
```

Запускаємо checkpatch.pl

```
./checkpatch.pl --no-tree -f $( find . -name "*.c" -type f)
```

Компілюємо модуль.

Команда компіляції для 32-bit цільової платформи:

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- KDIR=${BUILD_KERNEL}
```

Команда компіляції для 64-bit цільової платформи:

```
make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- KDIR=${BUILD_KERNEL}
```

На host-комп’ютері переглянемо інформацію про скомпільований модуль:

```
modinfo ./my_module .ko
```

За протоколом SCP копіюємо файл модуля my_module.ko в домашній каталог цільової машини Raspberry Pi:

```
scp ~/my_module/my_module .ko user@192.168.1.142:/home/user/
```

В цьому прикладі:

user - ім’я користувача на цільовій машині;

192.168.1.142 - IP-адреса цільової машини в локальній мережі.

Підключаємось до цілювої машини Raspberry Pi за протоколом SSH командою:

```
ssh user@zero.local -p 22
```

або командою:

```
ssh user@192.168.1.142 -p 22
```

Далі керуємо віддалено цільовою машиною Raspberry Pi з host-комп’ютера через створене SSH-з’єднання.

Якщо на цільовій машині Raspberry Pi ще не активований користувач root, то активуємо командою:

```
sudo passwd root
```

На запит системи придумуємо і вводимо пароль для користувача root цільової машини Raspberry Pi.

Переходимо в контекст користувача root цільової машини Raspberry Pi командою:

```
su -
```

Копіюємо на цільовій машині Raspberry Pi файл my_module.ko з домашнього каталога користувача user (/home/user) у каталог /lib/modules

```
cp -f /home/user/my_module.ko /lib/modules/my_module.ko
```

Виходимо з контексту користувача root в контекст користувача user віддаленої машини командою:

```
exit
```

Щоб побачити інформаційні повідомлення всіх рівнів від модуля my_module.ko, підвищуємо current consol_loglevel до рівня 8 командою:

```
sudo dmesg -n 8
```

або з оточення root командою

```
echo 8 > /proc/sys/kernel/printk
```

Завантажити модуль командою:

```
sudo insmod /lib/modules/my_module.ko
```

або

```
sudo insmod /lib/modules/my_module.ko num1=2 num2=3
```

де num1 та num2 — параметри цього конкретного модуля.

Переглянути інформацію про запущені модулі можна командами:

```
cat /proc/modules
lsmod
```

Інформайійні повідомлення від модуля попадають в консоль tty1. Тому якщо запущена графічна оболонка чи віддалене керування, то повідомлення від pr_info можна побачити лише у файлі /var/log/kern.log. Тому для перегляду інформаційних повідомлень модуля необхідно переглянути файл kern.log:

```
cat /var/log/kern.log
```

Якщо модуль пише в Proc_FS, наприклад, у файл-буфер з ім’ям “status”, що в каталозі /proc/my_module/, вичитаємо цей буфер командою:

```
cat /proc/my_module/status
```

і подивимось, що він писав в kern.log в процесі вичитування буфера:

```
cat /var/log/kern.log
```

Вивантажуємо модуль командою:

```
sudo rmmod  my_module
```

Останні повідомлення переглядаємо у лог-файлі kern.log командою:

```
cat /var/log/kern.log
```

Віддалено вимикаємо Raspberry Pi командою:

```
sudo shutdown -P 0
```

чи завершуємо сеанс зв’язку командою:

```
exit
```

Посилання

1 How to compile Linux kernel for Raspberry Pi - Cross compilation. See:

https://www.raspberrypi.com/documentation/computers/linux_kernel.html

2 Репозиторій вихідного коду Raspberry Pi OS

https://github.com/raspberrypi/linux.