Embedded Linux

A Tour inside ARM's Kernel

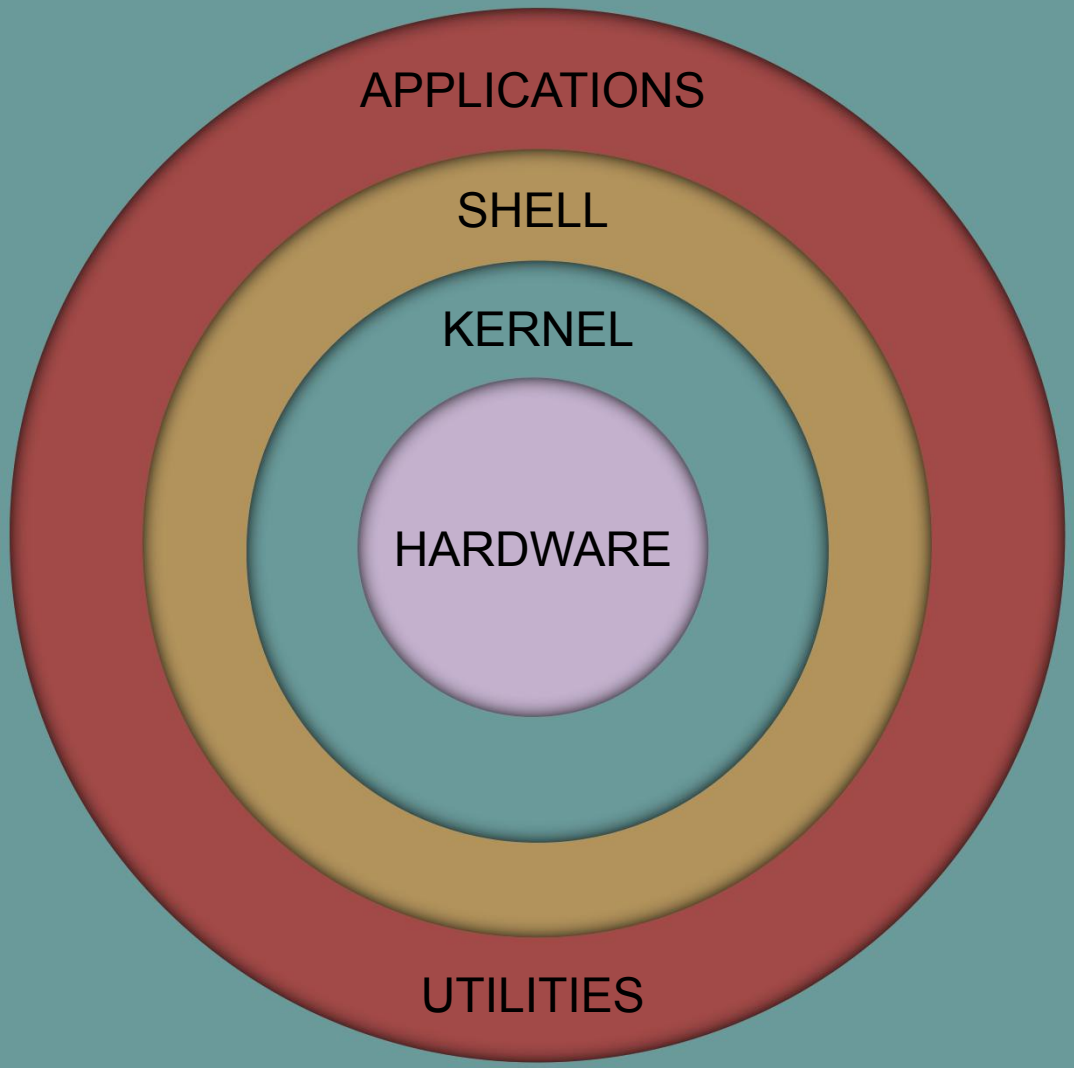
Contents

1. Shell basics
2. Introduction to Embedded Linux
3. Kernel Programming for Module / Driver Installation
4. Module / Device Driver in RPi
5. Cross Compiling & Tool Chain

Shell

- A Shell provides you with an interface to the Unix system.
- It gathers input from you and executes programs based on that input.
- When a program finishes executing, it displays that program's output.
- Types of Shell
    1. *BASH*
    2. *SH*
    3. *CSH*
    4. *KSH*

Shell in Sys Architecture

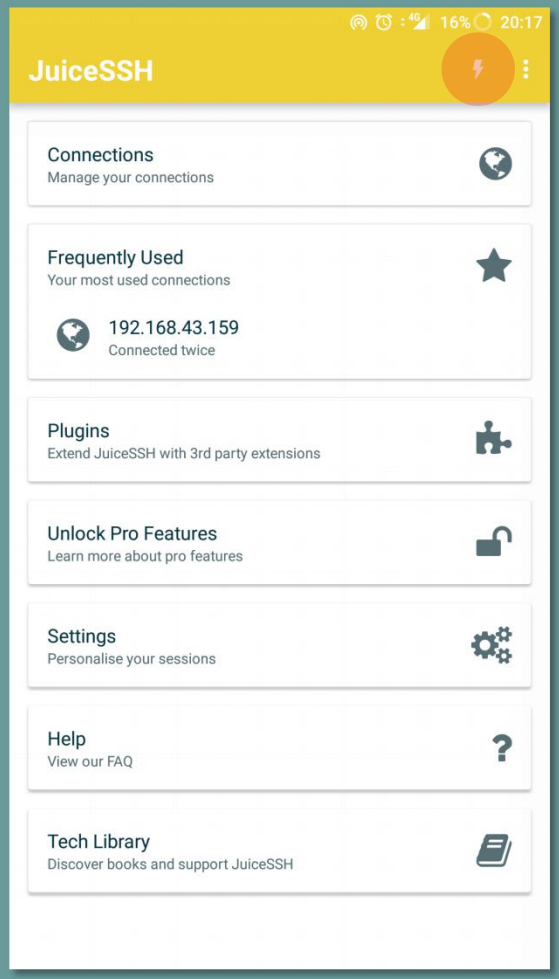


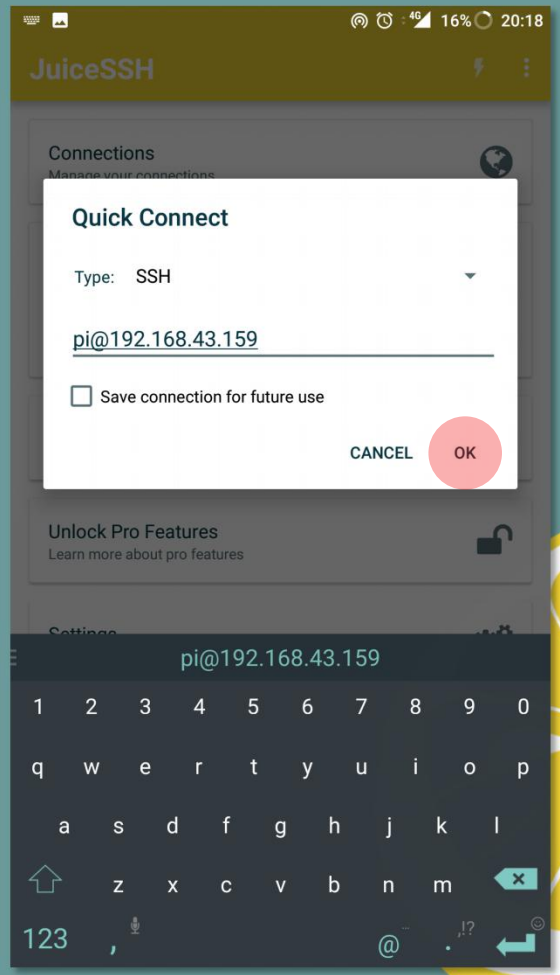
Shell basics

- To start with basics, try the command
  - `echo Hello World`
  - `a=1`
  - `b=2`
  - `` `expr $a + $b` ``

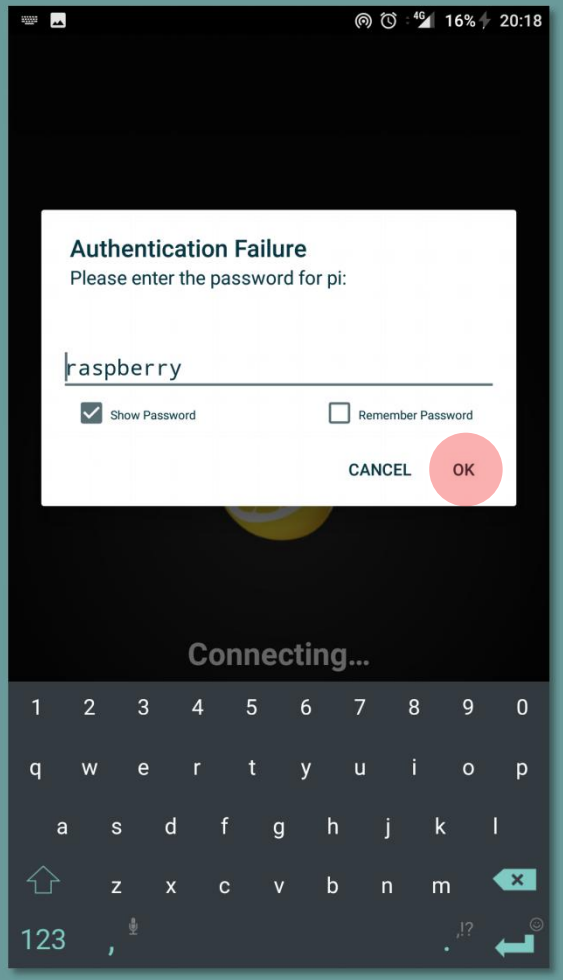
But how do we try this?

Let's have The Juice SSH





Authentication



Embedded Linux

- A Linux Kernel specially designed for ARMx processors can be said as Embedded Linux
- So far, maximum hardware for development; run such Linux based environment only
  - Raspberry Pi
  - BeagleBone
  - Banana Pi

    and many more
- These usually include a ported Linux kernel with cross-development tools, and sometimes with real time extensions

Why Embedded Linux?

- Royalty-free
- Strong networking support
- Has already been ported to many different CPU architectures
- Relatively small for its feature set
- Easy to configure
- Huge application base
- Modern OS (eg. memory management, kernel modules, etc.)

Embedded Linux OS

UBUNTU MATE

SNAPPY UBUNTU CORE

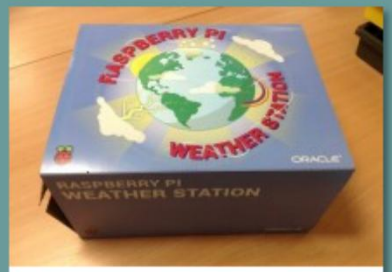
WINDOWS 10 IOT CORE

OSMC

LIBREELEC

PINET

RISC OS



WEATHER STATION

Kernel Programming

- Kernel modules are piece of code, that can be loaded and unloaded from kernel on demand.
- Kernel modules offers an easy way to extend the functionality of the base kernel without having to rebuild or recompile the kernel again.
- Most of the **drivers** are implemented as a **Linux kernel modules**.
- When those drivers are not needed, we can unload only that specific driver, which will reduce the kernel image size.
- The kernel modules will have a **.ko** extension.
- On a normal linux system, the kernel modules will reside inside **/lib/modules/<kernel_version>/kernel/** directory.

Module / DD in RPi 3

- PRE-REQUISITES
  - sudo apt-get install linux-source
  - sudo apt-get install git bc
  - sudo wget https://raw.githubusercontent.com/notro/rpi-source/master/rpi-source -O /usr/bin/rpi-source && sudo chmod +x /usr/bin/rpi-source && /usr/bin/rpi-source -q --tag-update
  - git clone --depth=1 https://github.com/raspberrypi/linux

Module / DD in RPi 3

PREPING THE MODULE

- Change your directory to cloned Linux
  - cd linux
- Create a directory of your Module
  - mkdir hello
- Create 2 files inside it using any editor

Module / DD in RPi 3

- Writing a Device Driver / Module

```
 1 #include <linux/module.h>     // included for all kernel modules
 2 #include <linux/kernel.h>     // included for KERN_INFO
 3 #include <linux/init.h>       // included for __init and __exit macros
 4
 5 MODULE_LICENSE("GPL");
 6 MODULE_AUTHOR("Mister T");
 7 MODULE_DESCRIPTION("This Module is Cool Bro!");
 8
 9 static int __init hello_init(void)
10 {
11     printk(KERN_INFO "Hey Man! I'm inside your Kernel now!\n");
12     return 0;    // Non-zero return means that the module couldn't be loaded.
13 }
14
15 static void __exit hello_cleanup(void)
16 {
17     printk(KERN_INFO "Adios homey!\n");
18 }
19
20 module_init(hello_init);
21 module_exit(hello_cleanup);
```

Module / DD in RPi 3

- Writing Makefile for creating targets

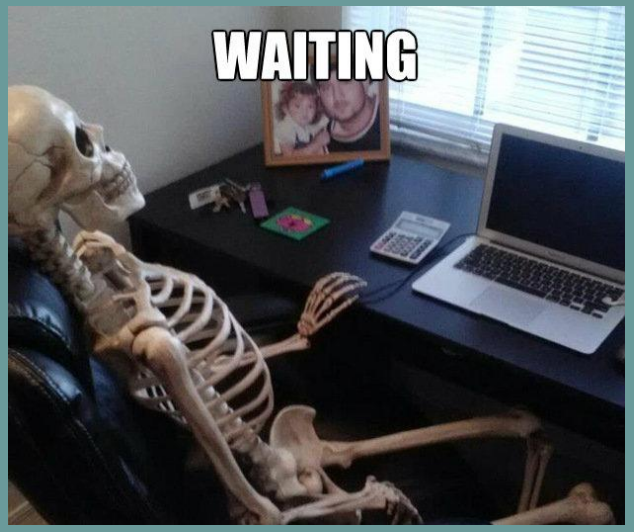
```
1 obj-m += hello.o
2 all:
3         make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
4 clean:
5         make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Module / DD in RPi 3

COMPILATION

- Start with Building a configuration
  - cd linux
  - KERNEL=kernel7
  - make bcm2709_defconfig

  - make -**j4** zImage modules dtbs
  - sudo make modules_install



CROSS-COMPILING

- First, you will need a suitable Linux cross-compilation host.
- Ubuntu is preferred; since Raspbian is also a Debian distribution, it means many aspects are similar, such as the command lines.
- You can either do this using VirtualBox (or VMWare) on Windows, or install it directly onto your computer.

Wikihow

TOOL CHAIN

- **`git clone https://github.com/raspberrypi/tools`**
  - Will Help you get the tools of RPi into Home folder

- **`echo PATH=\$PATH:~/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin >> ~/.bashrc`**
- **`source ~/.bashrc`**
  - Updating the $PATH environment variable makes the system aware of file locations needed for cross-compilation.

FIN

@mitu_skillologies

/mITuSkillologies

@mitu_group

mitu.co.in

/company/mitu-skillologies