Simona Toaca

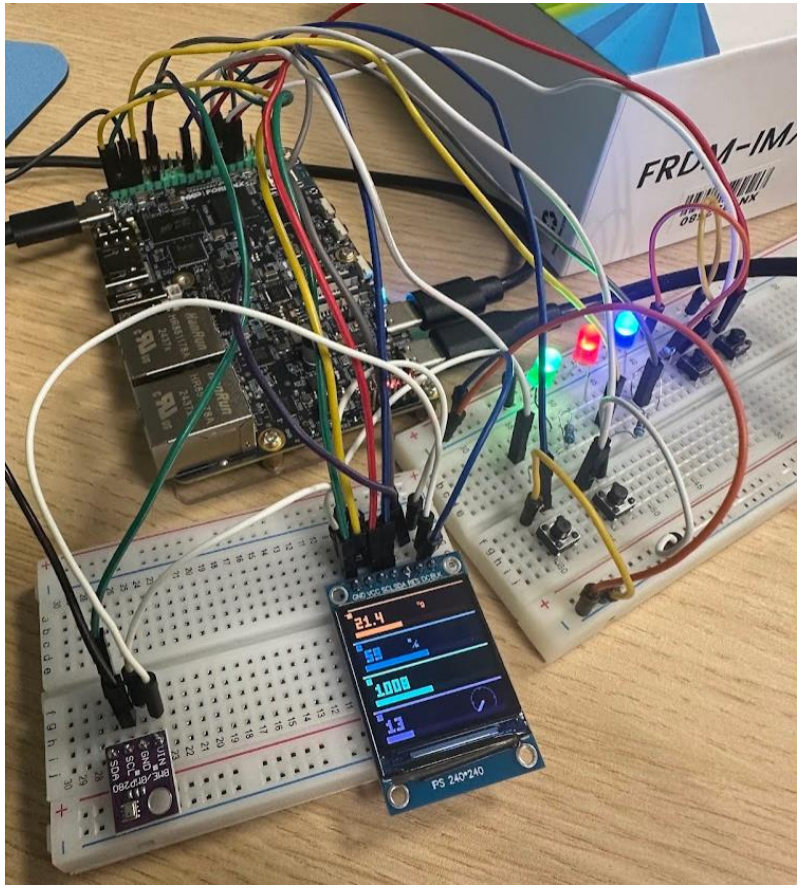
Laurentiu Mihalcea

Iuliana Prodan

Daniel Baluta

What is all about?

- Series of hands-on exercises and courses to learn Linux kernel development
- Hardware platform: NXP i.MX93 FRDM board
- Schedule
  - Day 1: Introduction to the Linux kernel
  - Day 2: Character devices, GPIOs and IRQs
  - Day 3: SPI bus and ST7789 display
  - Day 4: I2C bus and BMP280 sensor
  - Day 5: HACKATHON

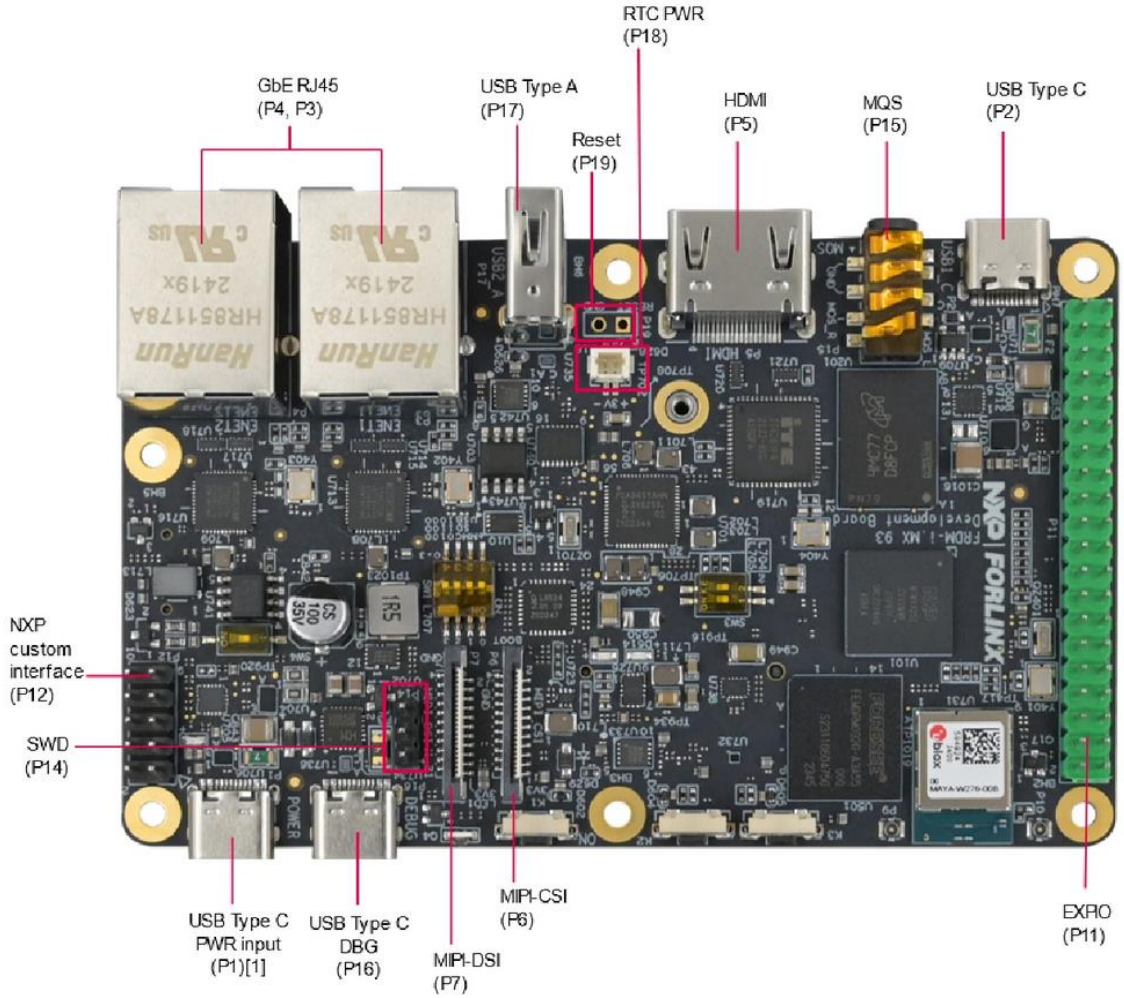


00

Day 1: Introduction to the Linux kernel*

- With a focus on embedded systems

Hardware support - NXP i.MX93 FRDM board

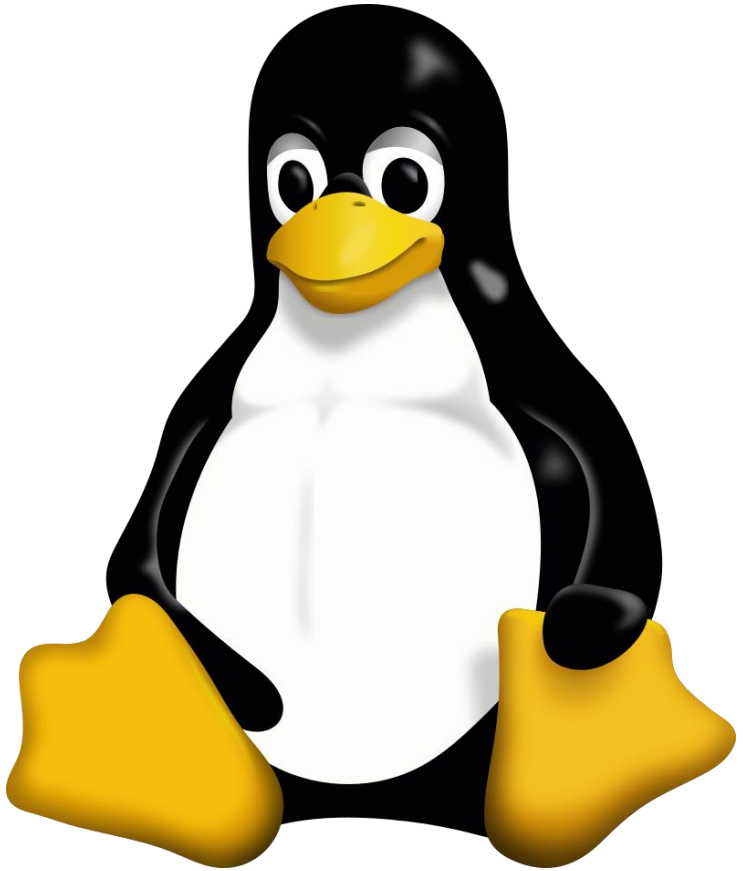


- SoC: NXP i.MX93 FRDM board
  - CPUs: 2× ARM Cortex-A55 @1.7Ghz
  - CPUs: 1 x ARM Cortex-M33 @250M
  - Memory: Up to 2G LPPDR4
  - Storage: 32G eMMC
  - **I/O**: USB, UART, GPIO**, I2C, SPI**, CAN, Ethernet, MIPI-CSI
  - Audio: MQS

Notes and sources section

Embedded Linux vs Desktop Linux

- Purpose and Use cases
  - General purpose vs Specialized
  - Ubuntu, Fedora, Debian vs Yocto, buildroot, openwrt
- Hardware requirements
  - Power consumption, memory footprint
- Operating system design
  - Full fledged OS vs stripped down version of Linux
- System on a Chip vs Discrete Component System



Notes and sources section

Embedded Linux usage

- Consumer electronics
  - Smart TVs and Set-Top boxes
  - Smartphones and Tables
- Wearables
  - Smartwatches and fitness trackers
- Automotive Systems
  - In-Vehicle Infotainment (IVI)
  - Advanced Driver Assistance Systems (ADAS)
- Internat of Things (IoT)
  - Smarthome devices
- Industrial Automation, Medical Devices, Energy and Utilities

Linux kernel

- Started by Linus Torvalds, in 1991
- Split into sub-subsystems handled by maintainers
- https://kernel.org/
- Development
  - Current mainline version: **7.2**
  - Releases every 9-10 weeks
  - Long Term Support
- Stats:
  - Version 7.1 has around 42M lines of code
  - Every release they are around 2000 contributors

Protocol	Location
HTTP	https://www.kernel.org/pub/
GIT	https://git.kernel.org/
RSYNC	rsync://rsync.kernel.org/pub/

mainline:	7.2-rc1	2026-06-28	[tarball]		[patch]		[view diff]	[browse]
stable:	7.1.2	2026-06-27	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
stable:	7.0.14 [EOL]	2026-06-27	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
longterm:	6.18.37	2026-06-27	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
longterm:	6.12.94	2026-06-19	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
longterm:	6.6.143	2026-06-19	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
longterm:	6.1.176	2026-06-19	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
longterm:	5.15.210	2026-06-19	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
longterm:	5.10.259	2026-06-19	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
linux-next:	next-20260626	2026-06-26						[browse]

Notes and sources section

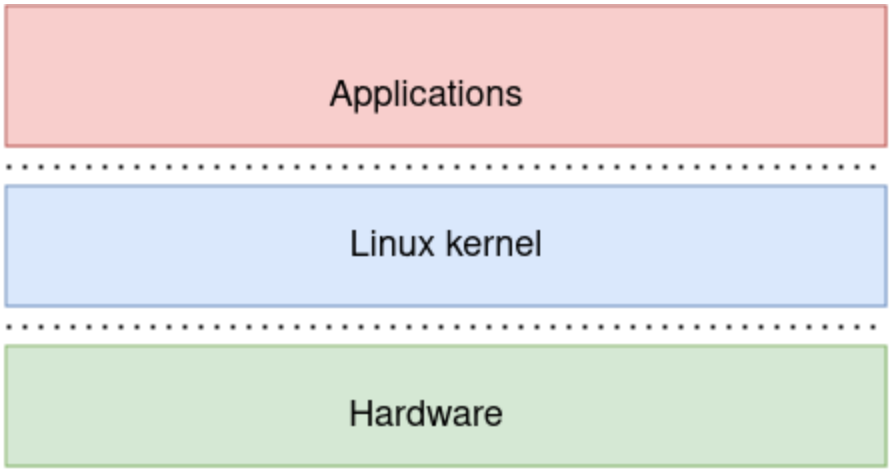
Linux kernel contributors

Most active 7.1 employers

By changesets			By lines changed		
(Unknown)	2160	13.6%	Meta	156583	15.8%
Intel	1435	9.1%	AMD	136736	13.8%
Google	1227	7.7%	(Unknown)	76738	7.7%
AMD	828	5.2%	Qualcomm	75847	7.6%
Qualcomm	787	5.0%	Samsung	55391	5.6%
Red Hat	745	4.7%	Google	54537	5.5%
(None)	694	4.4%	Intel	48917	4.9%
NVIDIA	595	3.8%	NVIDIA	37374	3.8%
Meta	535	3.4%	Red Hat	35706	3.6%
(Consultant)	500	3.2%	(None)	35178	3.5%
SUSE	459	2.9%	Oracle	15837	1.6%
Oracle	390	2.5%	SerNet	14403	1.5%
Arm	334	2.1%	Arm	13030	1.3%
Renesas Electronics	289	1.8%	SUSE	12208	1.2%
NXP Semiconductors	254	1.6%	NXP Semiconductors	11777	1.2%
IBM	215	1.4%	Huawei Technologies	11580	1.2%
Huawei Technologies	210	1.3%	Realtek	10314	1.0%
Kylin	181	1.1%	(Consultant)	9984	1.0%
Linutronix	171	1.1%	Marvell	8282	0.8%
SerNet	155	1.0%	Amutable	7483	0.8%

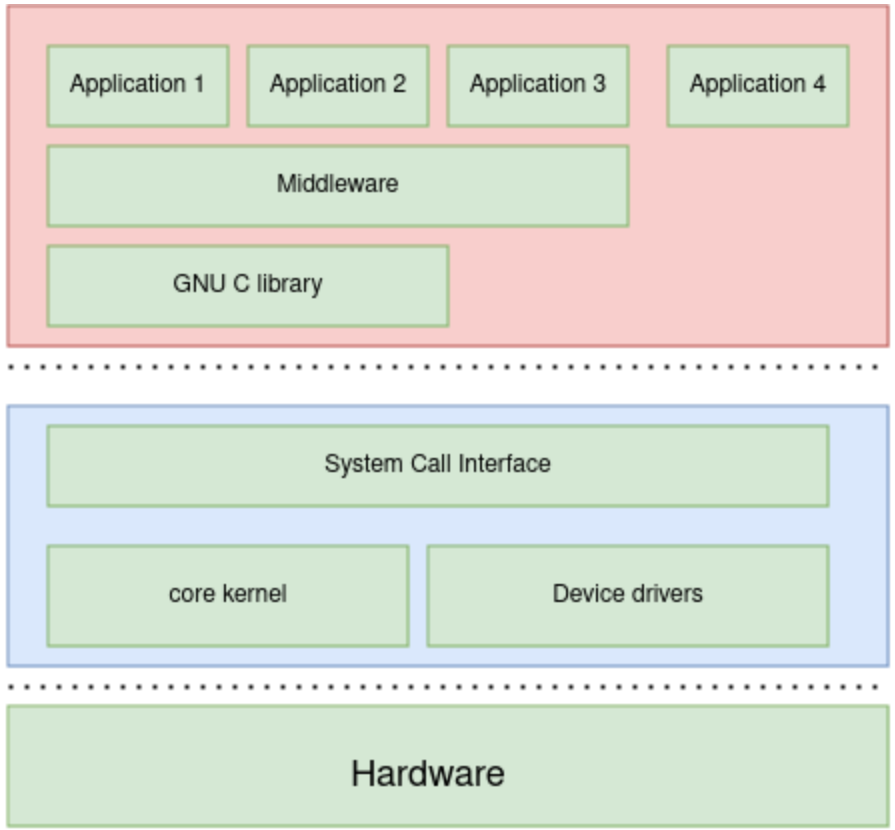
Linux kernel roles (1)

- Linux kernel is the **core component** of a Linux operating system
- Resource management
  - Processes, files, memory, scheduling, networking
- Hardware management
  - Device drivers
  - Allows user space apps to use the hardware
- Security



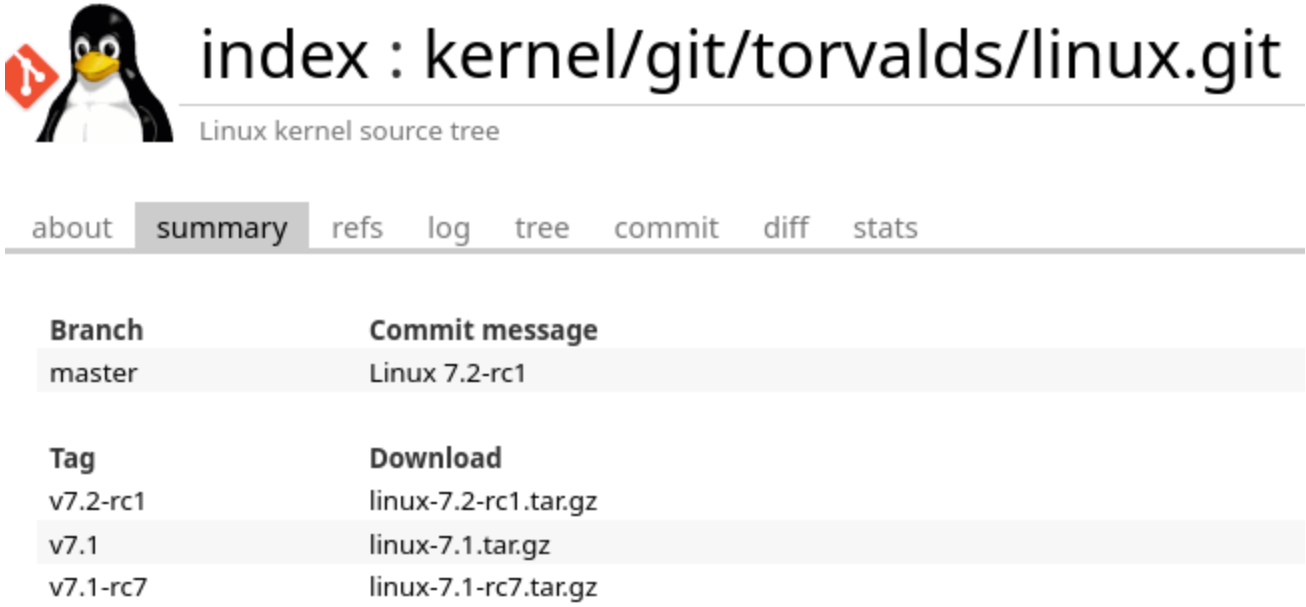
Linux kernel roles (2)

- Applications rely on kernel for services
  - functionalities are implemented via libraries
- User – kernel communications
  - via System Calls
- Linux kernel is monolithic
  - Everything happens in a single executable **Image**
  - **..but it has loadable modules!**



Linux kernel tree

- git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
- Linux kernel is written in C
- Compiled with GCC or Clang/LLVM
- There is also some assembly code
- Rust support
- Development *happens on email*
  - *git send-email*
- Distributed git repo
  - Each maintainer with its own tree
  - Linus Torvalds does the release



Exploring the source code

- vim
- Visual Studio Code
- https://elixir.bootlin.com/linux/latest/source

/ include / linux / sched.h

```
742  #ifdef CONFIG_KMAP_LOCAL
743          int                             idx;
744          pte_t                           pteval[KM_MAX_IDX];
745  #endif
746  };
747
748  struct task_struct {
749  #ifdef CONFIG_THREAD_INFO_IN_TASK
750          /*
751           * For reasons of header soup (see current_thread_info()), this
752           * must be the first element of task_struct.
753           */
754          struct thread_info              thread_info;
755  #endif
756          unsigned int                    __state;
```

Minimal binaries required to boot Linux

- U-boot
  - Used to load kernel Image and device tree
- Linux kernel image
- Device Tree Blob (dtb)
  - imx93-11x11-frdm.dtb
  - Used to describe the hardware on the board
- Rootfs (root file system)
  - Contains minimal userspace (/init, /bin, /lib)

Linux kernel development environment (1)

- See: https://nxp-research.github.io/lkss-main/infrastructure/main.html
- Cross-compiler et all

```
sudo apt-get update
sudo apt-get install -y build-essential libncurses-dev bc \
                        flex bison libssl-dev \
                        libelf-dev gcc-aarch64-linux-gnu
```

- Raw setup
  - Manually clone, install and configure necessary components
- Use LKSS script
  - lkss-main/scripts/lkss.py

Linux kernel development environment (2)

- git clone https://github.com/NXP-Research/lkss-main.git
- Initialize the environment
- Manually:
  - copy rootfs, uuu, flash.bin
  - Manually clone the linux kernel
- Use LKKS scripts

```
# create the virtual environment
python3 -m venv ./.venv

# activate the virtual environment
source ./.venv/bin/activate
```

  - ./scripts/lkss.py init --runner native –f
    - Clones the linux kernel
    - Downloads rootfs, uboot (flash.bin) and uuu tool

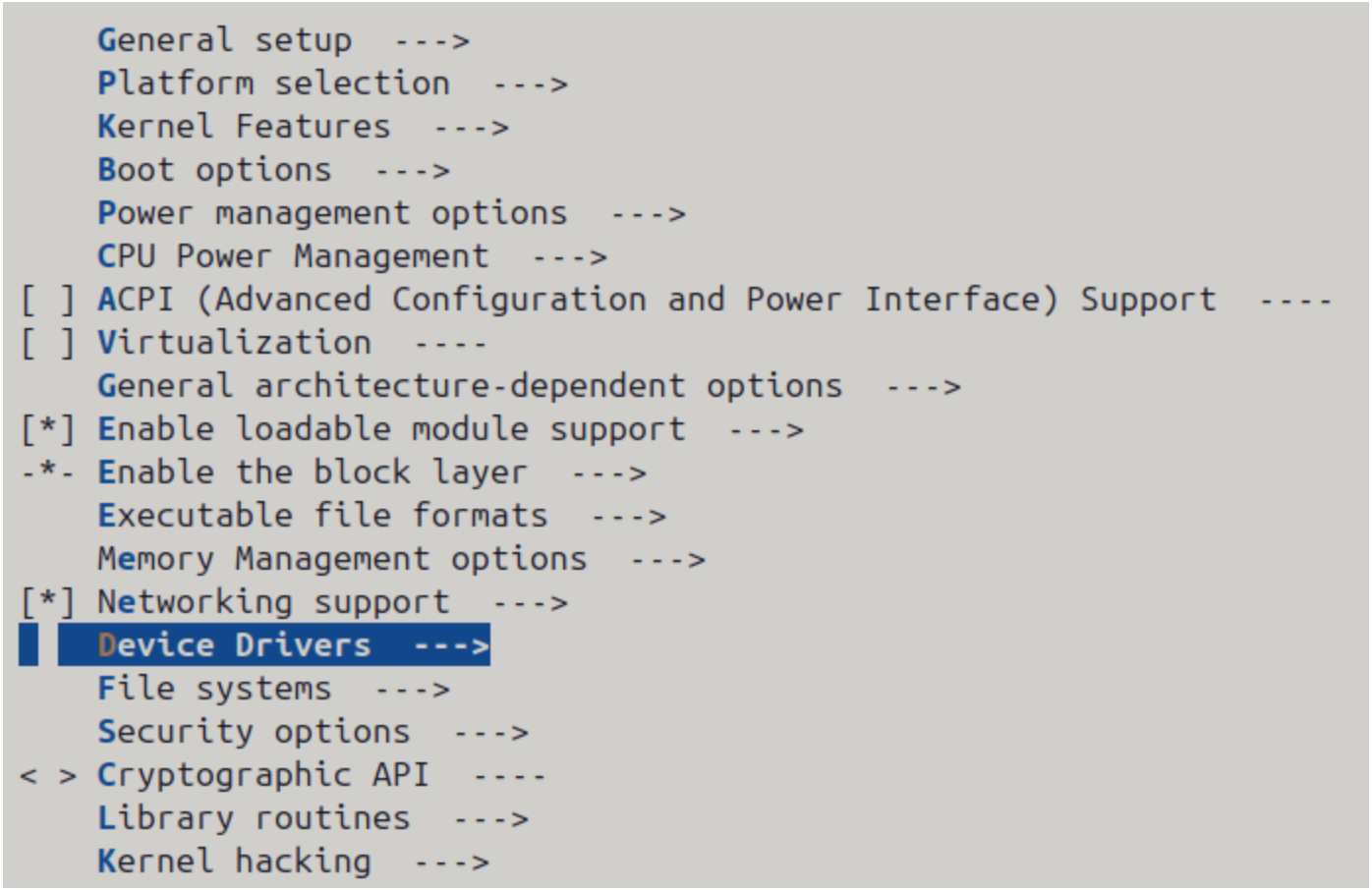
```
daniel@firefly:~/work/repos/lkss-main$ tree bin
bin
├── flash.bin
├── rootfs
└── uuu
```

Linux kernel development environment (3) - kernel config

- Linux kernel is huge!
- We need to be able to select parts of the code to be compiled in
- Configuration symbols (e.g CONFIG_NET)
- Default configuration
  - arch/x86/configs
  - arch/arm64/configs
- Configuration symbols
  - **Y**, code is compiled inside the Linux kernel image
  - **M,** code is compiled as a separate Linux kernel module
  - **N,** code is not considered for compilation
  - ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- make **imx93frdm_lkss_defconfig**

Create your own configuration

- Manually: ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- make menuconfig
- ./scripts/lkss.py menuconfig



Compiling the Linux kernel source code

- Manually
  - ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu-
  - INSTALL_MOD_PATH=/path/to/modules make modules_install
- LKSS script
  - ./scripts/lkss.py compile --install-modules

Booting the kernel

- Manually:
- LKSS script
  - ./scripts/lkss.py boot

- … and now lets have a look at the code

Simple "Hello World" kernel module

```c
// SPDX-License-Identifier: GPL-2.0

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

static int __init hello_init(void)
{
        printk(KERN_INFO "Hello from kernel space!\n");
        return 0;
}

static void __exit hello_exit(void)
{
        pr_info("Goodbye from kernel space!\n");
}

module_init(hello_init);
module_exit(hello_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("NXP Linux Kernel Summer School");
MODULE_DESCRIPTION("A simple Hello World kernel module");
```

Linux kernel logging

- Mechanism
  - **printk**, similar to printf but outputs to kernel buffer
- Log access tools
  - dmesg
  - journalctl
- Log levels
  - 0 (emerg) to 7 (debug)
  - printk(KERN_INFO "Hello World");
  - pr_info("Hello World")
- Serial console

```
root@nxp-lkss:~# cat /proc/sys/kernel/printk
4       4       1       7
```

```
root@nxp-lkss:~# dmesg
[    0.000000] Booting Linux on physical CPU 0x0000000000 [0x412fd050]
[    0.000000] Linux version 6.19.11-gebaf0e697859 (daniel@firefly) (aar
[    0.000000] Machine model: NXP i.MX93 11X11 FRDM board
[    0.000000] efi: UEFI not found.
[    0.000000] OF: reserved mem: node linux,cma compatible matching fail
[    0.000000] OF: reserved mem: 0x000000002021e000..0x000000002021efff
[    0.000000] OF: reserved mem: 0x00000000a4000000..0x00000000a4007fff
[    0.000000] OF: reserved mem: 0x00000000a4008000..0x00000000a400ffff
[    0.000000] OF: reserved mem: 0x00000000a4010000..0x00000000a4017fff
[    0.000000] OF: reserved mem: 0x00000000a4018000..0x00000000a401ffff
[    0.000000] Reserved memory: created DMA memory pool at 0x00000000a40
[    0.000000] OF: reserved mem: initialized node vdevbuffer@a4020000, c
[    0.000000] OF: reserved mem: 0x00000000a4020000..0x00000000a411ffff
[    0.000000] Zone ranges:
[    0.000000]   DMA      [mem 0x0000000080000000-0x00000000ffffffff]
[    0.000000]   DMA32    empty
[    0.000000]   Normal   empty
[    0.000000] Movable zone start for each node
```

Build / Load the Linux kernel module

- Build (on your laptop)
  - ./scripts/lkss.py menuconfig (select your module)
  - ./scripts/lkss.py compile --install-modules
- On the target board
  - modprobe / insmod
  - rmmod

Kernel errors: Oops vs Panic

- Oops
  - A non-fatal error detected in kernel space
  - System may continue running
  - Prints diagnostic info: stack trace, registers
- Panic
  - A fatal error detected in kernel space
  - System will halt

```
   6.015961] Call trace:
   6.018397]  oops_sample_init+0x24/0x1000 [oops] (P)
   6.023354]  do_one_initcall+0x60/0x1d4
   6.027184]  do_init_module+0x54/0x22c
   6.030928]  load_module+0x1704/0x1cac
   6.034672]  init_module_from_file+0xdc/0xf8
   6.038936]  __arm64_sys_finit_module+0x174/0x3d8
   6.043633]  invoke_syscall.constprop.0+0x50/0xe0
   6.048331]  do_el0_svc+0x40/0xc0
   6.051641]  el0_svc+0x3c/0x164
   6.054779]  el0t_64_sync_handler+0xa0/0xf0
   6.058956]  el0t_64_sync+0x198/0x19c
   6.062617] Code: 95d98157 d2800001 52800542 52800000 (b9000022)
   6.068699] ---[ end trace 0000000000000000 ]---
```

```
[    5.832254] oops: about to dereference NULL pointer...
[    5.837445] Unable to handle kernel NULL pointer dereference at virtual address 0000000000000000
[    5.846261] Mem abort info:
[    5.849042]   ESR = 0x0000000096000044
[    5.852789]   EC = 0x25: DABT (current EL), IL = 32 bits
[    5.858096]   SET = 0, FnV = 0
[    5.861146]   EA = 0, S1PTW = 0
[    5.864283]   FSC = 0x04: level 0 translation fault
[    5.869160] Data abort info:
[    5.872039]   ISV = 0, ISS = 0x00000044, ISS2 = 0x00000000
[    5.877517]   CM = 0, WnR = 1, TnD = 0, TagAccess = 0
[    5.882578]   GCS = 0, Overlay = 0, DirtyBit = 0, Xs = 0
```

Kernel API: Timers

- allows scheduling activities in the future
- **Jiffies,** stores the number of timer ticks
- Timer API
  - struct timer_list
  - timer_setup
  - mod_timer
  - del_timer_sync

```c
#define TIMER_INTERVAL_MS 1000  /* 1000 milliseconds = 1 second */

static struct timer_list my_timer;

/* Timer callback */
static void my_timer_callback(struct timer_list *t)
{
        pr_info("Timer callback executed at jiffies=%lu\n", jiffies);
        /* TODO schedule the timer again, using `mod_timer` function */

}

static int __init lkss_timer_init(void)
{
        pr_info("Initializing the timer module\n");

        /* Initialize the timer */
        timer_setup(&my_timer, my_timer_callback, 0);

        /* schedule the timer for the first time */
        mod_timer(&my_timer, jiffies + msecs_to_jiffies(TIMER_INTERVAL_MS));
        return 0;
}

static void __exit lkss_timer_exit(void)
{
        /* Delete the timer if still active */
        timer_delete_sync(&my_timer);

        pr_info("Timer module exited\n");
}
```

Device tree

- Device tree
  - A data structure used to describe hardware
  - Used at boot to inform the kernel about the HW hierarchy
  - Kernel matches compatible string with platform driver

```
lkss-bus {
    compatible = "simple-bus";

    my-sensor {
        compatible = "lkss,lab1-sensor";
        label = "temperature-sensor";
        sample-rate-hz = <100>;
        max-temp-celsius = <85>;
        status = "okay";
    };
};
```

- Platform drivers
  - Used for non-discoverable hardware
  - Handle platform devices described in device tree
  - Key API
    - Probe
    - Remove

```
magnetometer: magnetometer@1e»  {
»       compatible = "st,lsm9ds1-magn";
»       reg = <0x1e>;
»       pinctrl-names = "default";
»       pinctrl-0 = <&pinctrl_mag>;
»       interrupt-parent = <&gpio3>;
»       interrupts = <22 IRQ_TYPE_LEVEL_HIGH>;
»       vdd-supply = <&reg_vdd_sen>;
»       vddio-supply = <&reg_vdd_1v8>;
};
```

Platform driver

```
13 static int sensor_probe(struct platform_device *pdev)
14 {
15         dev_info(&pdev->dev, "sensor probed!\n");
16
17         return 0;
18 }
19 static void sensor_remove(struct platform_device *pdev)
20 {
21 }
22 static const struct of_device_id sensor_dt_ids[] = {
23         { .compatible = "lkss,lab1-sensor" },
24         { /* sentinel */ },
25 };
26
27 static struct platform_driver sensor_driver = {
28         .probe  = sensor_probe,
29         .remove = sensor_remove,
30         .driver = {
31                 .name           = "lkss_sensor",
32                 .of_match_table = sensor_dt_ids,
33         },
34 };
35 module_platform_driver(sensor_driver);
```