Linux kernel

Ngoc Nguyen Tran

Contents

- Introduction
- Features
- Linux kernel sources
- Building the kernel

Linux kernel introduction

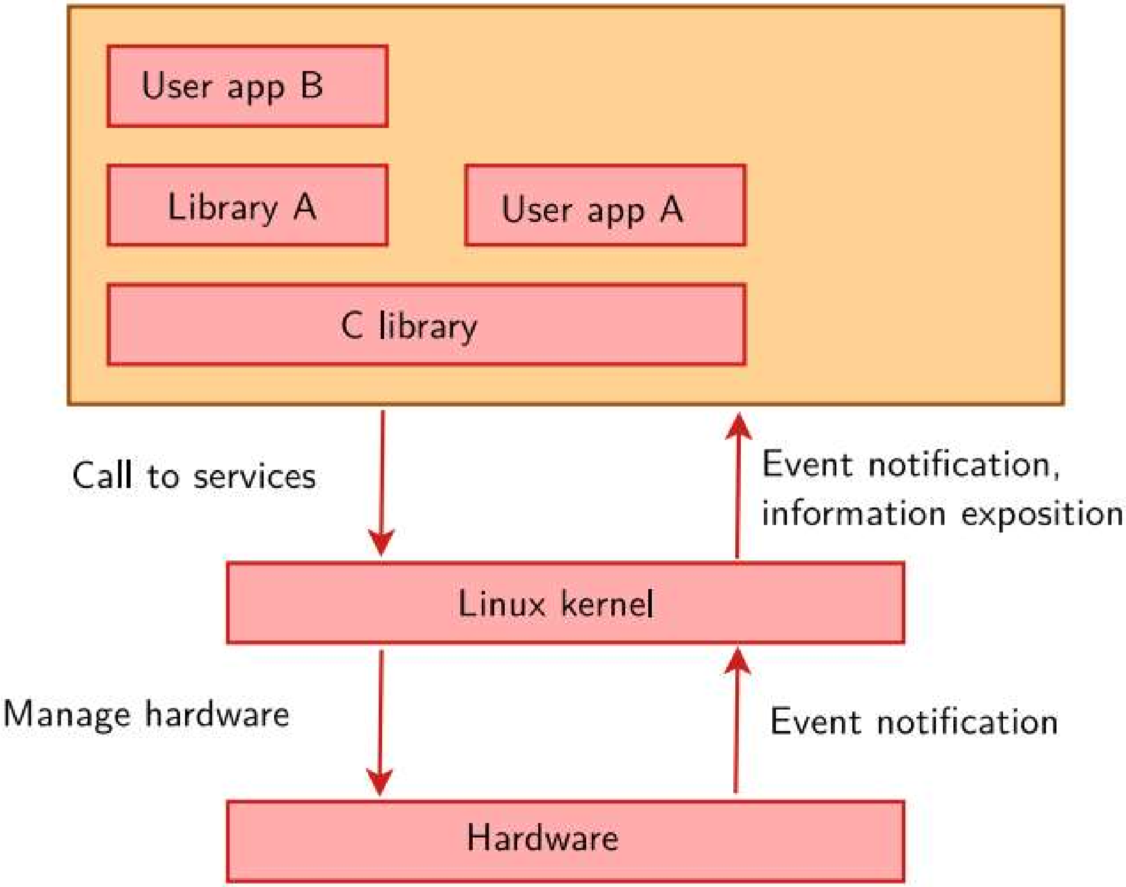
- The Linux kernel is one component of a system
- The Linux kernel was created as a hobby in 1991 by a Finnish student, Linus Torvalds

Linus Torvalds (2014)

Linux kernel features

- Portability and hardware support.
    - Runs on most architectures.
- Scalability. Can run on super computers as well as on tiny devices (4 MB of RAM is enough).
- Compliance to standards and interoperability.
- Exhaustive networking support
- Security. It can’t hide its flaws. Its
- code is reviewed by many experts.
- Stability and reliability.
- Modularity. Can include only what a system needs even at run time.
- Easy to program. You can learn from existing code. Many useful resources on the net.

Linux kernel in the system



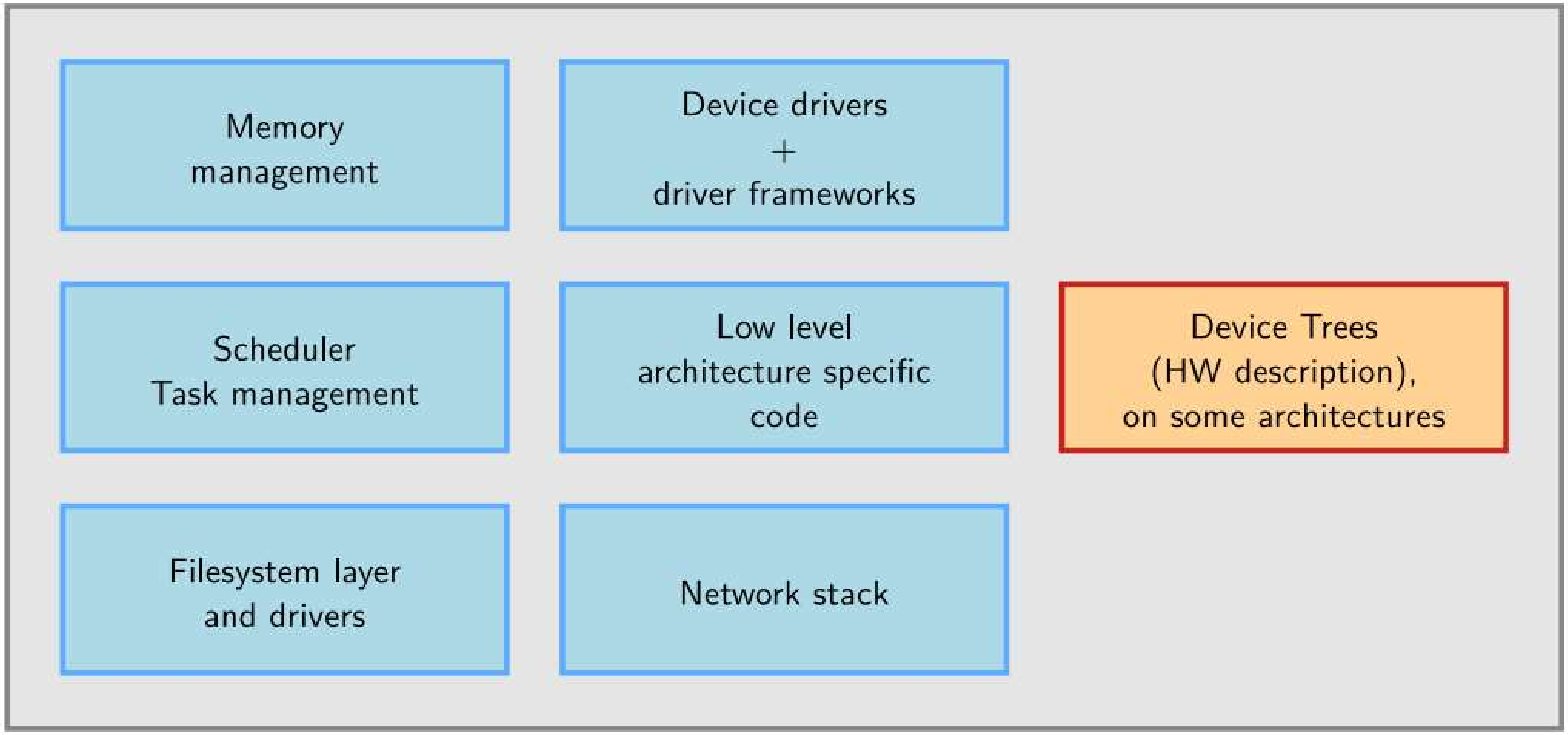
Pseudo filesystems

- Linux makes system and kernel information available in user space through **pseudo filesystems**, or **virtual filesystems**
- Pseudo filesystems allow applications to see directories and files that do not exist on any real storage: they are created and updated on the fly by the kernel
- The two most important pseudo filesystems are
  - proc, usually mounted on /proc:

    Operating system related information (processes, memory management parameters...)
  - sysfs, usually mounted on /sys:

    Representation of the system as a set of devices and buses. Information about these devices

Inside the Linux kernel



Memory management

Device drivers + driver frameworks

Scheduler Task management

Low level architecture specific code

Device Trees (HW description), on some architectures

Filesystem layer and drivers

Network stack

Linux kernel sources

- Available at https://kernel.org
- Many chip vendors supply their own kernel sources
- Many kernel sub-communities maintain their own kernel, with usually newer but less stable features
  - Architecture communities (ARM, MIPS, PowerPC, etc.), device drivers communities (I2C, SPI, USB, PCI, network, etc.), other communities (real-time, etc.)

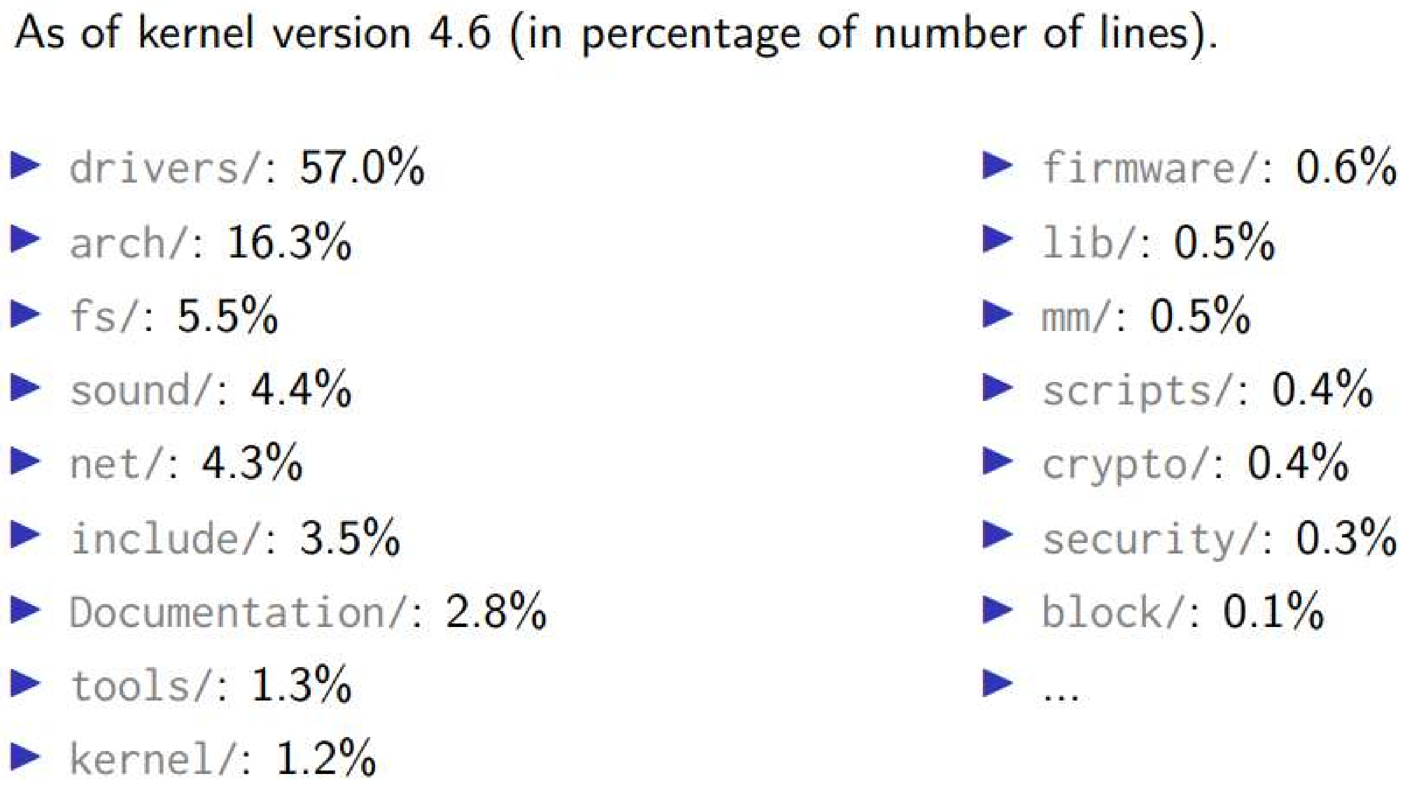
Getting Linux sources

- Available from https://kernel.org/pub/linux/kernel
  - Full tarballs: complete kernel sources - `https://kernel.org/pub/linux/kernel/v4.x/patch-4.20.xz`
  - Patches: differences between two kernel versions - https://kernel.org/pub/linux/kernel/v4.x/linux-4.20.13.tar.xz
- Get by git
- Using ***patch*** command to apply a patch to a current system
  - Need to run the patch command inside the toplevel kernel source directory

Linux kernel size

- Linux 5.4 sources:
  - 66031 files (git ls-files | wc -l)
  - 27679764 lines (git ls-files | xargs cat | wc -l)
  - 889221135 bytes (git ls-files | xargs cat | wc -c)
- A minimum uncompressed Linux kernel just sizes 1-2 MB
- The Linux core (scheduler, memory management...) is pretty small!
- Why are these sources so big?

Linux kernel size (2)



As of kernel version 4.6 (in percentage of number of lines).

- `drivers/`: 57.0%
- `arch/`: 16.3%
- `fs/`: 5.5%
- `sound/`: 4.4%
- `net/`: 4.3%
- `include/`: 3.5%
- `Documentation/`: 2.8%
- `tools/`: 1.3%
- `kernel/`: 1.2%
- `firmware/`: 0.6%
- `lib/`: 0.5%
- `mm/`: 0.5%
- `scripts/`: 0.4%
- `crypto/`: 0.4%
- `security/`: 0.3%
- `block/`: 0.1%
- ...

Building the kernel

- The kernel configuration is the process of defining the set of options
  - Target architecture and device drivers
  - Your kernel capabilities (network, filesystem, real-time, etc.)
- Steps to build a kernel
  - Specifying the target architecture – under arch/:export ARCH=arm
  - Kernel configuration and build system – use make
  - Kernel configuration details - .config file at the root of kernel source
    - Edit manually or use tools such as **xconfig**, gconfig, menuconfig, nconfig
  - Initial configuration – find one configuration that works for your system first
    - Desktop: inside /boot
    - Embedded platform: default configuration are available for each CPU family, /arch/<arch>/configs/, the file .config
    - Run make help to find it
  - Create your own default configuration

Kernel or module?

- The kernel image is a single file, resulting from the linking of all object files that correspond to features enabled in the configuration
    - This is the file that gets loaded in memory by the bootloader
    - All included features are therefore available as soon as the kernel starts, at a time where no filesystem exists
- Some features (device drivers, filesystems, etc.) can however be compiled as modules
    - These are plugins that can be loaded/unloaded dynamically to add/remove features to the kernel
    - Each module is stored as a separate file in the filesystem, and therefore access to a filesystem is mandatory to use modules
    - This is not possible in the early boot procedure of the kernel, because no filesystem is available

Kernel option types

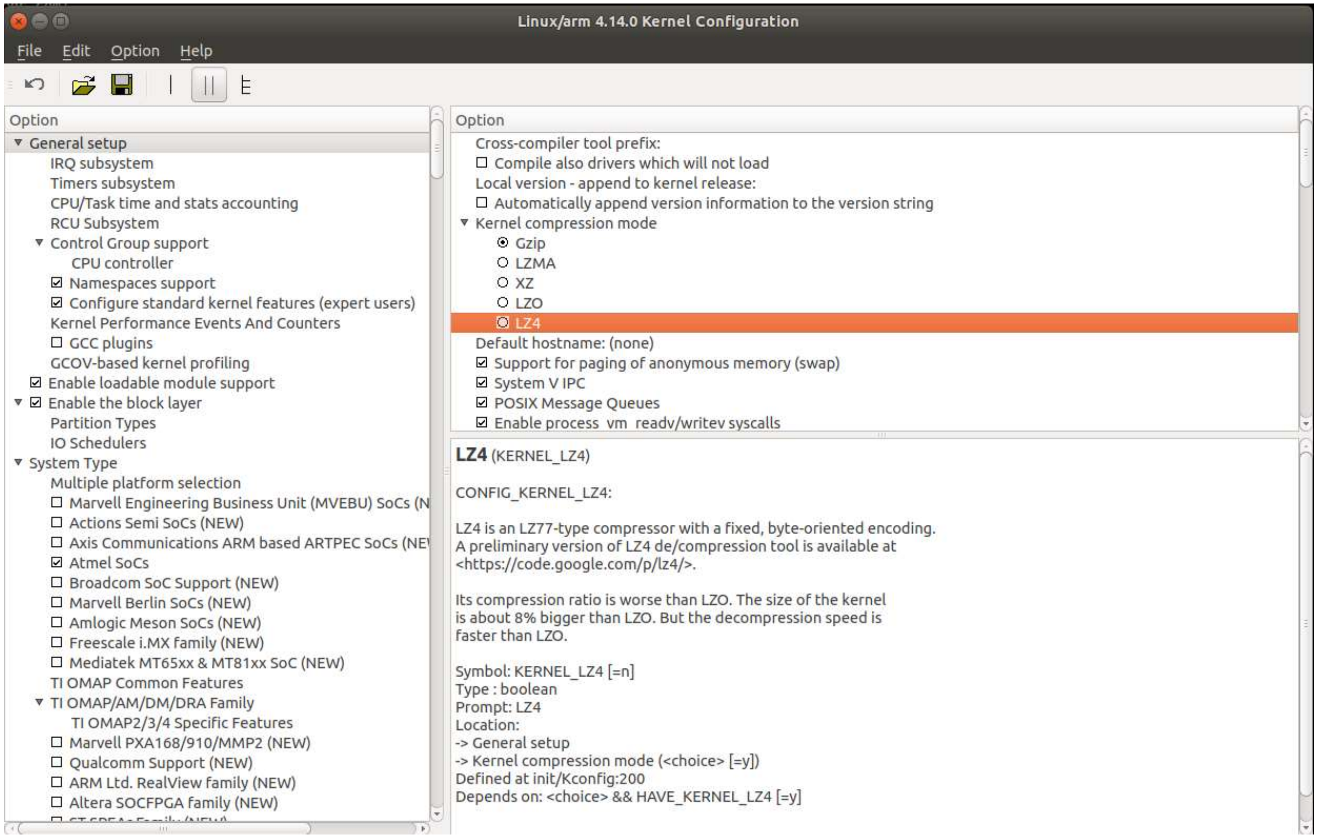
There are different types of options

- bool options, they are either
  - true (to include the feature in the kernel) or
  - false (to exclude the feature from the kernel)
- tristate options, they are either
  - true (to include the feature in the kernel image) or
  - module (to include the feature as a kernel module) or
  - false (to exclude the feature)
- int options, to specify integer values
- hex options, to specify hexadecimal values
- string options, to specify string values

make xconfig

- make xconfig
  - The most common graphical interface to configure the kernel.
  - File browser: easier to load configuration files
  - Search interface to look for parameters
  - Required Debian / Ubuntu packages: qt5-default
- make gconfig
  - GTK based graphical configuration interface. Functionality similar to that of make xconfig.
  - Just lacking a search functionality.
  - Required Debian packages: libglade2-dev

make xconfig screenshot



```
Option
▼ General setup
    IRQ subsystem
    Timers subsystem
    CPU/Task time and stats accounting
    RCU Subsystem
  ▼ Control Group support
      CPU controller
    ☑ Namespaces support
    ☑ Configure standard kernel features (expert users)
    Kernel Performance Events And Counters
    ☐ GCC plugins
    GCOV-based kernel profiling
  ☑ Enable loadable module support
▼ ☑ Enable the block layer
    Partition Types
    IO Schedulers
▼ System Type
    Multiple platform selection
    ☐ Marvell Engineering Business Unit (MVEBU) SoCs (N
    ☐ Actions Semi SoCs (NEW)
    ☐ Axis Communications ARM based ARTPEC SoCs (NE
    ☑ Atmel SoCs
    ☐ Broadcom SoC Support (NEW)
    ☐ Marvell Berlin SoCs (NEW)
    ☐ Amlogic Meson SoCs (NEW)
    ☐ Freescale i.MX family (NEW)
    ☐ Mediatek MT65xx & MT81xx SoC (NEW)
    TI OMAP Common Features
  ▼ TI OMAP/AM/DM/DRA Family
      TI OMAP2/3/4 Specific Features
    ☐ Marvell PXA168/910/MMP2 (NEW)
    ☐ Qualcomm Support (NEW)
    ☐ ARM Ltd. RealView family (NEW)
    ☐ Altera SOCFPGA family (NEW)
    ☐ ST SPEAr Family (NEW)
```

```
Option
  Cross-compiler tool prefix:
  ☐ Compile also drivers which will not load
  Local version - append to kernel release:
  ☐ Automatically append version information to the version string
▼ Kernel compression mode
    ● Gzip
    ○ LZMA
    ○ XZ
    ○ LZO
    ○ LZ4
  Default hostname: (none)
  ☑ Support for paging of anonymous memory (swap)
  ☑ System V IPC
  ☑ POSIX Message Queues
  ☑ Enable process vm readv/writev syscalls
```

LZ4 (KERNEL_LZ4)

CONFIG_KERNEL_LZ4:

LZ4 is an LZ77-type compressor with a fixed, byte-oriented encoding.
A preliminary version of LZ4 de/compression tool is available at
<https://code.google.com/p/lz4/>.

Its compression ratio is worse than LZO. The size of the kernel
is about 8% bigger than LZO. But the decompression speed is
faster than LZO.

Symbol: KERNEL_LZ4 [=n]
Type : boolean
Prompt: LZ4
Location:
-> General setup
-> Kernel compression mode (<choice> [=y])
Defined at init/Kconfig:200
Depends on: <choice> && HAVE_KERNEL_LZ4 [=y]

Other similar tools

- gconfig
- menuconfig
- nconfig
- oldconfig

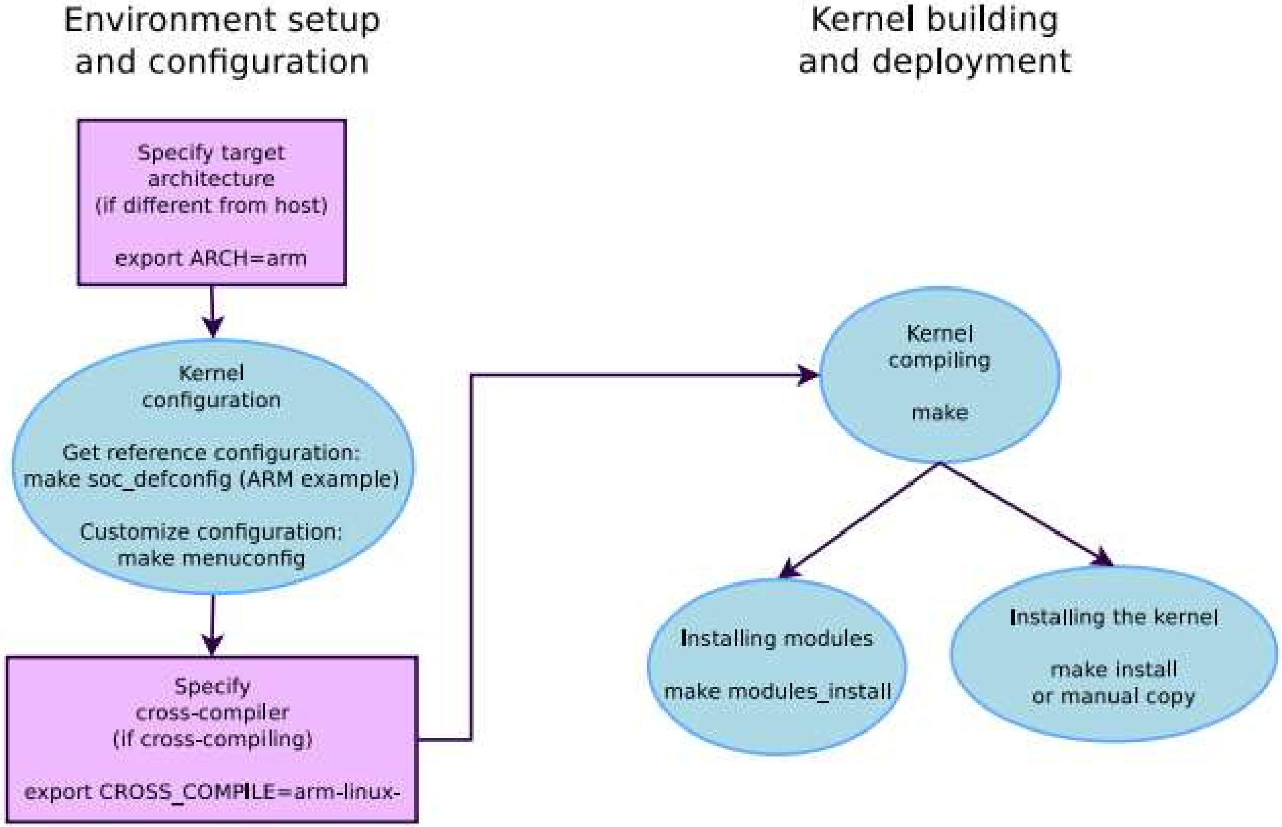
Undoing configuration changes

- A frequent problem:
- After changing several kernel configuration settings, your kernel no longer works.
- If you don’t remember all the changes you made, you can get back to your previous configuration:

```
$ cp .config.old .config
```

- All the configuration interfaces of the kernel (xconfig, menuconfig, oldconfig...) keep this .config.old backup copy.

Kernel building overview



Environment setup and configuration

Specify target architecture (if different from host)

export ARCH=arm

Kernel configuration

Get reference configuration:
make soc_defconfig (ARM example)

Customize configuration:
make menuconfig

Specify cross-compiler (if cross-compiling)

export CROSS_COMPILE=arm-linux-

Kernel building and deployment

Kernel compiling

make

Installing modules

make modules_install

Installing the kernel

make install
or manual copy

Compiling a kernel

- Choose a compiler *($(CROSS_COMPILE)gcc*)
  - Native: undefined CROSS_COMPILE
  - Cross-compiler → Example: *export CROSS_COMPILE=arm-linux-gnueabi-* for *arm-linux-gnueabi-gcc*
- Specifying ARCH and CROSS_COMPILE
  - Pass *ARCH* and *CROSS_COMPILE* on the make command line:
    Example: *make ARCH=arm CROSS_COMPILE=arm-linux-*
  - Define *ARCH* and *CROSS_COMPILE* as environment variables: (*export*)
- Kernel compilation
  - make
    - Can run multiple jobs in parallel if having multiple CPU cores (make –j 8)
  - Generates:
    - vmlinux, the raw uncompressed kernel image (cannot be booted)
    - arch/<arch>/boot/*Image, the final, usually compressed, kernel image that can be booted
      Example: bzImage for x86, zImage for ARM, vmlinux.bin.gz for ARC, etc.

Kernel installation – native case

- make install
  - Does the installation for the host system by default, so needs to be run as root.
- Installs
  - */boot/vmlinuz-<version>*
    - Compressed kernel image. Same as the one in *arch/<arch>/boot*
  - */boot/System.map-<version>*
    - Stores kernel symbol addresses for debugging purposes (obsolete: such information is usually stored in the kernel itself)
  - */boot/config-<version>*
    - Kernel configuration for this version
- In GNU/Linux distributions, typically re-runs the bootloader configuration utility to make the new kernel available at the next boot.

Kernel installation: embedded case

- make install is rarely used in embedded development, as the kernel image is a single file, easy to handle.
- Another reason is that there is no standard way to deploy and use the kernel image.
- Therefore making the kernel image available to the target is usually manual or done through scripts in build systems.
- It is however possible to customize the make install behavior in

  arch/<arch>/boot/install.sh

Kernel cleanup target

- Clean-up generated files (to force re-compilation):

```
make clean
```

- Remove all generated files. Needed when switching from onearchitecture to another. Caution: it also removes your .config file!

```
make mrproper
```

- Also remove editor backup and patch reject files (mainly to generate patches):

```
make distclean
```

- If you are in a git tree, remove all files not tracked (and ignored) by git:

```
git clean -fdx
```