Linux Kernel

Peripheral Devices for Embedded Systems

Rafal Kapela

June 26, 2016

Outline

Outline

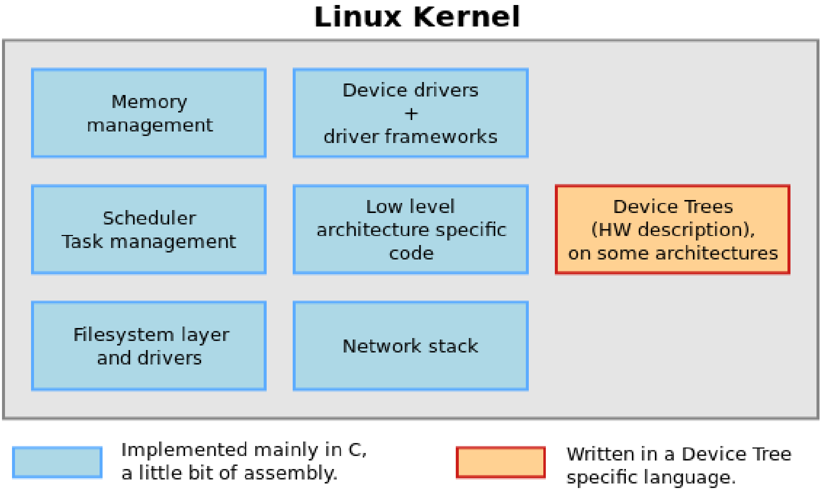
History

- The Linux kernel is one component of a system, which also requires libraries and applications to provide features to end users.
- The Linux kernel was created as a hobby in 1991 by a Finnish student, Linus Torvalds.
  - Linux quickly started to be used as the kernel for free software operating systems
- Linus Torvalds has been able to create a large and dynamic developer and user community around Linux.
- Nowadays, hundreds of people contribute to each kernel release, individuals or companies big and small.

Linux kernel main roles

- **Manage all the hardware resources**: CPU, memory, I/O.
- Provide a **set of portable, architecture and hardware independent APIs** to allow userspace applications and libraries to use the hardware resources.
- **Handle concurrent accesses and usage** of hardware resources from different applications.
  - Example: a single network interface is used by multiple userspace applications through various network connections. The kernel is responsible to "multiplex" the hardware resource.

Inside the Linux kernel



Linux license

- The whole Linux sources are Free Software released under the GNU General Public License version 2 (GPL v2).
- For the Linux kernel, this basically implies that:
  - When you receive or buy a device with Linux on it, you should receive the Linux sources, with the right to study, modify and redistribute them.
  - When you produce Linux based devices, you must release the sources to the recipient, with the same rights, with no restriction..

Linux kernel key 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.

Supported hardware arch.

- See the arch/ directory in the kernel sources
- Minimum: 32 bit processors, with or without MMU, and gcc support
- 32 bit architectures (arch/ subdirectories)
  Examples: arm, avr32, blackfin, m68k, microblaze, mips, score, sparc, um
- 64 bit architectures:
  Examples: alpha, arm64, ia64, sparc64, tile
- 32/64 bit architectures
  Examples: powerpc, x86, sh
- Find details in kernel sources: arch/< *arch* >/Kconfig, arch/< *arch* >/README, or Documentation/< *arch* >/

System calls

- The main interface between the kernel and userspace is the set of system calls
- About 300 system calls that provide the main kernel services
  - File and device operations, networking operations, inter-process communication, process management, memory mapping, timers, threads, synchronization primitives, etc.
- This interface is stable over time: only new system calls can be added by the kernel developers
- This system call interface is wrapped by the C library, and userspace applications usually never make a system call directly but rather use the corresponding C library function

Virtual filesystems

- Linux makes system and kernel information available in user-space through virtual filesystems.
- Virtual filesystems allow applications to see directories and files that do not exist on any real storage: they are created on the fly by the kernel
- The two most important virtual 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.

Outline

Location of kernel sources

- The official version of the Linux kernel, as released by Linus Torvalds is available at `http://www.kernel.org`
  - This version follows the well-defined development model of the kernel
  - However, it may not contain the latest development from a specific area, due to the organization of the development model and because features in development might not be ready for mainline inclusion
- 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.)
  - They generally don't release official versions, only development trees are available

Getting Linux sources

- The kernel sources are available from http://kernel.org/pub/linux/kernel as **full tarballs** (complete kernel sources) and **patches** (differences between two kernel versions).
- But for kernel development, one generally uses the Git version control system:
  - Fetch the entire kernel sources and history

    git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
  - Create a branch that starts at a specific stable version
    git checkout -b $< name - of - branch >$ v3.11
  - Web interface available at

    http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/
  - Read more about Git at http://git-scm.com/

Linux kernel size

- Linux 3.10 sources:
  Raw size: 573 MB (43,000 files, approx 15,800,000 lines) gzip compressed tar archive: 105 MB bzip2 compressed tar archive: 83 MB (better) xz compressed tar archive: 69 MB (best)
- Minimum Linux 2.6.29 compiled kernel size with CONFIG_EMBEDDED, for a kernel that boots a QEMU PC (IDE hard drive, ext2 filesystem, ELF executable support): 532 KB (compressed), 1325 KB (raw)
- Why are these sources so big?
  Because they include thousands of device drivers, many network protocols, support many architectures and filesystems...
- The Linux core (scheduler, memory management...) is pretty small!

Outline

Kernel configuration

- The kernel configuration and build system is based on multiple Makefiles
- One only interacts with the main Makefile, present at the **top directory** of the kernel source tree
- Interaction takes place
  - using the make tool, which parses the Makefile
  - through various **targets**, defining which action should be done (configuration, compilation, installation, etc.). Run make help to see all available targets.
- Example
  - cd linux-3.6.x/
  - make $<$ *target* $>$

Kernel configuration (1)

- The kernel contains thousands of device drivers, filesystem drivers, network protocols and other configurable items
- Thousands of options are available, that are used to selectively compile parts of the kernel source code
- The kernel configuration is the process of defining the set of options with which you want your kernel to be compiled
- The set of options depends
  - On your hardware (for device drivers, etc.)
  - On the capabilities you would like to give to your kernel (network capabilities, filesystems, real-time, etc.)

Kernel configuration (2)

- The configuration is stored in the .config file at the root of kernel sources
  - Simple text file, key=value style
- As options have dependencies, typically never edited by hand, but through graphical or text interfaces:
  - make xconfig, make gconfig (graphical)
  - make menuconfig, make nconfig (text)
  - You can switch from one to another, they all load/save the same .config file, and show the same set of options
- To modify a kernel in a GNU/Linux distribution: the configuration files are usually released in /boot/, together with kernel images: /boot/config-3.2.0-31-generic

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**
  - Those 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
  - string options, to specify string values

Kernel option dependencies

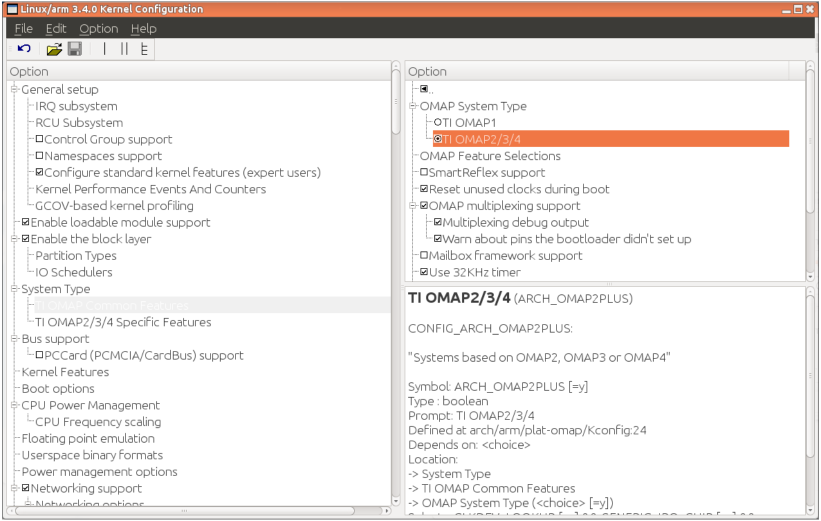
- There are dependencies between kernel options
- For example, enabling a network driver requires the network stack to be enabled
- Two types of dependencies
  - depends on dependencies. In this case, option A that depends on option B is not visible until option B is enabled
  - select dependencies. In this case, with option A depending on option B, when option A is enabled, option B is automatically enabled
  - make xconfig allows to see all options, even those that cannot be selected because of missing dependencies. In this case, they are displayed in gray

make xconfig

alertmake xconfig

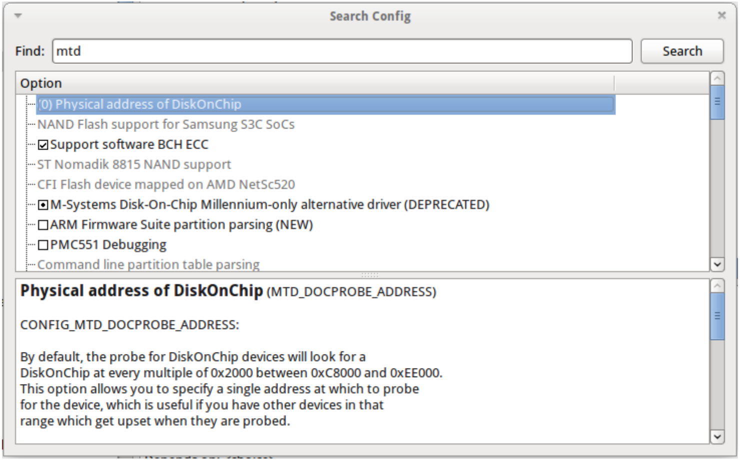
- The most common graphical interface to configure the kernel.
- Make sure you read help - introduction: useful options!
- File browser: easier to load configuration files
- Search interface to look for parameters
- Required Debian / Ubuntu packages: libqt4-dev g++ (libqt3-mt-dev for older kernel releases)

make xconfig screenshot

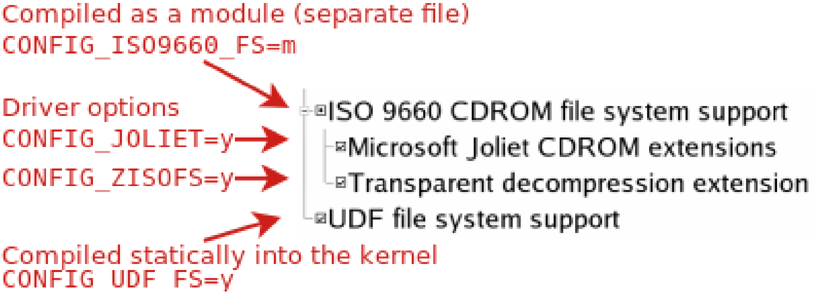


make xconfig search interface

Looks for a keyword in the parameter name. Allows to select or unselect found parameters.



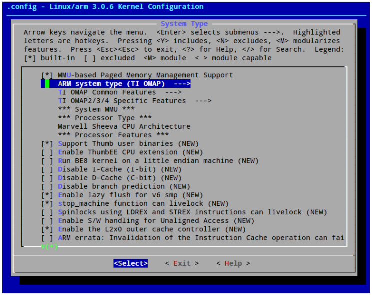
Kernel configuration options



make menuconfig

make menuconfig

- Useful when no graphics are available. Pretty convenient too!
- Same interface found in other tools: BusyBox, Buildroot...
- Required Debian packages: libncurses-dev



make oldconfig

make oldconfig

- Needed very often!
- Useful to upgrade a .config file from an earlier kernel release
- Issues warnings for configuration parameters that no longer exist in the new kernel.
- Asks for values for new parameters

If you edit a .config file by hand, it's strongly recommended to run make oldconfig afterwards!

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.

Configuration per architecture

- The set of configuration options is architecture dependent
  - Some configuration options are very architecture-specific
  - Most of the configuration options (global kernel options, network subsystem, filesystems, most of the device drivers) are visible in all architectures.
- By default, the kernel build system assumes that the kernel is being built for the host architecture, i.e. native compilation
- The architecture is not defined inside the configuration, but at a higher level
- We will see later how to override this behaviour, to allow the configuration of kernels for a different architecture

Overview of kernel options (1)

- General setup
  - *Local version - append to kernel release* allows to concatenate an arbitrary string to the kernel version that a user can get using uname -r. Very useful for support!
  - *Support for swap*, can usually be disabled on most embedded devices
  - *Configure standard kernel features (expert users)* allows to remove features from the kernel to reduce its size. Powerful, but use with care!

Overview of kernel options (2)

- Loadable module support
  - Allows to enable or completely disable module support. If your system doesn't need kernel modules, best to disable since it saves a significant amount of space and memory
- Enable the block layer
  - If CONFIG_EXPERT is enabled, the block layer can be completely removed. Embedded systems using only flash storage can safely disable the block layer
- Processor type and features (x86) or System type (ARM) or CPU selection (MIPS)
  - Allows to select the CPU or machine for which the kernel must be compiled
  - On x86, only optimization-related, on other architectures very important since there's no compatibility

Overview of kernel options (3)

- Kernel features
  - Tickless system, which allows to disable the regular timer tick and use on-demand ticks instead. Improves power savings
  - High resolution timer support. By default, the resolution of timer is the tick resolution. With high resolution timers, the resolution is as precise as the hardware can give
  - Preemptible kernel enables the preemption inside the kernel code (the userspace code is always preemptible). See our real-time presentation for details
- Power management
  - Global power management option needed for all power management related features
  - Suspend to RAM, CPU frequency scaling, CPU idle control, suspend to disk

Overview of kernel options (4)

- Networking support
  - The network stack
  - Networking options
    - Unix sockets, needed for a form of inter-process communication
    - TCP/IP protocol with options for multicast, routing, tunneling, Ipsec, Ipv6, congestion algorithms, etc.
    - Other protocols such as DCCP, SCTP, TIPC, ATM
    - Ethernet bridging, QoS, etc.
  - Support for other types of network
    - CAN bus, Infrared, Bluetooth, Wireless stack, WiMax stack, etc.

Overview of kernel options (5)

- Device drivers
  - MTD is the subsystem for flash (NOR, NAND, OneNand, battery-backed memory, etc.)
  - Parallel port support
  - Block devices, a few misc block drivers such as loopback, NBD, etc.
  - ATA/ATAPI, support for IDE disk, CD-ROM and tapes. A new stack exists
  - SCSI
    - The SCSI core, needed not only for SCSI devices but also for USB mass storage devices, SATA and PATA hard drives, etc.
    - SCSI controller drivers

Overview of kernel options (6)

- Device drivers (cont)
  - SATA and PATA, the new stack for hard disks, relies on SCSI
  - RAID and LVM, to aggregate hard drives and do replication
  - Network device support, with the network controller drivers. Ethernet, Wireless but also PPP
  - Input device support, for all types of input devices: keyboards, mice, joysticks, touchscreens, tablets, etc.
  - Character devices, contains various device drivers, amongst them
    - serial port controller drivers
    - PTY driver, needed for things like SSH or telnet
  - I2C, SPI, 1-wire, support for the popular embedded buses
  - Hardware monitoring support, infrastructure and drivers for thermal sensors

Overview of kernel options (7)

- Device drivers (cont)
  - Watchdog support
  - Multifunction drivers are drivers that do not fit in any other category because the device offers multiple functionality at the same time
  - Multimedia support, contains the V4L and DVB subsystems, for video capture, webcams, AM/FM cards, DVB adapters
  - Graphics support, infrastructure and drivers for framebuffers
  - Sound card support, the OSS and ALSA sound infrastructures and the corresponding drivers
  - HID devices, support for the devices that conform to the HID specification (Human Input Devices)

Overview of kernel options (8)

- Device drivers (cont)
  - USB support
    - Infrastructure
    - Host controller drivers
    - Device drivers, for devices connected to the embedded system
    - Gadget controller drivers
    - Gadget drivers, to let the embedded system act as a mass-storage device, a serial port or an Ethernet adapter
  - MMC/SD/SDIO support
  - LED support
  - Real Time Clock drivers
  - Voltage and current regulators
  - Staging drivers, crappy drivers being cleaned up

Overview of kernel options (9)

- For some categories of devices the driver is not implemented inside the kernel
  - Printers
  - Scanners
  - Graphics drivers used by X.org
  - Some USB devices
- For these devices, the kernel only provides a mechanism to access the hardware, the driver is implemented in userspace

Overview of kernel options (10)

- File systems
  - The common Linux filesystems for block devices: ext2, ext3, ext4
  - Less common filesystems: XFS, JFS, ReiserFS, GFS2, OCFS2, Btrfs
  - CD-ROM filesystems: ISO9660, UDF
  - DOS/Windows filesystems: FAT and NTFS
  - Pseudo filesystems: proc and sysfs
  - Miscellaneous filesystems, with amongst other flash filesystems such as JFFS2, UBIFS, SquashFS, cramfs
  - Network filesystems, with mainly NFS and SMB/CIFS
- Kernel hacking
  - Debugging features useful for kernel developers

Resources

If you want to gain some knowledge by your own...

- Wikipedia – Embedded system
  http://en.wikipedia.org/wiki/Embedded_system
- Embedded System Market – Global Industry Analysis
  http://www.prnewswire.com/
- Free Electrons - embedded Linux experts
  http://free-electrons.com/

Questions ?

Rafal Kapela
rafal.kapela@put.poznan.pl