# Introduction to Kernel

**D. Balakrishna,**
Research Associate, IIIT-H

In this article we are going to learn about Linux kernel source code and then further explore the Linux kernel and cross compile Linux kernel source code for ARM devices. Here we are using Arndale 5250 development board.

The linux kernel is the heart of Desktop system or embedded systems. The kernel is responsible for memory allocation, process and thread creation, memory management and scheduling and communication between hardware and peripheral devices. The Linux kernel is one of the components of a system, which also requires libraries and applications to provide features to end users. The initial development of Linux kernel was done by Linus Torvalds in 1990's and he has been able to create a large and dynamic developer and user community around linux. Linux can be referred as in two ways one is "Linux" and other is "GNU/Linux", the reason behind this is Linux is the kernel of an operating system and the wide range of applications that make operating system useful are the GNU software. The GNU software tools are like compiler, editors, variety of shells, utilities and other development tools are exist outside of the kernel. The figure1 shows the fundamental architecture of Linux operating system.

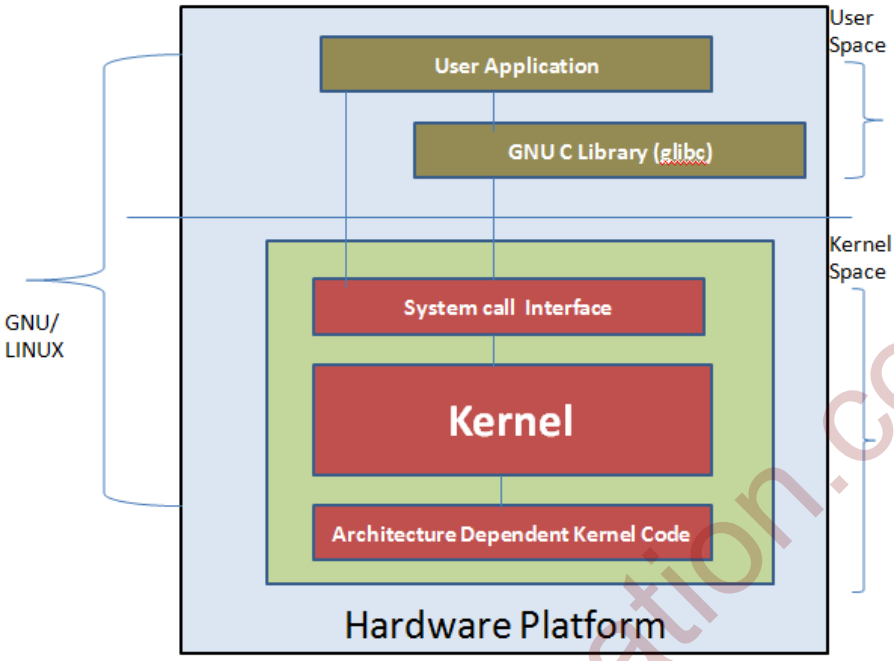



Figure1: Architecture of Linux operating system

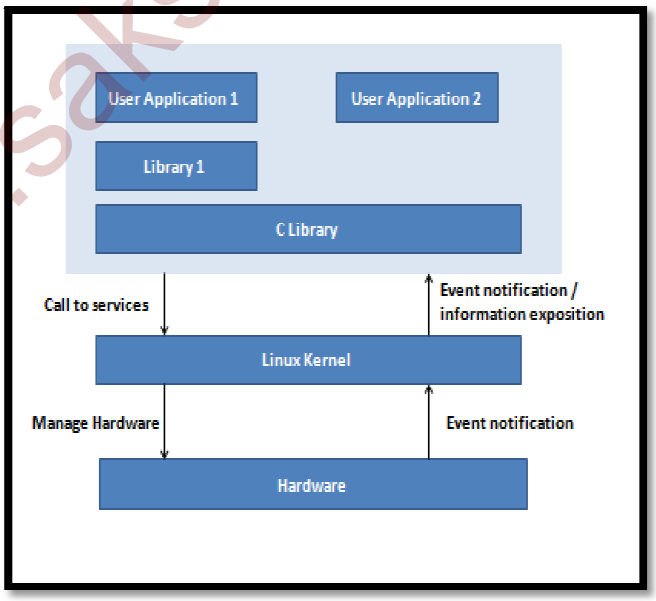
The top of the architecture is the user space. This is where the user applications are executed. Bottom of the architecture is the kernel space where the Linux kernel exists. There is also the GNU C Library (glibc). This provides the system call interface that connects user space to the kernel space and provides the mechanism to transition between the user space applications to the kernel space. This is important because the kernel and user application occupy different protected address spaces. And while each user space process occupies its own virtual address space, the kernel occupies a single address space. The Linux kernel can be further divided into three levels. At the top is the system call interface, which implements the basic functions such as read and writes. Below the system call interface is the kernel code, which can be more accurately defined as the architecture-independent kernel code. This code is common to all of the processor architectures supported by Linux. At bottom is the architecture dependent code, mostly called a BSP (Board Support Package). This code serves as the processor and platform specific code for the given architecture. Linux is also a dynamic kernel, supporting the addition and removal of software components on the fly. These are called dynamically loadable kernel modules, and they can be inserted at the boot time when they are needed or at any time by the user.

## Linux kernel key features:

- Linux can be easy to program. It is open source and many useful resources available on the internet and you can learn from existing code.
- It is easily portable and more hardware support. It runs on the most of the architecture.
- Scalability. It can run on supercomputer as well as small tiny devices.
- Stability and reliability.
- Security.
- Modularity.

The linux kernel key roles are to manage all the hardware resources like CPU, Memory and I/O. Provide a set of portable, architecture and hardware independent APIs to allow user applications and libraries to use the hardware resources and handle concurrent accesses and usage of hardware resources form different applications. The kernel information available in user space through pseudo file systems also called as virtual file system. Pseudo file systems 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 file systems are

- Proc: usually mounted on /proc – it contain operating system related information like processes and memory management parameters.
- Sysfs: usually mounted on /sys




Figuere2: Linux kernel in the system.

## Major subsystems of the Linux kernel

The Figure3 shows some major components of the Linux kernel.

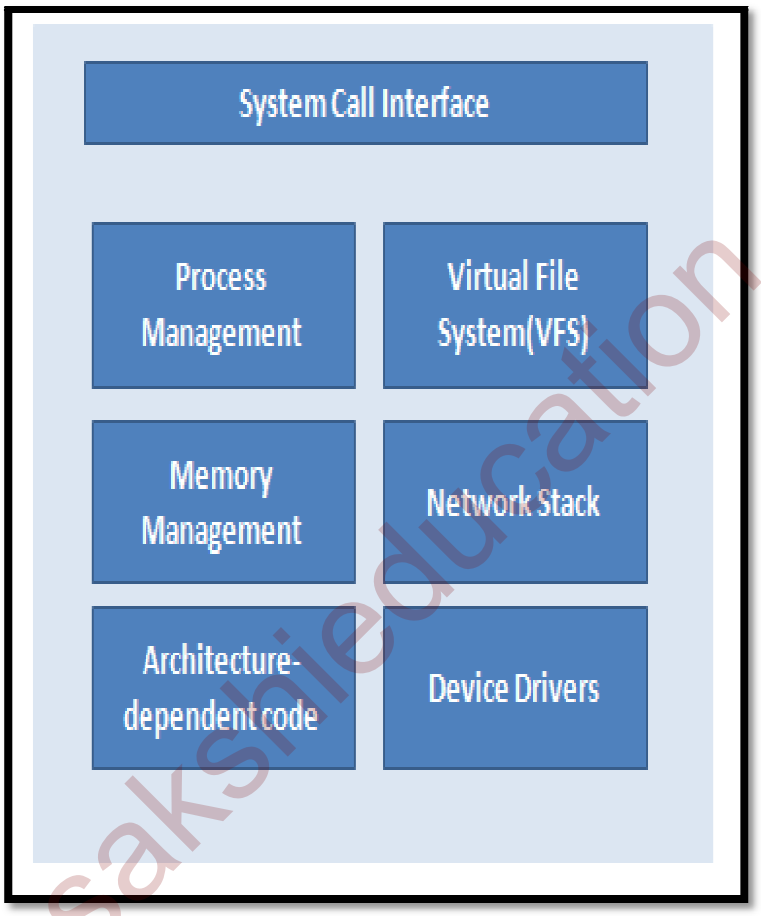



Figure3: Major components of Linux kernel

## System call interface

- It is the main interface between the user space and kernel space.
- It is architecture dependent, even within the same processor family.
- There are more than 300 system calls that provide the main kernel services, like File operations, networking operations, inter process communication, memory management, process management, threads, etc.
- It is stable over time, only new system calls to be added by the kernel developers.

- This system call interface is wrapped by the C library, and user space applications never make a system call directly but rather use the corresponding C library function.
- The system call interface implementation in ./linux/kernel and architecture dependent portions in ./linux/arch.

## Process management

- Process management is focused on the execution of processes.
- In the user space these are called process.
- In the kernel space these are called threads and represent an individual virtualization of the process like CPU registers, data segment, stack segment and thread code.
- The kernel provides an application program interface (API) through system call interface to create new process using fork, exec, POSIX (portable operating system interface) functions, to stop a process kill and exit functions and to communicate and synchronizing between them is signal and posix mechanisms.
- The process management is also responsible for sharing CPU between the active threads. The kernel implements a novel scheduling algorithm that operates in constant time, regardless of the number of threads waiting for the CPU.

## Memory management

- Linux provides abstractions over 4KB buffers, such as the slab allocator. This memory management scheme uses 4KB buffers as its base, but then allocates structures from within, keeping track of which pages are full, partially used, and empty.
- Supporting multiple users of memory, there are times when the available memory can be exhausted. For this reason, pages can be moved out of memory and onto the disk. This process is called swapping because the pages are swapped from memory onto the hard disk.
- The memory management sources in ./linux/mm.

## Virtual File System (VFS)

- The virtual file system is an important aspect in the Linux kernel; it provides a common interface abstraction for file systems. The VFS provides a switching layer between the system call interface and the file system supported by the kernel.

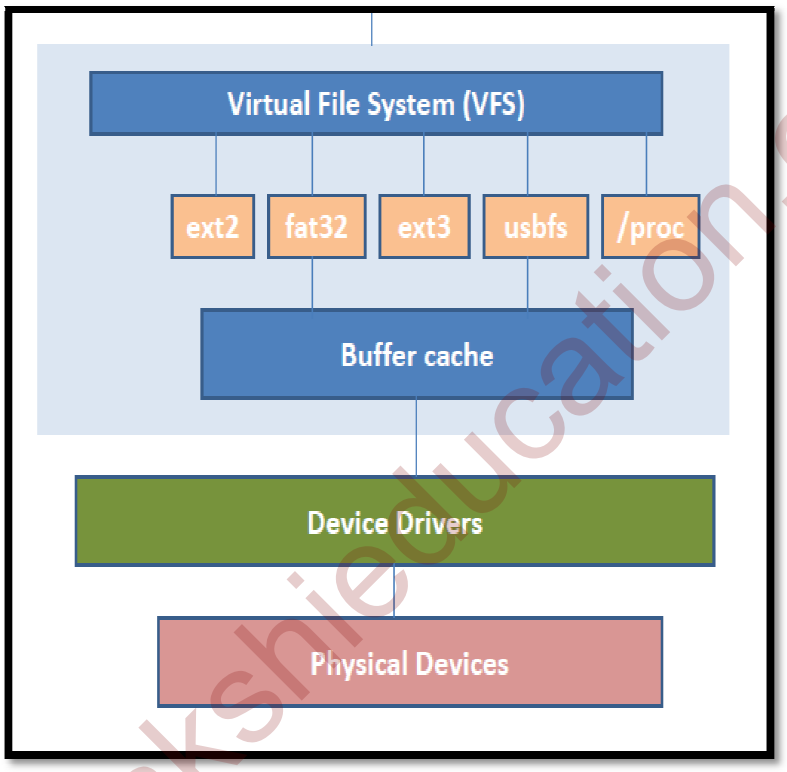



Figure4: Virtual File System

- At the top of the virtual file system is a common API abstraction of functions such as open, close, read and write.
- At the bottom of the virtual file system are the file system abstractions that define how the upper layer functions are implements.
- Below the file system layer is buffer cache which provides a common set of functions to the file system layer independent of any file system. This caching layer optimizes access to the physical devices by keeping data around for a short time.
- Below the buffer cache are the device drivers which implement the interface for the particular physical device.

- The system call interface implementation in ./linux/fs

## Network Stack

- The network stack follows a layered architecture modeled after protocols themselves. The internet protocol is the core network layer protocol that sits below the transport protocol, above transport control protocol is the socket layer which is invoked through system call interface.
- The sockets layer is the standard API to the networking subsystem and provides a user interface to a variety of networking protocols.
- The system call interface implementation in ./linux/net

## Device Drivers

- The vast majority of the source code in the Linux kernel exists in device drivers that make a particular hardware device usable.
- The Linux source tree provides a drivers subdirectory that is further divided by the various devices that are supported, such as Bluetooth, I2C, serial, and so on.
- The device driver sources implementation in ./linux/drivers.

## Supported Hardware Architectures

The ./linux/arch subdirectory defines the architecture dependent portion of the kernel source contained in a number of subdirectories that are specific to the architecture.

- 32 bit architectures with MMU and without MMU, gcc support.

  Examples: arm, avr32, mips, microblaze – (arch/subdirectories)

- 64 bit architectures

  Examples: arm64, alpha – (32/64 bit architectures)

## Linux versioning scheme and development process

The versioning system of Linux kernel tells about two types of kernel one is stable kernel which is fully developed and ready to use and it is almost bugs free, and second one is unstable source which is under development.

- Stable branch will be every 2 or 3 years. It identified by an even middle number. For example 1.0.x, 2.0.x, 3.0.x.
- Development branch is to integrate new functionalities and major changes. Identified by an odd middle number. For example 1.1.x, 2.1.x, 3.1.x.
- These development versions can be stable version after some time.

It is easy to identify what kernel version we are using. The first few lines of the top level makefile in the kernel source tree give details regarding the kernel version as shown in below.

```
VERSION = 3
PATCHLEVEL = 4
SUBLEVEL = 5
EXTRAVERSION =
NAME = Saber-toothed Squirrel
```

$ cat /proc/version – it will give the following output with kernel version.

Linux version 3.11.0-26-generic (buildd@komainu) (gcc version 4.6.3 (Ubuntu/Linaro 4.6.3-1ubuntu5) ) #45~precise1-Ubuntu SMP Tue Jul 15 04:02:35 UTC 2014

## Linux kernel source

The official versions of the Linux Kernel sources released by Linua Torvalds are available at http:/www.kernel.org/. Many chip vendors supply their own kernel sources focusing on hardware support. The kernel sources are available from http://kernel.org/pub/linux/kernel as complete kernel sources and patches. Using git tool we can download the kernel source from internet.

$ git clone git://android.git.linaro.org/device/samsung/manta.git

$ cd manta

$ git checkout -b linaro_android_4.4.2 origin/linaro_android_4.4.2

## Linux kernel size

- Linux 3.10 source
  - Raw size – 573 MB
  - gzip compressed tar archive – 105MB
  - bzip2 compressed tar archive – 83MB
  - xz compressed tar archive – 69MB
- These sources are includes so many device drivers, many protocols and support for many architectures. The linux core which memory management and scheduler are very small.
  - drivers/ -- 49.4%
  - arch/ -- 21.9%
  - fs/ -- 6.%
  - include/ -- 4.7%
  - sound/ -- 4.4%
  - documentation/ -- 4%
  - net/ -- 3.9%
  - firmware/ -- 1.0%
  - tools/ -- 0.9%
  - scripts/ -- 0.5%
  - mm/ -- 0.5%
  - lib/ -- 0.4%
  - block/ -- 0.2%
  - kernel/ -- 1.0%

Download linux kernel source and uncompress it.

http://www.kernel.org/pub/linux/kernel/v3.x/linux-3.10.9.tar.xz

tar -xvf linux-3.10.9.tar.xz

# Kernel configuration

The kernel configuration and build system is based on multiple Make files. All Makefiles inside the sub directories in kernel source interacts with the main Makefile which is present in the top directory of the kernel source tree. Interaction between all make files takes place using the make tool, which parses the Makefile, through various targets, defining which action should be done like configuration, compilation, installation, etc. The kernel contains thousands of device drivers, network protocols, file systems, other configurable devices and 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 can compile your own kernel source. The configuration for the specific target is stored in the .config file at the root of kernel source.

The .config file looks like as shown in below

```
# Automatically generated file; DO NOT EDIT.
# Linux/arm 3.8.0-rc4 Kernel Configuration
#
CONFIG_ARM=y
CONFIG_ARM_RUNTIME_PATCH=y
CONFIG_SYS_SUPPORTS_APM_EMULATION=y
CONFIG_GENERIC_GPIO=y
CONFIG_HAVE_PROC_CPU=y
CONFIG_NO_IOPORT=y
CONFIG_STACKTRACE_SUPPORT=y
CONFIG_HAVE_LATENCYTOP_SUPPORT=y
CONFIG_LOCKDEP_SUPPORT=y
CONFIG_TRACE_IRQFLAGS_SUPPORT=y
CONFIG_RWSEM_GENERIC_SPINLOCK=y
CONFIG_ARCH_HAS_CPUFREQ=y
CONFIG_GENERIC_HWEIGHT=y
CONFIG_GENERIC_CALIBRATE_DELAY=y
CONFIG_NEED_DMA_MAP_STATE=y
```

```
CONFIG_VECTORS_BASE=0xffff0000
CONFIG_ARM_PATCH_PHYS_VIRT=y
CONFIG_ARM_RUNTIME_PATCH_TEST=y
CONFIG_GENERIC_BUG=y
CONFIG_DEFCONFIG_LIST="/lib/modules/$UNAME_RELEASE/.config"
CONFIG_HAVE_IRQ_WORK=y
CONFIG_BUILDTIME_EXTABLE_SORT=y
#
# General setup
#
# CONFIG_EXPERIMENTAL is not set
CONFIG_BROKEN_ON_SMP=y
CONFIG_INIT_ENV_ARG_LIMIT=32
CONFIG_CROSS_COMPILE=""
CONFIG_LOCALVERSION=""
CONFIG_LOCALVERSION_AUTO=y
CONFIG_HAVE_KERNEL_GZIP=y
CONFIG_HAVE_KERNEL_LZMA=y
CONFIG_HAVE_KERNEL_XZ=y
CONFIG_HAVE_KERNEL_LZO=y
CONFIG_KERNEL_GZIP=y
# CONFIG_KERNEL_LZMA is not set
# CONFIG_KERNEL_XZ is not set
# CONFIG_KERNEL_LZO is not set
CONFIG_DEFAULT_HOSTNAME="(none)"
CONFIG_SWAP=y
# CONFIG_SYSVIPC is not set
# CONFIG_FHANDLE is not set
CONFIG_HAVE_GENERIC_HARDIRQS=y
# Multiple platform selection
#
```

```
# CPU Core family selection
#
# CONFIG_ARCH_MULTI_V6 is not set
CONFIG_ARCH_MULTI_V7=y
CONFIG_ARCH_MULTI_V6_V7=y
# CONFIG_ARCH_MULTI_CPU_AUTO is not set
# CONFIG_ARCH_MVEBU is not set
# CONFIG_ARCH_BCM is not set
# CONFIG_KEYBOARD_GPIO_POLLED is not set
# CONFIG_ARCH_HIGHBANK is not set
# CONFIG_ARCH_MXC is not set
# CONFIG_ARCH_SOCFPGA is not set
# CONFIG_ARCH_SUNXI is not set
CONFIG_ARCH_VEXPRESS=y
#
# Versatile Express platform type
#
CONFIG_ARCH_VEXPRESS_CORTEX_A5_A9_ERRATA=y
# CONFIG_ARCH_VEXPRESS_CA9X4 is not set
CONFIG_PLAT_VERSATILE_CLCD=y
CONFIG_PLAT_VERSATILE_SCHED_CLOCK=y
# CONFIG_ARCH_VT8500 is not set
# CONFIG_ARCH_ZYNQ is not set
CONFIG_PLAT_VERSATILE=y
CONFIG_ARM_TIMER_SP804=y
CONFIG_ARCH_FLATMEM_ENABLE=y
CONFIG_ARCH_DISCONTIGMEM_ENABLE=y
#
# Processor Type
#
CONFIG_CPU_V7=y
```

```
CONFIG_CPU_32v6K=y
CONFIG_CPU_32v7=y
CONFIG_CPU_ABRT_EV7=y
CONFIG_CPU_PABRT_V7=y
CONFIG_CPU_CACHE_V7=y
CONFIG_CPU_CACHE_VIPT=y
CONFIG_CPU_COPY_V6=y
CONFIG_CPU_TLB_V7=y
CONFIG_CPU_HAS_ASID=y
CONFIG_CPU_CP15=y
CONFIG_CPU_CP15_MMU=y
#
# Boot options
#
# CONFIG_S3C_BOOT_WATCHDOG is not set
# CONFIG_S3C_BOOT_ERROR_RESET is not set
CONFIG_S3C_BOOT_UART_FORCE_FIFO=y
CONFIG_S3C_LOWLEVEL_UART_PORT=2
CONFIG_SAMSUNG_CLKSRC=y
CONFIG_SAMSUNG_IRQ_VIC_TIMER=y
CONFIG_SAMSUNG_IRQ_UART=y
CONFIG_SAMSUNG_GPIOLIB_4BIT=y
CONFIG_S3C_GPIO_CFG_S3C24XX=y
CONFIG_S3C_GPIO_CFG_S3C64XX=y
CONFIG_S3C_GPIO_PULL_UPDOWN=y
CONFIG_S5P_GPIO_DRVSTR=y
CONFIG_SAMSUNG_GPIO_EXTRA=0
CONFIG_S3C_GPIO_SPACE=0
CONFIG_S3C_GPIO_TRACK=y
CONFIG_S3C_ADC=y
CONFIG_S3C_DEV_ADC=y
```

```
# CONFIG_S3C_DEV_ADC1 is not set
CONFIG_S3C_DEV_HSMMC2=y
CONFIG_S3C_DEV_HSMMC3=y
# CONFIG_S5P_DEV_MSHC is not set
CONFIG_S3C_DEV_HWMON=y
CONFIG_S3C_DEV_I2C1=y
CONFIG_S3C_DEV_I2C2=y
CONFIG_S3C_DEV_I2C3=y
CONFIG_S3C_DEV_I2C4=y
CONFIG_S3C_DEV_I2C5=y
CONFIG_S3C_DEV_I2C7=y
CONFIG_EXYNOS_DEV_SS_UDC=y
CONFIG_S3C_DEV_WDT=y
CONFIG_S3C_DEV_RTC=y
CONFIG_SAMSUNG_DEV_ADC=y
CONFIG_SAMSUNG_DEV_PWM=y
CONFIG_SAMSUNG_DEV_BACKLIGHT=y
CONFIG_S3C24XX_PWM=y
CONFIG_S3C_PL330_DMA=y
# CONFIG_DMA_M2M_TEST is not set
#
# MMC/SD slot setup
#
# SELECT SYNOPSYS CONTROLLER INTERFACE DRIVER
#
CONFIG_EXYNOS5_DEV_DWMCI2=y
#
# Use 8-bit bus width
# CONFIG_EXYNOS4_SDHCI_CH2_8BIT is not set
CONFIG_EXYNOS5250_ABB_WA=y
```

The kernel image is a single file, resulting from the linking of all the object files that correspond to features enabled in the configuration. This is the file loaded in memory by the bootloader and all included features are available as soon as kernel starts when there is no root file system exists. Some features can be compiled as modules like device drivers and file systems. These modules can be loaded or unloaded dynamically at run time to add or remove features to the kernel. Each module is stored as a separate file in the file system therefore access to file system is mandatory to use modules. This is not possible in the early boot procedure of the kernel, because at that time no file system is available.

There are different types of kernel options to select different features in the kernel image.

- bool option – it tells
    - true – include the feature in the kernel image
    - false – exclude the feature in the kernel image
- tristate option – it tells
    - true – include the feature in the kernel image
    - module – include the feature as kernel module
    - false – exclude the feature in the kernel image
- int option – to specify integer values
- hex option – to specify hexadecimal values
- string option – to specify string values

There are dependencies between kernels options like if want enable a network driver requires the network stack to be enabled.

**Two types dependencies.**

1. Depend on dependencies – feature A depends on feature B, in this until feature B enable the feature A not visible.
2. Select dependencies – feature A depends on feature B, in this if feature A enabled the feature B is automatically enabled.

These options typically never edited by hand but through graphical or text interfaces.

- Text interfaces
    - Make menuconfig
    - Make nconfig
    - Make config
- Graphical interfaces
    - Make xconfig
    - Make gconfig

**Make menuconfig**

Text based with colored menus and radio lists. This option allows developers to save their progress. This is useful when no graphics are available. This type interfaces available for Buildroot and busybox. To run this we need to install ncurses package (sudo apt-get install libncurses5-dev).

**Make nconfig**

Text based with colored menus and user friendly. To run this we need to install libncurses package (sudo apt-get install libcdk5-dev).

**Make config -** Plain text interface.

**Make xconfig**

It is graphical interface to configure the kernel. Easier to load configuration files and search interface option is available to look parameters. To run this we need to install librt-dev package and g++ and libqt3-mt-dev package for older kernel releases.

**Make gconfig**

GTK based graphical configuration interface. It is similar to xconfig but lacking of searching functionality. To run this we need to install libglade2-dev package.

**Make oldconfig**

Plain-text interface that updates a .config file to be compatible with the newer kernel source code. Issues warnings for configuration parameters that are no longer exist in the new kernel. Asks for values for new parameters, where as in xconfig and menuconfig set default values foe new parameters.

**Make silentoldconfig**

The silentoldconfig is the same as oldconfig except the questions answered by the .config file will not be shown.

**Make olddefconfig**

The olddefconfig is like silentoldconfig except some questions are answered by their defaults.

**Make defconfig**

The defconfig option creates a .config file that uses default settings based on the current system architecture.

Before making changes to kernel configuration settings try to take back up the old configuration file because after changing several parameters the kernel no longer works, if something goes wrong then we can use this back up config file to run the kernel.

$ cp .config .config.old

## Compiling and installing the kernel for the host system

- **Make**
  - Make command given in the main kernel source directory. If you want run multiple jobs at a time then give make –j4 – which uses four CPU cores to compile the kernel. It speed up the compilation process.
  - It generates the following files

- Vmlinx – the raw uncompressed kernel image in the ELF format, useful for debugging purposes but it cannot be booted.
- zImage, Image, bzImage, vmImage.gz images are generated in arch/arm//boot directory.
  - zImage – for ARM architecture
  - Image – uncompressed image format
  - bzImage – for x86 architecture.
  - vmImage.gz for blackfin.
- The device tree files are generated in arch/arm//boot/dtb directory for some architecture.
- All kernel modules spread over the kernel source tree as .ko files.

- **Make install**
  - It will install for the host system by default. Generally it is not used when compiling for an embedded device, as it installs files on development workstation.
  - Installs:
    - Arch/boot/vmlinuz -- compressed kernel image.
    - arch/boot/System.map – stores kernel symbol addresses.
    - /boot/config – kernel configuration

- **Make modules_install**
  - It will install for the host system by default.
  - Install all modules in /lib/modules
  - Kernel/ -- kernel object modules as .ko files
  - Modules.alias – for module loading utilities.
  - Module.dep – module dependencies.
  - Modules.symbols – which module a given symbol belongs to.

- **Make clean** – It removes all the generated files while compilation.

- **Make mrproper** – It also removes all generated files and also removes your .config file also. It is useful when switching from one architecture to other architecture.

- **Make distclean** – It also removes all generated files as well as backup and patch reject files.

## Cross compiling the kernel

To make the difference between native compilers, cross-compiler executables are prefixed by name of the target board and architecture. The CPU architecture and cross compiler prefix are defined through the ARCH and CROSS_COMPILE variables in the top level makefile.

- ARCH is the name of the architecture. It is defined by the name of the subdirectory in arch/ folder in the kernel sources.
  - Example: for ARM – ARCH=arm
- CROSS_COMPIE is the prefix of the cross compilation tool chain.
  - Example: for ARM – CROSS_COMPILE=arm-linux-gcc
- Two solutions to define ARCH and CROSS_COMPILE
  - Pass ARCH and CROSS_COMPILE on the make command line while compiling the kernel.
    - Example: make ARCH=arm CROSS_COMPILE=arm-linux-gcc
    - Problem with this is we will forget to pass these variables when you run any make command.
  - Define ARCH and CROSS_COMPILE as environment variable.
    - Export ARCH=arm
    - CROSS_COMPILE=arm-linux-gcc
    - Problem with this it works inside the current shell or command line terminal. You can overcome by this putting this settings in a file that your source every time you start working on the project. If you work on a single architecture with always same tool chain then

place this settings in. /bashrc file to make them permanent and visible for any terminal.

## Predefined configuration files

Default configuration files are available for every architecture or CPU family in arch/<arch>/configs as .config files. Run make help to find if one is available for your platform. To load default configuration file just run make old_defconfig. This will overwrite your existing .config file.

## Device Tree

Many embedded architectures have a lot of non discoverable hardware. Depending on the architecture such hardware is either described using C code directly with in the kernel or using a special hardware description language in a device tree. ARM, power PC, ARC, Micro blaze are the examples of architecture using device tree. The Device Tree is a data structure for describing hardware, rather than hard coding every detail of a device into an operating system, many aspect of the hardware can be described in a data structure that is passed to the operating system at boot time. A device tree is written by kernel developers and is compiled into a binary device tree blob passed at boot time to the kernel. There are different device tree for each board or platform supported by the kernel. It will be there arch/arm/boot.dts/board.dtb. The boot loader must load both kernel image and device tree blob in memory before starting the kernel.

## Building and installing the kernel

- **Make**
  - This command creates kernel images in /arch/<arch>/boot/ (can uImage, zImage,vmlinux,bzImage) and copy this kernel image onto target board.
  - Make dtbs creates the device tree blob in arch/<arch>/boot/dts/ and copy this file also onto target board.
- **Make install**
  - It is rarely used in embedded development, as kernel image is a single file.

- **Make modules_install**
    - It installs many modules and description files.

## Booting with U-Boot

- Latest versions of u-boot can boot the zImage binary file where as older version of u-boot require special kernel image format uImage.
- uImage is generated from zImage using mkimage tool and it also done by automatically by the kernel make uImage target.
- On some ARM platforms make uImage requires passing a LOADADDR environment variable, which indicates at which physical memory address the kernel will be executed.
- U-Boot also needs to pass a device tree blob to the kernel.
    - Load zImage or uImage at address X in memory.
    - Load board.dtb at address Y in memory
    - Start the kernel with bootz X – Y or bootm X – Y. The – in the middile indicates no initramfs.

## Kernel Command line

- The kernel behavior can be adjusted with no recompilation using the kernel command line.
- This kernel command line is passed by boot loader. In u-boot the contents of bootargs environment variable is automatically passed to the kernel.
- Built into the kernel using the CONFIG_CMDLINE option.
- The kernel command line is string that defines various arguments to the kernel.
    - It is very important for system configuration.
    - root – for root filesystem
    - Console – to print kernel messages.

## Using kernel modules

- Modules are useful to keep the kernel image size is to minimum.
- Modules make it easy to develop drivers without rebooting and reduce boot time. That is at boot time it will not initialize devices and kernel features, this will be done later.
- Some kernel modules can depend on other kernel modules which need to be loaded first.
    - Dependencies are described in /lib/modules/<kernel-version>/modules.dep. this files are generated when we run make modules_install
    - Example: usb-storage module depends on the scsi_mod, libusual and usbcore modules.

## Kernel log

When a new module is loaded the related information is available in the kernel log. The kernel keeps its messages in a circular buffer. Kernel messages are available through the "dmesg" command. Kernel log messages also displayed in the system console and can filtered using "loglevel" kernel parameter or completely disabled with "quit" kernel parameter.