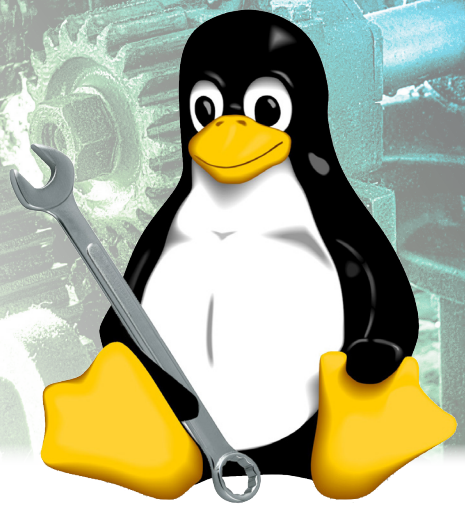
EMBEDDED IN THIN SLICES

# Linux System Configuration (Part 1)



## The Linux Kernel

This article series discusses configuring Linux for embedded systems. While the series won't include design details, it will provide information about the available possibilities. This article focuses on Linux kernel configuration.

*By Bob Japenga (US)*

One of the embedded systems my company designed is being deployed in ways in which it was not designed. I am sure that never happens to you. In fact, we specifically told customers not to install these systems in this particular way. Since you can never tell a customer he is wrong by not reading your instructions, nor can you ask him to re-install thousands of units, we were looking into ways we could survive this installation problem through software changes.

We determined that if we could come up with a way to reduce the overall power the unit draws, we may be able to mitigate the problem. Our customer's Chief Technical Officer asked if we could put the processor in Sleep mode to reduce power draw during idle times. I knew this could be done with the hardware, but I did not know if our Linux kernel was configured to do so.

Linux kernels have hundreds of parameters you can configure for your specific application. Since Sleep mode is mostly used on battery-powered units, we had not configured the kernel to support Sleep mode. Although we can update the kernel remotely, it has a high cost relative to the terms of the cellular data plan. (We are using a data plan where we can use 1 MB of data per month. The kernel update would be about 3 MB.) So because we didn't plan ahead enough early on, this solution was not as viable.

Hopefully with this article under your belt, you can be forewarned to think ahead since many of the options have very little cost in terms of memory and real-time usage. This article will examine the kinds of features that can be configured to help you think about these things during your system design. At a minimum, it is important for you to know what features you have configured if you are using an off-the-shelf Linux kernel or a Linux kernel from a reference design. Of course, as always, we will be looking at this only in thin slices.

### WHY CONFIGURE THE LINUX KERNEL?

Certainly if you are designing a board from scratch you will need to know how to configure and build the Linux kernel. However, most of us don't build a system from scratch. If we are building our own board, we still use some sort of reference design provided by the microprocessor manufacturer. We have found these to be awesome. And the reference design usually comes with a prebuilt kernel and file system.

Even if you use a reference design, you almost always change something. You use different memory chips, physical layers (PHY), or real-time clocks (RTCs). In those cases, you will need to configure the kernel to add support for these hardware devices. If you are fortunate enough to use the same hardware, the reference design's kernel may

have features you don't need and you are trying to reduce the memory footprint (which, by the way, is needed not just because of your on-board memory, but because of the over-the-air costs of updating, as I mentioned in the introduction). Or, the reference design's kernel may not have all of the software features that you want.

Let's say for example you are using an off-the-shelf Linux board (e.g., a Raspberry Pi or BeagleBoard.org's BeagleBone). It comes with everything you need, right? Not necessarily. As with the reference design, it may use too many resources and you want to trim it, or it may not have some features you want. So, whether you are using a reference design or an off-the-shelf SBC, you need to be able to configure the kernel.

## HOW IS THE KERNEL CONFIGURED?

Many things about the Linux kernel can be tweaked in real time. (This is the subject of a future article series.) However, many things (e.g., handling Sleep mode and support for new hardware) require a separate compilation and kernel build. The Linux kernel is written in the C programming language, which supports code that can be conditionally compiled into the software through what is called a preprocessor DEFINE.

A DEFINE is associated with each configurable feature. Configuring the kernel involves selecting the features you want with the associated DEFINE, recompiling, and rebuilding the kernel.

## CONFIGURING THE LINUX KERNEL

Okay, I said I wasn't going to tell you how to configure the Linux kernel, but here is a thin slice: One file contains all the DEFINES. Certainly, one could edit that file. But the classic way is to invoke `menuconfig`. Generally you would invoke this by specifying the specific architecture with the command `make ARCH=arm menuconfig`.

There are other ways to configure the kernel—such as xconfig (QT based), gconfig (GTK+ based), and nconfig (ncurses based–that are graphical and purport to be a little more user-friendly. We have not found anything unfriendly with using the classical method. And in fact, since it is terminal-based, it works well when we remotely log into the device.

**Photo 1** shows the opening screen for one of our configurations. The options are reasonably well grouped to enable you to navigate the menus. Most importantly, the mutual dependencies of the DEFINEs are built into the tool. Thus if you choose a feature that requires another to be enabled, that feature will also automatically be selected.

In addition to the out-of-the-box version, you can easily tailor all the configuration tools if you are adding your own drivers or drivers you obtain from a chip supplier. This

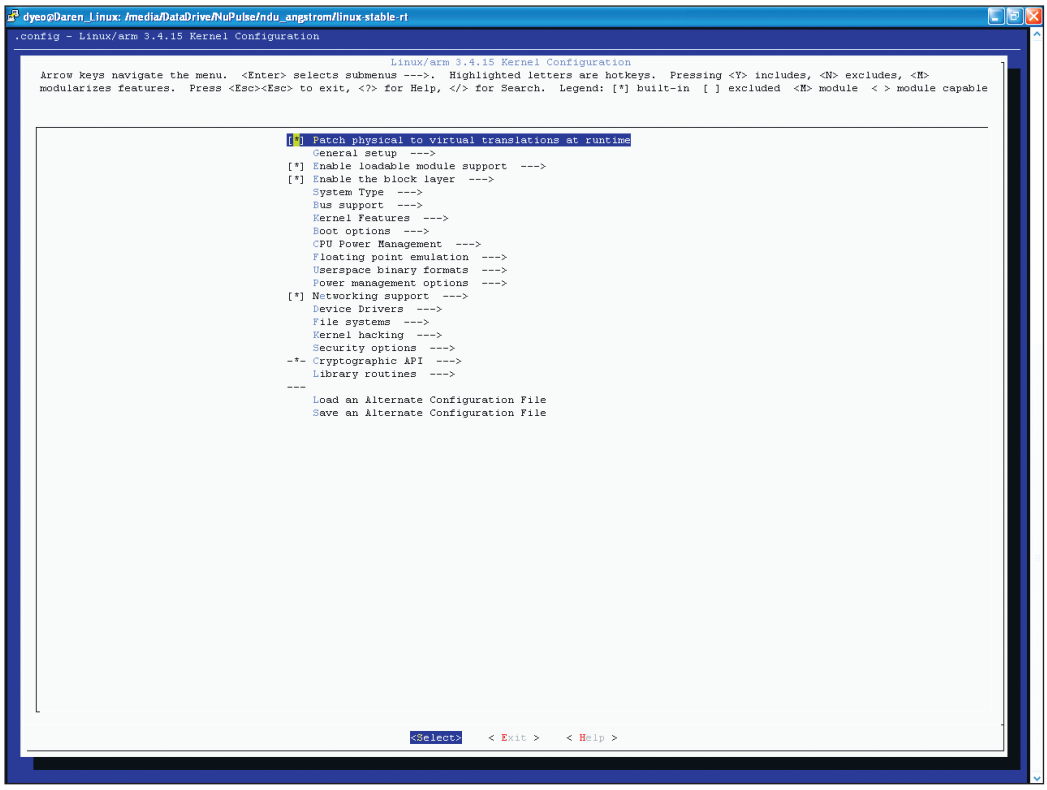



**PHOTO 1**
This opening screen includes well-grouped options for easy menu navigation.

means you can create your own menus and help system unique for your system. It is so simple; I will leave it to you to find out how to do this. The structure is defined as Kconfig, for kernel configuration (see Resources for more information).

## HARDWARE CONFIGURATIONS

As I have already mentioned, one of the chief reasons for configuring the kernel is to select the specific hardware for which you are building the kernel. It starts with the processor and or system type. For example, ARM's ARM9 family of processors, which we heavily use, has a dizzying array of options.

menuconfig walks you through those options. You can always learn more about an option by selecting the context-sensitive help available for each option. Specific features (e.g., your chipset's power management) can be configured. For example, the different level of caching on the ARM processor: I-cache (instruction) and D-cache (data). Also you can choose the kind of frequency throttling you want to take place to conserve power. (Check out how often the clock frequency is changed on your smartphone some time. The algorithms to conserve power are amazing.)

Once you have specified the processor and system type, you need to select the specific hardware drivers you want loaded. Before you do that, you should decide if you want the ability to maintain "loadable module support." If you don't choose this option, all drivers will be part of the kernel image.

Although it may seem better to keep things fixed in your embedded system, enabling modules to be loadable lets you update a driver without updating the entire kernel. We like to make every driver possible loadable to enable us to easily replace one small file rather than require a change to the kernel image. Loadable driver modules are stored on the file system, whereas embedded drivers are stored as part of the kernel image. Loadable modules can help you greatly enhance your system's features without expanding the kernel partition.

menuconfig enables you to select from an array of memory devices, real-time clocks, Ethernet PHYs, keyboards, mice, joysticks, watchdog timers, I/O expanders, ADCs, DACs, sound modules, I²C and SPI EEPROMs, 802.11 wireless options, Bluetooth options, touchscreen drivers, display drivers, and many more. Once selected, some drivers can configure certain features. For example, the serial driver may support direct memory access (DMA). This can enable you to transport data at much higher than 115.2 kbps.

When we are designing a board and need to add new hardware, we start in menuconfig to determine what hardware is already supported. Most of our projects don't give us the leisure to develop a driver from scratch. Encourage your hardware designer to become familiar with the options available from menuconfig and you will save a lot of future grief.

## SOFTWARE CONFIGURATIONS

Not only can you configure the Linux kernel with respect to the hardware, you can choose what software features you want to include. One of the most important features is the file systems that you want to support.

As I discussed in previous articles, Linux supports several flash file systems. You need to select one or more you think you will be supporting, including the network file system (NFS).

Security is an ever-increasing concern and Linux is far and away the most powerful tool we have in our arsenal to provide the kind of security our customers demand. Security algorithms are like wading in alphabet soup: ECB, LRW, PCBC, XTS, HMAC, and XCBC cyrotographic schemes are available. Various algorithms are also available for cryptography including: SHA, MD5, Blowfish, RIPEMD, AES cipher, ARC cipher, DES, triple DES, and many others. Do you need all of these? Most likely not, but they don't take a lot of resources, and you never know when you will need what part of the alphabet!

Certain system features can be configured including POSIX message queues, System V IPC (see my article, "Concurrency in Embedded Systems Part 6: POSIX FIFOs and Message Queues," *Circuit Cellar* 273, 2013), and support for triggering timers, signals, and events from file activity. Most embedded

circuitcellar.com/ccmaterials

### RESOURCES

3C Portal, "Applications and Tools for Android," www.3c71.com.

ARM,Ltd., "ARM Infocenter," http://infocenter.arm.com.

B. Japenga, "Concurrency in Embedded Systems Part 6: POSIX FIFOs and Message Queues," *Circuit Cellar* 273, 2013.

The Linux Kernel, "Documentation Extracted from the Linux Kernel and Mirrored on the Web Where Google Can Find It," www.kernel.org/doc.

Real-Time Embedded, "Working with Kconfig," www.rt-embedded.com.

SourceForge, "xconfig," http://sourceforge.net.

Wikipedia, "List of Algorithms."

### SOURCES

**ARM9 Family of microprocessors**
ARM, Ltd. | www.armcom

**BeagleBone Linux computer**
BeagleBoard.org | www.beagleboard.org

systems did not previously need to support plug-in modules. However, with the advent of USB ports, all of our embedded systems with USB ports permit hot loading of devices via USB. Point-to-Point Protocol (PPP) is another helpful option we use with all of our modems (both cell modems and dial-up modems as a configuration item).

Support for IPv6 is also a kernel option you will want to provide even if you are currently using IPv4. We are quickly running out of 32-bit IP addresses, so IPv6 will be required before we know it.

A very important feature for us in designing real-time systems is the preemption models available. Through menuconfig you can choose to use various forms of preemption. Using the full real-time control, we can write user space threads with guaranteed determinacy in the sub-millisecond range.

Last but not least, the kernel enables you to add a significant number of debugging features. In most cases these will only be used during development because of their overhead in RAM and real time. However, through configuration, you can enable kernel core dump tracing, stack dumps, and kernel-level printf support. Serial analyzers for SPI, $I^2C$, USB, and RS-232 are available, if enabled. We often instrument portions of the kernel during debug and then turn these features off prior to release for verification.

## KNOW YOUR CONFIGURATION

Even if you are using a reference design or an off-the-shelf Raspberry Pi, as a system designer, you should be aware of what extra overhead is being configured into your kernel and know what features are missing that may be needed to develop impressive systems. Hopefully this article gave you a thin slice of how to do this.


### *ABOUT THE AUTHOR*

Bob Japenga has been designing embedded systems since 1973. In 1988, along with his best friend, he started MicroTools, which specializes in creating a variety of real-time embedded systems. With a combined embedded systems experience base of more than 200 years, they love to tackle impossible problems together. Bob has been awarded 11 patents in many areas of embedded systems and motion control. You can reach him at rjapenga@microtoolsinc.com.