Introduction to embedded Linux

Date @November 25, 2024

There’s no difference between Linux we run on our computers and embedded linux apart that in an embedded distribution you cut whatever is not needed. So whatever you do on your computer can work on an embedded device with some small differences.

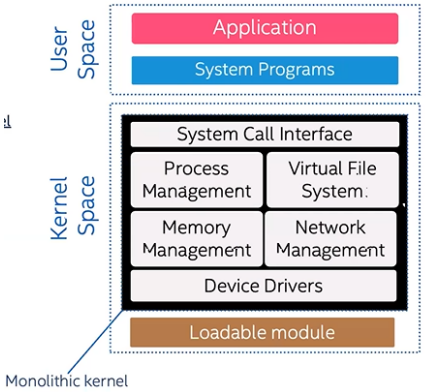
Whenever you need a complex OS in embedded applications, the big benefit of Linux is that it is open source. Linux is super flexible and currently available for any microprocessor architecture. In embedded Linux we work with the same kernel we use on computers.

Linux embedded system are used for in-flight entertainment systems, since it is simple, not weighed down by accompanying programs and easily adaptable to may environments. It is also used in bake shops or cafes, as well as in gas station pumps.

Linux kernel

There are not so many differences between Linux and OSs we’have seen so far. we have process management, process scheduling, inter-process communication, memory management, I/O management, file system, networking, and so on, as in FreeRTOS. The difference lays in the complexity.

Linux has a strict separation between in memory between user processes and OS tasks. So whatever is application runs in user mode, whatever is OS runs in kernel mode. This means we need a CPU implementing at least two execution modes and that implements memory protection. Without these two elements Linux can still potentially run but would be super slow.



Since there is distinction between user and kernel space, Linux is not a flat OS and it is a monolithic kernel. When Linux was initially designed the kernel was monolithic and everything was built in a single image of the kernel. With new versions, kernel developers decided to reproduce kernel modules which are still part of the monolithic

kernel but are modules that can be loaded at run time, so now it is not only a single kernel image.

Since the separation between kernel and user mode, in case of bugs in the user space, the kernel space is not corrupted. Furthermore, bugs in one Kernel component crashes the whole system.

Device tree

To manage hardware resources the Kernel must know which resources are available in the embedded system, such as I/O devices, memory, and so on, for the hardware description.

Two solutions are available to provide such information to the kernel:

- they are hardcoded into the kernel binary code; each modification to the hardware definition requires recompiling the source code
- they are provided the kernel by the bootloader using a binary file, the device tree blob

The DTB (device tree blob) file is produced starting from a device tree source (DTS), so that

hardware definition can be changed more easily as recompilation of the DTS is needed only and Kernel compilation is not needed upon changes to the hardware definition, and this is a bit time saver.

System programs

There are a lot of programs that you can use in Linux, which are not actually Linux but are application programs. Some of them are just programs that break system calls.

Root file system

Linux requires storing lots of information on mass storage. The Linux kernel needs a file system, called Root File System, which contains configuration files needed to prepare the execution environment for applications and the first user-level process.

The root file system can be either a portion of RAM treated as a file system (known as Initial RAM disk), a persistent storage device in the embedded system, or a persistent storage device accessed over the network.

```
/                   # Disk root
     /bin           # Repository for binary files
     /lib           # Repository for library files
     /dev           # Repository for device files
          console c 5 1  # Console device file
          null c 1 3     # Null device file
          zero c 1 5     # All-zero device file
          tty c 5 0      # Serial console device file
          tty0 c 4 0     # Serial terminal device file
          tty1 c 4 1     #
          tty2 c 4 2     #
          tty3 c 4 3     #
          tty4 c 4 4     #
          tty5 c 4 5     #
     /etc           # Repository for config files
          inittab        # The inittab
          /init.d        # Repository for init config files
               rcS  # The script run at sysinit
     /proc               # The /proc file system
     /sbin               # Repository for accessory binary files
     /tmp           # Repository for temporary files
     /var           # Repository for optional config files
     /usr           # Repository for user files
     /sys           # Repository for system service files
     /media              # Mount point for removable storage
```

Most of the system programs rely on this directory organization.

In Linux most of the applications and programs are in/bin directory, and when you install a new program this is one of the directories in which it can be installed.

Differently from FreeRTOS, we don’t have to link all the code in a single binary, but we can use external libraries in order to reuse code. External libraries are usuallly installed in the /lib directory.

/dev directory is the directory in which we have I/O devices.

/etc is the configuration directory and contains text files for configuration settings for programs.

/proc directory contains a special file system that is called the Proc file system and exposes a set of files that represent resources in your computer (the CPU model you have, how many cores there are, etc).

/sbin directory is the same as the /bin directory but for system programs.

/var is for various, when you don’t know where to put stuff you put them into /var.

/usr is for user and it is where user programs should be installed.

/sys is a special file system where you have special files connected to the services that the OS can manage.

You can find a slightly different directories organization in for different Linux distribution.