Embedded Linux Introduction

Zilogic Systems

1. Why Embedded Linux?

Pros

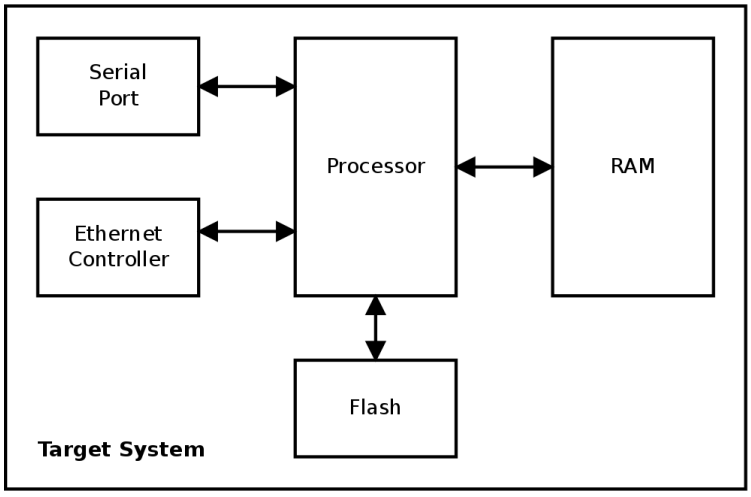
- No royalties or licensing fee.
- Support is available from the community, as well from multiple vendors. This avoids vendor lock in.
- Linux kernel supports a wide range of microprocessors and peripherals devices.
- Re-use existing Linux application base.
- Re-use existing Linux resource pool in embedded systems.

Cons

- Large memory foot print.
- The stock kernel only offers soft real-time capabilities. Third party patches are available for latency reduction and adding hard real-time capabilities.

2. Hardware Components

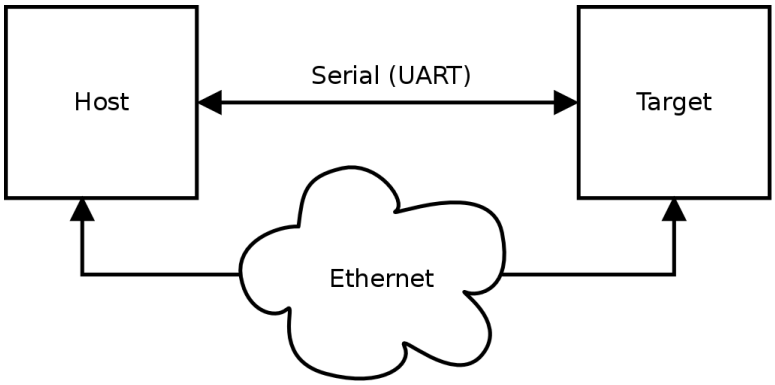
1. Processor
2. Flash
3. SDRAM
4. Serial Port
5. Ethernet



3. Development Setup

- Host System - development env.
- Target System - execution env.
- The application running in the target system read input from the serial port, and send output to the serial port.

- In other words, the standard input and standard output of the applications is attached to the serial port.
- Host system uses a serial terminal program like minicom and hyperterminal to communicate with the target system.
- Kernel image and filesystem image download is done through Ethernet interface. Serial interface is not fast enough.



4. Software Components

1. Boot Loader
2. Kernel
3. File System
4. C library
5. Shell & Utilities
6. Init & Boot Scripts

4.1. Boot Loader

- Role of Boot Loader
  1. Low-level hardware initialisation, memory controller initialisation, cache initialisation
  2. Initialisation of peripherals used by the boot loader.
  3. Provide a interface through which software can be flashed on the board.
  4. Provide a mechanism to locate, load and execute kernel present in the flash.
  5. Provides a mechanism to pass board related information and boot arguments to the kernel.
- Popular embedded boot loaders are
  - U-boot
  - Redboot, uses drivers from eCOS RTOS.

4.2. Kernel

- Role of the kernel
  1. Create higher level abstractions like files, processes, pipes, sockets, etc on top of the hardware.
  2. Manage sharing of system resources - CPU, Memory, Devices.

4.3. File System

4.3.1. File System Tree

- File - important abstraction in Linux.
- A file system tree, with all the necessary files is required for Linux system to boot.
- Import files present in the file system tree include
  - system programs
  - application programs
  - configuration files
  - device files

4.3.2. File System Format

- Specifies how files are laid out in a storage medium.
- Each file system format requires a corresponding filesystem driver.
- Various filesystem drivers are available, and depending on the requirement, the appropriate filesystem driver should be used.
  - ext2/ext3 - filesystems commonly used in desktops
  - jffs2 - filesystem tuned for Flash based memory devices
  - cramfs - crams a filesystem into a small ROM - useful for holding static data and programs
  - nfs - makes remote filesystem locally available - useful for development purposes, where a large filesystem with the required development tools can be mounted on to the board.
  - tmpfs - files are stored in RAM, and are not written to any secondary storage device. Contents are lost after a reboot. Useful for /tmp, /dev, and /var.

4.4. C Library

- C library is an integral part of a Linux system - since any user space code requires it for execution.
- Available C libraries to choose from
  - glibc - the GNU C library
  - uClibc - C library developed for embedded Linux systems - smaller than glibc, because does not support OSes other than Linux. Required features can be selected during compilation.

4.5. Shell and Utilities

- The shell is used to access and control the embedded Linux system.
- The shell and other utilities are used during system boot-up and initialisation.
- Standard shell and utilities can be cross-compiled, but the standards tools are a feature bloat and occupy a lot of precious space.
- Busybox provides many standard command line tools, trimmed down to suite embedded systems.
- Generally individual programs have separate binary files. Busybox is a single binary, containing multiple programs, each of which is invoked by executing binary with a appropriate name. This is done by creating soft links to busybox binary, one for each supported command.
- By combining multiple programs into single binary, most of the code is shared among the programs, resulting in great size optimisation.

4.6. Init & Boot Scripts

- Init is required for system initialisation, and starting/stopping of services when switching between run-levels.
- Init is available through busybox.

5. Storage

- Hard disks are not suitable for embedded systems
  - Contain moving parts
  - Sensitive to physical shock
  - Requires multiple power supply voltages
- Replaced by non-volatile memory devices - Flash
- Embedded Linux systems require > 4MB of Flash memory.

5.1. NOR Flash

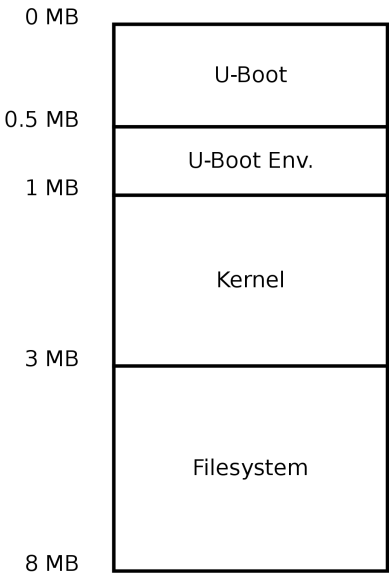
- NOR Flash is a memory mapped device, that is each storage memory location is given a unique address, just like RAM.
- Reading Flash memory is like reading any memory mapped device.
- When Flash is written to, bits in a memory location can be turned from 1 to 0, by writing to the corresponding memory location.
- Turning bits from 0 to 1, can only be done for a chunk of bytes called a block, using a special command sequence. This process of converting all bits in block to 1 is called erasing. The block is more specifically called an erase block.
- Another way to look at this is that, a proper write to a memory location can be performed only once, after a erase operation. To write to a memory location again, the entire block has to be erased.
- Typical erase block sizes are 64KB, 128KB, 256KB. Which is much larger that the typical sector size of a hard disk (512 bytes).
- Each Flash memory cell, has a max. limit on the no. of erase cycles, that it can go through. Flash degradation due to erase cycles is called memory wearing. Most Flash devices allow 100K erase cycles per memory cell.

5.2. NAND Flash

- NAND Flash has a different cell architecture, which allows Flashes with higher densities to be developed.
- NOR Flashes are typically under 64MB in size. But NAND Flashes are available in sizes of 256MB, 512MB, 1GB, 2GB ...
- NAND Flashes are not memory mapped, i.e. each memory cell does not have a separate address in the memory map. NAND Flash contents are accessed by sending commands to the controller on the chip. This is similar to how traditional hard disks are accessed.
- NAND Flashes have erase blocks like NOR Flash. And the erase blocks are further divided into pages of size about 512 bytes, 1KB, 2KB ... It is only possible to read/write **a page** of data at a time.
- NAND Flash erase blocks can become defective during operation, and the block can no longer be used for storing data. Such blocks are called bad blocks. Bad block management software is required to detect the occurrences of bad blocks, and keep track of bad blocks.

5.3. Flash Memory Usage

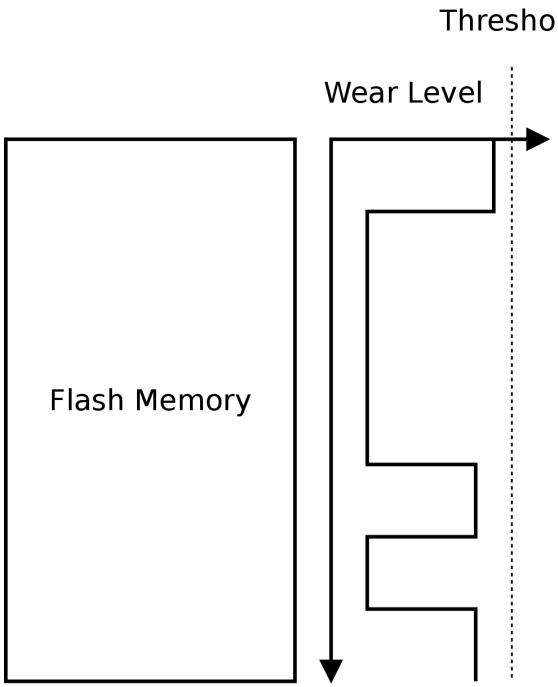
- Different software components in an embedded system can be laid out in Flash memory based on the architecture requirements.
- One way of layout of files in memory is shown in the diagram.
- Depending on the location of the reset vector the boot-loader is placed in the top or bottom of Flash.
- Configuration information required for the boot-loader is placed right next to the boot loader.
- The kernel image and the file system image are also placed appropriately in Flash.



5.4. Flash File System

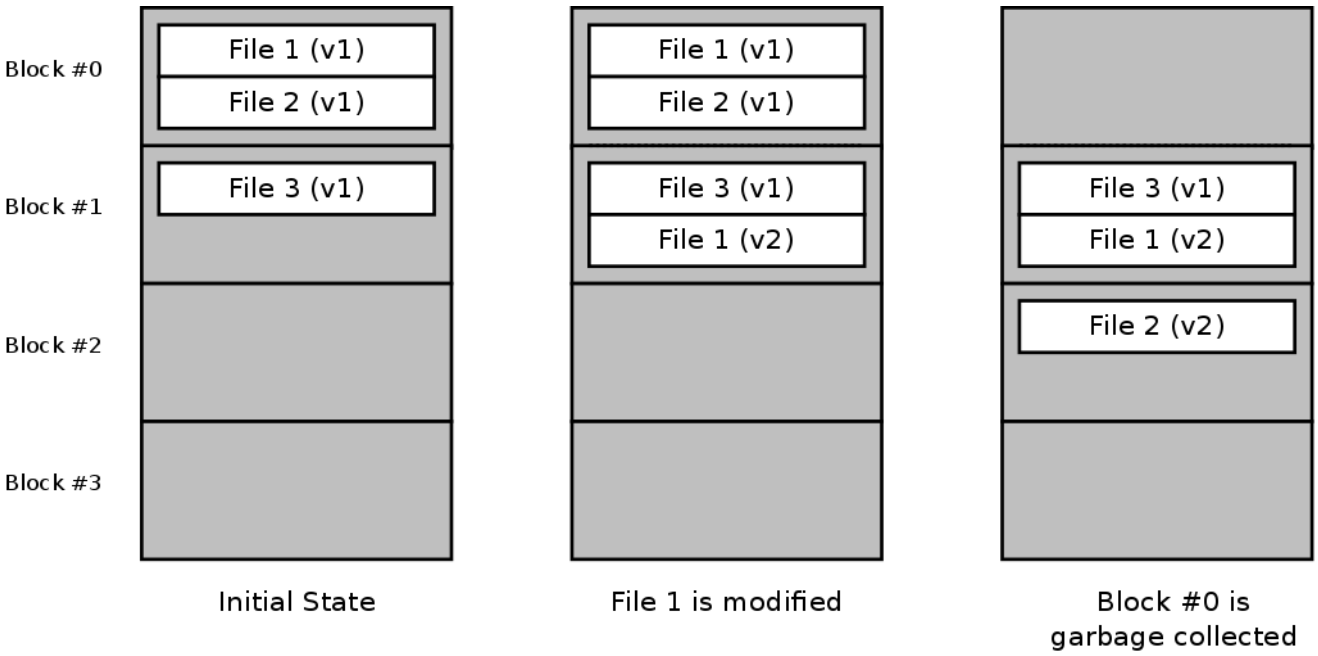
- Consider that a regular filesystem is used on a Flash device. If a file (a database file, for example) in the file system is changed frequently, that portion of the Flash will get worn out soon, reducing the effective life-time of the Flash.

Figure 1. Non-uniform Memory Wear



- If a regular filesystem is used, data loss can occur due to the large erase times of Flash devices. A typical write cycle with a regular filesystem will be `read - modify - erase - write`. But if this turns out to be `read - modify - erase - poweroff`, they the data will be lost.
- Log-structured filesystems, are used to overcome such problems in Flash devices.

Figure 2. Log-structured Filesystem



6. Development Environment Setup

- The development environment setup for a Debian Wheezy based system is provided in this section.

- For other distributions please refer the corresponding distribution documentation.

6.1. NFS Server setup

- The following packages need to be installed on the host:
  - `nfs-kernel-server`
  - `nfs-common`
  - `portmap`

```
# apt-get install nfs-kernel-server nfs-common portmap
```

- The `/etc/exports` file has the details of the clients and the corresponding folders they are permitted to access in the following format:

```
<folder-to-be-exported> <client-ip-addr>(<options>,...)
```

- The `client-ip-addr` specifies the client for which the exported folder should be accessible. If this is `*` any client can access the exported folder.
- The following options can be set
  - `rw` - Allow both read and write requests on the NFS filesystem
  - `ro` - Allow only read request on the NFS filesystem
- Add the following line to `/etc/exports` file:

```
/home/resources/tools/rootfs *(rw)
```

- Restart the `nfs-kernel-server` using the following command:

```
# /etc/init.d/nfs-kernel-server restart
```

- Now the `rootfs` folder will be accessible to the client.

6.2. TFTP server setup

- The following packages need to be installed on the host:
  - `tftpd` - Trivial File Transfer Protocol(TFTP) Server
  - `xinetd` - Enhanced Internet Superserver

```
# apt-get install tftpd xinetd
```

- Create a file `tftp` in `/etc/xinted.d` folder with the following contents:

```
service tftp
{
protocol        = udp
port            = 69
socket_type     = dgram
wait            = yes
user            = root
server          = /usr/sbin/in.tftpd
server_args     = /home/resources/tftpboot
disable         = no
}
```

- `server_args` corresponds to the directory which contains the image files for the client to access.
- `disable` has to be set to `no` for running TFTP service.
- Change the permissions of the `/home/resources/tftpboot` using the following command:

```
# chmod -R 777 /home/resources/tftpboot
```

- Restart the `xinetd` service

```
# /etc/init.d/xinetd restart
```

- Now our tftp server is up and running.

6.3. Toolchain setup

- Download the ARM Linux toolchain "arm-none-linux-gnueabi" from http://sourcery.mentor.com/public/gnu_toolchain/arm-none-linux-gnueabi/
- Multiple versions of the toolchain are available. Each version is has the following name: `arm-YYYY.MM-xx-arm-none-linux-gnueabi.bin`. Where `YYYY` is year and `MM` is the month of the toolchain release.
- Download the required version of the toolchain.
- Assign execute permissions to the downloaded binary file and run the installer.
- Follow the instructions given by the installer. After completion of installation, verify the installation using the following command:

```
$ arm-none-linux-gnueabi-gcc -v
```

- The above command will print the toolchain version.
- Download the bare metal toolchain "arm-none-eabi" from http://sourcery.mentor.com/public/gnu_toolchain/arm-none-eabi/ and repeat instructions provided for the "arm-none-linux-gnueabi" toolchain.

7. Further Reading

- Embedded Linux Applications: http://www.ibm.com/developerworks/linux/library/l-embl.html
- Embedded Linux on Wikipedia: http://en.wikipedia.org/wiki/Embedded_Linux
- Embedded Linux Introduction at Free Electrons: http://free-electrons.com/doc/training/embedded-linux/
- Flash Memory on Wikipedia: http://en.wikipedia.org/wiki/Flash_memory