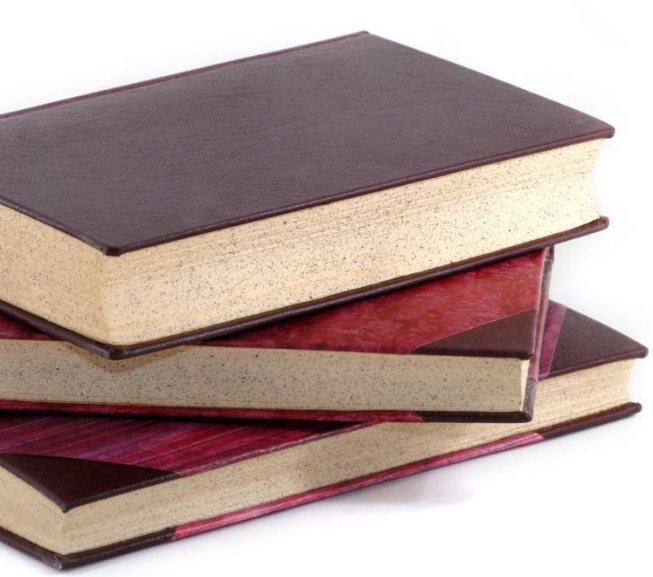
MPSIT

Lecture 2



2

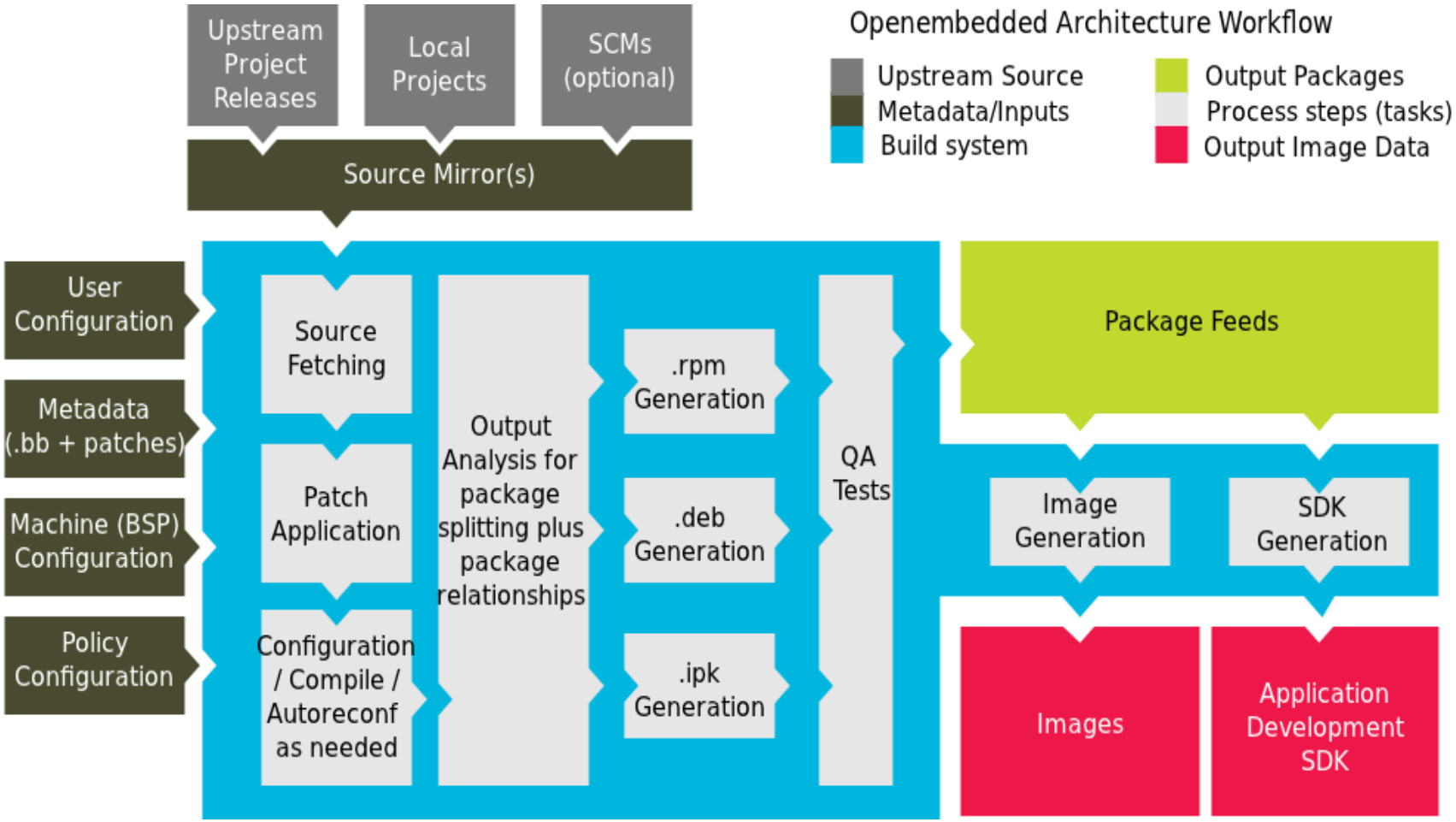
Yocto Project & Embedded Linux

18 octombrie 2016

Outline

- Cross-compiling
- Bootloaders
- Linux kernel
- Linux root filesystem
- Yocto Project

Recap: The Yocto Project



GNU Toolchain

- What does GNU toolchain means?
- Includes
  - GNU make
  - GNU Compiler Collection (GCC)
  - GNU Binutils
  - GNU Bison
  - GNU Debugger (GDB)
  - GNU m4
  - GNU build system (autotools)
    - Autoconf
    - Autoheader
    - Automake
    - Libtool

What is GNU?

- GNU is an operating system that is free software
- https://www.gnu.org/
- GNU stands for “GNU's Not Unix”
- Free software means that the software's users have freedom.
- The Free Software Foundation is the principal organizational sponsor of the GNU Operating System.
- List of maintained and developed packages available here: https://www.gnu.org/software/software.html

Binutils

- The GNU linker, that is **ld**
- The GNU assembler, that is **as**
- A utility that converts addresses into filenames and line numbers, that is **addr2line**
- A utility to create, extract, and modify archives, that is **ar**
- A tool used to listing the symbols available inside object files, that is **nm**
- Copying and translating object files, that is **objcopy**
- Displaying information from object files, that is **objdump**
- Generating an index to for the contents of an archive, that is **ranlib**
- A compiler for Windows resource files, that is **windres**
- Displaying information from any ELF format object file, that is **readelf**

Binutils (2)

- Listing the section sizes of an object or archive file, that is **size**
- Listing printable strings from files, that is **strings**
- Discarding the symbols utility that is **strip**
- Filtering or demangle encoded C++ symbols, that is **c++filt**
- Creating files that build use DLLs, that is **dlltool**
- A new, faster, ELF-only linker, which is still in beta testing, that is **gold**
- Displaying the profiling information tool, that is **gprof**
- Converting an object code into an NLM, that is **nlmconv**
- A Windows-compatible message compiler, that is **windmc**

ABI

- Binary interface between two modules: information on how functions are called and their information
- Set of rules that offer to the linker the possibility to unite compiled modules without recompilation
- Dependent on the platform
- Dependent on the programming language & compiler
- Best example: the citizen of a region/country, if they move to another region/country they will need to learn a new language.

GCC

- GNU Compiler Collection represents a compiler system
- Initially known as GNU C Compiler now also represents languages as: Objective C, Fortran, Java, Ada and Go
- Started by Richard Stallman in 1987 but it was a failure
- In 1997 a group of developers gathered as the E**xperimental/Enhanced GNU Compiler System** (**EGCS**) workgroup started merging several GCC forks in one project with great success, making EGCS the official GCC version
- They united when GCC 2.95 appeared

GCC functionality

- The frontend generates a tree from the source code
- Initially used LALR parsers (Bison generated), but moved to recursive-descendent parsers (GENERIC, GIMPLE)
- Middle stage involves code analysis and optimization, starts from GENERIC and continue to the RTL (Register Transfer Language) representation
- The backend represents preprocessor macros and specific architecture functions (endianness definition, calling convention, word size)
- In the end the machine code is obtained.

C library

- There are a number of options available:
  - glibc
  - eglibc
  - Newlib
  - bionic
  - musl
  - uClibc
  - dietlibc
  - Klibc
- The main focus will be the glibc C library

Generating the toolchain

- Toolchain build process has 8 steps
- Inside Yocto Project the toolchain is generated without notice
- Interaction with the Yocto Project generated toolchain is done calling **meta-ide-support**
- The first step is **the setup**: Create top-level directories and source subdirectories and define variables such as TARGET, SYSROOT, ARCH, COMPILER, PATH
- The second step is **the source code download**: including the above presented packages together with various patches

GNU Binutils setup

- The third step:
  - Unzip the sources available
  - Patch the sources accordingly
  - Configure the package accordingly
  - Compile the sources
  - Install the sources in the corresponding location

Linux kernel headers setup

- The fourth step:
  - Unzip the sources available
  - Patch the sources accordingly
  - Configure the kernel for the selected architecture, the corresponding kernel config file is also generated here
  - Compile the Linux kernel headers and copy them in the corresponding location
  - Install the headers in the corresponding location

Glibc headers setup

- The fifth step:
  - Unzip the glibc source and headers
  - Patch the glibc sources if this applies
  - Configure the glibc sources to the corresponding kernel headers by enabling the ***-with-headers*** variable to link the libraries with the Linux kernel headers
  - Compile the glibc headers
  - Install the headers in the corresponding location

GCC first stage setup

- The sixth step:
  - Unzip the gcc sources
  - Patch the sources accordingly
  - Configure the gcc sources enabling the necessary features
  - Compile the C runtime components
  - Install the sources in the corresponding location

Build the Glibc sources

- The seventh step:
  - Configure the glibc library by setting the corresponding ***march*** a ***mabi*** variables
  - Compile the glibc sources
  - Install the glibc in the corresponding location

GCC second stage setup

- The eighth and last step:
  - Configure the gcc sources
  - Compile the gcc sources
  - Install the binaries in the corresponding location

The Yocto Project paradigm

- **cd poky**
- **source oe-init-build-env ../build-test**
- **bitbake meta-ide-support**
- **source tmp/environment-setup**

Outline

- Cross-compiling
- Bootloaders
- Linux kernel
- Linux root filesystem
- Yocto Project

Bootloaders

- **U-Boot**: This is also called the Universal Bootloader, and is available mostly for PowerPC and ARM architectures for embedded Linux systems
- **Barebox**: This was initially known as U-Boot v2 and was started in 2007 with the scope to solve the limitations of U-Boot; it changed its name over time because the design goals and community changed
- **RedBoot**: This is a RedHat bootloader derived from eCos, an open-source real-time operating system that is portable and devised for embedded systems
- **rrload**: This is a bootloader for ARM and is based on embedded Linux systems
- **PPCBOOT**: A bootloader for PowerPC and is based on embedded Linux systems
- **CLR/OHH**: This represents a flash bootloader for embedded Linux systems based on an ARM architecture
- **Alios**: This is a bootloader that is written mostly in assembler, does ROM and RAM initializations, and tries to completely remove the need for firmware on embedded systems

U-boot

- tree -d -L 1

```
.
├── api
├── arch
├── board
├── common
├── configs
├── disk
├── doc
├── drivers
├── dts
├── examples
├── fs
├── include
├── lib
├── Licenses
├── net
├── post
├── scripts
├── test
└── tools

19 directories
```

Porting u-boot

- Create a new board directory in **board/vendor**
- Write your board specific code. It can be split across multiple headers and C files.
- Create a **Makefile** referencing your code.
- Create a configuration header file
- Create a **Kconfig** file defining at least **SYS_BOARD**, **SYS_VENDOR** and **SYS_CONFIG_NAME**
- Add a target option for your board and source your **Kconfig** either from **arch/arm/<soc>/Kconfig** or **arch/arm/Kconfig**
- Optional: create a defconfig
- Optional: create a **MAINTAINERS** file

- PREFERRED_PROVIDER_virtual/bootloader = "u-boot-at91"
- UBOOT_MACHINE ?= "sama5d3_xplained_nandflash_config"
- UBOOT_ENTRYPOINT = "0x20008000"
- UBOOT_LOADADDRESS = "0x20008000"
- AT91BOOTSTRAP_MACHINE ?= "sama5d3_xplained"

Outline

- Cross-compiling
- Bootloaders
- Linux kernel
- Linux root filesystem
- Yocto Project

Linux kernel

- 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.

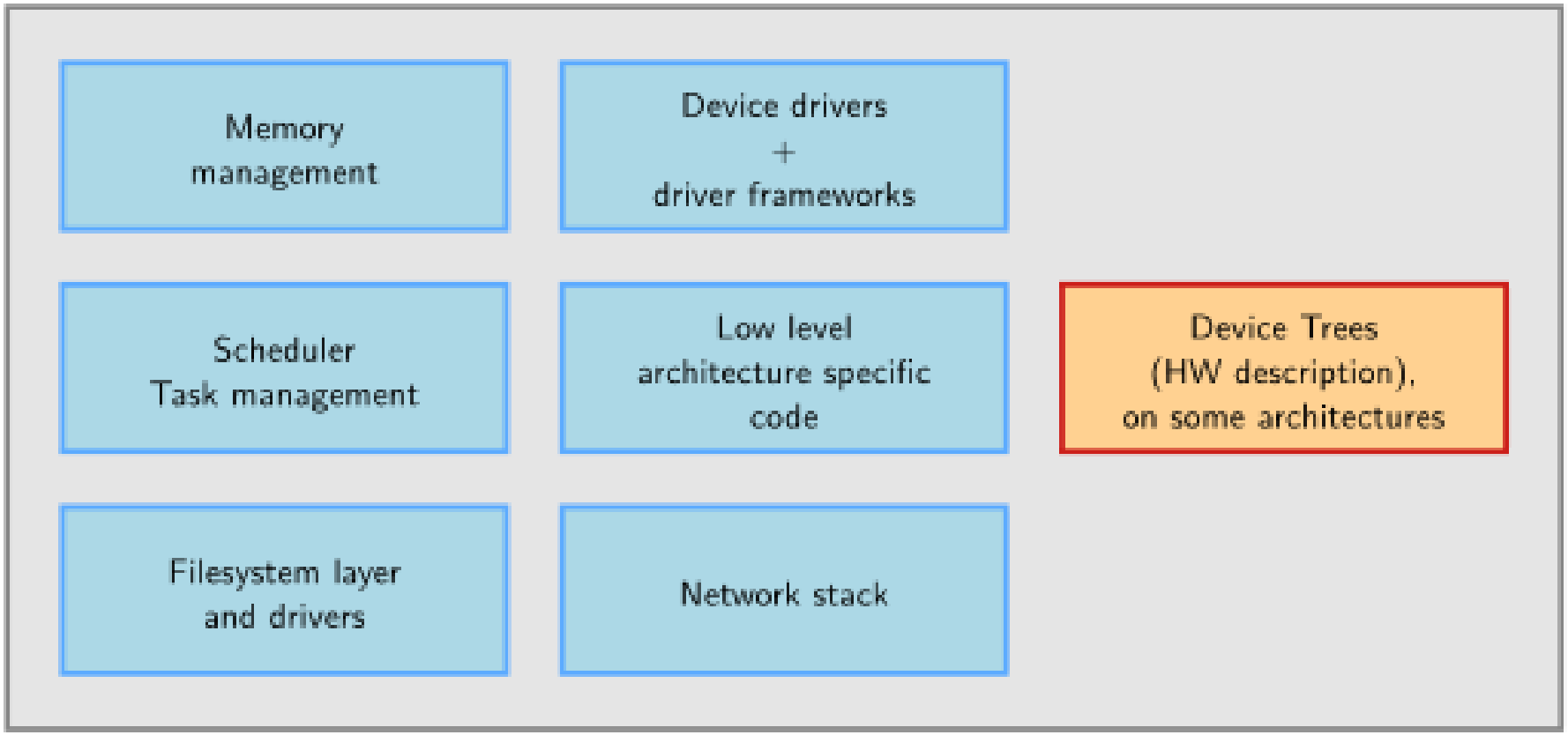
Main roles

- Manage all the hardware resources: CPU, memory, I/O.
- Provide a set of portable, architecture and hardware independent APIs to allow user space 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 user space applications through various network connections. The kernel is responsible to ``multiplex'' the hardware resource.

System calls

- The main interface between the userspace and kernel
- About 300 system calls
- The interface is stable: only new system calls can be added by the developers
- Is wrapped by the C library and user space applications which usually never make the system call directly but rather use the corresponding glibc function

Inside the Linux kernel



Linux kernel size

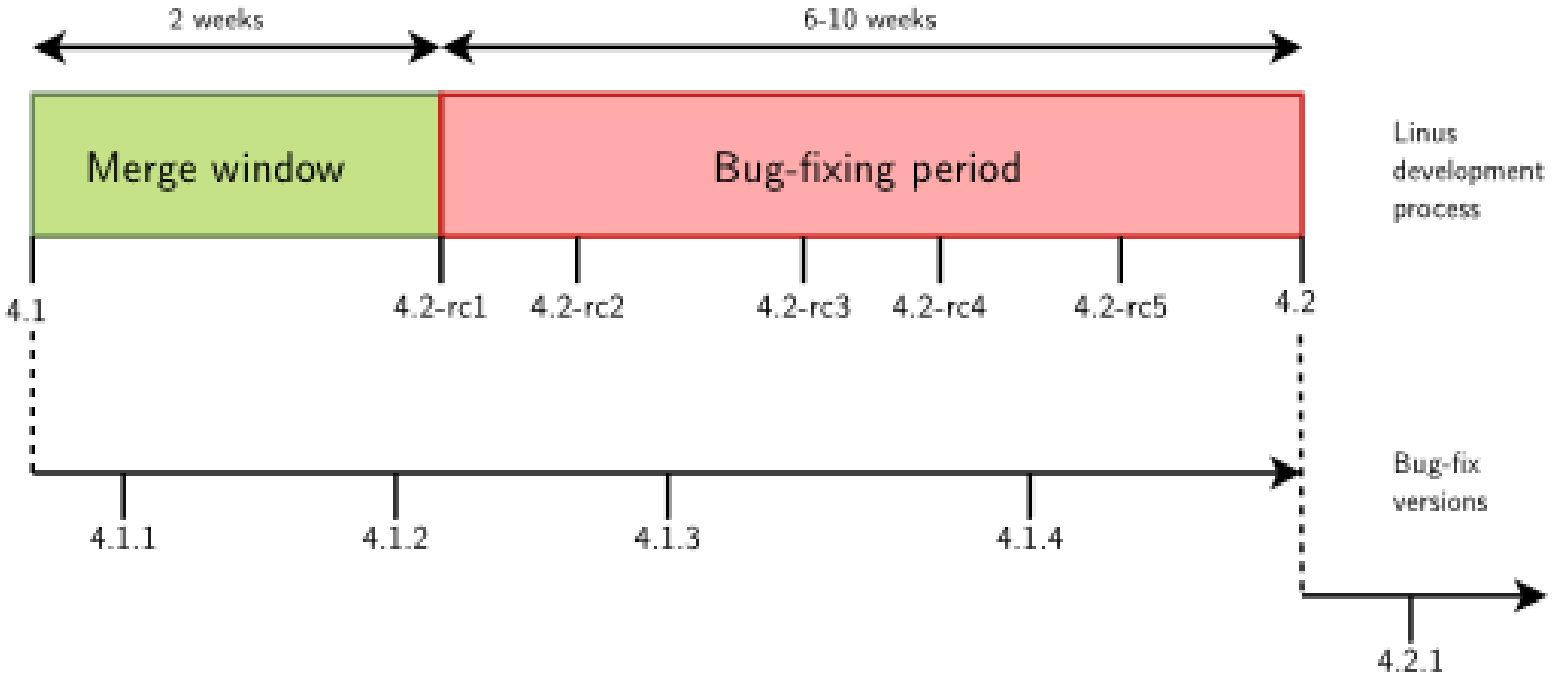
- As of kernel version 4.6 (in lines).

drivers/: 57.0%
arch/: 16.3%
fs/: 5.5%
sound/: 4.4%
net/: 4.3%
include/: 3.5%
Documentation/: 2.8%
tools/: 1.3%
kernel/: 1.2%

firmware/: 0.6%
lib/: 0.5%
mm/: 0.5%
scripts/: 0.4%
crypto/: 0.4%
security/: 0.3%
block/: 0.1%
...

New development model

- Using merge and bug fixing windows



New development model details

- After the release of **4.x** version, a two weeks merge windows opens, during which major additions are merged
- The merge window is closed by the release of test version **4.(x+1)-rc1s**
- The bug fixing period opens, for 6 to 10 weeks
- At regular intervals during the bug fixing period, **4.(x+1)-rcY** test versions are released
- When considered sufficiently stable, kernel **4.(x+1)** is released and the process starts again

- KERNEL_DEVICETREE = " at91-sama5d3_xplained.dtb “
- SERIAL_CONSOLES ?= "115200;ttyS0 115200;ttyGS0“
- SOC_FAMILY = "sama5:sama5d3“
- PREFERRED_PROVIDER_virtual/kernel_sama5 ?= "linux-at91"

Outline

- Cross-compiling
- Bootloaders
- Linux kernel
- Linux root filesystem
- Yocto Project

Filesystems

- Organize data in directories and files on network storage or a storage devices
- A single global hierarchy is used, based on FSH
- Root filesystem is identified by **/**
- The global hierarchy can be composed of multiple filesystems
- Filesystems are mounted in a specific location (called mount point)
  - The content of this directory reflects the content of the storage device
  - When the unmount operation is done the mount point is free again

Important directories

- /bin Basic programs
- /boot Kernel image
- /dev Device files
- /etc System-wide configuration
- /home Directory for users home directories and files
- /lib Basic libraries
- /media Mount point for removal media
- /mnt Mount point for static media
- /proc Mount point for the proc virtual filesystem

Important directories

- /root Home directory for the root user
- /sbin Basic system programs
- /sys Mount point for the sysfs virtual filesystem
- /tmp Temporary files
- /usr User specific files
  - /usr/bin Non-basic programs
  - /usr/lib Non-basic libraries
  - /usr/sbin Non-basic system programs
- /var System variable data files, including logging data and administrative files

Basic applications

- An **init** application which is the first userspace application started by the kernel after mounting the root filesystem
- A **shell**, to allow a user to interact with the system
  - The kernel tries to execute **/sbin/init**, **/bin/init**, **/etc/init** and **/bin/sh**.
  - If none of them is found the kernel panics and the boot process is stopped
- Basic Unix applications for file interation (commands like **mv**, **cp**, **mkdir**, **cat**, etc.)

Busybox

- In normal Linux system each of the previously presented components would be provided by a different project:
  - coreutils, bash, grep, sed, tar, wget, modutils etc.
  - A lot of components to integrate
  - Not all designed with embedded systems constrains in mind
- Busybox is an alternative solution
- Integrates all in a single project, all utilities are compiled into a single executable /bin/busybox
  - The rest of the applications are only symbolic links to it
- Really common in the embedded world

Minimal rootfs

- ldd /sbin/init
- /lib
- /bin
- /etc
- /dev

```
linux-gate.so.1 (0xb7785000)
libc.so.6 => /lib/libc.so.6 (0x4273b000)
/lib/ld-linux.so.2 (0x42716000)
```

```
lib
|-- ld-2.3.2.so
|-- ld-linux.so.2 -> ld-2.3.2.so
|-- libc-2.3.2.so
'-- libc.so.6 -> libc-2.3.2.so
```

```
bin
|-- busybox
'-- sh -> busybox
```

```
etc
'-- init.d
    '-- rcS
dev
'-- console
```

- minimal size is below 2 MB and around 80 percent of its size is due to the C library package

- meta/recipes-core/images/core-image-minimal.bb

```
SUMMARY = "A small image just capable of allowing a device to boot."

IMAGE_INSTALL = "packagegroup-core-boot ${ROOTFS_PKGMANAGE_BOOTSTRAP}
${CORE_IMAGE_EXTRA_INSTALL} ldd"

IMAGE_LINGUAS = " "

LICENSE = "MIT"

inherit core-image

IMAGE_ROOTFS_SIZE ?= "8192"
```

- bitbake core-image-minimal

Things to remember

- Embedded Linux is easier with Yocto Project
- Linux is easier to standardize

also

- Test next lecture: will cover the first two lectures
- End of lecture 3 deadline for project selection
  - Recommended the use of a versioning system
  - Integration with Yocto Project is a plus
  - Personal ideas/project are appreciated

The End