EMBEDDED OPERATING SYSTEMS

Embedded Linux on Beaglebone Black

Embedded Linux

- An Embedded Linux project involves
  - Obtaining, customizing and deploying 4 elements
    - Toolchain
      - Compiler and tools needed to create binaries from source
    - Bootloader
      - Pre-boot initialization of system and handover to OS
    - Kernel
      - The heart of the system, managing resources and hardware
    - Root file-system
      - Contains libraries and programs run after the kernel “boots”
      - Also holds the kernel modules

Embedded Linux: Toolchain

- We are using
  - GNU toolchain
    - For ARM EABI
    - with HF support
  - CROSS_COMPILE prefix
    - arm-linux-gnueabihf-

- Install this on Ubuntu using

  $ sudo apt-get install arm-linux-eabihf-gcc

Embedded Linux: Bootloader

- We are using U-Boot (v2021.01 branch)

  $ git clone https://git.denx.de/u-boot.git --depth=1 -b v2021.01
  $ cd u-boot
  *$ make **am335x_evm_defconfig***
  $ make CROSS_COMPILE=arm-linux-gnueabihf- -j 4

- This compiles U-Boot from source; binaries we will use
  - *MLO*
  - *u-boot.img*

- Format a microSD card by using ***bb-sdcard/format-sdcard.sh***
- Copy ***MLO*** and ***u-boot.img*** to 64M FAT32 partition (named ***boot***)
- Copy ***bb-boot/uEnv.txt*** to this partition
  - Note that it needs a kernel and a DTB (device tree binary)

Ubuntu: Utils needed

- For kernel configuration and compilation
  - ncurses, bison (for menuconfig)

    $ sudo apt-get install ncurses5-dev bison
  - lz4 (for compression)

    $ sudo apt-get install lz4
  - libssl-dev for ssh headers

    $ sudo apt-get install libssl-dev
  - u-boot-tools (for mkimage)

    $ sudo apt-get install u-boot-tools

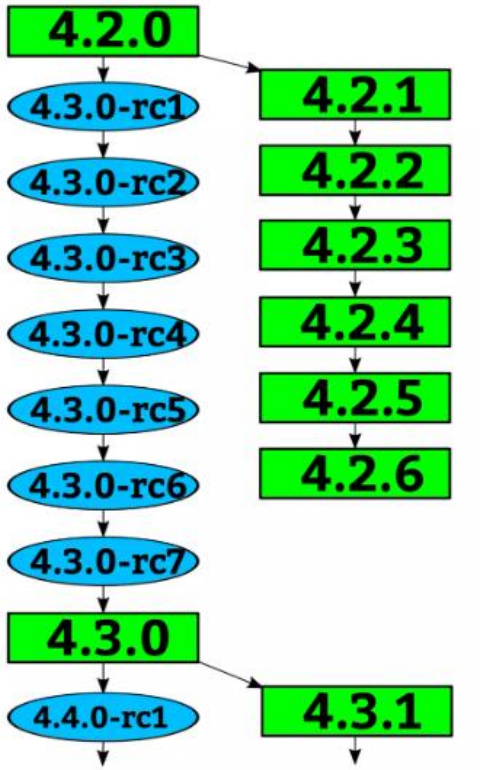
Embedded Linux: Kernel

- Linux kernel – popular open-source project
  - Generic kernel: https://kernel.org
- SCM using ***git***
- Kernel versioning
  - New kernel version every 8-12 weeks
  - Refer this link for detailed explanation by ***greg-kh***
  - Schema
    - *<Major number>.<Minor number>.<Subminor number>*
      - *Example: 5.19.8*
    - *Followed by "localversion"*
      - *User created strings to differentiate*

Kernel versioning (1/2)

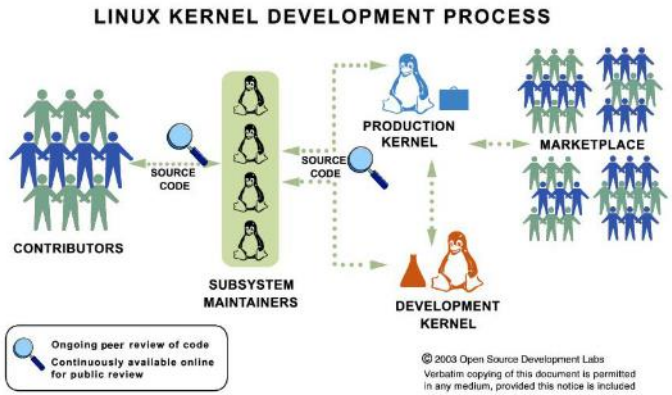
- Say we are at **v4.2.0**
  - Development starts for **v4.3.0**
    - In the form of rc's (Release Candidates)
  - In parallel, v4.2.0 feature additions
    - Give rise to sub-versions (**v4.2.x**)

- But what about stability?
  - We have branches
    - -stable
      - Linus Torvalds controls this
    - -next
      - Andrew Morton is responsible

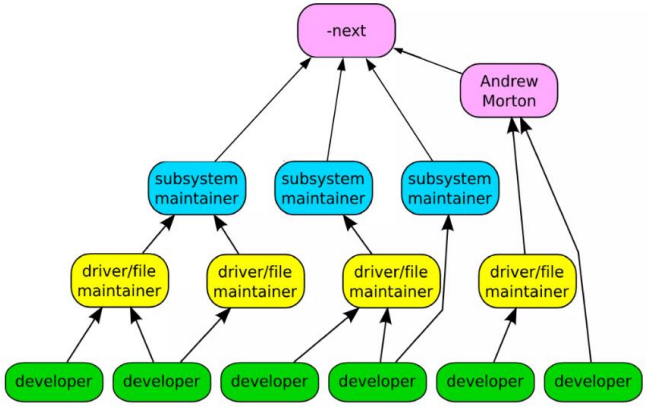


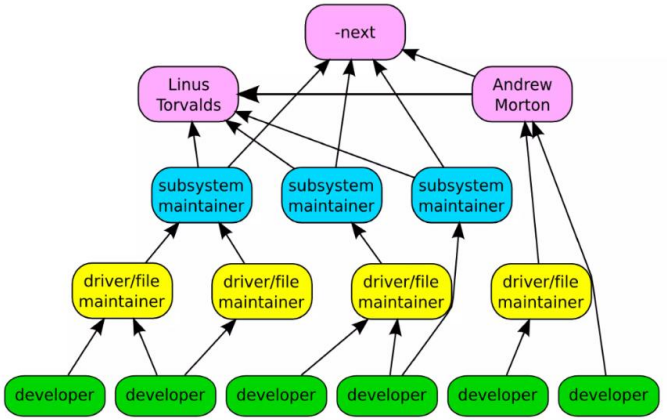
Kernel versioning (2/2)

- Gate-keeping hierarchy



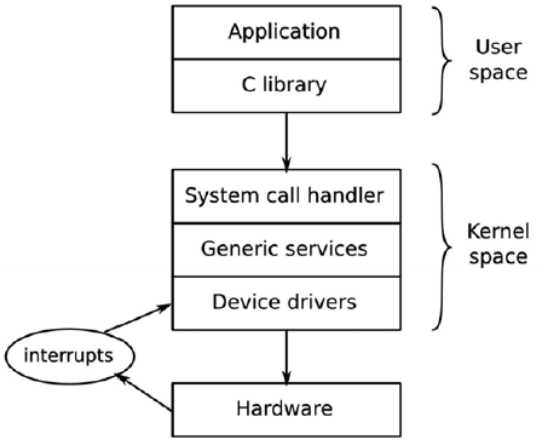
- Branch maintenance





The role of the kernel

- Kernel roles
  - Initialize system, control hardware, handle interrupts
  - Operates in **kernel space**
- Application
  - Perform user-defined tasks
  - Operates in **user space / userland**
- Kernel space – User space
  - Bridged by
    - C library, System call interface



Types of Linux-based systems

- Linux-based systems are combination of
  - Kernel
  - User-space framework

- Examples
  - Kernel + GNU user-space = GNU/Linux system
    - Ubuntu, Fedora, RedHat Enterprise Linux (RHEL)
    - Called Linux distribution (distro)
  - Kernel + Android user-space
    - Mobile / Smartphone OS
  - Kernel + custom user-space
    - Most embedded systems

The kernel sources

- Directories in the kernel source tree
  - arch – *CPU architecture specific*
  - Documentation
  - drivers – *device drivers for hardware*
  - fs - *filesystems*
  - include – *include headers for kernel*
  - init – *initialization code*
  - kernel – *kernel proper, scheduler, lock, timers, etc.*
  - mm – *memory management*
  - net – *network stack*
  - scripts – *dtc scripts, etc.*
  - tools – *perf tools, etc.*

Compiling kernel for BBB

- BBB kernel: https://github.com/beagleboard/linux.git
- Commands:

  *$ git clone https://github.com/beagleboard/linux.git -b **5.10.168-ti-r72** --depth=1*
  *$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- **bb.org_defconfig***

  $ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
  *# **to set local version***

  $ make ARCH=arm kernelversion

  *$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- **uImage** dtbs LOADADDR=0x82000000 –j 6*

- Clean up if you made a mistake

  *$ make mrproper # **cleaning up gracefully***

Adding kernel artifacts to system

- Kernel compile output artifacts *(/arch/arm/boot)*
  - vmlinux
    - *ELF binary*
  - vmlinuz
    - *compressed ELF binary*
  - Image
    - *vmlinux in raw binary form*
  - uImage
    - *uncompressed kernel for U-Boot*
  - zImage
    - *compressed kernel for U-Boot*

- Copy kernel artifacts to microSD card (to */boot* partition)
  - */arch/arm/boot/uImage*
  - */arch/arm/boot/dts/am335x-boneblack.dtb*

Booting the system

- Try booting the system
  - U-Boot SPL comes up
  - U-Boot TPL comes up
    - Displays hardware details
    - Loads kernel and dtb from microSD card to RAM
    - Starts the kernel
  - Kernel boots
    - Detects and configures resources
    - Hangs! Panics!
      - Doesn’t find a root file system (***root=mmcblk0p2***)

Kernel command line

- U-Boot calls the kernel with a command line

- Options:
  - console=<console-device>
    - *Eg. console=ttyS0,115200n8*
  - init=<init-program>
    - *Eg. init=/sbin/init*
  - root=<rootfs-device>
    - *Eg. root=/dev/mmcblk0p2*
  - ro / rw (read-only / readwrite)
  - rootwait (wait for rootfs to be ready)
  - quiet

Embedded Linux: Root FS

- The ‘bulkiest’ part of the Embedded system!

- Contains:
  - init – *the first program*
  - shell – *user-interactive prompt*
  - Daemons – *background services*
  - Shared libraries – *needed for applications*
  - Config files – *define system/application behaviour*
  - Device nodes – *special files for devices on FS*
  - /proc and /sys – *kernel data structures*
  - Kernel modules – *additional runtime object code*

Rootfs structure (FHS)

- Rootfs follows the following directory structure
  - `/bin`: Programs essential for all users
  - `/dev`: Device nodes and other special files
  - `/etc`: System configuration files
  - `/lib`: Essential shared libraries, for example, those that make up the C library
  - `/proc`: Information about processes represented as virtual files
  - `/sbin`: Programs essential to the system administrator
  - `/sys`: Information about devices and their drivers represented as virtual files
  - `/tmp`: A place to put temporary or volatile files
  - `/usr`: Additional programs, libraries, and system administrator utilities, in the `/usr/bin`, `/usr/lib`, and `/usr/sbin` directories, respectively
  - `/var`: A hierarchy of files and directories that may be modified at runtime, for example, log messages, some of which must be retained after boot

Minimal rootfs – busybox

- What is **busybox** ?
  - Software suite
  - Provides multiple utils in a single executable
  - Used in embedded systems to reduce size

- Building busybox for BB

  $ git clone git://busybox.net/busybox.git --depth=1

  $ cd busybox; make defconfig

  *$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig # **select static binary***

  $ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -j 6

Creating minimal rootfs

- ‘Install’ busybox

  *$ sudo make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- CONFIG_PREFIX=**/path/to/rootfs** install*

- Create other directories and files
  - /lib (and glibc shared libs)
  - /dev
  - /proc , /sys (kernel FS)
  - /etc/inittab, /etc/fstab, /etc/hostname, /etc/passwd

- This system when booted gives a minimal root shell

Commands for minimal rootfs

```
$ mkdir dev etc lib usr/lib proc sys root
$ mknod dev/console c 5 1
$ mknod dev/null c 1 3
$ mknod dev/zero c 1 5
$ rsync -a /usr/arm-linux-gnueabihf/lib/ ./lib/
$ rsync -a /usr/arm-linux-gnueabihf/lib/ ./usr/lib/

$ echo "cdac-eos" > etc/hostname
$ echo "root::0:0:root:/root:/bin/sh" > etc/passwd
```

```
$ cat <<EOF1 >> etc/inittab
null::sysinit:/bin/mount -a
null::sysinit:/bin/hostname -F /etc/hostname
null::respawn:/bin/cttyhack /bin/login root
null::restart:/sbin/reboot
EOF1

$ cat <<EOF2 >> etc/fstab
proc  /proc proc  defaults  0 0
sysfs /sys  sysfs defaults  0 0
EOF2
```

The init program

- Linux boot process:
  - Kernel comes up – detects most hardware
  - Mounts the root filesystem
    - Initrd / initramfs / final rootfs
  - Seeks an ‘init’ program on the root filesystem
    - This is the first *user space* program run by Linux
      - Options
        - /init
        - /bin/init
        - /sbin/init
    - Can be changed on the kernel command line (***init=***)
  - The ‘init’ program then starts services and shell

A new init system: systemd

- A software suite providing an array of system components for Linux
  - System and service manager
  - Enables aggressive parallelization options
  - On-demand daemon starts
- Aims to unify service configuration and system behavior across Linux distros
- Components
  - systemd – *service manager*
  - systemctl – *inspect and control state of services*
  - systemd-analyze – *determine boot performance and stats*
  - journald – *logging utils*
  - udevd – *framework for device management*

Better rootfs creation methods

- Minimal rootfs does not serve all purposes
  - Many utils and drivers are missing

- There are better frameworks to create rootfs
  - LTIB (Linux Target Image Builder) – *old, deprecated*
  - Buildroot – *good for embedded systems*
  - Yocto – *complex, powerful*

THANK YOU!