#LEARN_IN DEPTH

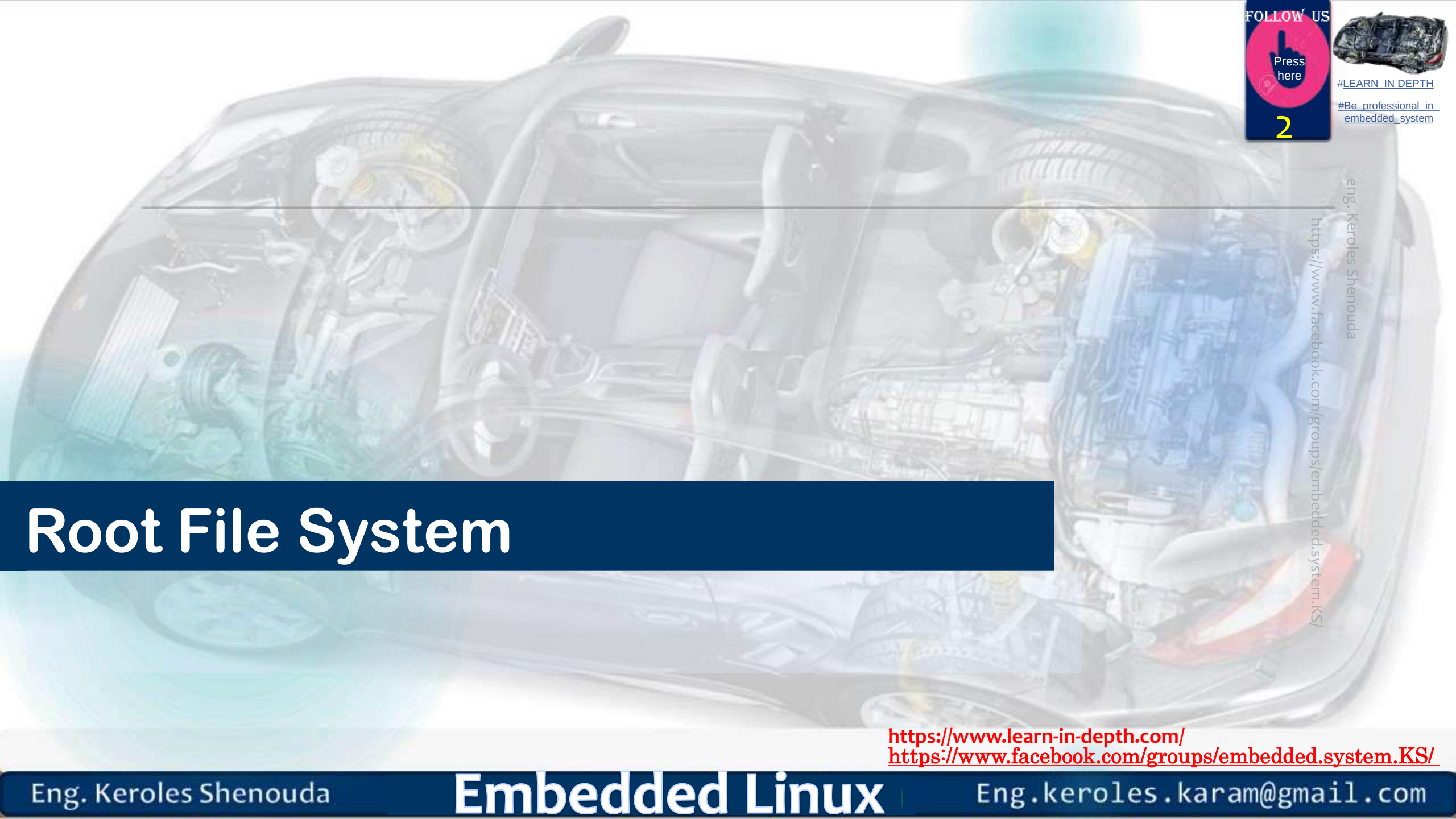
#Be_professional_in embedded_system

Embedded Linux (PART 10)

- ROOT FILE SYSTEM
- FILE SYSTEM CONTENTS
- HOW TO CREATE A MINIMAL ROOT FILESYSTEM ?
- THE INIT PROCESS
- BUSYBOX
- FILE SYSTEM TYPES
- BUILD ROOT
- THIRD PROJECT
- PART9 APPENDIX

ENG.KEROLES SHENOUDA

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

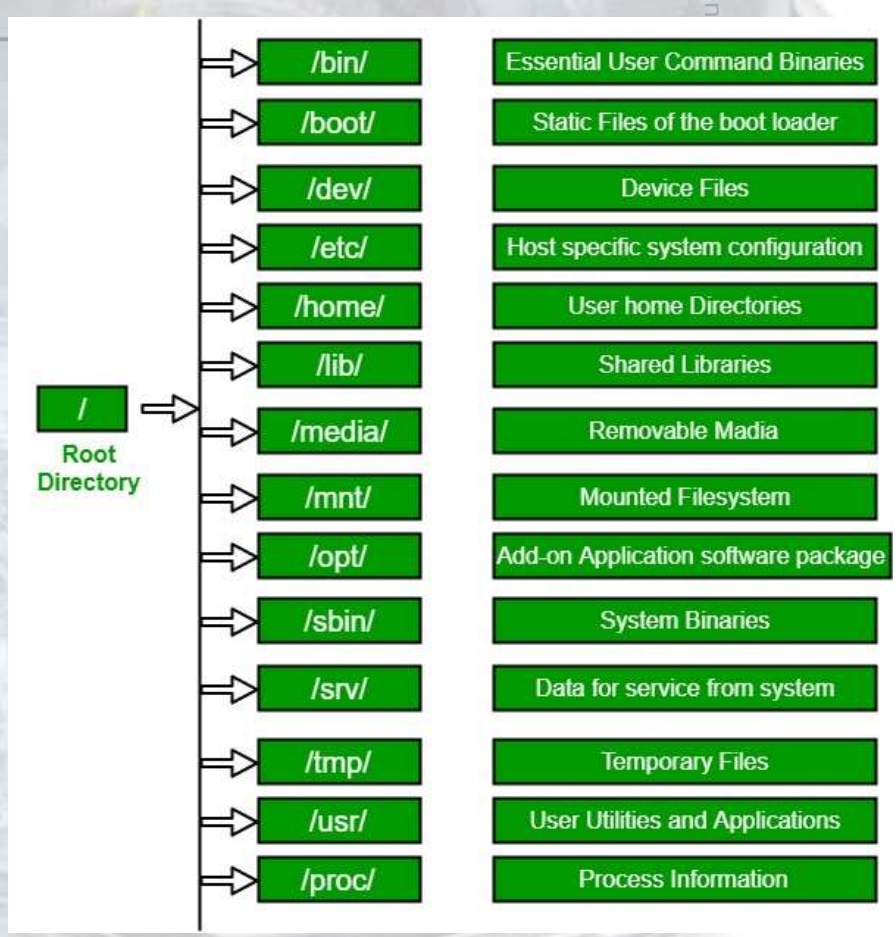
#Be_professional_in_embedded_system

Root File System

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Root File System

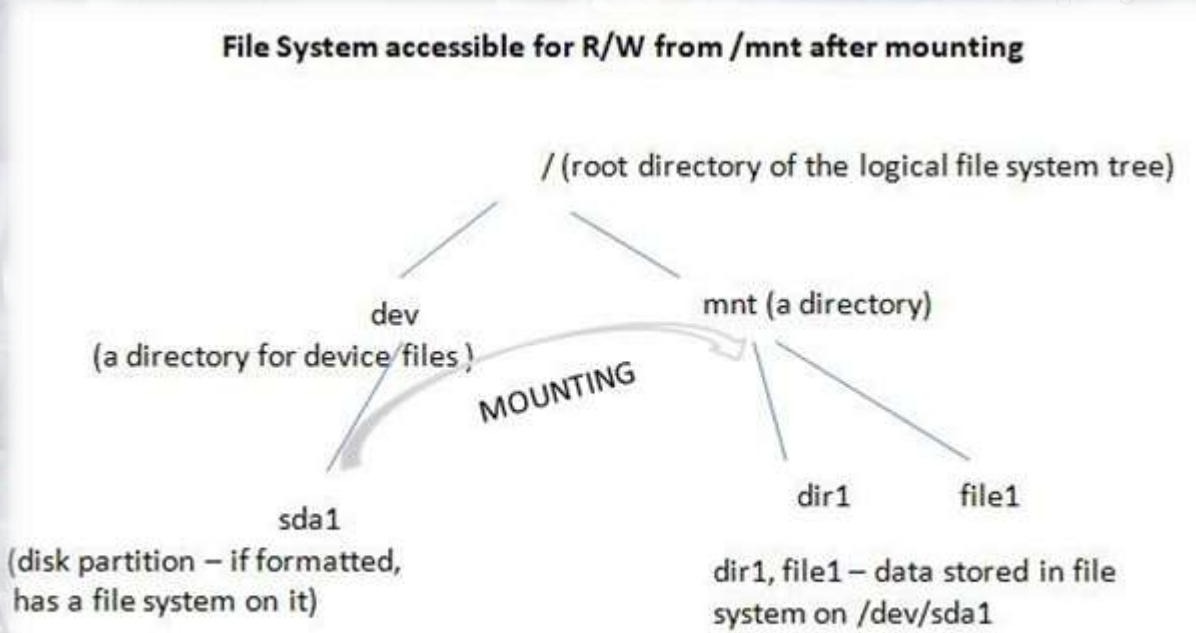
- File system is **an approach** on how the data can be organized in order to have a meaningful read or write in a system
- The method used to manage these groups of data can be called as **"File systems"**
- The most commonly used media to store the data are
  - Storage Devices
    - Hard Drive
    - Optical Discs
    - Flash Memories
    - RAM (Temporary)
  - Network like NFS



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Filesystems

- Filesystems are **mounted** in a specific location in this hierarchy of directories
  - When a filesystem is mounted in a directory (called ***mount point***), the contents of this directory reflects the contents of **the storage device**
- When the filesystem is unmounted, the ***mount point*** is empty again.
- This allows applications to access files and directories easily, regardless of their exact storage location

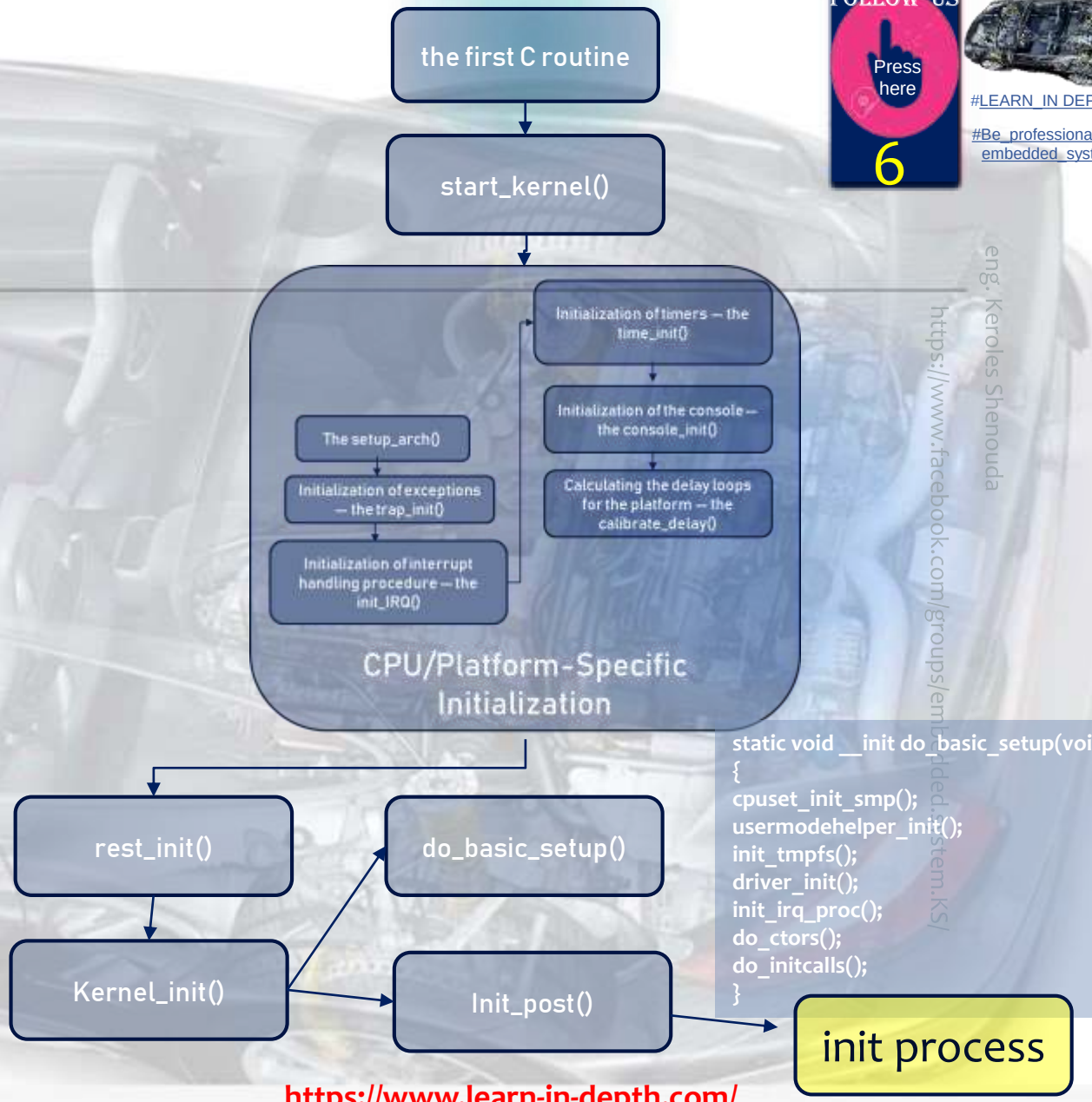


mount / umount

- **mount** allows to mount filesystems
  - mount -t type device mountpoint
  - type is the type of filesystem (optional for non-virtual filesystems)
  - device is the storage device, or network location to mount
  - mountpoint is the directory where files of the storage device or network location will be accessible
  - mount with no arguments shows the currently mounted filesystems
- **umount** allows to unmount filesystems
  - This is needed before rebooting, or before unplugging a USB key, because the Linux kernel caches writes in memory to increase performance. umount makes sure that these writes are committed to the storage.

Root filesystem

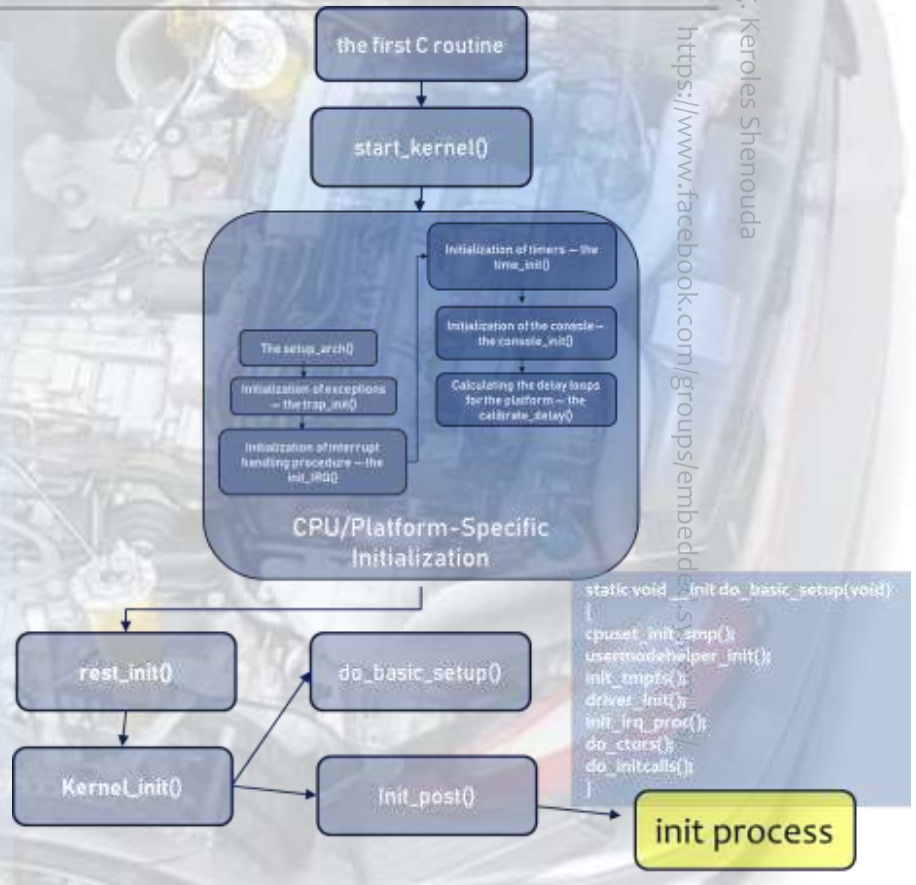
- As the root filesystem is the **first** mounted filesystem, it cannot be mounted with the normal mount command
  - It is mounted directly by the kernel, according to the **root=** kernel option
  - When no root filesystem is available, the kernel panics
- Please append a correct "root=" boot option
- Kernel panic - not syncing: VFS: Unable to mount root fs on unknown block(0,0)



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Kernel initialization

- The bootloader initializes the processor and board, and **uncompresses** the kernel code to RAM, and calls the kernel's start_kernel() function.
- Copies the command line from the bootloader
- Identifies the processor and machine.
- Initializes the console.
- Initializes kernel services (memory allocation, scheduling, file cache...)
- Creates a new kernel thread (future init process) and continues in the idle loop
- Initializes devices and execute initcalls



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Location of the root filesystem

- It can be mounted from different locations
  - From the partition of a **hard disk**
  - From the partition of a **USB key**
  - From the partition of an **SD card**
  - From the partition of a **NAND flash** chip or similar type of storage device
  - From the network, using the **NFS protocol**
  - From memory (**initramfs**), using a pre-loaded filesystem (by the bootloader) etc.
- It is up to the system designer to choose the configuration for the system, and configure the kernel behavior with root=

or

- root=/dev/sdXY, where X is a letter indicating the device, and Y a number indicating the partition
- /dev/sdb2 is the second partition of the second disk drive (either USB key or ATA hard drive)

- root=/dev/mmcblkXpY, where X is a number indicating the device and Y a number indicating the partition
- /dev/mmcblk0p2 is the second partition of the first device

- root=/dev/mtdblockX, where X is the partition number
- /dev/mtdblock3 is the fourth partition of a NAND flash chip (if only one NAND flash chip is present)

```
setenv bootargs console=${console} root=/dev/nfs rootfstype=nfs
ip=dhcp nfsroot=10.0.0.20:/srv/nfs/test_fileystem
```

```
root=/dev/ram0
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

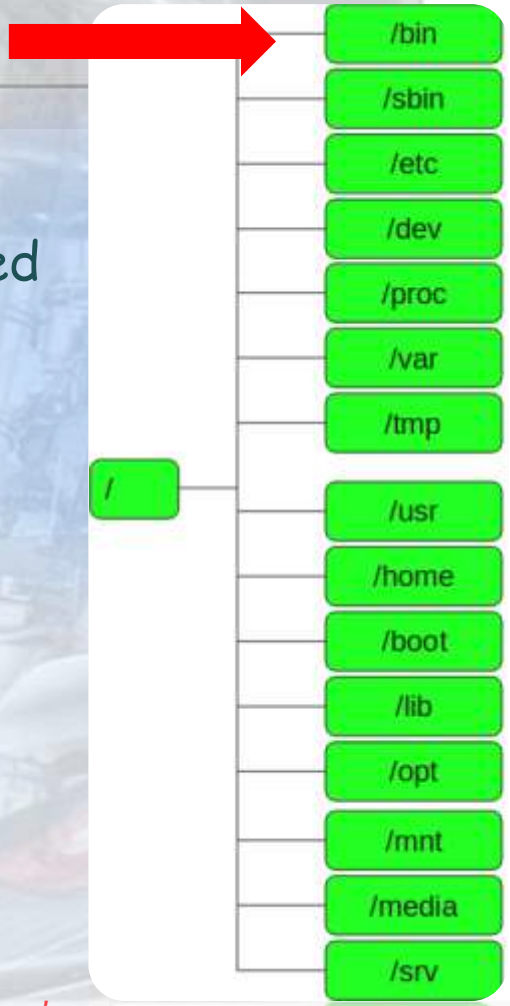
File System Contents (bin)

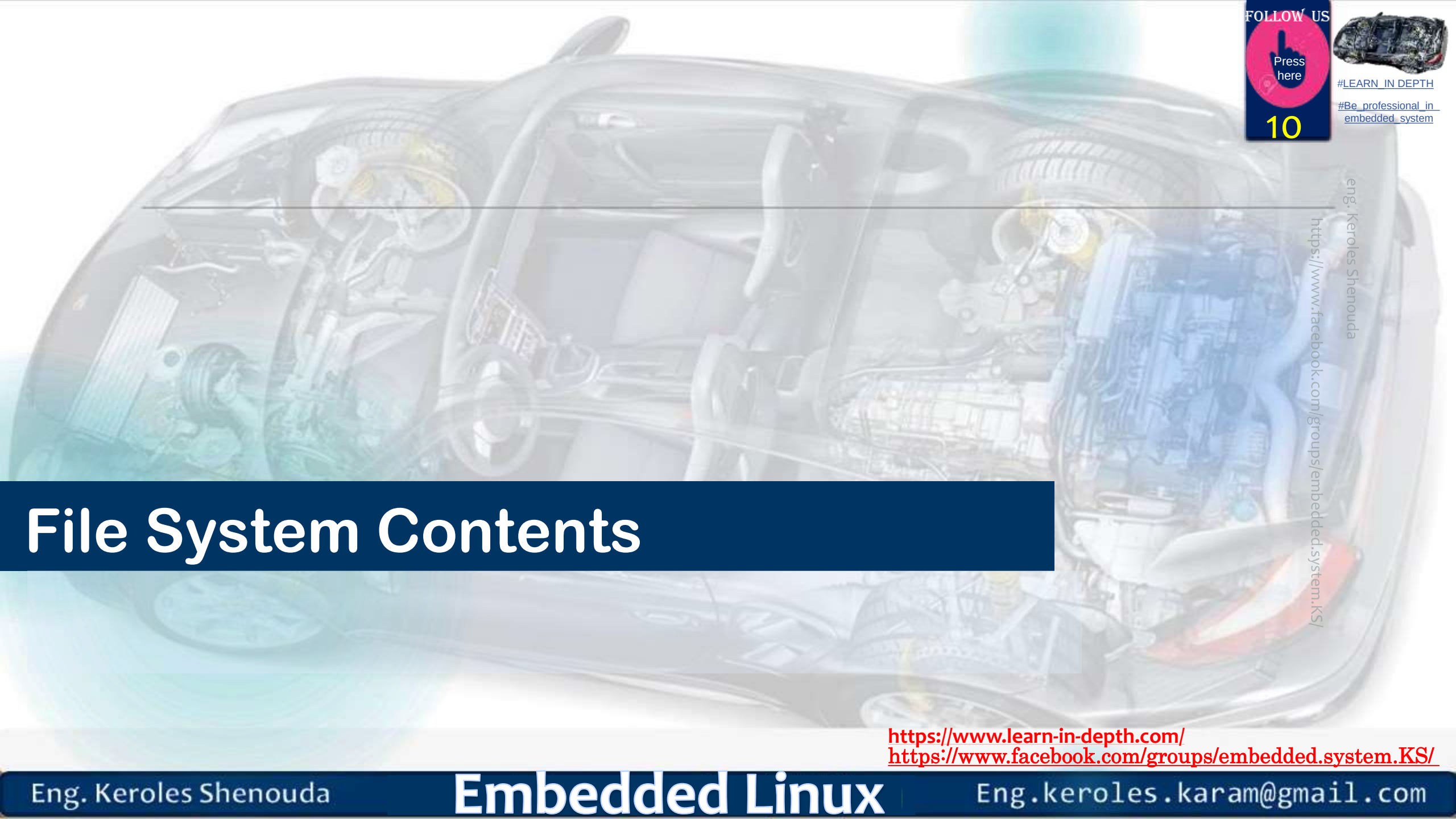
Basic programs

- Place for most commonly used terminal commands
- Common Linux commands you need to use in single-user modes are located under this directory.
- Commands used by all the users of the system are located here.

Examples:

- - ls
- - ping
- - grep
- - cp





File System Contents

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

File System Contents (sbin)

Basic system programs

- Just like /bin, **/sbin** also contains **binary executables**.
- But, the Linux commands located under this directory are used typically by **system administrator**, for system maintenance purpose
- Examples:
  - reboot
  - fdisk
  - ifconfig

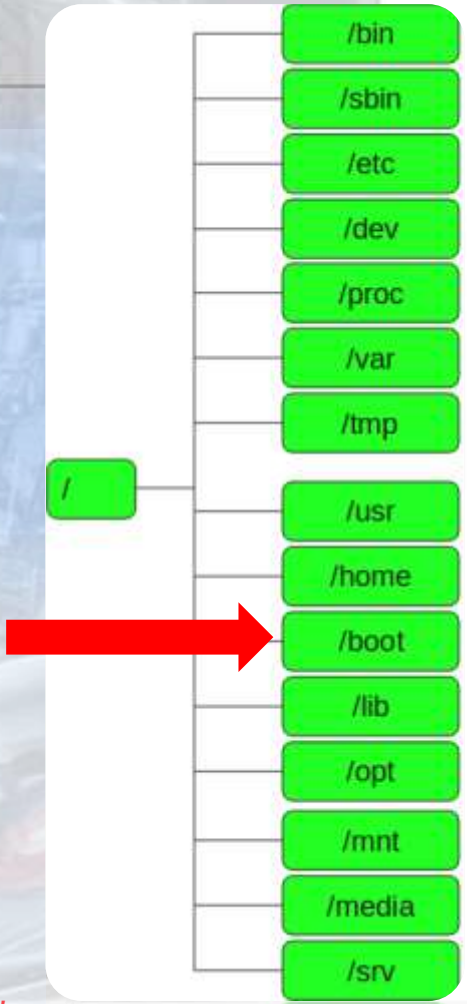
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

File System Contents (boot)

Kernel image (only when the kernel is loaded from a filesystem, not common on non-x86 architectures)

- Contains files needed to start up the system, including the Linux kernel, a RAM disk image and bootloader configuration files

- Kernel image (only when the kernel is loaded from a file system, not common on non-x86 architectures)



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

File System Contents (etc)

System-wide configuration

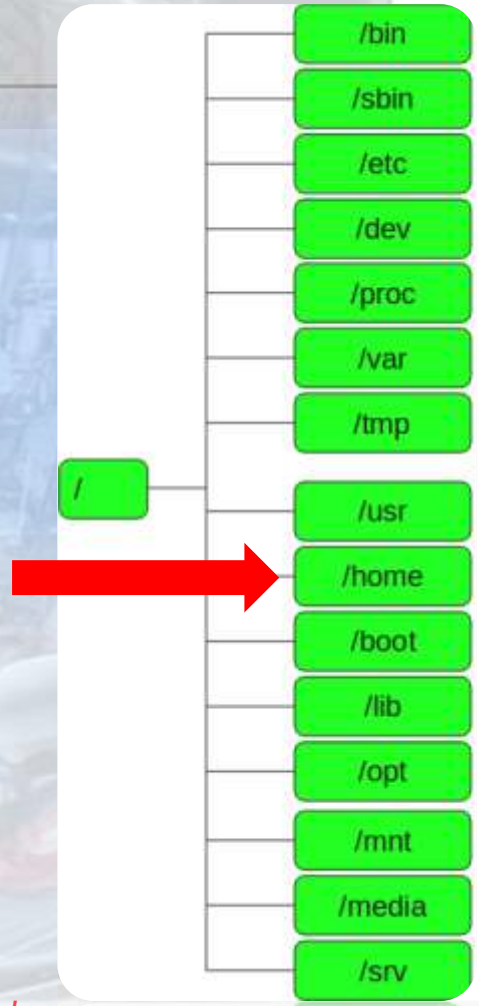
- Contains **configuration files** required by all programs.
- This also contains **startup** and **shutdown** **shell scripts** used to **start/stop individual programs.**
- Examples:
  - /etc/resolv.conf
  - /etc/logrotate.conf

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

File System Contents (home)

Directory for the users home directories

- Home directories for all users to store their personal files.
- Example:
  - /home/karas
  - /home/keroles

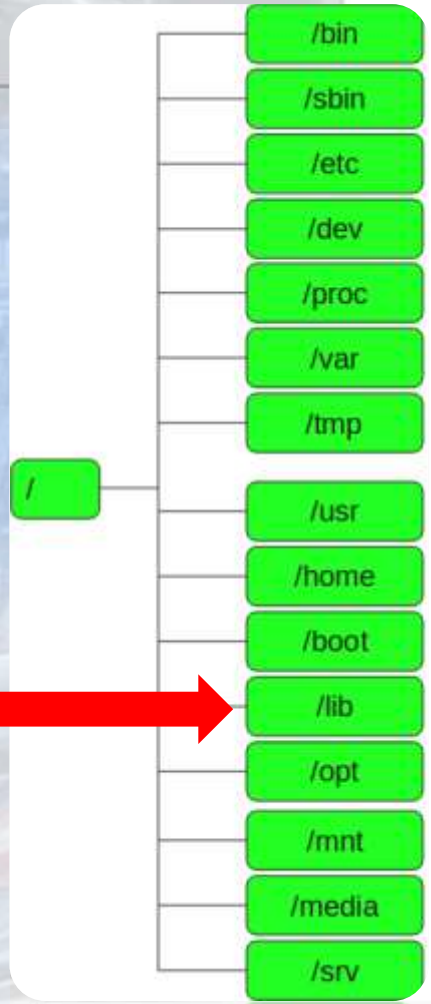


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

File System Contents (lib)

Basic libraries

- Contains **library files** that **supports** the binaries located under /bin and /sbin



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

File System Contents (media)

Mount points for removable media

- Temporary mount directory for **removable devices**.

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

File System Contents (mnt)

Mount points for static media

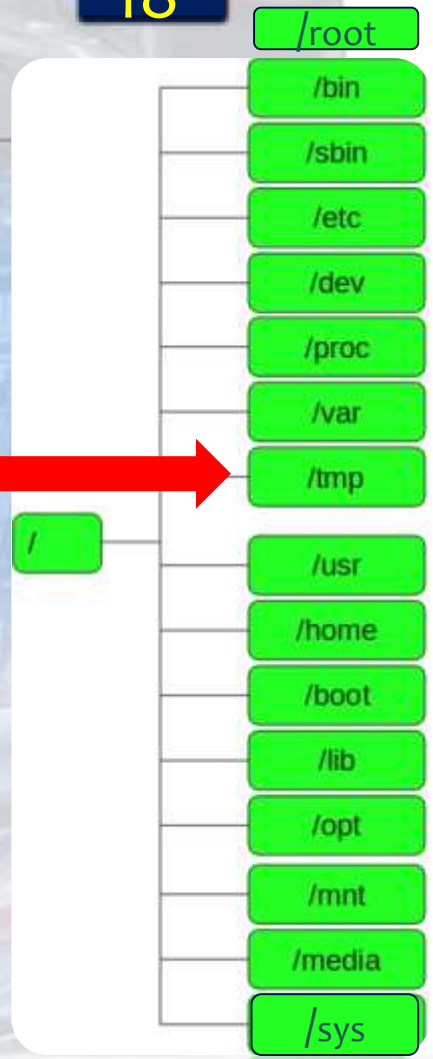
- Temporary mount directory where **sysadmins** can mount file systems

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

File System Contents (tmp)

Temporary files

- Directory that contains temporary files created by system and users
- Files under this directory are **deleted** when system is **rebooted**.

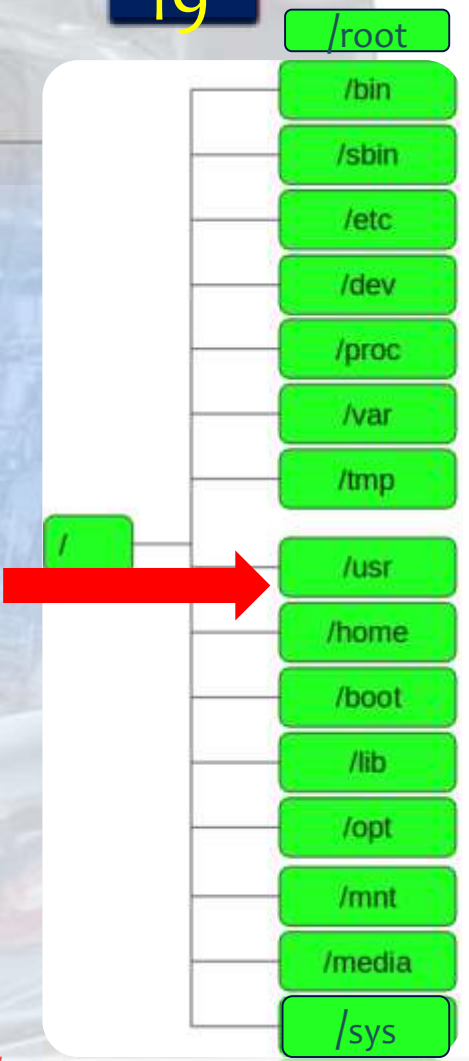


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

File System Contents (usr)

/usr/bin Non-basic programs
/usr/lib Non-basic libraries
/usr/sbin Non-basic system programs

- Contains binaries, libraries, documentation, and source-code for user programs.
- **/usr/bin** contains binary files for user programs. If you can't find a user binary under /bin, look under /usr/bin.
- **/usr/sbin** contains binary files for system administrators. If you can't find a system binary under /sbin, look under /usr/sbin.
- **/usr/lib** contains libraries for /usr/bin and /usr/sbin
- **/usr/local** contains users programs that you install from source

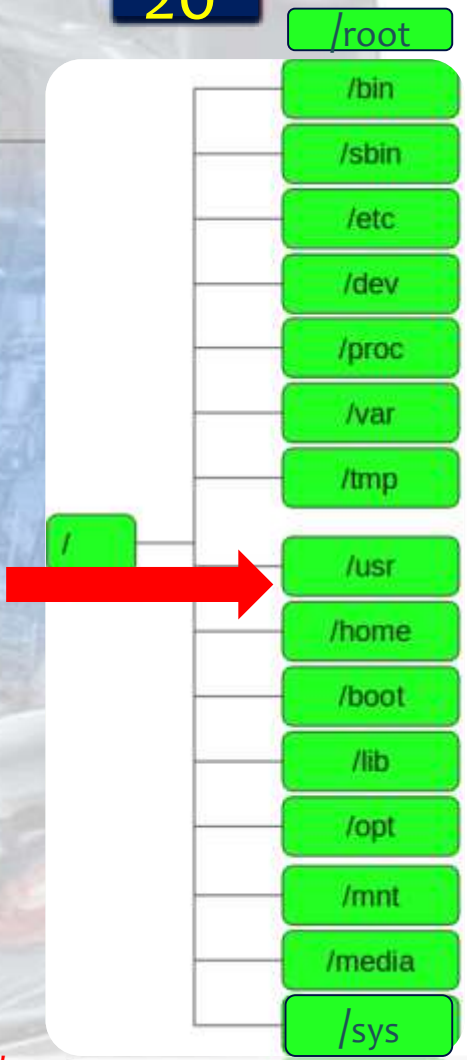


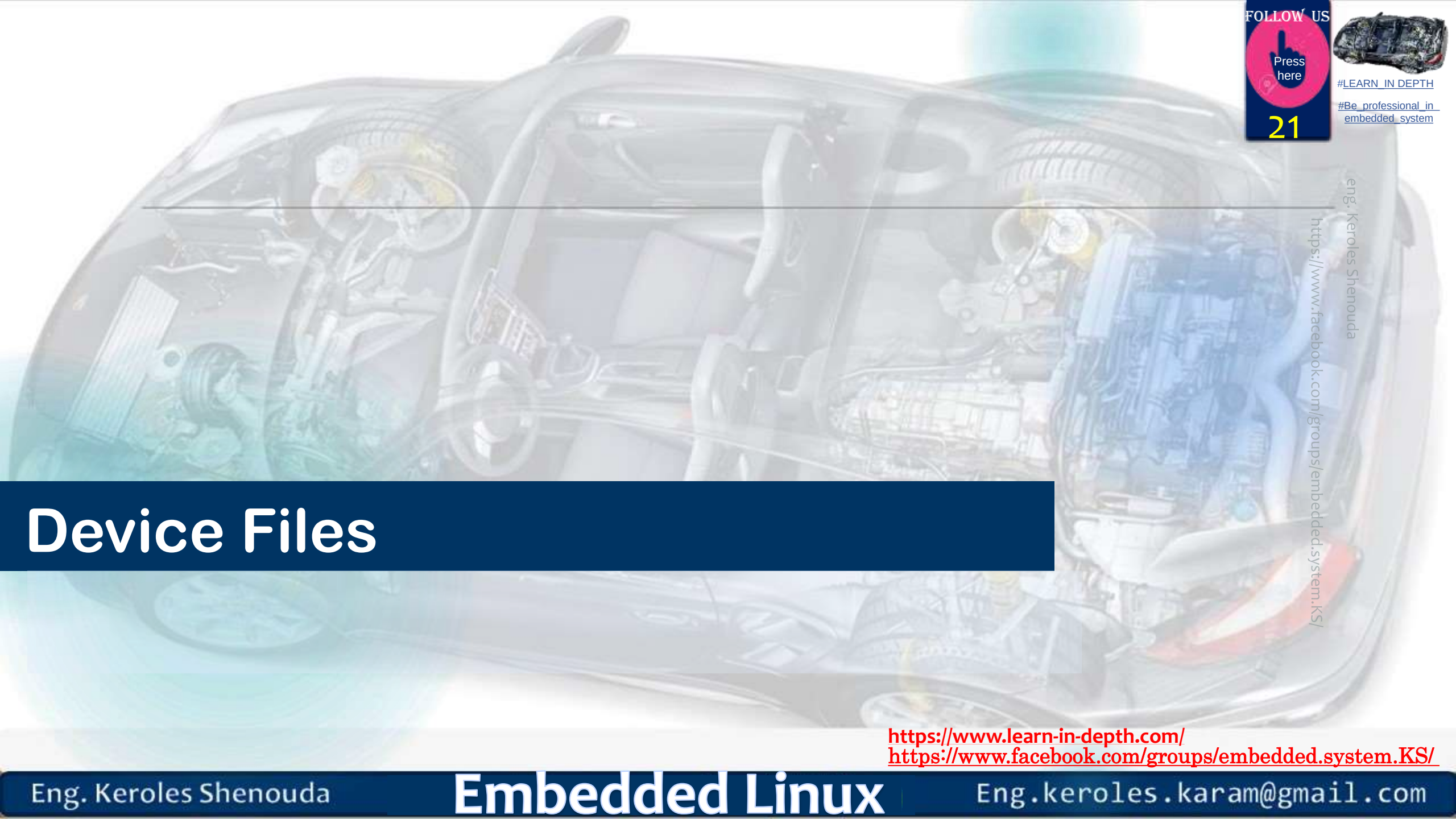
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

File System Contents (var)

Variable data files

- **var** stands for variable files.
- Content of the files that are expected to grow can be found under this directory.
- This includes
  - /var/log - system log files
  - /var/lib - packages and database files
  - /var/mail - emails
  - /var/spool - print queues
  - /var/lock - lock files
  - /var/tmp - temp files needed across reboots





#LEARN_IN DEPTH

#Be_professional_in_embedded_system

Device Files

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

File System Contents (dev)

Device files

- These include terminal devices, usb, or any device attached to the system.
- Examples:
  - /dev/tty1
  - /dev/usbmon0
- One of the kernel important roles is to **allow applications to access hardware devices**

File System Contents (dev)

Device files Cont.

- In the Linux kernel, most devices are presented to user space applications through **two different abstractions**
  - **Character** device
    - Originally, **an infinite stream of bytes**, with **no beginning**, **no end**, **no size**. The pure example: a serial port.
    - Used for **serial ports**, **terminals**, but also **sound cards**, **video acquisition devices**, **frame buffers**.
    - Most of the devices that are not block devices are represented as character devices by the Linux kernel
  - **Block** device
    - A device composed of fixed-sized blocks, that can be read and written to store data
    - Used for hard disks, USB keys, SD cards, etc.
- Internally, the kernel identifies each device by a triplet of information
  - **Type** (character or block)
  - **Major** (typically the category of device)
  - **Minor** (typically the identifier of the device)

File System Contents (dev)

Device files Cont.

- **Devices>>>>>> everything is a file**
  - Most system objects are represented in Linux as **files**.
  - It allows applications to manipulate all *system objects* with the normal file API (open, read, write, close, etc.)
  - So, **devices** had to be **represented** as **files** to the **applications**
  - This is done through a special artifact called a **device file**
  - It is a special type of file, that associates **a file name visible** to ***user space applications*** to the triplet (**type, major, minor**) that the kernel understands
  - *All device files* are by convention stored in the /dev directory
- Example of device files in a Linux system
  - brw-rw---- 1 root disk 8, 1 2011-05-27 08:56 /dev/sda1
  - crw-rw---- 1 root dialout 4, 64 2011-05-27 08:56 /dev/ttyS0

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

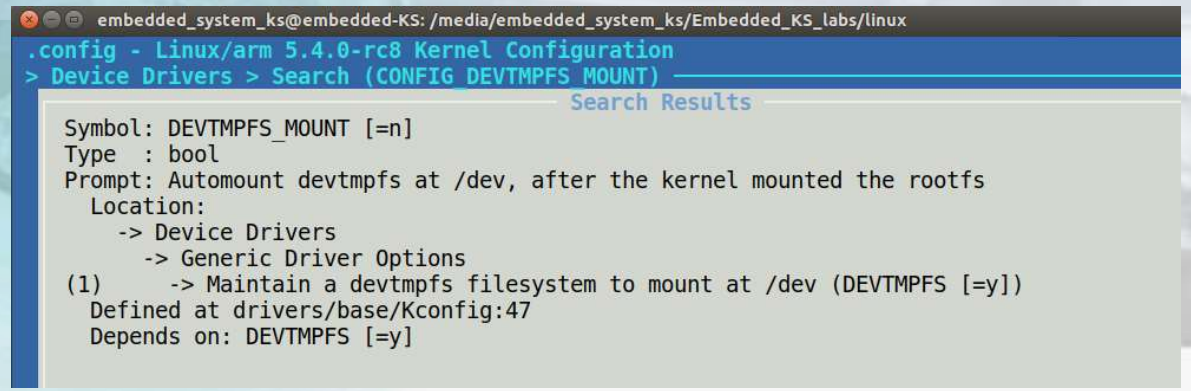
File System Contents (dev)

Device files Cont.

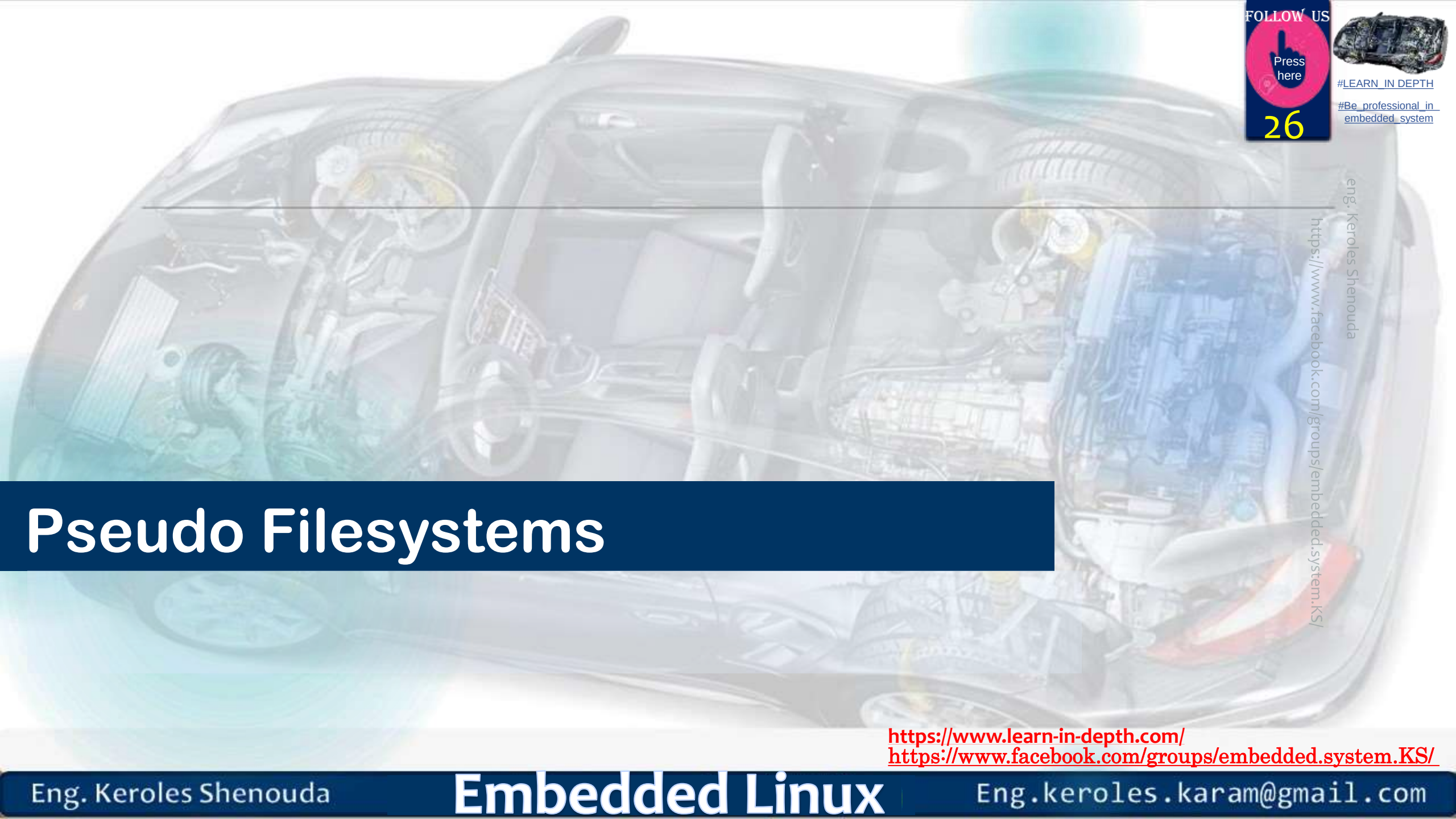
- Example C code that uses the usual file API to write data to a serial port

```
int fd;
fd = open("/dev/ttyS0", O_RDWR);
write(fd, "Hello", 5);
close(fd);
```

- The devtmpfs virtual filesystem can be mounted on /dev and contains all the devices known to the kernel. The CONFIG_DEVTMPFS_MOUNT kernel configuration option makes the kernel mount it automatically at boot time



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



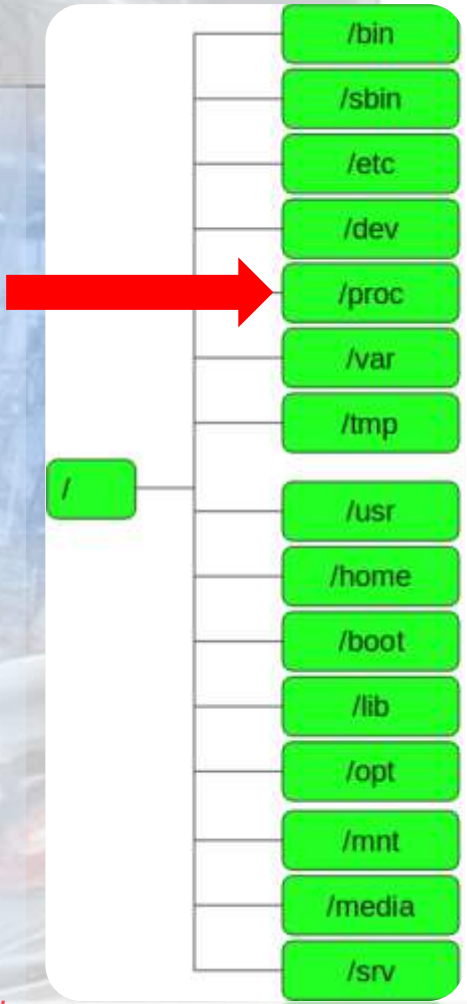
Pseudo Filesystems

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

File System Contents (proc)

Mount point for the proc virtual filesystem

- Contains information about **system process**.
- This is **a pseudo file system** contains information about running process.
  - It allows
    - The kernel to expose statistics about running processes in the system
    - The user to adjust at runtime various system parameters about process
- management, memory management, etc.
  - **One directory** for **each running process** in the system
    - **/proc/<pid>** example: cat /proc/3840/cmdline
    - It contains details about the files opened by the process, the CPU and memory usage, etc.
    - /proc/interrupts, /proc/devices, /proc/iomem, /proc/ioports contain general device-related information. (review the first three parts ☺)
    - /proc/cmdline contains the kernel command line
    - /proc/sys contains many files that can be written to adjust kernel parameters
      - They are called *sysctl*. See Documentation/sysctl/ in kernel sources. Example: echo 3 > /proc/sys/vm/drop_caches



File System Contents (proc)

Mount point for the proc virtual filesystem (Examples)

- init Process /proc
- init Process Memory Segments from /proc

```
# cat /proc/1/maps
00008000-0000f000 r-xp 00000000 00:0a 9537567     /sbin/init
00016000-00017000 rw-p 00006000 00:0a 9537567     /sbin/init
00017000-0001b000 rwxp 00017000 00:00 0
40000000-40017000 r-xp 00000000 00:0a 9537183     /lib/ld-2.3.2.so
40017000-40018000 rw-p 40017000 00:00 0
4001f000-40020000 rw-p 00017000 00:0a 9537183     /lib/ld-2.3.2.so
40020000-40141000 r-xp 00000000 00:0a 9537518     /lib/libc-2.3.2.so
40141000-40148000 ---p 00121000 00:0a 9537518     /lib/libc-2.3.2.so
40148000-4014d000 rw-p 00120000 00:0a 9537518     /lib/libc-2.3.2.so
4014d000-4014f000 rw-p 4014d000 00:00 0
befeb000-bf000000 rwxp befeb000 00:00 0
#
```

```
# ls -l /proc/1
total 0
-r--------    1 root     root            0 Jan  1 00:25 auxv
-r--r--r--    1 root     root            0 Jan  1 00:21 cmdline
lrwxrwxrwx    1 root     root            0 Jan  1 00:25 cwd -> /
-r--------    1 root     root            0 Jan  1 00:25 environ
lrwxrwxrwx    1 root     root            0 Jan  1 00:25 exe -> /sbin/init
dr-x------    2 root     root            0 Jan  1 00:25 fd
-r--r--r--    1 root     root            0 Jan  1 00:25 maps
-rw-------    1 root     root            0 Jan  1 00:25 mem
-r--r--r--    1 root     root            0 Jan  1 00:25 mounts
-rw-r--r--    1 root     root            0 Jan  1 00:25 oom_adj
-r--r--r--    1 root     root            0 Jan  1 00:25 oom_score
lrwxrwxrwx    1 root     root            0 Jan  1 00:25 root -> /
-r--r--r--    1 root     root            0 Jan  1 00:21 stat
-r--r--r--    1 root     root            0 Jan  1 00:25 statm
-r--r--r--    1 root     root            0 Jan  1 00:21 status
dr-xr-xr-x    3 root     root            0 Jan  1 00:25 task
-r--r--r--    1 root     root            0 Jan  1 00:25 wchan
```

File System Contents (sys)

Mount point of the sysfs virtual filesystem

- Mount point of the sysfs virtual file system
- This is **a pseudo file system**
- It allows to **represent** in **user space** the **vision** that the kernel has of the buses, devices and drivers in the system
- It is useful for various user space applications that need to list and query the available hardware, for example **udev** or **mdev**.
- ls /sys/
  - block bus class dev devices firmware fs kernel module power

```
# dir /sys
total 0
drwxr-xr-x   21 root     root     0 Jan  1 00:00 block
drwxr-xr-x    6 root     root     0 Jan  1 00:00 bus
drwxr-xr-x   10 root     root     0 Jan  1 00:00 class
drwxr-xr-x    5 root     root     0 Jan  1 00:00 devices
drwxr-xr-x    2 root     root     0 Jan  1 00:00 firmware
drwxr-xr-x    2 root     root     0 Jan  1 00:00 kernel
drwxr-xr-x    5 root     root     0 Jan  1 00:00 module
drwxr-xr-x    2 root     root     0 Jan  1 00:00 power
#
```

```
[root@vexpress ]# ls -l /sys/firmware/devicetree/base/
total 0
-r--r--r--    1 0        0               4 Feb 22 01:52 #address-cells
-r--r--r--    1 0        0               4 Feb 22 01:52 #size-cells
drwxr-xr-x    2 0        0               0 Feb 22 01:52 aliases
-r--r--r--    1 0        0               4 Feb 22 01:52 arm,hbi
-r--r--r--    1 0        0               4 Feb 22 01:52 arm,vexpress,site
drwxr-xr-x    2 0        0               0 Feb 22 01:52 cache-controller@1e00a000
drwxr-xr-x    2 0        0               0 Feb 22 01:52 chosen
drwxr-xr-x    3 0        0               0 Feb 22 01:52 clcd@10020000
-r--r--r--    1 0        0              34 Feb 22 01:52 compatible
drwxr-xr-x    6 0        0               0 Feb 22 01:52 cpus
drwxr-xr-x   15 0        0               0 Feb 22 01:52 dcc
drwxr-xr-x    2 0        0               0 Feb 22 01:52 hsb@e0000000
drwxr-xr-x    2 0        0               0 Feb 22 01:52 interrupt-controller@1e001000
-r--r--r--    1 0        0               4 Feb 22 01:52 interrupt-parent
drwxr-xr-x    2 0        0               0 Feb 22 01:52 memory-controller@100e0000
drwxr-xr-x    2 0        0               0 Feb 22 01:52 memory-controller@100e1000
drwxr-xr-x    2 0        0               0 Feb 22 01:52 memory@60000000
-r--r--r--    1 0        0               8 Feb 22 01:52 model
-r--r--r--    1 0        0               1 Feb 22 01:52 name
drwxr-xr-x    2 0        0               0 Feb 22 01:52 pmu
drwxr-xr-x    3 0        0               0 Feb 22 01:52 reserved-memory
drwxr-xr-x    2 0        0               0 Feb 22 01:52 scu@1e000000
drwxr-xr-x    3 0        0               0 Feb 22 01:48 smb@4000000
drwxr-xr-x    2 0        0               0 Feb 22 01:52 timer@100e4000
drwxr-xr-x    2 0        0               0 Feb 22 01:52 timer@1e000600
drwxr-xr-x    2 0        0               0 Feb 22 01:52 watchdog@100e5000
drwxr-xr-x    2 0        0               0 Feb 22 01:52 watchdog@1e000620
[root@vexpress ]#
```

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Systool utility

- Many utilities use this information to determine the characteristics of system devices
- You can see from this listing the variety of system information available from sysfs

```
$ systool
Supported sysfs buses:
        i2c
        ide
        pci
        platform
Supported sysfs classes:
        block
        i2c-adapter
        i2c-dev
        input
        mem
        misc
        net
        pci_bus
        tty
Supported sysfs devices:
        pci0000:00
        platform
        system
```

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

How to create a minimal root filesystem ?

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

How to create a minimal root filesystem ?

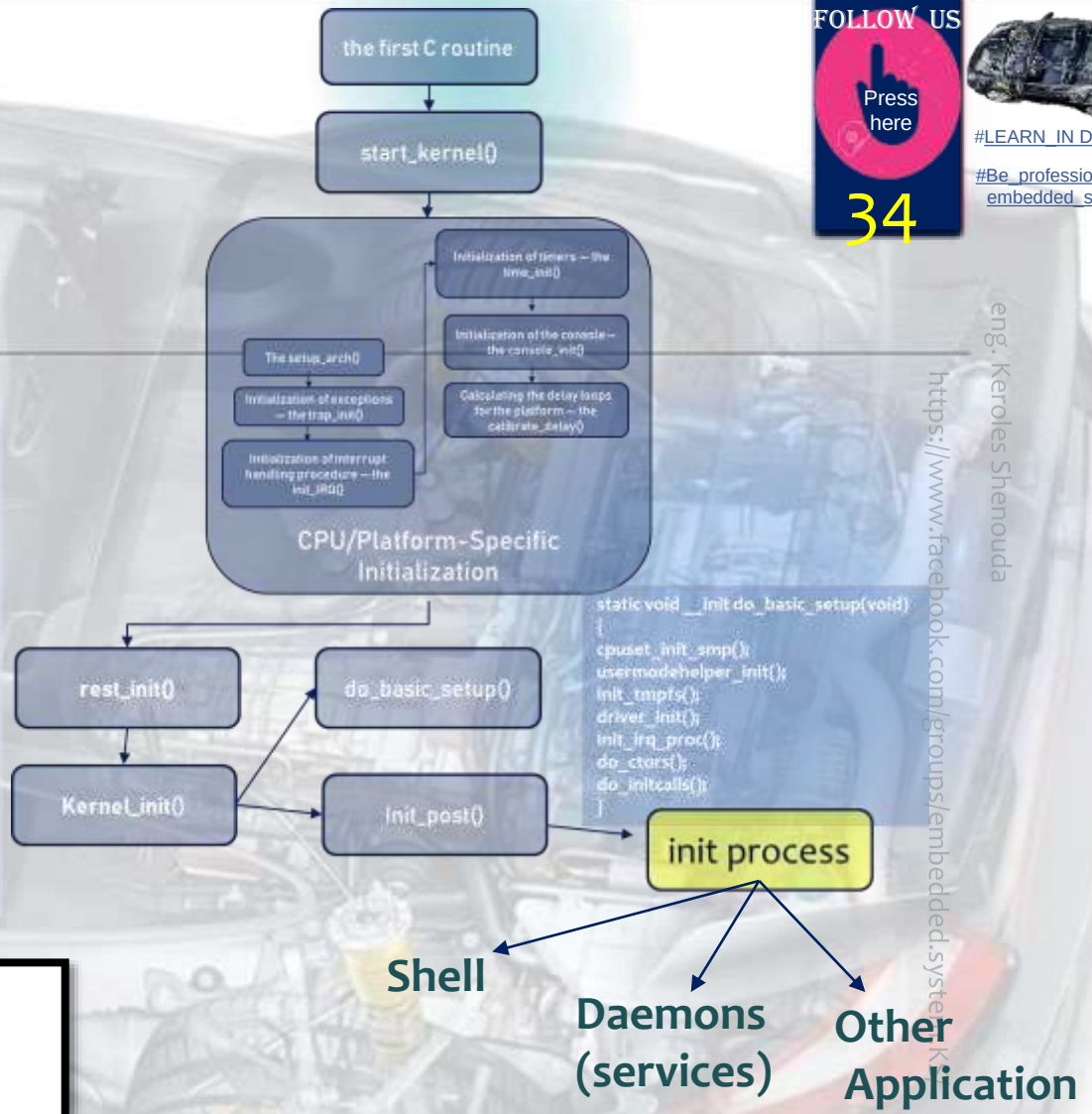
- you need these components:
  - **init**: This is the program that starts everything off, usually by running a series of scripts.
  - **Shell**: You need a shell to give you a command prompt, also to run the shell scripts called by init and other programs.
  - **Daemons**: A daemon is a **background program** that provides a service to others. Good examples are the **system log daemon** (**syslogd**) and the **secure shell daemon** (**sshd**). The init program must start the initial population of daemons to support the main system applications. In fact, init is itself a daemon: it is the daemon that provides the service of launching other daemons.

How to create a minimal root filesystem ?Cont.

- **Shared libraries**: Most programs are linked with shared libraries, and so they must be present in the root filesystem.
- **Configuration files**: The configuration for init and other daemons is stored in a series of text files, usually in the /etc directory.
- **Device nodes**: These are the special files that give access to various device drivers.
- **/proc and /sys**: These two pseudo filesystems represent kernel data structures as a hierarchy of directories and files.
- **Kernel modules**: If you have configured some parts of your kernel to be modules, they need to be installed in the root filesystem, usually in /lib/modules/[kernel version].

The init program

- init application is **the first user space application started by the kernel** after mounting the root filesystem
- The kernel tries to run **/sbin/init**, **/bin/init**, **/etc/init** and **/bin/sh**.
- The init path can be supplied by the **init=** kernel argument.
  - If none of them are found, **the kernel panics** and **the boot process is stopped**.
- The **init** application is responsible for starting all **other user space applications** and **services**



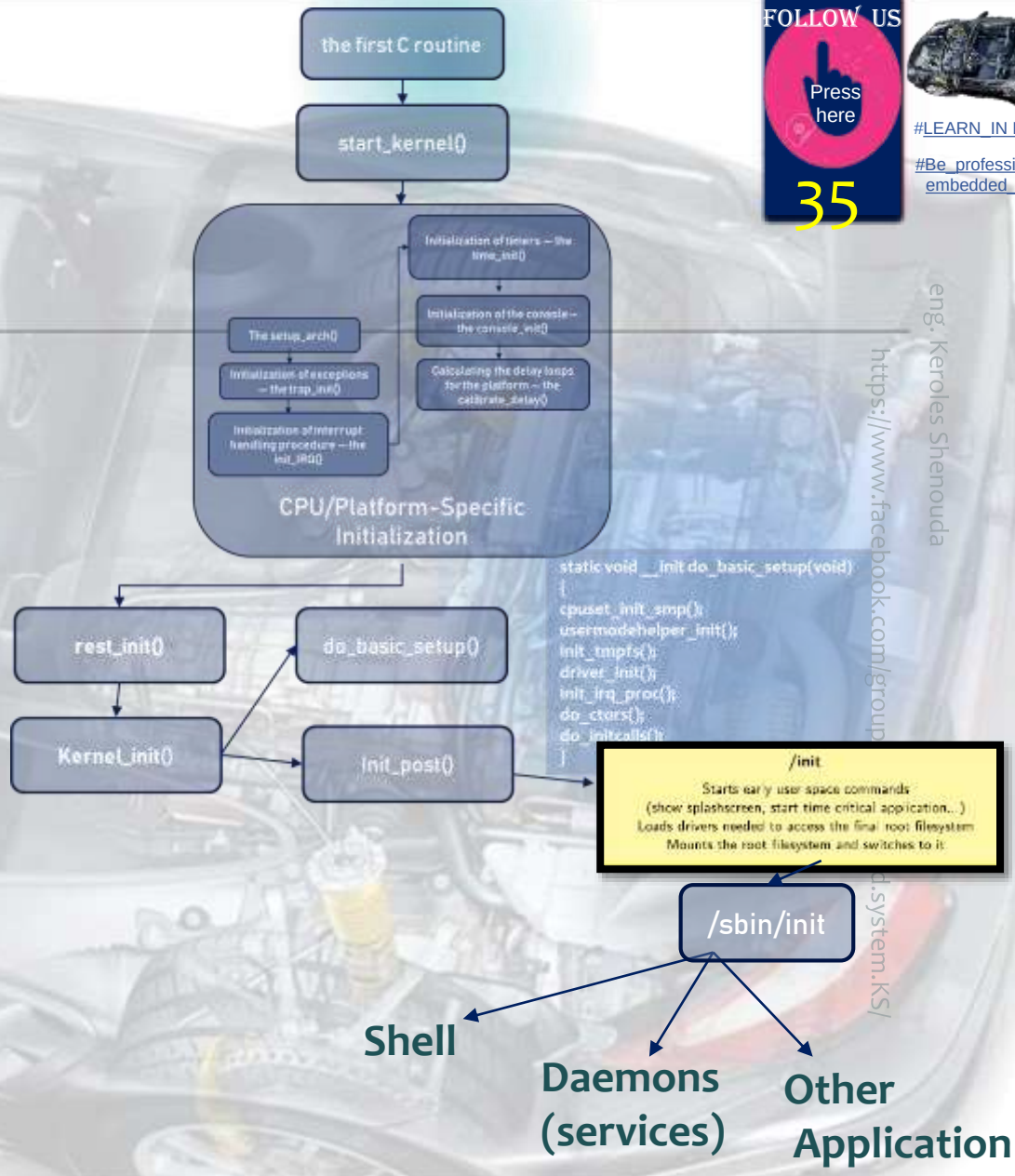
The final snippet of code from **.../init/main.c**

```
...
  if (execute_command) {
          run_init_process(execute_command);
          printk(KERN_WARNING "Failed to execute %s.  Attempting "
                              "defaults...\n", execute_command);
  }

  run_init_process("/sbin/init");
  run_init_process("/etc/init");
  run_init_process("/bin/init");
  run_init_process("/bin/sh");

  panic("No init found.  Try passing init= option to kernel.");
```

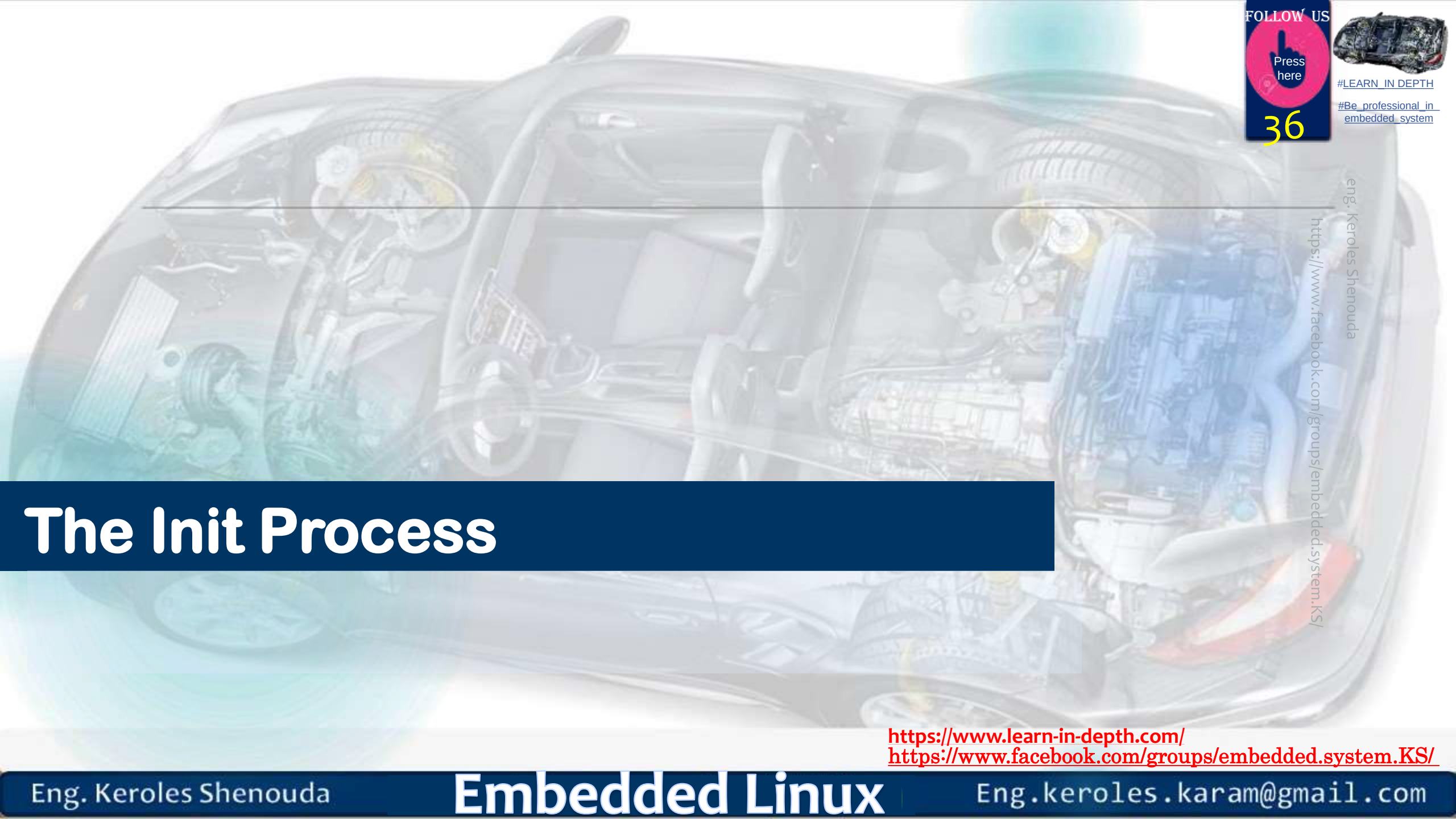

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

In case we initramfs

- In the case of an initramfs, it will only look for /init
  - The init path can be supplied by the **rdinit=** kernel argument
- Then will invoke the regular system startup (/sbin/init)



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



The Init Process

The init Program

- There are many possible implementations of init
  - BusyBox init
  - System V init
  - Systemd init
- init manages the lifecycle of the system from boot up to shutdown
- The most used in embedded linux is BusyBox init

Metric	BusyBox init	System V init	systemd
Complexity	Low	Medium	High
Boot-up speed	Fast	Slow	Medium
Required shell	ash	ash or bash	None

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

BusyBox init

```
null::sysinit:/bin/mount -t proc proc /proc
null::sysinit:/bin/mount -t sysfs sysfs /sys
console::askfirst:-/bin/sh
```

- init begins by reading /etc/inittab. This contains a list of programs to run, one per line, with this format:
- <id>::<action>:<program>
  - **id**: This is the controlling Terminal for the command
  - **action**: This is the conditions to run this command
  - **program**: This is the program to run
- The actions are as follows:
  - **sysinit**: Runs the program when init starts before any of the other types of Actions.
  - **respawn**: Runs the program and restarts it if it terminates. It is typically used to run a program as a daemon.
  - **askfirst**: This is the same as respawn, but it prints the message *Please press Enter to activate this console* to the console, and it runs the program after ***Enter*** has been **pressed**. It is used to start an interactive shell on a Terminal without prompting for a username or password.
  - **once**: Runs the program once but does not attempt to restart it if it terminates.
  - **wait**: Runs the program and waits for it to complete.
  - **restart**: Runs the program when init receives the signal SIGHUP, indicating that it should reload the inittab file.
  - **ctrlaltdel**: Runs the program when init receives the signal, SIGINT, usually as a result of pressing *Ctrl + Alt + Del* on the console.
  - **shutdown**: Runs the program when init shuts down.

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

The init program

- The **init** program begins by reading the configuration file, /etc/inittab
  - simple example

```
::sysinit:/etc/init.d/rcS
::askfirst:-/bin/sh
```

    - The first line runs a shell script **rcS**, when init is started.
    - The second line **prints the message** Please press Enter to activate this console to the console and starts a shell when you press Enter.
- The script called **/etc/init.d/rcS** is the place to put initialization commands that need to be performed at boot, for example, mounting the **proc** and **sysfs** filesystems
- Make sure that you make rcS executable like this:
  - **$ cd ~/rootfs**
  - **$ chmod +x etc/init.d/rcS**

```
#!/bin/sh
mount -t proc proc /proc
mount -t sysfs sysfs /sys
mount -t devtmpfs devtmpfs /dev
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Mdev in /etc/init.d/rcS

- allow you to modify the permissions of device nodes as they are created.
- you begin by running **mdev** with the **-s** option, which causes it to scan the **/sys** directory looking for information about current devices.
- If you want to **keep track of new devices coming online *and create nodes for them as well***, you need to make mdev a hot plug client by writing to /proc/sys/kernel/hotplug.
- These additions to /etc/init.d/rcS will achieve all of this:

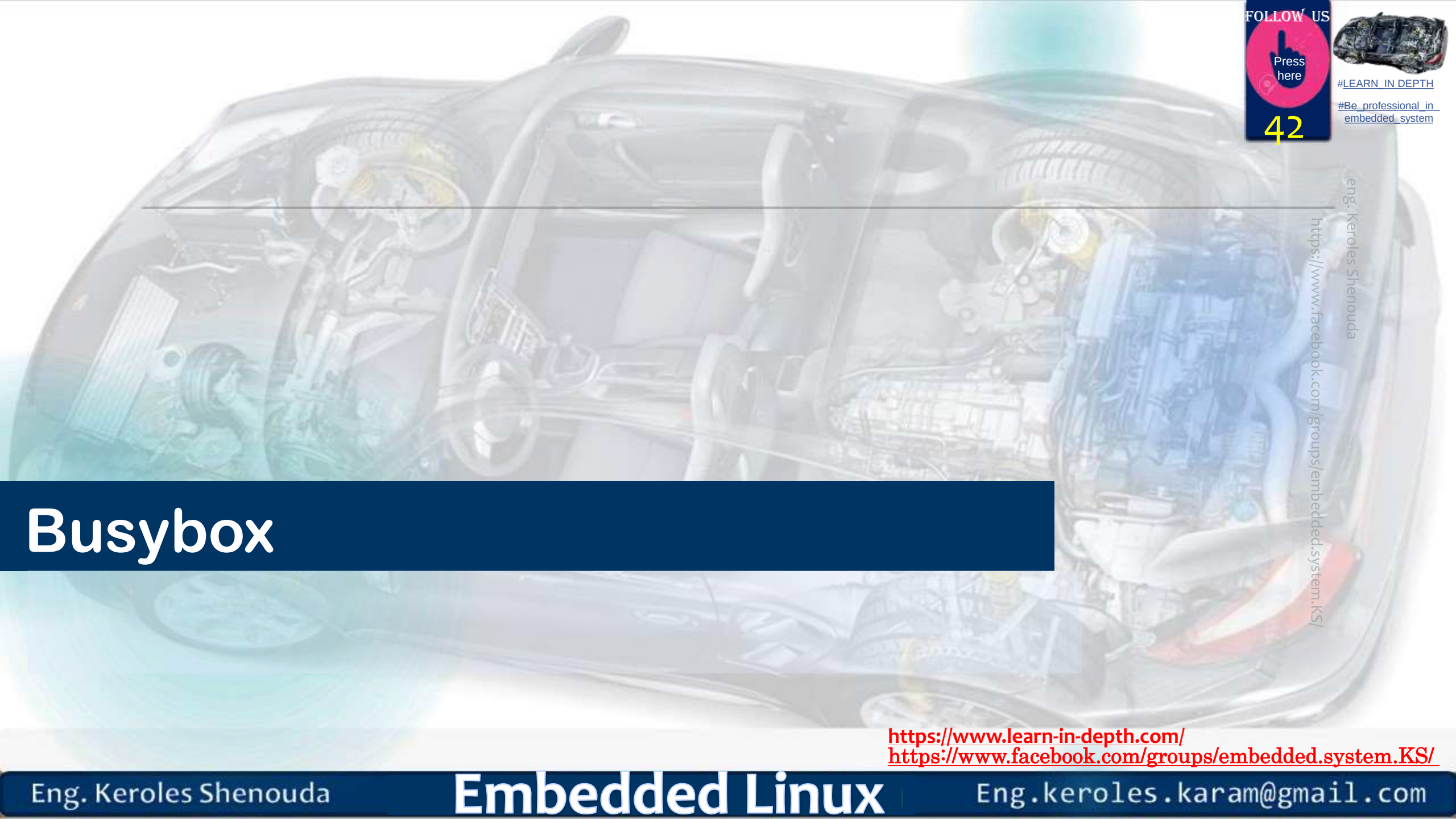
```
#!/bin/sh
mount -t proc proc /proc
mount -t sysfs sysfs /sys
mount -t devtmpfs devtmpfs /dev
echo /sbin/mdev > /proc/sys/kernel/hotplug
mdev -s
```

Vexpress Arm Cortex A9 inittab

embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_KS_labs/LABS/Linux_LABS/lab1/mount_p2/etc

```
# /etc/inittab
::sysinit:/etc/init.d/rcS
console::askfirst:-/bin/sh
::ctrlaltdel:/sbin/reboot
::shutdown:/bin/umount -a -r
::restart:/sbin/init

~
~
~
~
~
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Busybox

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Why Busybox?

- A Linux system needs a basic set of programs to work
  - An init program
  - A shell
  - Various basic utilities for file manipulation and system configuration
- In normal Linux systems, these programs are provided by different projects
  - coreutils, bash, grep, sed, tar, wget, modutils, etc. are all different projects
  - A lot of different components to integrate
- Busybox is an alternative solution, extremely common on embedded systems
- For a fairly featureful configuration, less than 500 KB (statically compiled with uClibc) or less than 1 MB (statically compiled with glibc).
- http://www.busybox.net/

BusyBox commands

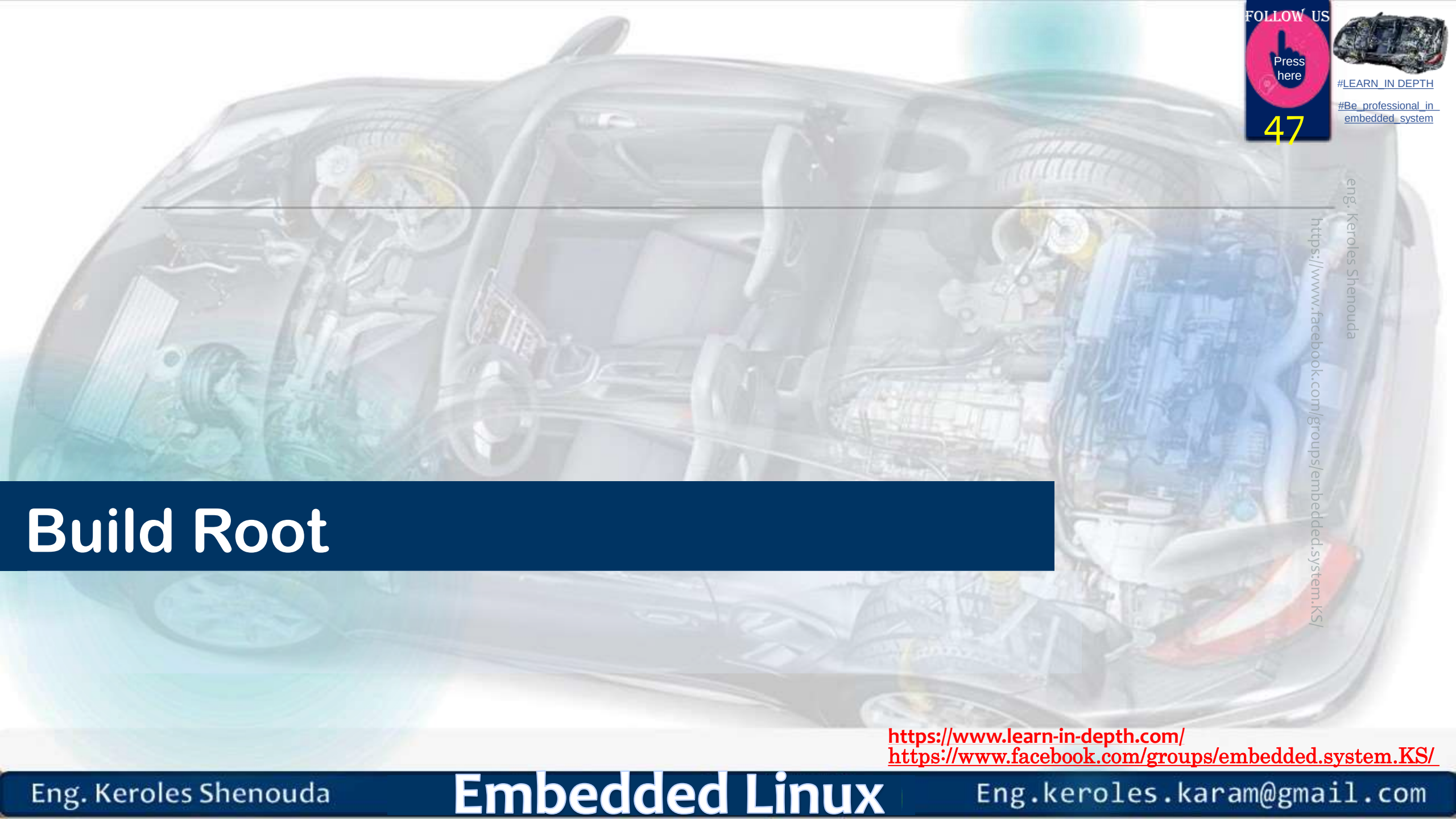
```
[, [[, acpid, addgroup, adduser, adjtimex, ar, arp, arping, ash,
awk, basename, beep, blkid, brctl, bunzip2, bzcat, bzip2, cal, cat,
catv, chat, chattr, chgrp, chmod, chown, chpasswd, chpst, chroot,
chrt, chvt, cksum, clear, cmp, comm, cp, cpio, crond, crontab,
cryptpw, cut, date, dc, dd, deallocvt, delgroup, deluser, depmod,
devmem, df, dhcprelay, diff, dirname, dmesg, dnsd, dnsdomainname,
dos2unix, dpkg, du, dumpkmap, dumpleases, echo, ed, egrep, eject,
env, envdir, envuidgid, expand, expr, fakeidentd, false, fbset,
fbsplash, fdflush, fdformat, fdisk, fgrep, find, findfs, flash_lock,
flash_unlock, fold, free, freeramdisk, fsck, fsck.minix, fsync,
ftpd, ftpget, ftpput, fuser, getopt, getty, grep, gunzip, gzip, hd,
hdparm, head, hexdump, hostid, hostname, httpd, hush, hwclock, id,
ifconfig, ifdown, ifenslave, ifplugd, ifup, inetd, init, inotifyd,
insmod, install, ionice, ip, ipaddr, ipcalc, ipcrm, ipcs, iplink,
iproute, iprule, iptunnel, kbd_mode, kill, killall, killall5, klogd,
last, length, less, linux32, linux64, linuxrc, ln, loadfont,
loadkmap, logger, login, logname, logread, losetup, lpd, lpq, lpr,
ls, lsattr, lsmod, lzmacat, lzop, lzopcat, makemime, man, md5sum,
mdev, mesg, microcom, mkdir, mkdosfs, mkfifo, mkfs.minix, mkfs.vfat,
mknod, mkpasswd, mkswap, mktemp, modprobe, more, mount, mountpoint,
mt, mv, nameif, nc, netstat, nice, nmeter, nohup, nslookup, od,
openvt, passwd, patch, pgrep, pidof, ping, ping6, pipe_progress,
pivot_root, pkill, popmaildir, printenv, printf, ps, pscan, pwd,
raidautorun, rdate, rdev, readlink, readprofile, realpath,
reformime, renice, reset, resize, rm, rmdir, rmmod, route, rpm,
rpm2cpio, rtcwake, run-parts, runlevel, runsv, runsvdir, rx, script,
scriptreplay, sed, sendmail, seq, setarch, setconsole, setfont,
setkeycodes, setlogcons, setsid, setuidgid, sh, sha1sum, sha256sum,
sha512sum, showkey, slattach, sleep, softlimit, sort, split,
start-stop-daemon, stat, strings, stty, su, sulogin, sum, sv,
svlogd, swapoff, swapon, switch_root, sync, sysctl, syslogd, tac,
tail, tar, taskset, tcpsvd, tee, telnet, telnetd, test, tftp, tftpd,
time, timeout, top, touch, tr, traceroute, true, tty, ttysize,
udhcpc, udhcpd, udpsvd, umount, uname, uncompress, unexpand, uniq,
unix2dos, unlzma, unlzop, unzip, uptime, usleep, uudecode, uuencode,
vconfig, vi, vlock, volname, watch, watchdog, wc, wget, which, who,
whoami, xargs, yes, zcat, zcip
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

File System Types

File System Types

- RAM File Systems
  - initrd
  - initramfs
- Disk File Systems
  - ext2
  - ext3
  - ext4
- Flash File Systems
  - Intro to MTD
  - JFFS2
- Distributed File Systems
  - NFS
- Special Purpose File Systems
  - Squashfs
    - Mostly used in live CDs and live USB distros

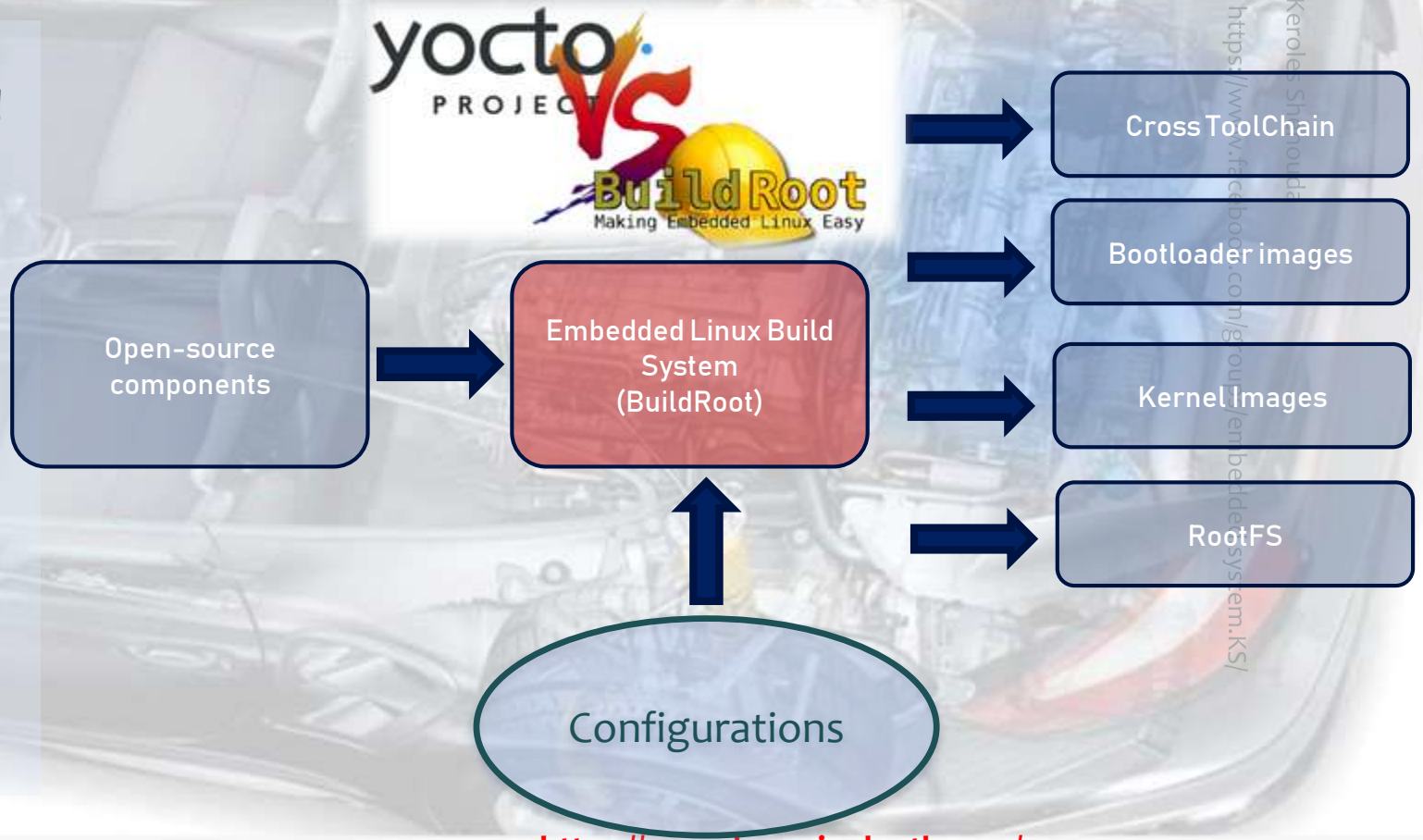


Build Root

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Embedded Linux build system: principle

- Building from source ! lot of flexibility
- Cross-compilation ! leveraging fast build machines
- Recipes for building components ! Easy
- two solutions are emerging as the most popular ones
- **Yocto/OpenEmbedded**
  - Builds a complete Linux distribution. Powerful, but somewhat complex, and quite steep learning curve.
- **Buildroot**
- Builds a root filesystem image, Much simpler to use, understand and modify.

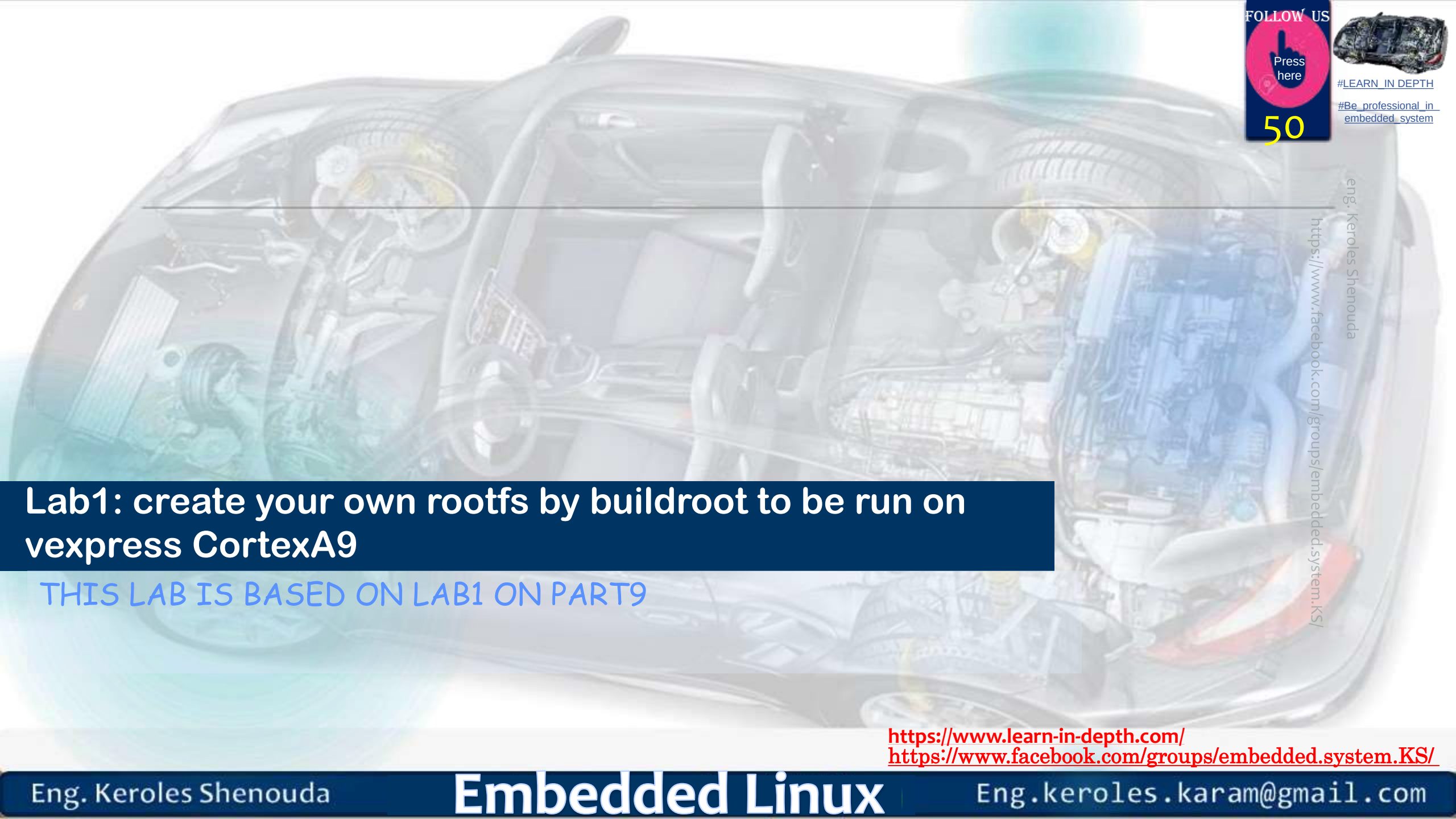


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Who's using Buildroot?

- **System makers**
  - Tesla
  - GoPro
  - Barco
  - Rockwell Collins
- **Processor vendors**
  - Imagination Technologies
  - Marvell
  - Microchip (formerly Atmel)
- **SoM and board vendors**
- Many companies when doing R&D on products
- Many, many **hobbyists** on development boards: Raspberry Pi, BeagleBone Black, etc.

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Lab1: create your own rootfs by buildroot to be run on vexpress CortexA9

THIS LAB IS BASED ON LAB1 ON PART9

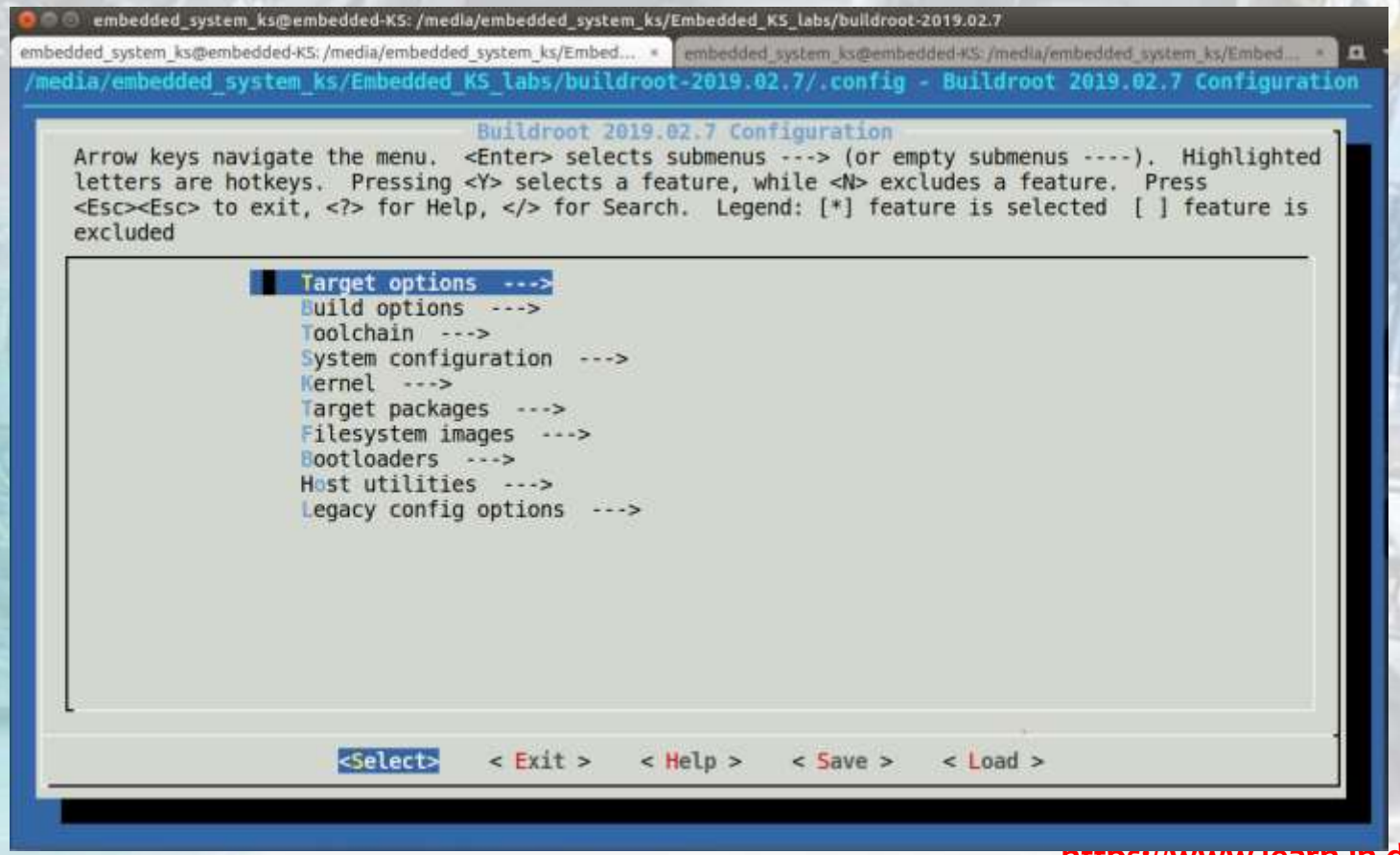
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Configuring Buildroot for vexpress CortexA9 based on RPI 2 Configuration as both based on armv7 (think in Depth)

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/buildroot-2019.02.7$ make raspberrypi2_de
fconfig
#
# configuration written to /media/embedded_system_ks/Embedded_KS_labs/buildroot-2019.02.7/.config
#
```

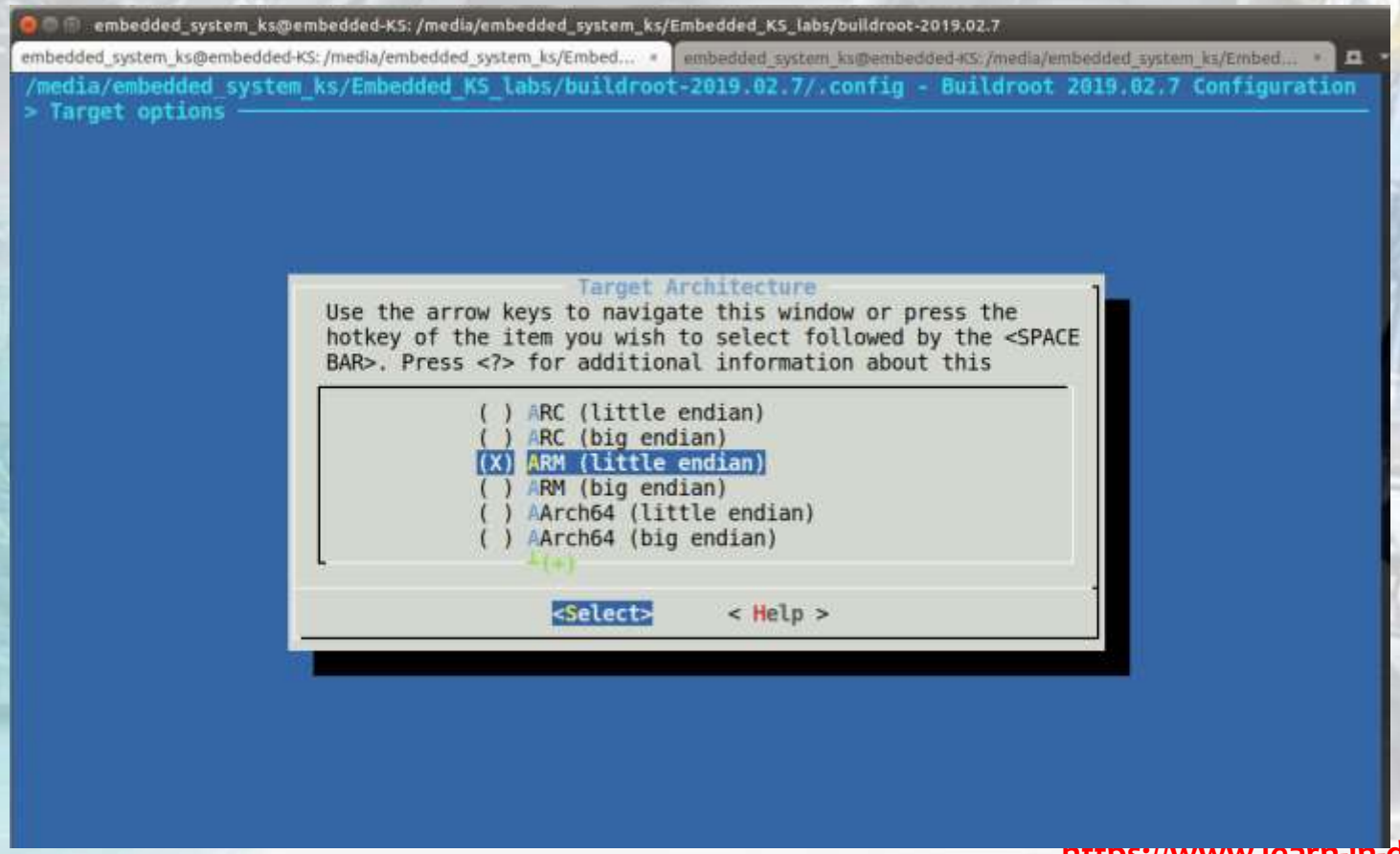

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Configuring Buildroot for vexpress CortexA9.



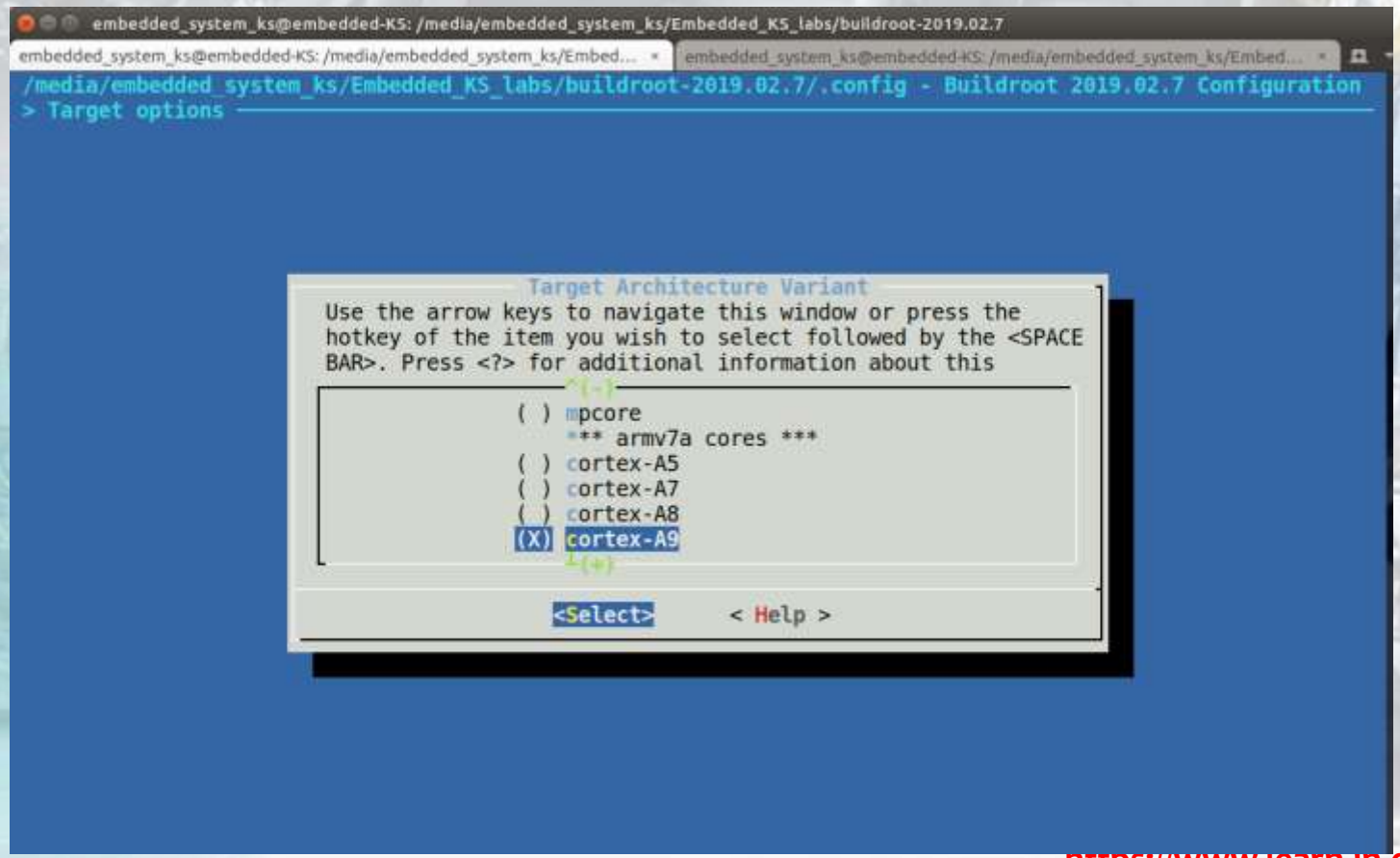
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Configuring Buildroot for vexpress CortexA9.



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

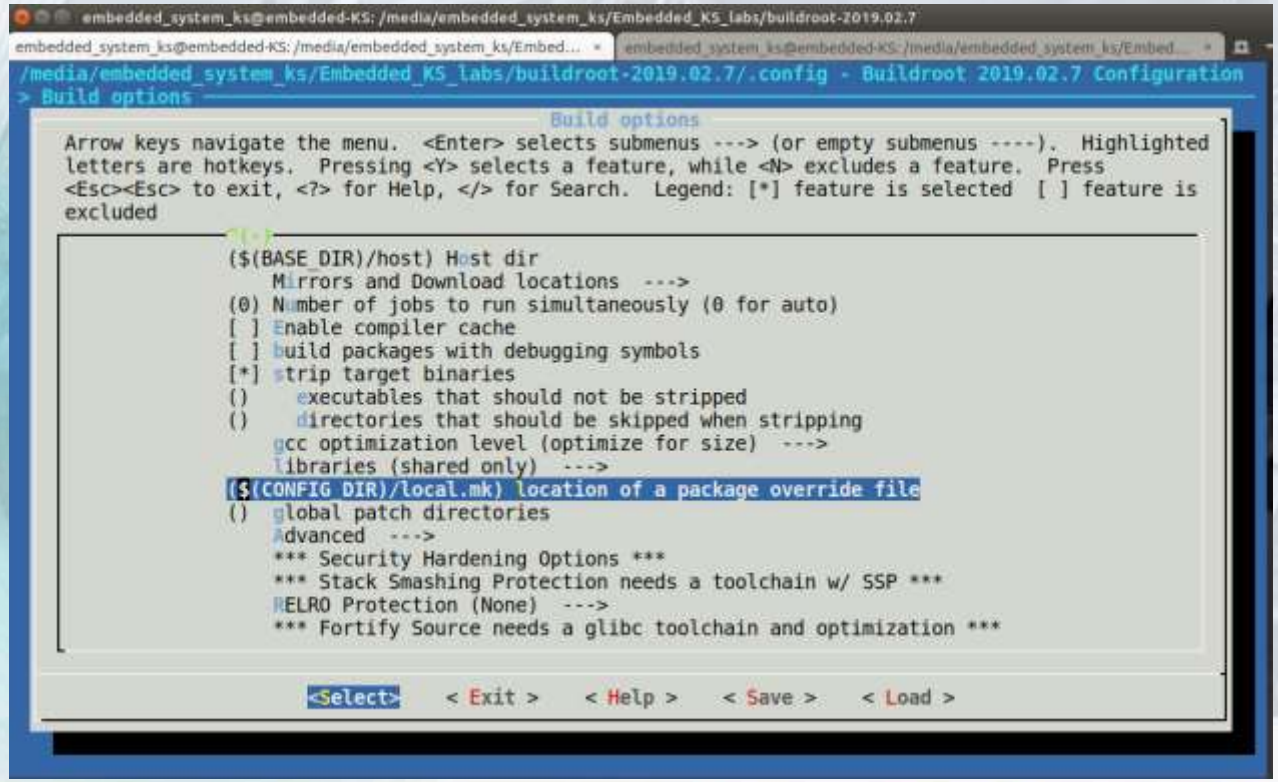
Configuring Buildroot for vexpress CortexA9.



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Configuring Buildroot for vexpress CortexA9.

- Build options
  - How Buildroot will built the code. Leave default values

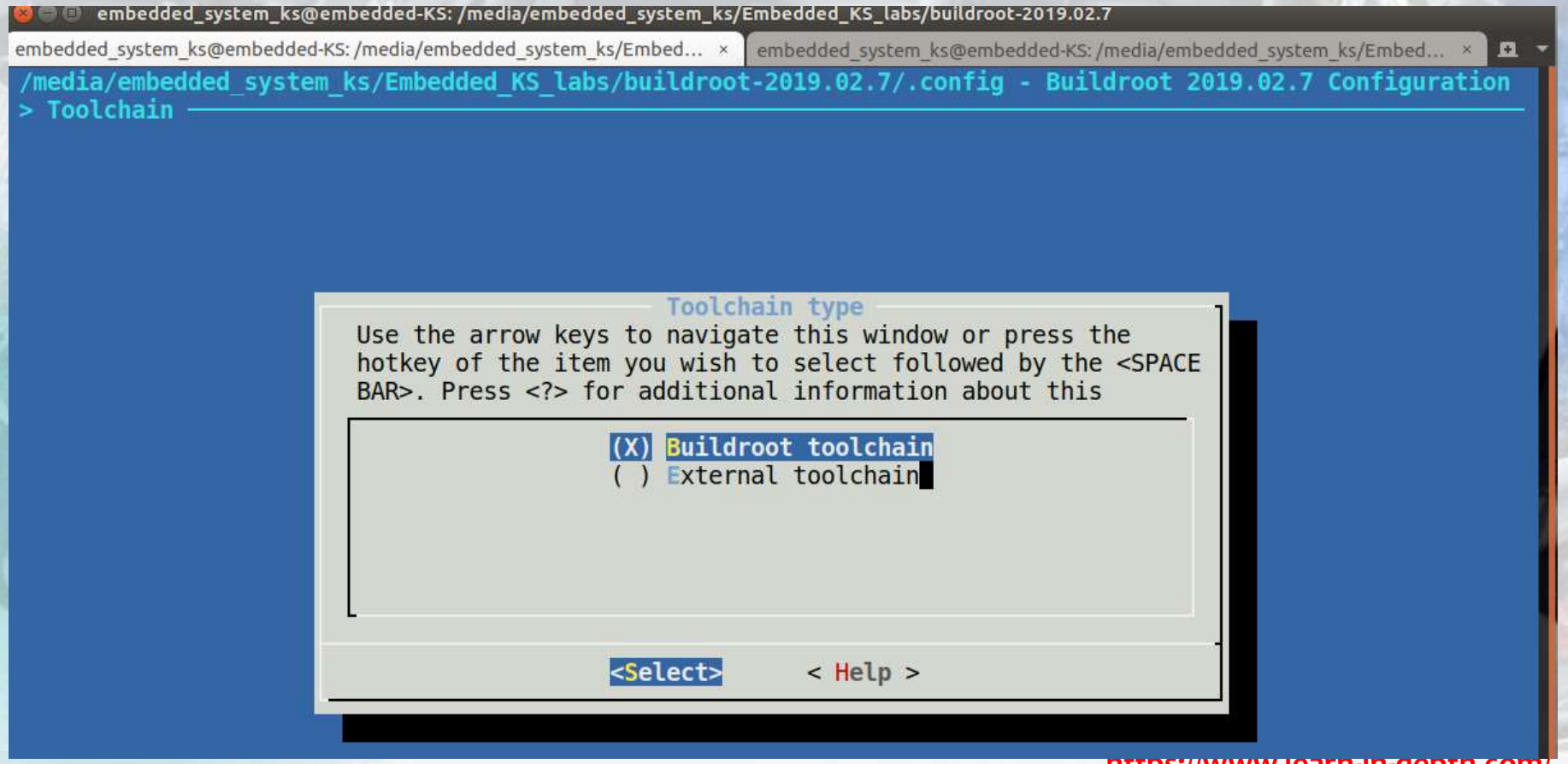


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Configuring Buildroot for vexpress CortexA9.

Toolchain

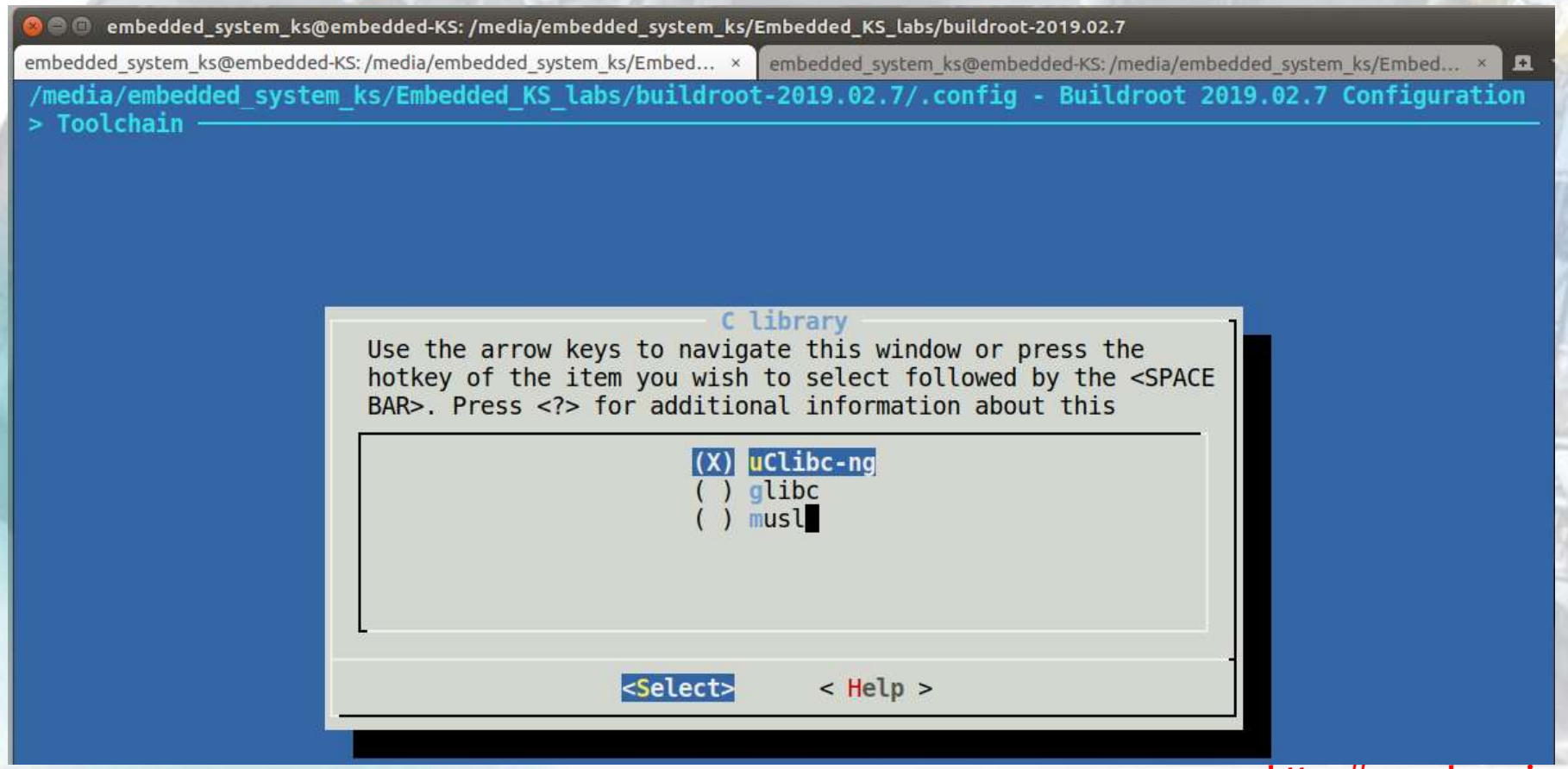
Embedded system will be compiled with tools integrated in Buildroot



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Configuring Buildroot for vexpress CortexA9.

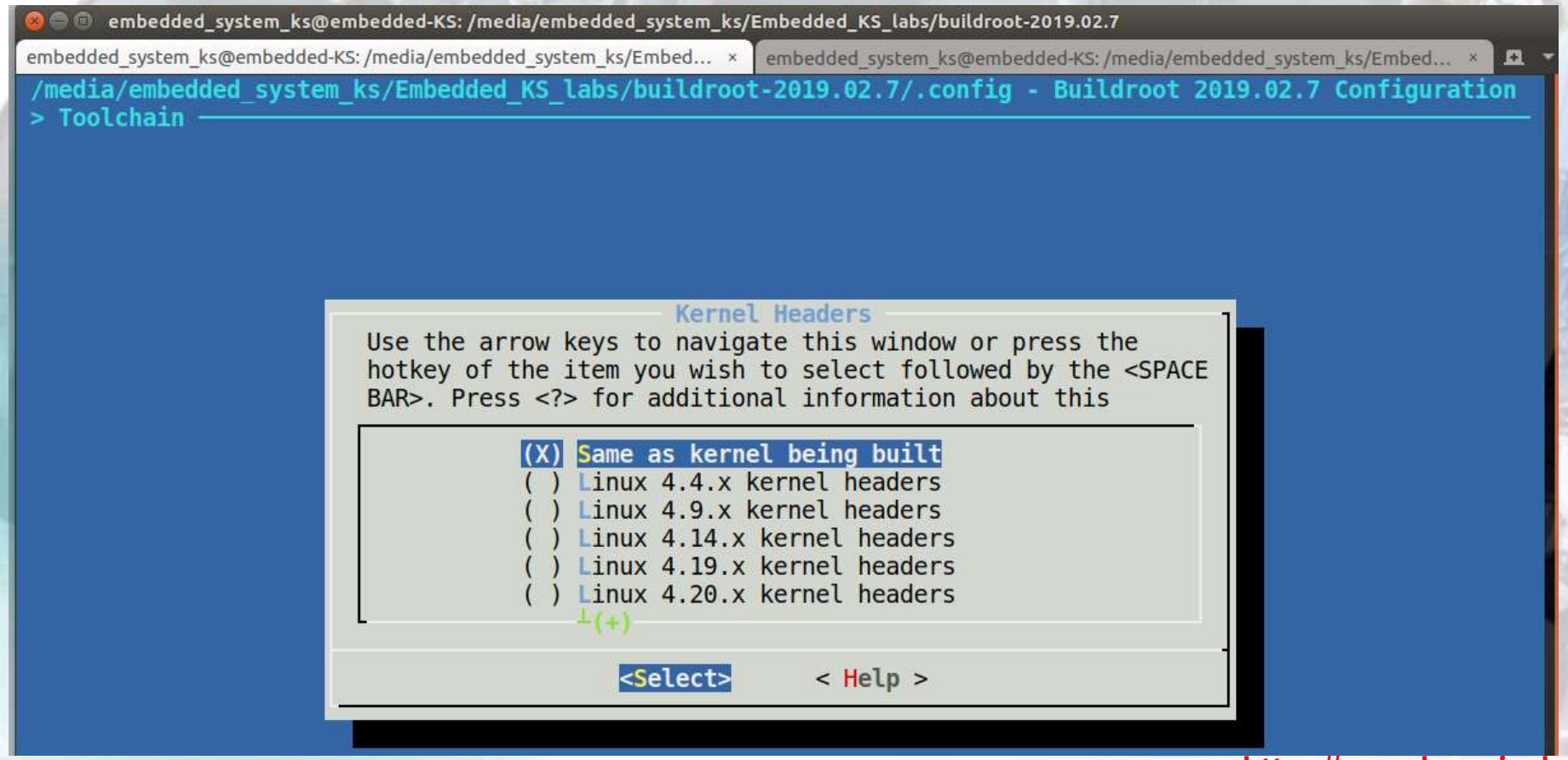
Toolchain



Library (small size version) containing the typical C libraries used in Linux environments (stdlib, stdio, etc)

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

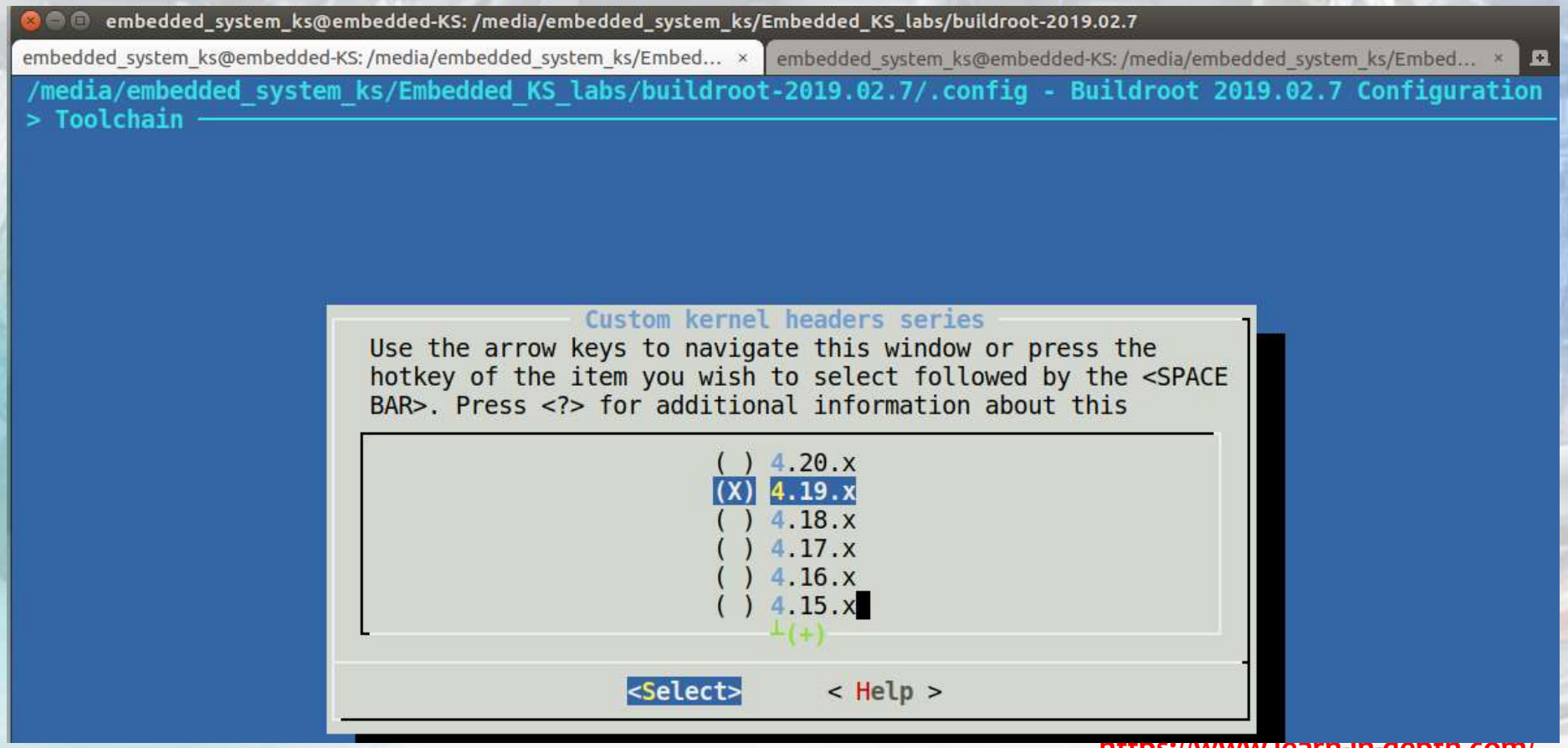
Configuring Buildroot for vexpress CortexA9.
Toolchain



Source header files of the Linux Kernel.

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Configuring Buildroot for vexpress CortexA9.
Toolchain



Configuring Buildroot for vexpress CortexA9.
Toolchain

embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_KS_labs/buildroot-2019.02.7

embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embed... × | embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embed... ×

/media/embedded_system_ks/Embedded_KS_labs/buildroot-2019.02.7/.config - Buildroot 2019.02.7 Configuration
> Toolchain

Custom kernel headers series

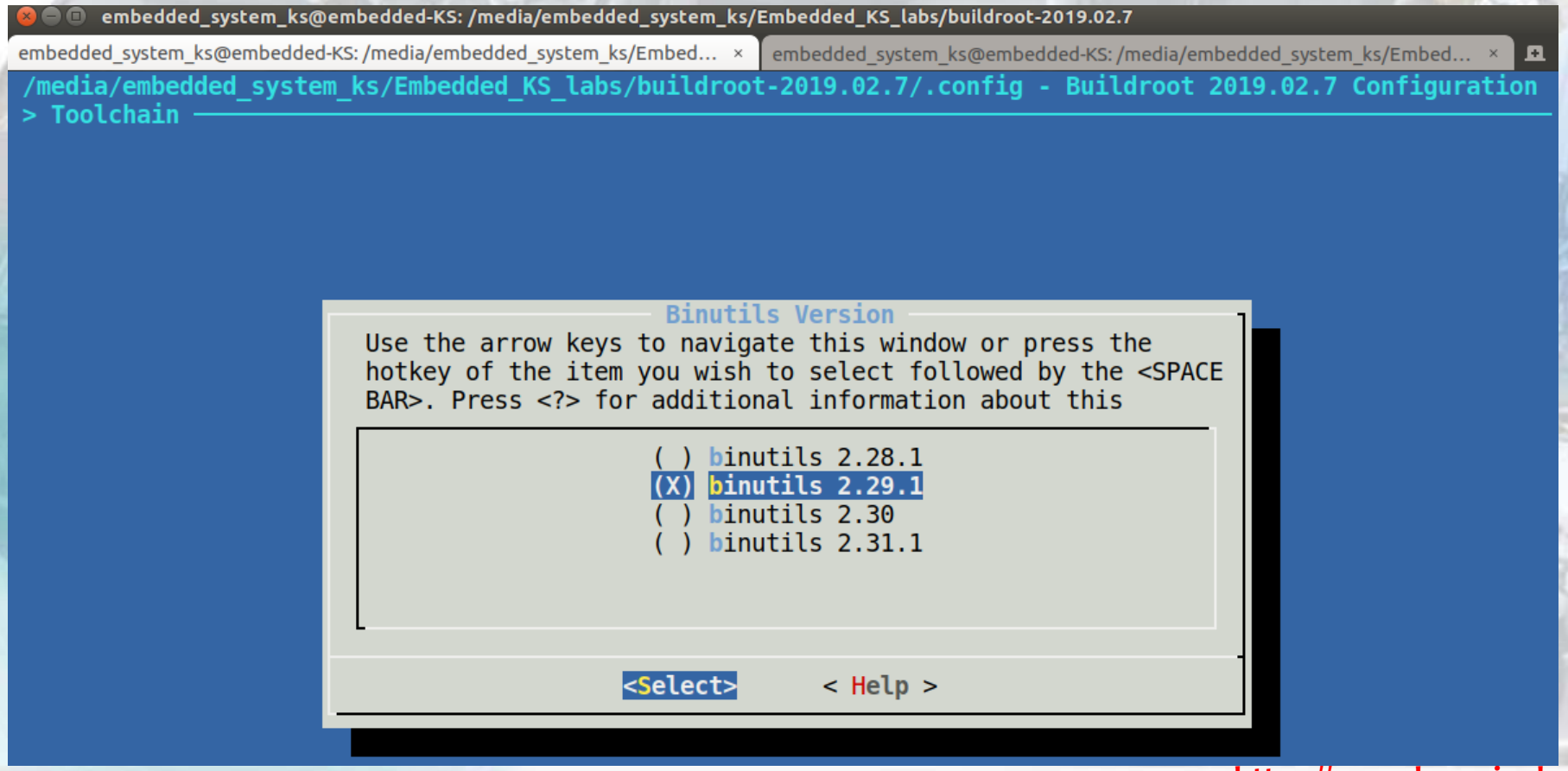
Use the arrow keys to navigate this window or press the hotkey of the item you wish to select followed by the <SPACE BAR>. Press <?> for additional information about this

( ) 4.20.x
(X) 4.19.x
( ) 4.18.x
( ) 4.17.x
( ) 4.16.x
( ) 4.15.x
(+)

<Select> < Help >

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

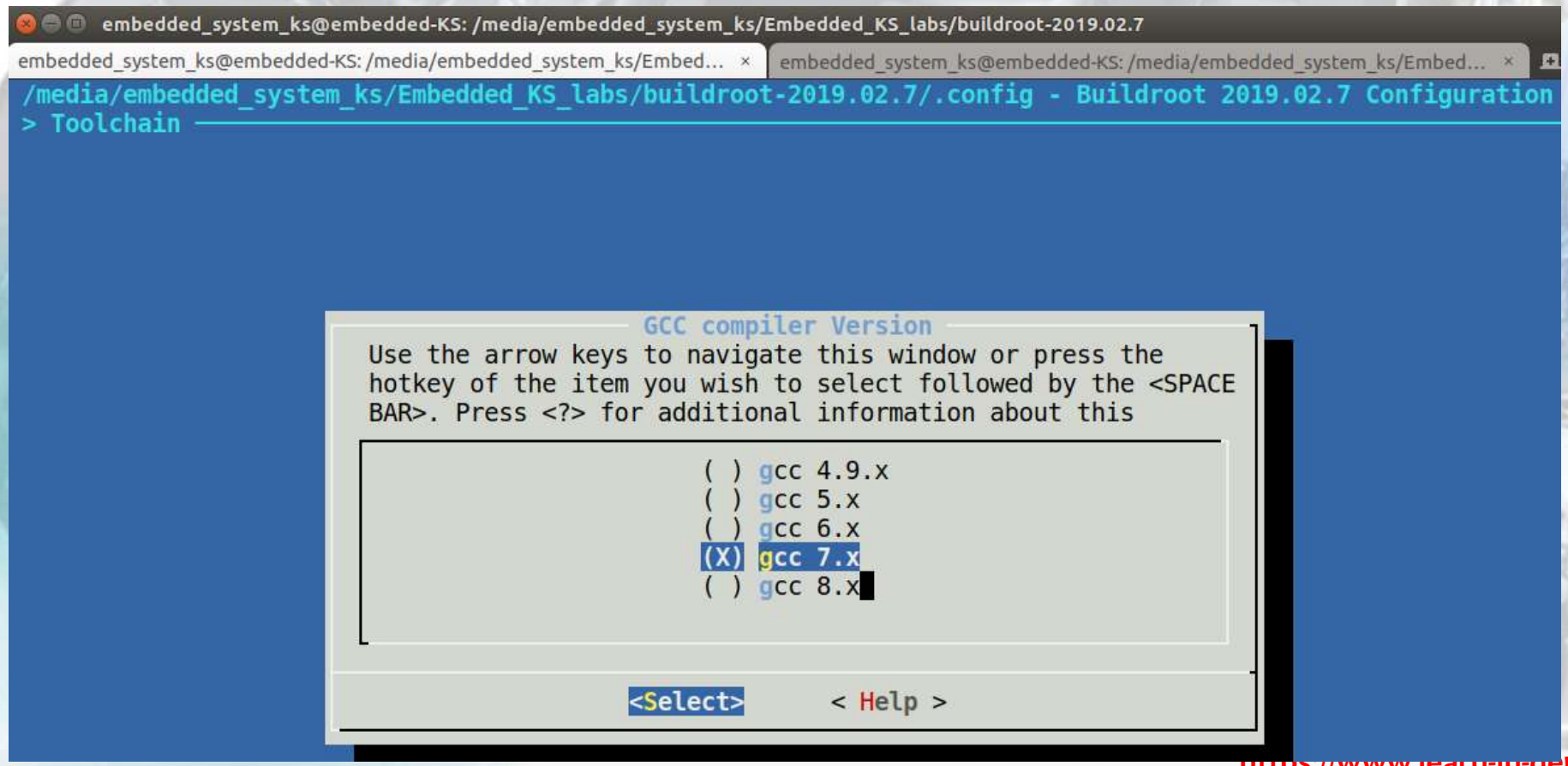
Configuring Buildroot for vexpress CortexA9.
Toolchain



Binutils contains tools to manage the binary files obtained in the compilation of the different applications

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

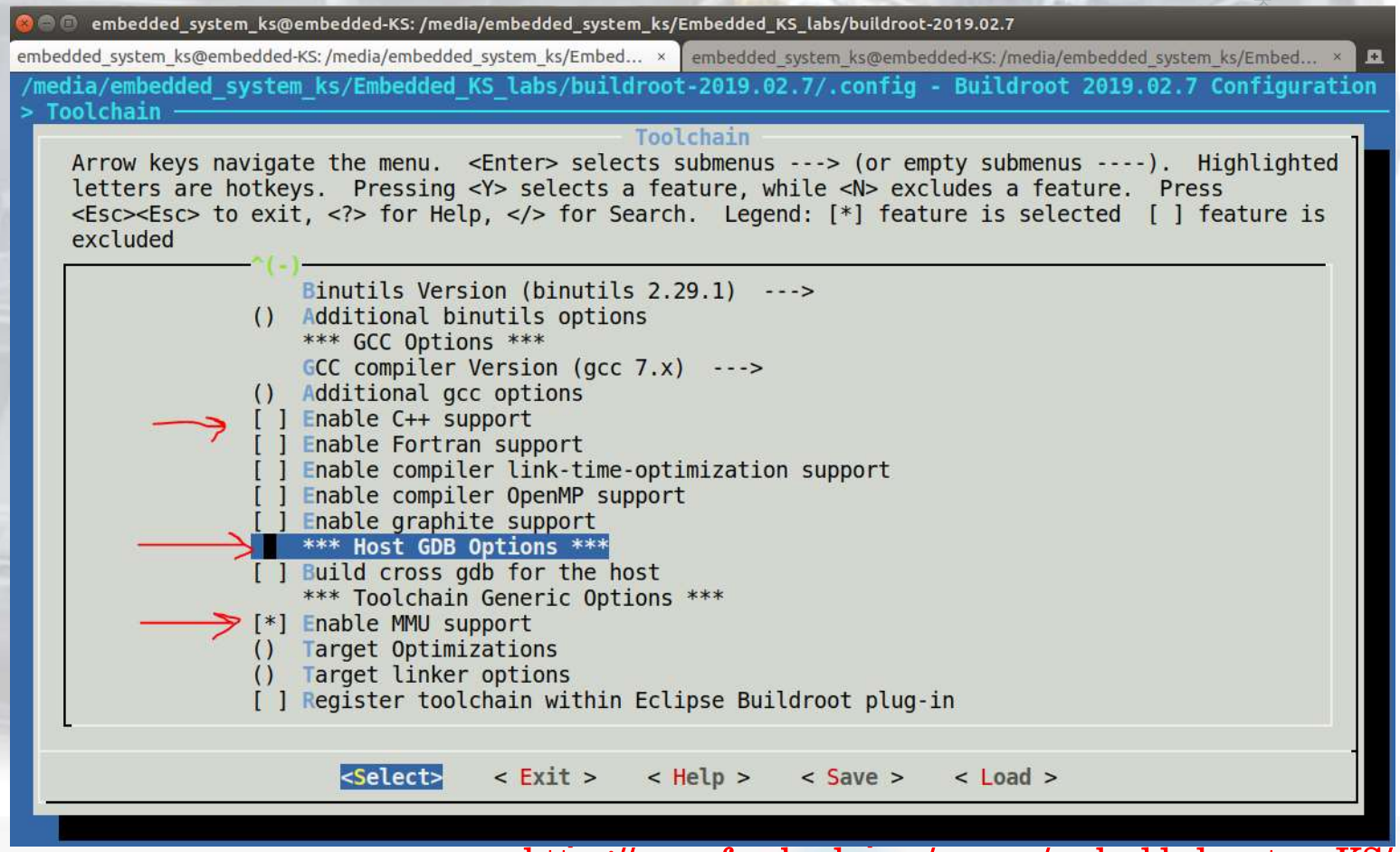
Configuring Buildroot for vexpress CortexA9.
Toolchain



GCC tools version to be installed

Configuring Buildroot for vexpress CortexA9. Toolchain

- Build cross gdb for the host
  - Includes the support for GDB. GCC debugger.
- Enable C++ support
  - Including support for C++ programming, compiling, and linking.
- Enable MMU support
  - Yes
  - Mandatory if building a Linux system

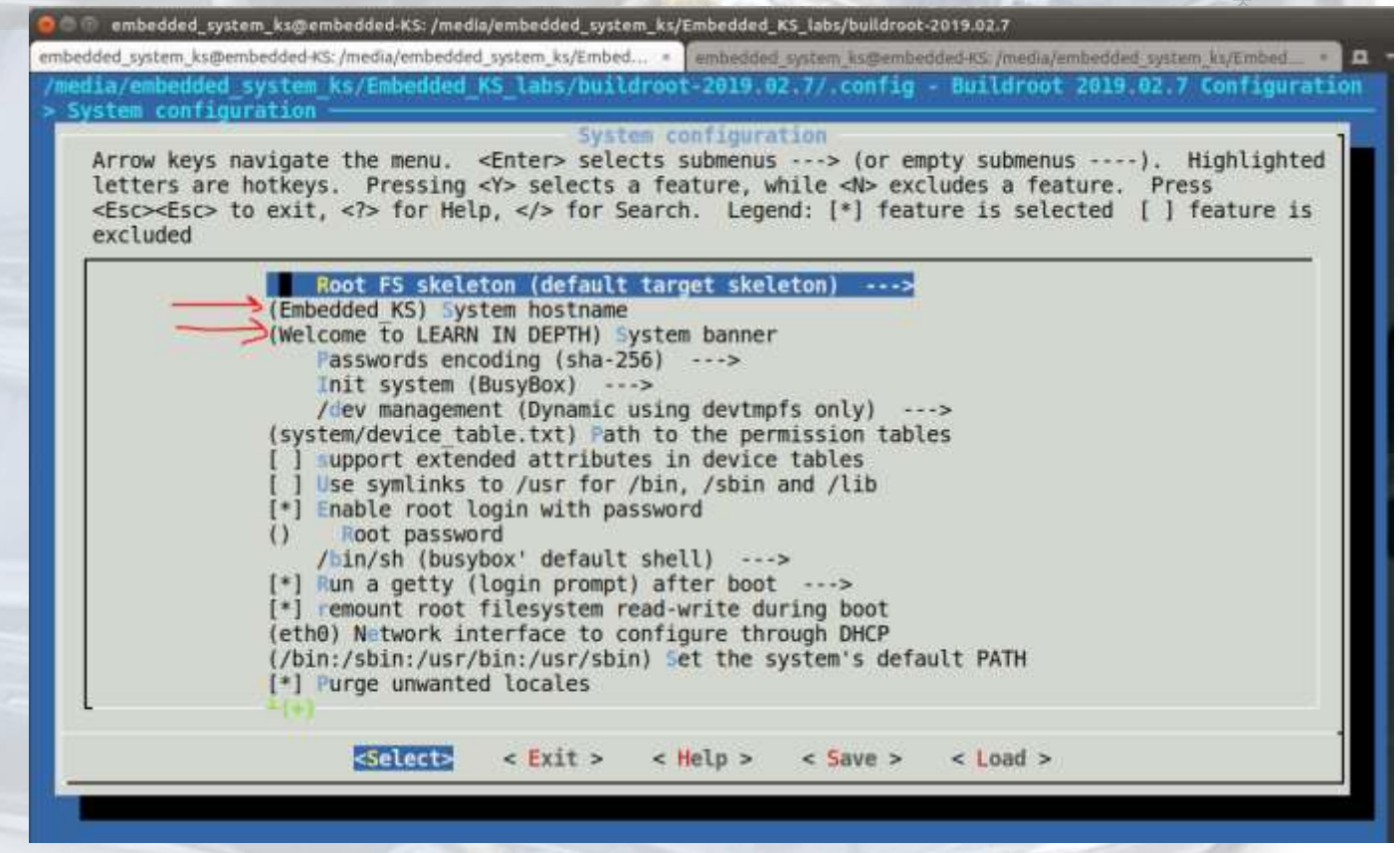


https://www.facebook.com/groups/embedded.system.KS/

Configuring Buildroot for vexpress CortexA9.

System Configuration

- System Hostname
  - Name of the embedded system
- System Banner
  - Banner



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Configuring Buildroot for vexpress CortexA9.

System Configuration

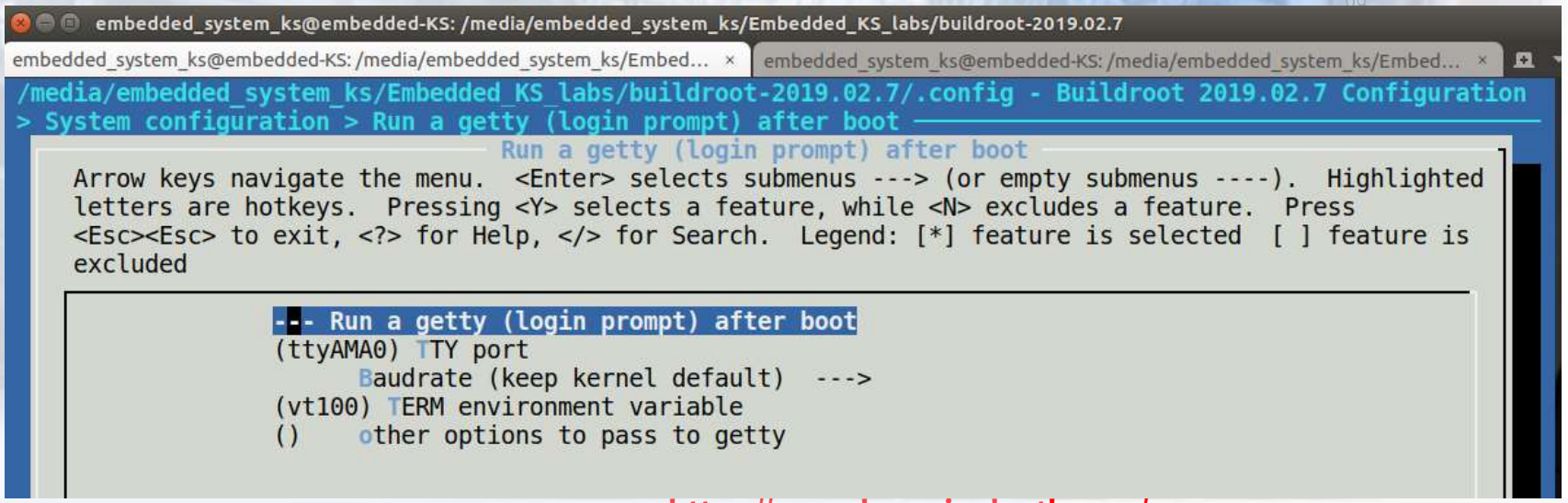
- Init System
  - Busybox
- /dev management
  - Dynamic using devtmpfs only

```
                    Init system
Use the arrow keys to navigate this window or press the
hotkey of the item you wish to select followed by the <SPACE
BAR>. Press <?> for additional information about this

(X) BusyBox
( ) systemV
    *** systemd needs a glibc toolchain w/ SSP, headers >=
( ) None

                <Select>        < Help >
```

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_KS_labs/buildroot-2019.02.7
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embed... ×  embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embed... ×
/media/embedded_system_ks/Embedded_KS_labs/buildroot-2019.02.7/.config - Buildroot 2019.02.7 Configurati
> System configuration
                    System configuration
Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted
letters are hotkeys.  Pressing <Y> selects a feature, while <N> excludes a feature.  Press
<Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] feature is selected  [ ] feature is
excluded

        Root FS skeleton (default target skeleton)  --->
    (Embedded_KS) System hostname
    (Welcome to LEARN IN DEPTH) System banner
        Passwords encoding (sha-256)  --->
        Init system (BusyBox)  --->
        /dev management (Dynamic using devtmpfs only)  --->
```

Configuring Buildroot for vexpress CortexA9.
System Configuration

- Run a getty: Port to run a getty
- **/dev/ttyAMA0 console**
  - Linux device file with the port to run getty (login) process. Uses ttyAMA0 for serial port
- Baud rate to use
  - 115200



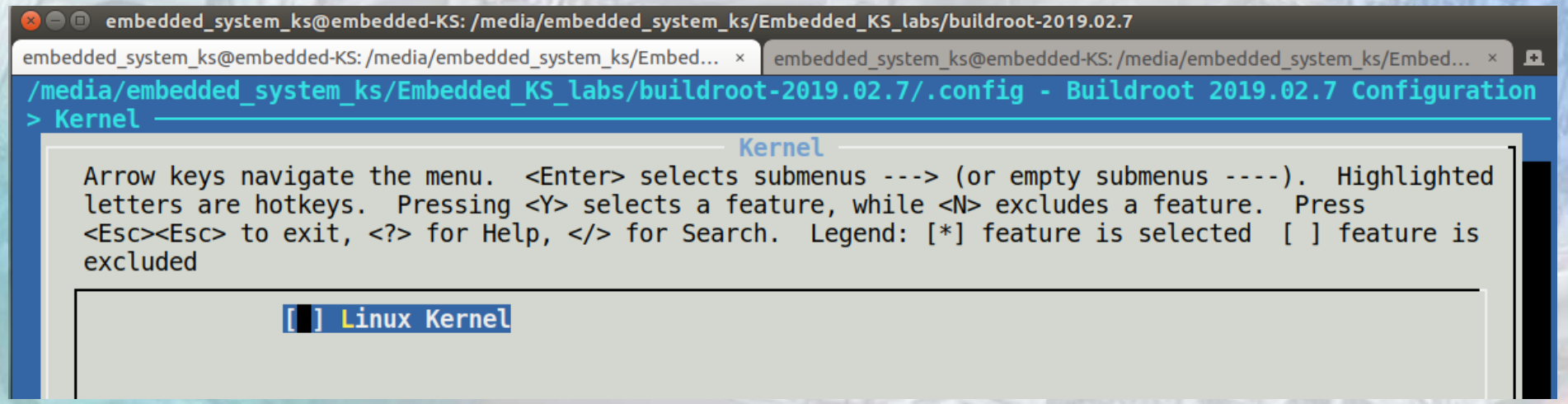
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Configuring Buildroot for vexpress CortexA9.

System Configuration

```
    /bin/sh (busybox default shell)  --->
[*] Run a getty (login prompt) after boot  --->
[*] remount root filesystem read-write during boot
(eth0) Network interface to configure through DHCP
(/bin:/sbin:/usr/bin:/usr/sbin) Set the system's default PATH
```

Configuring Buildroot for vexpress CortexA9.
kernel

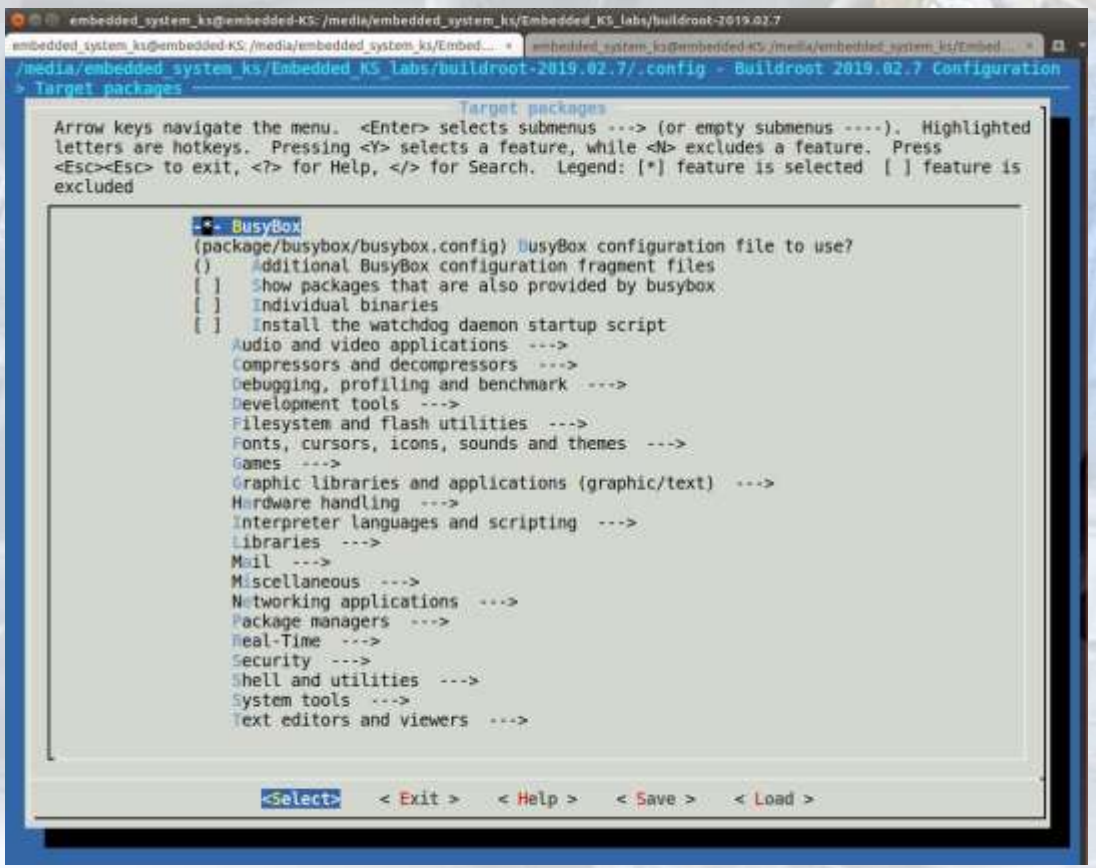


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Configuring Buildroot for vexpress CortexA9.

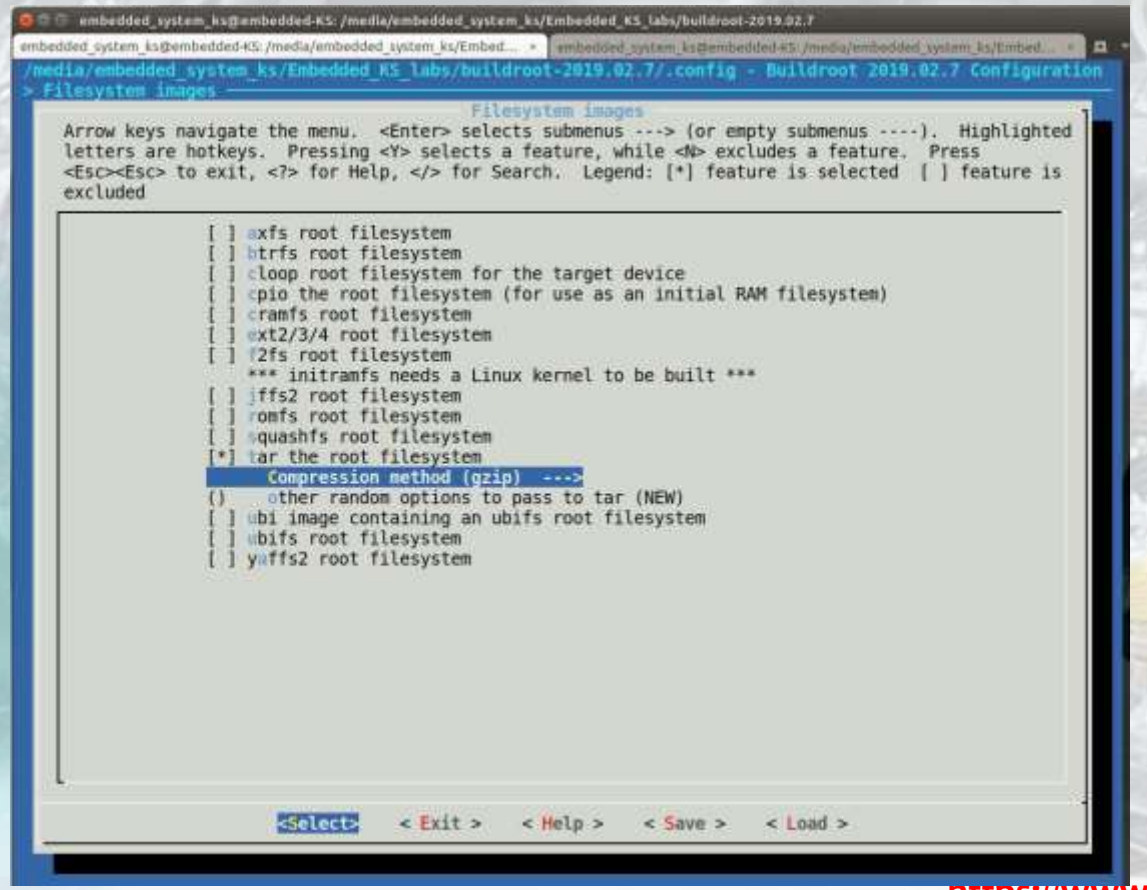
Target Packages

- Busybox



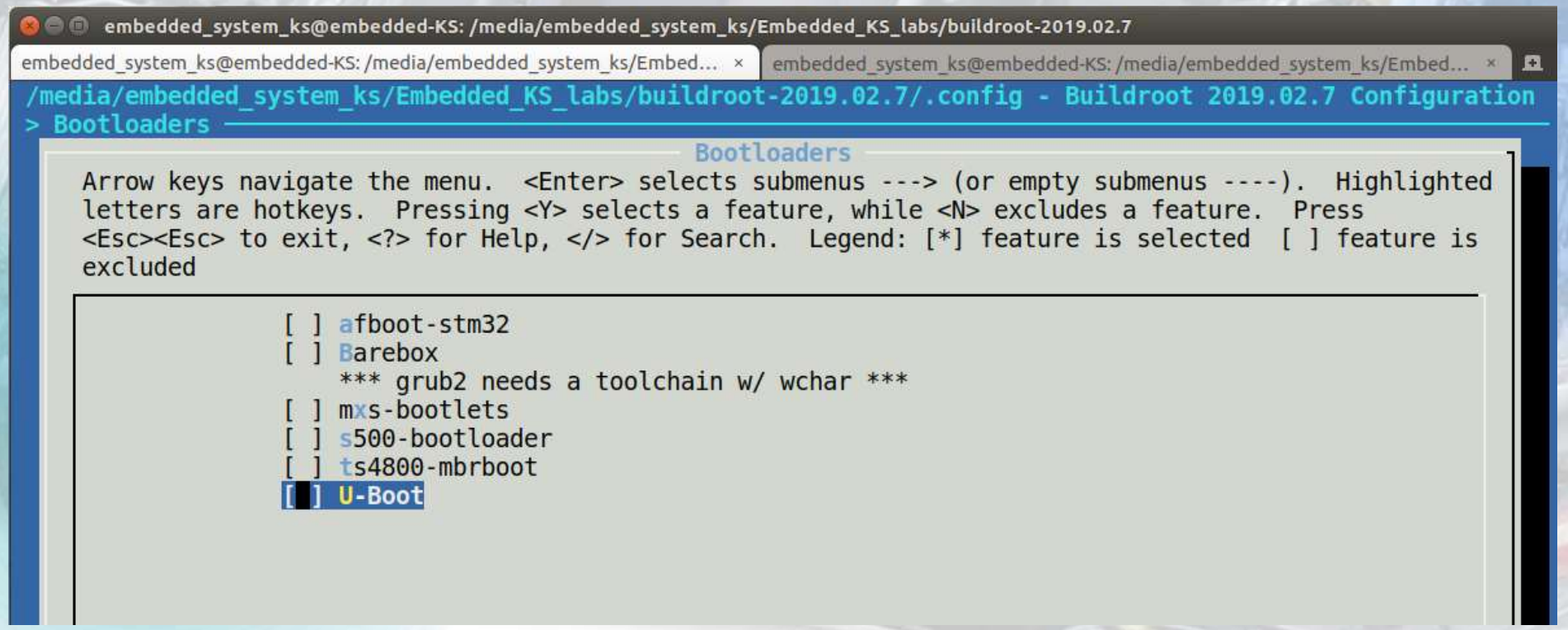
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Configuring Buildroot for vexpress CortexA9.
Filesystem Images



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Configuring Buildroot for vexpress CortexA9.
Bootloaders

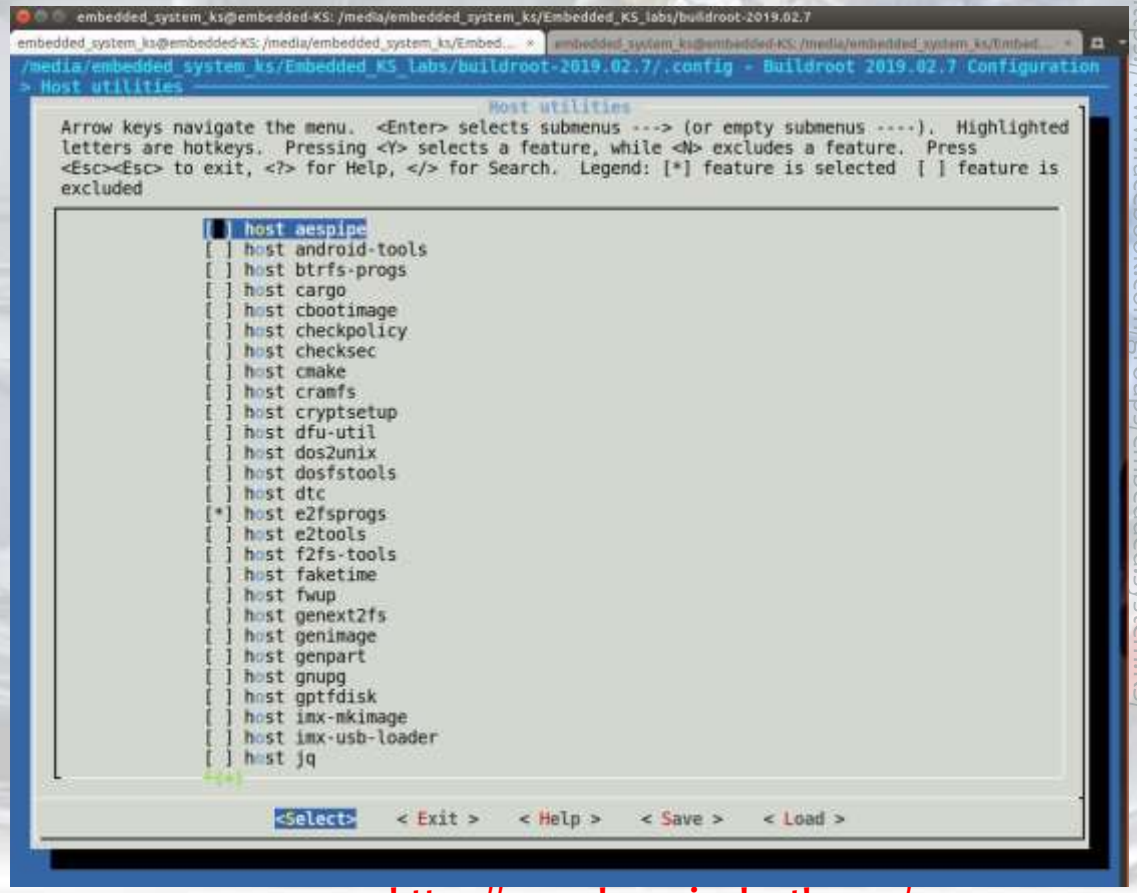


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Configuring Buildroot for vexpress CortexA9.

Host utilities

- Choose the packages which will be needed by host.



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Compiling buildroot

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/buildroot-2019.02.7$ make -j 18
/usr/bin/make -j1 O=/media/embedded_system_ks/Embedded_KS_labs/buildroot-2019.02.7/output HOSTCC="/usr/bin/gcc" HOSTCXX="/usr/bin/g++" syncconfig
make[1]: warning: -jN forced in submake: disabling jobserver mode.
make[1]: Entering directory '/media/embedded_system_ks/Embedded_KS_labs/buildroot-2019.02.7'
make[1]: Leaving directory '/media/embedded_system_ks/Embedded_KS_labs/buildroot-2019.02.7'
>>> host-skeleton  Extracting
>>> host-skeleton  Patching
>>> host-skeleton  Configuring
>>> host-skeleton  Building
>>> host-skeleton  Installing to host directory
attr-2.4.48.tar.gz: OK (sha256: 5ead72b358ec709ed00bbf7a9eaef1654baad937c001c044fe8b74c57f5324e7)
>>> host-attr 2.4.48 Extracting
gzip -d -c /media/embedded_system_ks/Embedded_KS_labs/buildroot-2019.02.7/dl/attr/attr-2.4.48.tar.gz | tar --strip-components=1 -C /media/embedded_system_ks/Embedded_KS_labs/buildroot-2019.02.7/output/build/host-attr-2.4.48 -xf -
>>> host-attr 2.4.48 Patching

Applying 0001-build-with-older-GCCs.patch using patch:
patching file include/attributes.h

Applying 0002-Switch-back-to-syscall.patch using patch:
patching file libattr/syscalls.c
>>> host-attr 2.4.48 Updating config.sub and config.guess
for file in config.guess config.sub; do for i in $(find /media/embedded_system_ks/Embedded_KS_labs/buildroot-2019.02.7/output/build/host-attr-2.4.48 -name $file); do cp support/gnuconfig/$file $i; done; done
>>> host-attr 2.4.48 Patching libtool
patching file /media/embedded_system_ks/Embedded_KS_labs/buildroot-2019.02.7/output/build/host-attr-2.4.48/build-aux/ltmain.sh
Hunk #1 succeeded at 2741 (offset 54 lines).
Hunk #2 succeeded at 4331 (offset 54 lines).
Hunk #3 succeeded at 6627 (offset 73 lines).
Hunk #4 succeeded at 6637 (offset 73 lines).
Hunk #5 succeeded at 6930 (offset 73 lines).
Hunk #6 succeeded at 7222 (offset 73 lines).
Hunk #7 succeeded at 8192 (offset 80 lines).
Hunk #8 succeeded at 10822 (offset 112 lines).
>>> host-attr 2.4.48 Configuring
(cd /media/embedded_system_ks/Embedded_KS_labs/buildroot-2019.02.7/output/build/host-attr-2.4.48/ && rm -rf config.cache; PATH="/media/embedded_system_ks/Embedded_KS_labs/buildroot-2019.02.7/output/host/bin:/media/embe
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Prepare SDCARD

- Copy the RootFS to the second partition

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/LABS/Linux_LABS/lab1$ ./mount_sd.sh
```

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/LABS/Linux_LABS/lab1/mount_p2$ sudo
 rm -rf bin/ dev/ etc/ lib/ linuxrc mnt/ proc/ root/ rootfs.tar.gz sbin/ sys/ tmp/ usr/ var/
```

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/LABS/Linux_LABS/lab1/mount_p2$ sudo cp ../../../../build
root-2019.02.7/output/images/rootfs.tar .
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/LABS/Linux_LABS/lab1/mount_p2$
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Prepare SDCARD Cont.

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/LABS/Linux_LABS/lab1/mount_p2$ sudo tar xvf rootfs.tar
./
./bin/
./bin/arch
./bin/ash
./bin/base64
./bin/busybox
./bin/cat
./bin/chattr
./bin/chgrp
./bin/chmod
./bin/chown
./bin/cp
./bin/cpio
./bin/date
./bin/dd
./bin/df
./bin/dmesg
./bin/dnsdomainname
./bin/dumpkmap
./bin/echo
./bin/egrep
./bin/false
./bin/fdflush
./bin/fgrep
./bin/getopt
./bin/grep
./bin/gunzip
./bin/gzip
./bin/hostname
./bin/kill
./bin/link
./bin/linux32
./bin/linux64
./bin/ln
./bin/login
./bin/ls
./bin/lsattr
./bin/mkdir
./bin/mknod
./bin/mktemp
```

Don’t forget to crate a file under /dev with ttyAMAo

sudo mknod rootfs/dev/ttyAMA0 c 204 64

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/LABS/Linux_LABS/lab1/mount_p2$
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/LABS/Linux_LABS/lab1/mount_p2$ cd ..
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/LABS/Linux_LABS/lab1$ ./umount_sd.sh
losetup: /dev/loop1: detach failed: No such device or address
losetup: /dev/loop2: detach failed: No such device or address
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/LABS/Linux_LABS/lab1$
```

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Booting LINUX

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_KS_labs/LABS/Linux_LABS/lab1
Amazon
embedded_system_ks@embedded-KS: /media/emb...   embedded_system_ks@embedded-KS: /media/emb...   embedded_system_ks@embedded-KS: /media/emb...
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/LABS/Linux_LABS/lab1$ ./lab1.sh u-boot KS
qemu-system-arm: -redir tcp:5525::22: The -redir option is deprecated. Please use '-netdev user,hostfwd=...' instea
qemu-system-arm: -redir tcp:8000::8000: The -redir option is deprecated. Please use '-netdev user,hostfwd=...' inst
WARNING: Image format was not specified for 'KS_SD_70M.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restri
         Specify the 'raw' format explicitly to remove the restrictions.
pulseaudio: set_sink_input_volume() failed
pulseaudio: Reason: Invalid argument
pulseaudio: set_sink_input_mute() failed
pulseaudio: Reason: Invalid argument

U-Boot 2020.01-rc2-00035-g3ff1ff3ff7-dirty (Nov 21 2019 - 16:47:34 +0200)

DRAM:  512 MiB
WARNING: Caches not enabled
Flash: 128 MiB
MMC:   MMC: 0
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   smc911x-0
Hit any key to stop autoboot:  0
=> setenv bootargs  'rw earlyprintk loglevel=8 root=/dev/mmcblk0p2  console=ttyAMA0 dhcp'
=> fatload mmc 0:1 0x60000000 zImage
4594264 bytes read in 3065 ms (1.4 MiB/s)
=> fatload mmc  0:1 0x63000000 vexpress-v2p-ca9.dtb
14143 bytes read in 34 ms (405.3 KiB/s)
=> bootz 0x60000000 - 0x63000000
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

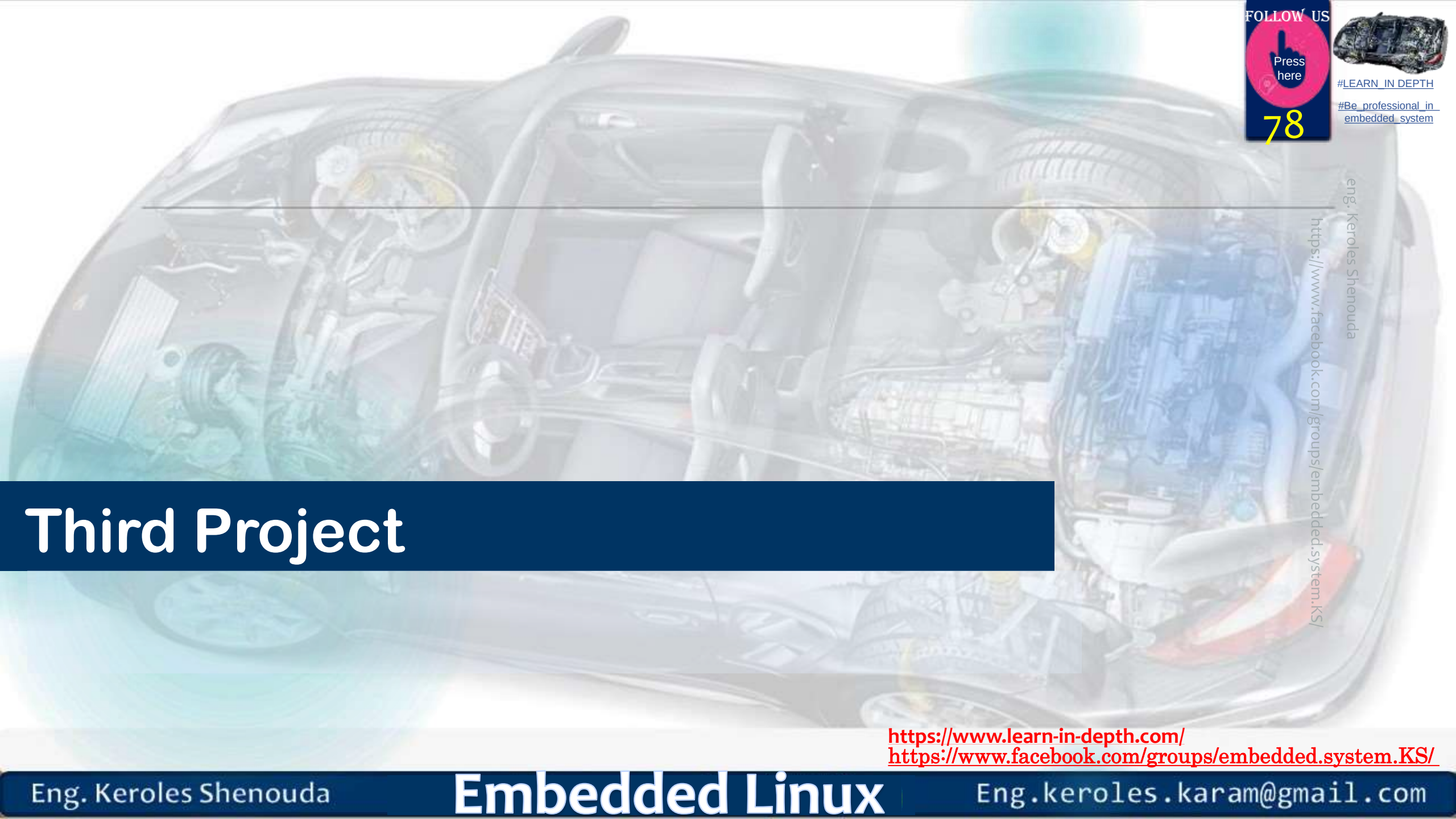
#LEARN_IN DEPTH

#Be_professional_in_embedded_system

Reaching to login

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_KS_labs/LABS/Linux_LABS/lab1
VFS: Mounted root (ext3 filesystem) on device 179:2.
Freeing unused kernel memory: 1024K
Run /sbin/init as init process
random: crng init done
EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)
ext3 filesystem being remounted at / supports timestamps until 2038 (0x7fffffff)
Starting syslogd: OK
Starting klogd: OK
Starting network: Generic PHY 4e000000.ethernet-ffffffff:01: attached PHY driver [Generic PHY] (mii_bus:phy_addr=4e
000000.ethernet-ffffffff:01, irq=POLL)
smsc911x 4e000000.ethernet eth0: SMSC911x/921x identified at 0xa08c0000, IRQ: 18
Error: Driver 'vexpress-muxfpga' is already registered, aborting...
drm-clcd-pl111 10020000.clcd: initializing Versatile Express PL111
drm-clcd-pl111 10020000.clcd: DVI muxed to daughterboard 1 (core tile) CLCD
udhcpc: started, v1.29.3
udhcpc: sending discover
udhcpc: sending select for 10.0.2.15
udhcpc: lease of 10.0.2.15 obtained, lease time 86400
deleting routers
adding dns 10.0.2.3
OK

Welcome to LEARN IN DEPTH
Embedded_KS login: root
# ls
# ls /
bin         lib32       opt         run         usr
dev         linuxrc     proc        sbin        var
etc         media       root        sys
lib         mnt         rootfs.tar.gz  tmp
#
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

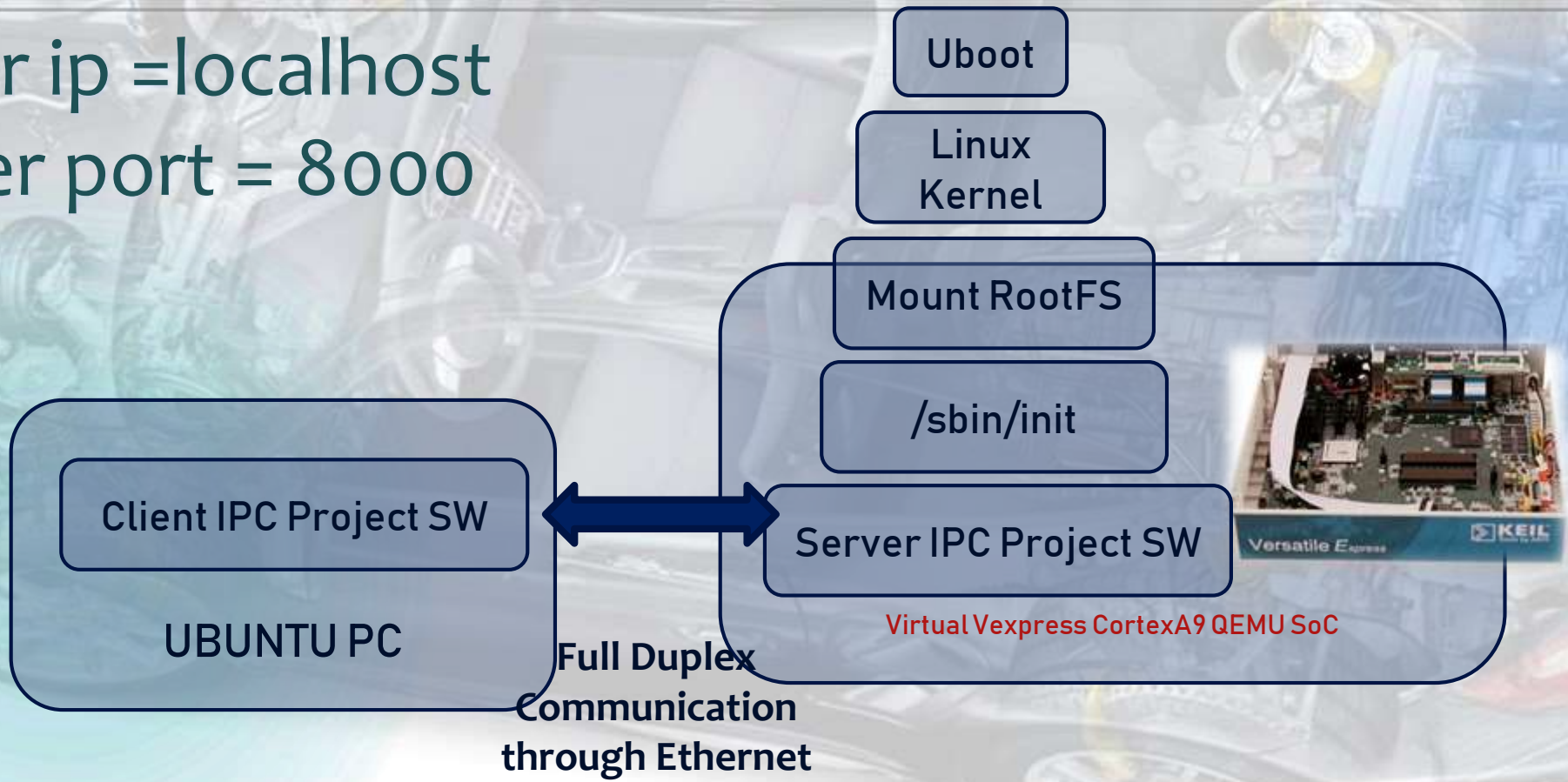


Third Project

Third Project

Server ip =localhost

Server port = 8000



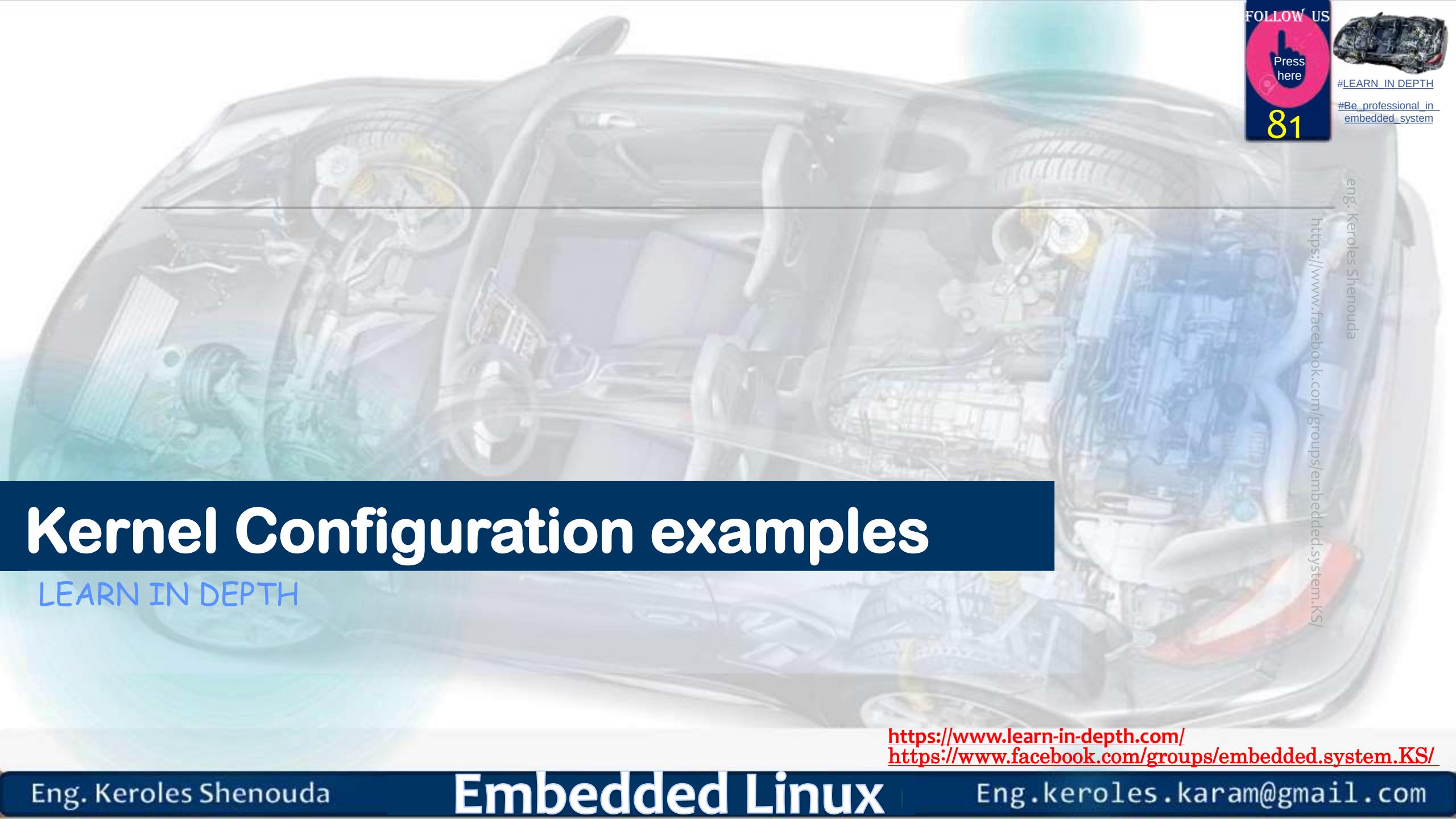
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

PART9 Appendix

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Kernel Configuration examples

LEARN IN DEPTH

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Kernel Configuration

- The kernel contains thousands of device drivers, filesystem drivers, network protocols and other configurable items
- The kernel configuration is the process of defining the set of options with which you want your kernel to be compiled
- The set of options depends
  - On the target architecture and on your hardware (for device drivers, etc.)
  - On the capabilities you would like to give to your kernel (network capabilities, filesystems, real-time, etc.). Such generic options are available in all architectures.

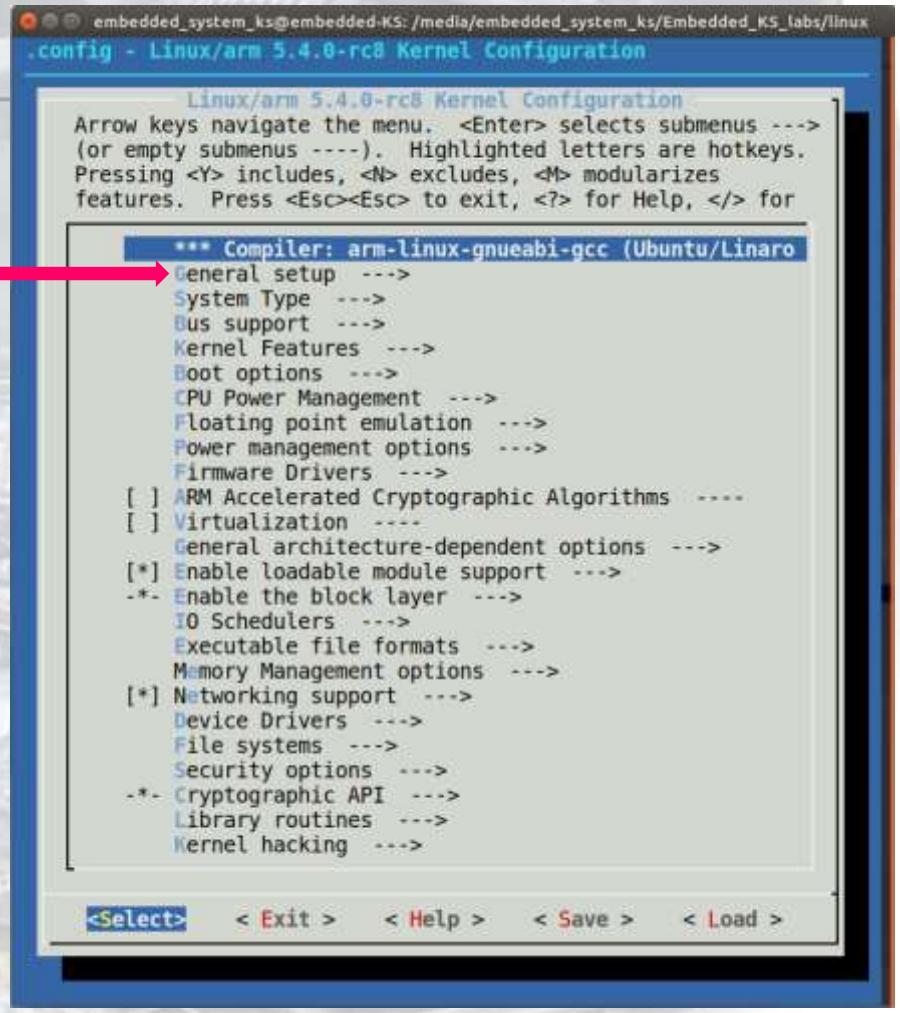
The following is a list of the main menu options available to all embedded Linux architectures:

- Code maturity level options
- General setup
- Loadable module support
- Block layer
- Networking
- Device drivers
- Filesystems
- Kernel hacking
- Security options
- Cryptographic options
- Library routines

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

General setup

- This section has three settings that you should turn on:
  - **Support for paging of anonymous memory** Enables swap disk and file support
  - **System V IPC** Enables interprocess communication
  - **Sysctl support** Enables kernel parameter changes through /proc

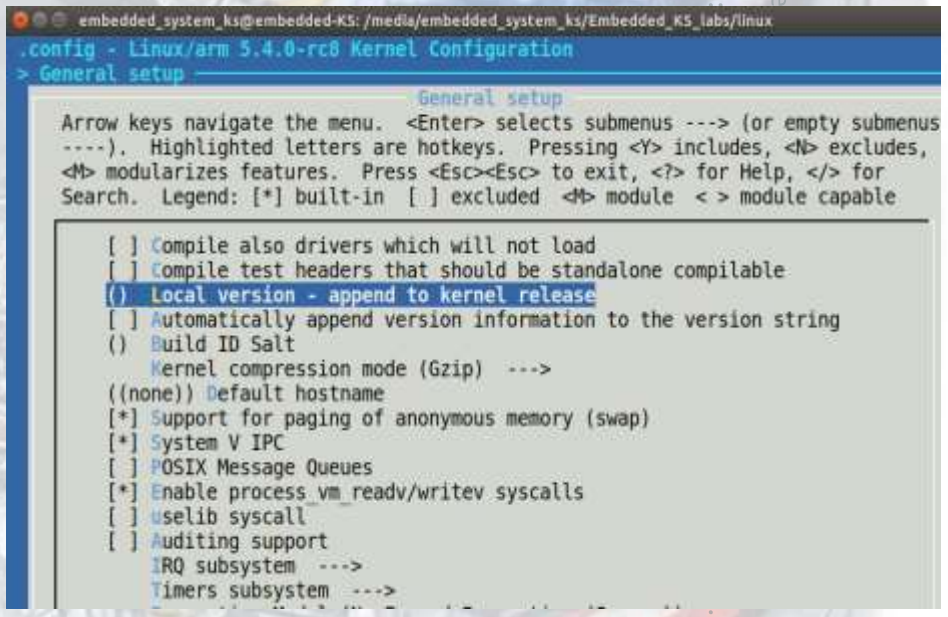


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

General setup Cont.

- **Option:** "Local version - append to kernel release"
  - Variable name: LOCALVERSION
  - Append an extra string to the end of your kernel version. This will show up when you type uname
- **Option: Automatically append version information to the version string**
  - Variable name: LOCALVERSION_AUTO

```
This will try to automatically determine if the current tree is a
release tree by looking for git tags that belong to the current
top of tree revision.
```



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

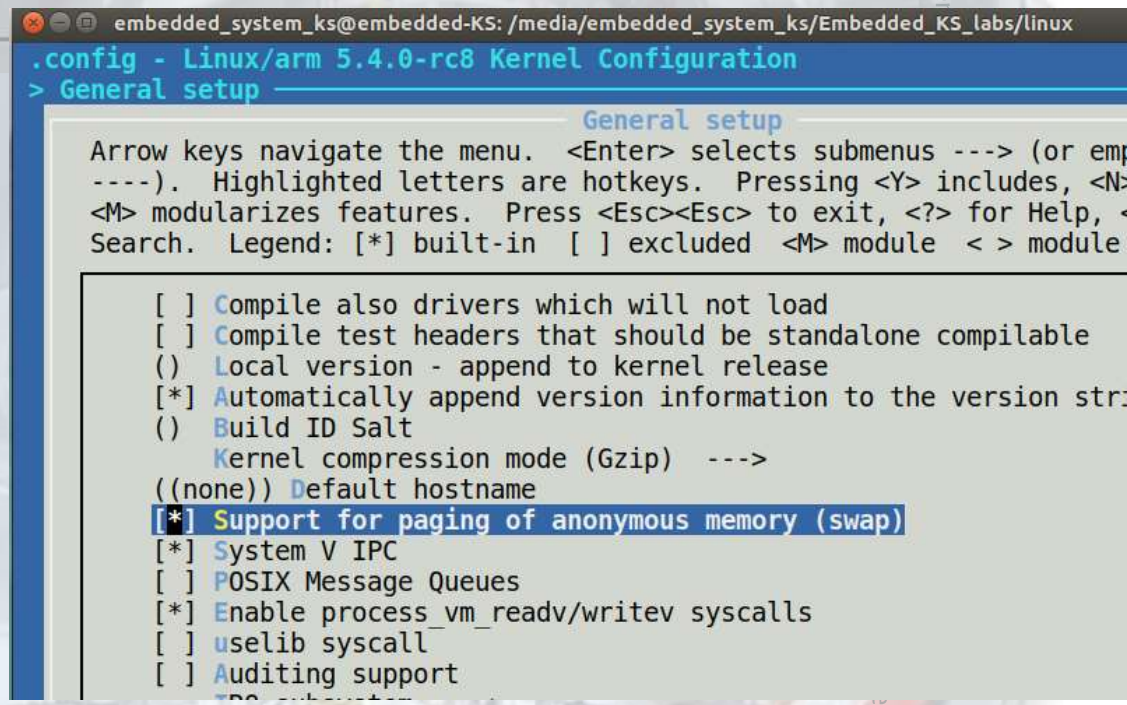
General setup Cont.

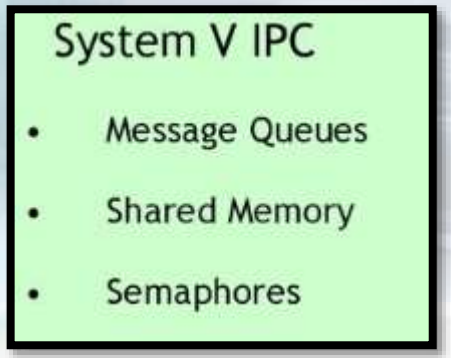
- Option: Support for paging of anonymous memory (swap)
  - Variable name: SWAP
  - depends on MMU
  - This option allows you to choose whether you want to have support for so called swap devices or swap files in your kernel that are used to provide more virtual memory than the actual RAM present in your computer


```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_KS_labs/linux
.config - Linux/arm 5.4.0-rc8 Kernel Configuration
> General setup
                              General setup
Arrow keys navigate the menu.  <Enter> selects submenus ---> (or emp
----).  Highlighted letters are hotkeys.  Pressing <Y> includes, <N>
<M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, <
Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module

    [ ] Compile also drivers which will not load
    [ ] Compile test headers that should be standalone compilable
    ()  Local version - append to kernel release
    [*] Automatically append version information to the version stri
    ()  Build ID Salt
        Kernel compression mode (Gzip)  --->
    ((none)) Default hostname
    [*] Support for paging of anonymous memory (swap)
    [*] System V IPC
    [ ] POSIX Message Queues
    [*] Enable process_vm_readv/writev syscalls
    [ ] uselib syscall
    [ ] Auditing support
```

General setup Cont.

- Option: System V IPC
  - Variable name: SYSVIPC
  - depends on MMU
  - Inter Process Communication is a suite of library functions and system calls which let processes (running programs) synchronize and exchange information. It is generally considered to be a good thing, and some programs won't run unless you say Y here



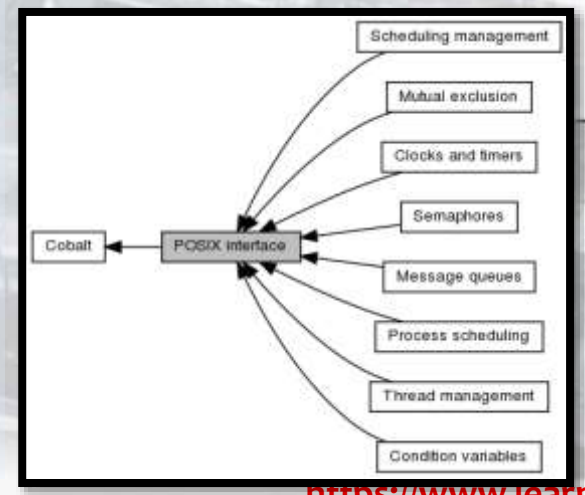


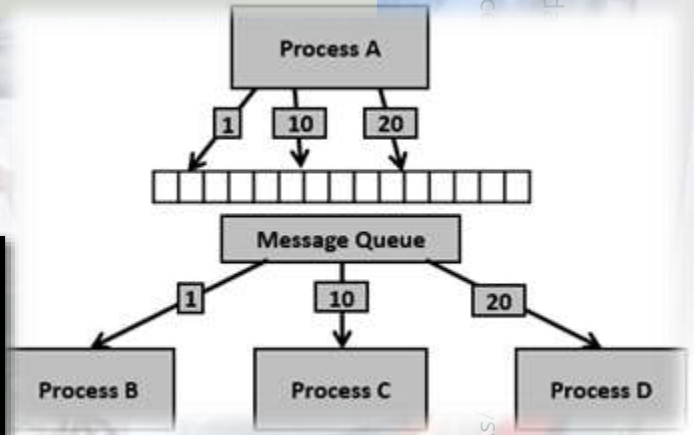
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

General setup Cont.

- Option: POSIX Message Queues
  - Variable name: POSIX_MQUEUE
  - depends on NET && EXPERIMENTAL
  - POSIX variant of message queues is a part of IPC. In POSIX message queues every message has a priority which decides about succession of receiving it by a process. If you want to compile and run programs written e.g. for Solaris with use of its POSIX message queues (functions mq_*) say Y here

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_KS_labs/linux
.config - Linux/arm 5.4.0-rc8 Kernel Configuration
> General setup
                        General setup
Arrow keys navigate the menu.  <Enter> selects submenus ---> (or emp
----).  Highlighted letters are hotkeys.  Pressing <Y> includes, <N>
<M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, <
Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module

    [ ] Compile also drivers which will not load
    [ ] Compile test headers that should be standalone compilable
    ()  Local version - append to kernel release
    [*] Automatically append version information to the version stri
    ()  Build ID Salt
        Kernel compression mode (Gzip)  --->
    ((none)) Default hostname
    [*] Support for paging of anonymous memory (swap)
    [*] System V IPC
    [ ] POSIX Message Queues
    [*] Enable process_vm_readv/writev syscalls
    [ ] uselib syscall
    [ ] Auditing support
```

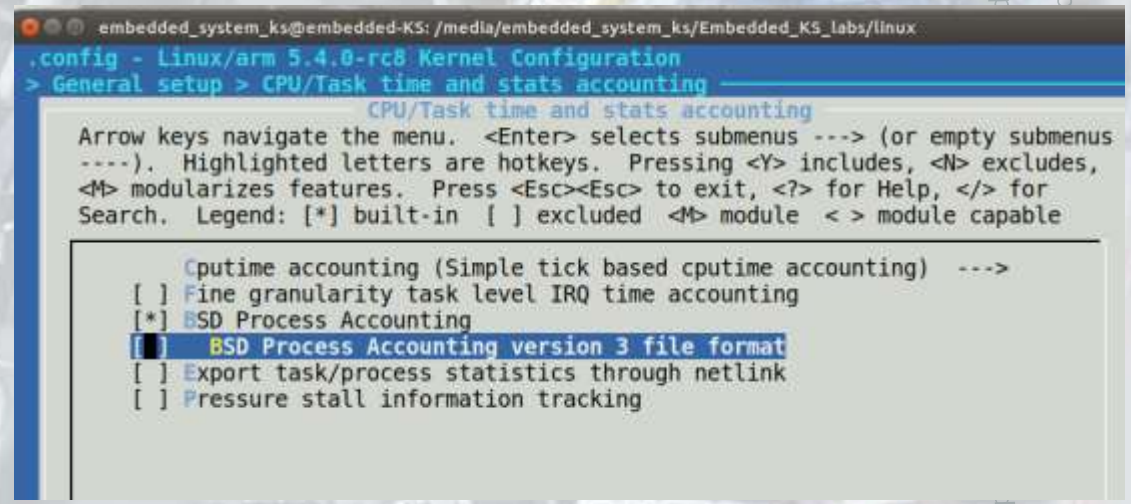




https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

General setup Cont.

- **Option: BSD Process Accounting**
  - Variable name: BSD_PROCESS_ACCT
  - If you say Y here, a user level program will be able to instruct the kernel (via a special system call) to write process accounting information to a file: whenever a process exits, information about that process will be appended to the file by the kernel
- **Option: BSD Process Accounting version 3 file format**
  - Variable name: BSD_PROCESS_ACCT_V3
  - If you say Y here, the process accounting information is written in a new file format that also logs the process IDs of each process



General setup Cont.

```
        RCU Subsystem  --->
    <M> Kernel .config support
    [*]   Enable access to .config through /proc/config.gz
```

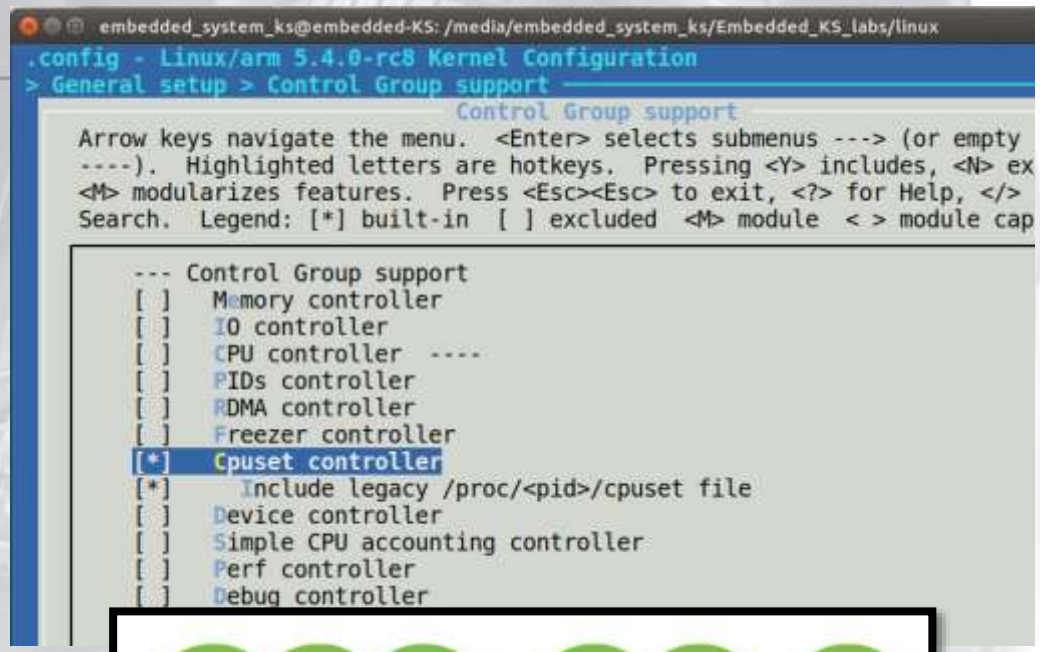
- **Option: Kernel .config support**
  - Variable name: IKCONFIG
  - This option enables the complete Linux kernel ".config" file contents to be saved in the kernel. It provides documentation of which kernel options are used in a running kernel or in an on-disk kernel. This information can be extracted from the kernel image file with the script scripts/extract-ikconfig and used as input to rebuild the current kernel or to build another kernel. It can also be extracted from a running kernel by reading /proc/config.gz if enabled (below).

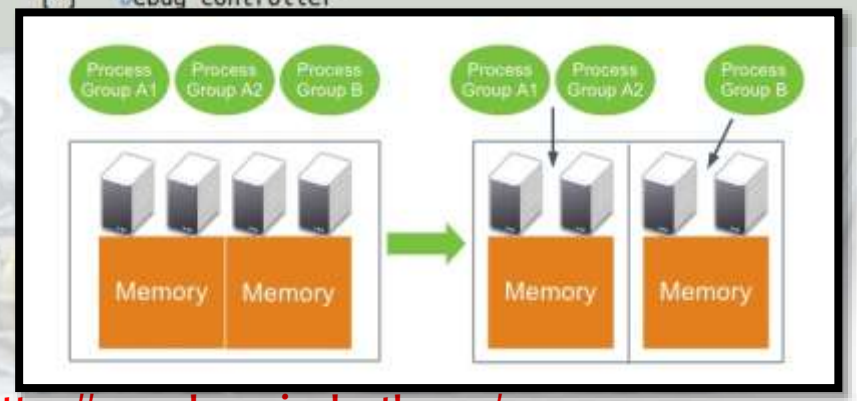
General setup Cont.

- **Option: Cpuset support**
  - Variable name: CPUSETS
  - This option will let you create and manage CPUSETs which allow dynamically partitioning a system into sets of CPUs and Memory Nodes and assigning tasks to run only within those sets. This is primarily useful on large SMP or NUMA (non uniform access memory) systems.

For example, the following sequence of commands will setup a cpuset named "Charlie", containing just CPUs 2 and 3, and Memory Node 1, and then start a subshell 'sh' in that cpuset:

```
mount -t cgroup -ocpuset cpuset /sys/fs/cgroup/cpuset
cd /sys/fs/cgroup/cpuset
mkdir Charlie
cd Charlie
/bin/echo 2-3 > cpuset.cpus
/bin/echo 1 > cpuset.mems
/bin/echo $$ > tasks
sh
# The subshell 'sh' is now running in cpuset Charlie
# The next line should display '/Charlie'
cat /proc/self/cpuset
```





https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

General setup Cont.

- **Option:** INITRAMFS_SOURCE
  - This can be either a single cpio archive with a **.cpio** suffix or a space-separated list of directories and files for building the initramfs image. A cpio archive should contain a filesystem archive to be used as an initramfs image
- **Option:** INITRAMFS_ROOT_UID
  - depends on INITRAMFS_SOURCE!=""
  - This setting is only meaningful if the INITRAMFS_SOURCE is contains a directory. Setting this user ID (UID) to something other than "0" will cause all files owned by that UID to be owned by user root in the initial ramdisk image.

```
[*] Initial RAM filesystem and RAM disk (initramfs/initrd) support
()    Initramfs source file(s)
```

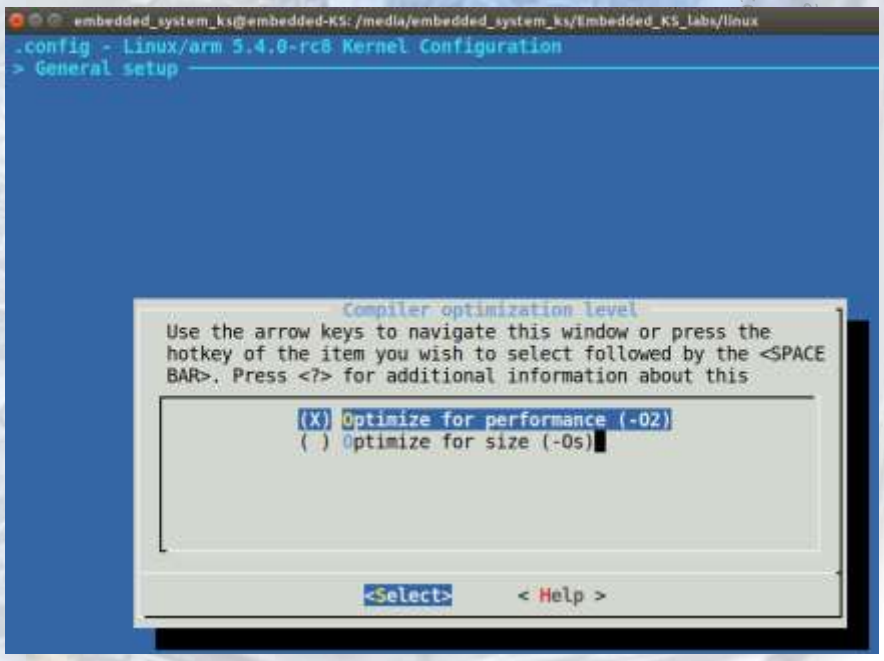
```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_KS_labs/linux
.config - Linux/arm 5.4.0-rc8 Kernel Configuration
> General setup > Search (INITRAMFS_SOURCE)
                                Search Results
Symbol: INITRAMFS_SOURCE [=]
Type  : string
Prompt: Initramfs source file(s)
  Location:
    -> General setup
(1)   -> Initial RAM filesystem and RAM disk (initramfs/initrd) support (BLK_DEV
  Defined at usr/Kconfig:6
  Depends on: BLK_DEV_INITRD [=y]
```

```
[*] Initial RAM filesystem and RAM disk (initramfs/initrd) support
(/home/dmanchon/work/project/sd_boot/rootfs.cpio) Initramfs source file(s)
(0)     User ID to map to 0 (user root)
(0)     Group ID to map to 0 (group root)
[*]   Support initial ramdisks compressed using gzip
[ ]   Support initial ramdisks compressed using bzip2
[*]   Support initial ramdisks compressed using LZMA
[ ]   Support initial ramdisks compressed using XZ
```

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

General setup Cont.

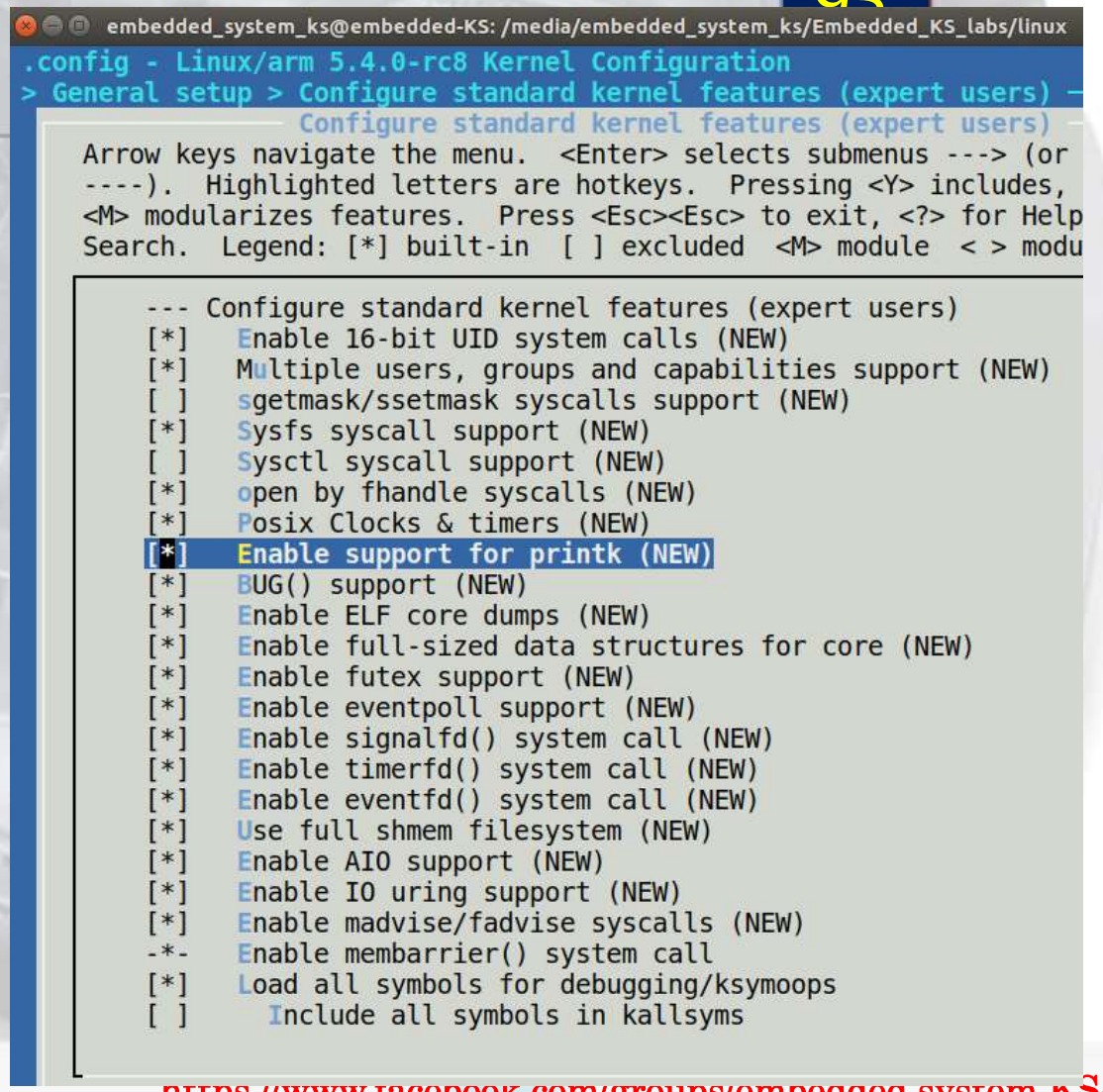
- Option: Optimize for size (Look out for broken compilers!)
  - Enabling this option will pass "-Os" instead of "-O2" to gcc resulting in a smaller kernel.
    WARNING: some versions of gcc may generate incorrect code with this option. If problems are observed, a gcc upgrade may be needed. If unsure, say N.



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

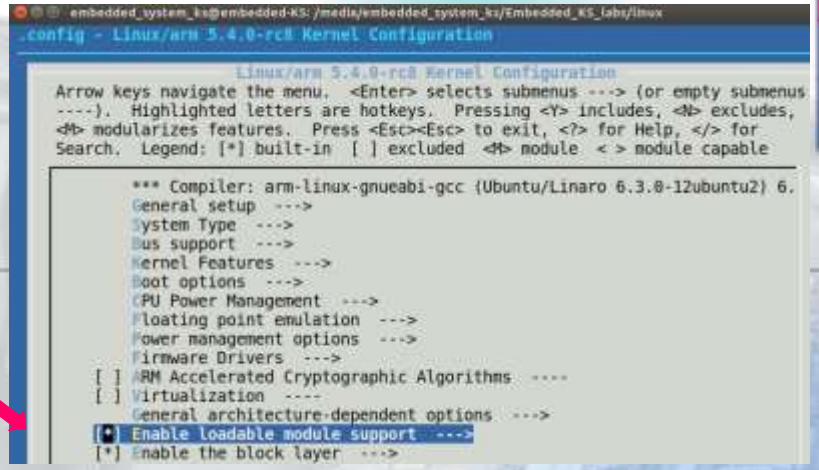
General setup Cont.

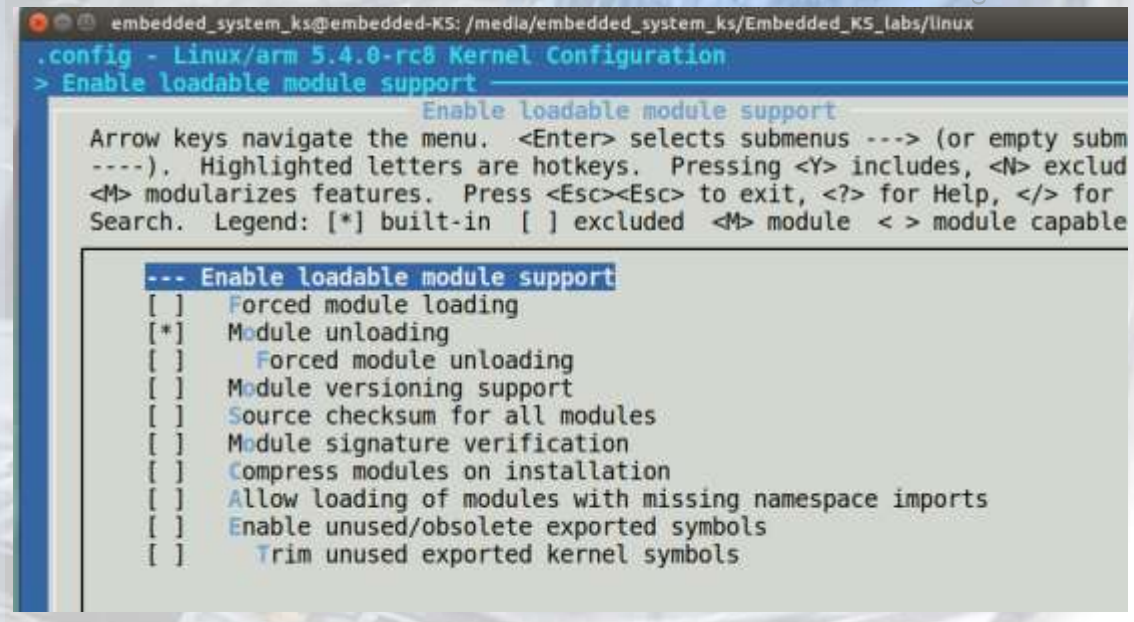
- **Option: Load all symbols for debugging/kksymoops**
  - Variable name: KALLSYMS
  - This increases the size of the kernel somewhat, as all symbols have to be loaded into the kernel image
- **Option: Include all symbols in kallsyms**
  - Variable name: KALLSYMS_ALL
  - Some debuggers can use kallsyms for other symbols too: say Y here to include all symbols, if you need them and you don't care about adding 300k to the size of your kernel. Say N.
- **Option: Enable support for printk**
  - Variable name: PRINTK
  - This option enables normal printk support. Removing it eliminates most of the message strings from the kernel image and makes the kernel more or less silent. As this makes it very difficult to diagnose system problems, saying N here is strongly discouraged.



Linux kernel/Loadable module support

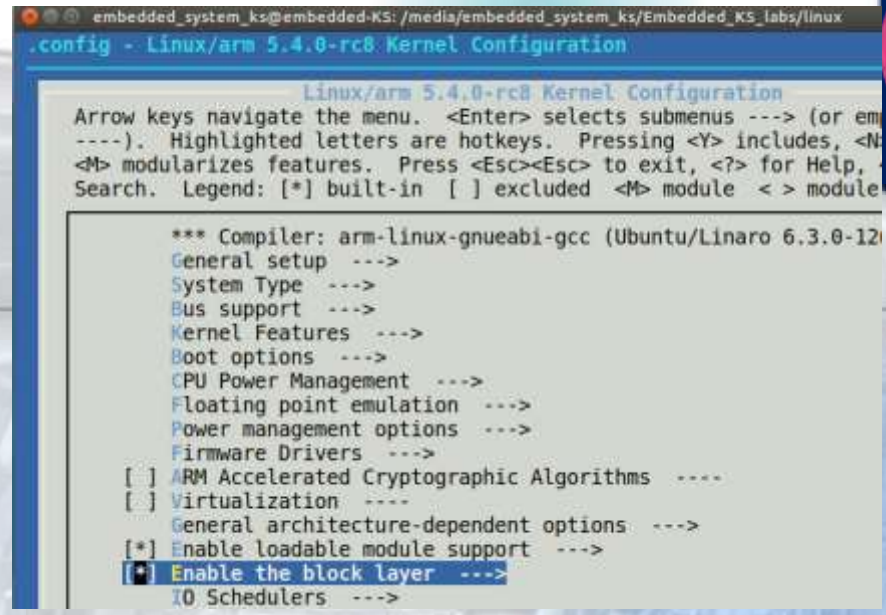
- **Option: Enable loadable module support**
  - variable name: MODULES
  - Kernel modules are small pieces of compiled code which can be inserted in the running kernel, rather than being permanently built into the kernel
- **Option: Module unloading**
  - variable name: MODULE_UNLOAD
  - Without this option you will not be able to unload any modules
- **Option: Forced module unloading**
  - variable name: MODULE_FORCE_UNLOAD
  - This option allows you to force a module to unload, even if the kernel believes it is unsafe: the kernel will remove the module without waiting for anyone to stop using it (using the -f option to rmmod).
- **Option: Module versioning support (EXPERIMENTAL)**
  - variable name: MODVERSIONS
  - Usually, you have to use modules compiled with your kernel. Saying Y here makes it sometimes possible to use modules compiled for different kernels, by adding enough information to the moduleS

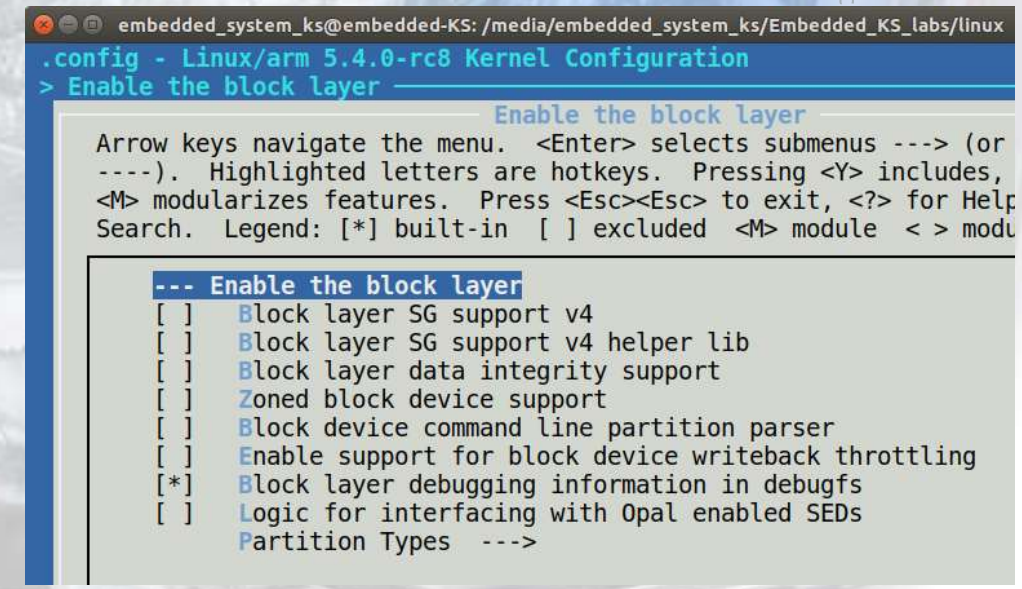




Linux kernel/Block layer

- **Option:** "Enable the block layer"
  - default y
  - If this option is disabled, then blockdev files will become unusable and some filesystems (such as ext3) will become unavailable

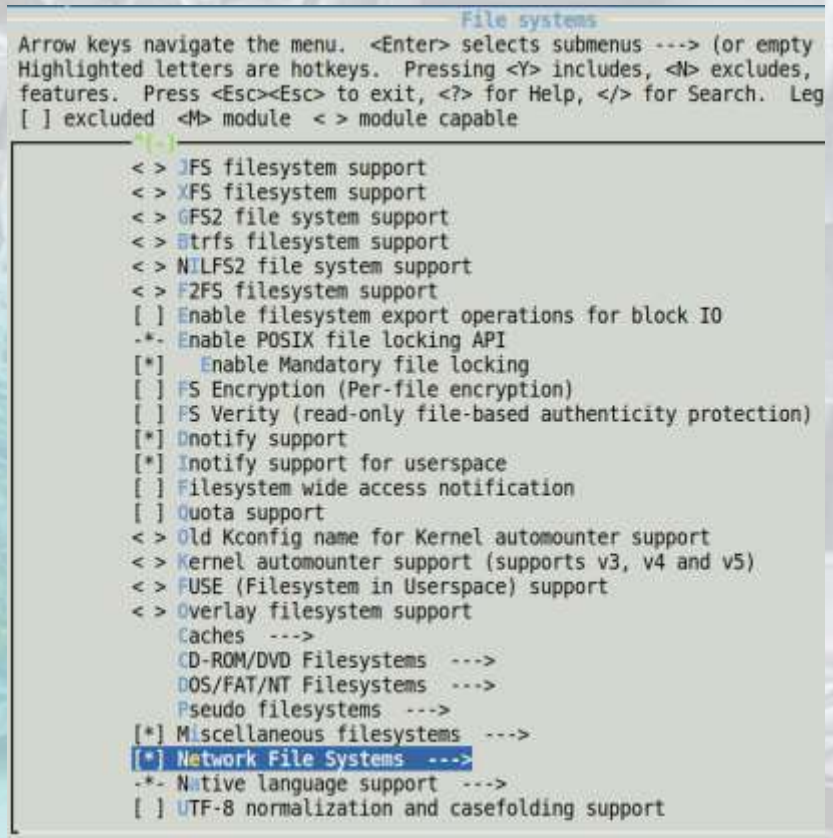


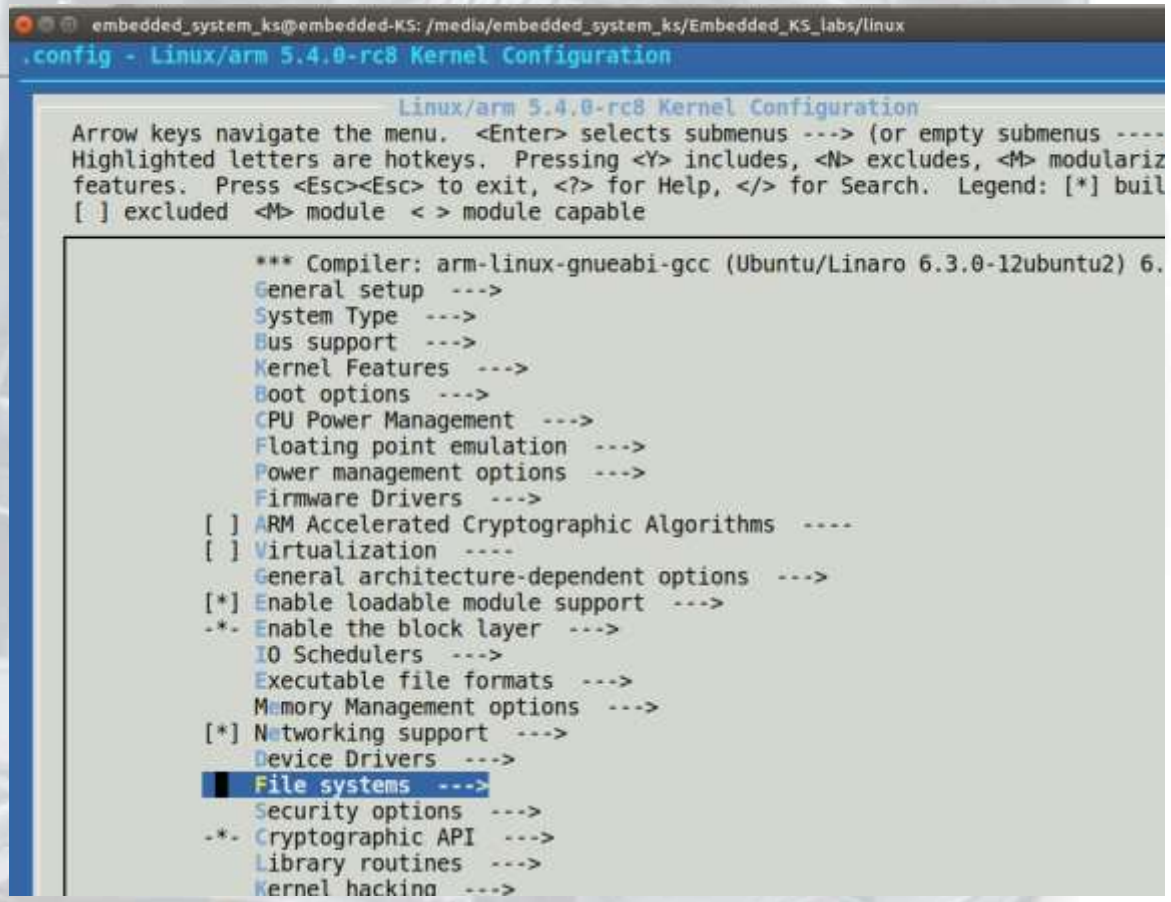


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

How to configure the Linux kernel/fs

- Select which extension you need.
- Pseudo filesystems
- RAMFS
- Miscellaneous filesystems





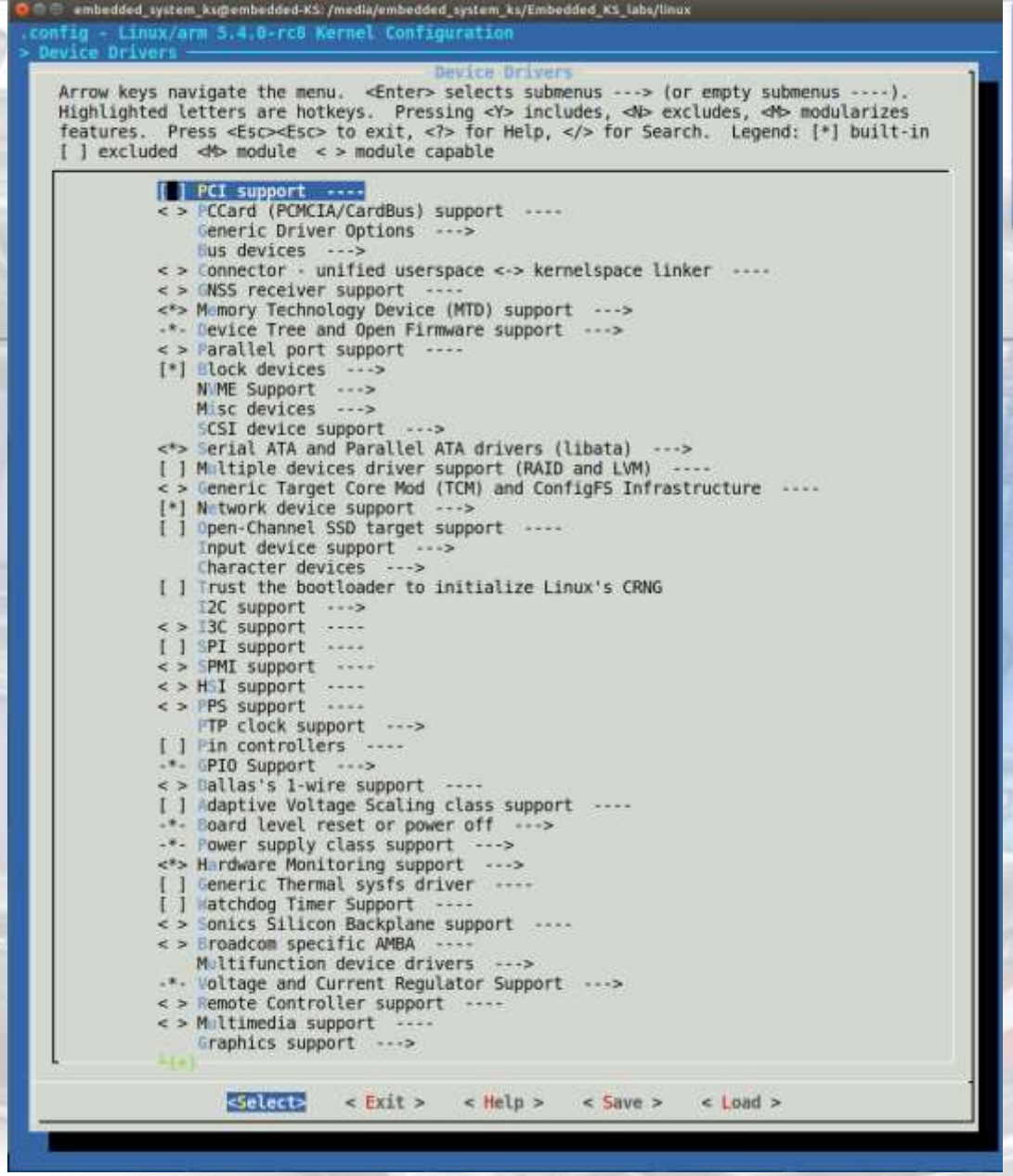
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Networking

```
                        Networking support
 keys navigate the menu.  <Enter> selects submenus ---> (or empt
ighted letters are hotkeys.  Pressing <Y> includes, <N> excludes
res.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  L
xcluded  <M> module  < > module capable

    --- Networking support
        Networking options  --->
    [ ] Amateur Radio support  ----
    < > CAN bus subsystem support  ----
    < > Bluetooth subsystem support  ----
    < > RxRPC session sockets
    < > KCM sockets
    [ ] Wireless  ----
    < > WiMAX Wireless Broadband support  ----
    < > RF switch subsystem support  ----
    <*> Plan 9 Resource Sharing Support (9P2000)  --->
    < > CAIF support  ----
    < > Ceph core library
    < > NFC subsystem support  ----
    < > Packet-sampling netlink channel  ----
    < > Inter-FE based on IETF ForCES InterFE LFB  ----
    [ ] Network light weight tunnels
    -*- Generic failover module
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Linux kernel/Device drivers



#LEARN_IN DEPTH

#Be_professional_in embedded_system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Linux kernel/Device drivers


```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_KS_labs/linux
.config - Linux/arm 5.4.0-rc8 Kernel Configuration
> Device Drivers > Search (CONFIG_DEVTMPFS_MOUNT)
                              Search Results
Symbol: DEVTMPFS_MOUNT [=n]
Type  : bool
Prompt: Automount devtmpfs at /dev, after the kernel mounted the rootfs
  Location:
    -> Device Drivers
      -> Generic Driver Options
(1)     -> Maintain a devtmpfs filesystem to mount at /dev (DEVTMPFS [=y])
  Defined at drivers/base/Kconfig:47
  Depends on: DEVTMPFS [=y]
```

Learn-in-depth.com

References

- Embedded Linux training
  - https://bootlin.com/training/
- Linux kernel and driver development training
- Yocto Project and OpenEmbedded development training
- Buildroot development training
- Building Embedded Linux Systems, 2nd Edition
- Mastering Embedded Linux Programming - Second Edition
- Christopher Hallinan, **Embedded Linux Primer**, Prentice Hall, 2006.
- **Silberschatz**, Galvin, and Gagne's **Operating System Concepts**, **Eighth Edition**.
- Robert love, linux kernel develoepemnt 3 rd edition 2010
- **Advanced Linux Programming By Mark Mitchell, Jeffrey Oldham, Alex Samuel**
- **Linux OS in Embedded Systems & Linux Kernel Internals(2/2)**
  - Memory Management, Paging, Virtual Memory, File system and its implementation, Secondary Storage(HDD), I/O systems

References Cont.

- Memory Management article
- Introduction to Operating Systems Class 9 - Memory Management
- Virtual Memory lecture for Introduction to Computer Architecture at Uppsala University.
- OS Lecture Note 13 - Address translation, Caches and TLBs
- CSE 331 Operating Systems Design lectures
- CSE 331 Virtual Memory
- CSE 332 Computer Organization and Architecture
- File Systems: Fundamentals.
- Linux System Programming
- System Calls, POSIX I/O
  CSE 333 Spring 2019, Justin Hsia

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

References Cont.

- **File Permissions in Linux/Unix with Example**
- Introduction to UNIX / Linux – 4
- **12.2 Basic I/O Concepts**
- Communicating with Hardware
- I/O Systems I/O Hardware Application I/O Interface
- COMPUTER ARCHITECTURE
- **OS**
- Linux Operating System
- W4118 Operating Systems, Instructor: Junfeng Yang
- CSNB334 Advanced Operating Systems 4. Process & Resource Management.
- Using the POSIX API
- **Linux Memory Management**
- Porting U-Boot and Linux on new ARM boards
- Chapter 2: Operating-System Structures
- **POSIX Threads Programming**
- U-Boot Environment Variables

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

References Cont.

- Using U-boot as production test strategy -- really?
- TFTP Boot using u-boot
- Loading the kernel with TFTP and U-boot
- https://blog.3mdeb.com/2013/2013-06-07-0x5-qemu-network-configuration-and-tftp-for-virtual-development-board/
- Installing and Testing TFTP Server in Ubuntu
- Pthreads: A shared memory programming model https://slideplayer.com/slide/8734550/
- Detailed Linux device tree
- Device Tree for DummIES
- Device Tree Usage
- Signal Handling in Linux Tushar B. Kute
- ARM case-study: the raspberry pi
- Introduction to Sockets Programming in C using TCP/IP
- The Broadcom chip used in the Raspberry Pi 2 Model B