FOLLOW US
Press here
1

#LEARN IN DEPTH

#Be professional in embedded system

eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

Embedded Linux (PART 6)

- CROSS-COMPILING TOOLCHAINS
- TOOLCHAIN COMPONENTS
- LABS

ENG.KEROLES SHENOUDA

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

Cross-compiling toolchains

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

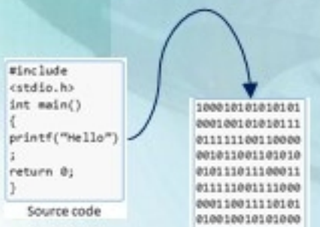
FOLLOW US
Press here
3
#LEARN IN DEPTH
#Be professional in embedded system

tool-chain

- The tool-chain consists of **a compiler**, **assembler**, **linker**, **debugger**, **libraries** and a number of helper programs.

For example
objcopy
objdump
readelf

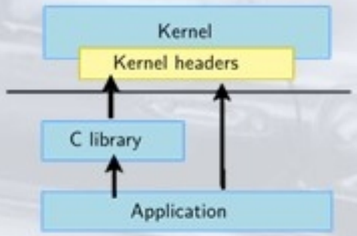
- By default the **host Pc (Ubuntu, redhat, etc)** contains some development tools which ate called **as native toolchain**
- **Toolchain Components:**



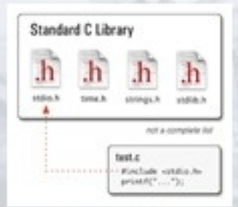
compiler

Binary Utilities

Debugger



Kernel Headers



C/C++ Libraries

Trace and profile (optional)

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

FOLLOW US
Press here
4
#LEARN IN DEPTH
#Be professional in embedded system

tool-chain Cont.

Instruction set:
Intel
OS: Linux/windows

C Project

Host machine
For Building process

Target machine
For run the executable

a.out

Instruction set:
AArch64
OS: Linux

C Project

Cross Compiling Toolchain

This toolchain runs run on **your workstation** but generate code for your **target**

Native Toolchain

This toolchain runs on **your workstation** and **generates code for your workstation,** usually x86

Instruction set:
Intel
OS: Linux/windows

C Project

Host machine
For Building process

Target machine
For run the executable

a.out

Instruction set:
Intel
OS: Linux/windows

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

FOLLOW US
Press here
5
#LEARN IN DEPTH
#Be professional in embedded system

tool-chain Cont.

- When you use these tool the code generation would be for native architecture (say x86)
- In case of developing embedded systems these toolchains does not **serve** the purpose
- The targets some times might have the following restrictions too!
  - Less memory footprints
  - Slow compared to host
  - You may not wish to have in a developed system finally
- So cross compiling toolchain are used

Instruction set:
AArch64
OS: Linux

C Project

Host machine For Building process

Target machine
For run the executable

a.out

Instruction set:
AArch64
OS: Linux

Not recommended as follow

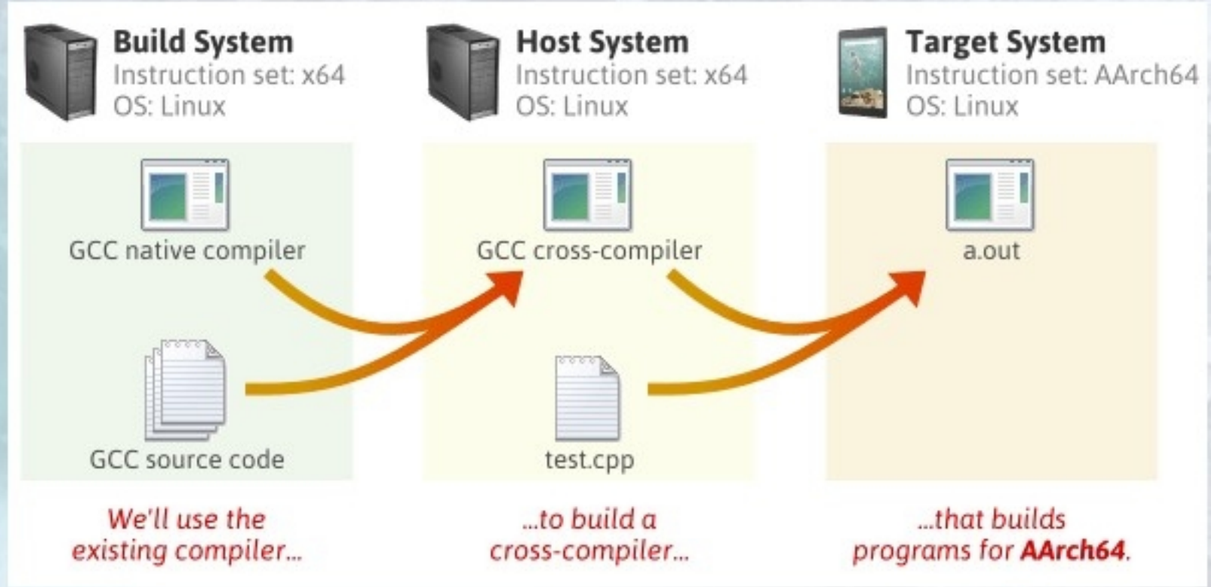
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Eng. Keroles Shenouda https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
6
#LEARN IN DEPTH
#Be professional in embedded system

Definition

- The **build machine**, where the toolchain is built.
- The **host machine**, where the toolchain will be executed.
- The **target machine**, where the binaries created by the toolchain are executed.

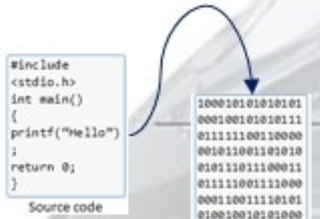


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
7

#LEARN IN DEPTH

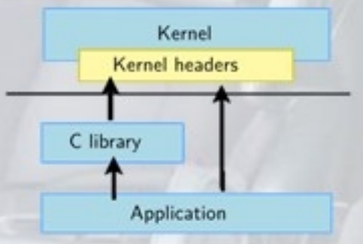
#Be professional in embedded system



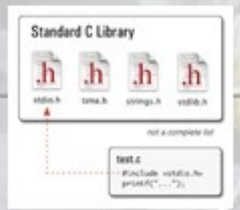
compiler

Binary Utilities

Debugger



Kernel Headers



C/C++ Libraries

Trace and profile (optional)

ToolChain Components

https://www.facebook.com/groups/embedded.system.KS/
Eng. Keroles Shenouda

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

FOLLOW US
Press here
8

#LEARN IN DEPTH

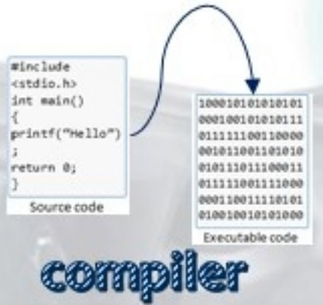
#Be professional in embedded system

ToolChain Components - gcc

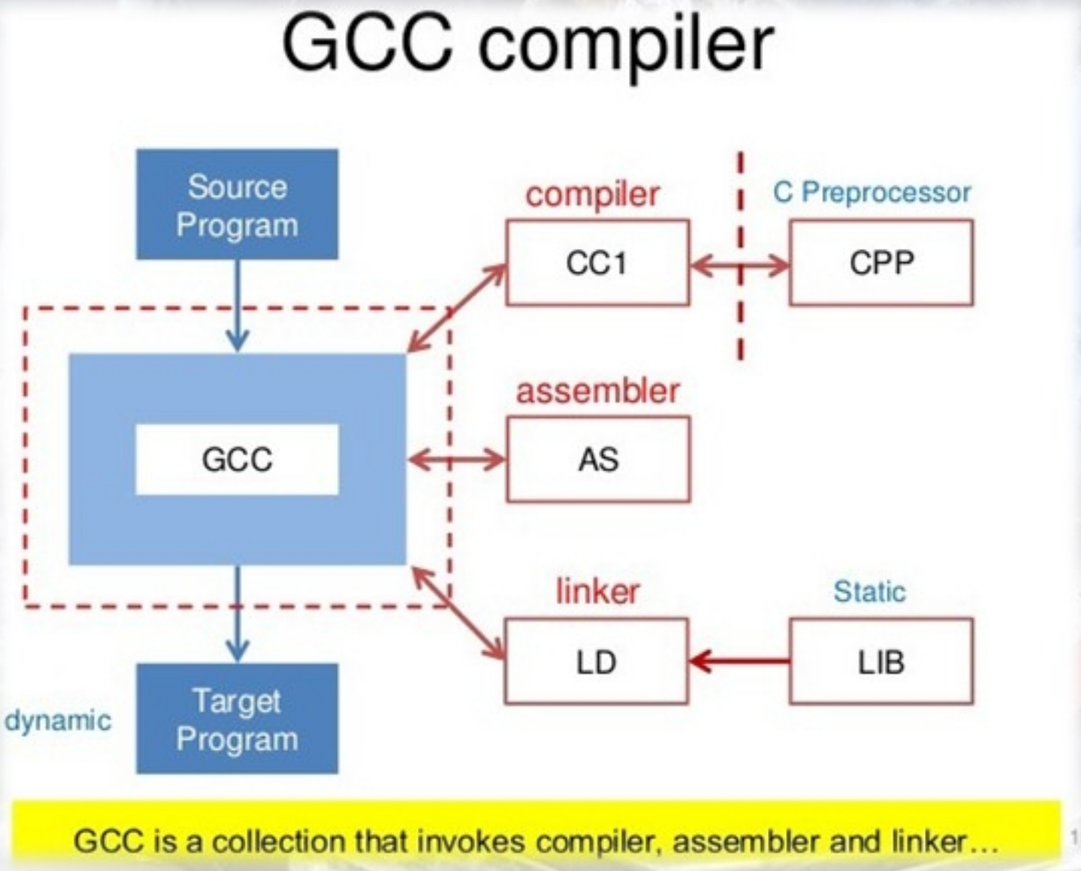
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
9
#LEARN IN DEPTH
#Be professional in embedded system

Components – GCC



- Can compile many programming languages and generate code for many different types of architecture.
- **GCC**: **G**NU **C**ompiler **C**ollection, the famous free software compiler
- http://gcc.gnu.org/



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

FOLLOW US
Press here
10

#LEARN IN DEPTH
#Be professional in embedded system

Components – GCC

- To perform complete compilation process of the main.c source
  - **$ gcc main.c**
    - This command will generate the a.out executable (assuming the program is composed of a single file)
- To change the name of the generated executable
  - **$ gcc main.c -o test**
    - This command will generate an executable with the name test
- To specify the Include path (path for header files)
  - **$ gcc main.c -I/usr/share/include -I. -I./inc/ -I../../inc**
- To enable all warning during compilation process
  - **$ gcc -Wall main.c -o test**
- To convert warnings into errors
  - **$ gcc -Wall -Werror main.c -o test**
- To pass options to gcc in a file (instead of in the command)
  - **$ gcc main.c @options-file**
    - Where options-file is a text file that will contain the required options

Eng.Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
11

#LEARN IN DEPTH
#Be professional in embedded system

Controlling Optimization Level

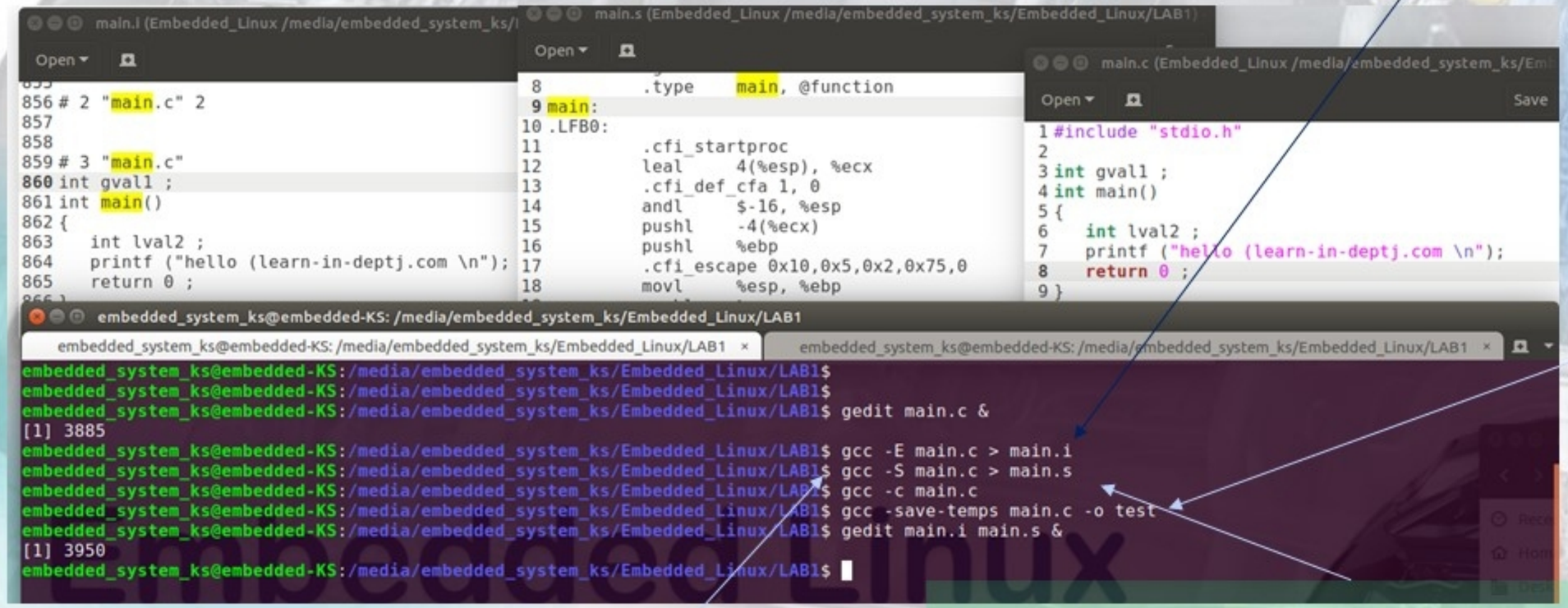
- The developer can set the compiler to optimize for one (or more) of these criteria on the expense of other criteria (for example better memory usage and processing time on the expense of compilation speed and debugging support)
- This is performed using the **-O** option
- There are multiple levels (those are some examples):
- Optimization Level Effect
  - **-O0** Default Level
  - **-O1, -O2, -O3** Optimize for processing speed
  - **-Os** Optimize for image size
  - **-Og** Optimize for debugging capabilities

Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
12
#LEARN IN DEPTH
#Be professional in embedded system

GCC Partial Compilation

This will only generate main.i file which is a pre-processed source file



This will save:

- File after pre-processing (main.i)
- File after compilation (main.s)
- File after assembly (main.o)
- Executable (test)

This will generate an assembly file main.s

This will generate a relocatable object file main.o

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

FOLLOW US
Press here
13

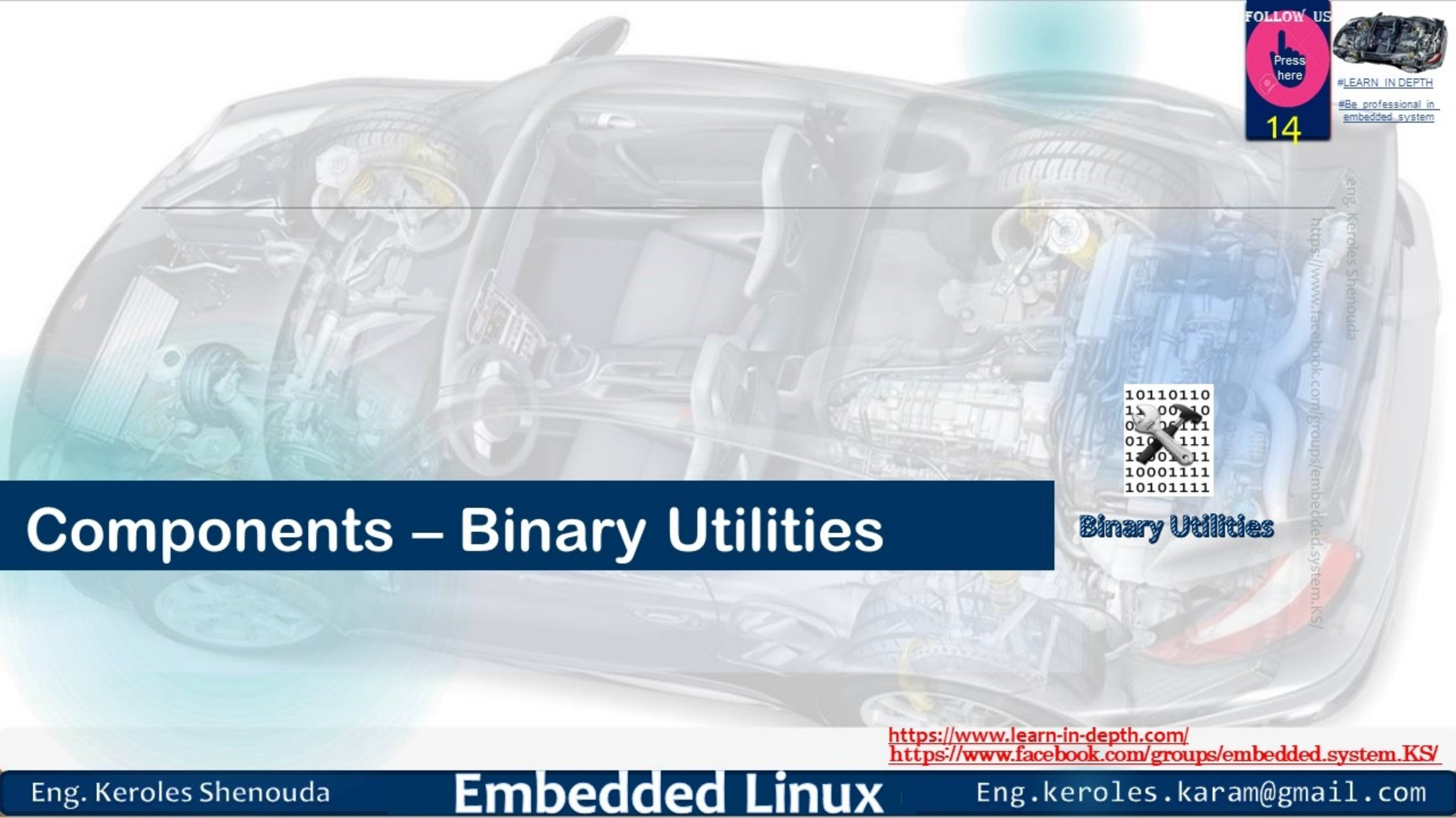
#LEARN IN DEPTH
#Be professional in embedded system

Object Files

- The Pre-processing/Compilation/Assembly steps **act on a single source file (independent of other files in the project)**
- The outcome is called ***a Relocatable Object File***
- The object file generated by the GNU toolchain follows the ELF file format
- Besides the binary code of the program, ELF files contain other info such **as a symbol table** which contain a list of symbols generated during the compilation process along with their addresses in the file
- Symbols in the Symbol table include (among other things):
  - Names of static and global variables **defined** in the file
  - Names of static and global variables **used** in the source file
  - Functions defined by the source file
  - Functions used (called) within the source file
- Symbols used in the file can be either,
  - **Defined** in the same file (**defined symbols**)
  - **Not be defined** in the file (**undefined symbols**)

Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
14
#LEARN IN DEPTH
#Be professional in embedded system



Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

Components – Binary Utilities

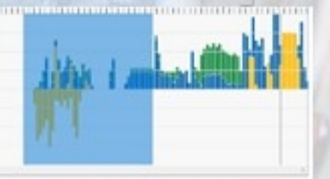
Binary Utilities

FOLLOW US
Press here
15
#LEARN IN DEPTH
#Be professional in embedded system

Components – Binary Utilities

- Set of **programming tools** for **creating** and **managing** binary programs, object files, libraries, profile data, and assembly source code
  - **as**, the assembler, that generates binary code from assembler source code
  - **ld**, the linker
  - **ar**, **ranlib**, to generate .a archives, used for libraries
  - **objdump, readelf, size, nm, strings**, to inspect binaries. Very useful analysis tools!
  - **objcopy**, to modify binaries
  - **strip**, to strip parts of binaries that are just needed for debugging (reducing their size).
- Often used with a compiler and libraries to design programs for Linux
- GNU Binary Utilities are called binutils

Binary Utilities



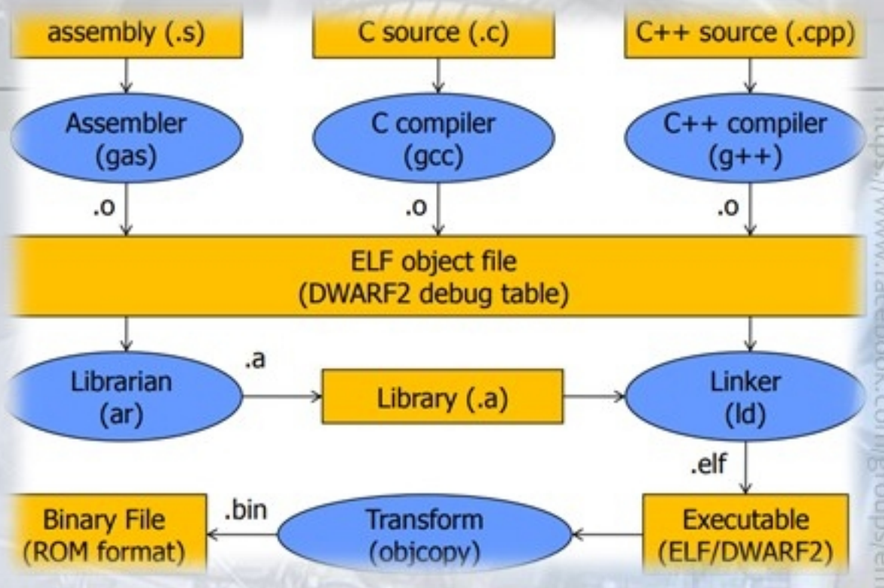
Trace and profile (optional)

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

FOLLOW US
Press here
16
#LEARN IN DEPTH
#Be professional in embedded system

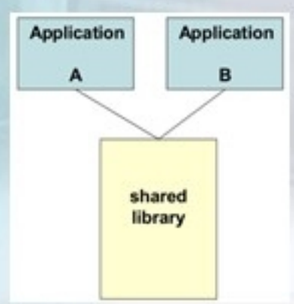
Linking Process

- $ ld [options] <object file(s)>
- Examples:
  - $ ld file1.o file2.o main.o
  - $ ld -o my-bin-file file1.o file2.o main.o
- Libraries



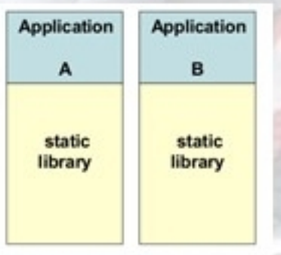
Dynamic Libraries (shared objects)

Linking occurs at run time
- Hence the executable image does not contain the required functionality
- The shared object should be available to the executable at run time



Static Libraries

A static library is a simple archive of pre-compiled object files
- Linking of a static library occurs at the same time of object files of the program
- The library becomes a part of the executable image



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

FOLLOW US
Press here
17

#LEARN IN DEPTH
#Be professional in embedded system

Creating a Static Library (ar Command)

- **$ ar [options] <library name> <object files>**
  - library object files should not contain **a main function**
  - Static library names should should start with "lib" and end with ".a" such as: libmath.a , libcontrol.a, libvision.a
- To create a static library file
  - $ ar rcs libmylib.a file1.o file2.o
- To add another object file to the library
  - $ ar r libmylib.a file3.o
- To remove an object file from the library
  - $ ar d libmylib.a file3.o
- To view the object files in the library
  - $ ar t libmylib.a
- To extract object files from the library
  - $ ar x libmylib.a

eng. Keroles Shenouda EMBEDDED_KS
https://www.facebook.com/groups/embedded.system.KS/

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

Creating a Static Library (ar Command) example

myPrint_API.c (Embedded_Linux /media/embedded_system_ks/

Open

```c
#include "myPrint_API.h"

  void learn_in_depth(char* buff)
{
        printf ("%s",buff);
}
```

myPrint_API.c

myPrint_API.h (Embedded_Linux /media/embedded_sys

Open

```c
extern void learn_in_depth(char* buff);
#include "stdio.h"
```

myPrint_API.h

main.c (Embedded_Linux /media/embedded_system_ks/Embedded_

Open

```c
#include "myPrint_API.h"

int gval1 ;
int main()
{
   int lval2 ;
   learn_in_depth ("hello (learn-in-deptj.com \n");
   return 0 ;
```

main.c

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB0$ gcc -c myPrint_API.c
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB0$ gcc -c main.c
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB0$ ls *.o
main.o  myPrint_API.o
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB0$ ar rcs lib_learn_in_depth.a myPrint_API.o
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB0$ gcc main.c -I . lib_learn_in_depth.a
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB0$ ./a.out
hello (learn-in-deptj.com
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB0$ ar t lib_learn_in_depth.a
myPrint_API.o
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB0$ ar x lib_learn_in_depth.a
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB0$ ls
lib_learn_in_depth.a  main.c  main.o  myPrint_API.c  myPrint_API.h  myPrint_API.o
```

To view the object files in the library

To extract object files from the library

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

FOLLOW US
Press here
19

#LEARN IN DEPTH
#Be professional in embedded system

Shared Object

- When we want our code to be compiled into a shared object to be used by other programs (instead of creating an independent executable)
- to create the shared object

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB0$ gcc -shared -o lib_learn_in_depth.so myPrint_API.o
```

- use gcc which calls ld internally -L <path of the lib> -l:<lib.so>

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB0$ gcc main.o -I . -L . -l:lib_learn_in_depth.so -o main.elf
```

- When the program loads in memory (at run time), the program binary is checked by the Dynamic Linker (ld-linux.so)
- We can see the executable is failing as Dynamic Linker can not find the lib_learn_in_depth.so

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB0$ ./main.elf
./main.elf: error while loading shared libraries: lib_learn_in_depth.so: cannot open shared object file: No such file or directory
```

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

FOLLOW US
Press here
20

#LEARN IN DEPTH

#Be professional in embedded system

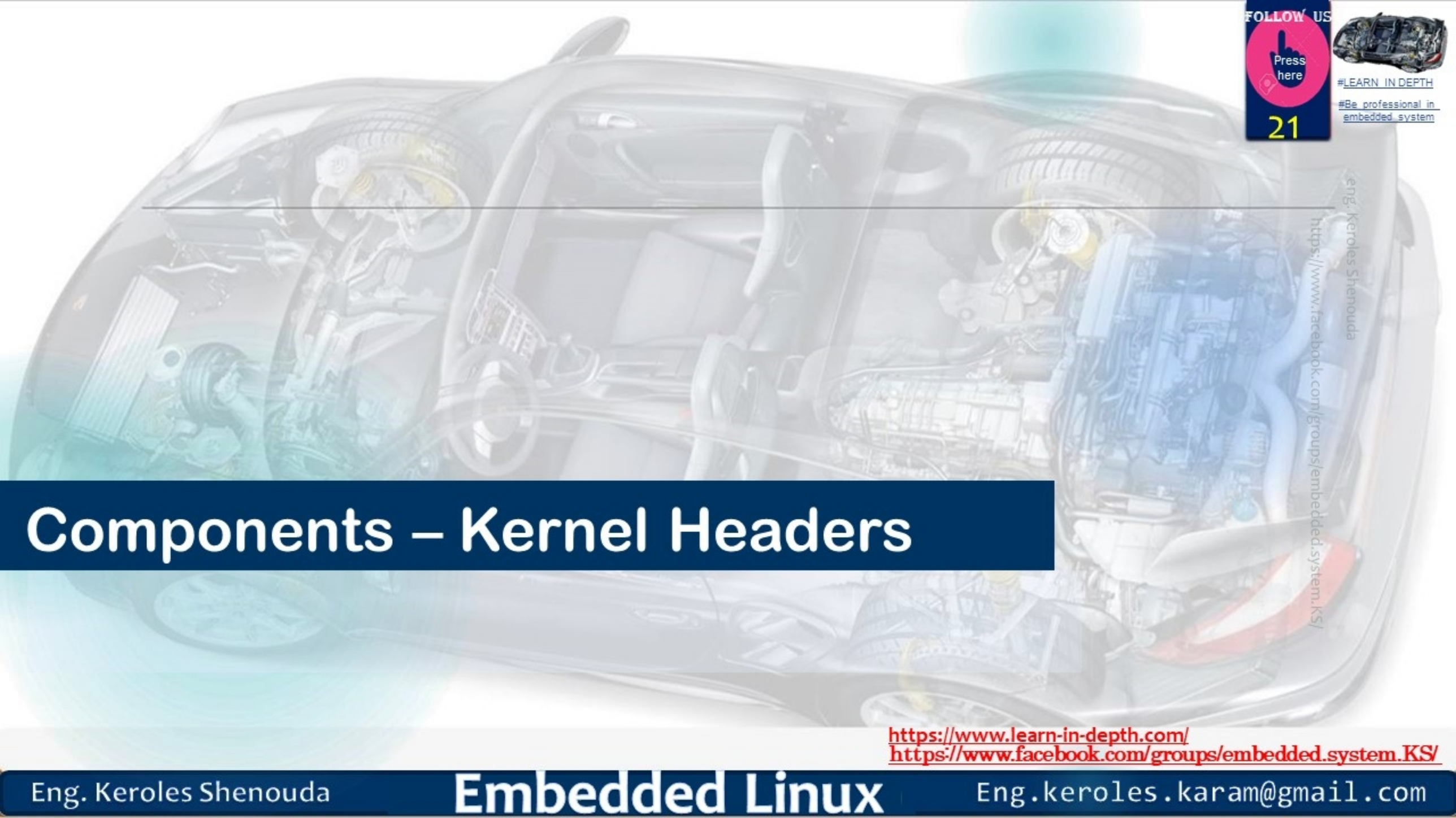
Shared Object Cont.

- Finding the Required Libraries (The ldd Command)
- Dynamic Linker Finds the Libraries,
  - by Library Path Environment Variable
    - It checks the environment variable **LD_LIBRARY_PATH**, which contain a list of directories (separated by a colon ":") that contain the libraries
  - Also by Dynamic Linker Cache
    - The dynamic linker cache is a binary file (/etc/ld.so.cache)
    - To add a directory to the dynamic linker cache,
      - Add the directory path to the file /etc/ld.so.conf
      - Run the utility ldconfig to generate the binary cache

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB0$ ldd main.elf
        linux-gate.so.1 =>  (0xb7768000)
        lib_learn_in_depth.so => not found
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb758d000)
        /lib/ld-linux.so.2 (0x80053000)
```

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB0$ export LD_LIBRARY_PATH=./
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB0$ ./main.elf
hello (learn-in-deptj.com
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB0$ ldd main.elf
        linux-gate.so.1 =>  (0xb7741000)
        lib_learn_in_depth.so => ./lib_learn_in_depth.so (0xb7739000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb7563000)
        /lib/ld-linux.so.2 (0x80078000)
```

arn-in-depth.com/
acebook.com/groups/embedded.system.KS/



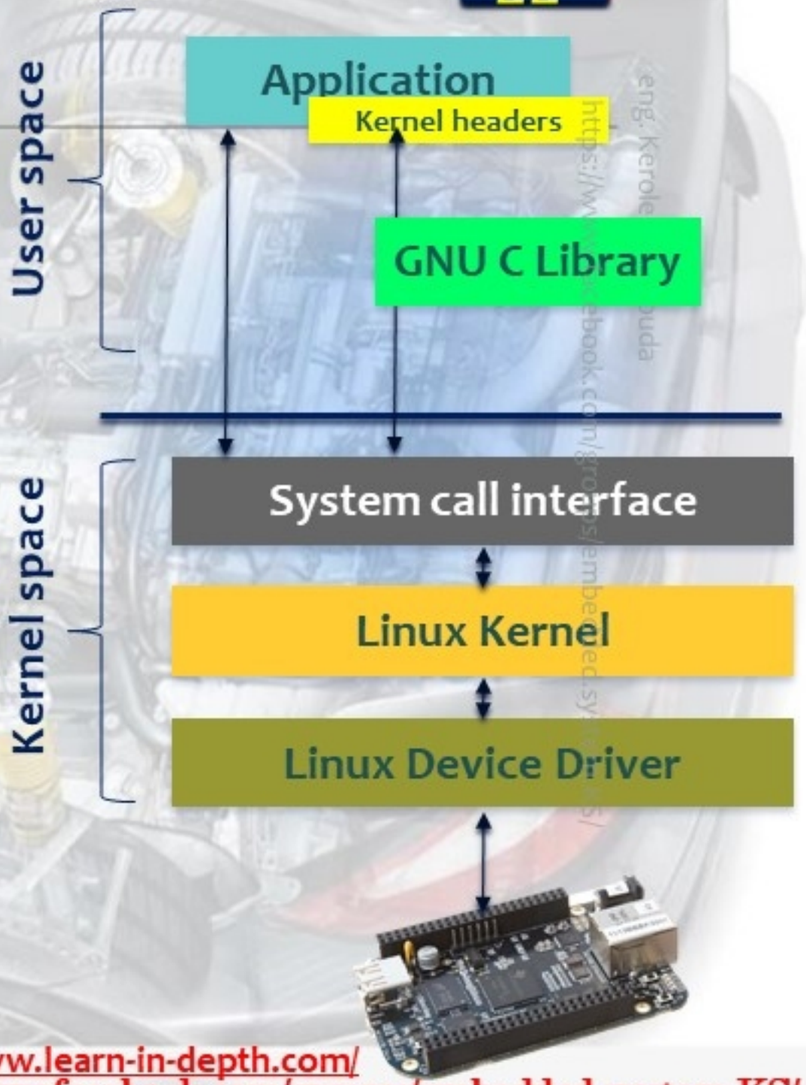
Components – Kernel Headers

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

FOLLOW US
Press here
22
#LEARN IN DEPTH
#Be professional in embedded system

Components – Kernel Headers

- The compiled applications and the libraries sometimes need to interact with the kernel
- So the Linux kernel need to expose the available interfaces for the libraries and applications to use, like
  - System calls and its numbers
  - MACRO definitions
  - Data structures, etc.
- You may find these in **<linux/…>** and **<asm/…>** and a few other directories corresponding to the ones visible in **include/** in the kernel sources
- **Compatibility with running kernel has to be considered**



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

FOLLOW US
Press here
23
#LEARN IN DEPTH
#Be professional in embedded system

Components – Kernel Headers Cont.

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/linux$ ls include/
acpi        clocksource  crypto  dt-bindings  Kbuild  kvm     math-emu  misc  pcmcia  rdma  soc     target  uapi  video
asm-generic  config      drm     generated    keys    linux   media     net   ras     scsi  sound   trace   vdso  xen
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/linux$ ls include/linux/
8250_pci.h                  dns_resolver.h          irqflags.h              of_iommu.h              sfi.h
acct.h                      dqblk_qtree.h           irq.h                   of_irq.h                sfp.h
acpi_dma.h                  dqblk_v1.h              irqhandler.h            of_mdio.h               sh_clk.h
acpi.h                      dqblk_v2.h              irqnr.h                 of_net.h                shdma-base.h
acpi_iort.h                 drbd_genl_api.h         irq_poll.h              of_pci.h                sh_dma.h
acpi_pmtmr.h                drbd_genl.h             irqreturn.h             of_pdt.h                sh_eth.h
adb.h                       drbd.h                  irq_sim.h               of_platform.h           sh_intc.h
adfs_fs.h                   drbd_limits.h           irq_work.h              of_reserved_mem.h       shmem_fs.h
adxl.h                      ds2782_battery.h        isa.h                   oid_registry.h          shm.h
aer.h                       dsa                     isapnp.h                olpc-ec.h               shrinker.h
agp_backend.h               dtlk.h                  iscsi_boot_sysfs.h      omap-dma.h              sh_timer.h
agpgart.h                   dw_apb_timer.h          iscsi_ibft.h            omapfb.h                signalfd.h
ahci_platform.h             dynamic_debug.h         isdn                    omap-gpmc.h             signal.h
ahci-remap.h                dynamic_queue_limits.h  isicom.h                omap-iommu.h            signal_types.h
aio.h                       earlycpio.h             iversion.h              omap-mailbox.h          siox.h
alarmtimer.h                ecryptfs.h              jbd2.h                  once.h                  siphash.h
alcor_pci.h                 edac.h                  jhash.h                 oom.h                   sirfsoc_dma.h
altera_jtaguart.h           edd.h                   jiffies.h               openvswitch.h           sizes.h
altera_uart.h               eeprom_93cx6.h          journal_head.h          oprofile.h              skb_array.h
amba                        eeprom_93xx46.h         joystick.h              osq_lock.h              skbuff.h
amd-iommu.h                 efi-bgrt.h              jump_label.h            overflow.h              skmsg.h
anon_inodes.h               efi.h                   jump_label_ratelimit.h  packing.h               slab_def.h
a.out.h                     efs_vh.h                jz4740-adc.h            padata.h                slab.h
apm_bios.h                  eisa.h                  jz4780-nemc.h           pageblock-flags.h       slimbus.h
apm-emulation.h             elevator.h              kallsyms.h              page_counter.h          slub_def.h
apple_bl.h                  elfcore-compat.h        kasan-checks.h          page_ext.h              sm501.h
apple-gmux.h                elfcore.h               kasan.h                 page-flags.h            sm501-regs.h
arch_topology.h             elf-fdpic.h             kbd_diacr.h             page-flags-layout.h     smc911x.h
armada-37xx-rwtm-mailbox.h  elf.h                   kbd_kern.h              page_idle.h             smc91x.h
arm-cci.h                   elfnote.h               kbuild.h                page-isolation.h        smpboot.h
arm_sdei.h                  elf-randomize.h         kconfig.h               pagemap.h               smp.h
arm-smccc.h                 enclosure.h             kcore.h                 page_owner.h            smsc911x.h
ascii85.h                   energy_model.h          kcov.h                  page_ref.h              smscphy.h
asn1_ber_bytecode.h         err.h                   kdb.h                   pagevec.h               soc
asn1_decoder.h              errno.h                 kdebug.h                pagewalk.h              sock_diag.h
asn1.h                      error-injection.h       kdev_t.h                parman.h                socket.h
assoc_array.h               errqueue.h              kd.h                    parport.h               sonet.h
assoc_array_priv.h          errseq.h                kernelcapi.h            parport_pc.h            sony-laptop.h
async.h                     etherdevice.h           kernel.h                parser.h                sonypi.h
async_tx.h                  ethtool.h               kernel-page-flags.h     pata_arasan_cf_data.h   sort.h
ata.h                       eventfd.h               kernel_stat.h           patchkey.h              soundcard.h
atalk.h                     eventpoll.h             kernfs.h                path.h                  sound.h
ata_platform.h              evm.h                   kern_levels.h           pch_dma.h               soundwire
ath9k_platform.h            exportfs.h              kexec.h                 pci-acpi.h              spi
atmdev.h                    export.h                keyboard.h              pci-ats.h               spinlock_api_smp.h
atmel-mci.h                 ext2_fs.h               keyctl.h                pci-dma-compat.h        spinlock_api_up.h
atmel_pdc.h                 extable.h               key.h                   pci-ecam.h              spinlock.h
atmel-ssc.h                 extcon                  key-type.h              pci-ep-cfs.h            spinlock_types.h
atm.h                       extcon.h                kfifo.h                 pci-epc.h               spinlock_types_up.h
atm_suni.h                  extcon-provider.h       kgdb.h                  pci-epf.h               spinlock_up.h
atm_tcp.h                   f2fs_fs.h               khugepaged.h            pci.h                   splice.h
atomic-fallback.h           f75375s.h               klist.h                 pci_hotplug.h           spmi.h
atomic.h                    falloc.h                kmemleak.h              pci_ids.h               sram.h
attribute_container.h       fanotify.h              kmod.h                  pci-p2pdma.h            srcu.h
audit.h                     fault-inject.h          kmsg_dump.h             pda_power.h             srcutiny.h
auto_dev-ioctl.h            fbcon.h                 kobject.h               pe.h                    srcutree.h
auto_fs.h                   fb.h                    kobject_ns.h            percpu_counter.h        ssb
```

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/linux$ find . -name "asm"
./arch/sparc/include/asm
./arch/sparc/include/uapi/asm
./arch/h8300/include/asm
./arch/h8300/include/uapi/asm
./arch/unicore32/include/asm
./arch/unicore32/include/uapi/asm
./arch/arm/include/asm
./arch/arm/include/generated/asm
./arch/arm/include/generated/uapi/asm
./arch/arm/include/uapi/asm
./arch/mips/include/asm
./arch/mips/include/uapi/asm
./arch/nios2/include/asm
./arch/nios2/include/uapi/asm
./arch/microblaze/include/asm
./arch/microblaze/include/uapi/asm
./arch/openrisc/include/asm
./arch/openrisc/include/uapi/asm
./arch/sh/include/asm
./arch/sh/include/uapi/asm
./arch/c6x/include/asm
./arch/c6x/include/uapi/asm
./arch/s390/include/asm
./arch/s390/include/uapi/asm
./arch/ia64/include/asm
./arch/ia64/include/uapi/asm
./arch/um/include/asm
./arch/powerpc/include/asm
./arch/powerpc/include/uapi/asm
./arch/m68k/include/asm
./arch/m68k/include/uapi/asm
./arch/hexagon/include/asm
./arch/hexagon/include/uapi/asm
./arch/xtensa/include/asm
./arch/xtensa/include/uapi/asm
./arch/x86/include/asm
./arch/x86/include/uapi/asm
./arch/x86/um/asm
./arch/nds32/include/asm
./arch/nds32/include/uapi/asm
./arch/arm64/include/asm
./arch/arm64/include/uapi/asm
./arch/riscv/include/asm
./arch/riscv/include/uapi/asm
./arch/csky/include/asm
./arch/csky/include/uapi/asm
./arch/arc/include/asm
./arch/arc/include/uapi/asm
./arch/alpha/include/asm
./arch/alpha/include/uapi/asm
./arch/parisc/include/asm
./arch/parisc/include/uapi/asm
./include/config/cc/has/asm
^C
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/linux$ ls ./arch/sparc/include/asm/
```

adi_64.h	barrier.h	checksum.h	delay_32.h	floppy_32.h	hvtramp.h	irqflags_32.h	machines
adi.h	bbc.h	chmctrl.h	delay_64.h	floppy_64.h	hw_irq.h	irqflags_64.h	mbus.h
agp.h	bitext.h	clock.h	delay.h	floppy.h	hypervisor.h	irqflags.h	mc146818
apb.h	bitops_32.h	clocksource.h	device.h	fpumacro.h	ide.h	irq.h	mc146818
asm.h	bitops_64.h	cmpxchg_32.h	dma.h	ftrace.h	idprom.h	jump_label.h	mc146818
asmmacro.h	bitops.h	cmpxchg_64.h	dma-mapping.h	futex_32.h	intr_queue.h	Kbuild	mdesc.h
asm-offsets.h	btext.h	cmpxchg.h	ebus_dma.h	futex_64.h	io_32.h	kdebug_32.h	memctrl.
asm-prototypes.h	bug.h	compat.h	ecc.h	futex.h	io_64.h	kdebug_64.h	mman.h
atomic_32.h	bugs.h	compat_signal.h	eeprom.h	hardirq_32.h	ioctls.h	kdebug.h	mmu_32.h
atomic_64.h	cacheflush_32.h	contregs.h	elf_32.h	hardirq_64.h	io.h	kgdb.h	mmu_64.h

Embedded Linux
ENG.KEROLES SHENOUDA
Eng. Keroles Shenouda

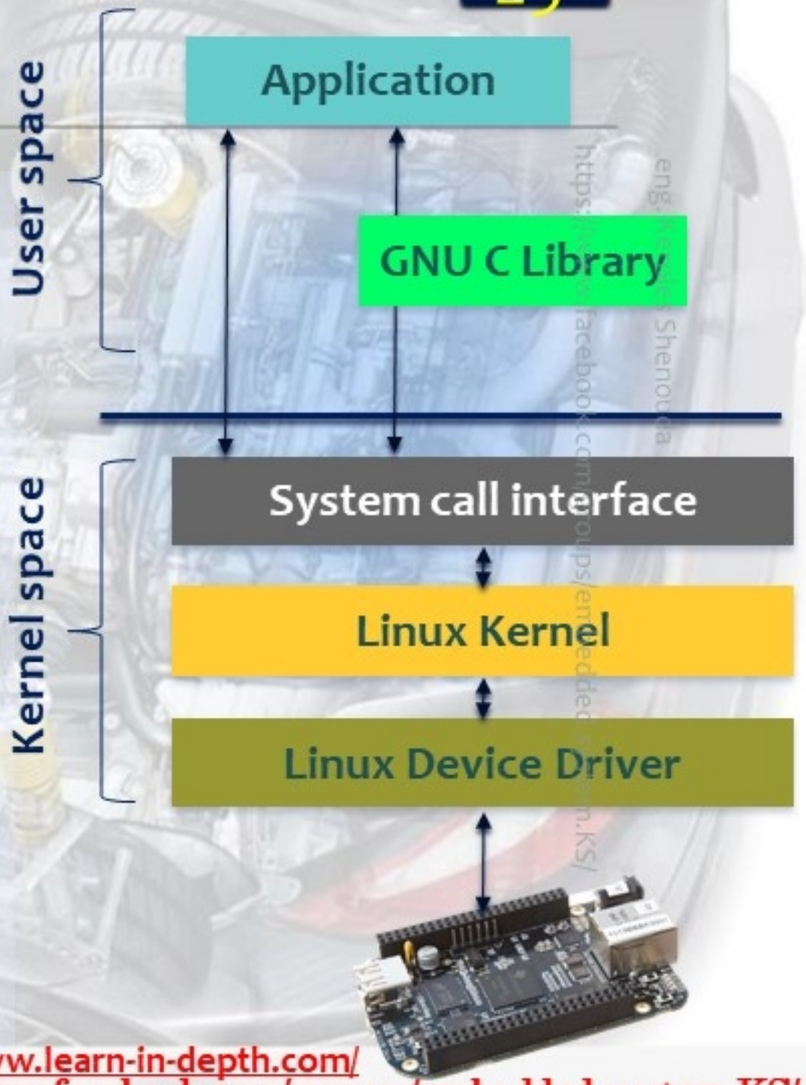
Components- Libraries

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

FOLLOW US
Press here
25
#LEARN IN DEPTH
#Be professional in embedded system

Components- Libraries

- **The C library** is an essential component of a Linux system
  - which provides **interface** between the **applications** and **the kernel**
  - Provides the well-known standard C API to ease application development
    - Provides **macros**, **type definitions**, and **functions** for tasks like *string handling, mathematical computations, input/output processing, memory allocation and several* other operating system services
- Several C libraries are available: glibc, uClibc, musl C, klibc, etc
- The choice of the C library must be made at cross-compiling toolchain generation time, as the GCC compiler is compiled against a specific C library.



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

FOLLOW US
Press here
26
#LEARN IN DEPTH
#Be professional in embedded system

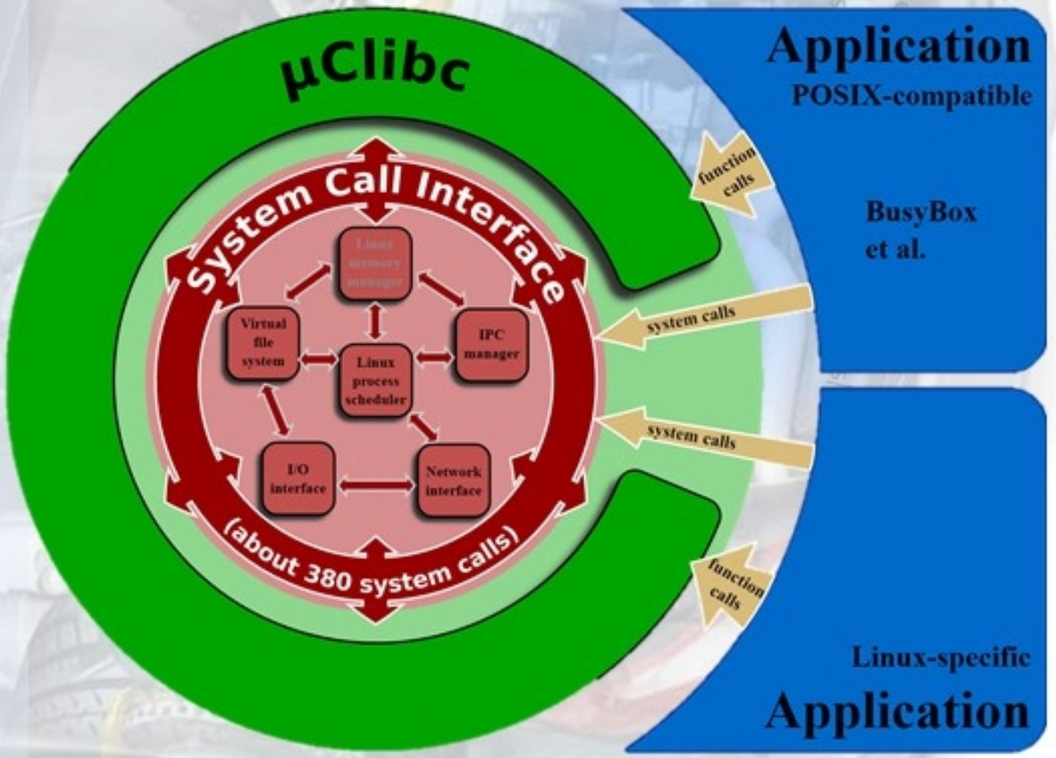
Components- Libraries- glibc

- C library from the GNU project
  - These APIs open, read, write, malloc, printf, pthread_create and more
- Good performance optimization, standards compliance and portability
- Well maintained and used on all GNU / Linux host systems
- Require large memory footprint making **it not suitable** for **small embedded systems**
- License: LGPL
- Website: http://www.gnu.org/software/libc/

Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

Components- Libraries- uClibc

- **Smaller than** the **GNU C Library**, but nearly all applications supported by glibc also work perfectly with uClibc
- **Recommended** for lower footprint embedded systems
- Works on most embedded architectures with embedded Linux
- Focus on size rather than performance with less compile time
- Supports most embedded architectures, including **MMU-less ones** (ARM Cortex-M, etc.).
  - **The only library supporting ARM noMMU.**
- http://uclibc-ng.org/
- Used on a large number of production embedded products, including consumer electronic devices
- Actively supported, but **Yocto** Project **stopped supporting it**



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

FOLLOW US
Press here
28

#LEARN IN DEPTH
#Be professional in embedded system

Components- musl C library

- A lightweight, fast and simple library for embedded systems
- Created while uClibc's development was stalled
- In particular, great at making small static executables
- Supported by build systems such as Buildroot and Yocto Project.

Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
29

#LEARN IN DEPTH
#Be professional in embedded system

C Library and Toolchain Comparison Table

Category	License	Features	Target Application	compile and strip BusyBox1.26.2 statically and compare the size	Maintainer User
glibc	LGPL 2.1	• StableABI • Backward compatibility • Fully symbol versioning • Stack smashing protection/ heap corruption detection • Profiling	• Performance • Security	With gcc 6.2, armel, glibc: 755088 bytes 737.39KB	GNU
uClibc-ng	LGPL 2.1	• No-MMU architecture support • Tiny size	• Resource Limited	With gcc 6.3, armel, uclibc-ng 1.0.22 : 210620 bytes 205.68KB	uclibc-ng.org
Musl	MIT	• StableABI • Backward compatibility • Stack smashing protection/ heap corruption detection	• Resource Limited • Security	With gcc 6.3, armel, musl 1.1.16: 183348 bytes 179.05 KB	musl-libc.org

Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

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

FOLLOW US
Press here
30

#LEARN IN DEPTH
#Be professional in embedded system

Obtaining a Toolchain

- Building a cross-compiling toolchain by yourself is a difficult and painful task!
  - Toolchain building utilities
- Get a pre-compiled toolchain
  - Advantage: it is the simplest and most convenient solution
  - Drawback: you can't fine tune the toolchain to your needs
- Check whether the available toolchains match your requirements.
- Possible choices
  - Toolchains packaged by your distribution
    - Ubuntu example: sudo apt install gcc-arm-linux-gnueabihf
  - Linaro, Sourcery CodeBench toolchains, now only supporting MIPS, NIOSII, AMD64, Hexagon. Old versions with ARM support still available through build systems (Buildroot…) and etc …
  - Toolchain provided by your hardware vendor.
  - Use an embedded build system (Yocto Project, OpenEmbedded, Buildroot) to generate one

Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
31

#LEARN IN DEPTH
#Be professional in embedded system

Obtaining a Toolchain Cont.

- There are different releases for the toolchain
- Linaro toolchain
  - https://launchpad.net/linaro-toolchain-binaries
- GNU Arm Embedded Toolchain
  - https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm
- Toolchain for ARM Cortex M and R cores
  - https://launchpad.net/gcc-arm-embedded
- CodeWarrior (By Freescale)
  - http://www.freescale.com/webapp/sps/site/homepage.jsp?code=CW_HOME

Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
32
#LEARN IN DEPTH
#Be professional in embedded system

We will use prebuilt toolchain

- sudo apt-get install gcc-arm-none-eabi

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux$ arm
arm2hpdl                      arm-linux-gnueabi-gcc-nm        arm-linux-gnueabi-ld.gold       arm-none-eabi-c++filt       arm-none-eabi-ld.bfd
arm-linux-gnueabi-addr2line   arm-linux-gnueabi-gcc-nm-6      arm-linux-gnueabi-nm            arm-none-eabi-cpp           arm-none-eabi-nm
arm-linux-gnueabi-ar          arm-linux-gnueabi-gcc-ranlib    arm-linux-gnueabi-objcopy       arm-none-eabi-elfedit       arm-none-eabi-objcopy
arm-linux-gnueabi-as          arm-linux-gnueabi-gcc-ranlib-6  arm-linux-gnueabi-objdump       arm-none-eabi-g++           arm-none-eabi-objdump
arm-linux-gnueabi-c++filt     arm-linux-gnueabi-gcov          arm-linux-gnueabi-ranlib        arm-none-eabi-gcc           arm-none-eabi-ranlib
arm-linux-gnueabi-cpp         arm-linux-gnueabi-gcov-6        arm-linux-gnueabi-readelf       arm-none-eabi-gcc-5.4.1     arm-none-eabi-readelf
arm-linux-gnueabi-cpp-6       arm-linux-gnueabi-gcov-dump     arm-linux-gnueabi-size          arm-none-eabi-gcc-ar        arm-none-eabi-size
arm-linux-gnueabi-dwp         arm-linux-gnueabi-gcov-dump-6   arm-linux-gnueabi-strings       arm-none-eabi-gcc-nm        arm-none-eabi-strings
arm-linux-gnueabi-elfedit     arm-linux-gnueabi-gcov-tool     arm-linux-gnueabi-strip         arm-none-eabi-gcc-ranlib    arm-none-eabi-strip
arm-linux-gnueabi-gcc         arm-linux-gnueabi-gcov-tool-6   arm-none-eabi-addr2line         arm-none-eabi-gcov
arm-linux-gnueabi-gcc-6       arm-linux-gnueabi-gprof         arm-none-eabi-ar                arm-none-eabi-gcov-tool
arm-linux-gnueabi-gcc-ar      arm-linux-gnueabi-ld            arm-none-eabi-as                arm-none-eabi-gprof
arm-linux-gnueabi-gcc-ar-6    arm-linux-gnueabi-ld.bfd        arm-none-eabi-c++               arm-none-eabi-ld
```

Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

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

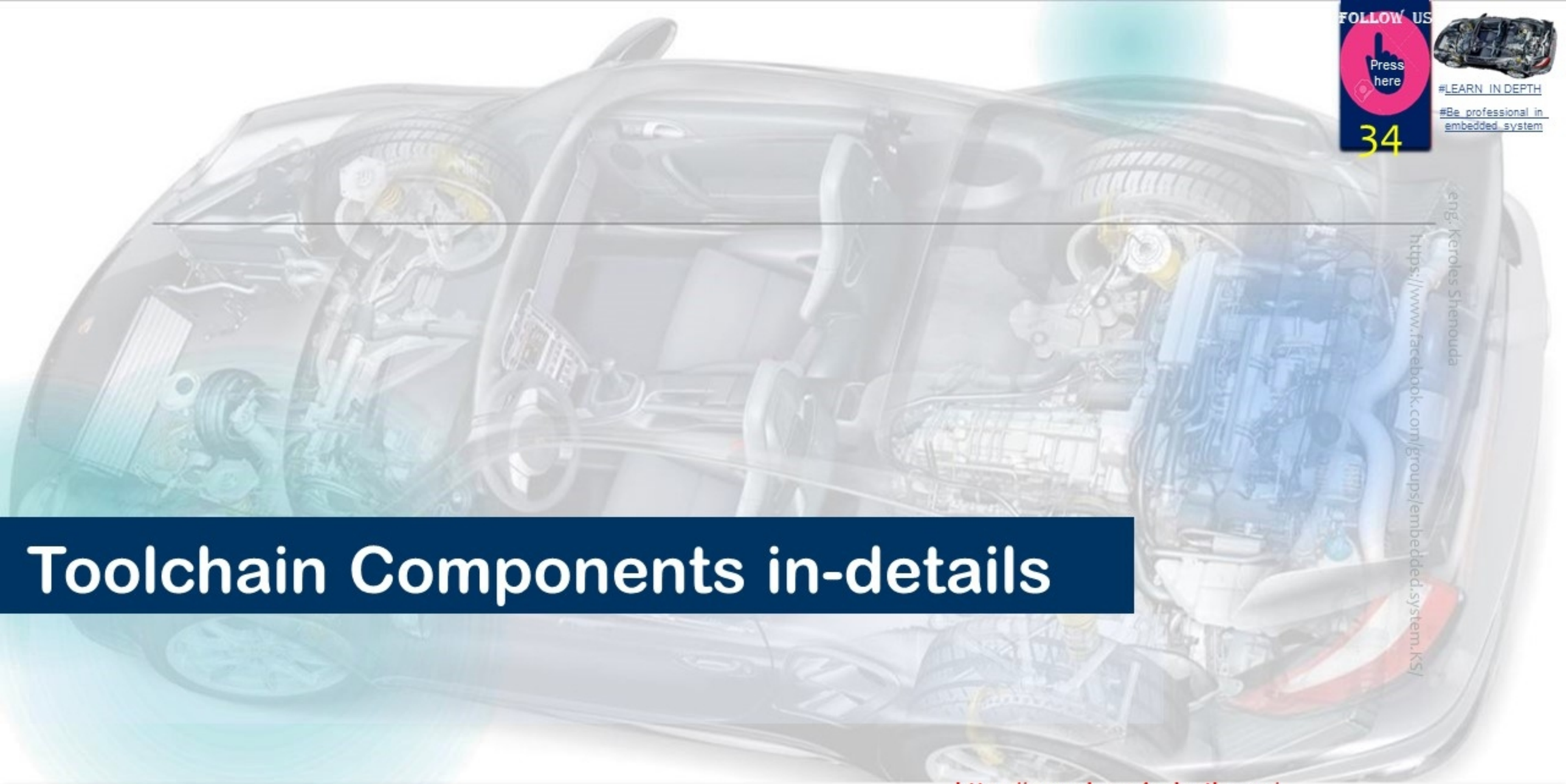
FOLLOW US
Press here

#LEARN IN DEPTH
#Be professional in embedded system

CPU optimization flags

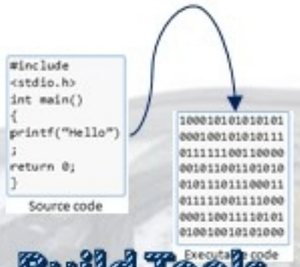
- A set of cross-compiling tools is specific to a CPU architecture (ARM, x86, MIPS, PowerPC).
- However, gcc offers further options:
  - -march allows to select a specific target instruction set
  - -mtune allows to optimize code for a specific CPU
  - For example: ***-march=armv7 -mtune=cortex-a8***
  - ***-mcpu=cortex-a8*** can be used instead to allow gcc to infer the target instruction set (-march=armv7) and cpu optimizations (-mtune=cortex-a8)

Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/



Toolchain Components in-details

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



Build Tools

Binary Utilities

Debugger

gcc	Compiling C/C++ (and other languages) code into object files Also it **calls a linker internally** to link object files into an executable
ld	**Links** object files into an executable (called internally by gcc)
make	Reads the **Makefile** to manage the **building process**
ar	**Archives** multiple Object files into **a static Library**

readelf	Reads the **contents** of an ELF File (**object file or executable**)
objdump	Reads the internals of an ELF file **including assembly code**
nm	Reads the **symbols** inside an object file or and executable
strings	Reads **text strings** inside a binary file
strip	**Strips** the **binary file** from **some optional sections**
addr2line	**Converts** an address in the binary to **a source file name and line number**
size	Display **the ELF file section sizes** and **total size**

gdb	Debugger

FOLLOW US
Press here
35
#LEARN IN DEPTH
#Be professional in embedded system
Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

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

Lab1: write a Bermatel Application on VersatilePB

We will go through the steps in depth ☺

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

FOLLOW US
Press here
37

#LEARN IN DEPTH
#Be professional in embedded system

ARM system emulated with QEMU

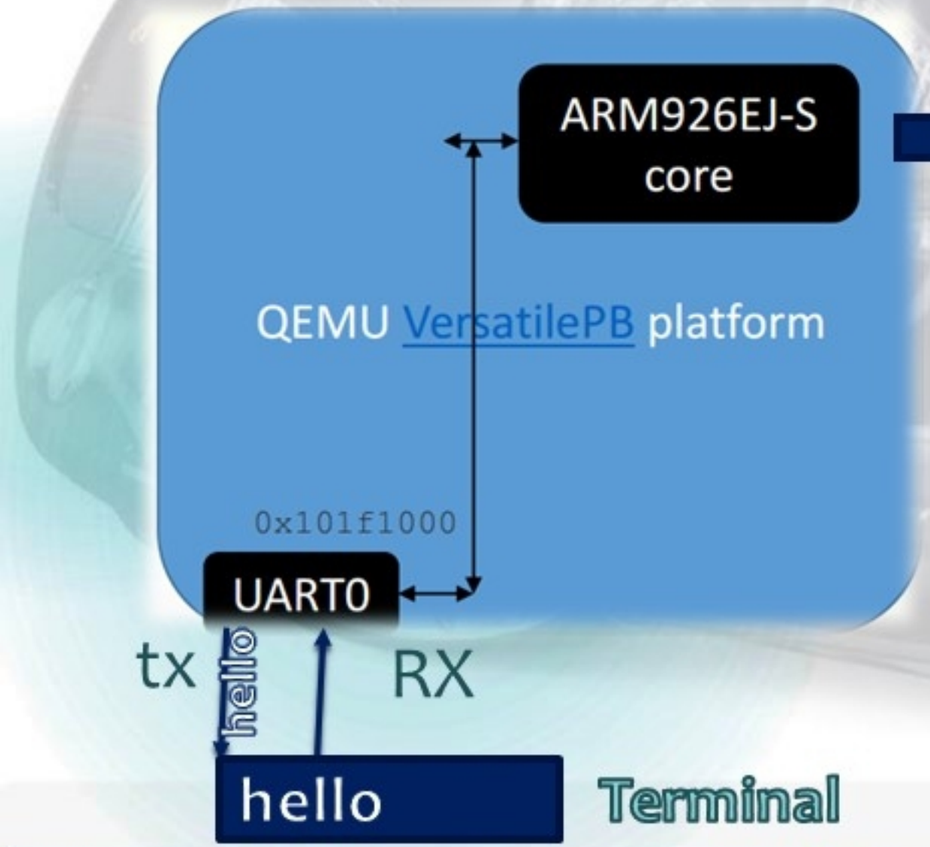
- qemu-system-arm is the software that emulates a VersatilePB platform
  For more information "VersatilePB physical Board'
  - sudo apt-get install qemu-system-arm
- http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dsi0034a/index.html
- http://www.arm.com/products/tools/development-boards/versatile-express
- http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0224i/index.html

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

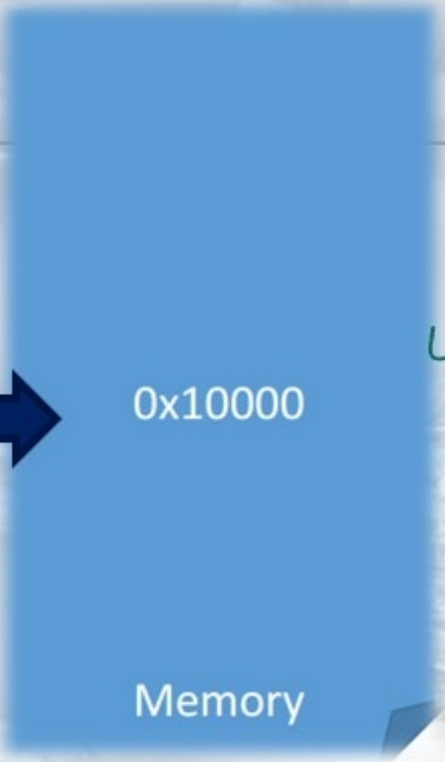
HELLO WORLD FOR BARE METAL

#LEARN IN DEPTH

#Be professional in embedded system



Entry point



You will need

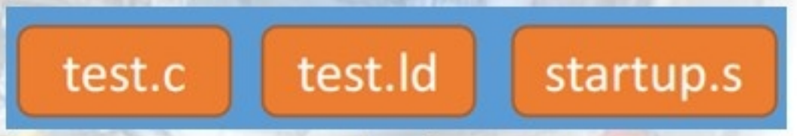
Startup.s

Linker Script.ld

C Code files

Cross Toolchai

Makefile



Burn it in the Memory

Exectuable File
Test.bin

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

Lab: Steps

- The QEMU emulator supports the VersatilePB platform, that contains an **ARM926EJ-S** core and, among
- other peripherals, **four UART serial ports**;
- the first serial port in particular **(UART0) works as a terminal**
- when using the -nographic or "-serial stdio" qemu option. The memory map of the VersatilePB board is implemented in QEMU in this board-specific C source;
- note the address where the UART0 is mapped: **0x101f1000**.

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_Linux/LAB1
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ touch test.c test.ld startup.s
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ gedit test.c test.ld startup.s &
[1] 7139
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$
```

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

FOLLOW US
Press here
#LEARN IN DEPTH

there is a register **(UARTDR)** that is used to transmit (when writing in the register) and receive (when reading) bytes; **this register is placed at offset `0x0`**, so you need to read and write at the beginning of the memory allocated for the UARTo

3.3 Register descriptions

This section describes the UART registers. The test registers are described in Chapter 4 *Programmer's Model for Test*. Table 3-1 on page 3-3 provides cross references to individual registers.

3.3.1 Data register, UARTDR

The UARTDR register is the data register.

For words to be transmitted:

- if the FIFOs are enabled, data written to this location is pushed onto the transmit FIFO
- if the FIFOs are not enabled, data is stored in the transmitter holding register (the bottom word of the transmit FIFO).

The write operation initiates transmission from the UART. The data is prefixed with a start bit, appended with the appropriate parity bit (if parity is enabled), and a stop bit. The resultant word is then transmitted.

For received words:

- if the FIFOs are enabled, the data byte and the 4-bit status (break, frame, parity, and overrun) is pushed onto the 12-bit wide receive FIFO
- if the FIFOs are not enabled, the data byte and status are stored in the receiving holding register (the bottom word of the receive FIFO).

The received data byte is read by performing reads from the UARTDR register along with the corresponding status information. The status information can also be read by a read of the UARTRSR/UARTECR register as shown in Table 3-2 on page 3-6.

Lab: Steps

- The QEMU emulator supports the VersatilePB platform, that contains an **ARM926EJ-S** core and, among
- other peripherals, **four UART serial ports**;
- the first serial port in particular **(UART0) works as a terminal**
- when using the -nographic or "-serial stdio" qemu option. The memory map of the VersatilePB board is implemented in QEMU in this board-specific C source;
- note the address where the UART0 is mapped: **0x101f1000**.

FOLLOW US
Press here
42

#LEARN IN DEPTH
#Be professional in embedded system

To implement the simple “Hello world!” printing, you should write this test.c file:

```
volatile unsigned int * const UART0DR = (unsigned int *)0x101f1000;
void print_uart0(const char *s) {
        while(*s != '\0') { /* Loop until end of string */
        *UART0DR = (unsigned int)(*s); /* Transmit char */
        s++; /* Next char */
        }
}
void c_entry() {
print_uart0("Hello world!\n");
}
```

- The **volatile** keyword is necessary to instruct the compiler that the memory pointed by UART0DR can change or has effects independently of the program.
- The **unsigned int** type enforces 32-bits read and write access.

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

startup.s

```
.global _Reset
_Reset:
    LDR sp, =stack_top
    BL c_entry
    B .
```

Jump to c_entry()

the linker script test.ld

```
ENTRY(_Reset)
SECTIONS
{
. = 0x10000;
.startup . : { startup.o(.text) }
.text : { *(.text) }
.data : { *(.data) }
.bss : { *(.bss COMMON) }
. = ALIGN(8);
. = . + 0x1000; /* 4kB of stack memory */
stack_top = .;
}
```

FOLLOW US
Press here
44
#LEARN IN DEPTH
#Be professional in embedded system

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

FOLLOW US
Press here
45
#LEARN IN DEPTH
#Be professional in embedded system

Create the binary file

Then run those commands

Generate startup object file

```
$ arm-none-eabi-as -mcpu=arm926ej-s -g startup.s -o startup.o
```

Generate test object file

```
$ arm-none-eabi-gcc -c -mcpu=arm926ej-s -g test.c -o test.o
```

Invoke the linker and pass the linker script

```
$ arm-none-eabi-ld -T test.ld test.o startup.o -o test.elf
```

Generate binary file

```
$ arm-none-eabi-objcopy -O binary test.elf test.bin
```

كل طالب فينا
جواه قوة روحيه
بتطلع لما يكون
interview فـ

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ arm-none-eabi-as -mcpu=arm926ej-s -g startup.s -o startup.o
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ arm-none-eabi-gcc -c -mcpu=arm926ej-s -g test.c -o test.o
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ arm-none-eabi-ld -T test.ld test.o startup.o -o test.elf
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ arm-none-eabi-objcopy -O binary test.elf test.bin
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ ls
startup.o  startup.s  test.bin  test.c  test.elf  test.ld  test.o
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$
```

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

FOLLOW US
Press here
46

#LEARN IN DEPTH

#Be professional in embedded system

To run the program in the VersatilePB QEMU

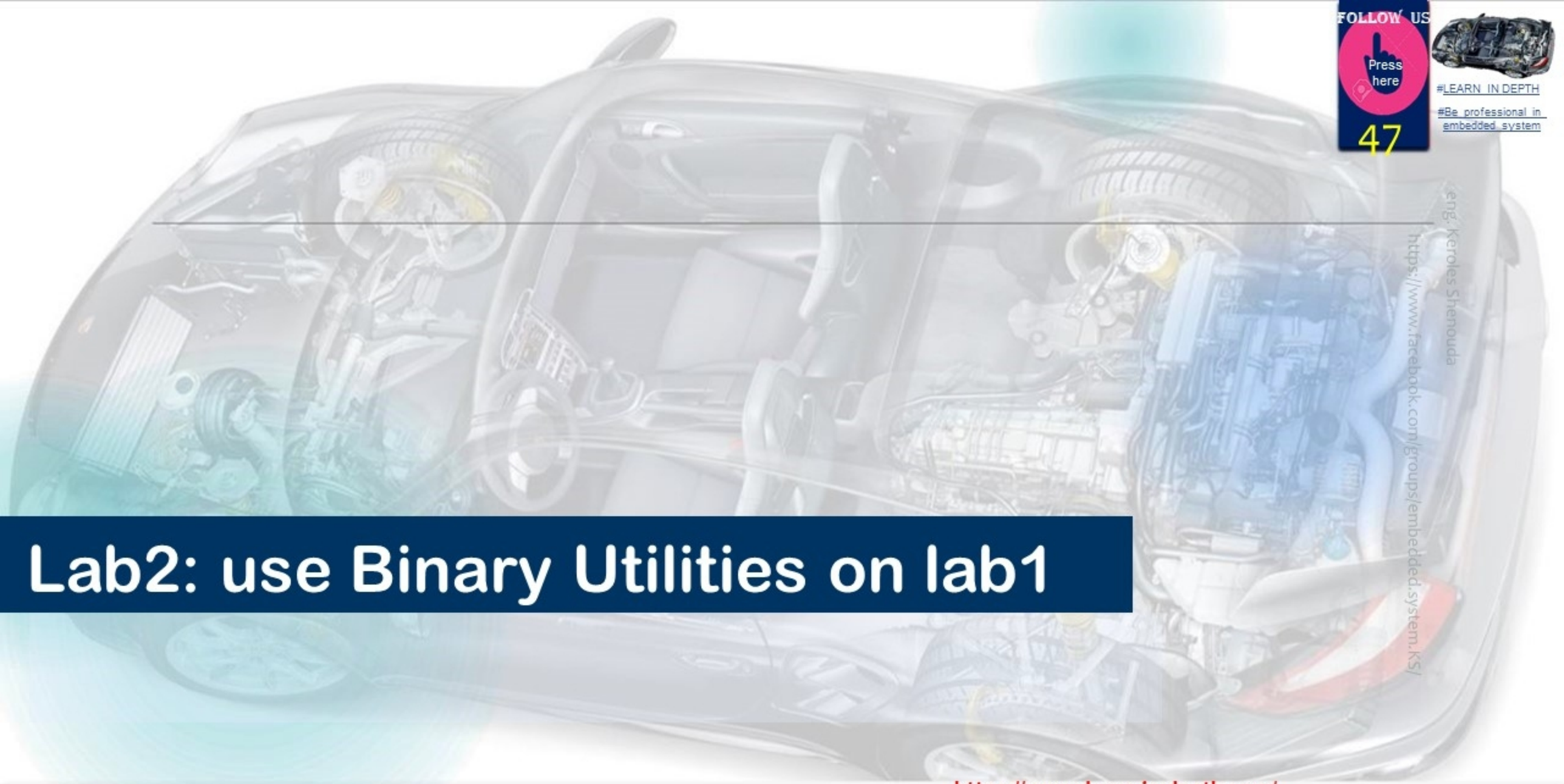
Run test.bin on the qemu by this Command

```
$ qemu-system-arm -M versatilepb -m 128M -nographic -kernel test.bin
```

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ ./run_versatilepb.sh test.bin
Hello world!
```

Embedded Linux
ENG.KEROLES SHENOUDA

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



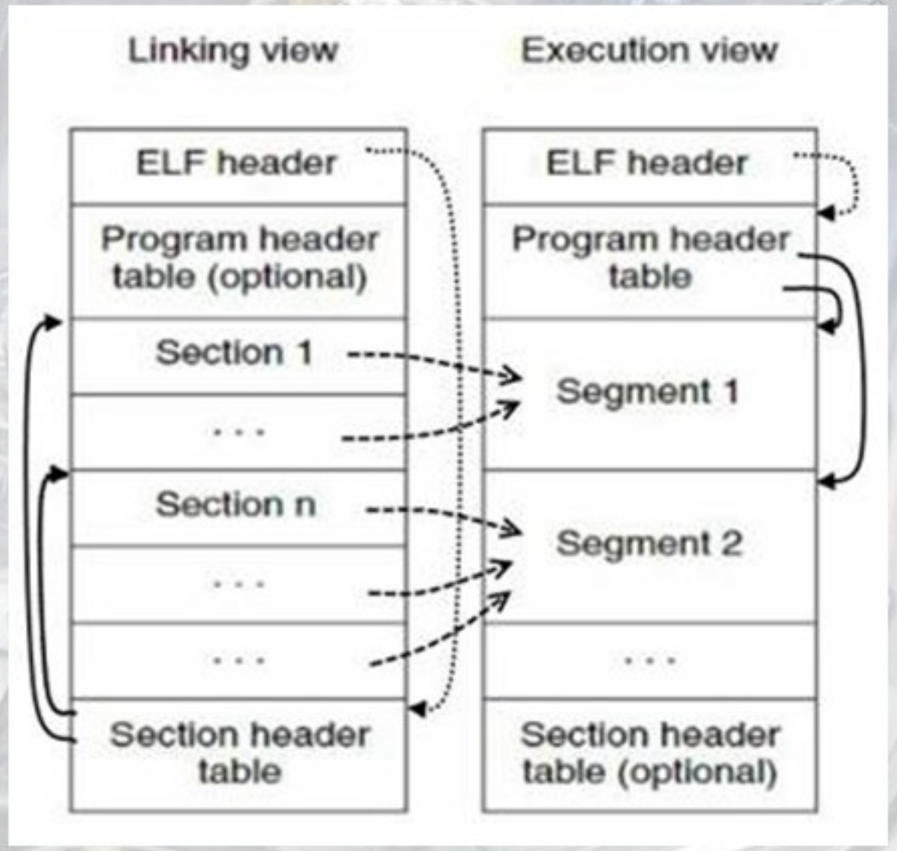
Lab2: use Binary Utilities on lab1

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

FOLLOW US
Press here
48
#LEARN IN DEPTH
#Be professional in embedded system

Read ELF Files (readelf Command)

- This command is used to read the contents of an ELF file
- This includes
  - Main file header
  - Section Headers
  - Sections
  - Symbol table
  - … etc
- Can be used for,
  - Resolving linking problems, such as "Unresolved Symbol" error
  - Debugging a crash
  - Hacking an executable
  - Reverse engineering a binary file



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

FOLLOW US
Press here
49
#LEARN IN DEPTH
#Be professional in embedded system

Try those commands and figure out the informations

- To get the header
  - $ arm-none-eabi-readelf -h test.elf
- To Show the file sections
  - $ arm-none-eabi-readelf -S test.elf
- To show the file segments
  - $ arm-none-eabi-readelf -l test.elf
- To show the symbol table
  - $ arm-none-eabi-readelf -s test.elf
- To show all the elf file contents
  - $ arm-none-eabi-readelf-a test.elf

Eng.Keroles Shenouda https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
50
#LEARN IN DEPTH
#Be professional in embedded system

$ readelf -h test.elf

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ readelf -h test.elf
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           ARM
  Version:                           0x1
  Entry point address:               0x10000
  Start of program headers:          52 (bytes into file)
  Start of section headers:          67228 (bytes into file)
  Flags:                             0x5000200, Version5 EABI, soft-float ABI
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         1
  Size of section headers:           40 (bytes)
  Number of section headers:         15
  Section header string table index: 12
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$
```

Little endian

Entry point = 0x10000

Arch = ARM

Embedded Linux
ENG.KEROLES SHENOUDA
Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
51
#LEARN IN DEPTH
#Be professional in embedded system

To Show the file sections

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ readelf
There are 15 section headers, starting at offset 0x1069c:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .startup          PROGBITS        00010000 010000 000010 00  AX  0   0  4
  [ 2] .text             PROGBITS        00010010 010010 000070 00  AX  0   0  4
  [ 3] .rodata           PROGBITS        00010080 010080 000012 00   A  0   0  4
  [ 4] .ARM.attributes   ARM_ATTRIBUTES  00000000 010092 00002e 00      0   0  1
  [ 5] .comment          PROGBITS        00000000 0100c0 00002b 01  MS  0   0  1
  [ 6] .debug_line       PROGBITS        00000000 0100eb 000076 00      0   0  1
  [ 7] .debug_info       PROGBITS        00000000 010161 0000f1 00      0   0  1
  [ 8] .debug_abbrev     PROGBITS        00000000 010252 00009a 00      0   0  1
  [ 9] .debug_aranges    PROGBITS        00000000 0102f0 000040 00      0   0  8
  [10] .debug_str        PROGBITS        00000000 010330 00008e 01  MS  0   0  1
  [11] .debug_frame      PROGBITS        00000000 0103c0 00004c 00      0   0  4
  [12] .shstrtab         STRTAB          00000000 010601 000098 00      0   0  1
  [13] .symtab           SYMTAB          00000000 01040c 0001b0 10     14  22  4
  [14] .strtab           STRTAB          00000000 0105bc 000045 00      0   0  1
```

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

To show the file segments

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ arm-none-eabi-readelf -l test.elf

Elf file type is EXEC (Executable file)
Entry point 0x10000
There are 1 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  LOAD           0x010000 0x00010000 0x00010000 0x00092 0x00092 R E 0x10000

 Section to Segment mapping:
  Segment Sections...
   00     .startup .text .rodata
```

We can see that the physical address = virtual address here ?

Think in depth why

Review part 1 and 2

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

Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
53
#LEARN IN DEPTH
#Be professional in embedded system

To show the symbol table

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ arm-none-eabi-readelf -s test.elf

Symbol table '.symtab' contains 27 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00010000     0 SECTION LOCAL  DEFAULT    1
     2: 00010010     0 SECTION LOCAL  DEFAULT    2
     3: 00010080     0 SECTION LOCAL  DEFAULT    3
     4: 00000000     0 SECTION LOCAL  DEFAULT    4
     5: 00000000     0 SECTION LOCAL  DEFAULT    5
     6: 00000000     0 SECTION LOCAL  DEFAULT    6
     7: 00000000     0 SECTION LOCAL  DEFAULT    7
     8: 00000000     0 SECTION LOCAL  DEFAULT    8
     9: 00000000     0 SECTION LOCAL  DEFAULT    9
    10: 00000000     0 SECTION LOCAL  DEFAULT   10
    11: 00000000     0 SECTION LOCAL  DEFAULT   11
    12: 00000000     0 FILE    LOCAL  DEFAULT  ABS startup.o
    13: 00010000     0 NOTYPE  LOCAL  DEFAULT    1 $a
    14: 0001000c     0 NOTYPE  LOCAL  DEFAULT    1 $d
    15: 00000000     0 FILE    LOCAL  DEFAULT  ABS test.c
    16: 00010080     0 NOTYPE  LOCAL  DEFAULT    3 $d
    17: 00010010     0 NOTYPE  LOCAL  DEFAULT    2 $a
    18: 00010060     0 NOTYPE  LOCAL  DEFAULT    2 $d
    19: 00010064     0 NOTYPE  LOCAL  DEFAULT    2 $a
    20: 0001007c     0 NOTYPE  LOCAL  DEFAULT    2 $d
    21: 00000010     0 NOTYPE  LOCAL  DEFAULT   11 $d
    22: 00010010    84 FUNC    GLOBAL DEFAULT    2 print_uart0
    23: 00011098     0 NOTYPE  GLOBAL DEFAULT    3 stack_top
    24: 00010064    28 FUNC    GLOBAL DEFAULT    2 c_entry
    25: 00010080     4 OBJECT  GLOBAL DEFAULT    3 UART0DR
    26: 00010000     0 NOTYPE  GLOBAL DEFAULT    1 _Reset
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$
```

Embedded Linux
ENG. KEROLES SHENOUDA
Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
54

#LEARN IN DEPTH
#Be professional in embedded system

Reading the ELF File (objdump Command)

- $ objdump [options] <.elf>
- This is another program to read from the ELF files
- It has a lot of usages depending of the chosen options
- For example, it can be used for,
  - Reading the ELF file Header (readelf does a better job with that)
  - Reading the ELF file sections headers
  - Reading the assembly code of the binary code
  - Reading the Symbol table(s)
  - Reading the Debug Information

Eng. Keroles Shenouda

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

FOLLOW US
Press here
55

#LEARN IN DEPTH
#Be professional in embedded system

Try those commands and figure out the informations

- Showing the ELF File Header
  - $ arm-none-eabi-objdump -f test.elf
- Display Sections Headers
  - $ arm-none-eabi-objdump -h test.elf
- Showing Assembly Code of an ELF
  - $ arm-none-eabi-objdump -d test.elf
- If you are interested in a specific section, then we need to identify what section to use for example ".startup"
  - $ arm-none-eabi-objdump --section=.startup -d test.elf

Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
56

#LEARN IN DEPTH

#Be professional in embedded system

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ arm-none-eabi-objdump -f test.elf

test.elf:     file format elf32-littlearm
architecture: arm, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x00010000

embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$
```

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ arm-none-eabi-objdump -h test.elf

test.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .startup      00000010  00010000  00010000  00010000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .text         00000070  00010010  00010010  00010010  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .rodata       00000012  00010080  00010080  00010080  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  3 .ARM.attributes 0000002e  00000000  00000000  00010092  2**0
                  CONTENTS, READONLY
  4 .comment      0000002b  00000000  00000000  000100c0  2**0
                  CONTENTS, READONLY
  5 .debug_line   00000076  00000000  00000000  000100eb  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_info   000000f1  00000000  00000000  00010161  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_abbrev 0000009a  00000000  00000000  00010252  2**0
                  CONTENTS, READONLY, DEBUGGING
  8 .debug_aranges 00000040  00000000  00000000  000102f0  2**3
                  CONTENTS, READONLY, DEBUGGING
  9 .debug_str    0000008e  00000000  00000000  00010330  2**0
                  CONTENTS, READONLY, DEBUGGING
 10 .debug_frame  0000004c  00000000  00000000  000103c0  2**2
                  CONTENTS, READONLY, DEBUGGING
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$
```

Embedded Linux
ENG. KEROLES SHENOUDA
Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

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

#LEARN IN DEPTH

#Be professional in embedded system

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ arm-none-eabi-objdump -d test.elf

test.elf:     file format elf32-littlearm


Disassembly of section .startup:

00010000 <_Reset>:
   10000:       e59fd004        ldr     sp, [pc, #4]    ; 1000c <_Reset+0xc>
   10004:       eb000016        bl      10064 <c_entry>
   10008:       eafffffe        b       10008 <_Reset+0x8>
   1000c:       00011098        .word   0x00011098

Disassembly of section .text:

00010010 <print_uart0>:
   10010:       e52db004        push    {fp}            ; (str fp, [sp, #-4]!)
   10014:       e28db000        add     fp, sp, #0
   10018:       e24dd00c        sub     sp, sp, #12
   1001c:       e50b0008        str     r0, [fp, #-8]
   10020:       ea000006        b       10040 <print_uart0+0x30>
   10024:       e59f2034        ldr     r2, [pc, #52]   ; 10060 <print_uart0+0x50>
   10028:       e51b3008        ldr     r3, [fp, #-8]
   1002c:       e5d33000        ldrb    r3, [r3]
   10030:       e5823000        str     r3, [r2]
   10034:       e51b3008        ldr     r3, [fp, #-8]
   10038:       e2833001        add     r3, r3, #1
   1003c:       e50b3008        str     r3, [fp, #-8]
   10040:       e51b3008        ldr     r3, [fp, #-8]
   10044:       e5d33000        ldrb    r3, [r3]
   10048:       e3530000        cmp     r3, #0
   1004c:       1afffff4        bne     10024 <print_uart0+0x14>
   10050:       e1a00000        nop                     ; (mov r0, r0)
   10054:       e24bd000        sub     sp, fp, #0
   10058:       e49db004        pop     {fp}            ; (ldr fp, [sp], #4)
   1005c:       e12fff1e        bx      lr
   10060:       101f1000        .word   0x101f1000

00010064 <c_entry>:
   10064:       e92d4800        push    {fp, lr}
   10068:       e28db004        add     fp, sp, #4
   1006c:       e59f0008        ldr     r0, [pc, #8]    ; 1007c <c_entry+0x18>
   10070:       ebfffffe6       bl      10010 <print_uart0>
   10074:       e1a00000        nop                     ; (mov r0, r0)
   10078:       e8bd8800        pop     {fp, pc}
   1007c:       00010084        .word   0x00010084
```

Embedded Linux
ENG.KEROLES.SHENOUDA
Eng.Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

#LEARN IN DEPTH

#Be professional in embedded system

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ arm-none-eabi-objdump --section=.startup  -d test.elf

test.elf:     file format elf32-littlearm


Disassembly of section .startup:

00010000 <_Reset>:
   10000:       e59fd004        ldr     sp, [pc, #4]    ; 1000c <_Reset+0xc>
   10004:       eb000016        bl      10064 <c_entry>
   10008:       eafffffe        b       10008 <_Reset+0x8>
   1000c:       00011098        .word   0x00011098
```

Embedded Linux
ENG.KEROLES SHENOUDA
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
59

#LEARN IN DEPTH
#Be professional in embedded system

Reduce ELF File Size (strip Command)

- $ strip [options] <object file>
- The strip command is used to reduce the size of the ELF file
- This applies for both executable or object files
- This is performed by removing some of the tables and sections that the binary can run without
- This is useful for:
  - Reduce the requirement for Flash and memory usage (specially useful for embedded systems)
  - Protect the code from being reverse engineered

Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
60
#LEARN IN DEPTH
#Be professional in embedded system

Reduce ELF File Size (strip Command)

- To strip an executable from its symbol table
  - $ arm-none-eabi-strip -s test.elf
  - So we can't see symbol table
- To remove debug symbols
  - $ strip --strip-debug my-bin-file
- Remove all un-needed symbols
  - $ strip --strip-unneeded my-bin-file

```
$ arm-none-eabi-strip -s test.elf
$ arm-none-eabi-strip -s test2.elf
$ arm-none-eabi-readelf -s test.elf
$
```

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

FOLLOW US
Press here
61

#LEARN IN DEPTH
#Be professional in embedded system

addr2line Command

- To know the location in the source code for a specific address
- This can be useful when handling a crash, and we know the address of the instruction that crashed, and need to know the source code line that caused the crash
- Try this command

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ arm-none-eabi-addr2line -f -e test.elf 0x10010
print_uart0
/media/embedded_system_ks/Embedded_Linux/LAB1/test.c:2
```

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

FOLLOW US
Press here
62

#LEARN IN DEPTH
#Be professional in embedded system

size Command

- $ size [options] <ELF File>
- This command is used to list the sizes of the different sections inside an elf file

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ arm-none-eabi-size test.elf
   text    data     bss     dec     hex filename
    146       0       0     146      92 test.elf
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$
```

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

Lab3: use gdb for debugging

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

Create the binary file -g

Then run those commands

Generate startup object file

```
$ arm-none-eabi-as -mcpu=arm926ej-s -g startup.s -o startup.o
```

Generate test object file

```
$ arm-none-eabi-gcc -c -mcpu=arm926ej-s -g test.c -o test.o
```

Invoke the linker and pass the linker script

```
$ arm-none-eabi-ld -T test.ld test.o startup.o -o test.elf
```

Generate binary file

```
$ arm-none-eabi-objcopy -O binary test.elf test.bin
```

كل طالب فينا
جواه قوة روحيه
بتطلع لما يكون
interview فـ

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_Linux/LAB1
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ arm-none-eabi-as -mcpu=arm926ej-s -g startup.s -o startup.o
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ arm-none-eabi-gcc -c -mcpu=arm926ej-s -g test.c -o test.o

embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ arm-none-eabi-ld -T test.ld test.o startup.o -o test.elf
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ arm-none-eabi-objcopy -O binary test.elf test.bin
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$
```

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

What is the gdb

- A debugger is a program that runs other programs, allowing the user to exercise control over these programs, and to examine variables when problems arise.
- GNU Debugger, which is also called **gdb**, is the most popular debugger for UNIX systems to debug C and C++ programs.
- GNU Debugger helps you in getting information about the following:
  - If a core dump happened, then what statement or expression did the program crash on?
  - If an error occurs while executing a function, what line of the program contains the call to that function, and what are the parameters?
  - What are the values of program variables at a particular point during execution of the program?
  - What is the result of a particular expression in a program?

Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

GDB – Commands

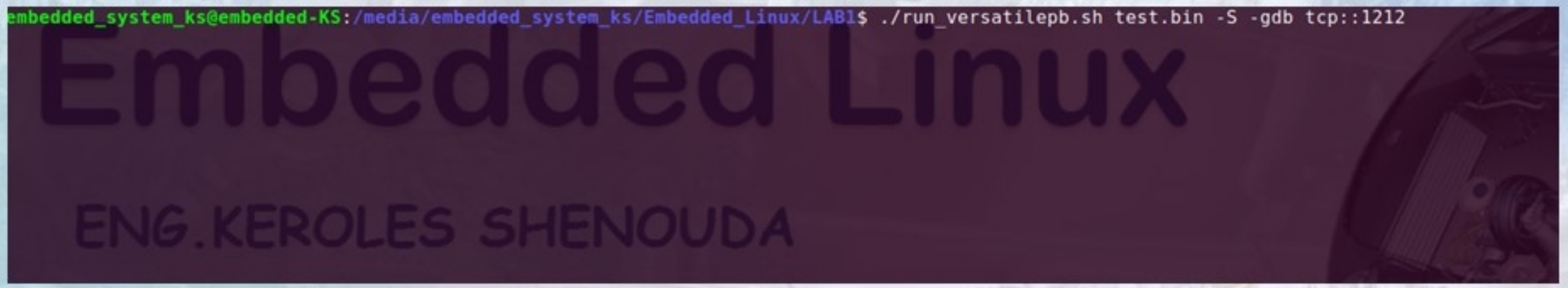
GDB offers a big list of commands, however the following commands are the ones used most frequently:

- **b main** - Puts a breakpoint at the beginning of the program
- **b** - Puts a breakpoint at the current line
- **b N** - Puts a breakpoint at line N
- **b +N** - Puts a breakpoint N lines down from the current line
- **b fn** - Puts a breakpoint at the beginning of function "fn"
- **d N** - Deletes breakpoint number N
- **info break** - list breakpoints
- **r** - Runs the program until a breakpoint or error
- **c** - Continues running the program until the next breakpoint or error
- **f** - Runs until the current function is finished
- **s** - Runs the next line of the program
- **s N** - Runs the next N lines of the program
- **n** - Like s, but it does not step into functions
- **u N** - Runs until you get N lines in front of the current line
- **p var** - Prints the current value of the variable "var"
- **bt** - Prints a stack trace
- **u** - Goes up a level in the stack
- **d** - Goes down a level in the stack
- **q** - Quits gdb

FOLLOW US
Press here
66
#LEARN IN DEPTH
#Be professional in embedded system
Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

So To Debug the Code

- Using gdb, because QEMU implements a gdb connector using a TCP connection. To do so, run the emulator with the correct options as follows
  $ qemu-system-arm -M versatilepb -m 128M -nographic -s -S -kernel test.bin



- This command freezes the system before executing any guest code, and waits for a connection on the TCP port 1212.
- From ARM ToolChan terminal, run arm-none-eabi-gdb and enter the commands:

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

FOLLOW US
Press here
68
#LEARN IN DEPTH
#Be professional in embedded system

Step 1

versatilepb

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_Linux/LAB1
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ ./run_versatilepb.sh test.bin -S -gdb tcp::1235
```

Waiting to be connected through port 1235

gdb

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_Linux/LAB1
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ arm-none-eabi-gdb
GNU gdb (7.11.90.20160917-0ubuntu1+9build1) 7.11.90.20160917-git
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-linux-gnu --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) file test.elf
Reading symbols from test.elf...done.
(gdb)
```

Read the debug info

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

FOLLOW US
Press here
69
#LEARN IN DEPTH
#Be professional in embedded system

Debug the Code

versatilepb

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_Linux/LAB1
[1]+  Done                    gedit run_versatilepb.sh
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ ./run_versatilep
b.sh test.bin -S-gdbtcp::1235
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ ./run_versatilep
b.sh test.bin -S -gdb tcp::1235
```

gdb

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_Linux/LAB1
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ arm-none-eabi-gdb
GNU gdb (7.11.90.20160917-0ubuntu1+9build1) 7.11.90.20160917-git
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-linux-gnu --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) file test.elf
Reading symbols from test.elf...done.
(gdb) target remote localhost:1235
Remote debugging using localhost:1235
0x00000000 in ?? ()
(gdb) l
1       .global _Reset
2       _Reset:
3       LDR sp, =stack_top
4       BL c_entry
5       B .
6
(gdb)
```

Embedded Linux
ENG.KEROLES SHENOUDA
https://www.facebook.com/groups/embedded.system.KS/
Eng.Keroles.Shenouda

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

FOLLOW US
Press here
70
#LEARN IN DEPTH
#Be professional in embedded system

Debug the Code

versatilepb

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_Linux/LAB1
[1]+  Done                    gedit run_versatilepb.sh
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ ./run_versatilep
b.sh test.bin -S-gdbtcp::1235
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ ./run_versatilep
b.sh test.bin -S -gdb tcp::1235
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ ./run_versatilep
b.sh test.bin -S -gdb tcp::1235
```

gdb

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_Linux/LAB1
1       .global  _Reset
2       _Reset:
3       LDR sp, =stack_top
4       BL c_entry
5       B .
6
(gdb) display/3i $pc
1: x/3i $pc
=> 0x0: mov     r0, #0
   0x4: ldr     r1, [pc, #4]    ; 0x10
   0x8: ldr     r2, [pc, #4]    ; 0x14
(gdb) b c_entry
Breakpoint 1 at 0x1006c: file test.c, line 9.
(gdb) si
0x00000004 in ?? ()
1: x/3i $pc
=> 0x4: ldr     r1, [pc, #4]    ; 0x10
   0x8: ldr     r2, [pc, #4]    ; 0x14
   0xc: ldr     pc, [pc, #4]    ; 0x18
(gdb) si
0x00000008 in ?? ()
1: x/3i $pc
=> 0x8: ldr     r2, [pc, #4]    ; 0x14
   0xc: ldr     pc, [pc, #4]    ; 0x18
   0x10:        andeq   r0, r0, r3, lsl #3
(gdb) c
Continuing.

Breakpoint 1, c_entry () at test.c:9
9       print_uart0("Hello world!\n");
1: x/3i $pc
=> 0x1006c <c_entry+8>: ldr     r0, [pc, #8]    ; 0x1007c <c_entry+24>
   0x10070 <c_entry+12>:        bl      0x10010 <print_uart0>
   0x10074 <c_entry+16>:        nop                     ; (mov r0, r0)
(gdb)
```

Embedded Linux

ENG.KEROLES SHENOUDA

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

FOLLOW US
Press here
71

#LEARN IN DEPTH

#Be professional in embedded system

Debug the Code

versatilepb

gdb

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_Linux/LAB1
[1]+  Done                    gedit run_versatilepb.sh
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ ./run_versatilep
b.sh test.bin -S-gdbtcp::1235
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ ./run_versatilep
b.sh test.bin -S -gdb tcp::1235
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ ./run_versatilep
b.sh test.bin -S -gdb tcp::1235
H
```

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_Linux/LAB1
   0x10074 <c_entry+16>:        nop                         ; (mov r0, r0)
(gdb) l
4               *UART0DR = (unsigned int)(*s); /* Transmit char */
5               s++; /* Next char */
6               }
7       }
8       void c_entry() {
9       print_uart0("Hello world!\n");
10      }
11
(gdb) b test.c:5
Breakpoint 2 at 0x10034: file test.c, line 5.
(gdb) c
Continuing.

Breakpoint 2, print_uart0 (s=0x10084 "Hello world!\n") at test.c:5
5               s++; /* Next char */
1: x/3i $pc
=> 0x10034 <print_uart0+36>:    ldr     r3, [r11, #-8]
   0x10038 <print_uart0+40>:    add     r3, r3, #1
   0x1003c <print_uart0+44>:    str     r3, [r11, #-8]
(gdb) display $s
2: $s = void
(gdb) print s
$1 = 0x10084 "Hello world!\n"
(gdb) s
3               while(*s != '\0') { /* Loop until end of string */
1: x/3i $pc
=> 0x10040 <print_uart0+48>:    ldr     r3, [r11, #-8]
   0x10044 <print_uart0+52>:    ldrb    r3, [r3]
   0x10048 <print_uart0+56>:    cmp     r3, #0
2: $s = void
(gdb) print s
$2 = 0x10085 "ello world!\n"
(gdb)
```

S++

Embedded Linux

ENG.KEROLES SHENOUDA

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

FOLLOW US
Press here
72
#LEARN IN DEPTH
#Be professional in embedded system

Debug the Code

versatilepb

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_Linux/LAB1
[1]+  Done                    gedit run_versatilepb.sh
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ ./run_versatilep
b.sh test.bin -S-gdbtcp::1235
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ ./run_versatilep
b.sh test.bin -S -gdb tcp::1235
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ ./run_versatilep
b.sh test.bin -S -gdb tcp::1235
He
```

Open Terminal
Copy
Paste
Profiles
Read-Only
✔ Show Menubar

gdb

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_Linux/LAB1
        breakpoint already hit 1 time
2       breakpoint     keep y   0x00010034 in print_uart0 at test.c:5
        breakpoint already hit 1 time
(gdb) info locals
No locals.
(gdb) l
1       volatile unsigned int * const UART0DR = (unsigned int *)0x101f1000;
2       void print_uart0(const char *s) {
3               while(*s != '\0') { /* Loop until end of string */
4               *UART0DR = (unsigned int)(*s); /* Transmit char */
5               s++; /* Next char */
6               }
7       }
8       void c_entry() {
9       print_uart0("Hello world!\n");
10      }
(gdb) backtrace
#0  print_uart0 (s=0x10085 "ello world!\n") at test.c:3
#1  0x00010074 in c_entry () at test.c:9
#2  0x00010008 in _Reset () at startup.s:4
Backtrace stopped: previous frame identical to this frame (corrupt stack?)
(gdb) c
Continuing.

Breakpoint 2, print_uart0 (s=0x10085 "ello world!\n") at test.c:5
5               s++; /* Next char */
1: x/3i $pc
=> 0x10034 <print_uart0+36>:    ldr     r3, [r11, #-8]
   0x10038 <print_uart0+40>:    add     r3, r3, #1
   0x1003c <print_uart0+44>:    str     r3, [r11, #-8]
2: $s = void
(gdb) backtrace
#0  print_uart0 (s=0x10085 "ello world!\n") at test.c:5
#1  0x00010074 in c_entry () at test.c:9
#2  0x00010008 in _Reset () at startup.s:4
```

Embedded Linux
ENG.KEROLES SHENOUDA

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

FOLLOW US
Press here
73

#LEARN IN DEPTH
#Be professional in embedded system

Debug the Code

versatilepb

gdb

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_Linux/LAB1
[1]+  Done                    gedit run_versatilepb.sh
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ ./run_versatilep
b.sh test.bin -S-gdbtcp::1235
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ ./run_versatilep
b.sh test.bin -S -gdb tcp::1235
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ ./run_versatilep
b.sh test.bin -S -gdb tcp::1235
Hello
```

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_Linux/LAB1
=> 0x10034 <print_uart0+36>:    ldr     r3, [r11, #-8]
   0x10038 <print_uart0+40>:    add     r3, r3, #1
   0x1003c <print_uart0+44>:    str     r3, [r11, #-8]
2: $s = void
(gdb) s
3               while(*s != '\0') { /* Loop until end of string */
1: x/3i $pc
=> 0x10040 <print_uart0+48>:    ldr     r3, [r11, #-8]
   0x10044 <print_uart0+52>:    ldrb    r3, [r3]
   0x10048 <print_uart0+56>:    cmp     r3, #0
2: $s = void
(gdb) s
4                       *UART0DR = (unsigned int)(*s); /* Transmit char */
1: x/3i $pc
=> 0x10024 <print_uart0+20>:    ldr     r2, [pc, #52]   ; 0x10060 <print_uart0+80>
   0x10028 <print_uart0+24>:    ldr     r3, [r11, #-8]
   0x1002c <print_uart0+28>:    ldrb    r3, [r3]
2: $s = void
(gdb) s

Breakpoint 2, print_uart0 (s=0x10088 "o world!\n") at test.c:5
5                       s++; /* Next char */
1: x/3i $pc
=> 0x10034 <print_uart0+36>:    ldr     r3, [r11, #-8]
   0x10038 <print_uart0+40>:    add     r3, r3, #1
   0x1003c <print_uart0+44>:    str     r3, [r11, #-8]
2: $s = void
(gdb) s
3               while(*s != '\0') { /* Loop until end of string */
1: x/3i $pc
=> 0x10040 <print_uart0+48>:    ldr     r3, [r11, #-8]
   0x10044 <print_uart0+52>:    ldrb    r3, [r3]
   0x10048 <print_uart0+56>:    cmp     r3, #0
2: $s = void
(gdb)
```

Embedded Linux

ENG.KEROLES SHENOUDA

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

gdb cheat sheet

GDB cheatsheet - page 1

Running

`# gdb <program> [core dump]`
Start GDB (with optional core dump).

`# gdb --args <program> <args…>`
Start GDB and pass arguments

`# gdb --pid <pid>`
Start GDB and attach to process.

`set args <args...>`
Set arguments to pass to program to be debugged.

`run`
Run the program to be debugged.

`kill`
Kill the running program.

Breakpoints

`break <where>`
Set a new breakpoint.

`delete <breakpoint#>`
Remove a breakpoint.

`clear`
Delete all breakpoints.

`enable <breakpoint#>`
Enable a disabled breakpoint.

`disable <breakpoint#>`
Disable a breakpoint.

Watchpoints

`watch <where>`
Set a new watchpoint.

`delete/enable/disable <watchpoint#>`
Like breakpoints.

<where>

function_name
Break/watch the named function.

line_number
Break/watch the line number in the current source file.

file:line_number
Break/watch the line number in the named source file.

Conditions

`break/watch <where> if <condition>`
Break/watch at the given location if the condition is met.
Conditions may be almost any C expression that evaluate to true or false.

`condition <breakpoint#> <condition>`
Set/change the condition of an existing break- or watchpoint.

Examining the stack

`backtrace`
`where`
Show call stack.

`backtrace full`
`where full`
Show call stack, also print the local variables in each frame.

`frame <frame#>`
Select the stack frame to operate on.

Stepping

`step`
Go to next instruction (source line), diving into function.

`next`
Go to next instruction (source line) but don't dive into functions.

`finish`
Continue until the current function returns.

`continue`
Continue normal execution.

Variables and memory

`print/format <what>`
Print content of variable/memory location/register.

`display/format <what>`
Like „print“, but print the information after each stepping instruction.

`undisplay <display#>`
Remove the „display“ with the given number.

`enable display <display#>`
`disable display <display#>`
En- or disable the „display“ with the given number.

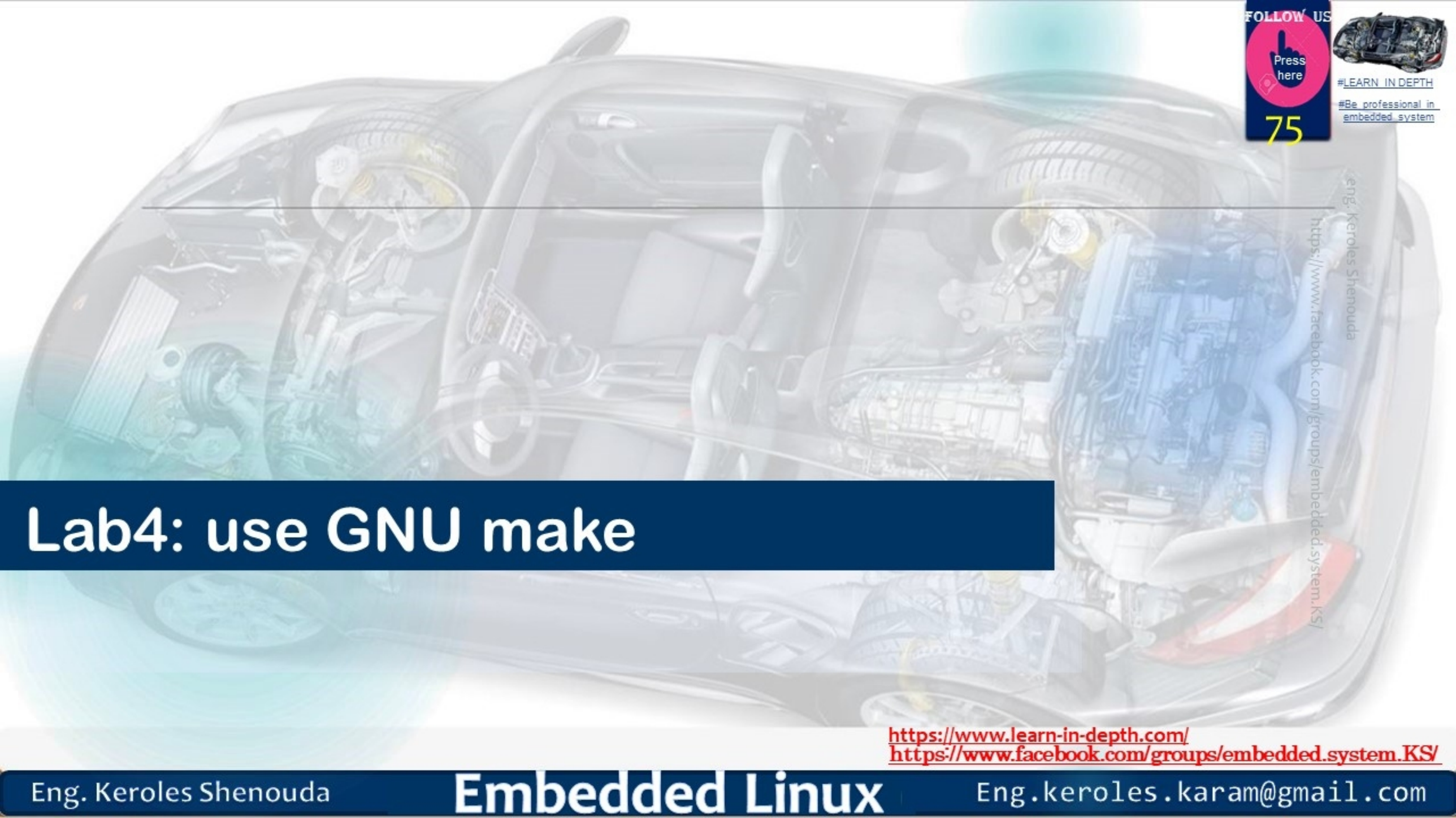
`x/nfu <address>`
Print memory.
n: How many units to print (default 1).
f: Format character (like „print“).
u: Unit.

Unit is one of:

b: Byte,
h: Half-word (two bytes)
w: Word (four bytes)
g: Giant word (eight bytes)).

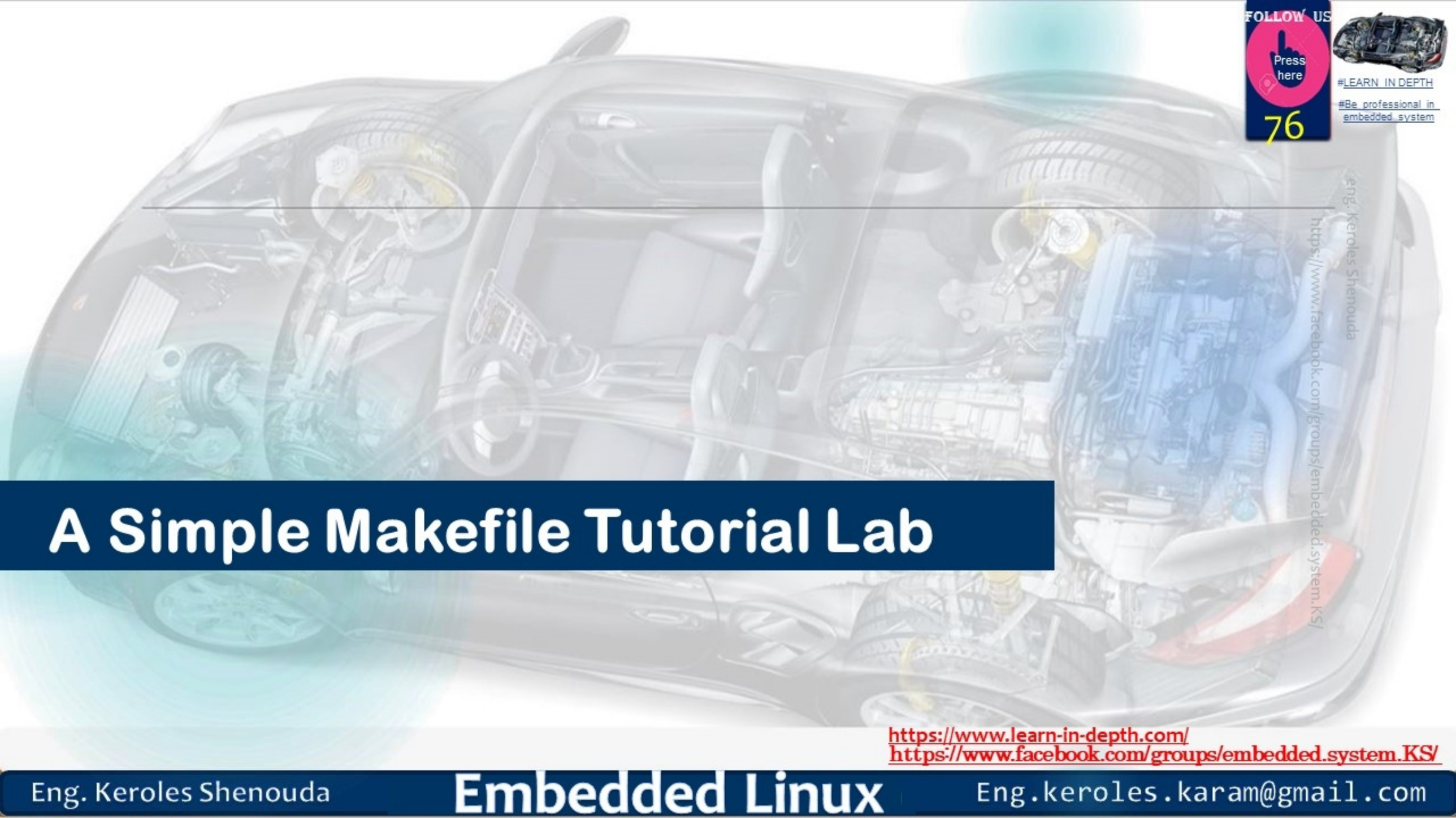
© 2007 Marc Haisenko <marc@darkdust.net>

FOLLOW US
Press here
74
#LEARN IN DEPTH
#Be professional in embedded system
Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/



Lab4: use GNU make

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



A Simple Makefile Tutorial Lab

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

FOLLOW US
Press here
77
#LEARN IN DEPTH
#Be professional in embedded system

GNU make

- GNU make is a tool to manage the build process for a software project
- It reads the set of build rules/targets inside a Makefile and use it to perform the build
- The **Makefile** contains a set of **rules/targets**, associated with some ***dependencies***

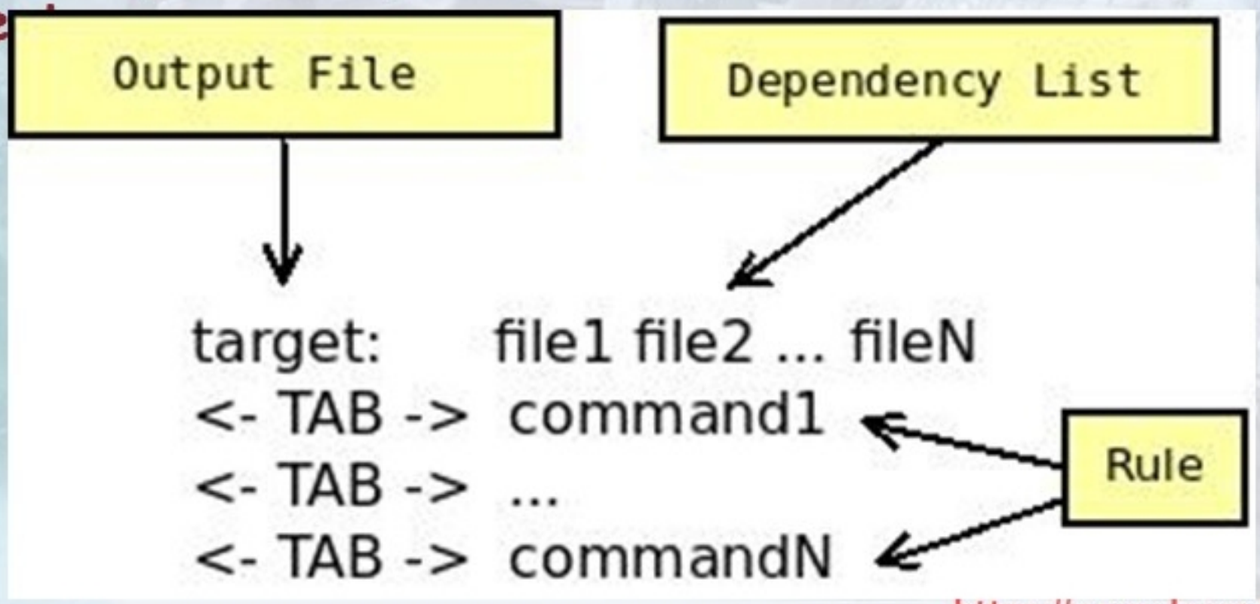
GNU make

Eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

FOLLOW US
Press here
78
#LEARN IN DEPTH
#Be professional in embedded system

GNU make

- Makefiles are a simple way to organize code compilation. This tutorial does not even scratch the surface of what is possible using *make*, but is intended as a starters guide so that you can quickly and easily create your own makefiles for small to medium-sized proje



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

FOLLOW US
Press here
79

#LEARN IN DEPTH
#Be professional in embedded system

Makefile 1

```
#prebared by Keroles Shenouda (Learn In Depth)
helloworld: test.c
        arm-none-eabi-as  -mcpu=arm926ej-s -g  startup.s  -o  startup.o
        arm-none-eabi-gcc -c -mcpu=arm926ej-s -g test.c -o test.o
        arm-none-eabi-ld -T test.ld test.o startup.o -o test.elf
        arm-none-eabi-objcopy -O binary test.elf test.bin
```

If you put this rule into a file called **Makefile** or **makefile** and then type **make** on the command line it will execute the compile command as you have written it in the makefile. Note that **make** with no arguments executes the first rule in the file. Furthermore, by putting the list of files on which the command depends on the first line after the :, make knows that the rule **helloworld** One very important thing to note is that there is a tab before the gcc command in the makefile. There must be a **tab** at the beginning of any command, and make will not be happy if it's not there.

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

FOLLOW US
Press here
80

#LEARN IN DEPTH

#Be professional in embedded system

Makefile 1

Makefile (Embedded_Linux /media/embedded_system_ks/Embedded_Linux/LAB1) - gedit

Open | Save

main.i | run_versatilepb.sh | Makefile

```
1 #prebared by Keroles Shenouda (Learn In Depth)
2 helloworld: test.c
3         arm-none-eabi-as  -mcpu=arm926ej-s  -g  startup.s  -o  startup.o
4         arm-none-eabi-gcc -c -mcpu=arm926ej-s -g test.c -o test.o
5         arm-none-eabi-ld -T test.ld test.o startup.o -o test.elf
6         arm-none-eabi-objcopy -O binary test.elf test.bin
7
```

Makefile | Tab Width: 9 | Ln 3, Col 10 | INS

embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_Linux/LAB1

```
mbedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ make
rm-none-eabi-as  -mcpu=arm926ej-s  -g  startup.s  -o  startup.o
rm-none-eabi-gcc -c -mcpu=arm926ej-s -g test.c -o test.o
rm-none-eabi-ld -T test.ld test.o startup.o -o test.elf
rm-none-eabi-objcopy -O binary test.elf test.bin
mbedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$
```

ENG.KEROLES SHENOUDA

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

FOLLOW US
Press here
81

#LEARN IN DEPTH

#Be professional in embedded system

Makefile 2

```
1  #prebared by Keroles Shenouda (Learn In Depth
2  CC=arm-none-eabi
3  CFLAGS=-I. -g
4
5  helloworld: test.c
6      $(CC)-as -mcpu=arm926ej-s $(CFLAGS) startup.s -o startup.o
7      $(CC)-gcc -c -mcpu=arm926ej-s $(CFLAGS) test.c -o test.o
8      $(CC)-ld -T test.ld test.o startup.o -o test.elf
9      $(CC)-objcopy -O binary test.elf test.bin
10
11
12
```

```
D:\courses\new_diploma\C\diploma4\ARM\bin>make helloworld
arm-none-eabi-as -mcpu=arm926ej-s -g startup.s -o startup.o
startup.s: Assembler messages:
startup.s: Warning: end of file not at end of a line; newline inserted
arm-none-eabi-gcc -c -mcpu=arm926ej-s -g test.c -o test.o
arm-none-eabi-ld -T test.ld test.o startup.o -o test.elf
arm-none-eabi-objcopy -O binary test.elf test.bin
```

https://www.facebook.com/groups/embedded.system.KS/ eng.Keroles Shenouda

defined some constants CC and CFLAGS. It turns out these are special constants that communicate to make how we want to compile the files test.c In particular, the macro CC is the C compiler to use, and CFLAGS is the list of flags to pass to the compilation command.

#LEARN IN DEPTH

#Be professional in embedded system

Makefile 3

```
1  #prebared by Keroles Shenouda (Learn In Depth)
2  CC=arm-none-eabi
3  CFLAGS=-I. -g -mcpu=arm926ej-s
4  DEPS = *.h
5
6  startup.o: startup.s
7      $(CC)-as $(CFLAGS)  $< -o $@
8
9  test.o: test.c
10     $(CC)-gcc -c -o $@ $< $(CFLAGS)
11
12 test.elf: startup.o test.o
13     $(CC)-ld -T test.ld test.o startup.o -o test.elf
14
15 test.bin: test.elf
16     $(CC)-objcopy -O binary $< $@
17
18
19
```

```
D:\courses\new_diploma\C\diploma4\ARM\bin>make test.bin
arm-none-eabi-as -I. -g -mcpu=arm926ej-s   startup.s -o startup.o
startup.s: Assembler messages:
startup.s: Warning: end of file not at end of a line; newline inserted
arm-none-eabi-gcc -c -o test.o test.c -I. -g -mcpu=arm926ej-s
arm-none-eabi-ld -T test.ld test.o startup.o -o test.elf
arm-none-eabi-objcopy -O binary test.elf test.bin
```

eng.Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

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

FOLLOW US
Press here
83

#LEARN IN DEPTH

#Be professional in embedded system

Makefile 4

```
1  #prebared by Keroles Shenouda (Learn In Depth)
2  CC=arm-none-eabi
3  #DEPS = *.h
4  DEPS =
5  #IDIR =../include
6  IDIR =
7  CFLAGS=-I$(IDIR) -g -mcpu=arm926ej-s
8  ODIR=obj
9  LDIR =
10
11 startup.o: startup.s
12     $(CC)-as $(CFLAGS) $(LIBS) $< -o $@
13
14 %.o: %.c $(DEPS)
15     $(CC)-gcc -c -o $@ $< $(CFLAGS) $(LIBS)
16
17 test.elf: startup.o test.o
18     $(CC)-ld -T test.ld test.o startup.o -o test.elf
19
20 test.bin: test.elf
21     $(CC)-objcopy -O binary $< $@
22
23
24 clean:
25     rm -f $(ODIR)/*.o *~ core $(INCDIR)/*~
26
27
28 .PHONY: test.bin
```

```
D:\courses\new_diploma\C\diploma4\ARM\bin>make test.bin
arm-none-eabi-as -I -g -mcpu=arm926ej-s   startup.s -o startup.o
startup.s: Assembler messages:
startup.s: Warning: end of file not at end of a line; newline inserted
arm-none-eabi-gcc -c -o test.o test.c -I -g -mcpu=arm926ej-s
arm-none-eabi-ld -T test.ld test.o startup.o -o test.elf
arm-none-eabi-objcopy -O binary test.elf test.bin
```

eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

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
- Ahmed ElArabawy Courses at http://linux4embeddedsystems.com/

eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

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

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
- Linux basic commands

eng. Keroles Shenouda
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

eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

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

References Cont.

- Chapter 2: Operating-System Structures
- **POSIX Threads Programming**
- Pthreads: A shared memory programming model https://slideplayer.com/slide/8734550/
- Signal Handling in Linux Tushar B. Kute
- Introduction to Sockets Programming in C using TCP/IP
- **How to Run Two or More Terminal Commands at Once in Linux**
- **How to Build a GCC Cross-Compiler**

eng. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/