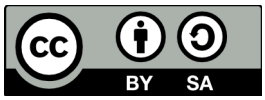
Exercises for

Embedded Linux

v 2012.02

WARNING:
The order of the exercises does not always follow the same order of the explanations in the slides. When carrying out the exercises, carefully follow the exercise requirements. Do **NOT** blindly type the commands as found in the slides. Read every exercise in its entirety before carrying out the instructions.



These exercises created by: Karim Yaghmour

Originals at: www.opersys.com/community/docs

Delivered and/or customized by:

Software components versions

Workspace:

Download location:	http://opersys.com/downloads/lite5200-workspace.tar.bz2
Download location:	http://opersys.com/downloads/tqm860-workspace.tar.bz2

Linux Kernel:

Download location:	http://www.kernel.org
Version:	2.6.37
Patch for EVB5200:	NONE
Patch for TQM860L:	NONE
Configuration for EVB5200:	52xx/lite5200b_defconfig
Configuration for TQM860L:	tqm8xx_defconfig
Image to use for EVB5200:	arch/powerpc/boot/images/cuImage.lite5200
Image to use for TQM860L:	arch/powerpc/boot/images/cuImage

Bootloader:

Package:	U-Boot
Download location:	http://www.denx.de/wiki/U-Boot
Version:	2010.12
Patch for EVB5200:	u-boot-2010.12-fix-lite5200.patch
Patch for TQM860L:	NONE
Makefile target for TQM860L:	TQM860L_config
Makefile target for EVB5200:	icecube_5200_LOWBOOT_config

TARGET SERIAL NUMBER:

Chapter 2:

1) Used “sudo adduser” to create a “stage” user. Follow the instructions to add the new user and set its password.
2) Using the “stage” account, download and extract the workspace for your target. Having selected and copied to /home/stage one of lite5200-workspace.tar.bz2 or tqm860-workspace.tar.bz2, use the “tar xjf” command to extract the workspace.
3) Make the proper changes so that your are able to access your serial port from the “stage” user account. You will need to log out completely (close all windows and log out), and log back in in order for the changes to take effect. You might be able to reuse existing group settings for the relevant files instead of those described in the slide, have a look at the current group settings for /dev/ttyS0. In order to configure minicom, you will need to log in as root and use the “minicom -s” command.
4) Verify that there is an existing configuration for your Ethernet card allowing connection to the Internet. Explore the “double-arrow” icon on the Ubuntu toolbar by clicking and right-clicking it.
5) Create a local-LAN configuration for your Ethernet card to allow connection to the target using a cross-over Ethernet cable (host: 192.168.202.100; target: 192.168.202.79).

Example Ethernet card configuration:

```
IPADDR=192.168.202.100
GATEWAY=192.168.202.79
NETWORK=192.168.202.0
NETMASK=255.255.255.0
BROADCAST=192.168.202.255
```

Chapter 3:

0) Apply any necessary kernel patch
1) Configure the kernel for the target. Use the configuration listed on the “Software components versions” sheet of the exercise set as the basis of your configuration. In addition to the values already pre-set, add support for config.gz in /proc (“Kernel .config support”).
2) Build the kernel
3) Install the kernel
4) Build the kernel modules
5) Install the kernel modules

Chapter 4:

0) Apply any necessary U-Boot patch
1) Build U-Boot
2) Boot with U-Boot and try out the online help
3) Print U-Boot's environment variables

4) Set the environment variables as listed on the bottom of p. 79 and save them. For the TQM860L, you might need to replace the “off” string in the “nfscmd” environment variable with “eth0”. The “rootpath” will have to match your own development environment.
5) Create a U-Boot image of the kernel built earlier
6) Configure your host for serving TFTP requests
7) Use TFTP to download image to target. For the EVB5200, you will need to download the image to 0x00200000 instead of 0x00100000 because of its size.
8) Boot with image (for the EVB5200, you will need to set the “bootargs” environment variable to “console=ttyS0,115200”.)

Chapter 5:
1) Create essential rootfs directories
2) Copy glibc libraries to target's rootfs and strip them. NOTE: contrary to the slides, the libraries are in ${TARGET_PREFIX}/${TARGET}/sysroot/lib.
3) Copy kernel modules to target's rootfs
4) Create nodes on target's rootfs
5) Configure BusyBox using the “make menuconfig” command
6) Build and install BusyBox
7) Create initialization scripts for BusyBox init. Don't forget to set /etc/init.d/rcS to allow for it to execute (chmod 755 /etc/init.d/rcS). Contrary to the examples in the slides, do **not** put an entry in /etc/fstab for NFS, and do **not** put an entry for “custom-app” in /etc/inittab.

Chapter 6:
1) Build and install the MTD utilities for your host (p. 149). **NOTE:**
- For zlib, you must edit the “configure” file and change lines 268 and 269 for these:
```
CFLAGS="${CFLAGS}"
SFLAGS="${SFLAGS}"
```
- Again for zlib, use this line for the install instead of the one on the slide:
```
$ make prefix=${TARGET_PREFIX}/${TARGET}/sysroot/ install
```
- For lzo, use these configure and install instructions instead of those on the slide:
```
$ CC=${TARGET}-gcc ./configure --enable-shared --host=${TARGET}
$ make prefix=${TARGET_PREFIX}/${TARGET}/sysroot/ install
```

2) Reconfigure the target kernel we built earlier to add support for the MTD subsystem (use menuconfig, look under “Device Drivers”). See p. 152 for the options to set. Set all options as built-in (“*”) not as modules (“M”). Ignore the mapping drivers. Also, in the “Device Drivers”->“Block devices” submenu, and make sure “RAM block device support” is built-in (not as modules.)
3) Rebuild the kernel and update the kernel for U-Boot image in the images/ directory.
4) Build the MTD tools for the target and install them in your target's root filesystem:
```
$ cp -a ${TARGET_PREFIX}/${TARGET}/sysroot/include/* include/
$ make CROSS="${TARGET}-"
$ cp powerpc-unknown-linux-gnu/flash* ${PRJROOT}/rootfs/sbin/
```

5) Create the appropriate /dev entries for the MTD partitions as shown on p.154.

Chapter 7:

1) Build and install the cramfs utilities (p. 166)
2) Create a CRAMFS image of your target's root filesystem (p. 167)
3) Build and install the romfs utilities (p. 168)
3) Create a ROMFS image of your target's root filesystem (p. 168)
4) Build and install the squashfs utilities (p. 169)
5) Create a squashfs image of your target's root filesystem (p. 169)
6) Create a UBIFS image of your target's root filesystem (p. 170)
7) Create a JFFS2 image of your target's root filesystem (p. 171)
8) Create a RAM disk image of your target's root filesystem (p. 172-174)
9) Create an initramfs images of your target's root filesystem (175)
10) Compare the filesystem image sizes
11) Configure your host and your target so that the target's configuration is obtained at boot time via DHCP and the rootfs is mounted on NFS.

Chapter 8:

1) Add a file called my_panic.c in the kernel's kernel/ directory. That file should contain the code on pages 217 and 218. You will need to replace the call to “notifier_chain_register” with a call to “atomic_notifier_chain_register”. In the write_raw_user_screen(void), put the following:

printk(“This box has just panicked ... \n”);

2) Modify kernel/Makefile to reflect the changes in #1
3) Cross-compile the modified kernel as we did in chapter 3
4) Boot this new kernel directly using the bootm command. Given that there is no root filesystem, the kernel should panic and the above message should be displayed.

Chapter 10: 3 hours

Write a dynamically loadable device driver for the target that:

a) Implements a character device with the open(), release(), read(), and write() functions.
b) Has a fixed-size array (~20 entries) where it records for each call to read():
    a) the PID of the process that called read()
    b) the exact time at which that call was made
c) Upon a write(), the driver should print out to the console the content of its array.

O'Reilly's “Linux Device Drivers, 3rd ed.” is a useful reference for this exercise. It is found online at: http://lwn.net/Kernel/LDD3/. There is a copy of a makefile to use for building your driver in LDD3. Using that makefile, you can use a command that has “ARCH=...” and “CROSS_COMPILE=...” to build your module.

mmap:

NOTE: Read ALL these exercises once before starting. They are all linked together.

1) Create a character driver that allocates a large (X*MB) buffer and exports the mmap() file-op thereby allowing user-space processes to mmap() the buffer to user-space. Make sure all bytes in the buffer are zeroed out before returning from mmap(). Have a look at vmalloc_user() and remap_vmalloc_range().

2) Create a user-space application that opens the device corresponding to your character driver and calls on mmap to map the driver's buffer in its address space.

3) Add an ioctl() call to your driver allowing a user-space process to set to “zero” all bytes in the buffer, and write a separate application that opens your device and uses said ioctl() call to clear all bytes in the buffer.

4) Modify the kernel module implementing your character driver to launch a kernel thread in its initialization function. Have the thread write a 0xAA byte at the first unused address of the buffer and periodically check whether the next byte is 0x55 and, if so, then writes 0xAA right after 0x55. Make sure to implement a circular buffer.

5) Modify the application coded in #2 to periodically monitor the buffer's first unused address for the 0xAA value and if found, write 0x55 value in the next byte. Again, make sure you circle around to the first byte when you get to the end of the buffer.

Chapter 13:

1) Create a small program, compile it with the debugging flags, and install it on your target's root filesystem. Use the gdb server to remotely step through your program, as is explained on page 276. You will find the “gdbserver” binary in: ${TARGET_PREFIX}/${TARGET}/debug-root/usr/bin

2) Use strace on the target to observe the behavior of a few processes (p. 279 and 280.) The “strace” binary is in the same location as the “gdbserver”.

Chapter 16:

1) Build and install crosstool-ng (p. 383). Make sure you have all the prerequisite packages installed.

2) Configure a toolchain for the target and build it (p. 383-384).

3) Rebuild the kernel and regenerate the rootfs using the new toolchain. Recreate all the appropriate images and restart the target using those images in order to validate the new toolchain.