H2 1. Download or develop the driver source code in the proper directory

The driver file and makefile and Kconfig file should be put inside the char directory.

H2 2. Download or develop the the application source code in the proper directory

User program should be put outside the linux system

`arm-linux-gcc` compiling our user source code to generate machine code for ARM Linux machines.

H2 3. Driver compilation

> Use menuconfig tool to automatically generate/configure the Makefile, which has als o a per directory configuration file called kconfig.
>
> In the interface choose M which means I want to compile this driver as an individual module to be loaded into kernel during runtime.
>
> Make will traverse recursively all the subdirectories and use Makefiles instructions to trigger the compiler GCC to do compilation and linking of the local files.
>
> The compilation in this sense is automated and customized.

`make menuconfig` this will bring out the graphical user interface for you to automate the configuration of all the make files. So we're going to tell the make file system that we are adding one more driver and so we go to the option device driver, enter, choose it and then this is a character device. This item coms from Kconfig.

`choose yes` Choose yes, then it will modify all the make files accordingly according to your configuration.

`make` Now, with the Makefiles configured properly, I can type make to compile and link the entire kernel.

We have the loadable module `helloworld_driver.ko`

H2 4. Driver loading and device file set up

`insmod /path/to/helloworld_driver.ko`

This loads the driver code into the kernel, run the helloworldinit() to register the driver and allocates a major number and 2 minor number

After this, `lsmod` and `cat proc/devices` (fake as regular file and to show all the current modules that loaded onto the kernel) are available.

`mknod /dev/helloworld c 250 1` establish the interface/access point (file) from the user space (package module into a device file), and this links the file operations with driver function calls

H2 5. Application execution

`./helloworld`

Run the user program to interact with the driver, you issue a open/read/close function system call. This read system call will be forwarded by the kernel and kernel check the character device driver table to look for the row of 250, the major number of the device driver. OK, the role of 250 and looking to the column called read function and there will be a function pointer pointing to the my read function of the driver. So the myopen/myread/myclose function will be executed and in the my read it will do this copy message to user to return.

`printk` has higher priority than `printf` -> the close message print first

6. Device file deletion and driver unloading

```
rm /dev/helloworld
```

Remove the access point from the user program first. Then we do module release:

```
rmmod helloworld_driver
```

This will call the __exit routine in the driver file and which is mapped to helloworldexit function to unregisters the character device and frees the major/minor numbers

7. The student can successfully and correctly complete small changes to satisfy ad hoc change requests proposed in the Q & A section

See the Q & A

Nfs will fake a local file system based on remote hard disk drive