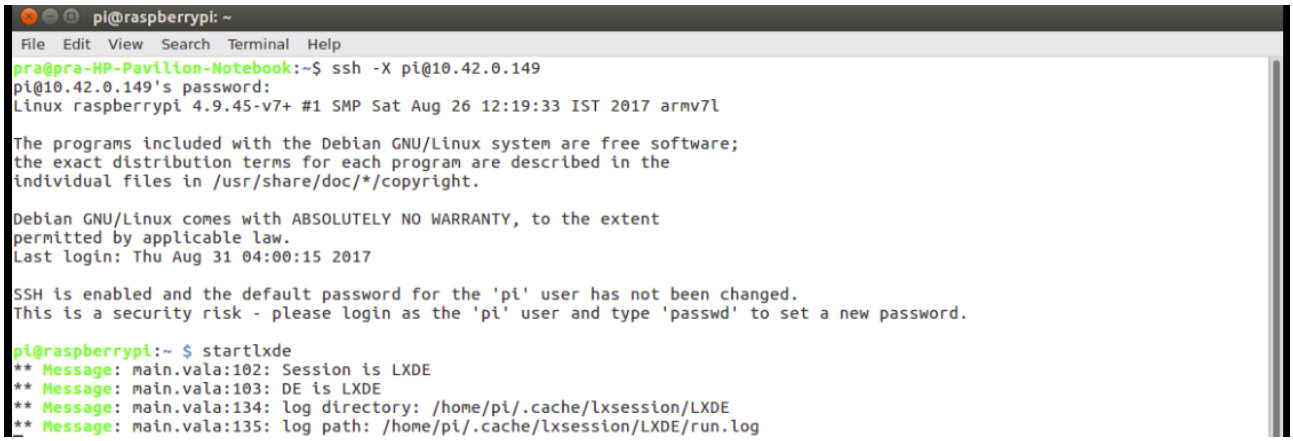
Kernel Modules on Raspbian Stretch

Submitted by:Prashanthi S.K
MT2016520

Procedure followed

I booted my Raspberry Pi with the Raspbian stretch distribution.Thereafter,I tried to set up networking on the pi so that I could connect to it using my laptop instead of using a monitor,keyboard and mouse.I learnt that the network interfaces had undergone changes in Stretch.I first noticed the enx..(mac address) naming instead of the ususal eth0/eth1.Also,the normal configuration of /etc/network/interfaces was now split up into two:/etc/network/interfaces and /etc/dhcpcd.conf.I initially tried using a combination of vnc server and viewer on Windows.I was able to log in to the pi,but internet sharing(ICS) on Windows was very troublesome.It wouldn't work half of the time.I also had to frequently reboot.I struggled with setting up netwroking on Windows for hours before I decided to move to Linux(Ubuntu) and use ssh(with x11 forwarding) with lxde.This setup was very easy and worked flawlessly.The only glitch was that ssh needed to be enabled before using this method,which is kind of circular,because you need a monitor to enable ssh when you're trying to avoid using a monitor in the first palce.I pondered over this and decided that there must be an alternate method of enabling ssh.Somewhere on the net,I read that this is possible using a few hacks.I didn't try those because I had easy access to a monitor in the lab and directly used raspi-config to enable it.Now,I could connect to my pi using my laptop.



The next step was to look up how to set up the pi so that it is ready to compile kernel modules.The first resource I found suggested using ap-get to install the kernel headers and that would be sufficient.To be honest,this looked too simple.I took another look at the assignment and in the references I found links to kernel building,firmware and rpi-source.After seeing this,I felt this was the path to follow--to manually compile the kernel,update to the latest and appropriate firmware and then download the source for that version.I was very puzzled because this seemed a very roundabout way of doing what kernel-headers offered in one line.I decided that this second method would teach me a lot more than the first(that being more like a black box) so I started following the steps to build the kernel.

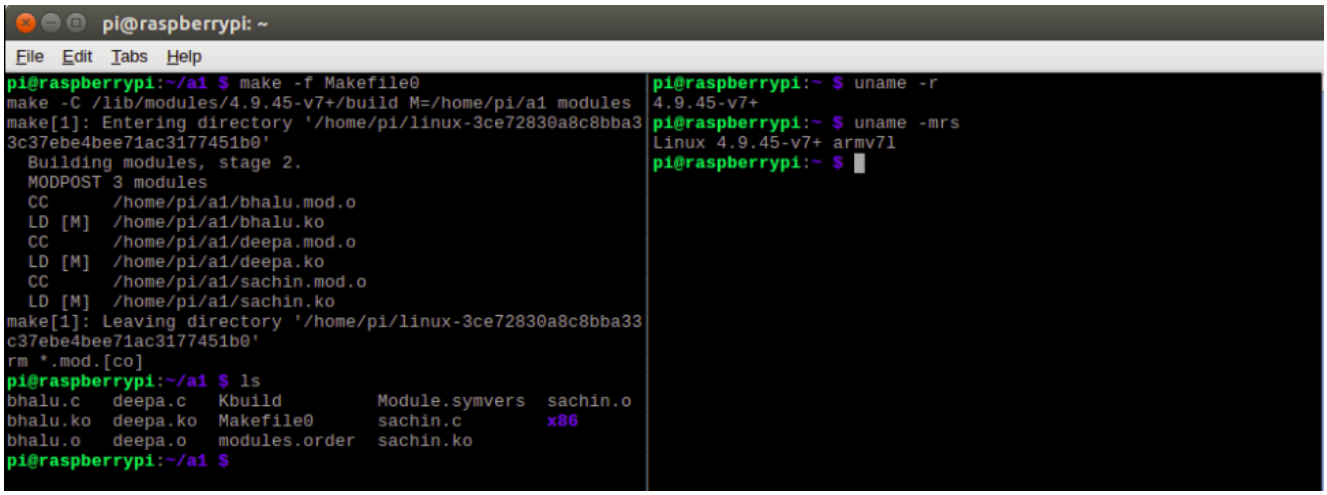
During kernel building using cross-compilation,the first error I encountered was that I was using the vanilla gcc instead of arm gcc.This was quickly solved by an apt-get isntall.After that,make took very long despite using the -jn flag.So I made the mistake of letting it run on its own and not looking at the screen output.After it was done,I took a look and everything seemed fine.I put the SD card back in and booted it up,only to find that both the keyboard and the mouse weren't working.I instantly understood that the kernel hadn't been compiled properly and felt miserable becuase I had ruined my sd card and I now had to repeat the whole ordeal again.I went back to looking at the steps I had followed.Since I had typed out the commands manually instead of copy-pasting them,I had missed out a space.Instead of arm-gnueabihf- INSTALL I had typed arm-gnueabihf-INSTALL.This had thrown a few errors initially which I missed out due to the deluge of statements being printed on the screen.I redid the process keeping my eye on the screen throughout.It was then that I realised that I now had a kernel7-backup.img which was a backup of the original vanilla kernel.I felt happy about this because I could always go back to this by adding a line in the config.txt.This gave me some reassurance and confidence and I was glad I followed this procedure.Another point that puzzled me was the existence of two kernel files:kernel and kernel7.I looked it up to find that Pi2 and 3 used 7 and the older ones use kernel.

This time around,my pi booted successfully.I saw that I had two kernels now--the original one and the one I had compiled.I tried compiling the module sachin using the makefile but it gave me an error saying no directory build.I naviagted to the build directory of the currently used kernel to sees an exclamatory mark on both build and source--they were not present.I figured out that this is why rpi-source is required.I installed and ran rpi-source following the instructions on the github wiki page.I downgraded to a lower version of gcc so that it matched the gcc version that my kernel was compiled on.Therafter,when I ran rpi-source,it gave me an error saying it could not obtain the gcc version.This made me go through the spource code of rpi-source and I was astonished to find that it was pretty simple.I did a check of proc/version and gcc --version to find that they matched.So,I decided to use the skip-gcc flag and get the sources directly.I did this and the source files were downloaded--the exclamatory marks over build and source vanished.I did a make and all 3 modules compiled successfully.

I happily did an insmod only to get a layout mismatch error.I called up modinfo and found that they were compiled for the wrong version of the kernel---the older one instead of the currently used one.I checked the build and source directories that rpi-source had populated to find the same thing--the source was for a different kernel version.This made me go trhough the source code of rpi-source again and I noticed rpi-update in one of the lines--that's when it struck me.I hadn't updated the firmware.I immediately did an rpi-update and then re-ran rpi source.This time,the modules were compiled for the right version of the kernel and I had no issue inserting or removing them.Though this whole process was extremely laborious and I had to repeat several of the steps,I learnt a lot from it--far more than if I had just used install headers.Therefore,I have no regrets about having taken the long route.It has taught me a lot.

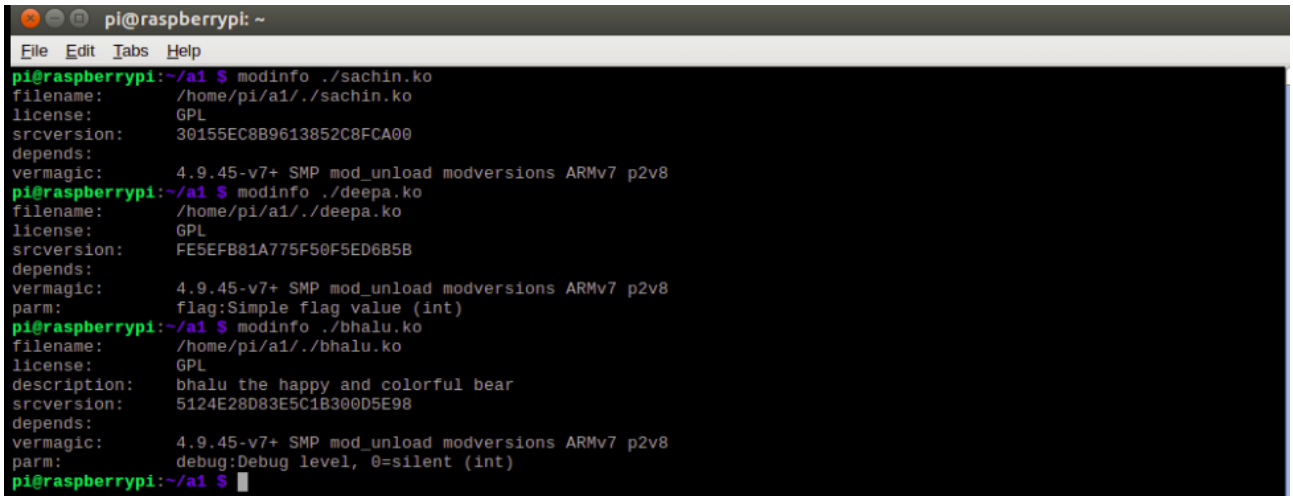
Observations

- Run make in demo1 directories to build the modules successfully

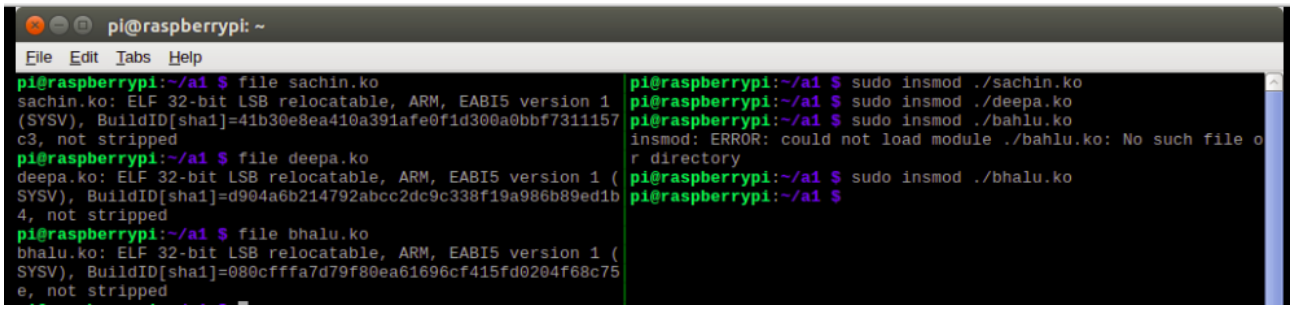


- Check the proper binary (x86 or ARM) format

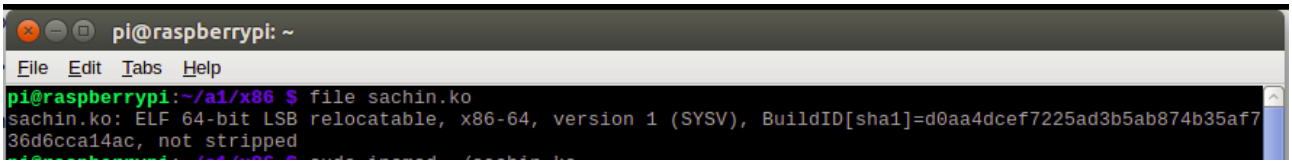
By printing modinfo for all the modules,we can see whether it is compiled for ARM or x86



Another method of checking is using the file command.

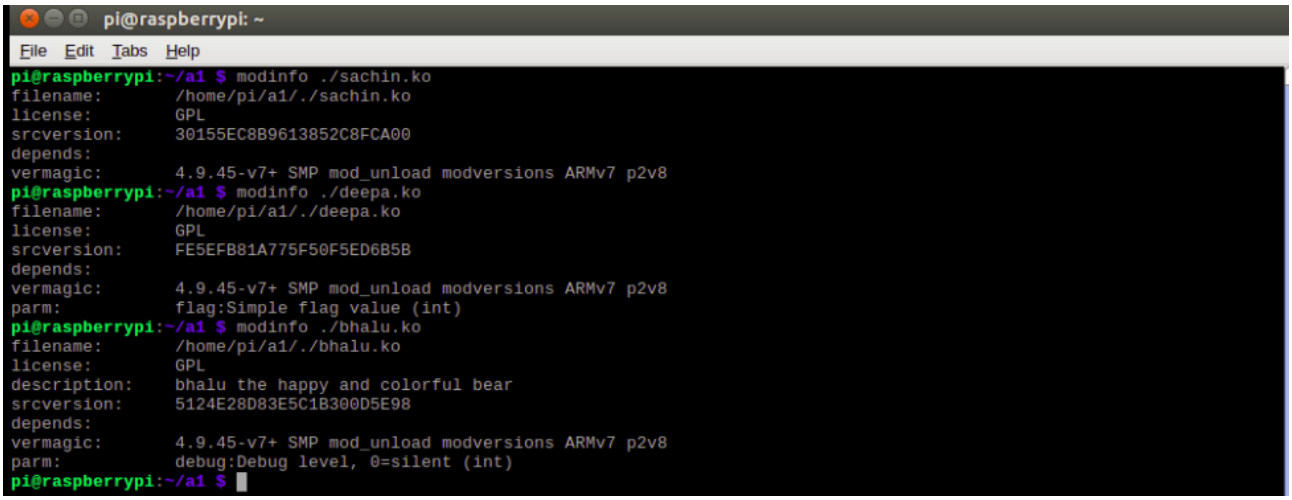


If the same command is run on an x86 kernel object,the result is:



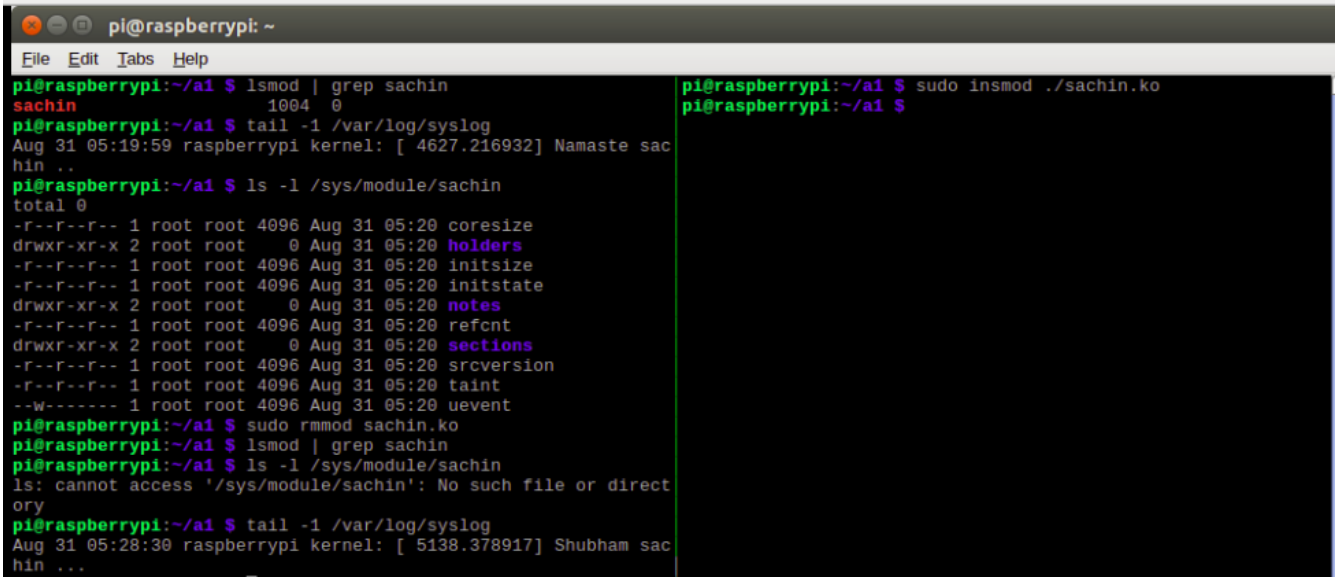
- Print module info for both modules and point out the parameter details

A parameter is a runtime option defined for a module whereas an attribute is defined for a device.Therefore,only paramters are displayed in modinfo.
Sachin does not have any paramters therefore,this is empty.Deepa has a paramter called flag which is an integer and has the description simple flag value.
Bhalu has a parameter called debug also an integer bearing the description debug level,0=silent.

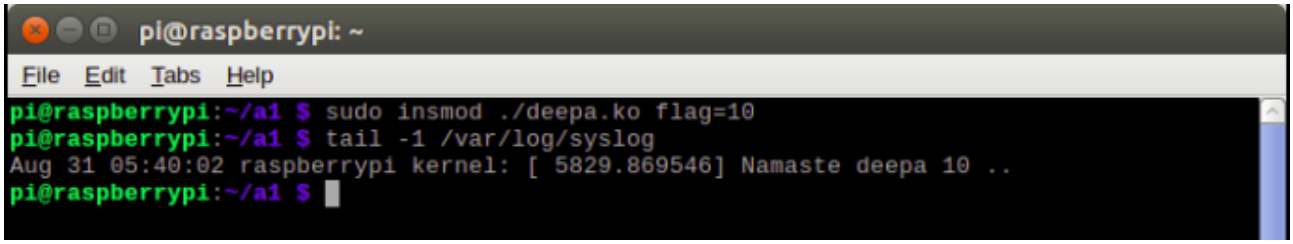


- Load sachin module and show the registration of the module in sysfs

The registration can be seen by doing an ls -l on sys/module/sachin.After the module is loaded,an entire directory whose name is same as the module name is created which represents the hierarchy for that module.After removing the module, it is no longer present under sys/module.

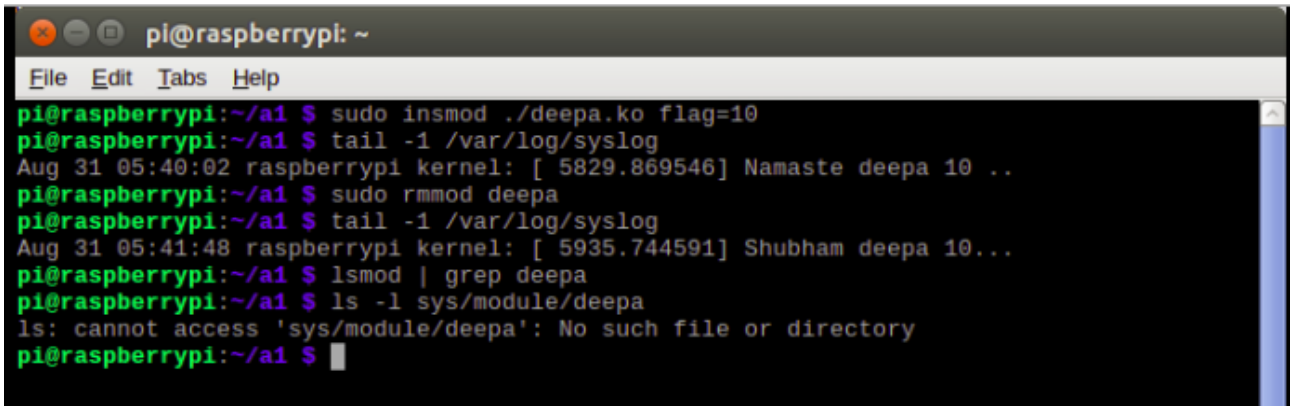


- Load deepa module and show its flag values in syslog



Deepa is loaded with a flag value of 10 which is reflected in syslog

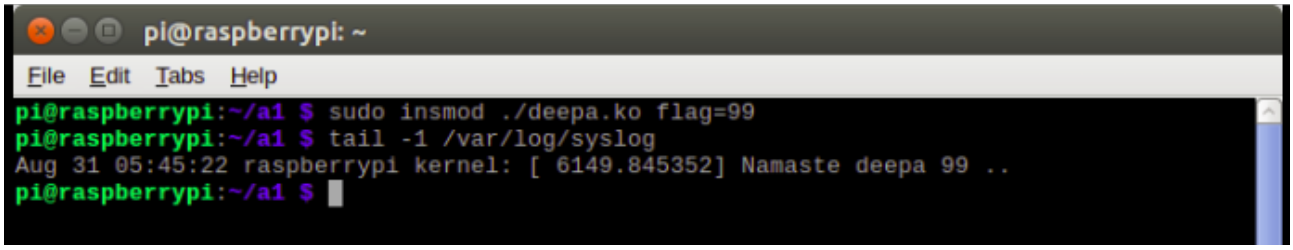
- Unload deepa module and show that it is no longer loaded



Deepa is no longer present in sys/module or proc/modules since it has been removed.

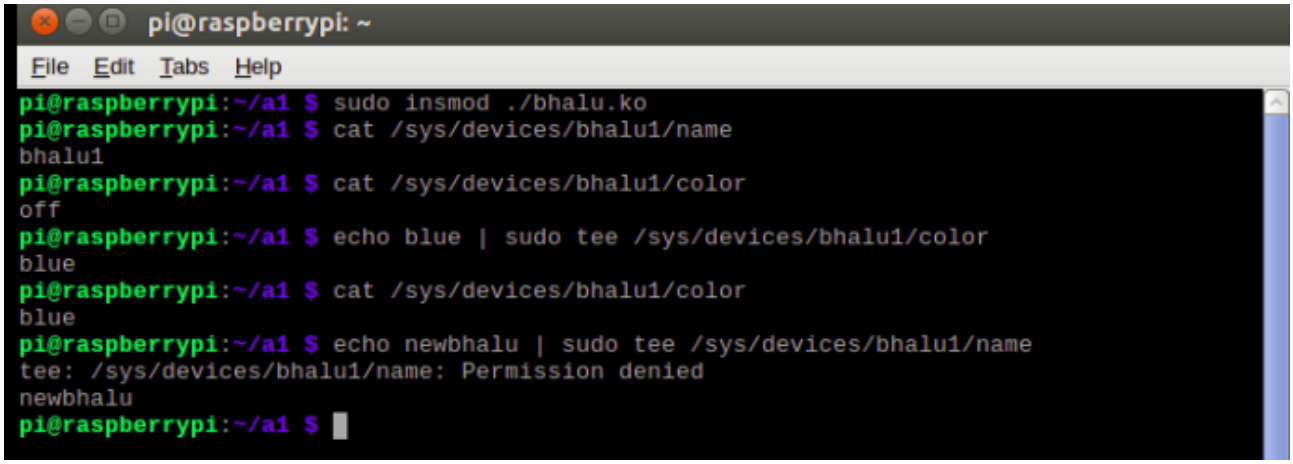
- Reload deepa with a different parameter and show its log entries

This time flag is set to 99.



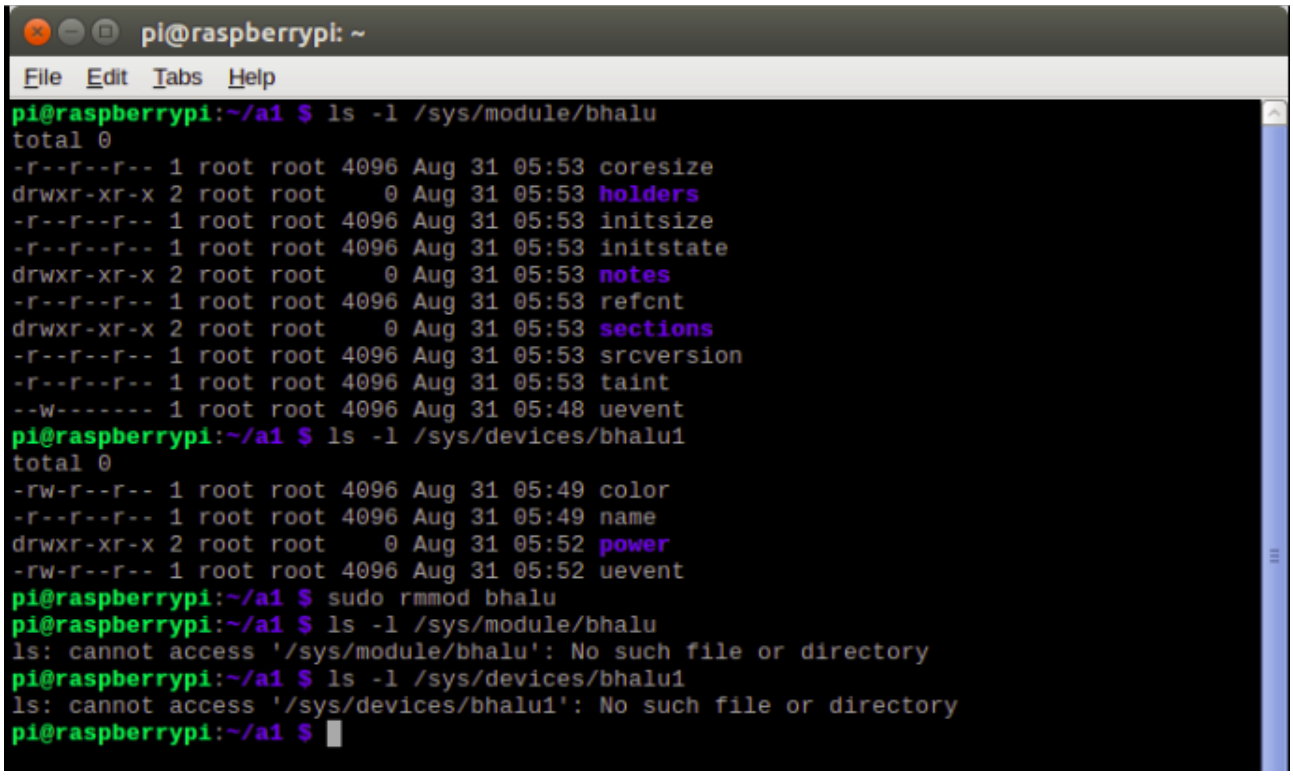
- Load bhalu module and show read/write ops on its attributes in sysfs

Color can be written to whereas name is readonly and this can be seen while trying to write to name



- What happens in /sys/module/ and /sys/devices/ directories when you load and unload bhalu module.

A directory called bhalu is created in sys/module and a directory called bhalu1(device name) is created in sys/devices.These directories are deleted when bhalu is unloaded.The device attributes are present under bhalu1 whereas the module details under bhalu.

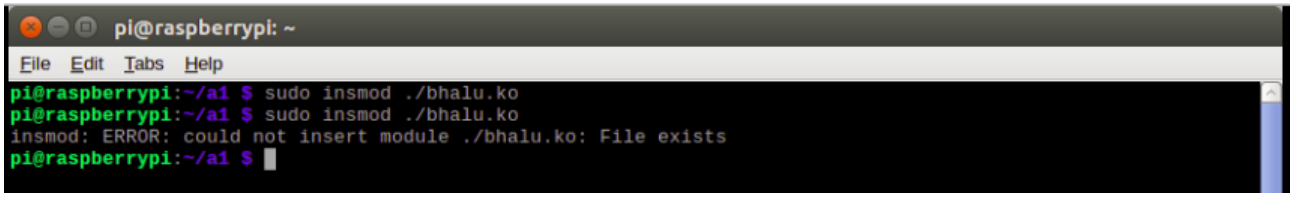


- Module bhalu registers a device but not a driver. Observe what happens in devices/ and drivers/ of /sys/bus/platform/ when you load and unload bhalu module.

There is no change in both devices and drivers

- What happens if you load bhalu a second time?

The loading fails because the module already exists in the kernel.

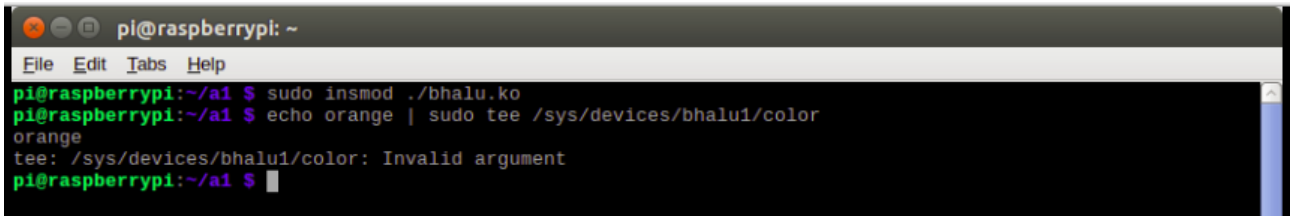


- What are the valid color values for bhalu?

The valid colors are off,red,green,blue,white which internally translate to 0,1,2,3,4.

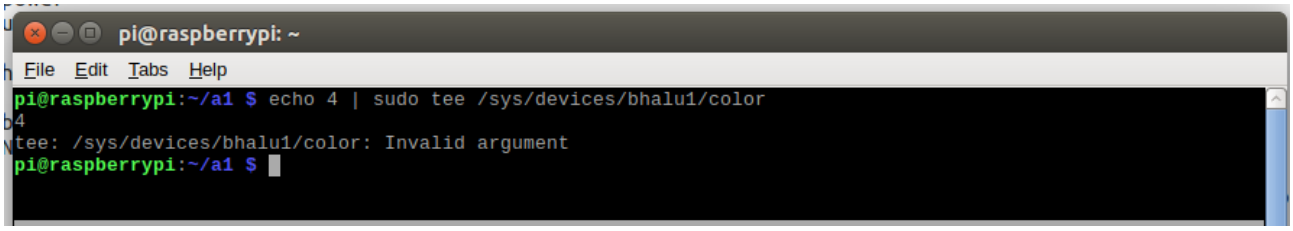
- What happens if you use an invalid color value?

Tee throws an invalid argument error.This is because in the setter function for color in bhalu,the for loop iterates through all the valid colors comparing them with the entered color at each step.If no match is found,it returns EINVAL which is the errno for invalid argument.



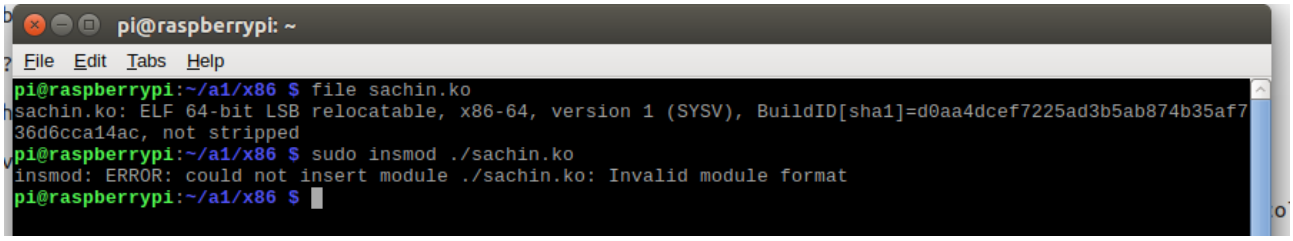
- What happens if you use a number to set color?

We still get the same invalid argument error.This is because although the strings are internally mapped to numbers using #define,the reverse mapping is not present i.e numbers are not translated to the strings they represent.Therefore,the strcmp fails and it returns EINVAL.



- What happens if you try to load a module compiled for x86 kernel on Raspberry Pi?

The load operation fails with an invalid module format error.Insmod checks whether the kernel that the module was compiled for and the kernel that it is ciurrently being loaded on are the same.If not,it throws this error.The kernel that the module has been compiled for can be verified using modinfo.Current kernel is obtained using uname -r.



Observations from Readme

- insmod uses file paths while lsmod and rmmod uses module name. Why?

This is because before the file is loaded the kernel is not aware of the module name--hence the path is to be specified.Whereas once the module is loaded,it is given a unique name by the kernel.To unload it,this name can be used.

- bhalu combines both driver code and device spec into a single module. Can you guess which lines deal with driver and which deal with device?

Driver and device have separate structures referring to them.The setter and getter functions for the attributes,the register and release device all are part of the device code.Whereas module init and exit,bhalu_init and bhalu_exit are part of the driver code.

- In /sys/devices/bhalu1/ you will notice attributes other than the ones you defined. Read Documentation/driver-model/overview.txt for clues.

The other attributes present apart from color are name,power and uevent.These are created by the kernel and are part of the global layer in the driver model.The name file contains the name assigned by the kernel to the module and power indicates the current power state.

Time taken:Around 22 hrs spread out over 6 days