|  |
| --- |
| **Project 1: Linux and Pi**  **CSE 330 - Operating Systems** |
| **Instructor: Ming Zhao**  **TAs: Eugene Kuznetsov, Sungho Hong, Yitao (Tom) Chen** |

[1. Install Raspbian [20pts]](#_pu75zqrvsk9h)

[2. Install a New Kernel [20pts]](#_yu59y95jn47s)

[3. Add a Kernel Module [30pts]](#_523vknrfts5p)

[4. Adding a System Call [30pts]](#_6yp16wheuj84)

[5. Submission Requirements](#_l7nsmlr1u8m6)

The first project is a warm-up exercise which helps you become familiar with Linux and Raspberry Pi that you will use extensively for your kernel development throughout this semester. In this exercise, you will complete the following tasks:

1. Install a Linux OS (Raspbian) on the Raspberry Pi
2. Install the latest kernel on this Linux system
3. Add a kernel module
4. Add a system callto the kernel.

**Note:** The references included in the following description contain only *generic* instructions for completing the tasks of this lab; they are by no means exact step-by-step instructions. You are expected to learn the environments and tools described below on your own in this lab so that you are prepared for the future projects. To get more help, attend the classes and recitations, ask the instructor and TAs, and use your favorite search engine (there are lots of useful materials on the web.)

To help you get over the learning curve quickly, we have also created a video tutorial on Youtube: <https://youtu.be/BezVqxqzXAc>

Before you start, you will need a Linux environment on your own computer. If your OS is not Linux, you can install virtual machine (VM) software (e.g., VirtualBox) and create a Linux VM. There are many Linux-based OSes; we recommend you to use a Debian-based GNU/Linux OSes such as Ubuntu and Raspbian, because of their ease of use. The instructions below are for Debian.

# 1. Install Raspbian [20pts]

* Download [*Etcher*](https://www.balena.io/etcher/) (<https://www.balena.io/etcher/>), a tool for flashing OS images to SD cards, to your own host computer (we will call it the **host** hereinafter)
* Download the [*Raspbian*](https://www.raspberrypi.org/downloads/) image (version 4.14.93), a Linux-based OS for Raspberry Pi (<https://www.raspberrypi.org/downloads/raspbian/>) to your host
* Flash the Raspbian image to a microSD card using Etcher
* Plug the card to the Raspberry Pi (we will call it the **Pi** hereinafter) and power it up

If your Raspbian boots up, congrats! now you have a working Linux environment on the Pi.

There are several different ways to access the Linux on the Pi.

**Directly accessing the Pi**

You can attach a monitor (via HDMI), mouse (via USB), and keyboard (via USB) to your Pi in order to access it. If you do not have these peripherals available, you can follow Option 2.

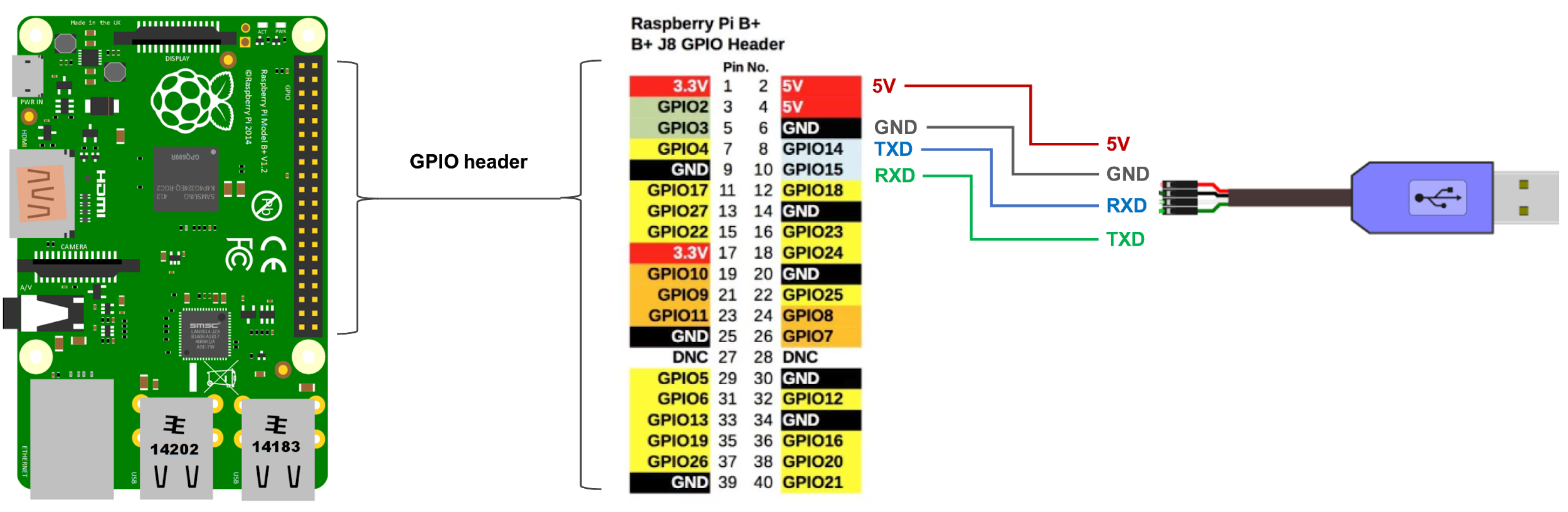
**Connecting through serial console**

You can use the provided USB serial console to connect to the Pi’s default serial port (serial 0) from you own computer. You need to do this because it will be used in the demos for all Projects

Since the PI uses the default serial port (serial0) to run bluetooth, we need to override the getty (console) service to the default serial port. Modify the file */boot/config.txt* on your Pi, and add the following line:

|  |
| --- |
| dtoverlay=pi3-miniuart-bt |

Connect the black, white, and green leads to the GPIO pins (You don’t need to use the red lead which is used for powering via cable)



Based on the OS that your host is using, you need to install a specific driver for your USB serial console. Download and install the driver on your host following the instructions here: https://www.pololu.com/docs/0J7/all#1

**Connecting via network**

You can use the Ethernet port for wired connection; alternatively, you can set up WiFi.

Use the following commands to check and enable the WiFi.

|  |
| --- |
| sudo iwlist wlan0 scan // Scan all available WIFIs  sudo raspi-config // Follow screen prompts to add WIFI  // Alternatively, add the following to the config file  /etc/wpa\_supplicant/wpa\_supplicant.conf  network={  ssid="NETGEAR" // WIFI name  psk="01234" // WIFI password  }  // Restart the Raspberry PI  sudo reboot |

To access the Pi more conveniently, you can setup an SSH server on the Pi, and then you will be able to connect to the Pi through SSH, instead of the serial port.

Enable the SSH on your Pi by following this command below.

|  |
| --- |
| *s*udo systemctl enable ssh sudo systemctl start ssh |

After this you will be able to connect to Pi with your host by communicating through SSH. You can find out the IP address of your Pi using the *ifconfig* command.

Then, use the following command in your laptop or workstation to SSH to Your Pi

|  |
| --- |
| ssh pi@10.218.109.78 // Pi’s IP |

# 2. Install a New Kernel [20pts]

We will build and install the updated kernel (version 4.14.93) on the Pi. Because kernel compilation is time consuming, we will build the kernel on our host or in our VM by cross-compiling it.

Use the following commands to download the tools from Github.

|  |
| --- |
| sudo apt-get install gcc-arm\* bc git bison flex libssl-dev |

Download the kernel source from git, and choose the correct version which is 4.14.93 9

|  |
| --- |
| git clone https://github.com/ychen404/linux.git |

Use the following commands to build the new kernel. When compiling the kernel, we recommend you to use *- j N* command which will enable parallel compilation on multiprocessor.

You will notice that you are specifying the type of architecture you are using for Raspberry PI.

|  |
| --- |
| // i.e. /home/sungho/linux  export KERNEL=kernel7  make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- bcm2709\_defconfig  make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- zImage modules dtbs -j 16 |

Having built the kernel and kernel modules, now you can copy them to the Pi. A simple way to do this is using the SD card. The device name of your card (*/dev/sd\**) may vary; you can use lsblk command to verify.

|  |
| --- |
| // i.e. /home/sungho/linux  mkdir /mnt/boot  mkdir /mnt/root  sudo mount /dev/sdd1 /mnt/boot // boot partition for Raspbian  sudo mount /dev/sdd2 /mnt/root // root partition for Raspbian |

Install the modules on SD card’s root partition.

|  |
| --- |
| // i.e. /home/sungho/linux  sudo make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- INSTALL\_MOD\_PATH=/mnt/root modules\_install |

Finally, copy the kernel image and Device Tree blobs onto the SD card’s boot partition.

|  |
| --- |
| // i.e. /home/sungho/linux  sudo cp arch/arm/boot/zImage /mnt/boot/kernel7.img  sudo cp arch/arm/boot/dts/\*.dtb /mnt/boot/  sudo cp arch/arm/boot/dts/overlays/\*.dtb\* /mnt/boot/overlays/  sudo cp arch/arm/boot/dts/overlays/README /mnt/boot/overlays/ |

Unmount the SD card

|  |
| --- |
| sudo umount /mnt/boot  sudo umount /mnt/root |

Finally, plug the SD card back to Pi, and power it up. Use the command *uname -a*to check your Linux version. If you see 4.14.93, congratulations! Now you have the latest kernel running on your Pi!

Now we can start our kernel hacking journey. We will start with something simple; in the next two steps, we will add a kernel module and a system call to our kernel on the Pi. For faster development and testing, we recommend to first complete these steps on the kernel of your host computer (physical or virtual) and then port them to the kernel on the device. Follow the instructions in the warm-up project to create this Linux environment on your host.

# 3. Add a Kernel Module [30pts]

Now let’s create a new kernel module to our new kernel. As what we will discuss in class, modules are an important means for extending the functionality of a monolithic OS like Linux. In this step, you will implement a new module to add a function to your kernel. This function should print out the names of our new kernel developer, i.e., you, to the kernel log when the module is loaded.

You can follow this simple hello world example to create your module. We will do it on the host, since it is much faster and already has all the tools installed.

|  |
| --- |
| // Create your own directory, for example, /home/sungho/test\_module/hello.c  #include <linux/module.h>  #include <linux/kernel.h>  int hello\_init(void)  {  pr\_alert("Hello World!\n");  return 0;  }  void hello\_exit(void)  {  pr\_alert("Goodbye World!\n");  }  // We need at least two functions in a module, init and exit. Init function is called when the  // module is loaded (insmod) into the kernel, and exit function is called when the module is  // removed from the kernel (rmmod).  module\_init(hello\_init); // defines the hello\_init to be called at module loading time  module\_exit(hello\_exit); // defines the hello\_exit to be called at module unload time |

Create a makefile for compiling the kernel module.

|  |
| --- |
| export CCPREFIX=/home/username/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin/arm-linux-gnueabihf- // The location of the libraries used for cross-compilation  sysroot := "/home/sungho/linux" // The location of linux source code  obj-m += hello.o  all:  make ARCH=arm CROSS\_COMPILE=${CCPREFIX} -C $(sysroot) M=$(PWD) modules  clean:  make -C $(sysroot) M=$(PWD) clean |

After a successful build, you will see the *hello.ko* file generated.

Copy the kernel module (hello.ko) to the Pi.

On the Pi, install the kernel module using *insmod*

|  |
| --- |
| pi@raspberrypi:~ $ sudo insmod hello.ko |

Check the hello world output using *dmesg* command

|  |
| --- |
| dmesg  [19430.106244] Hello World! |

Check the goodbye world output after removing the module using *rmmod*

|  |
| --- |
| sudo rmmod hello  dmesg  [19527.837037] Goodbye World! |

Congrats! Now you know how to create kernel modules for the Pi.

# 4. Adding a System Call [30pts]

As we will also discuss in class, system calls are the main interface for user-space software to interact with the kernel. In this step, you will implement a new system call that simply prints out the names of our new kernel developer, i.e., you, in kernel log and a user-space program that tests this system call.

Follow the simple hello world example below to create the new syscall on your host.

To add a new call to the system, we need to modify the syscall table file,

|  |
| --- |
| // Add your system call number, for example, in /home/sungho/linux/arch/arm/tools/syscall.tbl,  // we specify our syscall as call 398  393 common pwritev2 sys\_pwritev2  394 common pkey\_mprotect sys\_pkey\_mprotect  395 common pkey\_alloc sys\_pkey\_alloc  396 common pkey\_free sys\_pkey\_free  397 common statx sys\_statx  398 common sayhello sys\_sayhello // our new syscall |

modify syscalls.h,

|  |
| --- |
| // Define the system call, for example, in /home/sungho/linux/include/linux/syscalls.h, we add // the following line  asmlinkage long sys\_sayhello(void); |

and add the target directory of your syscall in kernel Makefile

|  |
| --- |
| // Modify the kernel Makefile, i.e. /home/sungho/linux/Makefile  // Modify the line from  core-y += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/  // to  core-y += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ mycall/ |

Create the source code, header file, and make file for our new syscall. This syscall prints a message in the kernel log.

|  |
| --- |
| // i.e. /home/sungho/linux  mkdir mycall |

|  |
| --- |
| // i.e. /home/sungho/linux/mycall  // Create mycall.h  asmlinkage long sys\_sayhello(void);  // mycall.c  #include<linux/kernel.h>  #include<linux/init.h>  #include<linux/sched.h>  #include<linux/syscalls.h>  #include "mycall.h"  asmlinkage long sys\_sayhello(void)  {  printk("Hello world\n");  return 0;  }  // create Makefile  obj-y := mycall.o |

After adding the syscall, you will need to recompile the kernel.

Create a user-space program to invoke our new syscall

|  |
| --- |
| #include <stdio.h>  #include <linux/kernel.h>  #include <sys/syscall.h>  #include <unistd.h>  int main()  {  printf("Say hello");  long int ret\_status = syscall(398); // the number of the syscall in the system-call table  if(ret\_status == 0)  printf("System call executed correctly. Use dmesg to check the syscall output.\n");  else  printf("System call did not execute as expected\n");  return 0;  } |

You should be able to see Hello world in the dmesg.

|  |
| --- |
| dmesg  [ 133.370174] Hello world |

Congrats! Now you have your own syscall.

# 5. Submission Requirements

**1. Source code submission on Canvas**

Submit a zip file that contains the source-code of the kernel module and the syscall by **Feb 6th 11:59:59pm**. Only one submission each team.

The zip file should include the following contents:

* A plain text *README* file that lists the full names of all members of your team;
* A folder named “*mod*” that includes only the Makefile and source code of your module.
* A folder named “*syscall*” includes only the system call source code, test program source code, and the changes that you made to the kernel files. You can record the modifications in your kernel using the command “*git diff > patch*”. (“git diff” will show the differences of your modified kernel compared with the original one).

**Notes:**

* Do not submit any other source code
* Do not submit any binary
* Put everything in a single zip file named by your full names
* Late submissions will not be accepted.

Failure to follow the above instructions will cause penalty to your grade.

**2. Live demo to the TAs**

In the recitation session on **Feb 4th**, **Feb 5th**, or **Feb 6th**, you need to provide a live demo of project to the TAs, using your Pi and laptop (connected through serial port). Make sure to bring all the equipment that you need and be prepared for the demo. Each team will have only five minutes to complete the demo.

TAs expect to see outputs as follows:

|  |
| --- |
| // demonstrate your updated kernel version  uname -a  **Linux raspberrypi 4.14.93-v7+ #1 SMP Sun Dec 23 14:55:34 KST 2018 armv7l GNU/Linux**  // show your kernel module that prints your team member names in kernel log  insmod kernel\_module.ko  dmesg  **Eugene Kuznetsov, Sungho Hong, Yitao (Tom) Chen**  // show your syscall that prints your team member names in kernel log  ./say\_hello  dmesg  **Eugene Kuznetsov, Sungho Hong, Yitao (Tom) Chen** |