Pre-requisites:

1. VirtualBox (optional).
2. Installation of Ubuntu 16.04 in your VirtualBox.
3. Time and Patience.

Part 1: Obtaining Source of kernel

For this part, if you wish, you can follow along with this guide.

1. Open Ubuntu in VirtualBox
2. Find **Software & Updates** in your Applications
3. Open **Software & Updates**
4. In the tab, **Ubuntu Software** make sure the box next to **Source Code** is checked off.
    1. If this isn't checked off before you run the following command in step 6, you'll get the error:

        ```
        Reading package lists... Done
        E: You must put some 'source' URIs in your sources.list
        ```

    2. A window will pop up telling you to reload the information about available software. Select reload.
    3. If it *is* checked off, skip to step 6.
5. Exit out of **Software & Updates**.
6. Open the terminal/command line.
7. To download the source of your currently running kernel, enter the command:

    ```
    apt-get source linux-image-$(uname -r)
    ```

    1. This will take a while. Break #1.

Congratulations you have just obtained the source of your kernel.

Part 2: Adding Your System Call

Now we'll edit the syscall table to include your custom syscall.

1. Open your File directory.
2. In **Home**, you'll see a folder of the kernel source.
    1. In this tutorial, the folder you'll see will be **linux-hwe-4.13.0**

3. From **Home**, navigate to **linux-hwe-4.13.0/arch/x86/entry/syscalls** folder to find the syscall table files
    1. Here you'll find the syscall tables for 32-bit processors (syscall_32.tbl) and 64-bit processors (syscall_64.tbl). To add your custom syscall to the syscall tables, you'll have to edit both syscall_32.tbl and syscall_64.tbl.

Editing syscall_32.tbl

1. Open **syscall_32.tbl** located in **linux-hwe-4.13.0/arch/x86/entry/syscalls**
2. Mentioned at the top of the table, the format is:

    ```
    <number> <abi> <name> <entry point> <compat entry point>
    ```

3. Scroll to the bottom of the table and insert your custom syscall.

    In the following example, the final number in the syscall table was 384, so I incremented my syscall by one and followed the format to add my syscall, `supworld` to the syscall table.

    ```
    385    i386    supworld        sys_supworld
    ```

4. Save the changes you made to the **syscall_32.tbl** file

Editing syscall_64.tbl

1. Open **syscall_64.tbl** located in **linux-hwe-4.13.0/arch/x86/entry/syscalls**
2. Mentioned at the top of the table, the format is:

    ```
    <number> <abi> <name> <entry point>
    ```

3. However, unlike **syscall_32.tbl** , scrolling to the bottom of the table gets you to the x32-specific system call numbers. Instead, scroll to the part of the table where you find the last "common" abi, right before the x32-specific system call numbers.

    Similar to the **syscall_32.tbl**, we increment the number by one. In this case, however, we add "common" as the abi since the syscall is not special in some way.

    ```
    333    common   supworld      sys_supworld
    ```

4. Save the changes you made to the **syscall_64.tbl** file.

Editing the syscall headers

After the syscall tables have been edited, we now must edit the syscall header file.

1. Navigate to **linux-hwe-4.13.0/include/linux** and find the **syscalls.h** file.

2. Open **syscalls.h**.
3. Scroll to the bottom and find the line `#endif`. Here, you'll add the function of your custom syscall.

   For example, with the custom syscall, `supworld` this is what the addition after `#endif` would look like:

   ```
   #endif


   asmlinkage int sys_supworld(void);
   ```

4. Save the changes made to the **syscalls.h** file.

Creating the system call

We've edited the syscall tables and the syscall header file, now it's time to create the system call C file (if you haven't done so).

1. Using a text editor of your choice, create and save your syscall as a C file.
2. Make sure the C file is the same name as your system call. For example, my custom syscall, would be saved as: `supworld.c`.

1. Make sure the C file is saved in the **kernel** folder, located in your kernel source file. For example, `supworld.c` is saved in **linux-hwe-4.13.0/kernel**.

Editing the Makefile(s)

Now it's time to edit the Makefile in the kernel to include your custom syscall.

1. Navigate to **~/linux-hwe-4.13.0/kernel** and find the **Makefile**.
2. Open the **Makefile**. Upon opening the **Makefile**, you'll find an obj-y list.
3. Add your syscall to the obj-y list as an object file, with the suffix, .o. Make sure to add your syscall after `sys.o` in the list.
   1. For example, my custom syscall would be saved as `supworld.o`, and overall the obj-y list would look like so:

      ```
      obj-y     = fork.o exec_domain.o panic.o \
              cpu.o exit.o softirq.o resource.o \
              sysctl.o sysctl_binary.o capability.o ptrace.o user.o \
              signal.o sys.o supworld.o kmod.o workqueue.o pid.o task_work.o
      \
              extable.o params.o \
              kthread.o sys_ni.o nsproxy.o \
              notifier.o ksysfs.o cred.o reboot.o \
              async.o range.o smpboot.o ucount.o
      ```

4. Save the changes made to the **Makefile**.
5. Now, navigate to **~/linux-hwe-4.13.0** and find the **Makefile**.
6. Open the **Makefile**. At the top you'll see the following:

   ```
   VERSION = 4
   PATCHLEVEL = 13
   SUBLEVEL = 13
   EXTRAVERSION =
   NAME = Fearless Coyote
   ```

7. Have EXTRAVERSION equal `.syscall` so now it looks like this:

   ```
   VERSION = 4
   PATCHLEVEL = 13
   SUBLEVEL = 13
   EXTRAVERSION = .syscall
   NAME = Fearless Coyote
   ```

8. Save the changes you made to the **Makefile**.

Congratulations you have just added your custom syscall to your kernel. Now it's time to build the kernel.

Part 3: Building the kernel

Now it's time to build the kernel. This is where time and patience comes in.

1. Now run the following command to download the packages needed to build the kernel if you haven't built one before:

   `sudo apt-get build-dep linux-image-$(uname -r)`
2. If you're building your kernel in your Virtual Machine, then run the following command in your terminal to change the number of processors devoted to your Virtual Machine:

   `export CONCURRENCY_LEVEL=2`

   In this example, I'm allotting 2 processors to run the kernel.
3. In your Ubuntu terminal, change the directory to the root of the kernel source. In this example, the root of the kernel source would be: `linux-hwe-4.13.0`.
   1. If you're unsure of the root of your kernel source, run the command `ls` in your terminal. Here you'll see your kernel source in blue. It should not have the suffix, `.gz`, or `.dsc`.
4. Now the moment of truth. Building the kernel.
5. Run the command `fakeroot debian/rules clean` .

6. After the command has finished, run the command:

   `fakeroot debian/rules binary-headers binary-generic binary-perarch`

   1. This will take a **long** time, so make sure that your Virtual Machine or whatever you're using to build the kernel, will not go on standby while building the kernel.

7. To check that the build was successful, change the directory to the directory above the linux root and lists the files within the system. Essentially, after the build finishes, run the commands:

   `cd ..` and then `ls`

   1. `cd ..` changes the directory and `ls` lists the file in the directory.
   2. You should be able to find the following .deb files:

      ```
      linux-headers-4.13.0-37_4.13.0-37.42~16.04.1_all.deb
      linux-headers-4.13.0-37-generic_4.13.0-37.42~16.04.1_amd64.deb
      linux-image-4.13.0-37-generic_4.13.0-37.42~16.04.1_amd64.deb
      ```

Congratulations, you've now finished building your new kernel.

Part 4: Testing your new kernel

Now to test your new kernel, you must install the previously mentioned packages on your build system.

1. To install the packages, use the following command: `sudo dpkg -i linux*4.13.0-37.42~16.04.1*.deb`

   Most likely, you'll get the following error:

   ```
   dpkg: error processing package linux-cloud-tools-4.13.0-37-generic (--
   install): dependency problems - leaving unconfigured
   Errors were encountered while processing:
    linux-hwe-cloud-tools-4.13.0-37
    linux-hwe-tools-4.13.0-37
    linux-tools-4.13.0-37-generic
    linux-cloud-tools-4.13.0-37-generic
   ```

2. To fix the error use the following commands:
   1. `sudo apt-get install -- install-recommends linux-generic-hwe-16.04`
   2. `sudo apt-get update`
   3. `sudo apt-get -f install`
3. Run the command from step 1 again.
4. Once complete, run the command `sudo reboot` to reboot the machine.

1. If using VirtualBox, hold down `shift` when you see the VirtubalBox splash screen to open the GRUB boot loader.
2. Go to Advanced Options
3. Select your custom kernel to use it

Congratulations, you have successfully created your own custom kernel.