Building your own System Call

WARNING: Installing a new kernel may render your system unusable or unstable. If you proceed with the installation using the instructions below, make sure you back up any important data you have to an external hard drive.

Objectives

- Defining and creating your own system call.
- Testing the newly created system call.
- Compare how system calls have been added across linux kernel versions 3.x and 4.x as for e.g., 3.13.x, 3.19.x, 4.2.x, 4.7.x, 4.14.x and 4.20.x

- Recall from last experiment: sudo -s to execute root commands

System Call

- Code running under kernel mode
    - Interact with hardwares or other kernel modules
    - With highest privilege
- To add a system call under linux, we need to recompile the kernel and install the new kernel we compiled
- Surprisingly easy!

Setting up the environment

You need:

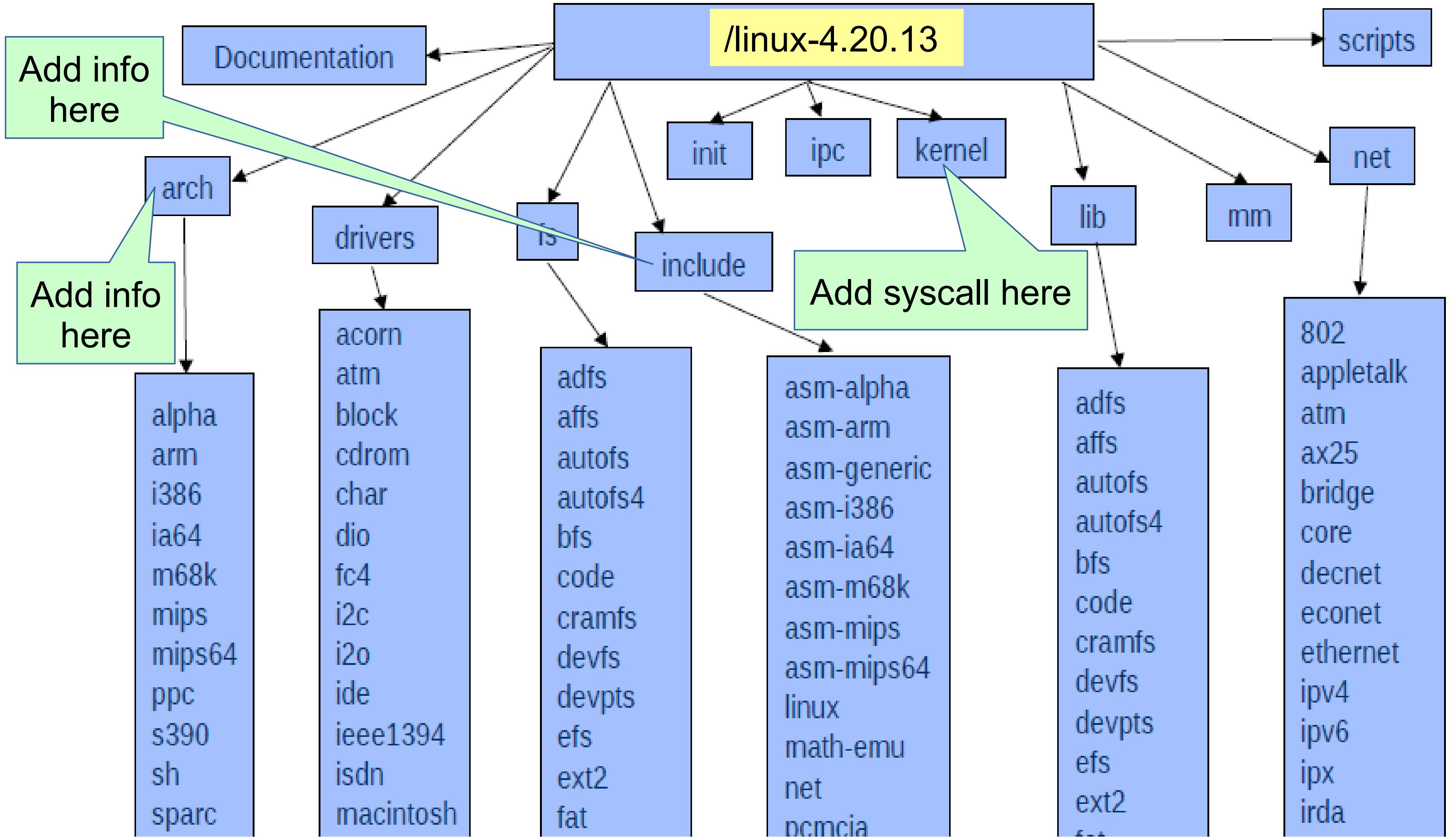
- Compiler tools
- Kernel source code

Run the following command to download

- apt-get build-dep linux // builds dependency tree before getting source
- apt-get source linux // gets current kernel source – git version

Or use the wget option discussed during kernel build exercise

View of the linux source tree



Linux kernel 4.20.13 release date Feb 27, 2019

Remember

If you are on a 32-bit system you will need to change 'syscall_32.tbl'.

For 64-bit, change 'syscall_64.tbl'

Alternatively, it is a usual practice to update both the files.

Add system call entry

- cd /usr/src/linux-4.20.13

- Edit arch/x86/entry/syscalls/syscall_32.tbl to add your own system call entry

<last-number+1> i386 <yoursyscallname> sys_<yoursyscallname>
(Alternative to the use of i386 could be x32 or i586)

Find last-number for kernel 4.20.13 from the last entry in syscall_32.tbl. Let it be integer K. Therefore, your entry will be

```
K+1    i386       my_call             sys_my_call
```

- Edit arch/x86/entry/syscalls/syscall_64.tbl to add your own system call entry

Find last-number for kernel 4.20.13 from the last entry in syscall_64.tbl. Let it be integer M. Therefore, your entry will be

```
M+1     64   my_call             sys_my_call
```

- Edit include/linux/syscalls.h to declare your system call function:

```
asmlinkage int sys_my_call(int num1, int num2);
```

Add system call entry ...cont'd

- Create your directory(mkdir) kernel/my_call/ and create two files sys_my_call.c and Makefile inside to define a new system call function.
- sys_my_call.c in kernel/my_call/ should look like

```
#include<linux/kernel.h>
#include<linux/syscalls.h>
asmlinkage int sys_my_call(int num1, int num2){
return num1+num2;
}
```

- Makefile should look like

```
obj-y := sys_my_call.o
```

- Edit kernel/Makefile to include kernel/my_call/ directory
  - Right below obj-y +=power/, to add the following line

```
obj-y += my_call/
```

File Checklist

- arch/x86/syscalls/syscall_32.tbl
- arch/x86/syscalls/syscall_64.tbl
- include/linux/syscalls.h
- kernel/my_call/sys_my_call.c
- kernel/my_call/Makefile
- kernel/Makefile

Preparing to build kernel

- Following commands will only be necessary if the corresponding system utitilities / libraries etc are not available on your system. So, please check and proceed.

```
# sudo apt-get install gcc

# sudo apt-get install libcurses5-dev

# sudo apt-get install bison

# sudo apt-get install flex

# sudo apt-get install libssl-dev

# sudo apt-get install libelf-dev

# sudo apt-get update

# sudo apt-get upgrade
```

Compile/Build the kernel

- cd /usr/src/linux-4.20.13
- make clean # clean compiled object
- make mrproper # clean config file
- make menuconfig # generate config file
- make localmodconfig # generate module config
- make -j2 # compile kernel

  or make -j4 # if you have a quadcore CPU
- make modules_install install # kernel installation
- update-grub # Update grub start menu - optional

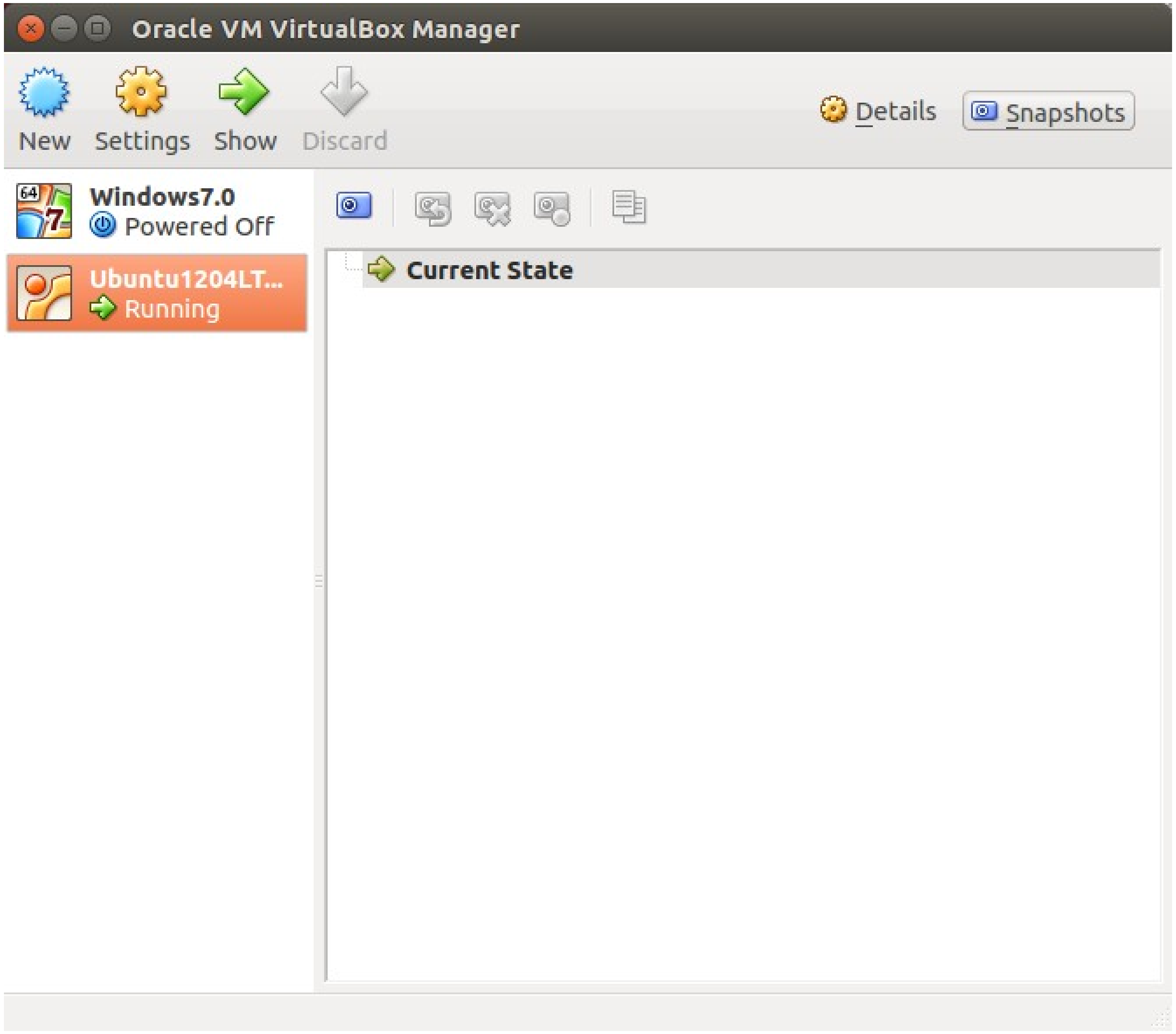
Reboot to the new kernel

- Reboot Ubuntu VM
- This new kernel has your newly defined system call in it for your use now onwards.

The argument -j2 is given to allow a parallel build using 2 processes. The optimal number of processes to use depends on the compute resource available. There is a lot of available parallelism in the build procedure, so using 16 or even 32 processes may speed things up considerably. Note however that the build happens in shared (not distributed) memory. So specifying more processes than there are processors in a shared memory node is generally not helpful (although the presence of hyperthreading or SMT on a node may provide an advantage to specifying twice the number of processors).

It is useful to redirect the output from make to a file for later reference. This file contains the exact commands that were issued to compile each file and the final command which links everything into an executable file. Relevant information from this file should be included when posting a bug report concerning a build failure.

Check which kernel to boot OR In case your reboot fails?



Reset the Ubuntu VM and go to the Advanced option to select the suitable kernel to boot.

Using & Testing the new system call

- Create a directory yourname/ (or in home dir) and create two files
  - my_call.h
  - testsyscall.c
- Header file my_call.h

```
#include <linux/unistd.h>
int my_call(int num1, int num2) {          //your system call name as given before
return syscall(syscallNO, num1, num2);    //syscallNO = K+1
}
```

- User code file testsyscall.c

```
#include <stdio.h>
#include "my_call.h"
int main() {
printf("The kernel system call returns %d\n", my_call(10, 20));
}
```

Useful links

- http://www.gnu.org/software/make/manual/make.html