

## Name of the Assignment:

Bonus Lab Assignment: Extending KVM

## Team Member:

Vishwanath Iyer [Individually done]

## Introduction:

The main purpose of the lab is to demonstrate the concept of VM Exits by instrumenting the Linux KVM Hypervisor and modifying the Linux Kernel Source code (in my case v4.5.2) so as to record whenever a VM takes exit. In this project, counters are added to each exit type summing up to around 45 exits. The owner of the hypervisor can see the number of exits at any point of time using the “**dmesg**” command.

## Environment Setup:

This work involves the following tools and software to achieve our goal of recording the VM Exits. The environment use the concept of nested virtualization where KVM is installed on top of the VM Ware virtual machine, instead of using a dual boot installation. The steps are divided into two stages - Installation on Host Machine, Installation in Virtual Machine:-

### Installation on Host Machine:

- VMWare Workstation Player 12
- Ubuntu LTS v.14.04 ISO Image 32-bit

### Installation in Virtual Machine:

- QEMU-KVM
- Latest Linux Kernel Source Code (v4.5.2 in our case)
- Virtual Machine Manager
- Ubuntu v.16.04

## Steps:

### Setting up VMWare and creating virtual machine instance

1. To Install the VMWare work station player go to the below website and download the VMware Workstation 12 Player for Windows/Linux 64-bit depending on the operating system used.  
<https://www.vmware.com/products/player/playerpro-evaluation>
2. Before installing the VMWare virtual machine, your physical device should fulfill the following system requirements :
  - 64-bit x86 Intel Core 2 Duo Processor or equivalent, AMD Athlon™ 64 FX Dual Core Processor or equivalent [1]
  - 1.3GHz or faster core speed
  - 2GB RAM minimum/ 4GB RAM recommended

- Workstation 12 Player installation:
  - 300MB of available disk space for the application. Additional hard disk space required for each virtual machine. Please refer to vendors recommended disk space for specific guest operating systems.
- 3. Now download Ubuntu ISO image from the following website. You can use any Linux distro of your choice. This image is used to create an instance of Virtual machine in VM Ware player:  
<http://www.ubuntu.com/download>
- 4. Now install the ISO image on top of the VM ware virtual machine. Open VM Ware and click on “Create a new Virtual Machine” and follow the steps as mentioned. It is recommended to select the disk size of VM to around 25 GB as it need to support another virtual machine for our project to work. Keep the number of processors to around 2-4.
- 5. Once the instance of VM is created, install the Ubuntu OS in your VM. You will be prompted when you open the VM Instance in VMWare.
- 6. Your Ubuntu Virtual Machine is setup. The next section describes the steps forgetting the latest version of Linux kernel source code and modifying files to achieve “printk” messages when a VM-Exit occurs.

### Linux Kernel Source Code and its Modifications

1. The Linux kernel source code is what that is present under the Linux OS and runs the programs as well as handles system calls through the Linux OS. Hence, we will download the Linux kernel source code and modify the contents of the file for achieving our goal.
2. However, modifying the code won't be sufficient. The VM Ware Ubuntu instance that we are running has a Linux kernel already running in place.
3. So, latest kernel source code is downloaded, modified and built on the current system to override the current kernel code.
4. Following are the steps:
  - a. Download the latest stable kernel from the below website  
<https://www.kernel.org/>
  - b. Download it and keep it in any folder but it is recommended to place the file in /usr/src Folder.
  - c. The kernel source code downloaded is usually a “.tar” file. Untar [3] the kernel source file for compilation.



Figure 1: Linux Kernel Source code zipped

d. Untar the zip file and you will get the Linux kernel source code folder as follows

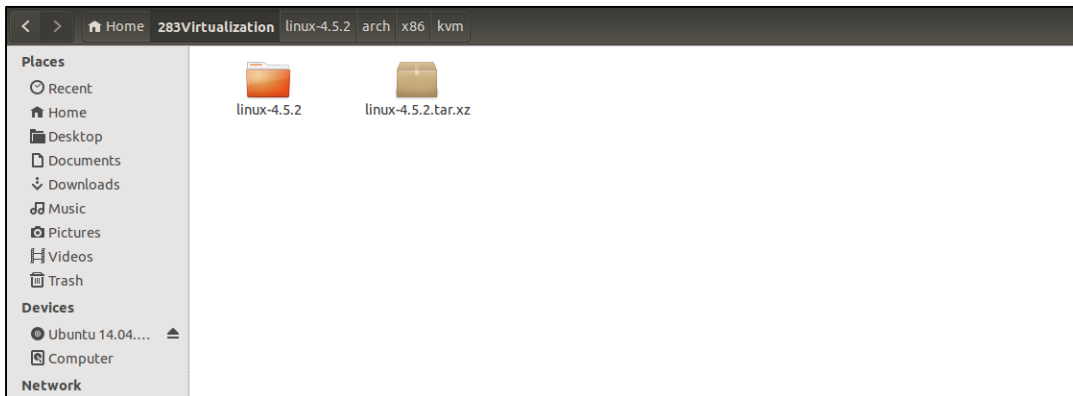


Figure 2: Unzipped Linux kernel source code folder

- e. Go to the following location in the Linux kernel source code folder  
/linux-4.5.2/arch/x86/kvm/
- f. Locate vmx.c in the folder. First create a copy of the file before modifying the file and store it in some USB drive or cloud storage.

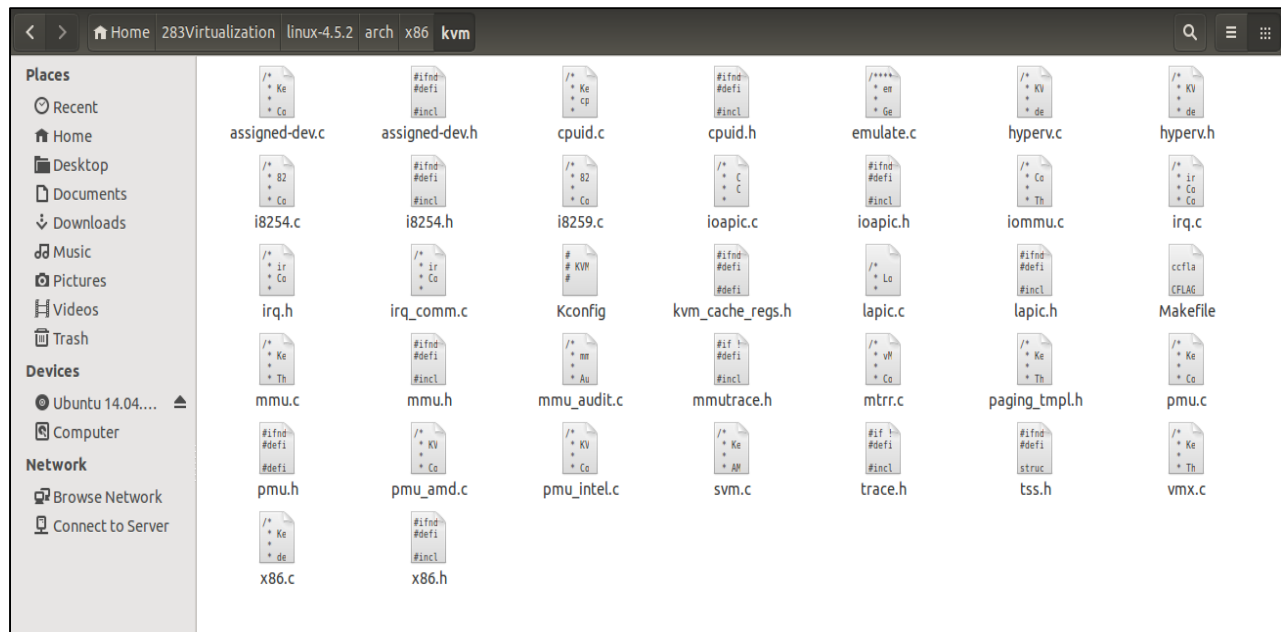


Figure 3: Location of vmx.c file

- g. VMX.c is the file which is to be modified for generating messages in the kernel log. By writing printk statements within VMX.c leads to the exits taken by VM to be recorded into the file called 'kern.log'. The messages can be displayed from kern.log using **dmesg command** in command line. The dmesg will be explained in later sections.
- h. Modification are performed once the KVM is setup and another VM is installed inside KVM. See "[Modifications in VMX.c](#)" section.

### Building the Kernel

- The first step is to configure the kernel. The kernel contains nearly 3000 configuration options. You can take any one of configuration from the distro, and on top of that you can add your own configuration.
- Open a Linux terminal and change the directory to where the Linux source code folder is present  
`cd /linux-4.5.2/`
- It is recommended to use the root user privileges using the command `sudo su`
- Before starting building we should install `libncurses` package and `ssl-devel` package using following commands one at a time

```
sudo add-apt-repository "deb http://archive.ubuntu.com/ubuntu  
$(lsb_release -sc) main universe"
```

```
sudo apt-get update  
sudo apt-get install libncurses5-dev
```

```
sudo apt-get install openssl-devel
```

- Now run the command  
`make menuconfig`

The make menuconfig, will launch a text-based user interface with default configuration options. The above packages are used for running this command.

- We will use the default config provided by the kernel. So select “Save” and save the config in the file name “.config”.

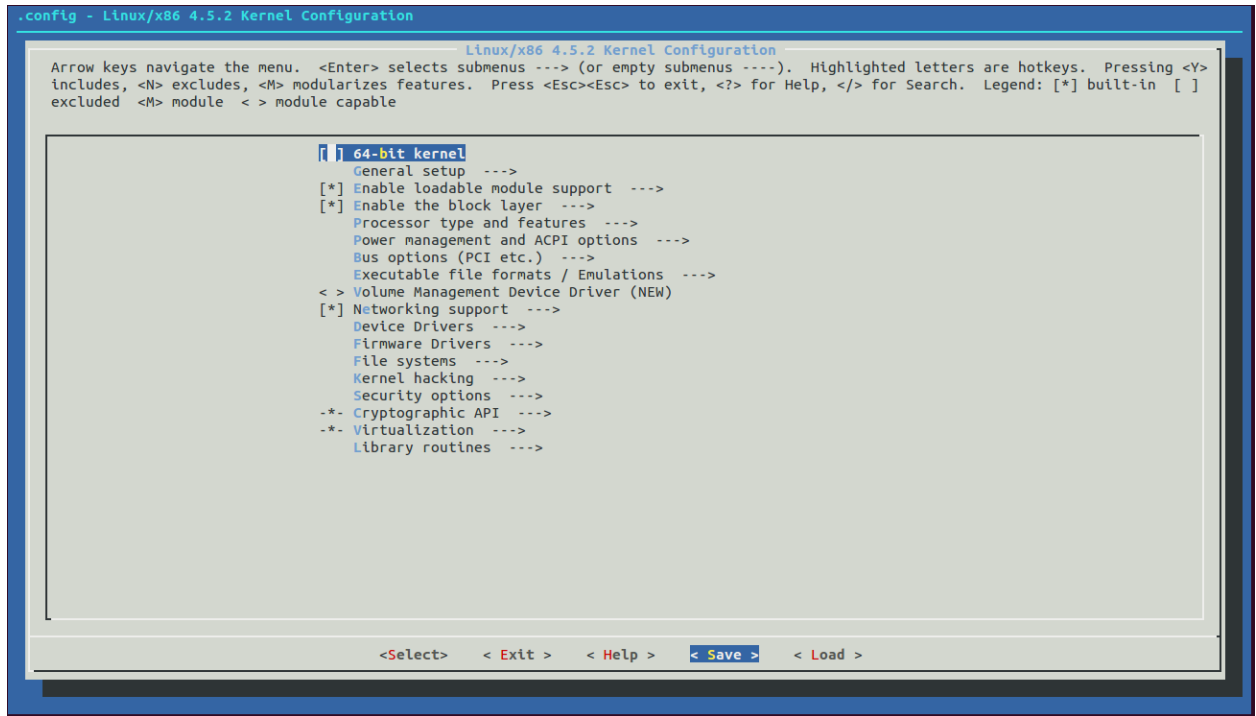


Figure 4: Make menu config interface

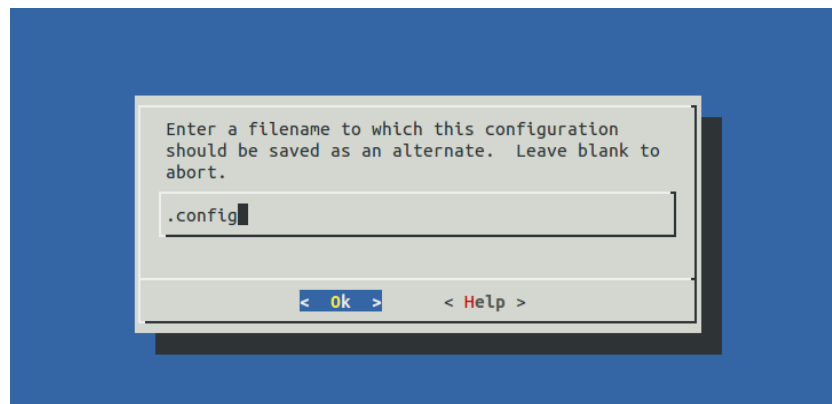


Figure 5: Creating Configuration file

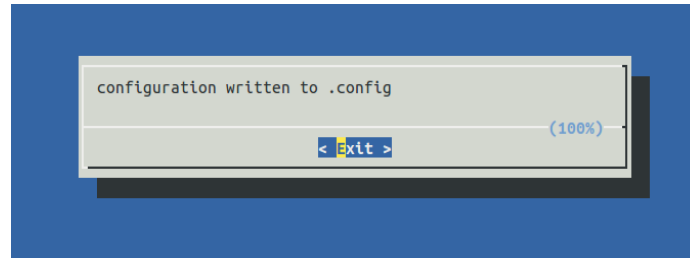


Figure 6: Configuration file created

- Once the configuration file is created we will start compiling the Linux main kernel using:  
# make  
Note: This takes approximately 3 to 5 hours if you are compiling for the first time.
- Once the Linux main kernel is compiled then we will compile modules using:  
# make modules
- After modules compilation install the kernel modules using command:  
# make modules\_install
- At this point, you should see a directory named /lib/modules/4.5.2/ in your system.
- Install the kernel using the following command:  
# make install

Note: The make install command will create the following files in the /boot directory.

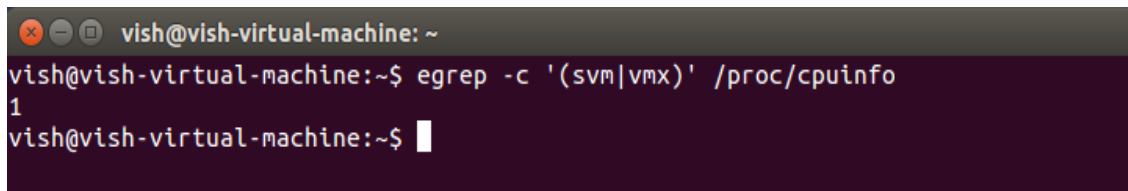
- vmlinuz-4.5.2 – The actual kernel
- System.map-4.5.2 – The symbols exported by the kernel
- initrd.img-4.5.2 – initrd image is temporary root file system used during boot process
- config-4.5.2 – The kernel configuration file
- To use the new kernel that you have just compiled, reboot the system.
- Once the system is up, use the “uname -r” command to get the version of the Linux kernel. It will show 4.5.2.
- The next section we will cover the pre requisites that will be necessary before installing KVM and KVM installation.

### KVM, Virtual Machine Installation and Pre requisites

1. KVM is a Linux based virtual machine that can run Windows/ Linux virtual machines. You can run KVM in both physical machine as well as Virtual machine provided the machine is running Linux environment. **KVM, we need to setup within the VMWare instance created**
2. KVM only works if your CPU has hardware virtualization support – either Intel VT-x or AMD-V.[2] To determine whether your CPU includes these features, run the following command in the Linux terminal:

```
egrep -c '(svm|vmx)' /proc/cpuinfo
```

3. A '0' value indicates that your CPU doesn't support hardware virtualization, while a value of '1' or more indicates that it does. You may still have to enable hardware virtualization support in your computer's BIOS, even if this command returns a 1 or more.



```
vish@vish-virtual-machine: ~  
vish@vish-virtual-machine:~$ egrep -c '(svm|vmx)' /proc/cpuinfo  
1  
vish@vish-virtual-machine:~$
```

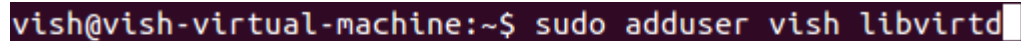
Figure 7: Checking the KVM compatibility

4. The following commands need to be run to install the KVM and all its supporting packages. The command also installs Virt-Manager which is graphical interface to manage your virtual machines.

```
sudo apt-get install qemu-kvm libvirt-bin bridge-utils virt-  
manager
```

5. Once the installation is done, we need add root users to operate the KVM as it runs only with root permissions. The name field below should be the user name [In my case vish]

```
sudo adduser name libvirtd
```

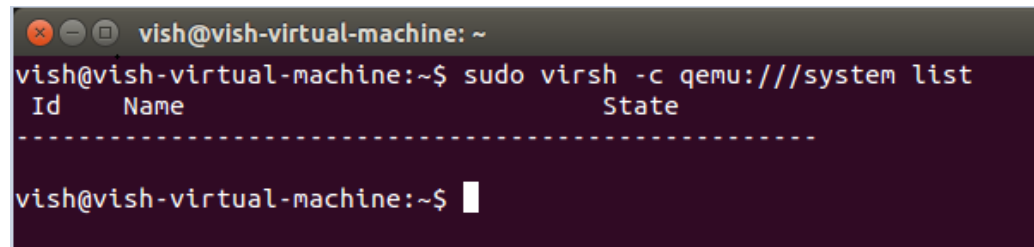


```
vish@vish-virtual-machine:~$ sudo adduser vish libvirtd
```

Figure 8: Adding super user to handle KVM

6. To see the list of virtual machines created using KVM we can run the following command:

```
virsh -c qemu:///system list
```



```
vish@vish-virtual-machine: ~  
vish@vish-virtual-machine:~$ sudo virsh -c qemu:///system list  
Id      Name                                     State  
-----  
  
vish@vish-virtual-machine:~$
```

Figure 9: Checking count of VMs created within KVM

7. Once the KVM has been setup we can create an instance of Virtual machine using Virtual Manager as follows:



Figure 10: KVM User Interface Application

- Download any Linux distro and create a new instance. I have installed Lubuntu v 16.04 which is a lighter version of Ubuntu in virtual machine manager. The steps are similar to steps discussed in “Setting up VMWare and creating virtual machine instance” section.

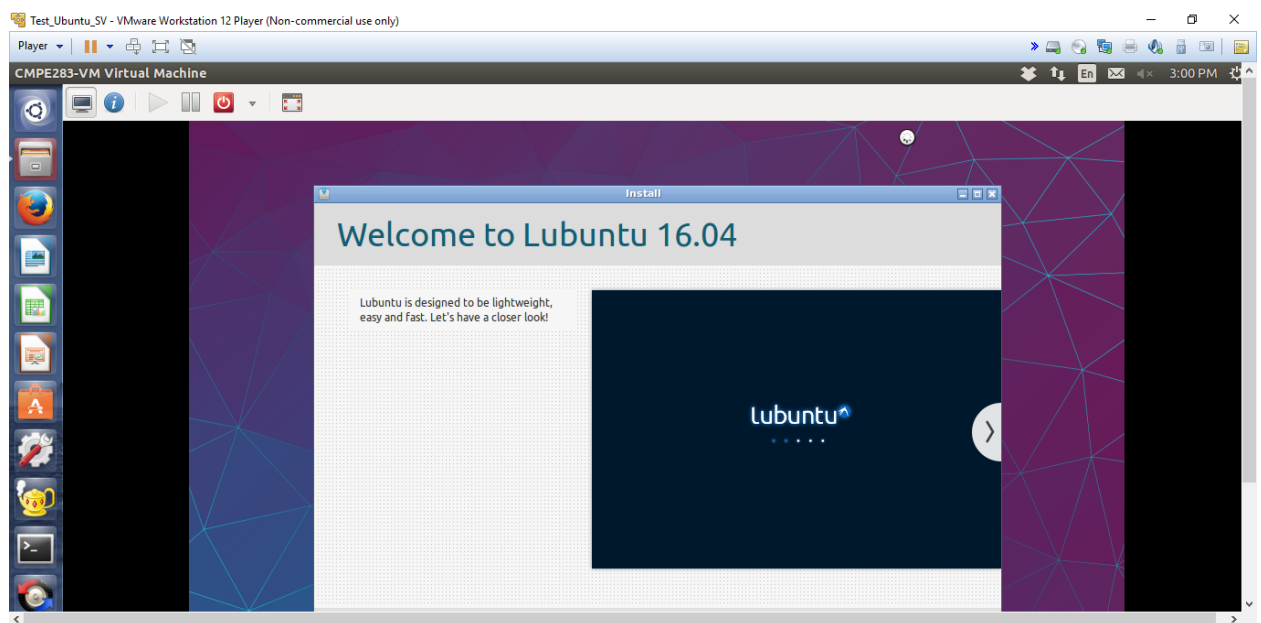


Figure 11: Installation of VM within KVM

### Modifications in VMX.c

- The first thing we need to find out is the number of exits reasons. In vmx.c, the exit handlers can be located under the structure:  
**static int (\*const kvm\_vmx\_exit\_handlers[])(struct kvm\_vcpu \*vcpu)**
- In the above structure you can find the names of various exit reasons and the handle functions for those exits such as handle\_triple\_fault, handle\_cpuid, handle\_vmcall etc. The count is approximately 45.



- Introduce 45 counter variables which are of the data type unsigned 64- bit integer as the number of exits are large enough.
- Now let's locate an exit handler and increment the corresponding counter variable. One example is demonstrated below:

```
u64 cpuid_exit_cntr; //counter variable
/* Exit handler where counter should be placed and incremented*/
static int handle_cpuid(struct kvm_vcpu *vcpu)
{
    cpuid_exit_cntr++; //counter is incremented as exit occurs
    kvm_emulate_cpuid(vcpu);
    return 1;
}
```

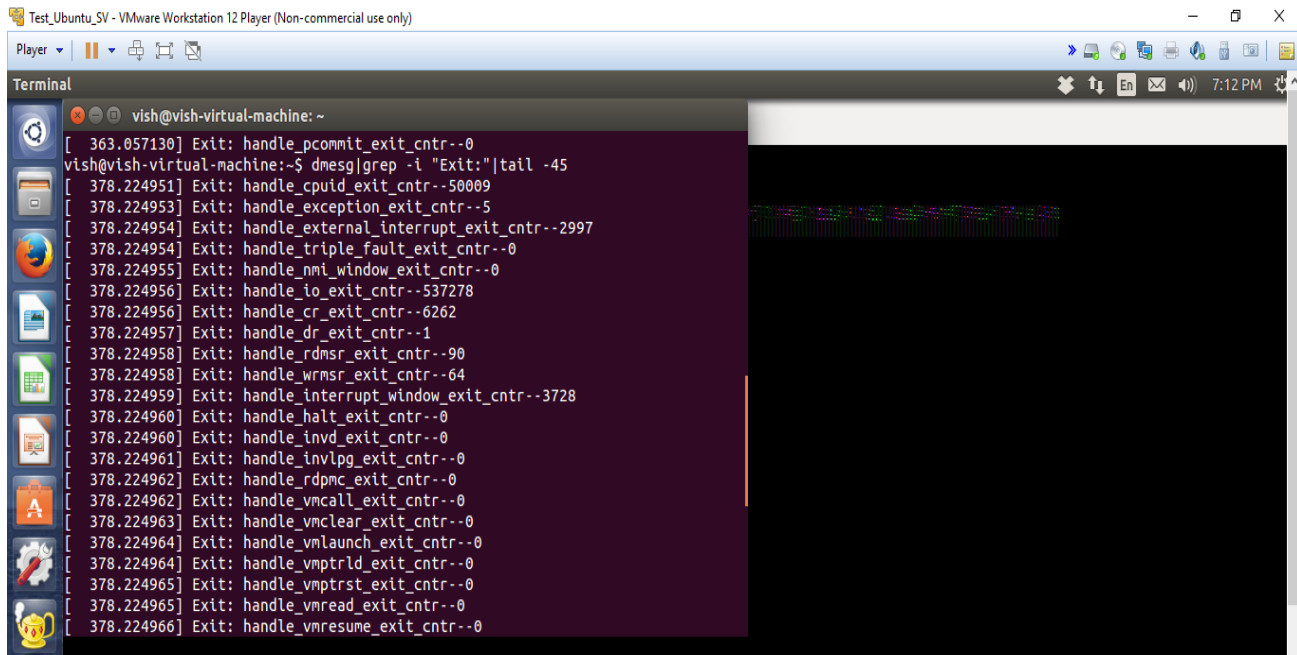
- Similarly, other counters should be placed within exit handlers and incremented. Whenever a VM in KVM takes exit, then the exit handlers come into picture and the counters are incremented.
- Once the counters are placed in corresponding locations, we now need to place printk statements to verify whether the counters are actually incremented or not. We can place either single printk statement or multiple statements under the function **vmx\_handle\_exit**. The print k statements are of the following format:  
Printk(KERN\_ERR "Exit: cupid exit--%d",cpuid\_exit\_cntr);
- Save the file vmx.c. Once we are completed, we should start [building the kernel](#) again for changes to reflect.

### Running the VM in KVM and recording exits

- Assuming, we have built the kernel and our system is up and running, we will now open the KVM and the run the virtual machine within the KVM.
- Open a Linux terminal in parallel and run the following command:

```
dmesg less | grep -i "Exit"| tail -45
```

- The command will give the following output as the number of exits that the VM is taking ordered by exit reason types.



```
vish@vish-virtual-machine: ~  
[ 363.057130] Exit: handle_pcommit_exit_cnr--0  
vish@vish-virtual-machine:~$ dmesg|grep -i "Exit:"|tail -45  
[ 378.224951] Exit: handle_cpuid_exit_cnr--50009  
[ 378.224953] Exit: handle_exception_exit_cnr--5  
[ 378.224954] Exit: handle_external_interrupt_exit_cnr--2997  
[ 378.224954] Exit: handle_triple_fault_exit_cnr--0  
[ 378.224955] Exit: handle_nmi_window_exit_cnr--0  
[ 378.224956] Exit: handle_io_exit_cnr--537278  
[ 378.224956] Exit: handle_cr_exit_cnr--6262  
[ 378.224957] Exit: handle_dr_exit_cnr--1  
[ 378.224958] Exit: handle_rdmr_exit_cnr--90  
[ 378.224958] Exit: handle_wrmsr_exit_cnr--64  
[ 378.224959] Exit: handle_interrupt_window_exit_cnr--3728  
[ 378.224960] Exit: handle_halt_exit_cnr--0  
[ 378.224960] Exit: handle_invd_exit_cnr--0  
[ 378.224961] Exit: handle_invlpg_exit_cnr--0  
[ 378.224962] Exit: handle_rdpmc_exit_cnr--0  
[ 378.224962] Exit: handle_vmcall_exit_cnr--0  
[ 378.224963] Exit: handle_vmclean_exit_cnr--0  
[ 378.224964] Exit: handle_vmlaunch_exit_cnr--0  
[ 378.224964] Exit: handle_vmptrld_exit_cnr--0  
[ 378.224965] Exit: handle_vmptrst_exit_cnr--0  
[ 378.224965] Exit: handle_vmread_exit_cnr--0  
[ 378.224966] Exit: handle_vmresume_exit_cnr--0
```

Figure 12: output of dmesg statement

## Troubleshooting

- Make sure you have enough memory space in the VM Ware virtual machine instance because it needs to support another nested VM within your VM through KVM.
- Use gparted tool to allocate disk space in your VM Ware instance if you intend to increase the disk space of your VM.
- Make sure all the packages and pre requisites are installed before building the kernel
- Make a copy of the original VMX.c file before you start modifying it.
- The exits printk statements introduced in the VMX.c will fill up the kern.log frequently. The logs are present in your system /var/log/ folder. Following steps can be done through command line to free up kern.log [4]. Open root mode in terminal using “sudo su” command and change directory to “/var/log”
  1. Optional: Copy log file  
`cp -av --backup=numbered file.log file.log.old`
  2. Optional: Use Gzip on copy of log  
`gzip file.log.old`
  3. Use /dev/null for clean file  
`cat /dev/null > file.log`

## Limitations

1. The main limitation in the implementation is the use of Nested Virtualization within a Virtual Machine which leads to more time in building the kernel as well as added requirements to be fulfilled. The limitation could be overcome by performing the same activities in a dual boot system wherein the Linux kernel is built upon the actual physical host.

2. The changes could have been built on the same Linux kernel version. Building a Linux kernel newer version leads to increased compilation time. This can be overcome by modifying vmx.c and installing the module in the same Linux kernel version.
3. Getting acquainted with the Linux kernel source code can help much in introducing changes without doing trial and error on certain files and testing repeatedly which was also one of my limitation.

## References

- [1] <https://www.vmware.com/products/player/faqs/install-requirements>
- [2] <http://www.howtogeek.com/117635/how-to-install-kvm-and-create-virtual-machines-on-ubuntu/>
- [3] <http://www.thegeekstuff.com/2013/06/compile-linux-kernel/>
- [4] <http://askubuntu.com/questions/515146/very-large-log-files-what-should-i-do>