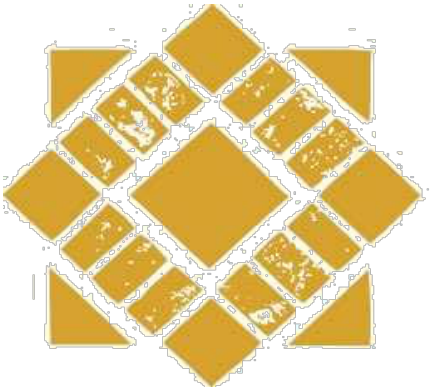

San José State UNIVERSITY

Master of Science (Software Engineering)

CMPE 283: Virtualization Technologies

Spring 2016

Assignment

on

"Extending KVM"

Presented by:

Members Name	SJSU ID
Jaya P Patil	010693540
Ketki N Gawande	010716043

UNDER THE GUIDANCE OF

Prof. Michael Larkin

Description

Installing and Configuring KVM

First step is to install and configure Kernel Virtual Machine (KVM). But before installing, we need to make sure that our CPU has virtualization support. To determine whether the CPU includes these features, following command is used.

$sudo grep -c "svm\|vmx" /proc/cpuinfo

A 0 indicates that your CPU doesn't support hardware virtualization, while a 1 or more indicates that it does.

Second step is to install KVM and supporting packages. To do this, we have run the following command.

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

We have also installed virt-manager which is a graphical application for managing virtual machines. Libvirt and Virt-manager simply the process.

Next step is to check whether everything is working correctly. To make sure that, we ran the following command.

$virsh -c qemu:///system list

After running this command, empty list of virtual machine was displayed.
This indicates that everything is working correctly.

We can open the virtual machine by running the following command.

$virt-manager

Downloading the Linux Kernel Source Code

We downloaded our kernel source code from https://www.kernel.org/ version 4.1.24. We can untar it by using following command.

$tar xvf linux-4.1.24.tar.xz

Modifying KVM

There can be several cases where VM Exits will be required. There are certain instructions, which can cause VM Exits. Examples include CPUID, HLT, INS and OUTS etc. Other cases may include Exceptions, External Interrupts and Task Switches etc.

We did an extensive research in order to locate the handling of exit types in kernel code. We ran 'grep' command on exit keyword. By analyzing the result, we found that there are many types of exits are defined in vmx.c code which can be found in /linux-4.1.24/arch/x86/kvm/ directory. Few of them are, EXIT REASON EXCEPTION NMI, EXIT REASON CPUID, EXIT REASON WBINVD and EXIT REASON EXTERNAL INTERRUPT. These exits are properly handled in the code. The exit handlers return 1 if the exit was handled fully and guest execution may resume. Otherwise they set the kvm run parameter to indicate what needs to be done to userspace and return 0. Each exit handler is a function pointer.

We have maintained one integer array of size equal to number of exit types to keep the counter of each exit type and the array is initialized to zero. Whenever any type of exit is handled, respective count of the type is incremented.

Add a facility to KVM to print the content of the counters

We have added a system call (sys_print_vmx_exit_counters()) to print the counters to dmesg. This system call in invoked from the simple binary printvmxexitcounters. This binary runs in userspace.
The system call sys_print_vmx_exit_counters() invokes a method print_vmx_exit_counters() which is defined in arch/x86/kvm/vmx.c. This file also handles all types of vm exits and maintains the counters. These counters are printed to system log (dmesg).

Procedure followed:

1) Go to system call table located at arch/x86/syscalls/syscall_64.tbl which contains one entry for every single syscall. We made an entry at 323 for our system call. We also made a declaration for our syscall in include/ linux/syscall.h file.
2) We wrote the syscall function in which we invoked our print functionality.
3) Wrote a new makefile for this function. And followed the steps for Kernel compilation and installation.
4) After booting into new kernel, we wrote a small function called userspace.c to test our system call.
5) After launching new VM and running this code , run the command dmesg. We were able to see our print functionality working.

Recompiling and Installing Kernel Source Code

We have followed the following procedure step by step to compile the kernel source code.

[1] Check the current kernel version by running ‘uname –r ‘ command.
[2] Go to linux source code directory. First step is to configure the environment for compiling the source code. We chose to use old configuration file used by the current version of the kernel. To do this, run the following commands.

#cp /boot/config-3.19.0-59-generic .config

After running this command, the old config file got copied to linux source code directory.
[3] To start the configuration, run command ‘*make oldconfig’* , it will ask you to select some changes, we can make default choice by pressing ENTER key.
[4] After configuring, compile the kernel by running *‘make’* command.
[5] Once the compilation is successful, install the modules by running ‘*make modules_install’* command.
[6] Once we have modules installed, install the kernel by running *‘make install’* command
[7] In the next step, check ‘*ls /boot’* and we can see new kernel image, config file, system.map and also we will be able to see initial ram disk (initrd.img) file corresponding to new kernel version. If you don’t see initrd.img file corresponding to new version, run the following command.

#mkinitramfs –o /boot/initrd.img-4.1.24 4.1.24

where 4.1.24 is our new kernel version.
[8] Last step is to update the bootloader. By running ‘update-grub’ command, we can achieve this.
[9] Reboot the machine and in grub, go to advanced options and select the new kernel version.
[10] To make sure, run the command ‘*uname -r*’, now it will show the new kernel version.

Now launch the KVM by typing virt-manager in the terminal. Follow the required steps to load the iso file in KVM and start the VM. Run the userspace.c program and run dmesg command. You will be able to see the output.

Contribution by team member

1) Jaya Patil :
    - Modifying KVM source code
    - Adding a facility to KVM to print the contents of the counters to the system log when requested.
2) Ketki Gawande:
    - Configuring KVM
    - Compiling and installing kernel source code.

Limitations

As there are many types of exits present in the code, it is not feasible to test each and every exit by performing specific activity in VM. So to test our implementation, we launched the VM in KVM, performed some activities like keyboard interrupt, mouse interrupt, shutting down the VM, suspending the VM. Then we observed the collective output of all the activities in dmesg.