# Project 3:Implementing a Sysfs interface to the VMCS

Menglin Miao,CS544,spring2016

**Abstract**

The VMCS is the virtual machine control structure. It is used to manage and track the virtual CPU in the VMX/KVM layer of the kernel. Sysfs is a virtual filesystem that can be used for bidirectional communication with the kernel. This document will introduce how to making use of CentOS 7 64-bit with Minnowboard and install the 4.1.5 linux kernel, and reach the requirement.

## Contents

**I Introduction** 2

**II Terminology** 2

**III VMCS Structure** 2

III-A What is VMCS. 2

III-B VMCS Region 2

III-C VMCS Data 3

III-D Guest-state area 3

III-E Host-state area 3

III-F VM-execution control fields 3

III-G VM-Exit Control Fields 3

III-H VM-Entry Control Fields 4

III-I VM-Exit Information Fields 4

III-J VM-Instruction Error Field 4

**IV CentOS7 install** 4

IV-A Installation 4

IV-A1 Make USB flash disk 4

IV-A2 Install CentOS 5

**V Linux 4.1.5 install** 5

V-A Download Linux 4.1.5 5

V-B Configuration 6

V-C Menuconfig 6

V-C1 General Setup 6

V-C2 Enable Loadable module support 7

V-C3 Enable the block layer 7

V-C4 I/O schedulers . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

V-C5 Processor types and features . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

V-C6 Power management and ACPI options . . . . . . . . . . . . . . . . . . . . . . . . . 8

V-C7 Device Drivers . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

V-C8 Networking support . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

V-C9 File system . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

**VI Implementing a sysfs interface to the VMCS** 8

VI-A Kconfig . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

VI-B Kobject . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

VI-B1 Manage and control Kobject . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

VI-B2 Reference Counts . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

VI-B3 kref . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

VI-C Ktypes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

VI-D Sysfs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

VI-D1 Add and delete kobject from sysfs . . . . . . . . . . . . . . . . . . . . . . . . . . 11

VI-D2 Add file into sysfs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

VI-E Implementation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

VI-E1 Makefile . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

**VII Modify Code** 11

**VIII Source Code** 13

**IX Test** 17

IX-A Test Read . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

IX-B Test Write . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

**References** 18

## I. Introduction

The purpose of this project is to build a VMCS Sysfs interface. So that can access kernel VMCS data in user-space. The Linux kernel can accept the changes and interact with client. This project is use Minnowboard as hardware, CentOS, Linux 4.1.5 as operating system. In this report, firstly will introduce some concept of VMCS, and introduce how to install the operating system, and then is how to implement the interface. In addition, some important concept and function are introduce in this report.

## II. Terminology

- VM: Virtual machine.
- VMX: Virtual-machine extensions
- VMM: Virtual-machine monitors.
- VMCS: Virtual-machine control data structures.

## III. VMCS Structure

### A. What is VMCS.

VMX operation is use for support virtualization. There are two kinds of VMX operation: VMX root operation and VMX non-root operation. A VMM is run in VMX root operation and guest software is run in VMX non-root operation. VMX Transitions are between VMX root operation and VMX non-root operation. VM entries is transition into VMX non-root operation. VM exits is transitions from VMX non-root operation to VMX root operation.
A logical processor uses virtual-machine control data structures (VMCSs) for its VMX operation. It is manipulated by the instructions of VMCLEAR, VMPTRLD, VMREAD, and VMWRITE.
A VMM can use a different VMCS for each virtual machine that it supports. It use different VMCS for each virtual processor of the virtual machine that have multiple processors.
A virtual processor may maintain many VMCSs that are active. VMX operation may optimized by the processor by maintaining the state of an active VMCS in the memory, on the processor, or both. On a specific time, the current VMCS at most have one from the active VMCSs. And the VMLAUNCH, VMREAD, VMRESUME, and VMWRITE instructions only can operate on the current VMCS.
The launch state of a VMCS is use for decide which VM-entry instruction will be used with the VMCS: The ”clear” launch state of a VMCS is for the VMLAUNCH instruction; the ”launched” launch state of a VMCS is for the VMRESUME instruction launched.

### B. VMCS Region

VMCS region is a region in memory that associates A logical processor with each VMCS. Software references a specific VMCS using the 64-bit physical address of the region (a VMCS pointer). VMCS pointers must be aligned on a 4-KByte

boundary (bits 11:0 must be zero). These pointers must not set bits beyond the processor's physical-address width.
VMCS revision identifier use the first 32 bits of the VMCS region. Processors that maintain VMCS data in different formats use different VMCS revision identifiers.
VMX-abort indicator use the next 32 bits of the VMCS region. It not control processor operation. A virtual processor writes a non-zero value into these bits if a VMX abort. Software may also write into this field.
VMCS data use the remain VMCS region, it use for control VMX non-root operation and the VMX transitions. The format of these data is implementation-specific.

## *C. VMCS Data*

The VMCS data have six part:

- Guest-state area. Processor state will saved into the guest-state area when VM exits and loaded from there when VM entries.
- Host-state area. Processor state is loaded from the host-state area when VM exits.
- VM-execution control fields. These fields controls processor behavior in VMX non-root operation. They decide some reasons for VM exits.
- VM-exit control fields. These fields control VM exits.
- VM-entry control fields. These fields control VM entries.
- VM-exit information fields. These fields receive information on VM exits and describe the reason and the nature of VM exits. They are read-only.

## *D. Guest-state area*

This section describes fields contained in the guest-state area of the VMCS. Processor state is loaded from these fields on every VM entry and stored into these fields on every VM exit.

## *E. Host-state area*

This section describes fields contained in the host-state area of the VMCS. Processor state is loaded from these fields on every VM exit. All fields in the host-state area correspond to processor registers:

## *F. VM-execution control fields*

The VM-execution control fields govern VMX non-root operation.

## *G. VM-Exit Control Fields*

The VM-exit control fields govern the behavior of VM exits.

### *H. VM-Entry Control Fields*

The VM-entry control fields govern the behavior of VM entries.

### *I. VM-Exit Information Fields*

The VMCS contains a section of read-only fields that contain information about the most recent VM exit.

### *J. VM-Instruction Error Field*

paragraphThe 32-bit VM-instruction error field does not provide information about the most recent VM exit. In fact, it is not modified on VM exits. Instead, it provides information about errors encountered by a non-faulting execution of one of the VMX instructions.

## IV. CENTOS7 INSTALL

CentOS (Community Enterprise Operating System) is a Linux distribution that provide a free enterprise-class, community-spported computing platform which aim to be functionally compatible with its upstream source, Red Hat Enterprise Linux (RHEL). It will be use in this project and install it on the Minnowboard Turbot.
Minnowboard Turbot is a MinnowBoard MAX-compatible board designed and produced by ADI Engineering. It include the processor of 64-bit Intel Atom E38xx Series SoC and a Integrated Intel HD Graphics that design for Linux OS. Therefore, we can install CentOS 7 on it, amd rim the Linux 4.1.5 on it. There have some port that important on the Minnowboard Turbot.

- HDMI: The MinnowBoard uses a Type D micro-HDMI connector. This is a standard port. Cables and adapters can be readily picked up from most electronics stores.
- Ethernet: The MinnowMax uses a Realtek RTL8111GS-CG PCIe based chipset to provide 10/100/1000 ethernet connection.
- Low Speed Expansion (Top): The low speed connector uses 0.1” (2.54 mm) Male header pins in a 2 x 13 array, for a total of 26 pins. Pin 1 is the in the row closest to the power connector, and closest to the board edge.
- High Speed Expansion (Bottom): The High speed connector uses a TE Connectivity compatible 60-pin header. The generally recommended header is the 3-5177986-2, or the 60POS .8MM FH 8H GOLD part that rises 7.85mm, allowing for 3/8” standoffs at the corners to be used to attach the lure to the minnowboard.

### *A. Installation*

This project required to use Centos

*1) Make USB flash disk:* Go to the website https://www.centos.org/download/, and get the CentOS 7 ISO image. after download the ISO image. Then make the USB system boot. There are a lot of software can create a bootable USB flash disk. After make USB, we can use the USB to install system.

### 2) *Install CentOS:*

- Connect HDMI.
- Insert SD card.
- Insert USB into the Minnowboard, and power on the Minnowboard.
- After power on, press F2 go to the BIOS menu, choose EFI USB to boot it.
- After success boot the system, choose the install CentOS 7 option to install the operating system.
- Select the language of English(United States).
- Choose the install location, and enter the disk partition interface.
- After the configuration of the install location. begin installation.
- Set the password of the root.
- After install the system, restart the operating system, and use root account to log in.
- Now is set network.
- cd /etc/sysconfig/network-scripts/ enter the network configuration directory.
- vi ifcfg-eno16777736
- Add and modify some content below, and save the the document.
  - TYPE=Ethernet
  - BOOTPROTO=static
  - DEFROUTE=yes
  - PEERDNS=yes
  - PEERROUTES=yes
  - IPV4_FAILURE_FATAL=no
  - IPV6INIT=yes
  - IPV6_AUTOCONF=yes
  - IPV6_DEFROUTE=yes
  - IPV6_PEERDNS=yes
  - IPV6_PEERROUTES=yes
  - IPV6_FAILURE_FATAL=no
  - NAME=eno16777736
  - UUID=ae0965e7-22b9-45aa-8ec9-3f0a20a85d11
  - ONBOOT=yes
- Enter: service network restart. #restart the network.

## V. Linux 4.1.5 install

### A. *Download Linux 4.1.5*

- Linux 4.1.5 need to install in centOS in Minnowboard.
- First enable the ethernet interface. ifup enpos3 command will be enable Ethernet, To enable Ethernet after boot, need to modify system interface config file. change the ONBOOT value to yes.

- Log in with root account.
- Install the dependency to compile Linux kernel.
  - yum install gcc ncurses ncurses devel
  - yum update
- Go to the directory of /tmp and download the Linux 4.1.5 from http://www.kernel.org/pub/linux/kernel/v4.x/linux-4.1.5.tar.xz
  - cd /tmp
  - wget http:// www.kernel.org/pub/linux/kernel/v4.x/linux-4.1.5.tar.xz
- extract the Linux 4.1.5 in /usr/src/
  - tar xvjf linux-4.1.5.tar.xz -C /usr/src
  - cd /ssr/src/linux-4.1.5

## B. *Configuration*

- Use the configure file that from class website of the minimal .config, and use command mv minimal.config .config to change the minimal.config for our current configuration.
- We also need to download the config files for Minnowboard, after download this config file from http : // www.elinux.org/images/e/e2/M
  we need to merge the two kernel config files.
- Now we need to compile and install kernel, use command make -j4 && make -j4 modules.
- To install the new kernel and modules, use command make install &&make modules_install.
- We also can use make menuconfig command to configure the Linux 4.1.5
  - make menuconfig
- After input make menuconfig command, we will be enter into a graphic interface to configure the core functions. Below is some useful option of the kernel configuration option:

## C. *Menuconfig*

After enter the graphic interface, their will be havce several option to config. Moreover, about the core functionality select, should be consider some point: If you sure the core function will be use in the future, complied directly into the kernel. If the function maybe use in the future, complied into module as possible. In short, try to keep he core small and beautiful, the rest of the function is compiled into a module.Below is some detail information that important.

*1) General Setup:* The most revelant program interact with Linux, the core version description, wether use the developing code and some information are set here. The program here are design for the correlation between core and program. In general, keep the default value is well, and do not cancel any project underneath, because it may cause difficulties in some programs that can not be executed simultaneously. If you have doubts, you can press the ¡Help¿ button, and find help. You can select the functions based Help proposals. Below is some useful information:

- Networking support: Internet support, this option must be choose.

- PCI support: PCI Support, if you use the PCI card, must be choose.
- PCI access mode: have BIOS, Direct and Any option. generally choose the any option.
- Support for paging of anonymous memory (swap): This is the use of swap or swap file for virtual memory.
- System V IPC: It used to synchronize the processor between the program and exchange the information.
- POSIX Message Queues: This is the POSIX message queue.
- Auditing support: Use for the kernel and certain submodules simultaneously.
- Kernel .config support The .config configuration information stored in the kernel.
- Optimize for size: This option cause gcc use -Os parameters instead of -O2 to optimization parameters to get smaller kernel.

*2) Enable Loadable module support:* If your core need to support dynamic kernel modules, you should be enable loadable module option. Below is some useful information of the option:

- Forced module unloading: Allow forced to unload a module that is being used (more dangerous). This option allows you to forcibly unmount module, even if the kernel think it's unsafe. Kernel will be remove the modules immediately, regardless of whether someone is using it. This is mainly offer for developers and users who are impulsive. If you are unsure, choose N.
- Forced module loading: Allow Forced load module.
- Module unloading: Allow unloading already loaded module.
- Module versioning support: Sometimes, you need to compile the module. This option will add some version information to the compiled modules to provide independent features. So different kernel can distinguish it from the original module when use the same module.

*3) Enable the block layer:* Block device support, hard disk/USB/SCSI devies required use this option makes the block device can be removed from the kernel. If not select, the blockdev file will not available ,some file system such as ext3 will be unavailable. The block layer is enable by default, you can also enter the breakdown set in this project, select the function you need. Below is some useful information of the option:

- Block layer SG support v4: Scsi generic block device the fourth Version support.
- Block layer data integrity support: Data integrity support for Block devices

*4) I/O schedulers:* IO scheduler I/O is input and output bandwidth control, mainly for the hard disk, is the core for kernel.

- Deadline I/O scheduler: Polling scheduler, simple and compact, providing a minimum read latency and throughput well, especially adapted to the environment that have more read(such as a database). Deadline I/O scheduler is simple and close, its performance is as well as the preemptive scheduler and work better when some of the data in.
- CFQ I/O scheduler: Use QoS policies and assigned the same amount of bandwidth for all tasks, avoid process starvation and to achieve low latency. CFQ scheduler attempts to provide the same bandwidth for all processes. It will provide equal working environment, very suitable for a desktop system.

*5) Processor types and features:*

*a) Enter the Processor type and features, choose the form of your actual CPU. Below is some useful information::*

- High memory support: Large memory support, it can support up to 4G, 64G, generally can not choose it.

*6) Power management and ACPI options:*

*a) If you select ”Power management and ACPI options”, After that, it will enter the power management mechanism system. In fact, the power management mechanism also need to match the motherboard and the associated power-saving features of the CPU, then can actually achieve power-saving.:*

*7) Device Drivers:*

*a) Enter ”Device Drivers” This is a driver library for all hardware devices. This interface lets you select features and parameters for the build. Features can either be built-in, modularized, or ignored. Parameters must be entered in as decimal or hexadecimal numbers or text.:*

*8) Networking support:*

*a) Network options, it is mainly about a number of options for network protocols. Linux is a network operating system, the most powerful feature of Linux is that the flexible support for network features.:*

*9) File system:*

*a) The file in the Linux file system is a collection of data, the file system contains not only the data of the files but also the structure of the file system. The files, directories, soft links and file protection information that can see by All Linux users and programs are all stored in the file system.:*

## VI. Implementing a sysfs interface to the VMCS

*b) This section will introduce how to modify the Kconfig file, and how to modify the vmx.c. In addition, we will introduce some function about Kconfig and Kset.:*

### A. Kconfig

To enable and disable the VMCS SYSFS interface, we need add some content to the /arch/x86/kvm/Kconfig. We need add information below:

```
config VMCS SYSFS
tristate ”Enable␣VMCS␣SYSFS”
depends on KVM \&\& TRACEPOINTS
```

### B. Kobject

Kobject is the heart of the device model. It is short for kernel object, which is represented by struct kobject and defined in linux/kobject.h. It provides basic facilities, such as reference counting, a name, and a parent pointer, enabling the creation of a hierarchy of object. Below is the structure.

```
struct kobject {
        const char *name;
        struct list_head entry;
        struct kobject *parent;
        struct kset *kset;
        struct kobj_type *ktype;
        struct sysfs_dirent *sd;
        struct kref kref;
        unsigned int state_initialized:1;
        unsigned int state_in_sysfs:1;
        unsigned int state_add_uevent_sent:1;
        unsigned int state_remove_uevent_sent:1;
        unsigned int uevent_suppress:1;
};
```

Here is some detail information about the element of he structure.

- the name pointer point to the name of this Kobject.
- The parent pointer points to this kobject's parent. In this manner, kobjects build an object hierarchy in the kernel and enable the expression of the relationship between multiple objects. This is the Sysfs, which is a user-space filesystem representation of the kobject object hierarchy inside the kernel.
- The sd pointer points to a sysfs_direct structure that represents this kobject in sysfs. Inside this structure, it is a n inode structure representing the kobject in the sysfs filesystem.
- The kref structure provides reference counting.
- the ktype and kset structures describe and group kobject.

Kobjects are usually embedded in other structures.

*1) Manage and control Kobject:* First is declaring and initializing a Kobject. Kobject is initialized via the function kobject_init. It is declared in linux/kobject.h.

The first parameter in the function kobject_init is the kobject that need to initialize. Before calling this function, the kobject must be cleaned. This might normally done during the initilization of the higher function which the kobject is embedded in. If not cleaned, call memset() to do this trick.

After cleaned, we can initialize the parent and kset safely.

We also can use kobject_create() to tackle it automatic, it return a new allocate kobject. Most time, we should be use kobject_create() or a related helper function rather than directly manipulate the structure.

*2) Reference Counts:* One of the primary feature provided by kobjects is a unified reference counting system. After initialization, the kobject's reference count is set to one. So long as the reference count is nonzero, the object continues to exist in memory and is said to be pinned. Below is some useful fuction:

- Increase reference count: Increment the reference count is done via kobject_get(), declared in linux/kobject.h
- Decrease reference count: Decrease reference count is done via kobject_put(), declared in linux/kobject.h

*3) kref:* The kobject reference counting is supported by the kref structure, it is define in linux/kref.h. before use kref, your must initialize it through kref_init(). If you want to get the reference of kref, we need call kref_get() function, it is declared in linux/kref.h. This function increase the reference count and it do not have return value. kref_put() decrease the reference fo kref, it declared in linux/kref.h. If the count is down to zero, then we need call release() function.

## *C. Ktypes*

Kobject is relative to a special type, it is Ktype(kernel object type). Ktype is represented by struct kobj_type, it defined in linux/kobject.h. Kobj type data structure contains three fields:A release method for releasing resources that kobject occupied; A sysfs_ops pointer point to the sysfs operating table; A default list of attributes of sysfs file system. Sysfs operating table includes two functions store () and show (). When a user reads the state property, show () function is called, the function encoding specified property values are stored in the buffer and returned to the user mode. The store () function is used to store property values that come from user mode.The detail structure is below:

```
struct kobj_type {
void (*release)(struct kobject *);
const struct sysfs_ops *sysfs_ops;
struct attribute **default_attrs;
};
```

Here is the detail introduce of the element in kobj_typ:

- Release pointer point to the deconstructor that called when the kobject reference count reach zero. This function is responsible for free memory and other clean up.
- sysfs_ops variable points to a sysfs_ops structure. This structure describe the behavior of sysfs files on read and write.
- Default_attrs points to a attribute structure array. This structs define the default attribute of this kobject. attribute describe the feature of the given object. If this kobject export to sysfs, this attribute will be export as a relative file. The last element in the array must ne NULL.

Ktype is for describe the behavior of a group of kobject, instead of each kobject define its own behavior, the normal kobject behavior defined in ktype structure once, then all the similar kobject can sharing this same feature.

## *D. Sysfs*

Sysfs is a ram-based filesystem initially based on ramfs. It provide a means to export kernel data structures, their attributes, and the linkages between then to userspace. Sysfs file system is a special file system that similar to a proc file system. It use for organize the device in the system into a hierarchy and provide detailed information of kernel data structures for user-mode programs. Under /sys is some directory about sysfs:

- Block directory: contains all block devices.
- Devices directory: Contains all of the device of system, and organized into a hierarchy according to the type of device attached bus.
- Bus directory: Contains all system bus types
- Drivers directory: Includes all registered kernel device driver
- Class directory: system device type.
- Kernel directory: kernel include the kernel configuration option and state information.

The most important directory is devices, this directory export teh device module into user-space.

*1) Add and delete kobject from sysfs:* Only initialize the kobject cannot export into sysfs. If want to export kobject to sysfs. We need function kobject_add(). In general, one or both of parent and kset should be set appropriately before kobject_add() is called. The help function kobject_create_and_add() combines the work of kobject_create() and kobject_add() into one function.

*2) Add file into sysfs:* Kobjects map to directories, and the complete object hierarchy maps to the complete sysfs structure. Sysfs is a tree without file to provide actual data.
Here we have some controls for attributes.

- Default attributes: the default files class is provide through the field of kobject and kset. Therefore, all the kobject that have same types have same default file class under the corresponding sysfs directory. kobj_type include a field, which is default_attrs, it is an attribute structure array. This attribute responsible map the kernel data into a file that in sysfs.
- Create new attribute: In special situation, some kobject instance need to have its own attribute. Therefore, kernel provide a function sysfs_create_file() interface to create new attribute.
- Remove new attribute: Removing a attribute need the function sysfs_remove_file().

### *E. Implementation*

*1) Makefile:* In order to compile the file. we meed change the Makefile file of /arch/x86/kvm/Makefile.

```
obj?$(CONFIG VMCS SYSFS) += vmcs?sysfs.o
```

## VII. Modify Code

Vmx.c is under the directory /arch/x86/kvm. VMCS sets is in this file. In order to access the VMCS, we should add code that how to read and write VMCS sysfs. Below is the functions that how reach the function that kernel read and write VMCS SYSFS.
We define several attribute to represent the files in vmcs directory. This attribute structures have different name and same mode. They store in d_abt[]. After define the attributes, we define Ktype. Its default_attrs is d_abt[]. When doing the

kobject_init_and_add() function we use this Ktype.

Below is some of the functions.

```
struct attribute rip_a = {
        .name = "g_rip_a",
        .mode = S_IRWXUGO,
};

struct attribute rsp_b = {
        .name = "g_rsp_b",
        .mode = S_IRWXUGO,
};

struct attribute cr0_c = {
        .name = "g_cr0_c",
        .mode = S_IRWXUGO,
};
static struct attribute * d_abt[] = {
        &rip_a ,
        &rsp_b ,
        &cro_c ,
        &cr3_c ,
        &cr4_c ,
        &G_interuptibility_info ,
        &G_rflags ,
        NULL,
}
struct kobj_type kt = {
        .release = vmcs_sysfs_release ,
        .sysfs_ops = &obj_sys_ops ,
        .default_attrs = d_abt ,
};
```

After define the attribute and ktype, we need define sysfs ops structure. It have two functions to operate the kobject. .show is using kobj_show() function, another is .store that using kobj_store() function. obj_sys_ops function is use in Kt structure. below is the function

```
static struct sysfs_ops obj_sys_ops = {
        .show = kobj_show ,
        .store = kobj_store ,
};
struct kobj_type kt = {
        .release = vmcs_sysfs_release,
        .sysfs_ops = &obj_sys_ops,
        .default_attrs = d_abt,
};
```

The function kobj_show and kobj_store tell provide the function that let the kernel read and write vmcs sysfs. The vmcs_readl() in kobj_show and vmcs_writel() in kobj_store are used for get the vmcs value from kernel and change the vmcs value. In the show two function, we have a switch that use for make sure which value in vmcs to read or write. This two function return the amount of the bit that the function read or write.

```
ssize_t kobj_show ( struct kobject * kobject , struct attribute * attr , char * buf )
{
        ssize_t count;
        unsigned long value;
        unsigned long field;
        field = get_field_addr(attr->name);
        printk("vmcs: read %ld\n", field);
        value = vmcs_readl(field);
        count = sprintf(buf, "%ld\n", value);
        return count;
}

ssize_t kobj_store( struct kobject * kobject , struct attribute * attr , const char *
{
unsigned long value;
unsigned long field;
field = get_field_addr(attr->name);
value = simple_strtoul(buf, NULL, 2);
printk("vmcs: write %ld\n", field);
vmcs_writel(field, value);
return count;
}
```

## VIII. Source Code

```
#include <linux/device.h>
#include <linux/init.h>
```

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/stat.h>
#include <linux/string.h>
#include <linux/sysfs.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Author");

struct kobject kobj;

extern unsigned long vmcs_readl(unsigned long field);
extern void vmcs_writel(unsigned long field, unsigned long value);
extern unsigned long get_field_addr(const char *field);

struct attribute rip_a = {
        .name = "g_rip_a",
        .mode = S_IRWXUGO,
};

struct attribute rsp_b = {
        .name = "g_rsp_b",
        .mode = S_IRWXUGO,
};

struct attribute cr0_c = {
        .name = "g_cr0_c",
        .mode = S_IRWXUGO,
};

struct attribute cr3_c = {
        .name = "g_cr3_c",
        .mode = S_IRWXUGO,
};

struct attribute cr4_c = {
        .name = "g_cr4_c",
        .mode = S_IRWXUGO,
};

struct attribute G_interuptibility_info = {
        .name = "guest_interuptibility_info",
        .mode = S_IRWXUGO,
};

struct attribute G_rflags = {
        .name = "guest_rflags",
        .mode = S_IRWXUGO,
};

static ssize_t kobj_show (struct kobject *kobject, struct attribute *attr, char *buf)
{
        ssize_t count;
        unsigned long value;
        unsigned long field;
        field = get_field_addr(attr->name);
        printk("vmcs:␣read␣%ld\n", field);
        value = vmcs_readl(field);
        count = sprintf(buf, "%ld\n", value);
        return count;
}

static ssize_t kobj_store(struct kobject *kobject, struct attribute *attr, const char
{
        unsigned long value;
        unsigned long field;
        field = get_field_addr(attr->name);
        value = simple_strtoul(buf, NULL, 2);
        printk("vmcs:␣write␣%ld\n", field);
        vmcs_writel(field, value);
        return count;
}

static void vmcs_sysfs_release(struct kobject *kobject)
{
        printk("vmcs:␣release\n");
}

static struct attribute * d_abt[] = {
        &rip_a,
        &rsp_b,
        &cro_c,
        &cr3_c,
        &cr4_c,
        &G_interuptibility_info,
        &G_rflags,
        NULL,
};

static struct sysfs_ops obj_sys_ops = {
        .show = kobj_show,
        .store = kobj_store,
};

struct kobj_type kt =
{
        .release = vmcs_sysfs_release,
        .sysfs_ops = &obj_sys_ops,
        .default_attrs = d_abt,
};

static int __init vmcs_sysfs_module_init(void)
{
        int ret;
        printk("vmcs:␣init\n");
        ret = kobject_init_and_add(&kobj, &ktype, NULL, "vmcs");
        return ret;
}

static void __exit vmcs_sysfs_module_exit(void)
{
        printk("vmcs:␣exit\n");
        kobject_del(&kobj);
}
```

```
module_init(vmcs_sysfs_module_init);
module_exit(vmcs_sysfs_module_exit);
```

## IX. Test

To test if we successfully finish the vmcs sysfs interface, we need go to the /sys directory, and find if there have a vmcs directory. If there have the vmcs, we need input some command to check if the system can read or write the value in the value.

### *A. Test Read*

Use the command cat /sys/vmcs/g_rsp_b to check if there have values response. If have value response, it is means we have read the sysfs file.

### *B. Test Write*

Use command echo 2 /sys/vmcs/g_rsp_b cat /sys/vmcs/g_rsp_b. if the value appear 2, it means we van change the sysfs file.

## References

[1] Intel. (2011). Intel 64 and IA-32 Architectures Software Developer’s Manual. Intel.

[2] Rachamalla, S. (2011). Kernel Virtual Machine.

[3] Aires, B. (2008). Hardware Assisted Virtualization Intel Virtualization Technology.

[4] Love, R. (2010). Linux Kernel Development