ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 4, April 2014

TECHNOLOGY IMPLEMENTATION OF COMPUTER HARDWARE IN THE VIRTUAL MACHINES

Dr. Osama Ahmad Salim Safarini

Computer Engineering Department, University of Tabuk, Tabuk, KSA

Abstract: This article deals with designing methods and interfaces for virtual devices and implementing these devices in the virtual machine (computer). The analysis of the functions performed by the virtual device, and on the basis of that analysis is generated a list of operations of virtual device interface. The purpose of this article is identification of the main features of the implementation and operation of computer hardware in the virtual machine.

Keywords- virtual machine (VM); operating system (OS); virtual machine monitor (VMM); virtual device (VD)

I. INTRODUCTION

At the present time structure of the virtual machine (VM) includes a main computer called the host, operating system (OS) of the host, virtual machine monitor (VMM), which consists generally of several components, and the VM itself, containing all the necessary devices of the modern computer. The design of VM includes in addition to developing the processor and memory virtual machines, and even the creation of virtual devices (VD) in the computer. The purpose of this article is identification of the main features of the implementation and operation of computer hardware in the virtual machine.

For X86-based minimum set of devices needed for the computer, entered in the microchips motherboard Northbridge and Southbridge [1, 2]. First of all it is necessary for computer the emulation of PCI bus. PCI Bus provides the functionality of the plug-and-play [3], which can further take advantage of virtual machine operating system. All other computer peripherals currently represent nodes on the PCI bus. First turn to such nodes is the south bridge chip. It contains the necessary devices such as interrupt controller, controllers: serial and parallel input-output ports, the controller PS / 2 connector for keyboard and mouse, controller ATA / ATAPI for connection the accumulators, and other controllers. We should also mention the video controller, which may be represented as a separate device on the PCI bus.

Most computer devices located on the PCI bus. To the PCI bus may be connected other buses for interaction with devices that support the standard plug-and-play. At present, more and more important becomes USB Bus [4].

II. DESIGNING METHODS FOR VIRTUAL DEVICES

All peripheral devices of the computer must be contained within the virtual machine. Design of devices can be based on one of the methods listed below:

1) As virtual devices in the virtual machine a device can be used from the host equipment. The device is connected directly to the input-output ports of VM;
2) as virtual devices in the virtual machine is used as a hardware device from the host equipment, but the virtual machine monitor controls information transmitted and if necessary makes adjustments;
3) part of the devices responsible for communicating with the VM can be implemented in software, while the basic functionality of virtual device may be provided with driver by the host operating system;
4) Virtual devices can be built entirely on a program basis

In fact, these methods are fully gradation of hardware to a fully software principle of building a virtual device. We are to consider the advantages and disadvantages of these methods.
The first method is presented in Fig.1.

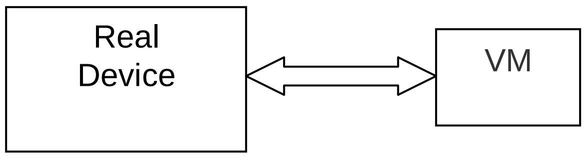

Fig.1 Connect the device directly to the VM

It suggests that a hardware device from the host equipment is taken out of the scope of the host operating system and connected to the memory address space and the I/O VM. This action is required to be performed by VMM. The control of the device is fully passed on to the OS VM. Host in the future cannot control the status of the device and the transmitted information on it.

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 4, April 2014

The advantages of the method include no reduction device performance as in this case there are no software components.
The method has the following disadvantages:
First, when the need to create multiple VMs, one and the same device cannot be connected simultaneously to more than one VM. If we are talking about a device such as a floppy disk drive, then such a device is almost always present in the system in no more than one instance.
Second, as already mentioned, when using such a method the host does not control the device and if necessary, cannot change the state of the device, for example, to limit power consumption.
The second method is a modification of the first (Fig.2).

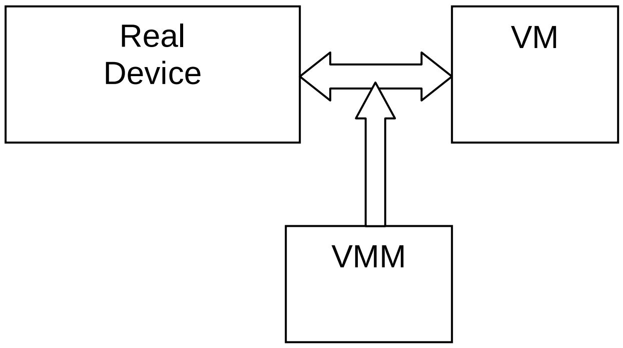

Fig.2 VMM control over plugged device

Although the device as in the first method is directly connected to the address space VM, VMM retains control of information and traffic of the hardware device.
At the side of the VMM, method requires some software processing information associated with the device. Such processing can be done using simple deterministic automaton.
The advantages to be added of the method is the ability to control the device from the VMM, however, as before, the disadvantages remains inability of separation the device between two or more virtual machines.
The third method involves maintaining functional device using the device driver OS host. VM communicates with the device not directly, but through the VMM.

VMM contains the emulator interface interaction of the device and programs by VD. VD, emulated by VMM will contain a complete set of device control registers. *The operating system* of VM can through these registers control the operation of VD. However, the actual work on the transmission of information using this method performs the real device under control of the driver OS host.
This method has several significant advantages.
First since the VMM emulates the interface device, then in the structure of VM, we can integrate as many VD$_s$ as necessary.
Second, it becomes possible for switching VD to real device only when the need arises.
Third, the VMM is able to manage power consumption of real devices.
Illustration of the method is shown in Fig. 3.

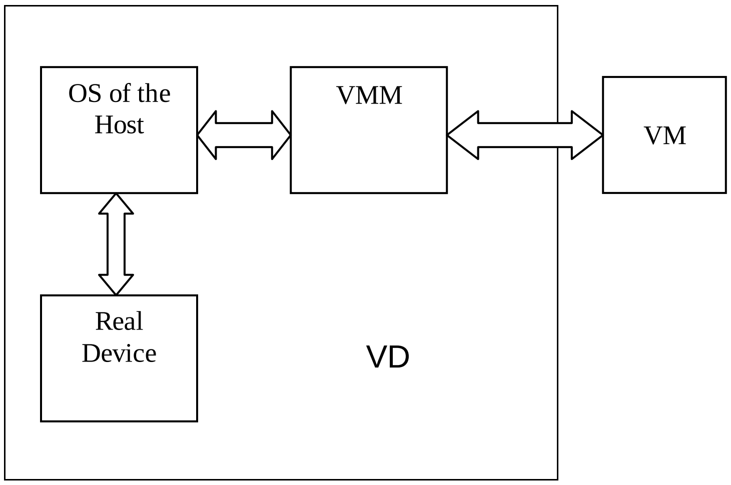

Fig.3. Connecting the virtual device by the operating system of the host

Disadvantages of the method should include a rather complex structure of the Device Emulator VMM. VMM must contain a full automate, repeating reaction of the present device. Another drawback of the method is the need for driver emulated device comprising the OS host and the hardware device itself.
Unified access interface from the side of OS host allows VMM to implement only one device of the total class devices. Thus, for video VMM it is enough to realize in VM comprising adapter VGA, during the work of VM is made reference to the same graphic card that is available on the host.

Finally, the fourth method of constructing VD is full emulation of the device by VMM software fig. 4.

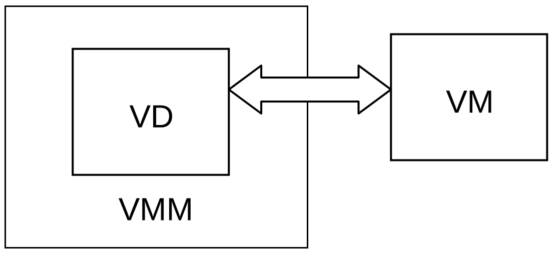

Fig.4. device emulation

Since all of the functionality of VD is emulated by CPU, the complexity of programs used for emulation grows and there are delays in the operation of the VM. These circumstances impose restrictions on the use of the method in conjunction with other methods which use real hardware devices. The delays imposed by the emulation can cause a

dissynchronization with the hardware devices used in the composition of the VM. Therefore, the method can be applied either for emulation of devices with a fairly simple operation algorithm, or by emulation of the entire complete VMs when delays in the VM no longer play a decisive role.
When developing VM, the decision of the choice of a particular method for Programming devices shall be taken for each specific device separately, depending on the structure and the work algorithm of the device, its speed, and on the basis of existing safety policy system in general. Device that makes high demands on system performance, functioning on the basis of a complex work algorithm, must be programmed using the available computer hardware. When programming simpler and slower VD$_s$, we can use full emulation. The application of methods of direct hardware control of the device by the VM is undesirable in terms of safety.

III. DESIGNING INTERFACES VIRTUAL DEVICES

Any peripheral device can interact with the computer program, executed BY processor via device interfaces. There are the following basic interaction interfaces:

1. display the device in the physical memory address space of the processor;
2. display the device in the address space of ports I/O;
3. gaining device access to the memory of the computer;
4. Setting device interrupt signal in interrupt controller and interaction through one of the buses of the computer.

This list of devices interaction interfaces is very important because it determines the construction principles of software interface VD. For VD as part of the VM, the list of base interfaces of interaction does not change. Within these interfaces, it is possible the following operations on the interaction of the VD with the processor of VM and between themselves:

1. reading from the memory of VD is mapped in the address space of physical memory;
2. writing to memory of VD is mapped into the address space of physical memory;
3. reading from the memory of VD is mapped into the address space of input-output processor;
4. writing to memory of VD is mapped into the address space of input-output processor;
5. reading of VD data from the memory of VM;
6. writing of VD data to the memory of VM;
7. the device exhibiting an interrupt signal on the peripheral bus, or an interrupt controller, or directly to the processor [5];
8. reading from the memory of VD is mapped into the address space of a peripheral bus;
9. writing to memory of VD is mapped into the address space of a peripheral bus;

Steps 5 and 6 are performed in a real computer by a device that is able to operate in the Bus Master mode, directly accessing the computer's RAM. In this mode typically operate storage media. Step 1 - 4, 8, 9 are initiated by a CPU. Step 5-7 initiated by the device itself. A device located in bus PCI, exposes interrupt signal to bus controller, bus controller exposes interrupt signal to interrupt controller, and interrupt controller sends the interrupt bus message [6] that an interrupt occurs to the processor.

Each of these operations should be implemented appropriate method VD interface. Particularly difficult in implementation are operation 5-7, that are initiated by the VD itself. For VD to perform one of these operations, it should be able to get control. If VD is implemented either by the host operating system, or by emulation method, is necessary to provide a mechanism under which VD could gain control. As one embodiment of such a mechanism can be proposed periodic timer, by which VD could gain control. In this approach, long intervals of standby VD must be programmed independently as the sum of the intervals periodic timer. Based on the foregoing, to the existing list of operations of VD is necessary to add the operation of connection to a periodic timer.

Connecting hardware devices of the VM passes the same steps as in the real computer system. Device comprising VM must first be "installed" in a VM, and then "Included" in the power supply emulation to the VM. Later the device passes configuration stage through the BIOS and OS of VM and only then starts to operate in the normal mode.
Shutting down VM goes through the stages that are reverse steps of starting the VM. First the driver comprising OS VM exits the device and resets its internal buffers. Then OS VM through ACPI BIOS turns «off» the device [7]. At the last stage, there is a "check-out" of VD from VM. This step is necessary for deinitialization VD (cleaning lists, freeing memory, etc.). It should be noted that during operation of the device comprising a real computer, and the VD comprising a VM could be several times turned "on" and "Off". These operations are performed at the request of the OS, the computer power manager.
Thus, to the above mentioned list of operations with VD, are added the following operations:

1. installation of VD in the VM;
2. Turning on VD;
3. Turning off VD;
4. Removing VD from the VM.

These operations must also be implemented as a VD interface methods.

Further expansion of the VD interface associated with the alignment of devices in the VM in ordered lists. Thus, PCI device must be contained in a set of other devices with the same interface. Furthermore, if the VM does not contain the PCI bus, all devices on a given bus does not have to go through the initialization, i.e. during "installation" the VD in the VM it should be able to receive information about VM configuration, enquiring about this VMM. To the list of VD operations, we should add the following operation: Request VMM about VM configuration.

Finally, every VD in the composition of the VM in the process of functioning performs actions that require memory. Requests for allocation and deallocation of memory can be realized only by the VMM. These requests constitute two more methods VD interface.

If the software part of VD designed with simultaneous operation of multiple instances of VD in parallel VM, there is a need for the organization of critical resources composed in VD. Such resources and access to them can be organized on the basis of critical sections. Methods for creating, access and remove critical sections are also included in VD interface.

IV. Conclusion

Methods of designing VD as part of the VM vary depending on the degree of utilization of the real computer hardware in the moment of creation of VD. Whichever method is selected for development VD from the four possible methods, the interface of VD must be uniquely organized. It should contain a set of operations implemented as interface methods providing operability of VD comprising VM. We List once again full comprehensive list of operations:

1. reading from the memory of VD is mapped into the address space of physical memory;
2. writing to memory of VD is mapped into the address space of physical memory;
3. reading from the memory of VD is mapped into the address space of input-output processor;
4. writing to memory of VD is mapped into the address space of input-output processor;
5. reading data of VD from the memory of VM;
6. writing of VD data to the memory of VM;
7. the device exhibiting an interrupt signal on the peripheral bus, or an interrupt controller, or directly to the processor [5];
8. reading from the memory of VD is mapped into the address space of a peripheral bus;
9. writing to memory of VD is mapped into the address space of a peripheral bus;
10. connection to a periodic timer;
11. installation of VD in the VM;
12. Turning on VD
13. Turning off VD
14. Removing VD from the VM;
15. Request VM configuration;
16. memory allocation;
17. freeing memory;
18. creating a critical section;
19. entrance to the critical section;
20. Exit from a critical section;
21. Remove critical section.

Combining these operations into a single set allows you to create a VD with unified interface. Currently VM manufacturers do not provide interface to VD in the composition of the VM. Creating a unified interface VD enables the design of a large list of different VD, corresponding to their real analogs. Currently author's group is developing prototype software controlling the operation of the VM, which allows showing all merits of the proposed methods.

References

[1]. Intel R _ 945G/945GZ/945GC/ 945P/945PL Express Chipset Family. Datasheet. — November, 2007. Intel Corporation (www.intel.com).

[2]. Intel R _ I/O Controller Hub 7 (ICH7) Family. Datasheet. — April, 2007.Intel Corporation (www.intel.com).

[3]. Anderson D., Shanley T., PCI system architecture. — 4th ed. — N.Y.: MindShare Inc. Addison-Wesley, 1999. — 788 p.

[4]. Enhanced Host Controller Interface Specification for Universal Serial Bus. Revision 1.0. — March 12, 2002. Intel Corporation (www.intel.com).

[5]. Intel R _ 64 and IA-32 Architectures Software Developer's Manual. Vol. 3A: System Programming Guide. Pt. 1. — September, 2008. Intel Corporation (www.intel.com).

[6]. 82093AA I/O Advanced Programmable Interrupt Controller (IOAPIC). — May, 1996. Intel Corporation (www.intel.com).

[7]. Advanced Configuration and Power Interface Specification. Revision 3.0b. — October 10, 2006 (www.acpi.info).

BIOGRAPHY

Dr. **Osama Ahmad Salim Safarini** had finished his PhD. from The Russian State University of Oil and Gas Named after J. M. Gubkin, Moscow, 2000, at a Computerized-Control systems Department. He obtained his BSC and MSC in Engineering and Computing Science from Odessa Polytechnic National State University in Ukraine 1996. He worked in different universities and countries.