MVM/DOS personality

This is an infrastructure for running VM's with unmodified OS's

MVM server

The MVM server is a central server of the MVM personality – the infrastructure for running multiple virtual machines on top of L4 microkernel. It is almost separate from OS/2 personality, but can be controlled by OS/2 programs via DosOpenVDD/DosCloseVDD/DosRequestVDD API's.

So, the MVM server exposes some interfaces to other OS personalities to be controlled by them. Also, it starts VM's, which are executed in the context of a VM monitor, running a guest OS (DOS, for example). The VM environment is defined by the VDD's [1)] loaded. The MVM server loads VDD's, which are a kind of plugins. Also, it exports some helper API's for VDD's via MVDM.DLL.

Virtual Device Drivers (VDD's)

A VDD is like a plugin for MVM Server, and it can communicate with outside programs via request API (DosRequestVDD for OS/2 programs). On the other side, from the VM point of view, the VDD emulates some BIOS services/Option ROM's, a hardware devices of guest platform. It installs interrupt handlers, catches I/O ports or MMIO registers accesses from inside the guest OS. A VDD implements its services based on VDD helpers API, served by the MVM server.

Originally, the MV(D)M were used to emulate a 8086 machine with BIOS and DOS/Win 3.1. But now it is a trend for many OS'es to have the kernel virtual machines, like kvm in Linux or VirtualPC in WinNT. Our MVM personality is our solution of the same kind, but it not so monolithic like qemu/kvm – it decomposed to several parts, in best IBM solutions design traditions. The VDD's are like plugins for MVM server, allowing to extend the MVM environment. They're loadable modules (DLL's).

Also, as we're told previously, to share the screen with OS/2 apps and apps of other personalities, IBM created the solution of “Seamless WinOS/2”. This is the possibility for Win 3.1 programs to share the same screen with OS/2 PM ones.

This is done with a special VDD's, like VVIDEO (VVGA, VSVGA, etc) implementiing a video mode support in a DOS window (or fullscreen). The DOS window is implemented as a special DLL (pmvdmp.dll) which was loaded by PM. It communicates with a VM via DosRequestVDD and implement the video functions via GPI calls (for windowed mode, so it is a PM application, based on VIO Shield (pmviop.dll), working via BVH drivers, like bvhwndw.dll for windowed OS/2 and DOS sessions, or bvhvga.dll+bvhsvga for fullscreen OS/2 or DOS sessions).

For windowed WinOS/2 sessions, it existed the solution of using a so-called PM shield (seamless.dll) and WinOS/2 shield (winsheld.exe). The 1st one is an “avatar” of Windows application in OS/2 PM world. And vice versa, the WinOS/2 shield is a representative of a PM app in WinOS/2 world.

Also, the second solution exists, based on GRADD video driver model. It works via VVMI (vmanwin.sys in the Intel OS/2). It is the VDD related to the communication of windows video driver (ifgdi2vm.drv for fullscreen, isgdi2vm.drv for seamless mode) with GRADD's VMAN [2)]. The special thread in VMAN polls the VVMI driver to communicate with Windows driver. So, the Windows driver is a generic one,

but it communicates with a “real” driver. This results in WinOS/2 and OS/2 PM shared the same screen using access to a common video driver. See the GRADD-related section for more details about multiple graphics engines sharing the same screen/video driver.

Virtual Machine Monitor

The VMM [3] is a program implementing the environment of guest hardware platform, with help of VDD's. It maintains the contexts of all VM's, and handles the traps redirecting them to the needed VDD, loads the IST [4] for different processor instruction sets, utilizes the hardware emulation features of the CPU, like VM86, AMD SVM, Intel VT-x etc.

It maintains the address space layout of a VM application, loads a firmware (for BIOS, the SeaBIOS can be used) and DOS emulation kernel.

The DOS emulation kernel (doskrnl)

The DOS emulation kernel is a special rehosted DOS kernel working via OS/2 (or PN [5]) services. For example, file system API's of int 21h are implemented via OS/2 (or PN) file API's. Access to OS/2 API's done via SVC API (supervisor call?) which is trap of HLT instruction followed by call number and inverted call number.

On doskrnl start SS:BP contains some structure which contains initial information for DOS kernel. Exact structure format not know.

Offset	Size	Description
0	2	First free segment after DOSKRNL
2	2	Size of memory - first free segment
4	2	Size of init area
????	???	Unknown
22	4	Pointer to linked list of VDD

Memory map on DOSKRNL start:

Init area	[BP+0+BP+2-BP+4]:0
Free	[BP+0]:0
DOSKRNL	60:0
CMOS Data	40:0
Interrupt vectors	0:0

As in most DOS kernels, DOSKRNL moves initialization code to higher memory using init structure. But uses init structure, not BIOS INT 12h, for memory information.

VDDs linked list (in init area) in standard DOS Device drivers format

DWORD	Pointer to next device (Must be set to -1 for last device)
WORD	Attributes
	Bit 15 = 1 if char device 0 if blk
	if bit 15 is 1
	Bit 0 = 1 if Current sti device
	Bit 1 = 1 if Current sto output
	Bit 2 = 1 if Current NUL device

WORD	Pointer to Device strategy entry point
WORD	Pointer to Device interrupt entry point
8-BYTE	character device name field. Character devices set a device name. For block devices the first byte is the number of units

DOSKRNL search and initialize XMS driver and moves some parts to HMA.

Instruction Set Translator (IST)

The IST is a DLL, emulating the instructions of Guest hardware via Host CPU instructions. It exports a set of entry points, each corresponding the emulated instruction.

The similar component exist in QEMU – but it is linked statically with the emulator binary.

VM86 on Intel, and Hardware-assisted virtualization

Some processors implement special compatibility modes (VM86 allows creation of special task in protected mode, which emulates a virtual i8086 processor) or special instruction to assist the Virtual Machine Monitors creation. (Like “hypercall” to change context to a hypervisor, to execute its service and exit “hypervisor” mode). Also, the very new processors implement the IOMMU (a hardware support for sharing a hardware).

These extensions can be used to run unmodified OS'es on top of a hypervisor (it is supported in newer versions of Xen, VBox, VMWare, VPC).

Microkernels as Hypervisors

Microkernels and Hypervisors are very similar things. Microkernels implement similar features. For example, the Fiasco.OC microkernel supports SVM and VT-x and allows to run unmodified Linux in very thin VM's. This feature can be utilized in our MVM personality too.

1)
Virtual Device Drivers
2)
Video Manager
3)
Virtual Machine Monitor
4)
Instruction Set Translator
5)
Personality Neutral