Architectural ideas behind OS/2 personality

Compatibility with OS/2 (Intel)

osFree combines OS/2 (Intel) ideas with those from OS/2 (PowerPC). It must run programs using OS/2 API, both console and PM ones. So, on Intel architecture, it must be binary compatible with IBM's OS/2. But on other architectures it will have different ABI [1)], but the same portable API [2)].

Also, like it was in OS/2 (PowerPC), the binary compatibility with OS/2 (Intel) is possible (even on non-Intel architectures). It includes the translator of PowerPC instructions to Intel ones. The Instruction Set Translator (IST) is a special DLL. It could be made on the base of QEMU binary translation.

LX executable format is unlikely to be used on other architectures, because it is tied to the Intel one. So, most possibly, the ELF format will be used. It is cross-platform, extensible and simple. It supports both a 32-bit and a 64-bit variant. IBM Microkernel used ELF too. And more, IBM used an extended ELF format with special binary sections semantics. It supported inline resources, imports and exports, for example. So, it was possible to use UNIX-style shared objects, as well as OS/2-style DLL's.

So, the userland must be compatible with OS/2 (Intel). This includes some support of 16-bit applications. Although there is a very little of pure 16-bits apps (like cmd.exe and hiew.exe), but some 16-bit API's are widely used today. For example, Kbd/Mou/Vio API's (I.e., Console API) are still 16-bits in OS/2 (Intel), though on OS/2 (PowerPC) they are made 32-bits, because binary compatibility makes no sense on other arch's (only source one makes sense).

TODO: Thunking. Getting rid of thunks in 32-bits executables calling the 16-bit API's.

Kernel-related parts of OS/2

The most of 'kernel' area was decided to be moved to userland of L4 microkernel. So, some services, commonly meant as kernel parts, like filesystems, drivers, memory management, thread scheduling etc. are moved outside the kernel. So, they are not significantly differ from usual applications, and confined inside their address spaces.

The OS/2 personality main server (or simply "OS/2 Server")

Filesystem server

Installlable file systems (IFS)

Filesystem router

IFS helpers

Exec server

Virtual Memory Arenas management

Instalable eXecutable Formats (IXF)

Shared memory management

Multiple Virtual Machines (MVM) server

A historical note

The term “MVM” goes from OS/2 (PowerPC). It replaces the term “MVDM” (Multiple Virtual **DOS** Machines). The PowerPC platform could not so easily emulate the Intel processor real mode, so it required more emulation. Though, we must say that it worked just fine, fast, and theoretically may be used not for DOS only. (IBM had plans to provide a binary compatibility with OS/2 (Intel), on the base of the same component which was used to run DOS Apps).

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 VDD's [3] 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 bhwndw.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, isgdi2m.drv for seamless mode) with GRADD's VMAN [4)]. 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 [5)] is a program implementing the environment of guest hardware platform, with help of VDD's. It handles the traps redirecting them to the needed VDD, loads the IST [6)] 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 DOS kernel working via OS/2 (or PN [7)]) services. For example, file system API's of int 21h are implemented via OS/2 (or PN) file API's.

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 correspoinding 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 Machines 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.

Other OS's personalities

"The one Ring to Bind Them all" (OS/2 as an integration platform for different types of applications)

The current OS/2 personality prototype

osFree PM

The osFree PM is an osFree version of FreePM, which was began by Evgeny Kotsuba, and then abandoned. After that, we did some changes to it, and called it “osFree PM”, to avoid name collision in case Evgeny will continue his initial version.

- Docs about FreePM, by Evgeny, now inaccessible
- FreePM on SourceForge Forum and code repository
- Documentation on PM implementation based on FreePM
- Some parts of FreePM docs written by Evgeny.

Graphical Program Interface (GPI)

The GPI [8)] is the graphical engine of Presentation Manager. It is based on PM GRE [9)]. The GPI/GRE is the counterparts of Windows GDI. The GPI/GRE pair is designed as an enhanced version of Windows Graphics Engine. Contrary to Windows, they are decomposed to two layers. The GPI is the high-level layer.

The Windows programs API's operate directly on DC [10)]. OS/2 PM is redesigned, so Programs operate on PS [11)], not the DC. The DC is something related to the instance of graphics device (the video screen, a window or printer).

The PS is a higher level abstraction. It can be treated as a canvas in other graphics libraries. It

maintain such things as current background/foreground color, a palette, a brush shape, a current font etc. Also, the PS can be associated/deassociated to/from the DC, or migrate from one DC to another. This gives a flexibility to graphics API.

The GPI functions have the “Gpi” prefix and reside in PMGPI.DLL. They are buillt on top of GRE functions.

Graphics Runtime Environment (OS/2 PM GRE) and Presentation Drivers

The GRE is a low-level graphics API which operates DC's. It handles all kinds of graphics deviices like video cards and printers.

The GPI contains an array of pointers called the Dispatch Table. This table stores pointers to the most lowlevel GRE functrions.

The graphics device drivers are called also “presentation drivers”. The presentation drivers get an instance of Dispatch Table, and hook in GRE by installing their own functions pointers in the Dispatch Table, and preserving the old ones.

The GRADD model

Video Protected Mode Interface (PMI)

VIO/KBD/MOU (Console API)

VIO and BVH's (Base Video Handlers)

Framebuffer Interface

Virtual Framebuffer over GRADD driver

BVH and PM GRE on top of a Framebuffer interface

The current osFree PM prototype

1)

Application Binary Interface

2)

Application Program Interface

3)

Virtual Device Drivers

4)

Video Manager

5)

Virtual Machine Monitor

6)

Instruction Set Translator
7)

Personality Neutral
8)

Graphical Program Interface
9)

Graphics Runtime Engine
10)

Device Context
11)

Presentation Space