2008

Virtual memory problems under OS/2 and eComStation

Sjoerd Visser

Subjects

- Memory
- Virtual memory
- Memory protection
- Memory addresses
- Paging and working set
- Virtual memory confusion
- Virtual memory models of OS/2 and eCS
- The need of physical memory in the OS/2 past
- The lack of usable virtual address space under eCS now
- Preventing out of virtual memory errors
- Optimally using the system arena
- The future?

Memory: some definitions of terms

• **Memory**: where data is stored (RAM, disk, tape, brain, paper, etc)

• **Computer memory**: data that can be retrieved by the processor.

• **Primary storage** (**main storage, working memory**): The contiguous Read Only Memory (**ROM**) and Random Access Memory (**RAM**) that can be accessed and addressed directly by the **processor** via the memory bus of the chipset of the motherboard.

• **Secondary storage** is accessed indirectly by the processor via the **drivers** of the **Basic Input Output System** (**BIOS)** and **Operating System** (**OS**) to the interface of the peripheral device (disk, tape) on the bus. Of course drivers and buffers of the secondary storage devices must be available in primary storage to make them usable by the processor.

• **Virtual memory** just consist of a range of **logical memory addresses** that are made available by the operating system (OS) to a user program. The content is not stored in the logical address,but in a physical address elsewhere (much like postboxes).

Why virtual memory?

• Programs needed more memory for data and code.

 + Secondary storage is abundant and cheaper than RAM.

 - But the processor has to wait for tape or disk most of the time.

 - Programming for tape or disk overlay technique's is difficult .

<u>Virtual memory succeeded in a multitasking environment.</u>

• **The operating system** provides an **API** for virtual storage/memory control.

• Centralized virtual memory management allows for better **memory protection**.

• Developers can work with a simple **flat memory model**.

With **priority based pre-emptive multitasking** and interrupt driven DMA controllers the processor does not need to wait, because during slow I/O operations other memory resident high priority tasks can be done.

A proper **page swapping** algorithm guarantees that only the most needed code and data will be in this sparse physical **working memory**.

So the most precious resources of most computers, RAM and processor time, can be optimally used.

**Protected mode processors were needed for memory protection.**

To make virtual memory work the help of the processor was needed.

• A program requests virtual memory via a call to the system API.

• The OS checks which RAM addresses can be made available. If needed it swaps some other code or data to disk. The OS provides the addresses.

Problems here:

• But how can the OS check if his provided addresses are really used? Maybe a program asks for much more memory than it needs.

• By overwriting another programs addresses a program could corrupt data and code of other programs.

• By overwriting the addresses of the operating system a program (or virus) could corrupt the whole operating system: The trouble with DOS.

To prevent this the OS would need to stand between user programs and the processor. But this can never be done - fast enough - without the aid of the fast hardware in the processor.

Kernel
Free memory 1
Free memory 2
Process A
Process B
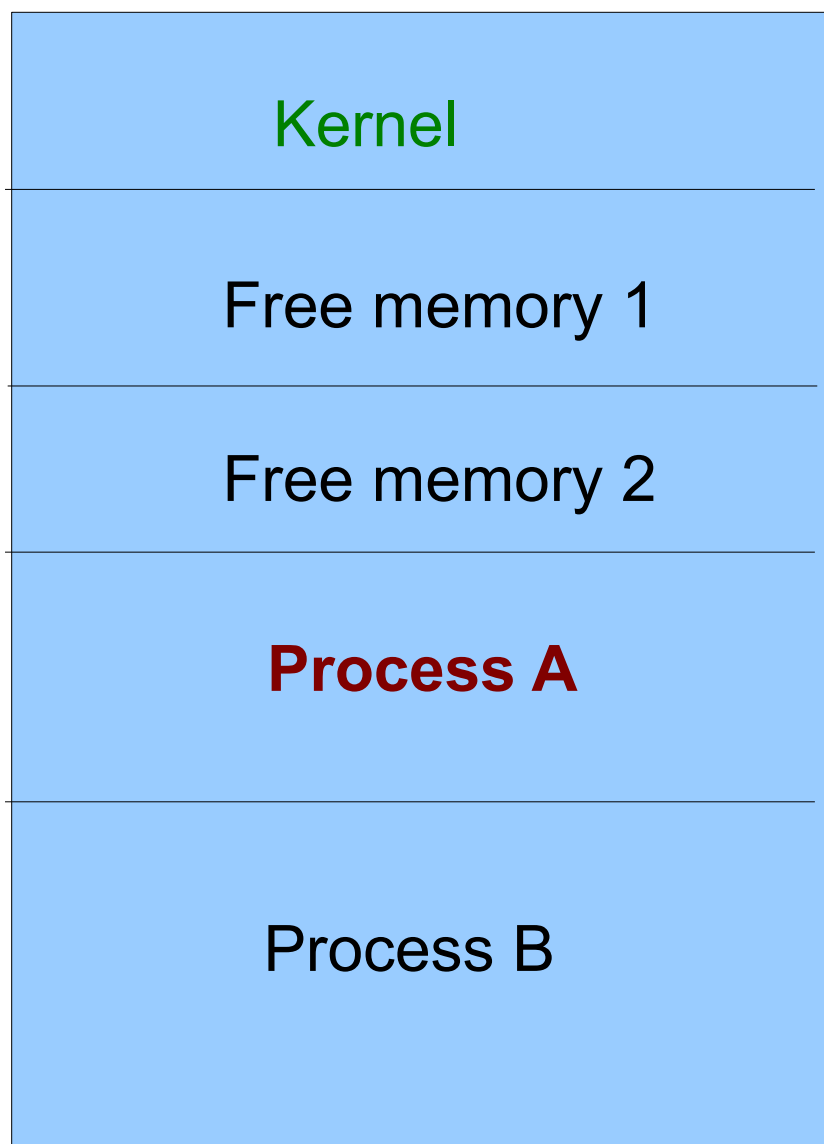
The situation without memory protection

(1) **Process A** requests memory.

(2) The OS says:

• Memory area 2 is yours.

• I will record it.

But what will Process A do if it has free access to the processor and memory?

(3) We don't know.

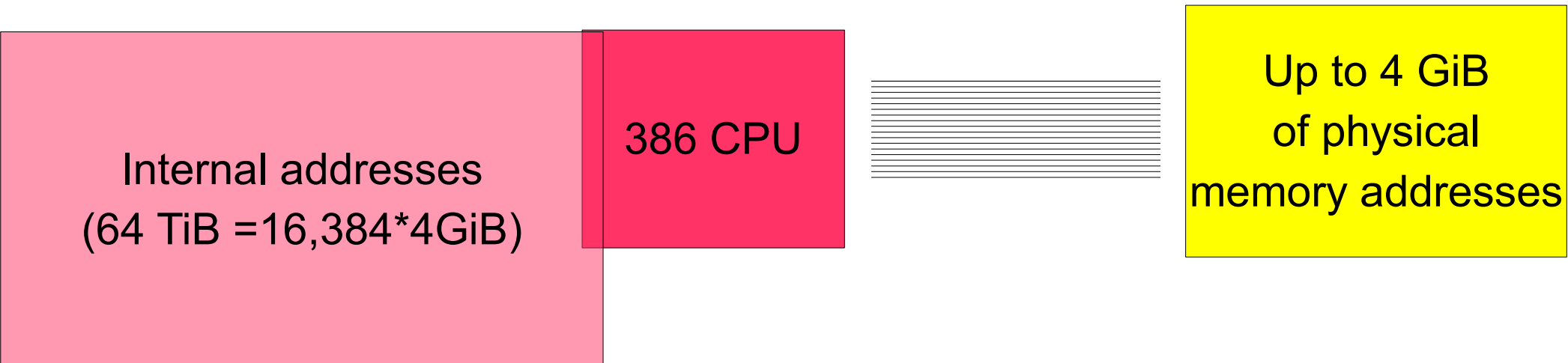• The program might be a virus or a Trojan.

• It might have an unknown bug.

A program that has free access to the processor could do anything.

It can access all memory and modify anything.

Internal and external memory addresses

Virtual memory technique makes use of the fact that most **processors** can *internally address* much more memory then they can do *externally*: via the lines of the memory bus of the chipset.

• The by **OS/2 1.0** used **Intel 80286** processor supported externally **16 MiB** (2^24) of physical memory via 24 address lines on its memory bus along with the 16 bit PC/AT data bus (used by the drivers of secondary memory), but handled internally already **1 GiB of virtual memory**.

• The by **OS/2 2.0** and **eComStation** used fully **32 bit Intel 386** (**DX**) processor and successors addressed externally **4 GiB** (2^32) of physical memory, but internally addressed **64 TiB of virtual memory**.
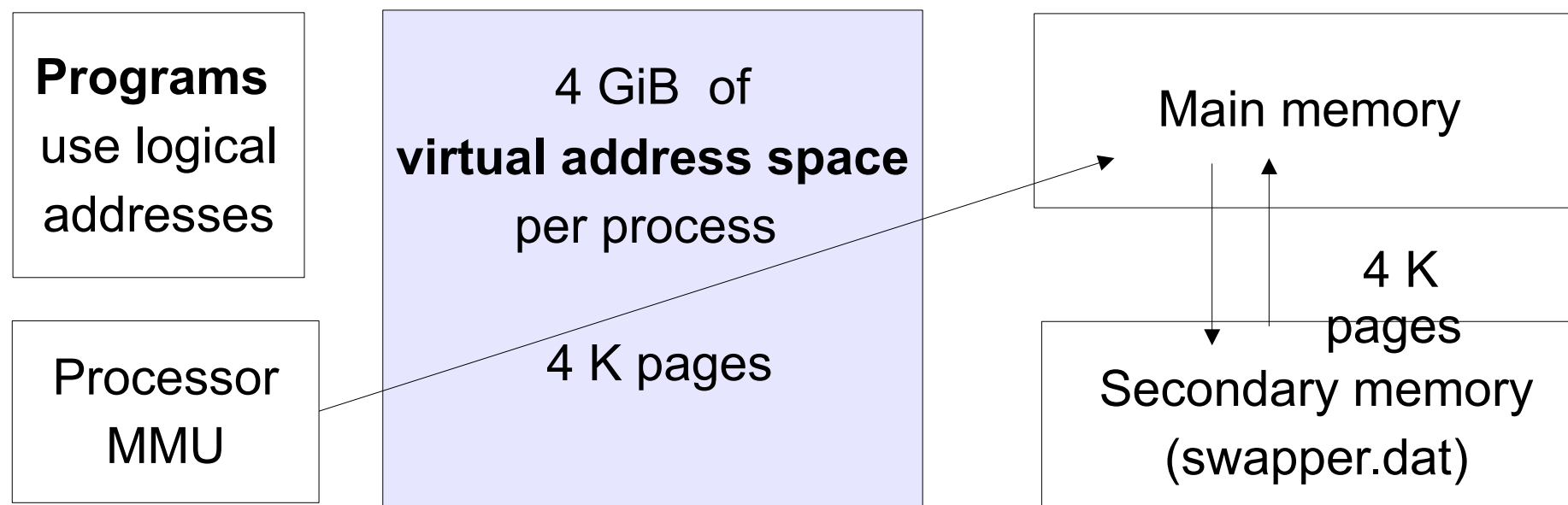
Internal addresses
(64 TiB =16,384*4GiB)

386 CPU

Up to 4 GiB
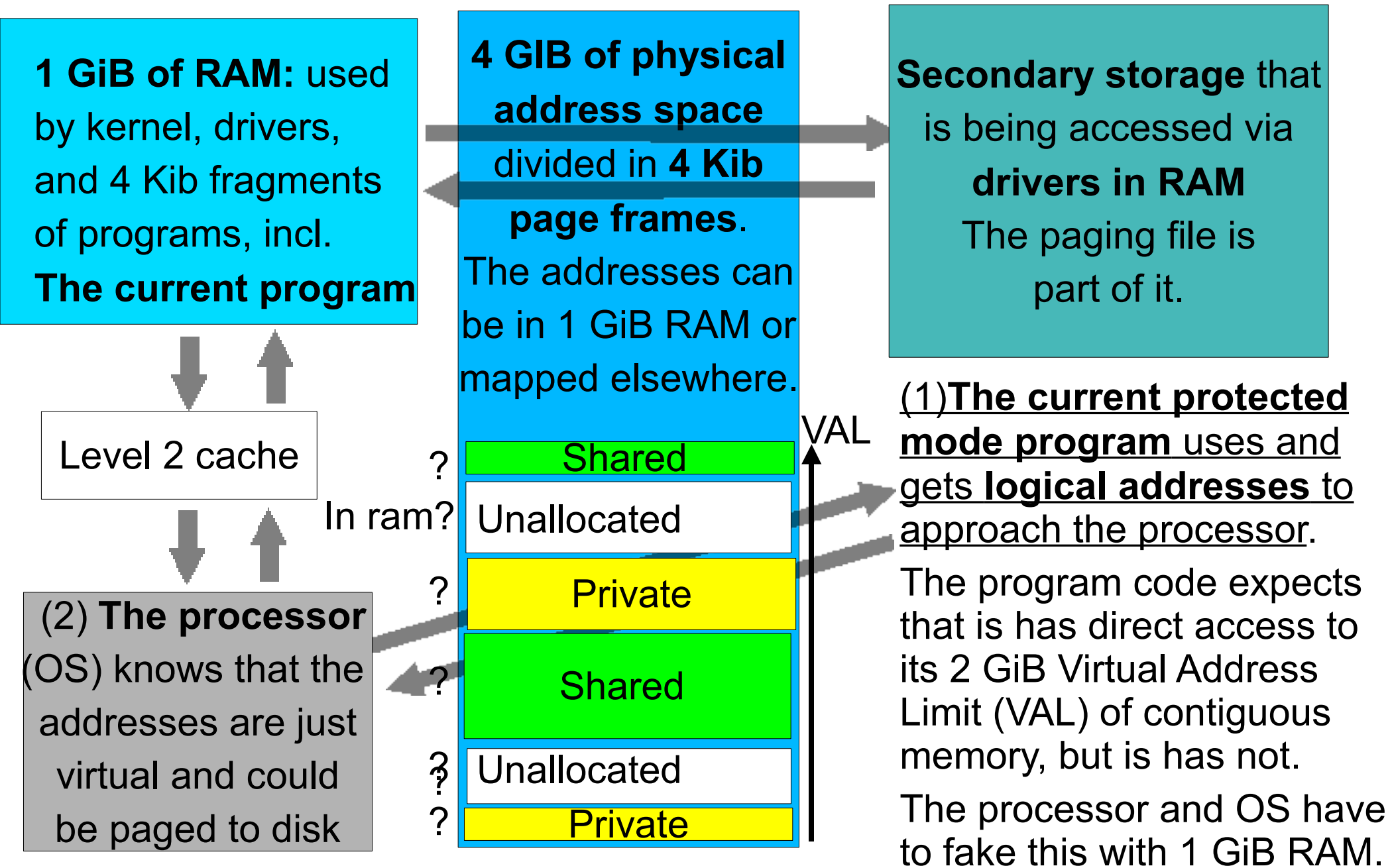of physical
memory addresses

Virtual memory makes use paging

Programs running **in protected mode** can only access its memory via the **processor** using **virtual (logical) addresses**.

The logical addresses are **translated** by the processor to **physical addresses** with the aid of the **memory management unit** (**MMU**).

These 4 GiB of unique physical addresses are divided in **4 K pages** that can be mapped somewhere in **main memory** or be **paged to disk**.

Programs use logical addresses	4 GiB of **virtual address space** per process	Main memory
Processor MMU	4 K pages	Secondary memory (swapper.dat)

4 K pages

1 GiB of RAM: used by kernel, drivers, and 4 Kib fragments of programs, incl. **The current program**

4 GIB of physical address space divided in **4 Kib page frames**. The addresses can be in 1 GiB RAM or mapped elsewhere.

Secondary storage that is being accessed via **drivers in RAM** The paging file is part of it.

Level 2 cache

(2) The processor (OS) knows that the addresses are just virtual and could be paged to disk

VAL

? Shared

In ram? Unallocated

? Private

? Shared

? Unallocated

? Private

(1)**The current protected mode program** uses and gets **logical addresses** to approach the processor.

The program code expects that is has direct access to its 2 GiB Virtual Address Limit (VAL) of contiguous memory, but is has not.

The processor and OS have to fake this with 1 GiB RAM.

Why does virtual memory technique work?

Virtual memory technique makes use of the fact that **only one process** (more precisely: **thread**) can have the attention of the processor during its time span, typically the n*32 ms (**time-slices**) during which it runs.

• If the by the thread needed **working set** (the per unit of time accessed memory of the thread) is already loaded in working memory, the program could run almost immediately.

• Ideally the working set of a running program should fit in the fast CPU cache and the working set of the system (incl. other programs) should fit in RAM.

• If not in RAM, memory pages have to be retrieved from the slower secondary storage by a mechanism called **memory paging**. This is done by the memory manager of the OS and results in much slower execution.

The on demand paging mechanism has many **similarities with banking**.

• When all customers demand their saved money, but the bank has not any financial liquids in house (because it is lend for mortgages or used in speculation), it must be lend from other banks. If they respond slowly, we have a **credit crisis**.

• It's equivalent to virtual memory technique is **disk trashing**.

Why is virtual memory so confusing?

Remember that the fully 32 bits Intel 386 processor had **4 GiB of physical address space**.

But my OS/2 system has **1024 MiB of physical RAM** attached to it.

So who else are connected to the 4 GiB of physical address space?

- **Your 128 MiB of working memory** will contain working set of the system.

- K**ernel, program code** and **data** that are **paged to disk.**

- **The chipset** claims unique 32 bit physical addresses for the **interfaces to hardware components** like video memory, AGP, ACPI and PCI tables. These addresses belong to the kernel.

- **Unused (unallocated) physical addresses** that can be made available by the kernel.

Next question: Who might use the 64 TiB of internal addresses?

The internal housekeeping of the kernel

A

B

C

D

E

Shared addresses

A (active)
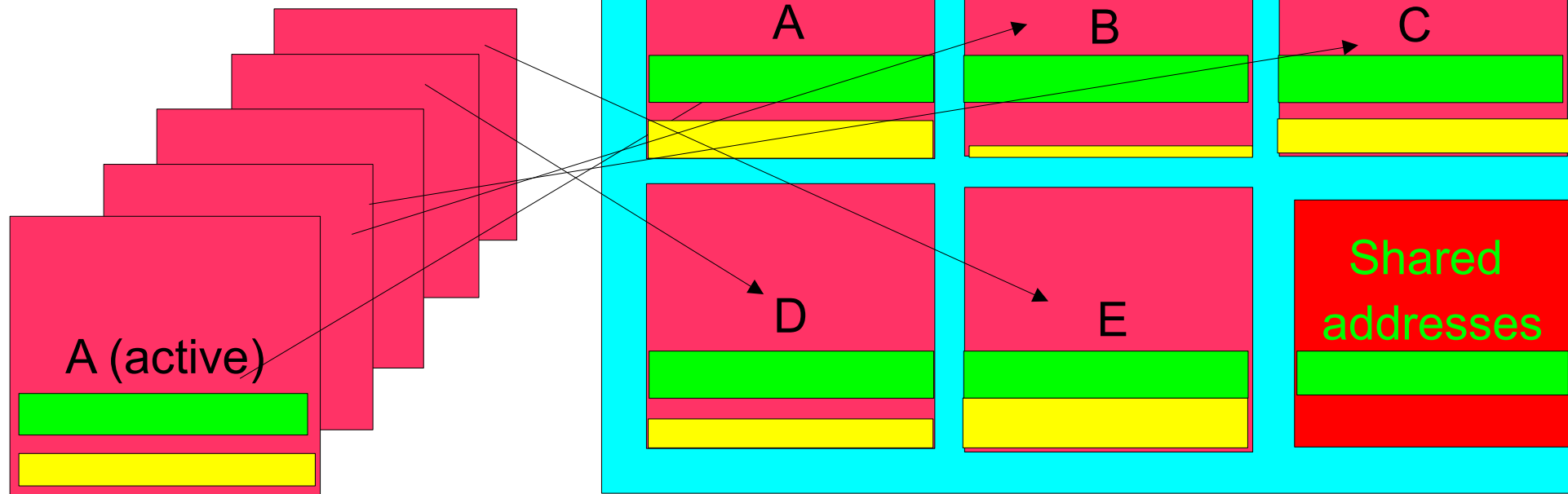
The Operating System could give all the logical addresses each process uses, an unique internal address in its kernel arena.

In so called **private arena's (yellow)** the content of the by A used logical address *x* will differ from B's logical *x*.

But when programs **share memory (green)**, the they are will share it at the **same logical address**. This makes context switching from A to B easier.

Sharing successful code and actual data in system or program dynamic link libraries between user programs in virtual memory is more economical then to let every program invent the wheel again.

The **shared libraries** (green) deliver their API functionality (like PM Windows) to programs from the same logical addresses.

The yellow program 1 sees the green shared code and data of 1, 2 and 3 at the same logical addresses as 2 and 3 see them.

But programs 1, 2 and 3 can only see their own yellow private addresses. Just as DOS on C cannot see OS/2 on the other primary partition C, but can share a logical partition D with DOS.

As the **kernel space** (blue) is out of reach of user programs, only the not shared content of the data and code in the **private arena should completely change** when a process is pre-emptively interrupted to make place for another process (**CPU context switching**). But if using lazy commit, OS/2;s lazy MemMan would say: Why and when should I take this effort? Only when it is really accessed of course! Only then the allocated memory is touched it is committed in physical memory.

Realize that the whole virtual memory address space remains the same when changing threads inside a single process. This makes cleverly designed **multithreaded programs run fast**. They need less context switching of the processor and the virtual memory manager when they launch their next high priority thread just in time.

OS/2 v. 2-4 kernel with fixed system arena of 3,5 Gb

Shared

Unallocated

1 2 3

Private arena

Under **OS/2 2-4** the **kernel** (blue) seemingly occupied 3,5 GiB address space (in fact kernel address space is much bigger than that: 64 TiB).

Protected mode user programs got **512 MiB** of **address space per process** at a time that OS/2 3 Warp power systems had 16 MiB of RAM.

The lowest logical addresses were used by the not shared DLLs and their EXE-files (**private arena**) and the higher addresses were used to load shared DLLs in (**shared arena**).

Between them was **a pool of unallocated virtual memory addresses**, that could be used to let programs grow.

But after loading and unloading of DLLs the shared free address space **became fragmented**, preventing the loading of large new DLLs.

Notice that the address space of the biggest program in the private arena determines the border between the private and shared arena.
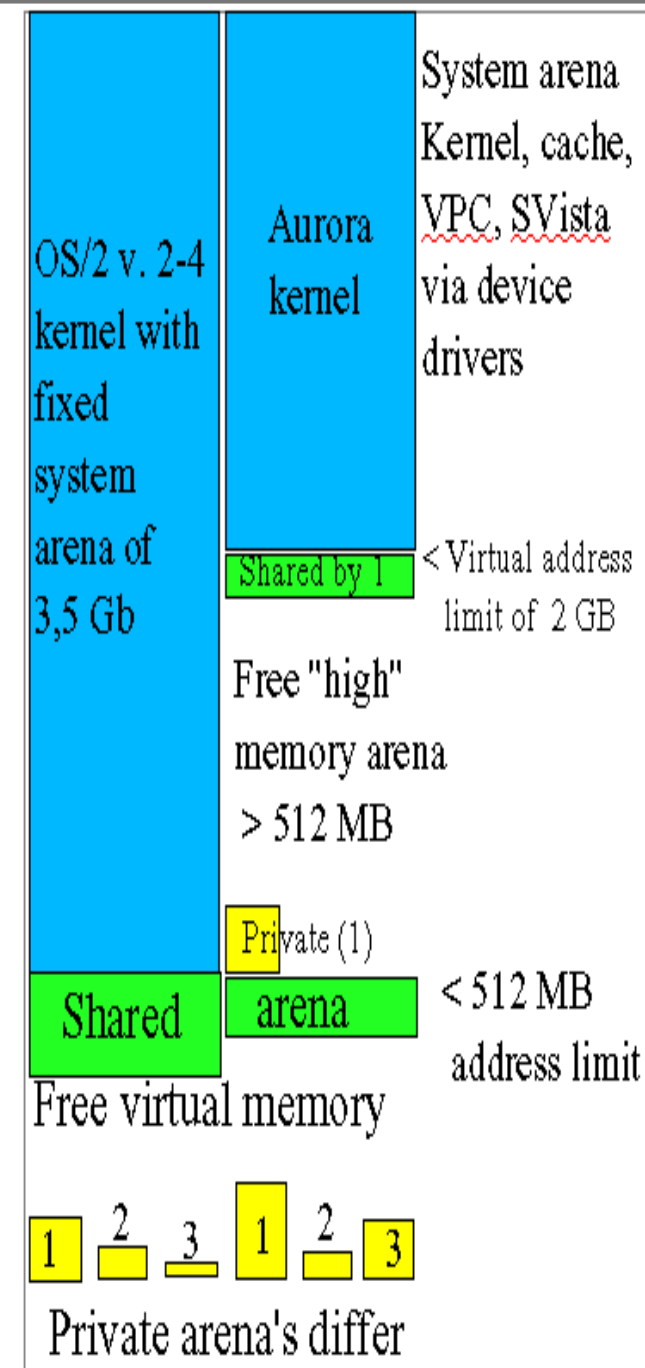
The aurora kernel of OS/2 Warp Server, OS/2 4.5 and eCS (blue) introduced **new logical address space** for specially compiled 32 bits protected programs like java in the **high memory arena** (**HMA**).

Just as their classic counterpart, the higher addresses are shared, the lower addresses are used privately and there is a pool of unallocated virtual memory addresses between them.

The address space mapped to the system is above the **Virtual Address Limit** (**VAL**), user virtual address space is below the VAL.

You set this value in the CONFIG.SYS:

VIRTUALADDRESSLIMIT=1536

Valid values are 512 (Warp 4 situation) to 3072 MiB. Practical values are between 1024 and 2048. A lower value might give loading problems of applications, higher values can prevent the loading of system buffers.

4 GB

System arena
Kernel, VRam caches, VPC, VBox via device drivers

OS/2 v. 2-4 kernel with fixed system arena of 3,5 Gb

Aurora kernel

Shared by 1

< Virtual address limit of 2 GB

Free "high" memory arena > 512 MB

Private
1   3

512 MB

Shared   arena   Fragmentation

Free virtual memory

1 2 3   1 2 3   1 2 3

Private arena's differ per process
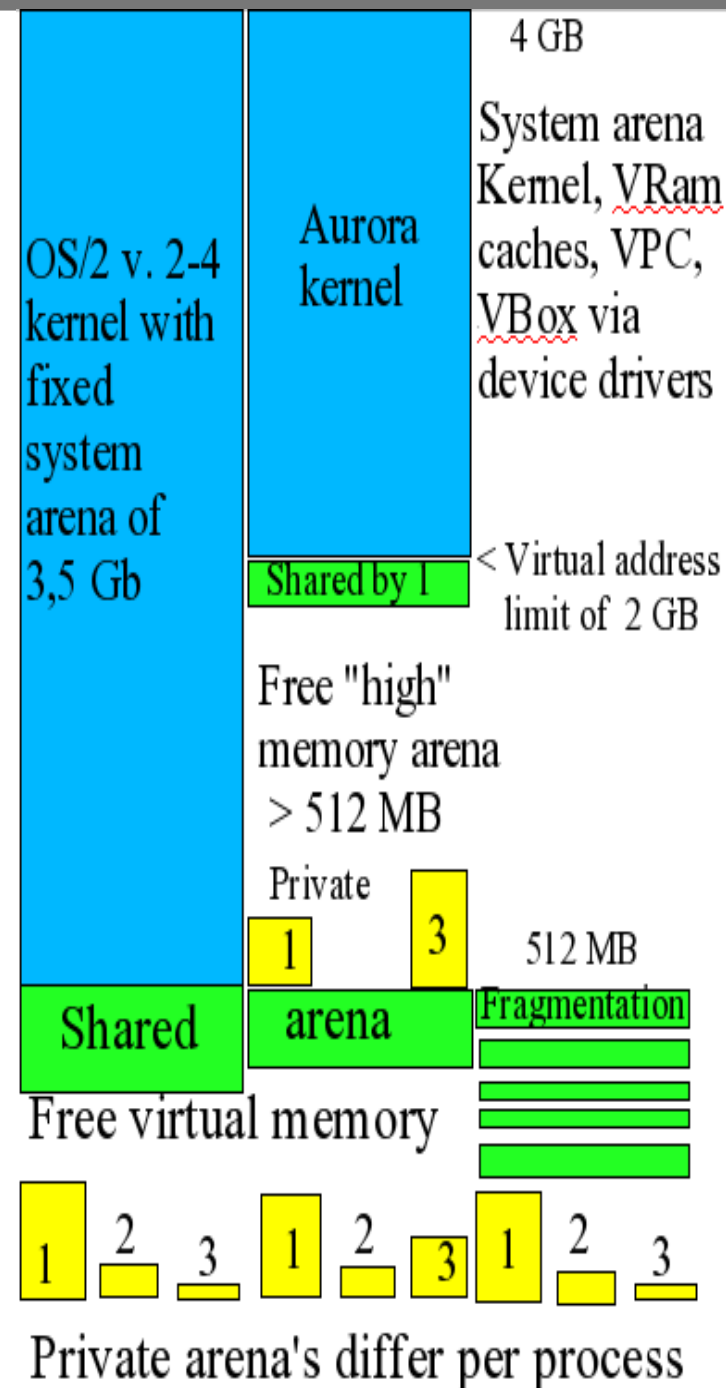
Problems with these high memory addresses

● High memory addresses can only be accessed by specifically for the **new virtual memory model** (re)**compiled programs.**

➔Some programs that load code and data party high: Mozilla, Java, OpenOffice, ODIN, PMView, most UNIX ports of Paul Smedley.

➔ The OS/2 loader will not always load the for high memory usage tagged DLL's high.

➔ Programs still need entries in the shared arena to make use of functionality of PM and the WPS.

If you have 2 GiB of RAM, don't be surprised to still have 1 GiB of RAM free, unless you use VPC or a large JFS cache.

The bottleneck: lack of contiguous virtual address space

• OS/2 was designed to give **hundred of MiBs of virtual memory to PC's with only 4-16 MiB of RAM at times that memory was very expensive**.

• Using its ingenious **lazy commit paging mechanism** (only commit virtual addresses when they are really touched), the OS/2 2.0 memory manager MEMMAN promised 480 MiB of contiguous address space to protected mode to each user programs (512 MiB to DLLs), but could keep the slow secondary storage in swapper.dat remarkably small.

• **Years later Windows 95 gave user programs a contiguous virtual address space of 2048 MiB**. Because of the success of the virtual memory technique under Windows and Linux, we now have **huge programs** that do not fit (together) in the 512 MiB OS/2 virtual address space anymore.

• With the aurora kernel the VAL *could* for certain programs be raised to 3 GiB. But as the OS/2 system **libraries** and **most user programs** reside mainly in the **300 MiB fragmented address space of the shared arena**, we cannot that easily load the current large DLLs in it as Windows or Linux can do.

Systeemfout - StarOffice 5.1 - [WSE2008 presentation template]

SYS0147

Not enough resources are available to process this command.
The process for which this pop-up is displayed will end.

Opties
→ End program/command/operation

OK   Help

This scanned picture of a **StarOffice pop-up screen** was printed on my printer by OS/2 PM using the Configuration / Screen / Tab **Print Screen** option.
Select the PM Windows and type the **PrtScrn** key. Your printer should display it
The workplace shell and PM were not accessible. So I could not use PMView.
BTW: I was working on this presentation. I had about 800 MiB RAM free.

Virtual memory under OS/2 2-4

● OS/2 2-4 promised each program (more precisely: their DLL's) a theoretically maximum of **512 MIB address space**.

● Part of it was private and part of it could be shared.

● If program 1 asks for 200 MiB private memory, it should be **allocated**. Even if only 100 MiB is used (**committed**).

● Now PM, the WPS and other shared libraries together can only allocate 512-200 = 312 MiB shared address space.

● When more memory must be allocated, PM and the WPS could crash because of lack of virtual memory address space.

● Lack of address space would not easily happen on a system with 16-64 MiB of RAM: 100 MiB committed on running WPS system would give a lot of swapping.

● But if you have 2 GiB of RAM, you have not the kind of configuration that OS/2 2.0 was developed for. 100 MiB committed is not your problem, but 312 MiB shared address space (mostly already occupied by GUI and network) is.. .

OS/2 v. 2-4 kernel with fixed system arena of 3,5 Gb

Shared

Unallocated

1   2   3

Private arena
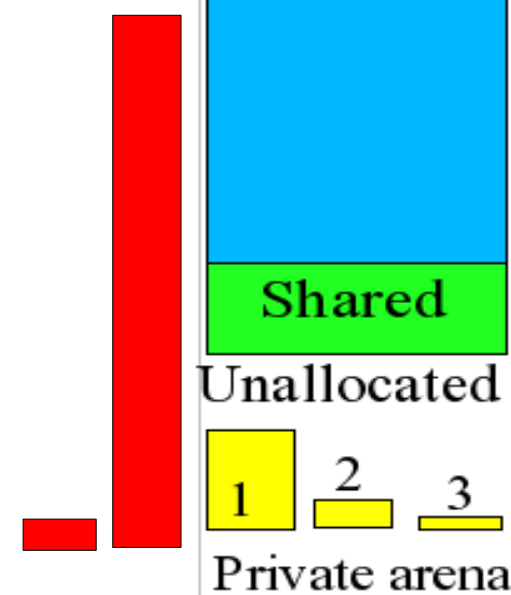
Comparison of VAL and RAM (64 MiB versus 2 GiB)

400 MiB allocated, 220 MiB committed in 4096 MiB Virtual address space

64 MiB RAM, lazy committing a relatively small swapper.dat of 150 MiB.

128 MiB RAM, less of committed memory in swapper.dat.

1024 MiB RAM, swapper.dat seldom used (free RAM).

3,5 GiB kernel space

classic

1536 MiB

With 1024 MIB RAM the problem is to find free address space to allocate virtual memory for large applications. OS/2 4.5 and eCS can have a 3 GIB Virtual Address Limit (VAL), but it is split up in two parts and most applications won't load in the high memory arena's.

high memory

2560 MiB kernel space

Top 3 of committed memory (RAM) users: probably more addresses allocated!

Open Office using this document   360 MiB in RAM

Long Time browsing with Firefox: 320 MiB in RAM

Thunderbird mail and newsgroups: 200 MiB (* system crashed with 400 MiB)

Note JFS, VPC and Vbox drivers can access much more memory in kernel space.

Monitoring the precious low virtual memory resources

I use shmemmon.exe 1 to monitor shared address space every minute.

Writing this presentation in OOOrg 2.4:

Monday 27-10-2008 19:39:15 - Available: 311 256Kb blocks =   77.8 Mb

Saving it as SDA, and closing OOOrg 2.4 frees 91.9 Mb.

Monday 27-10-2008 19:40:16 - Available: 679 256Kb blocks = 169.7 Mb

Opening this SDA-file in StarOffice 5.1 uses only 27.7 Mb.

Monday 27-10-2008 19:41:17 - Available: 568 256Kb blocks = 142.0 Mb

Opening another version in OOOrg 2.4 uses 86.3 Mb.

Monday 27-10-2008 19:46:20 - Available: 223 256Kb blocks =   55.7 Mb

Opening Approach database uses 14.2 Mb.

Monday 27-10-2008 19:48:23 - Available: 166 256Kb blocks =   41.5 Mb

Killing Firefox with Watchcat frees 106 Mb.

Monday 27-10-2008 19:49:23 - Available: 165 256Kb blocks =   41.2 Mb

Monday 27-10-2008 19:50:23 - Available: 590 256Kb blocks = 147.5 Mb

Resetting the WPS: failed. Opening CLI: failed. Reboot needed. Why?

The question: Resetting the WPS: failed. Opening CLI: failed. Reboot needed. Why?

• I had on my 2 GiB RAM system with a 720 MiB JFS cache not lack of working memory: 700 MiB was still free.

• When the system (here WPS, PM and 16 bits CLI) became unstable, I had still **590 of 256 Kb memory blocks** (**147.5 Mb**) available in the shared arena.

Monday 27-10-2008 19:50:23 - Available: 590 256Kb blocks = 147.5 Mb

• Presumably a larger than 256 Kb memory block (contiguous free address space in the shared arena) was needed, which could not "just in time" be provided by the OS/2 loader.

• Theseus would notice something like (my current status using StarOffice):

There is 0.000M between the private and shared arenas.

Shared arena starts at 04600000, which is 70.000M.

Free memory from 04600000 for 265.438M, which is equivalent to 4247 64K spaces.

Maybe I could load 500 256 KiB DLLs in the shared arena, but not a single 2 MiB DLL. This happens when there is no 2 MiB of contiguous address space. And it can occur when I have still hundreds of MiBs RAM free.

Circumstances under which virtual memory errors typically occur:

• **Enough free RAM and/or a small swapper.dat** (<< it's max of 2 GiB) on a hard disk with plenty of disk space free. So you can't blame errors on lack of physical memory

• Loading modern programs (libraries) that reserve much virtual memory. Even though their actual memory usage in RAM (working set) may be relatively small.

• Signs of **low virtual address space** or of **virtual address fragmentation** that starts with the loss of the free pool of virtual address space in Theseus:

There is 0.000M between the private and shared arenas.

• Though **shmemmon** may report that there are 590 of 256K blocks free, this does not guarantee that a single 2 MiB buffer of 8 contiguous 256Kb blocks can also be loaded.

Monday 27-10-2008 19:50:23 - Available: 590 256Kb blocks = 147.5 Mb

Fragmentation always occurs when you load, unload and reload DLLs in the virtual address space. Just like more fragmentation inevitably happens when you put files on a FAT file system, remove them and put them on the same file system again.

Note that you can still put large files on a fragmented FAT disk with enough free disk space, but programs need contiguous virtual address space. They have no file system driver that reassembles their scattered clusters to a large file again.

If after a while there is not enough contiguous virtual address space the OS/2 loader cannot load them. Programs will not load and modules loose their functionality. Killing of not responding programs may be troublesome now.

Symptoms that can be caused by lack of virtual memory:

• "Cannot find DLL" program loading errors when you start an application and the LIBPATH is right. It mostly means that a DLL could not be loaded in the shared virtual memory address space (EXE files load mostly in private arena's).

 An incompatible DLL might already be loaded (Mozilla does not start) or their might really not be enough free contiguous address space.

SYS0008: There is not enough memory available to process this command

All available memory is in use.B

• A constant beeping when starting up an application.

• Watchcat or other popping up programs forget their settings.

• WPS forgets its HOME directory (even if it is set in the CONFIG.SYS) or responds slow. Be prepared of "An error occurred.." by XWorkplace.

• Checkini cannot execute its operations. So make regular WPS backups.

• Widget libraries fail: Buttons are not drawn (seen quite often in Mozilla).

• Black PM Windows are very suspect of an approaching PM shell disaster.

• If a shell from CAD or WatchCat gives: [F:\][F:\][F:\][F:\][F:\] I had to reboot.

If system resources are low nearly all applications (modules) may loose their functionality! And if PM Shell or CMD are affected you better reboot.

- Unix servers do not fork or crash when starting.

JunkBuster: can't fork: errno = 60

- File Commander/2 fails to load with:

Memory allocation error 8

- At the same time df gives:

[F:\]df

LIBC PANIC!!

_linitheap: sbrk failed!

pid=0x01ea ppid=0x01e9 tid=0x0001 slot=0x00a2 pri=0x0200 mc=0x0000

I:\UNIXOS2\BIN\DF.EXE

Process dumping was disabled, use DUMPPROC / PROCDUMP to enable it.

Some of the chosen examples may have other causes than low virtual address space. Among them are incompatible libraries and bugs in the C library ports.

They could co-exist: with more libraries loaded there will be less unallocated address space, but also more risk of incompatibilities. The old **"DLL-hell"**.

How to differentiate between them?

Lotus Organizer does not start with 206.5 MB free.

WPS suggest to inspect the program object properties.

DLL tree says ORG32B.DLL could not be loaded.

Why? Even with only 1 KiB free this could not be lack of RAM!

But closing down some other applications will free virtual address space that will allow Organizer to start.

Another loading error because of lack of usable address space.

According to Theseus there is 128 MiB free in the shared arena, but the free unallocated address space consisted of small memory objects.
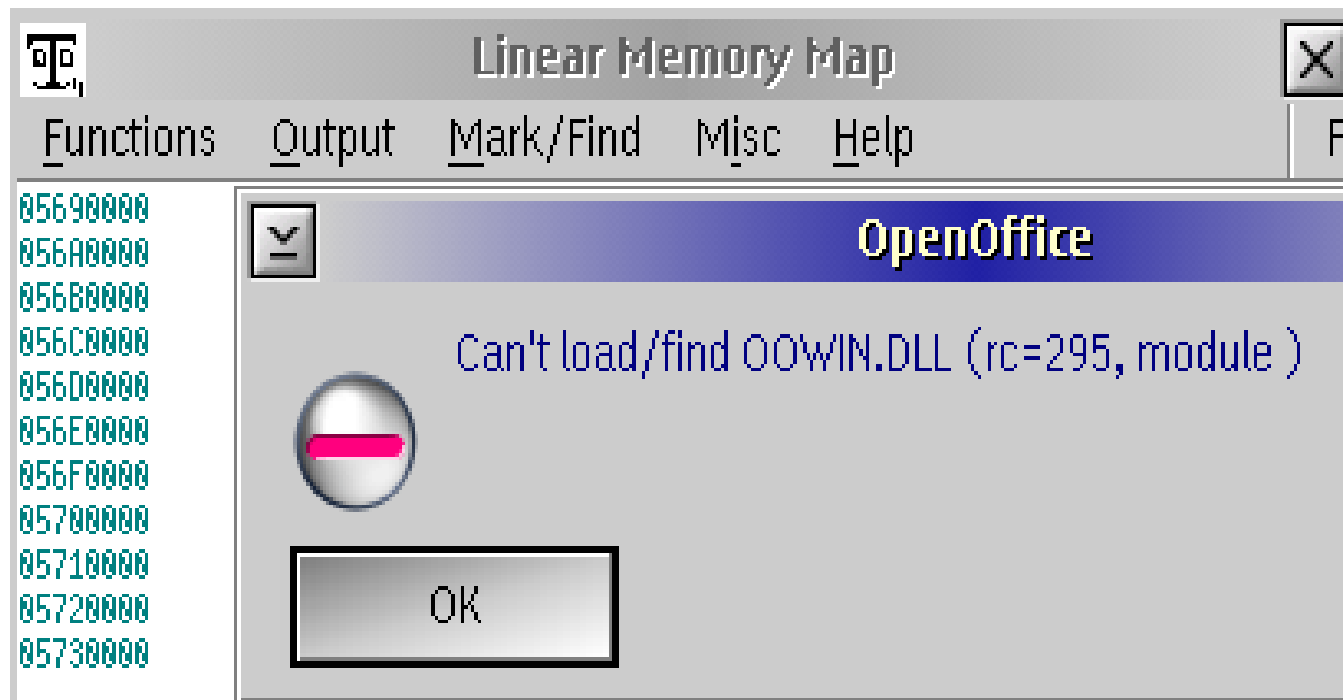

```
                     Linear Memory Map                          X

 Functions    Output    Mark/Find    Misc    Help

05690000
056A0000                                    OpenOffice
056B0000
056C0000              Can't load/find OOWIN.DLL (rc=295, module )
056D0000
056E0000
056F0000
05700000
05710000                            OK
05720000
05730000

There is 0.000M between the private and shared arenas.
Shared arena starts at 0B670000, which is 182.438M.
 Free memory from 0B670000 for 128.000M, which is equivalent to 2048 64K spaces.
13670000   .......W...WW...WW.WWWWW.WWWWWWW..W...WWWOW   1281  PMMERGE  allocated
 object is 5.063M
13B80000   .......W...WW...WW.WWWWW.WWWWWWW..W...WWWWW     16  PMMERGE  allocated
13B90000   .......W...WW...WW.WWWWW.WWWWWWW..W...WWWWW      3  PMMERGE  allocated
13BA0000   .......W...WW...WW.WWWWW.WWWWWWW..W...WWWWW     16  PMMERGE  allocated
 Free memory from 13BB0000 for 640K, which is equivalent to 10 64K spaces.
13C50000   .......................................o...      1  SOMD     allocated
 object is 7.875M
14430000   .......................................o...      1  PMCTLS   allocated
```
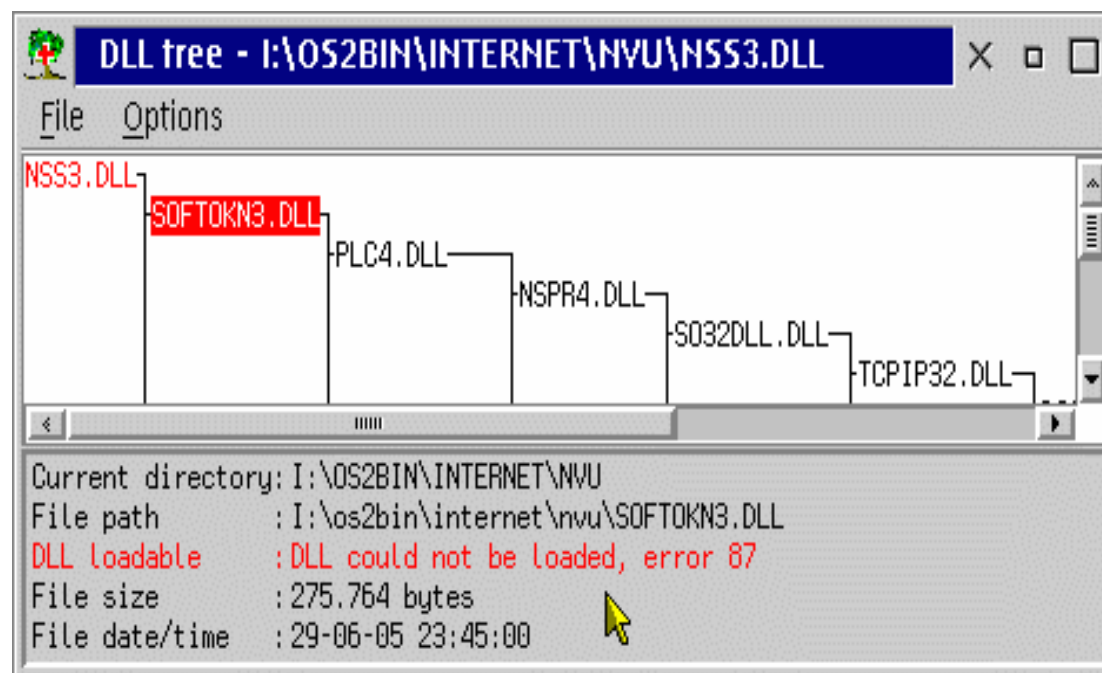
Example of incompatible libraries

NSS3.DLL does not load via NVU.EXE: '*an error occurred deeper down the tree*' says PMDLL.

An old 2005 version of SOFTOKN3.DLL, on which NSS3.DLL depends could not be loaded, with error 87.

According to Theseus this object was already loaded by a more recent (2008) Firefox 3.1 beta 1.

12A30000 ....o.   33 SOFTOKN3 #0001 (shared code)
 object is 192K

12A60000 ....i.   1 SOFTOKN3 #0002 (private)

Using Rick Walsh's run! utility NVU ran well and PmDLL showed no loading errors via NVU!!.EXE (as it did with NVU.EXE when Firefox was loaded before).

So apparently NVU.EXE could not use the against another C library compiled Firefox DLL. Indeed: This loading problem is reproducible on a fresh WPS.

Another example of incompatible libraries:

Firefox!l.exe loads, but Firefox.exe does not.

These problems can easily be solved using Rick Walsh's run! utility.
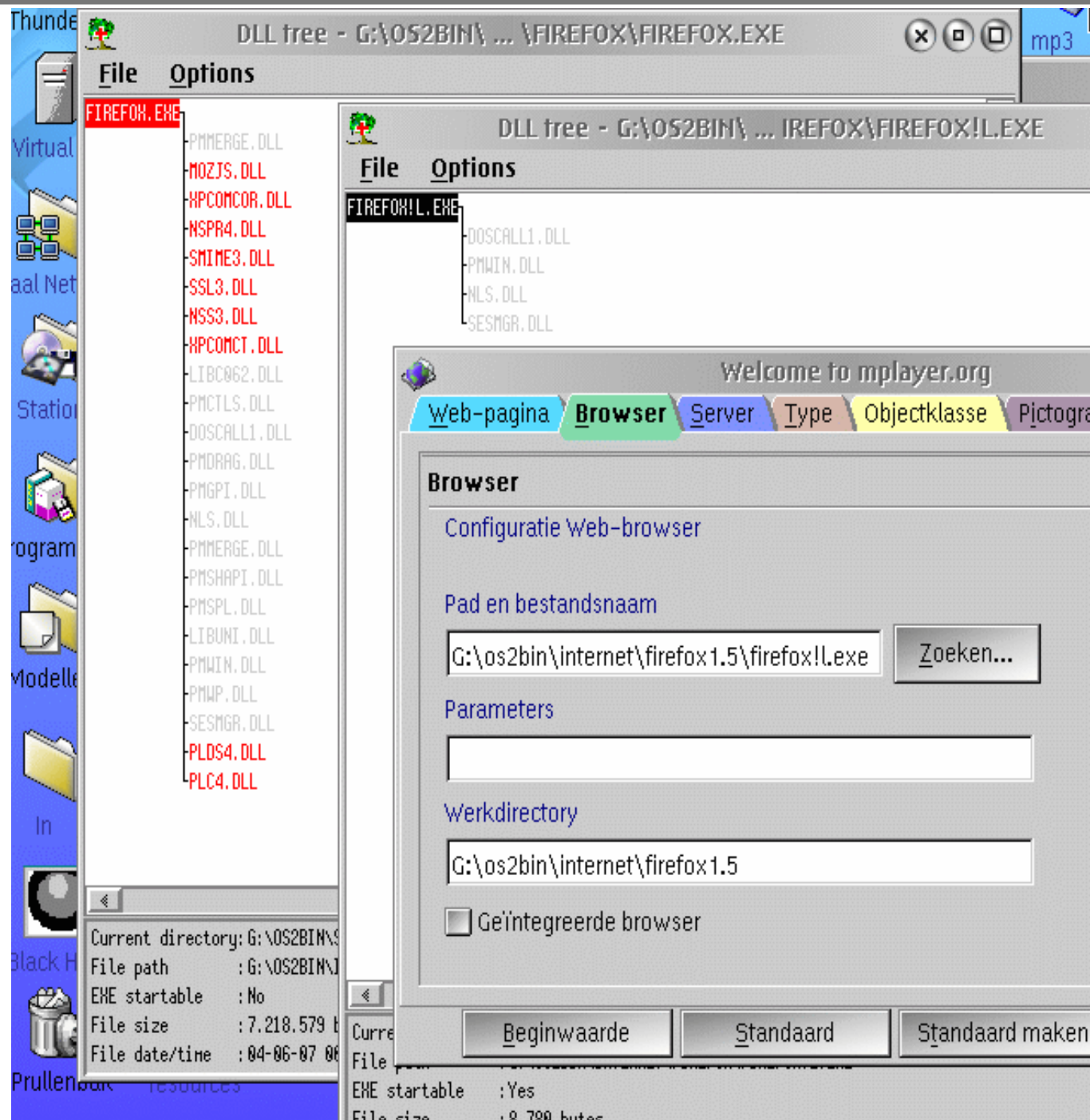
Black PM windows and **not responding applications that cannot be killed by Watchcat or CAD are serious**.

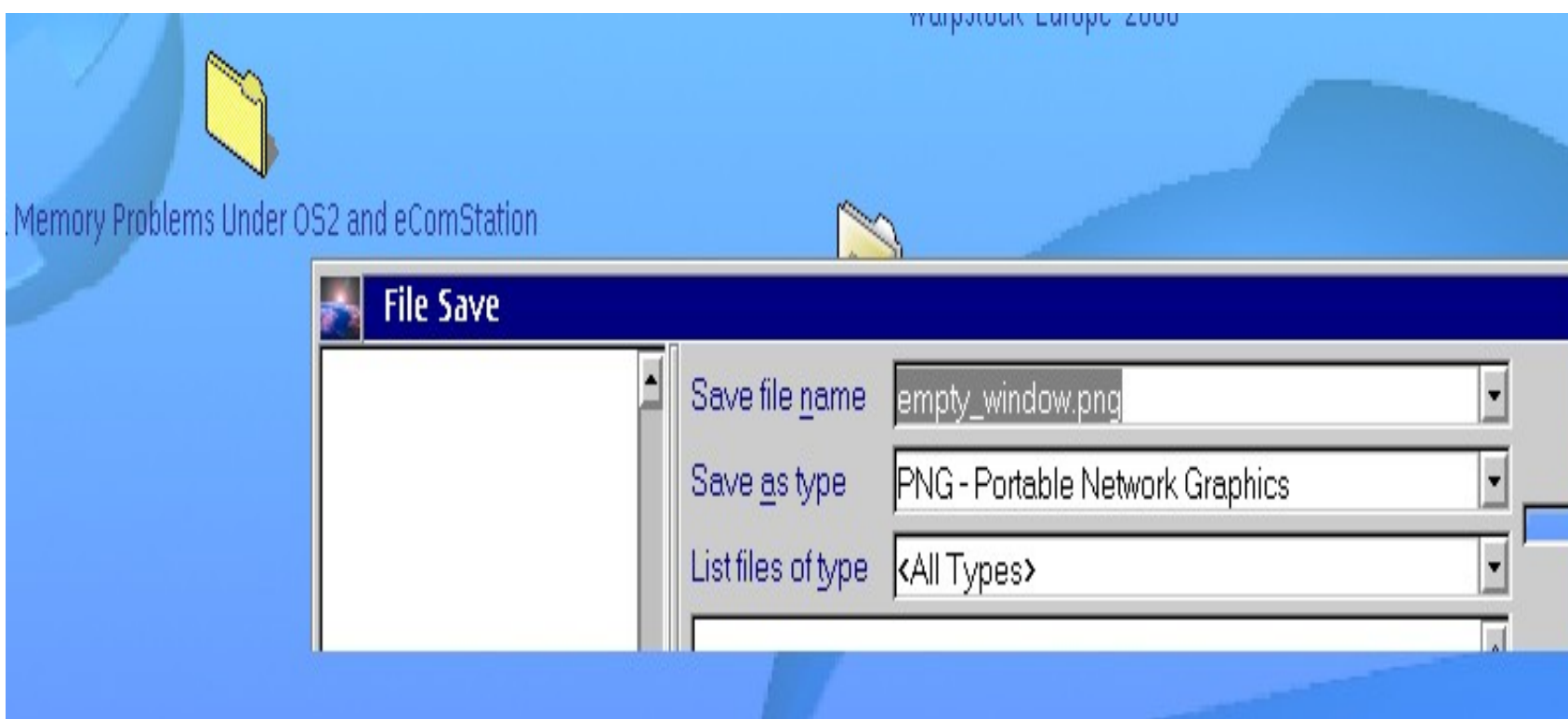AROS2.EXE hang with an transport error. It failed to respond.

The black PM windows could be Open Office. Even CMD may fail now.

In the XWorkplace taskbar you see I was trying to write about Virtual Memory errors under OS/2. And then they happened. Ugh. Picture taken with PMView. One of the programs that can use high private memory to load its pictures.

PM Windows are not drawn.

Even PM View cannot work when Presentation Manager runs out of virtual memory resources. I soon had to reboot.

Using Theseus to study memory under OS/2 - OpenOffice.org 1.1.5

Bestand   Bewerken   Beeld   Invoegen   Opmaak   Extra   Venster   Help

J:\Warp\OS2VOICE\VMonOS2\theseus.html

Tekstblok   Arial;Helvetica;sa   12

In the article <u>Virtual memory problems under OS/2</u> I made a lot of state

decided to leave the notes on virtual memory technique and Theseus o

nportant to give arguments for stateme
be nice if others could redo some of n
sical memory. Ideally, others may sugg
and solutions for the described loading problems of user programs.

These problems with font rendering might be a bug, but why does the bug disappear after I close Mozilla or other virtual memory consumers?

How to save virtual address space in the low shared arena?

● At the moment we have plenty of free addresses in high memory. The problem is how to **load programs and data high**.

➔ Users will make these logical addresses available by setting a reasonable VAL between 1024 and 2048 MiB using modern 4.5 kernels: VIRTUALADDRESSLIMIT=1536

● Programmers/porters have to (re)compile programs to make use of it. Users have to use them if they prefer the latest and biggest programs.

➔ Peter Weilbacher efforts with the Mozilla family are an example.

➔ The latest port of OpenOffice.Org 2.4 also makes use of hundreds MiB of high memory address space. Don't expect it to load in Warp 4.0.

➔ Alternatively, one can prefer to use **lean programs**. Or less lean programs that use private memory. For browsers we have little choice, but for most other tasks there are plenty of alternatives on Hobbes.

●The CONFIG.SYS setting DLLBASING=OFF is said to save 64-128 KiB.

● SET JAVA_HIGH_MEMORY=1 loads Java in High Private memory.

Getting rid of the bloatware

● **In OS/2 Warp 3 and 4 days**, OS/2 users tweaked the CONFIG.SYS and deregistered WPS DLLs, because they wanted to **save RAM**.

➜ More RAM meant less paging to disk and as the hardware was slow compared to current systems, there was less delay and disk trashing.

● But with 100 of MiB RAM free on eCS, only **lack of virtual address space** should be the major reason to critically evaluate the features you need and to disable the ones you consider unneeded bloatware.

➜ The principles (see VOICE articles) are still the same. But the ways you must make your choices are different. Happily you can rationalize your choices if you learn to make use of tools as Theseus or the simple **Shmemmon**. And learn to understand the concepts of virtual memory.

● Once **StarOffice** was considered bloatware. But I write this Presentation with StarOffice, because it was designed for the 512 MiB virtual memory model of OS/2 whilst its successor, the virtual address space monstrum **OOO.org** was certainly not.

➜ Porters have an enormous job to do to. Don't blame them, but provide adequate feedback. So that they can learn to load code high.

● Before the large office and internet programs, the feature rich WPS and its libraries (enhancers) was the most memory consuming application.

➔On a system with 32 MiB of RAM, **lack of RAM** causes early **swapping**, which results in a **slower**, but **still stable system**.

➔ On systems with 128 MiB of RAM or more, you are more likely to see **"out of memory"** and **loading errors** while part of your RAM is still unused. The loading of new DLL's or growth of buffers in virtual memory fails because the system runs out of usable virtual addresses.

➔ **PM Shell** and the **WPS** are always **memory resident**. They need logical address space in the shared arena that may cause **loading errors** when opening other programs (DLL's). Including their own.

➔The WPS is also an always growing application: As more features (WPS enhancers) are used, more virtual address space is needed. And this is only given back after a successful WPS reset.

➔ When virtual address space in the shared arena is low, WPS DLL's that are already loaded may find no address space to expand or load. This may even happen to PM Shell. And if you cannot kill the big spenders with Watchcat or CAD then you have to reboot.

Finding large DLLs

Start Theseus / System / Linear Usage by Process and find (Ctrl-F, Ctrl-A) the string "object is".

The cursor will halt with the larger shared memory objects and give their occupied addresses space.

WNICE   #0003 (private)

object is 32.000M
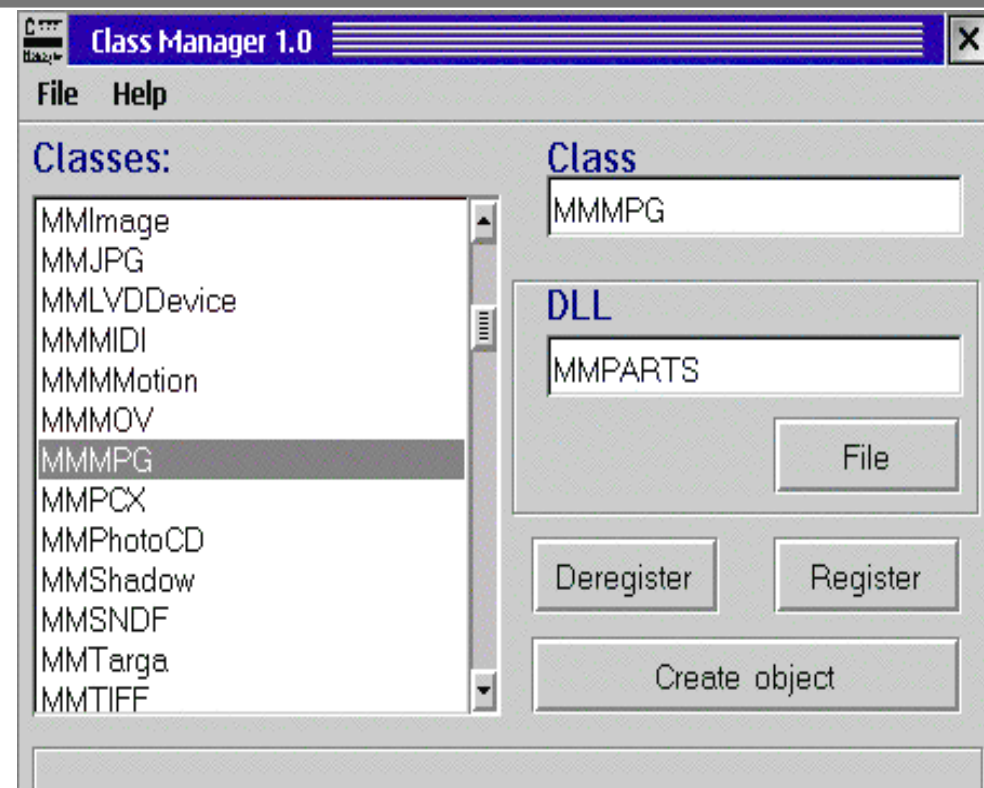
Deregistering DLLs

Unneeded WPS classes can be deregistered by tools like **Class manager**. After a WPS reset the DLL should not be loaded anymore.

If a DLL is used by more WPS classes, all of its classes should be deregistered.

Removing/renaming a DLL (first unlock it) and running checkini afterwards also works fine. Backup your INI files first. Keep more versions. Make a note of the removed classes and associated DLLs.

Not essential program DLLs can be disabled by removing them from the path or working directory (wnice.dll removed from Watchcat directory saved 30 MiB) .

WNICE.DLL was a nice Watchcat plugin: but it allocated 30 MiB.

The tower of Bable

Operating systems provide an **application programming interface** (API) to their functions that will be used by programmers.

The shared libraries of PM and the WPS provide basic functionality like widget toolkits to graphical OS/2 programs.

And the Win32 API makes Windows programs look and behave like Windows.

Porting the graphical Windows applications of **Lotus SmartSuite** to OS/2 took years for IBM. Because the PM and Win32 API's had become so different.

Interfaces must be well defined, preferably by its developers. But programmers come and go and its commercially (short time thinking) attractive not to define the API functionality that right.

IBM decided to build its own **Open32 libary** that contained more then 750 Win32 API functions to mimic the Windows API under OS/2. **Odin** and **Innotek** (VPC, Open Office, Acrobat) made use it to run minimally modified Windows applications under OS/2. But this **glue** uses virtual memory (address space).

StarOffice, the origin of OpenOffice.org, wanted to be a multiplatform Office Suite to compete with MS office. The developers decided to build an cross-platform C++ class library **StarView** that made only use of the host operating system API for the most elementary tasks.

Multiplatform graphical applications bring in their own **own graphical libraries**, but they need more memory and much more virtual address space - to achieve the same result an only for the standard GUI written program.

Their glue API enabled StarOffice to replace the WPS as the protected mode shell. And XUL makes that Mozilla looks the same under every operating system.

SmartSuite, **StarOffice**, **OpenOffice**, **Mozilla** do share memory in the shared arena, but unlike the PM API they are not accessed by others.

In fact **Mozilla** products using different (but very basic) **Innotek C libraries** are incompatible with each other. You need Rick Walsh's RUN! utility or do the **LibpathStrict** trick to load an incompatible DLL.

All these innovations extended the OS/2 (now eCS) operating system, but they laid a burden on its relatively small virtual address space.

When Microsoft Office 95 needed a VAL 2 GiB on 32-64 MiB systems many were surprised. But this has become the 32 bits standard.

How to save resources in the system arena?

Principle: Reserve the virtual memory addresses in the system arena only for those services that you and your OS really use or need.

➜ Disable ports and IRQ's in the BIOS if you do not use them. Think of unused serial and parallel ports and on the motherboard integrated devices. In PIC mode IRQ's are sparse.

➜ REMark in the CONFIG.SYS the drivers you do not expect to need. And document with a REM why you did so. Or with at least the date you changed it. So you can restore a former state

➜ SET SNAP_MAXVRAM_32MB=Y (see: Next sheet).

Don't expect wonders to occur:

➜ Most drivers have relatively small buffers that use little system memory. Video cards, TV cards and file system caches being the notable exceptions.

➜ But the system will boot faster.

➜ There are less potential driver conflicts.

Video memory in the system arena

• 2D applications do not profit from the large AGP framebuffers and Video Memory used by the last Windows and Linux 3D games.

• For OS/2 you probably only want a double buffered 2D display for your favorite resolution and colors. And still will have virtual screens of Pager.

For a 22" 1680x1050 monitor with 16M colors this is 2*24*1680*1050 10,1 MiB video ram.

• What happens to the unused Video RAM of a 256 MiB card?

• It is not used, but it could occupy virtual address space above the VAL.

• And this reduces the maximum value of your very usable JFS cache.

Resolutie (pixels)		Benodigd videoram in MiB voor een beeldscherm			
breedte	hoogte	8 bit (256)	15 bit (32k)	16 bit (64k)	24 bit (16M)
320	240	0,07	0,14	0,15	0,22
640	480	0,29	0,55	0,59	0,88
800	600	0,46	0,86	0,92	1,37
1024	768	0,75	1,41	1,50	2,25
1280	960	1,17	2,20	2,34	3,52
1400	1050	1,40	2,63	2,80	4,21
1600	1200	1,83	3,43	3,66	5,49
1920	1440	2,64	4,94	5,27	7,91
2048	1536	3,00	5,63	6,00	9,00

How to reduce the reserved address space for video memory in the system arena to sizes that eCS and OS/2 really can use?

• If the video memory is shared with working memory (graphics integrated in the motherboard) you can set it in the BIOS.

• Reducing video memory in the BIOS will increase working memory but might hinder 3D gaming under Windows and Linux.

• If you use SNAP, setting the by eCS used AGP and Video RAM is easy. It can be set lower than in the BIOS.

• **C:\SNAP>gaoption show** shows the current settings.

• Video Memory Limit....... 32 Mb

• Shared AGP memory size... 4096 Kb

•**C:\SNAP>gaoption vidmem [off | memsize]** sets the VRAM size.

•**C:\SNAP> gaoption agpmem [memsize]** sets the AGP window.

• Notice that AGP was designed to use (reserve) virtual memory. So it will not use RAM if is not used. But under eCS we often lack virtual memory addresses. And we do not need the virtual memory resources needed by Windows 3D games. So we better give it to drivers of VPC and JFS.

Cache memory

Generally, giving memory the caches and buffers of file system drivers is a good investment. Because caches generally serve all programs in a very efficient way. Without out of memory errors from the cache.

• With > 32 MiB of RAM, give HPFS cache the maximum of 2 MiB:

  IFS=C:\OS2 \HPFS.IFS /CACHE:2048 /CRECL:40 /AC:C

  But if you never use HPFS, REM it and save real memory too.

• Always allow for the maximum amount of 512 bytes disk access buffers that are used by HPFS and FAT (including FAT Floppy Disk access):

  BUFFERS=100

• Give a reasonable amount of memory to the FAT32 and FAT16 hard disk caches if you use them regularly on LVM mounted hard disk or USB partitions. But again REM the entries if you never intend to use them.

  Give the rest of you free RAM to JFS and really use it by always installing the big applications on JFS partitions.

Big applications that will profit from the larger JFS cache are:

• Virtual PC and Virtual Box and their disk images.

• Firefox, Thunderbird and other Mozilla family members. And do not forget to place their huge data paths on JFS.

SET MOZILLA_HOME=F:\home\default

SET MOZ_PLUGIN_PATH=F:\os2bin\internet\plugins

• OpenOffice.org, Acrobat reader, StarOffice and Lotus Smartsuite.

• Backups made by DFSee and others.

•Java runtime environments and their applications.

• Compilers, encoders, servers, games (and their TEMP and working directories). In fact most the Unix ports of Paul Smedley (;-).

And of course the WPS and its DLL's if directly booted from JFS will load faster.

Setting a large JFS cache:

You can try out several cache sizes and /LW parameters:

rem Place drivers/apps that use JFS after JFS.IFS!

rem IFS=F:\OS2\JFS.IFS /LW:16,64,4 /AUTOCHECK:* /CACHE:720000

IFS=F:\OS2\JFS.IFS /LW:32,128,4 /AUTOCHECK:* /CACHE:720000

rem IFS=F:\OS2\JFS.IFS /LW:5,20,4 /AUTOCHECK:* /CACHE:512000

Check it with cachejfs:

      SyncTime: 32 seconds # the **sync thread** runs every 32 sec.

      MaxAge: 128 seconds # changed **files** are written to disk < 128+32 sec.

      BufferIdle: 4 seconds   # **buffers** changed < 4 sec ago are not written.

      Cache Size: 720000 kbytes   # The JFS FS cache is 703 MiB

      Min Free buffers: 3600 (14400 K) # JFS maintains its own buffers,

      Max Free buffers: 7200 (28800 K) # which are 4K (block size)

Lazy Write is enabled   # HPFS and FAT share max 100 buffers of 0.5 K

When you set the /CACHE value too high (say 900 Mib with VAL 3 GiB), JFS will load with the default cache size of 12.5 % of RAM (if eCS loads at all).

Conclusions

When 32 bits **OS/2 2.0** came out for early PC's with 2-16 MiB RAM, **512 MiB of address space** for **protected mode user programs** seemed sky high.

But **software** was getting slower more rapidly than hardware became faster. More virtual address space, RAM and faster CPUs were needed.

Sharing data is the simplest strategy for inter process communication. Sharing **code** can reduce memory needs. This principle enabled PM and WPS applications to work together on an integrating platform.

This sharing also enabled large competing Office and Internet suites, that loaded their integrating libraries in the common shared arena. Without this address space being effectively used by all.

If programs use mainly the contiguous address space in the private arena (Java) or kernel space (via drivers), they will often load successfully.

But it they load large DLLs in the by many libraries reused and thus fragmented shared arena, memory allocation errors are likely to occur.

And this undermines the stability of eCS and OS/2 systems.

RAM and hardware became cheaper, but to get more address space asked for a **new memory model**, that was compliant with the classic 512 Mib address space of 16 and 32 bits OS/2 applications.

The **OS/2 4.5 kernel** gave new 32 bits programs **high virtual memory address space**, without having impact on the classic OS/2 system. Because only the specifically for the new kernel compiled programs could load their DLLs and data high.

Programs that do not load their code and data high may have to compete with legacy programs for unallocated address space in the now crowded classic 512 MiB address space (again, if they cannot join them).

● Some suggestions were given to prevent these problems, whilst still using your RAM.

➔ Let programs optimally make use of high memory addresses, being the most important. But this is easier said than done with a split up user space.

➔ Another way is to statically compile libraries and to load code privately like many Unix ports do. It uses more RAM, but who cares with MiBs of RAM free.

➔ If programs still cannot use your physical memory, then give them at least a large JFS cache.

Questions?