Plex86: an IA32 virtual machine

Kevin Lawton
MandrakeSoft, Inc.

Why virtualization?

- It is desirable to be able to run multiple operating systems simultaneously:
  - to concurrently run applications written for different operating systems without reboot.
  - to debug system software using a powerful debugger environment.
  - to consolidate management of a great number of heterogenous PC environments in an enterprise to a small set of virtual machine servers.
  - for call-in support centers, having many available VM configurations immediately ready for reference.

Related Technologies

- **Full software emulation: one could attempt to emulate the complete hardware environment - BOCHS**
  - **Pro: portability; works on any platform.**
  - **Con: quite slow.**
- **API emulation: one could attempt to emulate the API of an operating system, and run user applications natively - WINE**
  - **Pros: fairly lightweight strategy, guest OS not needed.**
  - **Cons: the emulated API needs to be fully documented or reverse assembled to offer a complete API translation. An API translation layer is needed for each guest OS.**

Pros of Virtualization

- Potential for fast execution.
- No knowlege about the guest OS or APIs is necessary.
- Many guest OSes can be supported.
- Can be used to debug kernel and driver code.
- Can gather instrumentation data from guest OS.

Virtualization Basics

- Create a virtualized environment which will allow both the guest OS and application software to execute quasi natively.
- Emulate access to IO peripherals, because only one OS (the host OS) can drive the real PC hardware.
- Emulate privileged instructions. Executing these would cause conflicts between multiple OSes.
- Many of the protection mechanisms are virtualized by configuring them to be similar to the requested guest values, only "deprivileged" to the application level to avoid conflicts with use in the host OS.

The challenge on ia32

- **The Intel 80x86 CPU series was not designed with full processor virtualization in mind:**
  - ***Most* system oriented instructions are protected, and thus generate a fault when deprivileged. This is an opportunity to emulate the behaviour accordingly.**
  - **There are *few* system oriented instructions which are *not* protected. Instructions such as sgdt, sidt, smsw, str allow code to examine system registers which virtualization may need to modify. Thus incorrect results may be returned.**
  - **There are also instructions which behave differently in privileged mode (pushf, popf, ...), than in non-privileged mode. So deprivileged guest kernel code may see incorrect values.**

The challenge on ia32

- Since the ia32 architecture doesn't provide a natural hardware solution to virtualize the small set of problematic instructions, plex86 uses a software technique to monitor execution of those instructions.

- The first (and current) architecture to handle this software instruction virtualization was a little easier to code and yielded a working mode. It has performance limitations however.

- The second architecture is a very lightweight dynamic translation technique (ia32-to-ia32 only). It is being prototyped currently. Tests thus far, show it to have very promising performance potential. It should soon be integrated into the main project.

Software technique #1: SBE

- The first and current implementation uses a technique dubbed "Scan Before Execute" (SBE).

- A private copy of each active code page is maintained. As instructions are dynamically scanned, problematic instructions are virtualized by replacing the private copy with the breakpoint instruction, int3. The private (modified) code pages are executed. The original pages are used for data accesses.

- To make this work, we must always monitor the execution flow, such that it never follows a code path which we have not yet scanned. Fortunately, each instruction only has to be scanned the first time it is executed, since we store information about such instructions in a cache.

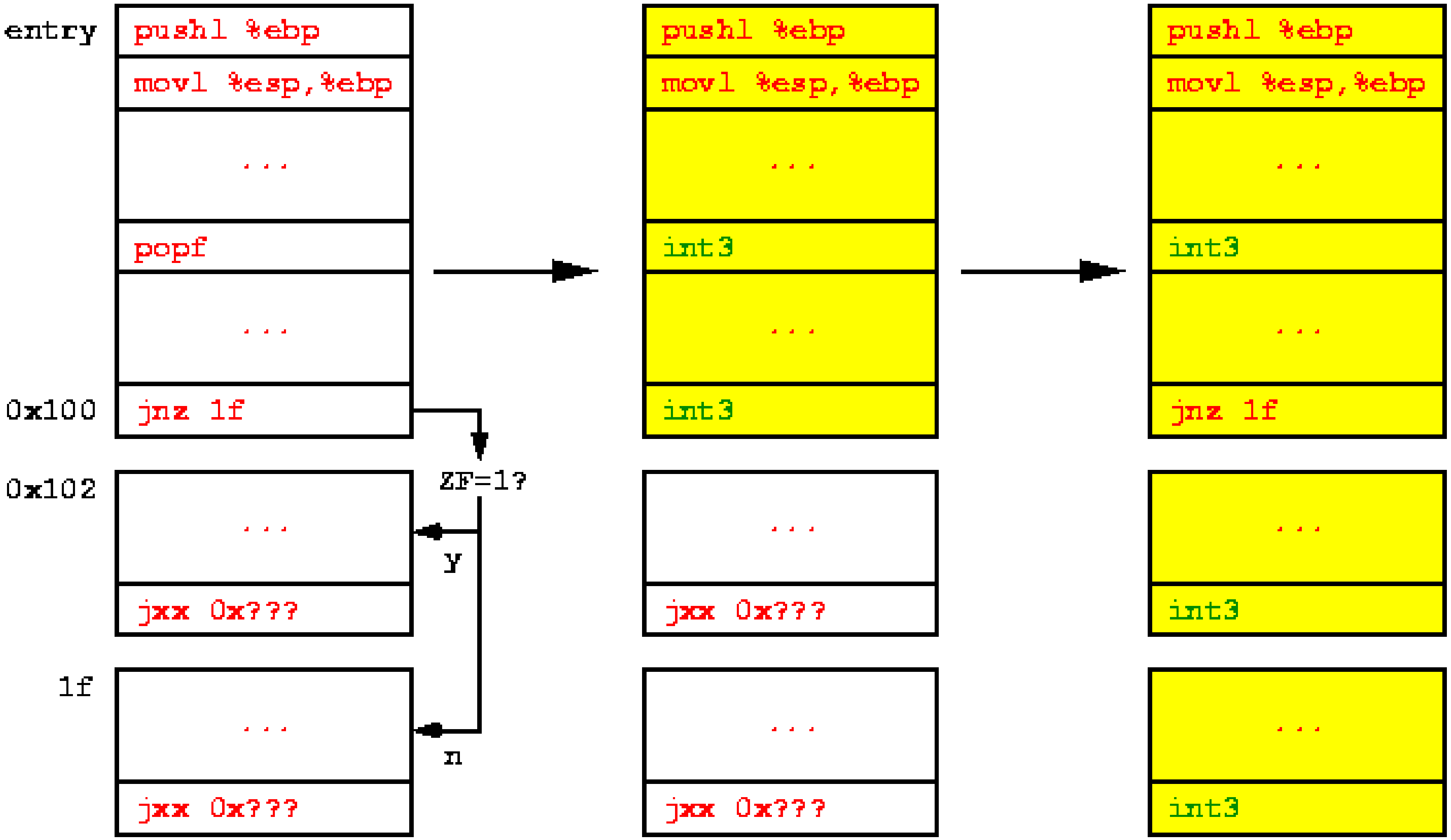
Software technique #1: SBE

- Because we must monitor the execution flow of the guest, computed branches and out-of-page branches must also be virtualized. This is accomplished the same way as for the problematic instructions - the instruction is replaced with the int3 instruction.

- Because the int3 instruction is only 1 byte long, it can be used to replace an instruction of any length. Therefore, no code expansion was necessary, allowing us to leave the instructions at the original offsets within each page.

- This technique is fairly simplistic, and it works. The downside is that the overhead involved with the int3 exception and monitor handling is quite expensive for performance.

SBE In Action



Software technique #2: DT

The second architecture is a very lightweight Dynamic Translation technique (ia32-to-ia32 only). If we remove the constraint that code fragments have to remain at their original offsets, we have more flexibility:

- Some instructions can be handled by generating a small amount of inline code. For example, it's quite simple to correct the RPL bits for a read of a segment selector (which we deprivileged for virtualization).

- For other instructions (like computed branches), we can generate some inline code to call a common and larger handler routine, placed at the user privilege level. This prevents too much code bloat, while eliminating the expensive transitions to/from the VM monitor.

Software technique #2: DT

This technique requires more maintenance. Because code sequences can be arbitrarily expanded and code offsets are not retained, we must maintain several constructs:

- a translated code sequence buffer
- a table of original code offset to translated sequence offset
- a reverse table of translated sequence offset to original code offset - for servicing exceptions orginating from translated code.
- some other structures for efficiency.

But tests of the prototype, show some very promising results.

When neither technique is needed

Most situations in ia32 where the CPU does not give us natural virtualization, relate to the privilege shift (deprivileging) we apply to guest OS code.

Given the right circumstances, application code could be executed without the intervention of technique #1 or #2. This is because it is already executing at the lowest privilege level, and there is no privilege shift applied.

So, for the duration of execution timeslices given to application code, execution could be completely native. Thus CPU intensive application code has excellent performance potential.

System code always has to be dynamically virtualized.

General architectural overview

There are three major logical components in plex86:

- The user space program is scheduled naturally by the host OS. It currently contains the IO device emulation and the GUI, both of which use the native libc and GUI library services. When not handling these duties, it calls the VM kernel module service to continue running the guest.

- The host kernel module provides the VM services interface with user space, allocates memory for each VM, and effects transfers to the VM context.

- The VM context contains a VM monitor and the guest OS/applications. More on the next slide...

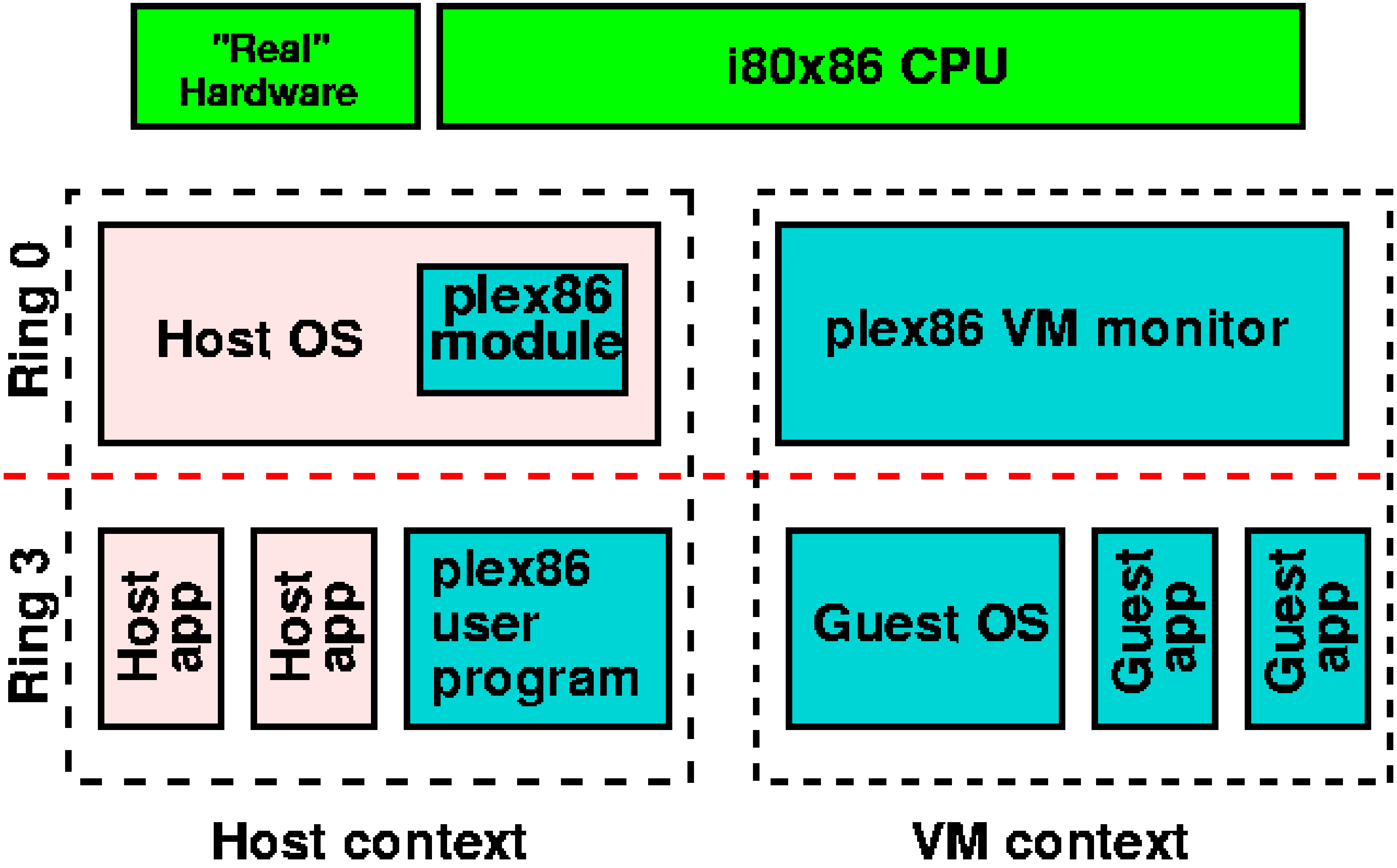
The VM Context

The VM environment provides separate OS-like facilities:

- **Page tables.**
- **Data / code segments.**
- **Hardware interrupt handlers (h/w interrupts are forwarded to the host OS).**
- **Exception handlers.**

These facilities allow some guest code to execute natively, and/or to emulate guest instructions more efficiently.

Architectural Block Diagram



Architectural overview

The user program component is fairly minimal. Much of the functionality is provided by plugins. A plugin architecture makes plex86 easy to extend and maintain.

- IO device emulation. Currently the device emulation from BOCHS is used as one large plugin. Ultimately each device emulation module will be its own plugin.

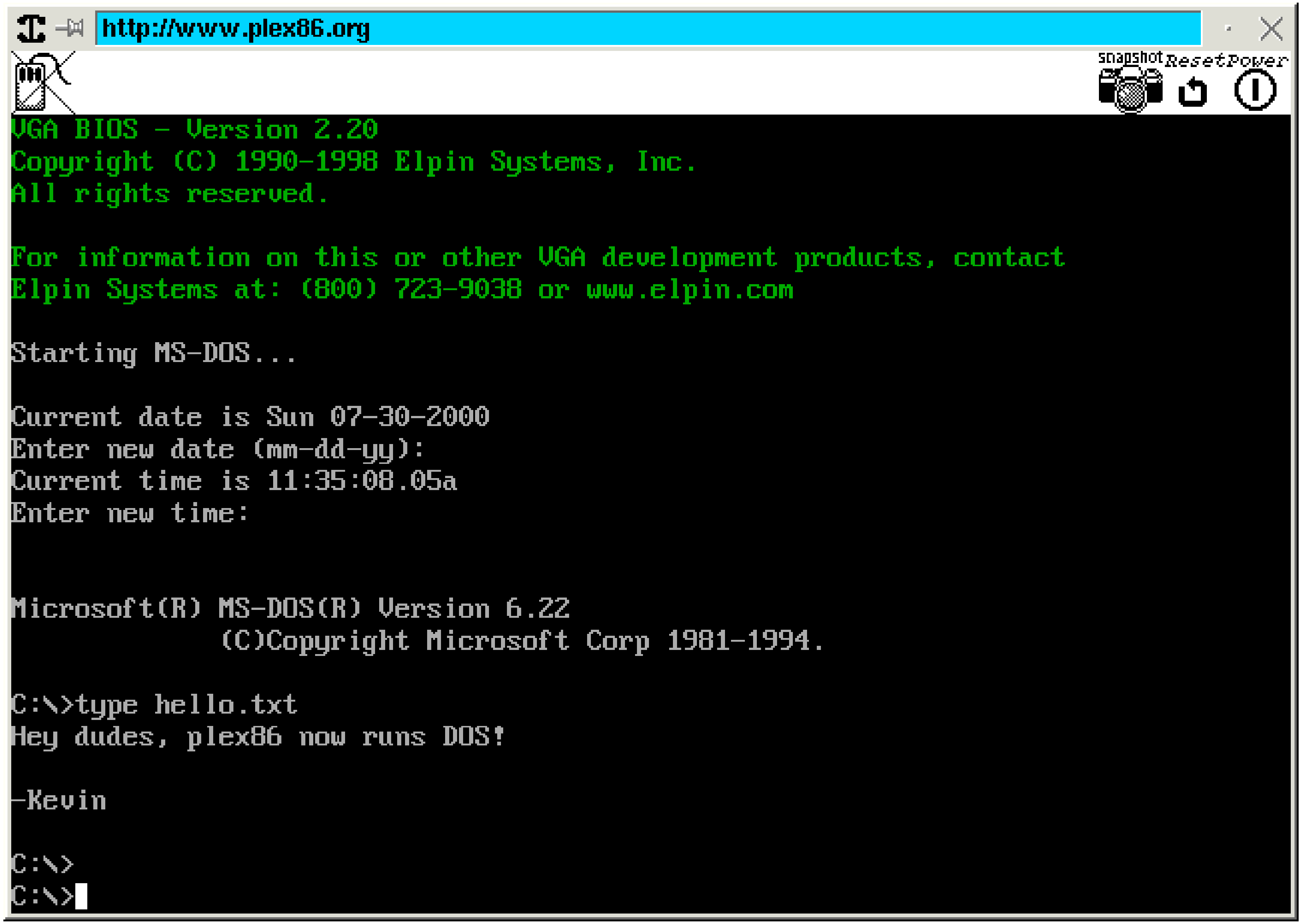
- Plugins can be used to extend the VM in many ways. For instance, the ICE plugin provides debugging through GDB.

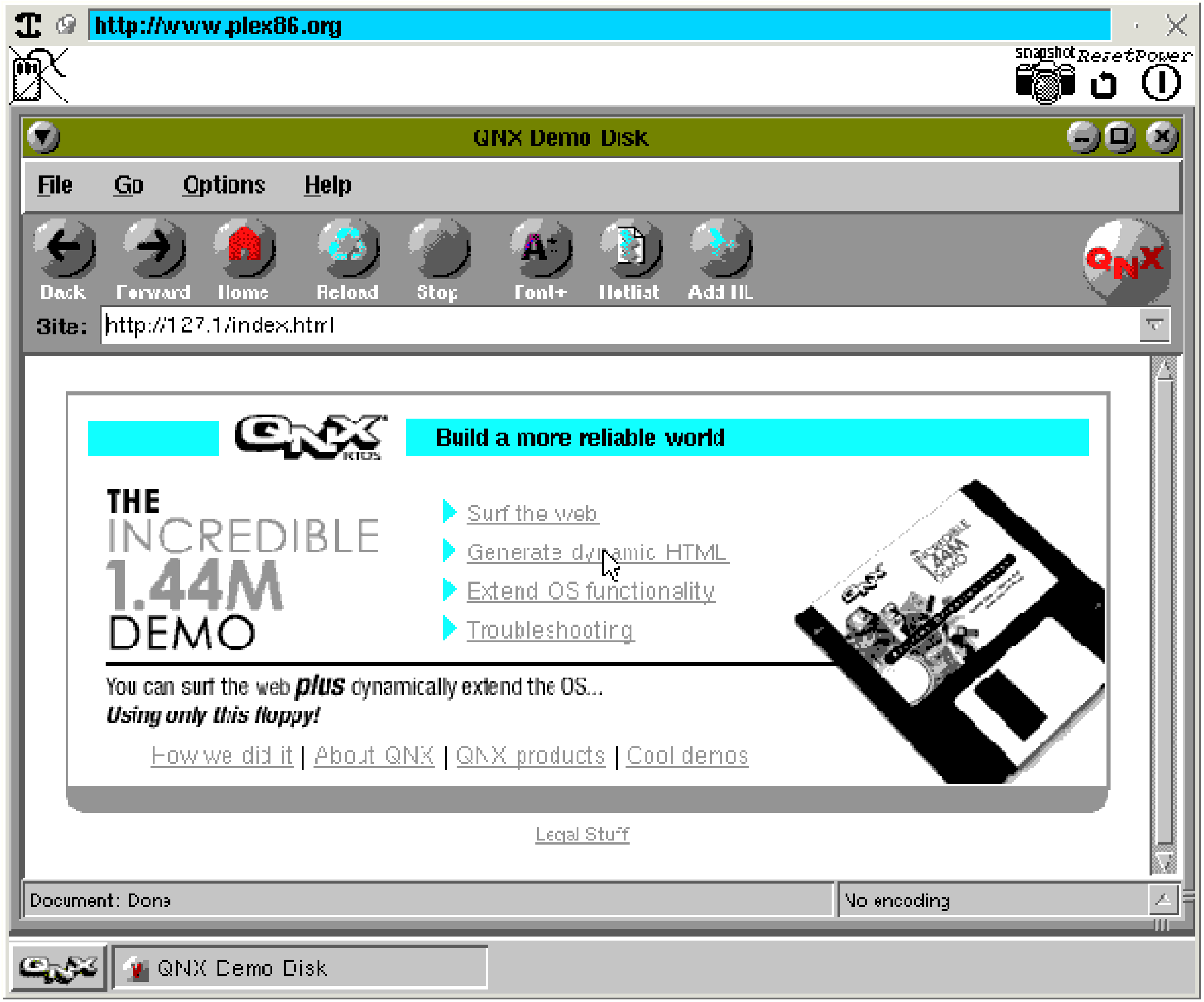
- A write-cache plugin allows modifications to the disk image file to be thrown away, while retaining the original contents.

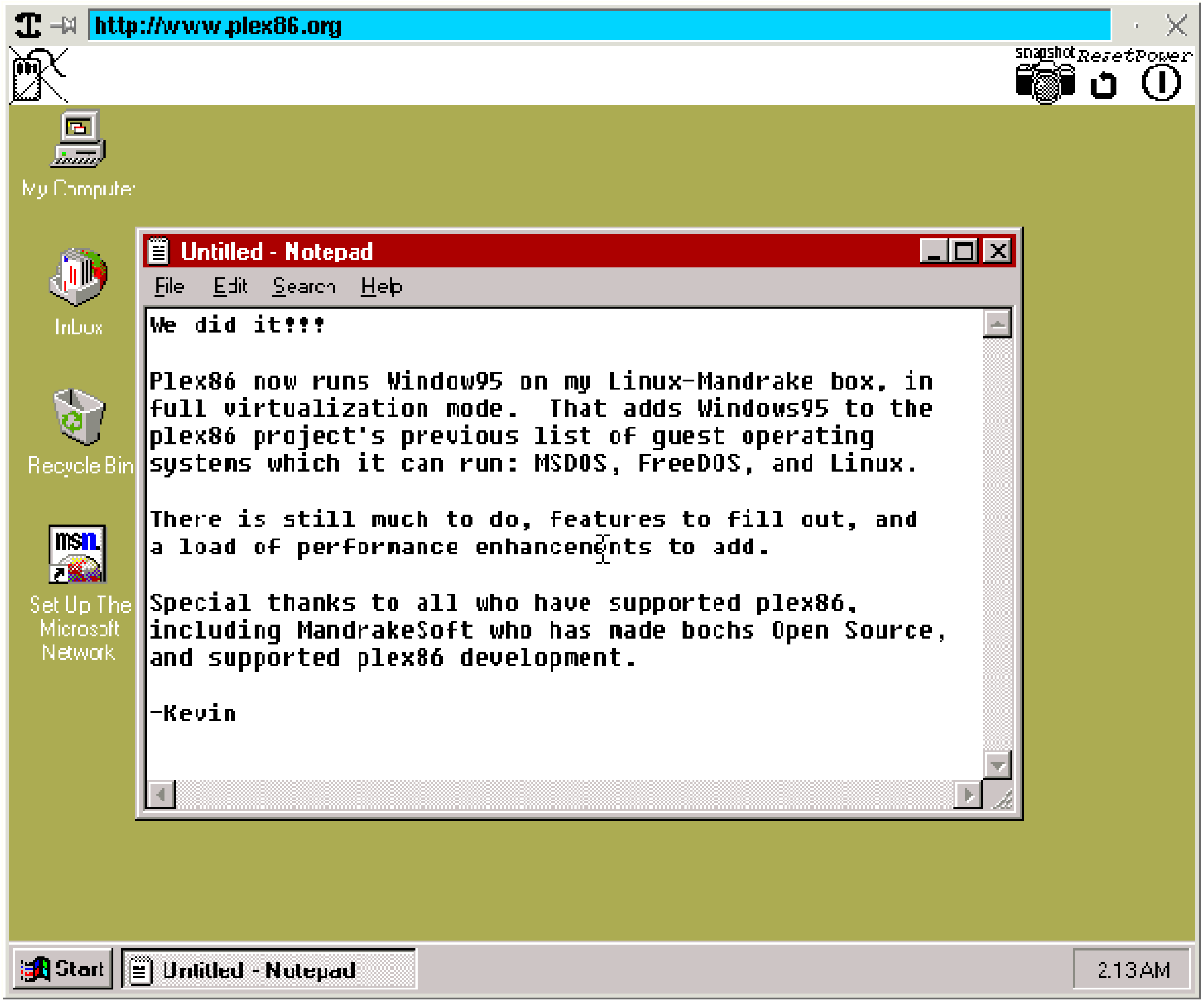
- Other plugins could be easily written to extend functionality.

Current status

- Most of the framework for the general architecture is built. Though there are many unoptimized areas remaining.
- The Dynamic Translation (DT) layer was prototyped and integrated into the code. Several OSes worked successfully.
- For now, pure emulation mode is used for all code. Support for executing User code natively will be added soon.
- Instrumentation data gathered dynamically from the emulation will tell us when to bother DT'ing kernel code.
- Many guest operating systems have been successfully executed inside of the plex86 VM: Win95, DOS, FreeDOS, Linux, WinNT4.0, ...


http://www.plex86.org
snapshot
Reset
Power
VGA BIOS - Version 2.20
Copyright (C) 1990-1998 Elpin Systems, Inc.
All rights reserved.
For information on this or other VGA development products, contact
Elpin Systems at: (800) 723-9038 or www.elpin.com
Starting MS-DOS...
Current date is Sun 07-30-2000
Enter new date (mm-dd-yy):
Current time is 11:35:08.05a
Enter new time:
Microsoft(R) MS-DOS(R) Version 6.22
(C)Copyright Microsoft Corp 1981-1994.
C:\>type hello.txt
Hey dudes, plex86 now runs DOS!
-Kevin
C:\>
C:\>


http://www.plex86.org
snapshot Reset Power
QNX Demo Disk
File Go Options Help
Back Forward Home Reload Stop Font+ Hotlist Add HL
Site: http://127.1/index.html
Build a more reliable world
THE INCREDIBLE 1.44M DEMO
Surf the web
Generate dynamic HTML
Extend OS functionality
Troubleshooting
You can surf the web plus dynamically extend the OS...
Using only this floppy!
How we did it | About QNX | QNX products | Cool demos
Legal Stuff
Document: Done
No encoding
QNX Demo Disk


http://www.plex86.org
snapshot
Reset
Power
My Computer
Inbox
Recycle Bin
Set Up The Microsoft Network
Untitled - Notepad
File
Edit
Search
Help
We did it!!!
Plex86 now runs Window95 on my Linux-Mandrake box, in full virtualization mode. That adds Windows95 to the plex86 project's previous list of guest operating systems which it can run: MSDOS, FreeDOS, and Linux.
There is still much to do, features to fill out, and a load of performance enhancements to add.
Special thanks to all who have supported plex86, including MandrakeSoft who has made bochs Open Source, and supported plex86 development.
-Kevin
Start
Untitled - Notepad
2:13 AM

Future directions

- Many performance enhancements

- Special guest OS specific drivers. For some devices, it is more efficient to define our own virtual hardware, and create guest OS specific drivers to manage the device and communicate to the host, than it is to emulate real hardware. Video, networking, and hard disk handling are areas which may benefit.

- VM checkpointing. We could add the ability to save the entire state of the VM to disk, and restore to this state later. This can save time, because no guest OS reboot is necessary.

- Persistent disk images. Modifications can be written to a differential file. This lets multiple VM instances use the same base disk image or partition, but each maintains a separate differential file.

Future directions

- Add a more complete debugging environment capability.

- Addition of a scheme to dynamically unlock memory allocated for the guest, and return it to host OS paging eligibility when it is not in recent use. This will adapatively shape the VM memory footprint according to use.

- Add ability to recognize when guest OS is idling, and return execution back to host to eliminate wasted CPU cycles.

- Use by Virtual Machine Service Providers (VMSPs). Companies have expressed interest in using plex86 to provide VM services from VM "server farms" to graphics terminals at other locations. Lightweight usage instrumentation would be helpful for VMSPs.

Future directions

- Perhaps plex86 framework can be of use on new architectures like the AMD x86-64, for running legacy ia32 code on ia64 OSes like Linux?

- Other ideas? Let us know!!!

Terminology

- CPL0/Ring0/System Code/Kernel Code: The inner most "ring" (most privileged) is where kernel code runs.
- CPL3/Ring3/User Code/Application Code: The outer most "ring" (least privileged) is where normal applications run.
- GDB: GNU DeBugger; a very popular and capable programmer debugger.
- RPL: The Requested Privilege Level. Each selector requests a privilege level of the segment to be loaded.
- ICE: In-Circuit Emulation. A plugin which communicates with GDB to provide a software facility similar to a hardware ICE.
- SBE: Scan Before Execute. A Techique where code is scanned and cached before executing the first time it is encountered.

For more information:

www.plex86.org