Windows™ “Chicago” Architecture

Tammy Steele
Technical Evangelist
Systems Marketing
Microsoft Corporation
What We’ll Talk About

- “Chicago” user architecture
- “Chicago” kernel architecture
- “Chicago” GDI architecture
Windows “Chicago” User Architecture
Overview Of New User Features

- Increased system capacity
- Robustness additions
- Win32® API support
- New Win32 APIs to support new user features
Increased System Capacity

- Up to 32K menu and window handles (each)
- Number of timers limited only by available memory
- Unlimited number of COM and LPT ports
- List box item limits raise from 8K to 32K, with data limited only by available memory
Robustness Additions

- More robust parameter validation
- Each 32-bit or 16-bit process or thread ID is used for object ownership of Ring 3 objects
- All 32-bit applications and version 4.0-marked 16-bit applications have unfreed resources cleared at application termination time
- Existing Windows 3.x-based application resources are cleared when “Chicago” determines no other Windows 3.x-based applications are running
# Robustness Additions

<table>
<thead>
<tr>
<th>Ring 3 objects</th>
<th>Tracked by:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hooks</td>
<td>thread</td>
</tr>
<tr>
<td>Windows</td>
<td>thread</td>
</tr>
<tr>
<td>Global Memory</td>
<td>process</td>
</tr>
<tr>
<td>Menus</td>
<td>process</td>
</tr>
<tr>
<td>Classes</td>
<td>process</td>
</tr>
<tr>
<td>Cursors</td>
<td>process</td>
</tr>
<tr>
<td>Icons</td>
<td>process</td>
</tr>
<tr>
<td>Pens</td>
<td>process</td>
</tr>
<tr>
<td>Brushes</td>
<td>process</td>
</tr>
<tr>
<td>Regions</td>
<td>process</td>
</tr>
<tr>
<td>Device Contexts</td>
<td>process</td>
</tr>
<tr>
<td>Fonts</td>
<td>process</td>
</tr>
<tr>
<td>Bitmaps</td>
<td>process</td>
</tr>
<tr>
<td>Palettes</td>
<td>process</td>
</tr>
</tbody>
</table>

| User     | |
|----------||
| GDI      | |

<table>
<thead>
<tr>
<th>Win32 User API Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetDebugErrorLevel</td>
</tr>
<tr>
<td>MessageBoxEx</td>
</tr>
<tr>
<td>ExitWindowsEx</td>
</tr>
<tr>
<td>CopyAcceleratorTable</td>
</tr>
<tr>
<td>CreateAcceleratorTable</td>
</tr>
<tr>
<td>DestroyAcceleratorTable</td>
</tr>
<tr>
<td>EnumPropsEx</td>
</tr>
<tr>
<td>PostThreadMessage</td>
</tr>
<tr>
<td>SendMessageCallback</td>
</tr>
<tr>
<td>SendMessageTimeout</td>
</tr>
<tr>
<td>SendNotifyMessage</td>
</tr>
<tr>
<td>FormatMessage</td>
</tr>
<tr>
<td>AttachThreadInput</td>
</tr>
<tr>
<td>GetThreadDesktop</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CreateIconFromResource</td>
</tr>
<tr>
<td>CreateIconIndirect</td>
</tr>
<tr>
<td>GetIconInfo</td>
</tr>
<tr>
<td>LookupIconIDFromDirectory</td>
</tr>
<tr>
<td>ActivateKeyboardLayout</td>
</tr>
<tr>
<td>GetKeyboardLayoutName</td>
</tr>
<tr>
<td>LoadKeyboardLayout</td>
</tr>
<tr>
<td>UnloadKeyboardLayout</td>
</tr>
<tr>
<td>WindowFromDC</td>
</tr>
<tr>
<td>CreateMDIWindow</td>
</tr>
<tr>
<td>WaitForInputIdle</td>
</tr>
<tr>
<td>MsgWaitForMultipleObjects</td>
</tr>
<tr>
<td>GetForegroundWindow</td>
</tr>
<tr>
<td>SetForegroundWindow</td>
</tr>
</tbody>
</table>
Added User Functionality

- New minimized window look
  - GetSystemMetrics, SM_ARRANGE
- Enhanced control capabilities
  - Bitmap buttons, thumbsize of scroll bars, etc.
- Enhanced support for context-sensitive Help and message boxes
Added User Functionality

- APIs for drawing 3-D windows and frame controls
- Enhanced menu support (default items, radio items, bitmap/icon items, and extended pop-up menus)
- Dialog manager support for other font styles in RC template
Windows “Chicago” Kernel Architecture
Topics

- “Chicago” system virtual machine overview
- Tasking/scheduling
- API serialization
Windows “Chicago” Architecture

Ring 0
- Protect mode file system
  - VFAT, CDFS, SCSI, Network
- Virtual machine manager
  - Pager, Scheduler, DPMI server

Ring 3 (System VM)
- Win32® application
- Win32 application
- Win32 application
- System services:
  - Kernel
  - Graphics
  - Window management

Ring 3
- Win16 application
- Ring 3 (MS-DOS® VM)
- Ring 3 (MS-DOS VM)
System VM Overview

Ring 3

32-bit side

- USER32
- GDI32
- Kernel32

16-bit side

- USER16
- GDI16
- Kernel16

Win32 application

Threads

Win16 application

Shared address space

Win16 application

Thread

Thread
16-Bit Tasking

- Cooperative tasking, same as Microsoft® Windows 3.1
- Each Win16 application runs as a thread, providing for resource tracking in Ring 0 (VxDs) and Ring 3 (GDI and USER)
- Synchronization between applications occurs via messages
32-Bit Tasking

- All 32-bit applications and threads are fully preemptive
- Any thread can call any API
- Fault handler on a separate thread for robustness
- Compatible with Windows NT™ model
The Scheduler

- Compatible with Windows NT model: 32 priority levels supported
- Also compatible with existing Win16 applications, VDDs, and VxDs
- Dynamic priority boosting with timed decay
- Priority inheritance boosting
API Serialization - General

- Every multithreaded OS must serialize some APIs
- Critical sections are commonly used

```c
void InsertObjectInList(OBJECT* pobj)
{
    EnterCriticalSection(&Lock);
    pObjectListHead = pobj;
    pobj->pNext = pObjectListHead;
    LeaveCriticalSection(&Lock);
}
```
Examples of subsystem serialization:

- GDI32 uses per-object locking allowing multiple clients to reenter using different objects.
- User32 maintains a single lock for all APIs. Only one thread at a time may be in a User32 API.

Also, applications (clients) and OS subsystems (servers) are on separate processes/threads in Windows NT.
“Chicago” uses a global system-critical section for all Win16 components called the Win16Lock.

When thunking from 32-bit to 16-bit code, “Chicago” always attempts to claim the Win16Lock.

If the current thread cannot own it (because another thread does), it blocks.
Results Of Win16Lock

- All 16-bit code is protected, including third-party DLL code
- 16-bit core components stay small and fast
- Compatibility is ensured by not changing API ordering and timing
- Win32 threads calling native 32-bit API do not block on Win16Lock.
Results Of Win16Lock

- Note that any lack of smoothness in multitasking caused by the Win16Lock will only be due to ill-behaved Win16 applications.
- A system with only Win32 applications will not be affected by the Win16Lock.
- The shell is a Win32 application.
Windows “Chicago” GDI Architecture
GDI Module Enhancements

- Performance
- Reduce system resource limitations
- Windows NT congruence
Reduce 64K Limitations

- Local heap limitations
  - Regions out
  - Physical objects out
  - Font management structures out
  - DCs still in 64K heap
  - GDI object ownership
Graphics

Application

Graphics device interface

Image color matcher

Display minidriver

Printer minidriver

Universal printer driver

Graphics DIB engine

Display

Printer
Bitmaps And DIBs

- Windows 3.1 distinction of bitmaps and DIBs
  - Bitmaps - device-dependent
  - DIBs - device-independent

- New DIB APIs
  - `CreateDIBSection()` - creates a DIB that both applications and GDI can write to
  - `SetDIBColorTable()`
  - `GetDIBColorTable()`
32-Bit DIB Engine

- Flat 32-bit code
- Will be the fastest software-only driver
- 1, 4, 8, 16 (555), 24 (RGB) bpp supported
- Straightforward to add other formats
Display Minidrivers

- Works best with:
  - Flat linear frame buffer
  - Local bus video memory

- We will have for “Chicago”
  - The fastest, best driver for frame buffers, with
  - Hooks for hardware accelerators
Advantages Of DIB Engine

- Add new DDI without requiring new drivers
  - Example: WideTextOut
- Printer drivers no longer dependent on quality of display driver
- Very robust display drivers
- Smaller display drivers
- Easy to add new formats
32-Bit TrueType® Rasterizer

- Fixes problems with current rasterizer
- Complicated glyphs, such as Han
- Better fidelity
- Better performance
- Uses memory mapped files
- No more .FOT files
- Faster boot with lots o’ fonts!
Congruence Of Windows NT

- Paths
- Beziers
- Enhanced metafiles
- Color cursors
Paths

- All Win32 path APIs have been implemented
- Primitives: Lineto, Moveto, and PolyBezier supported
- Not pel-perfect to Windows NT
Beziers

- PolyBezier
- PolyBezierTo
- Port of the Windows NT code to 386 assembly
- Not pel-perfect to Windows NT
Metafiles

- Windows 3.1 metafiles “lingua franca”
- Win32 enhanced metafiles
  - All enhanced metafile APIs supported
  - Partial support of world transforms
- Skip records “Chicago” does not understand
- Port of the metafile converter to Windows DLL
Enhanced Metafiles

- Open with reference device
- Goal: reproduce drawing from reference device
- Playback is original size by default
- World transform scales
- Clip regions scale and work, but
- Paths scale better than regions
Additions To Windows NT GDI

Windows NT GDI has but “Chicago” does not

- Wide-styled lines
- Forms
- Transforms
- Dithered pens and text
- Complete 32-bit internals and coordinates
- Masked blit
- Fine-grained reentrancy
Questions?
Windows “Chicago” Ring 3
Relative Code Distribution

32-bit side 16-bit side

**USER3**
2

**Kernel32**
Thread services, synchronization objects, memory management, memory-mapped files, file I/O, debug services, console, comm, etc.

**GDI32**
TrueType rasterizer, print subsystem, spooler, universal graphics engine (DIB engine)

**Kernel16**

**USER16**
Existing Windows 3.1 window and menu management services, plus new features (async input model, new styles, etc).

**GDI16**
Existing Windows 3.1 graphics management, plus new Bezier, path, EMFs, etc.

(One way)

Thunk bandwidth