A Comparison of OS/390 and Windows NT

Session 2828
MVS SCP Project
SHARE 92
22 February 1999

James Antognini

antognini @ us.ibm.com

IBM T. J. Watson Research Center
P. O. Box 704
Yorktown Heights NY 10598

Legalese

Overview

- Introduction.
- The ascent of Windows NT.
- Comparing and contrasting: Similarities, differences.
  - Address spaces.
  - Storage.
  - Scheduling.
  - Privileged mode and device drivers.
  - IRQL.
  - Multistage I/O.
  - Error handling.
  - Filter routines.
  - Threads.
  - Serialization and synchronization.
  - Memory mapping.
  - Services.
  - Job.
- Conclusions.

Introduction

- NT is more than a desktop, is challenging Unix systems.
- NT doesn’t (yet) directly confront OS/390, but that community uses it (front-ends and clients).
- Windows 2000 (NT 5.0) is estimated to be 30-40 million lines of code, getting near OS/390.
- Basis of comparison is NT 4.0 with SP4 and OS/390 2.6.
- Intel x86 will be the specific NT basis.

The Ascent of Windows NT

- CP/M –> DOS –> OS/2 –> NT (??)
- At DEC David Cutler led the RPX/11M project, then the VMS project.
- When DEC cancelled PRISM, the Cutler crew (already in Seattle) joined Microsoft in 1988 and began NT.
- It’s disputed how much of OS/2 went into NT. But NT still has an OS/2 subsystem.
- The seminal GUI work was at Xerox PARC, and Apple picked that up. It was incorporated into Windows.
- Virtual storage and preemptive multitasking were widely used beginning in the 1960s. Objects became common in the 1980s. The MACH kernel was a pioneering layered system.
- So the streams of influence on NT are almost as venerable as those on MVT-MVS-MVS/XA-MVS/ESA-OS/390.

Comparing: Address Spaces

- In NT:
  - An address space of separately addressable virtual storage is 4G. The private area (“user space”) is usually 0-7FFFFFFF.
  - Only “kernel-mode” programs may touch 80000000 and higher (“system space”), which is commonly addressable.
  - *Process* comprises separate virtual storage and threads.
- In OS/390:
  - An address space is 2G. Common areas (CSA, SQA, LPA and nucleus) straddle the 16M line.
  - *Address space* comprises separate virtual storage, TCBs and SRBs.`
- Commonalities:
  - Page is 4K and may be backed.
  - Virtual storage isolates programs from each other; provides more storage than real memory; is a point of ownership of resources.
  - An address space may be swapped out to reclaim its storage.

Comparing: Storage

In NT:

- Storage is composed of virtual pages.
- Page can be backed by a real frame and by hard disk paging space.
- *Paged* storage may not be currently backed. *Non-paged* storage always is.
- A page may be readable and writeable. For user-mode programs, it may be readable only, or neither readable nor writeable. Finally, a page may be copy-on-update (for forking).
- Paged storage is subject to stealing of the frames. The storage may be locked down (“fixed”). Stolen frames are kept on a queue for possible reclaim.
- Page tables are allowed to be paged out.

Sound familiar?

Not in NT is storage protection by key.

Comparing: Scheduling

In NT:

- Work has priority, which subjects it to losing the CPU in preemption by higher-priority work that becomes ready.
- A thread is given at most a quantum of CPU time. After that, it can lose the CPU to other work of equal priority. (OS/390 used time-slicing in the past.)
- An interrupt (e.g., I/O, timer) will temporarily displace the thread from the CPU and may make higher-priority work ready to go.

Comparing: Privileged Mode

In NT:

- User mode (“ring 3”) is like problem state in OS/390.
- Kernel mode (“ring 0”) is like supervisor state in OS/390.
- These causes execution in kernel mode:

  1. The boot process;
  2. Invocation by interrupt (e.g., I/O), by exception (e.g., invalid opcode) or by the INT instruction;

  or

  3. Call by another kernel-mode routine (e.g., an ISR called by the I/O interrupt handler).

The INT instruction:

- Behaves much like SVC: Status is saved, and control goes to the OS-defined handler and then to a routine designated by the instruction operand.
- INT 2E (decimal 46) gives control to the NT exception handler and then, due to parameters, to the indicated NT routine (e.g., NtCreateFile).

NT does not support anything comparable to OS/390’s PC routines, despite call gates in x86 architecture.

Comparing: Installing a Device Driver

In NT the device driver is the only supported mechanism for adding kernel-mode code.

The steps to install:

1. The entry point DriverEntry is loaded at boot due to registry specification or programmatically. Such registry updating or using the NT services requires Administrator authority.
2. DriverEntry (in kernel mode, in system space) creates a device object, possibly with an extension as an anchor for objects (“control blocks” manipulated by defined services). The device object might be named \Device\Ctrl2cap. Driver, device object and extension are in non-paged storage.
3. DriverEntry creates a symbolic link, \DosDevices\Ctrl2cap.
4. DriverEntry defines entry points for expected operations, minimally Create, DeviceControl and Close in this case.

The driver is ready to offer its services.

Comparing: Using the Device Driver

A user-mode (or kernel-mode) program can use the driver through these steps:

1. The program opens the device object by calling CreateFile with file name \\.\Ctrl2cap. The call goes to a stub, which issues INT 2E.
2. Running in kernel mode in system space, the NT Open effector understands the file name to refer to \DosDevices\Ctrl2cap and thus to driver Ctrl2cap. It branches to Ctrl2cap's Create entry point with IRQL PASSIVE_LEVEL.
3. The driver satisfies itself the program is a legitimate caller and does any necessary setup.
4. The program gets back a handle from CreateFile.
5. When a service is needed, the program specifies the handle and driver-defined parameters in calling DeviceIoControl.
6. The NT DeviceIoControl effector, running privileged, copies the parameters to non-paged storage in system space and, with IRQL PASSIVE_LEVEL, branches to Ctrl2cap's DeviceControl entry point.
7. The driver checks for a legitimate request and does whatever. (Touching user space might require locking down storage or using an APC intermediary.)
8. When done with Ctrl2cap's services, the program closes the file handle, which allows Ctrl2cap's Close entry point to clean up.

Comparing: Interrupt Request Level (IRQL)

IRQL has no direct OS/390 analog. IRQL is a scheduling mechanism composed of a hierarchy of values that mask, or suppress, “interrupts” (hardware interrupts, software requests) with lower IRQL.

- The NT scheduler gets control from a routine giving up the CPU and inherits the IRQL. The scheduler’s mission is to drain its CPU of queued-up requests at that IRQL and lower.
- The scheduler looks for ready work at that IRQL and gives it control. When that IRQL queue is empty, the scheduler drops the IRQL and looks at the next level’s queue. Eventually no work remains for the CPU.

Note that IRQL is for scheduling, not serialization: It affects only the current CPU.

A kernel-mode routine can raise or lower its IRQL. The NT routine effecting the change may accompany the new level with masking or unmasking of hardware interrupts. The kernel-mode routine must be careful to call only services that work at its current IRQL.

Higher IRQL brings greater importance and more restriction.

Comparing: IRQL continued

Name	Remarks
PASSIVE_LEVEL	"Normal" work.
APC_LEVEL	Highest level allowing thread scheduling. Used for APC routines.
DISPATCH_LEVEL	Suspends thread scheduling. Page-faulting disallowed. Used for device-driver Startio routines, for DPC routines and for most spin locks.
DIRQL 1 … } **DIRQL *n***	Device level, e.g., an ISR (device-driver back-end). Used for ISR spin locks. I/O and timer interrupts masked. NT in effect associates a particular DIRQL with a device.
PROFILE_LEVEL	Timer used for profiling.
CLOCK1_LEVEL	Interval timer.
CLOCK2_LEVEL	Interval timer.
IPI_LEVEL	CPU signalling.
POWER_LEVEL	Power failure.
HIGHEST_LEVEL	Machine-check and bus errors.

Comparing: Multistage I/O

In NT, this is I/O:

1. A program opens a file, and ultimately a kernel-mode device driver verifies the request's legitimacy and correctness and builds necessary objects. The program gets back a file handle.
2. The program asks for I/O, specifying the handle and parameters. The driver routine parses parameters and calls the device (if not busy) via lower-layer routines, where IN, OUT or something like is issued.
3. The device finishes and causes an interrupt.
4. The NT trap handler gets control in a random address space and calls the interrupt dispatcher, which calls the device driver's Interrupt Service Routine (ISR).
5. The ISR, getting control at Device IRQL, dismisses the interrupt, perhaps gathers error information, may start another I/O and queues up its Deferred Procedure Call (DPC) routine.
6. The DPC runs in a random address space at DISPATCH_LEVEL IRQL. It does what it can (perhaps updating structures and driver buffers) and may queue up an Asynchronous Procedure Call (APC) routine.
7. An APC runs in a specific address space at APC IRQL. The APC can touch user buffers and may make final status available to the program.

All very much like OS/390.

Comparing: Error Handling

NT's error handling makes possible determinate program response to errors. For example,

```
_try
  {
   // begin code coverage

   // end code coverage
  }
  _except(// filter routine ("recovery" exit)
         )
    {
     // begin exception handler ("retry")

     // end exception handler
    }
```

Notable features:

- Coverage guards specific code with specific error code.
- Coverage can be nested.
- Filter routine gets description of error, can choose to retry or to let percolate to a higher layer (if any).

But there are differences from OS/390:

- Error coverage has thread granularity.
- Some errors in "higher" kernel-mode states aren't caught.
- Unhandled errors typically cause process termination.
- NT's error handling, distinguished from C's or C++'s, gives control a second time to a percolating filter, after a higher filter elects to handle the error.

Comparing: Filter Routines

There is no true analog in OS/390.

- A driver can attach a device object of its making to that of another, target driver, such as the NTFS driver.
- Now the inserting, filter driver sees, and can alter, all I/O requests made to the target.

Uses include encryption, compression and virus filtering.

All very much like OS/390.

Comparing: Threads

- Threads are units of work, and most programs run under them.
- They have priority, are an error boundary and own resources (e.g., timers). When a thread terminates, its resources are cleaned up.
- Threads may wait on various events.
- To get a specific thread's context (addressability), a new thread in the process is required, or an APC may be queued to the existing thread, much like an IRB.
- Differences from OS/390: Only threads have priority; the process does not. And there are no lightweight threads like SRBs.

Comparing: Serialization and Synchronization

- Kernel-mode routines can define and use spin locks.
- Objects may be acquired with shared or exclusive control, like ENQ.
- Threads may wait on events such as I/O, timers and objects to be signalled.

Comparing: Memory Mapping

- Processes can share real frames by mapping pages to them. If a file is what is mapped, it is used for paging space.
- The mapped memory is subject to access control.
- OS/390 has IARVSERV to share memory and DIV for files. Only linear VSAM files are supported.

Comparing: Services

Rather like OS/390's subsystems and STCs, services provide important functions. Some characteristics:

- They execute with designated authority, typically that of system account.
- Usually they are started automatically late in boot, before operator logon.
- A service's start-up can depend on that of other services. NT's successful start-up can be affected by that of services.
- A service runs in a process, with its own thread.
- The Service Control Manager is used to start services and to pass other control directives and is network-aware. Thus SCM is somewhat like Master/JESx subsystems, CONSOLE, OMVS and XCF.

Examples of services: Anti-virus shields, TCP/IP, spooler, the logon service. Some device drivers are started via SCM, too.

Comparing: Job

Appearing in Windows 2000, the job groups processes for purposes of management (e.g., priority, working set minimum and maximum) and security.

In OS/390, there is the enclave for similar purposes.

Both mechanisms are relatively recent introductions.

Conclusions

- The list of similarities could have been longer, and a few more differences could have been adduced.
- Coming from different backgrounds and having different aims, OS/390 and Windows NT are involved, highly evolved systems that look like one another.
- I/O handling is very similar. Virtual addressability and the interrupt-driven, preemptive scheduler/dispatcher are rather similar.
- But scheduling, with IRQL in NT and more execution states (e.g., cross-memory) in OS/390, is different, too.
- The fundamental purpose of each is expediting units of work and handling interruptions generated by that work, so that mechanisms like IRQL seem less important.

Brothers? No. But probably cousins, one or two degrees removed.

References

These visuals and the paper are available via anonymous FTP:

ftp://ftp.s390.ibm.com/u/ftp/os390/xmemsrvc/OS390andWinNT.zip

- Ctrl2cap is a sample device driver. RegMon includes Instdrv, a sample routine to install a device driver. Both at http://www.sysinternals.com.
- *Inside Windows NT* (second edition), David A. Solomon. Microsoft Press, 1998. A wide-ranging survey of NT, including many internals.
- *Intel Architecture Software Developer's Manual, volume 3: System Programming Guide*, order number 243192 (this series of manuals pertains to Pentium II). Intel Corporation, 1997. The architecture and the instructions.
- Microsoft Windows NT 4 Device Driver Kit (DDK); this is part of the Microsoft Developer Network subscription. Specification of kernel services; numerous examples.
- *Showstopper!*, G. Pascal Zachary. Free Press, 1994. Blow-by-blow account of the development of Windows NT 3.0.
- *Windows NT Device Driver Development*, Peter G. Viscarola and W. Anthony Mason. Macmillan, 1998. Exceptionally thorough introduction to Windows NT and to writing device drivers.