Core System Mechanisms and Windows API

Roadmap for This Lecture

- Core System Mechanisms
  - Object Manager & Handles
  - System Worker Threads
  - Advanced Local Procedure Calls
  - Wow64
- The Windows APIs
  - Principles
  - Windows vs. Unix
  - File copy example

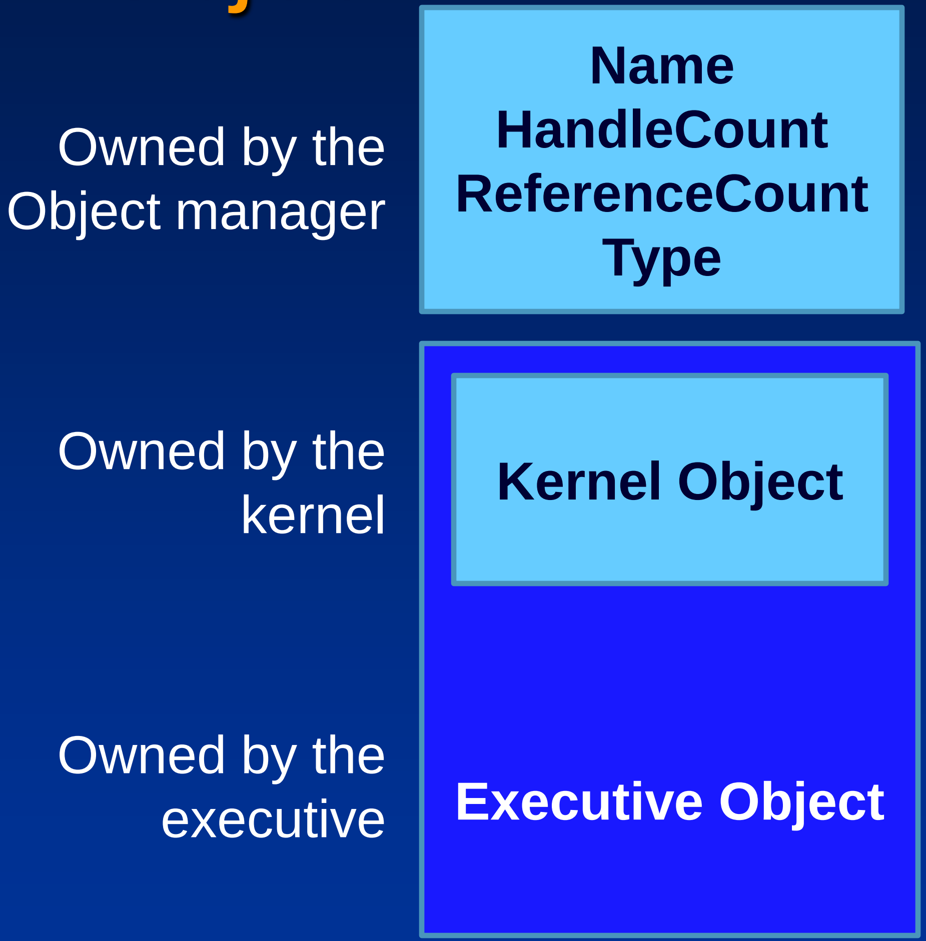
Object Manager (I)

- Executive component for managing system-defined “objects”
  - Manage: creating, deleting, protecting and tracking
  - Objects are data structures with optional names
  - “Objects” managed here include Executive objects and Kernel objects, but not Windows User/GDI objects (Win32k.sys)
  - Object manager implements **user-mode handles** and the **process handle table**
- Object manager is *not* used for *all* Windows data structures
  - Generally, only those types that need to be shared, named, or exported to user mode
  - Some data structures are called “objects” but are not managed by the object manager (e.g. “DPC objects”)

Object Manager (II)

- In part, a heap manager…
  - Allocates memory for data structure from system-wide, kernel space heaps (pageable or nonpageable)
- … with a few extra functions:
  - Assigns name to data structure (optional)
  - Allows lookup by name
  - Objects can be protected by ACL-based security
  - Provides uniform naming, sharing, and protection scheme
    - Simplifies C2 security certification by centralizing all object protection in one place
  - Maintains counts of handles and references (stored pointers in kernel space) to each object
    - Object cannot be freed back to the heap until all handles and references are gone

Executive Objects vs. Kernel Objects



Owned by the Object manager

Name
HandleCount
ReferenceCount
Type

Owned by the kernel

Kernel Object

Owned by the executive

Executive Object

Kernel objects are primitive objects implemented by the kernel

Executive objects are implemented by executive components e.g. process manager, memory manager, I/O subsystem, etc.
Executive objects can contain kernel objects

Executive Objects

Object type	Represents
Object directory	Container object for other objects: implement hierarchical namespace to store other object types
Symbolic link	Mechanism for referring to an object name indirectly
Process	Virtual address space and control information necessary for execution of thread objects
Thread	Executable entity within a process
Section	Region of shared memory (file mapping object in Windows API)
File	Instance of an opened file or I/O device
Port	Mechanism to pass messages between processes
Access token	Security profile (security ID, user rights) of a process or thread
Event	An object with a persistent state that can be used for synchronization or notification

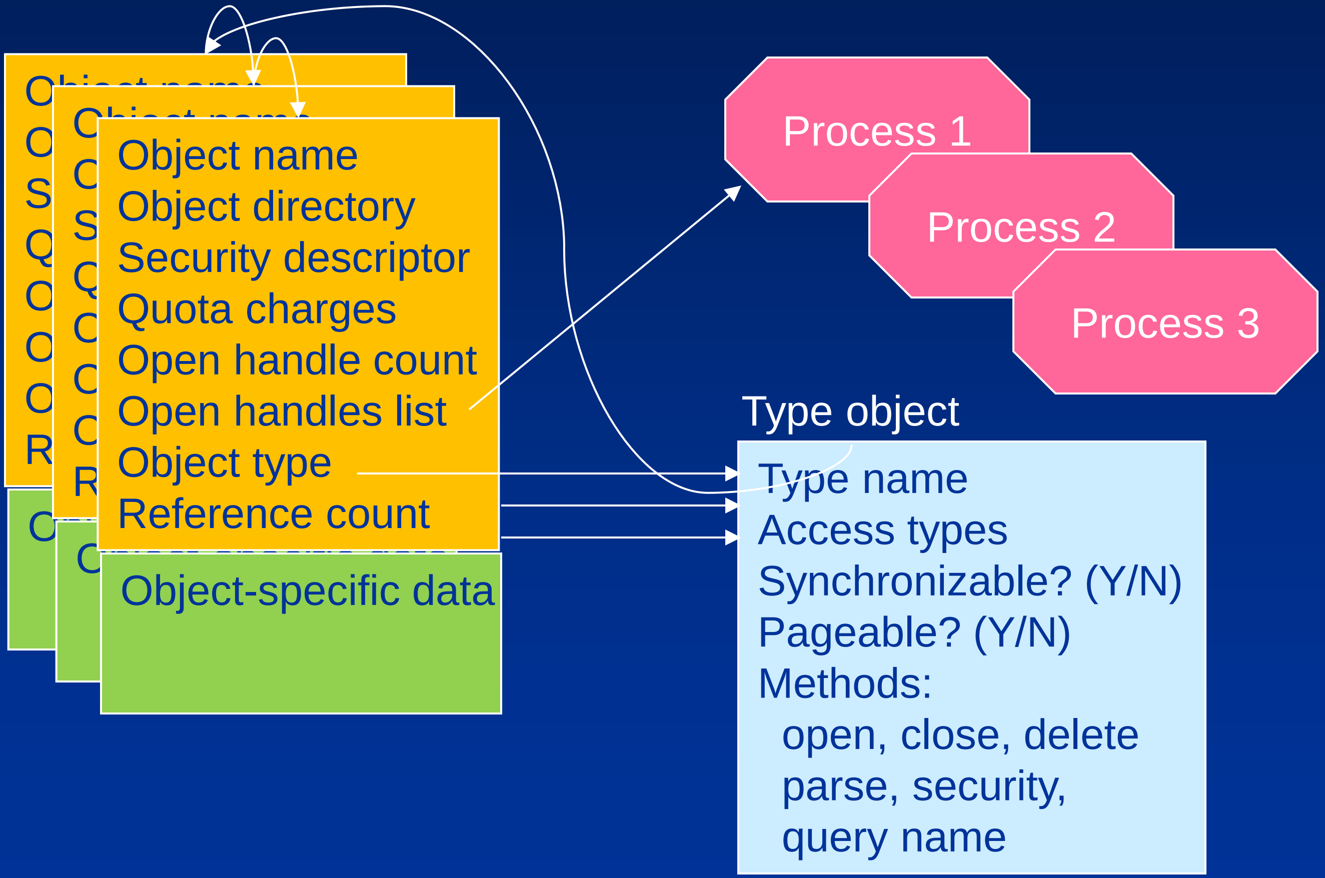
Executive Objects (contd.)

Object type	Represents
Semaphore	Counter and resource gate for critical section
Mutex	Synchronization construct to serialize resource access
Timer	Mechanism to notify a thread when a fixed period of time elapses
Queue	Method for threads to enqueue/dequeue notifications of I/O completions (Windows I/O completion port)
Key	Reference to registry data – visible in object manager namespace
Profile	Mechanism for measuring execution time for a process within an address range
Window Station	Contains a clipboard, a set of global atoms, a group of Desktop objects
Desktop	Has logical display surface and contains windows, menus and hooks

Object Structure

Object header
(owned by object manager)

Object body
(owned by executive component)



Object name
Object directory
Security descriptor
Quota charges
Open handle count
Open handles list
Object type
Reference count

Object-specific data

Process 1

Process 2

Process 3

Type object

Type name
Access types
Synchronizable? (Y/N)
Pageable? (Y/N)
Methods:
open, close, delete
parse, security,
query name

Object Header

Field	Purpose
Handle count	Number of currently opened handles to the object
Pointer count	Number of references to the object (>= handle count) Kernel components can refer to an object without opening a handle
Security descriptor	Determines who can use the object and what they can do with it. Unnamed objects cannot have security
Object type	Points to the **Type Object** that contains common attributes
Subheader offset	**Negative** offsets to the optional subheader structures, which if present, always precedes the object header
Flags	Characteristics and object attributes for the object

Type Object

- Contains data which remains constant for all objects of the same type
  - Type name
  - Access type
  - Some common methods (next slide)
- Saves memory
- If “object-tracking” flag is set, then type object links together all objects of the same type
  - Enumeration

Object Methods

Method	When method is called
Open	When an object handle is opened
Close	When an object handle is closed
Delete	Before the object manager deletes an object
Query name	When a thread requests the name of an object, such as a file, that exists in a secondary object domain
Parse	When the object manager is searching for an object name that exists in a secondary object domain
Security	When a process reads/changes protection of an objects, such as a file, that exists in a secondary object domain

Example:

- Process opens handle to object \Device\Floppy0\docs\resume.doc
- Object manager traverses name tree until it reaches Floppy0
- Calls parse method for object Floppy0 with arg \docs\resume.doc

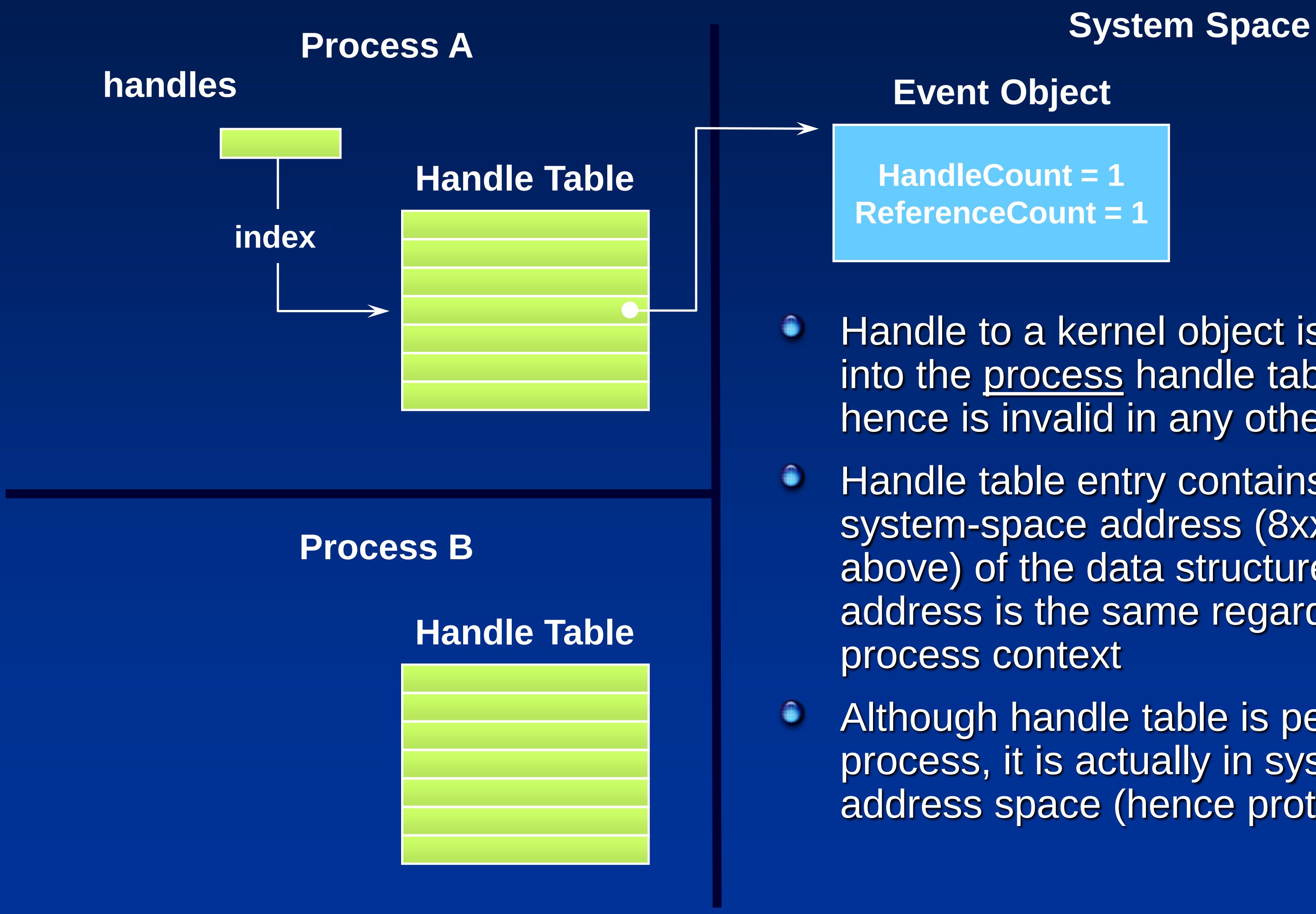
Objects and Handles

- When a process *creates* or *opens* an object, it receives a handle (or access) to the object
- Processes can also acquire handles by *inheritance*
- Benefits of handles:
  - Faster – no name lookups
  - Indirect pointers to objects – prevents direct fiddling with the system data structures
  - No difference between *file* handle, *process* handle or *event* handle – a consistent interface to reference *all* objects
  - All handle creation done by object manager – has exclusive rights to scrutinize every user action

Handles and Security

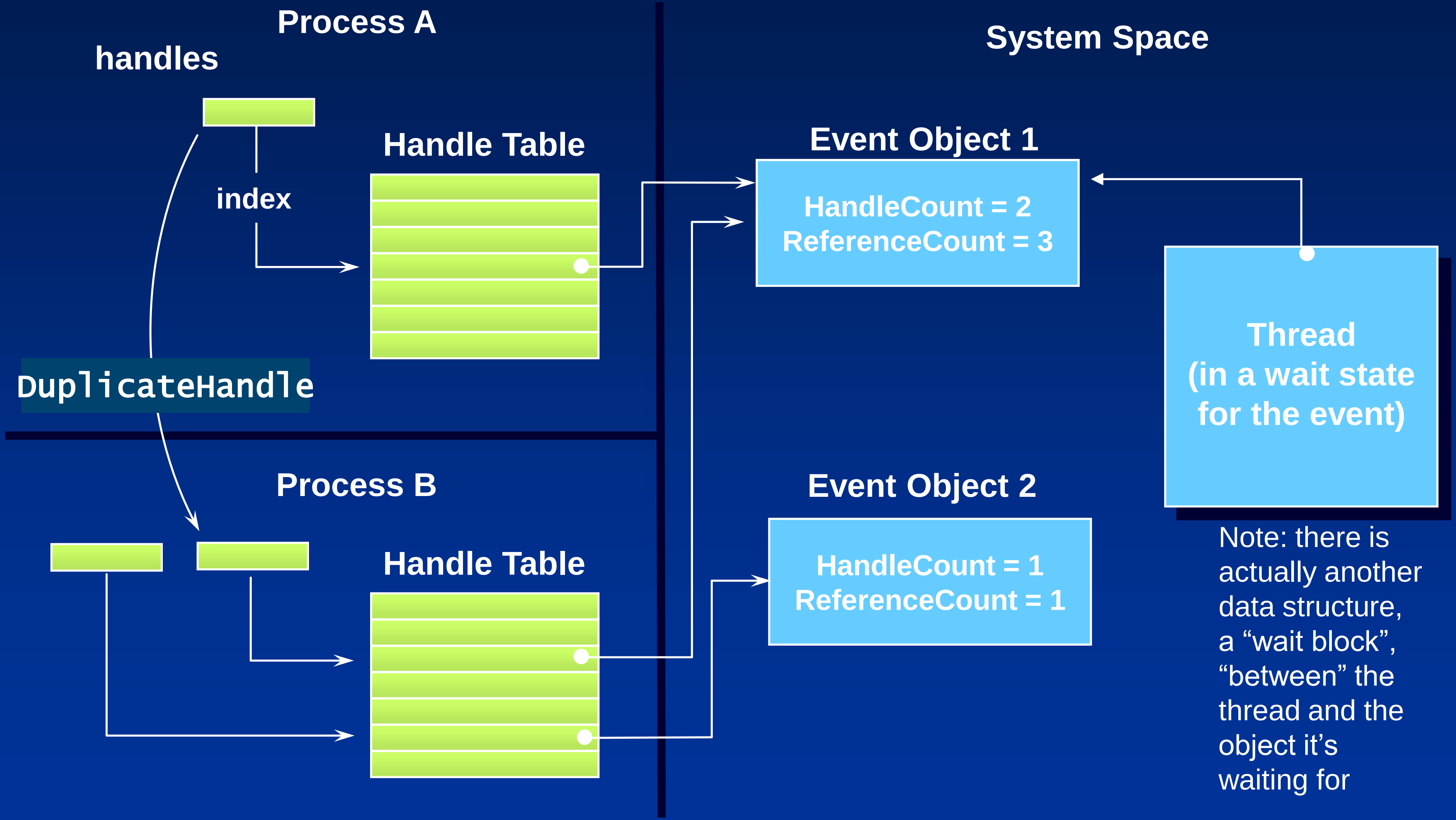
- Process handle table
    - Is unique for each process
    - But is in system address space, hence cannot be modified from user mode
    - Hence, is trusted
- Security checks are made when handle table entry is created
    - i.e. at CreateXXX time
    - Handle table entry indicates the "validated" access rights to the object
        - Read, Write, Delete, Terminate, etc.
- APIs that take an "already-opened" handle look in the handle table entry before performing the function
    - For example: TerminateProcess checks to see if the handle was opened for Terminate access
    - No need to check file ACL, process or thread access token, etc., on every write request---checking is done at file handle creation, i.e. "file open", time

Handles, Pointers, and Objects



- Handle to a kernel object is an index into the process handle table, and hence is invalid in any other process
- Handle table entry contains the system-space address (8xxxxxxx or above) of the data structure; this address is the same regardless of process context
- Although handle table is per-process, it is actually in system address space (hence protected)

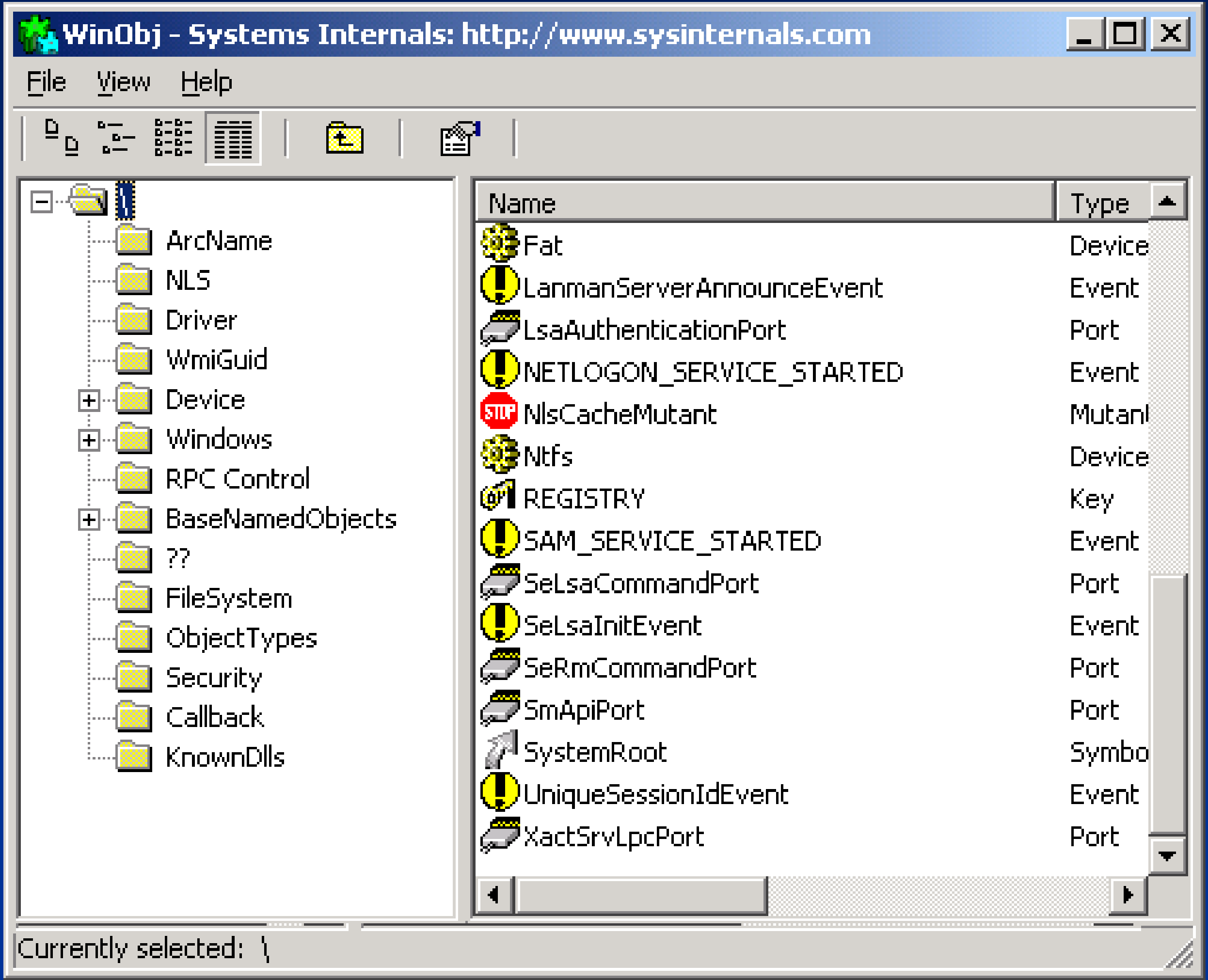
Handle and Reference Count



Note: there is actually another data structure, a “wait block”, “between” the thread and the object it’s waiting for

Object Manager Namespace

- System and session-wide internal namespace for all objects exported by the operating system
- View with Winobj from www.sysinternals.com



\
- ArcName
- NLS
- Driver
- WmiGuid
- Device
- Windows
- RPC Control
- BaseNamedObjects
- ??
- FileSystem
- ObjectTypes
- Security
- Callback
- KnownDlls

Name	Type
Fat	Device
LanmanServerAnnounceEvent	Event
LsaAuthenticationPort	Port
NETLOGON_SERVICE_STARTED	Event
NlsCacheMutant	Mutant
Ntfs	Device
REGISTRY	Key
SAM_SERVICE_STARTED	Event
SeLsaCommandPort	Port
SeLsaInitEvent	Event
SeRmCommandPort	Port
SmApiPort	Port
SystemRoot	Symbo
UniqueSessionIdEvent	Event
XactSrvLpcPort	Port

Currently selected: \

Interesting Object Directories

- in \ObjectTypes
  - objects that define types of objects
- in \BaseNamedObjects

  these will appear when Windows programs use CreateEvent, etc.

  - mutant (Windows mutex)
  - queue (Windows I/O completion port)
  - section (Windows file mapping object)
  - event
  - Semaphore
- In \GLOBAL??
  - DOS device name mappings for console session

Object Manager Namespace

- Namespace:
  - Hierarchical directory structure (based on file system model)
  - System-wide (not per-process)
    - With Terminal Services, Windows objects are per-session by default
    - Can override this with “global\” prefix on object names
  - Volatile (not preserved across boots)
    - As of Server 2003, requires SeCreateGlobalPrivilege
  - Namespace can be extended by secondary object managers (e.g. file system)
    - Hook mechanism to call external parse routine (method)
  - Supports case sensitive or case blind
  - Supports symbolic links (used to implement drive letters, etc.)
- Lookup done two occasions:
  - Creates a named object – check for existing names
  - Opens a handle to a named object
- Not all objects managed by the object manager are named
  - e.g. file objects are not named (they are named in the secondary obj manager (file system)
  - un-named objects are not visible in WinObj

System Worker Threads

- Created at system initialization time
- Perform work on behalf of other threads
- Most device drivers and executive components use system worker threads
- Request system worker thread service by calling
  - *ExQueueWorkItem* or *IoQueueWorkItem* functions
  - Put a work item on a *queue dispatcher* object
- System worker threads look for work from the queue dispatcher
- Three types of system worker threads (and default #):

Delayed worker threads (pri 12):	7	(deferred object deletion)
Critical worker threads (pri 13):	5	(used by time-critical items)
Hypercritical worker threads (Pri 15):	1	(used by process manager)

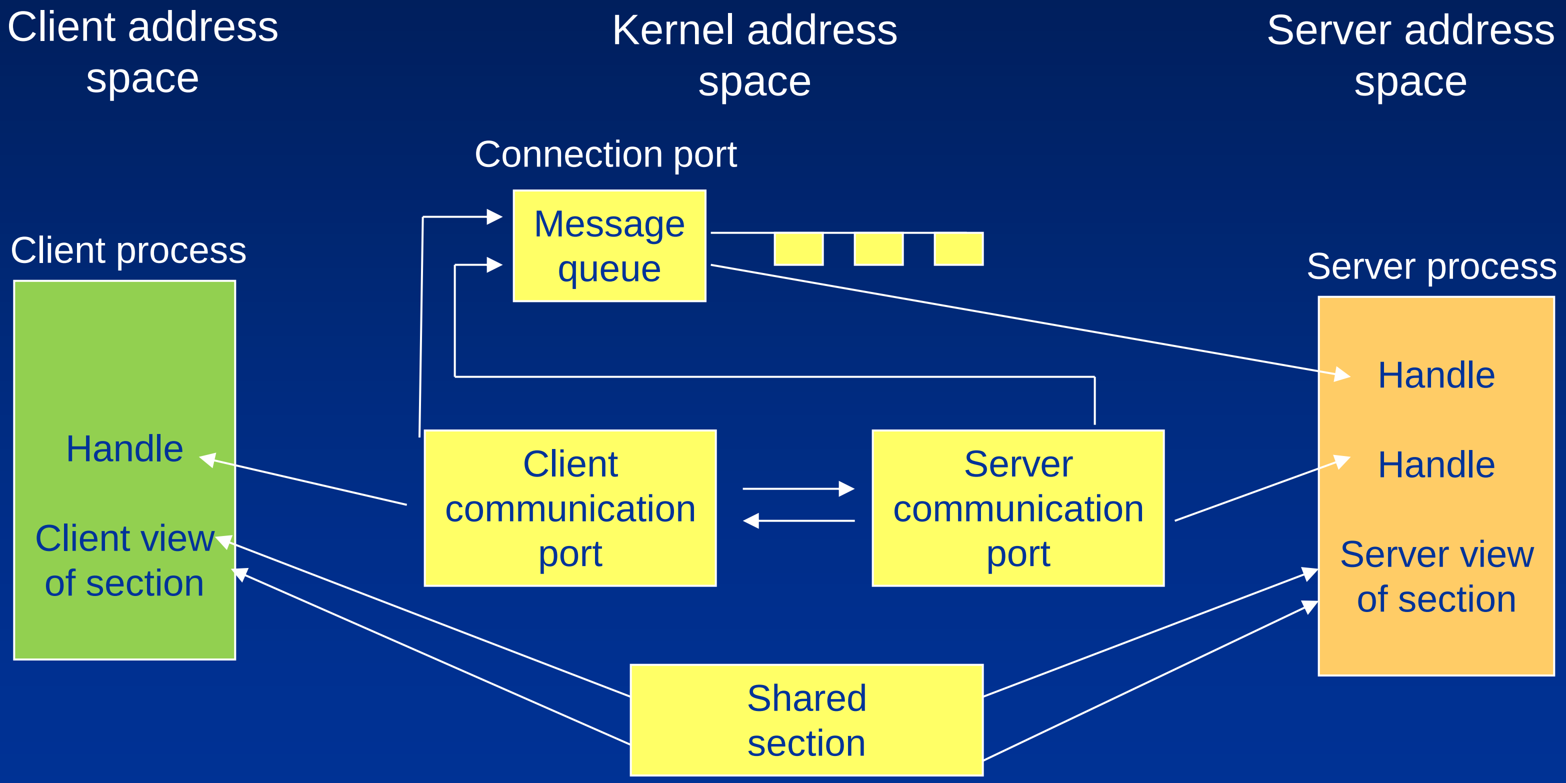
Advanced Local Procedure Calls (ALPCs)

- IPC – high-speed message passing
- Not available through Windows API – Windows OS internal
- Application scenarios:
  - RPCs on the same machine are implemented as ALPCs
  - Some Windows APIs result in sending messages to Windows subsystems processes
  - WinLogon uses ALPC to communicate with local security authentication server process (LSASS)
  - Security reference monitor uses ALPC to communicate with LSASS
- ALPC communication:
  - Short messages < 256 bytes are copied from sender to receiver
  - Larger messages are exchanged via shared memory segment
  - For data larger than will fit in shared section, server (kernel) may write directly in client‘s address space

Port Objects

- ALPC exports port objects to maintain state of communication:
  - **Server connection port**: named port, server connection request point
  - **Server communication port**: unnamed port, one per active client, used for communication
  - **Client communication port**: unnamed port a particular client thread uses to communicate with a particular server
- Typical scenario:
  - Server creates named connection port
  - Client makes connection request
  - Two unnamed ports are created, client gets handle to server port, server gets handle to client port
  - These two new ports will be used for communication

Use of ALPC ports



Client address space

Kernel address space

Server address space

Connection port

Message queue

Client process

Server process

Handle

Client view of section

Client communication port

Server communication port

Handle

Handle

Server view of section

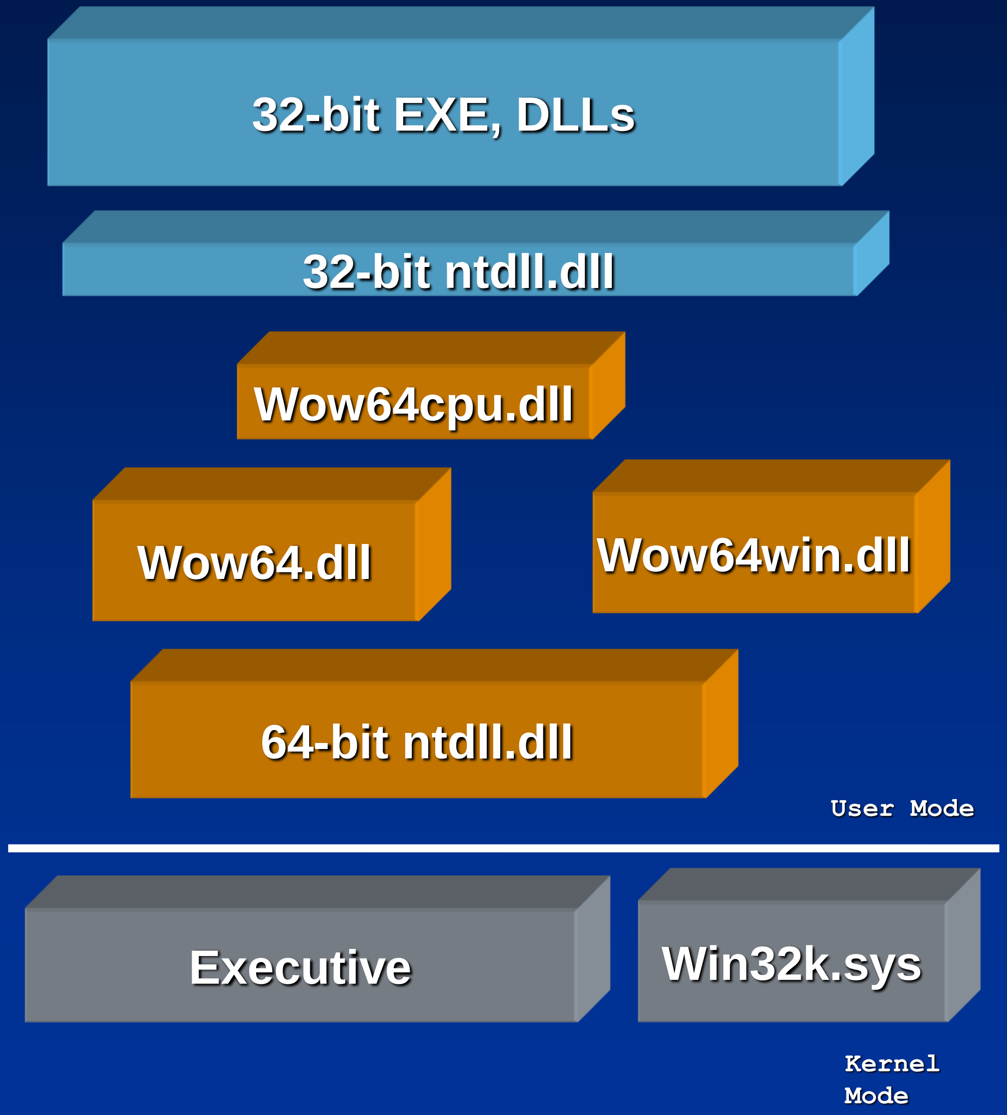
Shared section

Wow64

- Allows execution of Win32 binaries on 64-bit Windows
  - Wow64 intercepts system calls from the 32-bit application
    - Converts 32-bit data structure into 64-bit aligned structures
    - Issues the native 64-bit system call
    - Returns any data from the 64-bit system call
- *IsWow64Process()* function can tell if a 32-bit process is running under Wow64
- Performance
  - On x64, instructions executed by hardware
  - On IA64, instructions have to be emulated
    - New Intel IA-32 EL (Execution Layer) does binary translation of Itanium to x86 to improve performance
      - Downloadable now – bundled with Server 2003 SP1

Wow64 Components

- Wow64.dll - provides core emulation infrastructure, and hooks exception dispatching and base system calls by Ntoskrnl.exe
- Wow64win.dll - Intercepts GUI system calls exported by Win32k.sys
- Wow64cpu.dll – manages thread contexts, supports mode-switch instructions

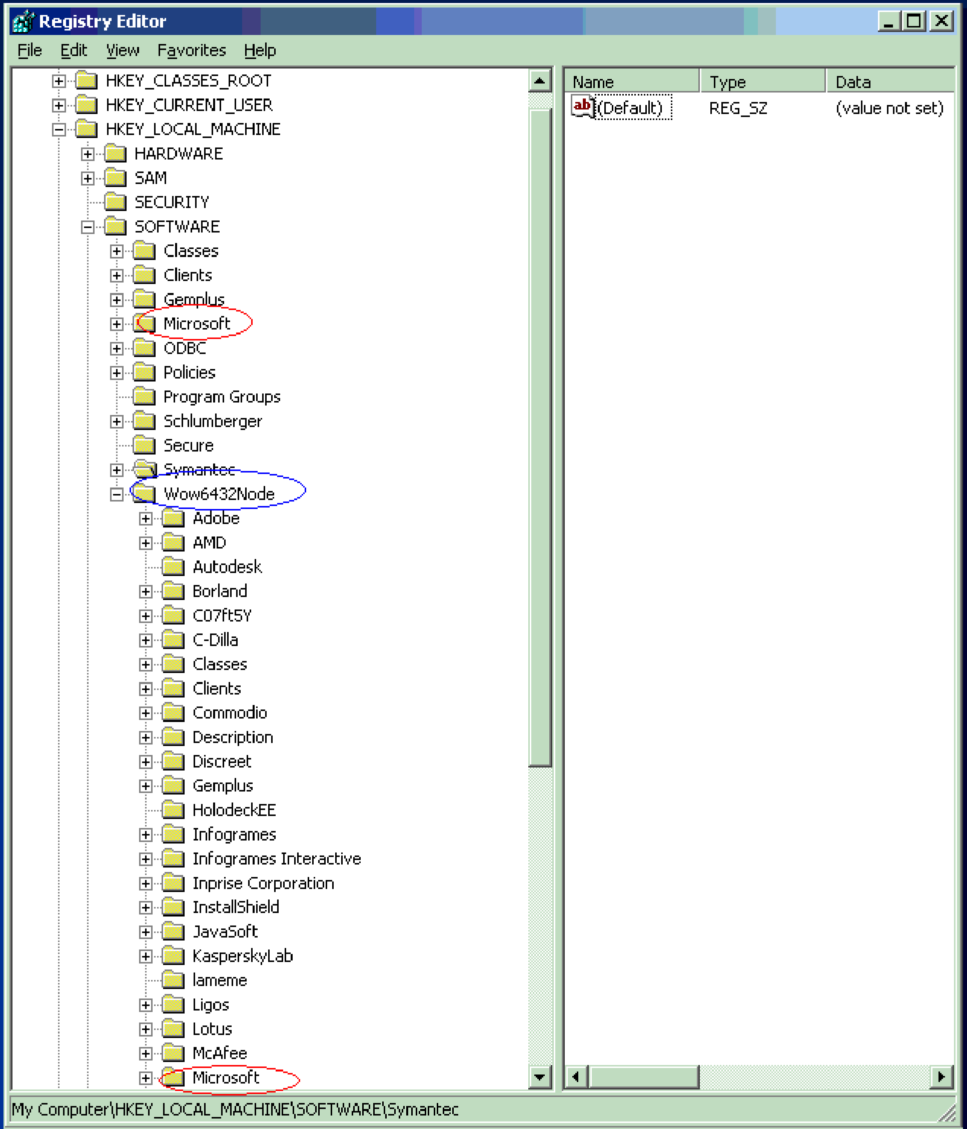


Wow64 File Locations

- Location of system files
  - 64-bit system files are in \windows\system32
  - 32-bit system files are in \windows\syswow64
  - 32-bit applications live in “\Program Files (x86)”
  - 64-bit applications live in “\Program Files”
- File access to %windir%\system32 redirected to %windir%\syswow64
- %PROGRAMFILES% set to the appropriate program directory
- Two areas of the registry redirected (see next slide)

Wow64 Registry Redirection

- Two registry keys have 32-bit sections:
  - HKEY_LOCAL_MACHINE\Software
  - HKEY_CLASSES_ROOT
  - Everything else is shared
- 32-bit data lives under \Wow6432Node
  - When a Wow64 process opens/creates a key, it is redirected to be under Wow6432Node



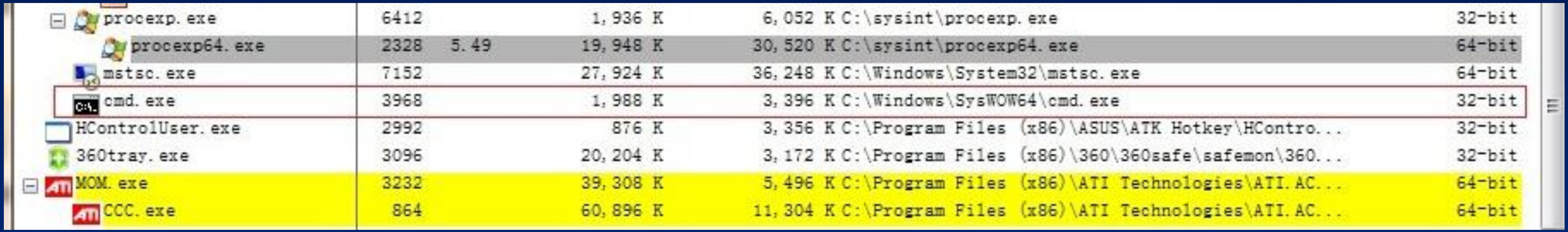
- HKEY_CLASSES_ROOT
- HKEY_CURRENT_USER
- HKEY_LOCAL_MACHINE
  - HARDWARE
  - SAM
  - SECURITY
  - SOFTWARE
    - Classes
    - Clients
    - Gemplus
    - Microsoft
    - ODBC
    - Policies
    - Program Groups
    - Schlumberger
    - Secure
    - Symantec
    - Wow6432Node
      - Adobe
      - AMD
      - Autodesk
      - Borland
      - C07ft5Y
      - C-Dilla
      - Classes
      - Clients
      - Commodio
      - Description
      - Discreet
      - Gemplus
      - HolodeckEE
      - Infogrames
      - Infogrames Interactive
      - Inprise Corporation
      - InstallShield
      - JavaSoft
      - KasperskyLab
      - lameme
      - Ligos
      - Lotus
      - McAfee
      - Microsoft

Name	Type	Data
(Default)	REG_SZ	(value not set)

My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Symantec

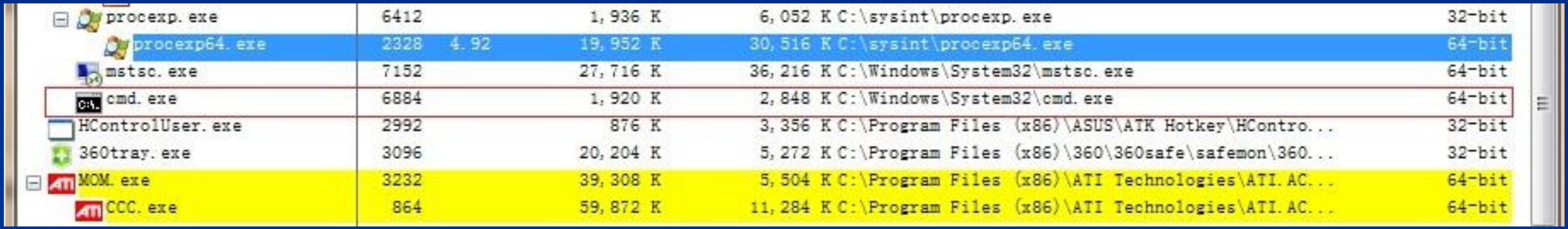
Example: Cmd.exe on 64-bit System

- 32-bit Cmd.exe process:



procexp.exe	6412		1,936 K	6,052 K	C:\sysint\procexp.exe	32-bit
procexp64.exe	2328	5.49	19,948 K	30,520 K	C:\sysint\procexp64.exe	64-bit
mstsc.exe	7152		27,924 K	36,248 K	C:\Windows\System32\mstsc.exe	64-bit
cmd.exe	3968		1,988 K	3,396 K	C:\Windows\SysWOW64\cmd.exe	32-bit
HControlUser.exe	2992		876 K	3,356 K	C:\Program Files (x86)\ASUS\ATK Hotkey\HContro...	32-bit
360tray.exe	3096		20,204 K	3,172 K	C:\Program Files (x86)\360\360safe\safemon\360...	32-bit
MOM.exe	3232		39,308 K	5,496 K	C:\Program Files (x86)\ATI Technologies\ATI.AC...	64-bit
CCC.exe	864		60,896 K	11,304 K	C:\Program Files (x86)\ATI Technologies\ATI.AC...	64-bit

- 64-bit Cmd.exe process:



procexp.exe	6412		1,936 K	6,052 K	C:\sysint\procexp.exe	32-bit
procexp64.exe	2328	4.92	19,952 K	30,516 K	C:\sysint\procexp64.exe	64-bit
mstsc.exe	7152		27,716 K	36,216 K	C:\Windows\System32\mstsc.exe	64-bit
cmd.exe	6884		1,920 K	2,848 K	C:\Windows\System32\cmd.exe	64-bit
HControlUser.exe	2992		876 K	3,356 K	C:\Program Files (x86)\ASUS\ATK Hotkey\HContro...	32-bit
360tray.exe	3096		20,204 K	5,272 K	C:\Program Files (x86)\360\360safe\safemon\360...	32-bit
MOM.exe	3232		39,308 K	5,504 K	C:\Program Files (x86)\ATI Technologies\ATI.AC...	64-bit
CCC.exe	864		59,872 K	11,284 K	C:\Program Files (x86)\ATI Technologies\ATI.AC...	64-bit

Wow64 Limitations

- Cannot load 32-bit DLLs in 64-bit process and vice versa
- Does not support 32-bit kernel mode device drivers
  - Drivers must be ported to 64-bits
  - Special support required to support 32-bit applications using *DeviceIoControl* to driver
    - Driver must convert 32-bit structures to 64-bit

Wow64 Feature Support on 64-bit Windows	Platforms	
	IA64	x64
16-bit Virtual DOS Machine (VDM) support	N/A	N/A
Physical Address Extension (PAE) APIs	N/A	Yes
GetWriteWatch() API	N/A	Yes
Scatter/Gather I/O APIs	N/A	Yes
Hardware accelerated with DirectX version 7,8 and 9	Software-Emulation Only	Yes

Windows API - Overview

- APIs to Windows systems evolved over time:
    - Win16 - introduced with Windows 2.0
    - Win32 - introduced with Windows NT, Windows 95
    - Win64 – introduced with Windows 64-bit edition
- “Windows API” summarizes all of the above
    - In this course, Windows API refers to Win32 and Win64

Windows API - major functionality

- File System and Character I/O
- Direct File Access and File Attributes
- Structured Exception Handling
- Memory Management and Memory-Mapped Files
- Security
- Process Management
- Inter-process Communication
- Threads and Scheduling, Windows Synchronization

Windows API Principles

- System resources are *kernel objects* referenced by a *handle* (handle vs. UNIX file descriptors & PIDs)
- *Kernel objects* must be manipulated via Windows API
- Objects – files, processes, threads, IPC pipes, memory mappings, events – have security attributes
- Windows API is rich & flexible:
  - convenience functions often combine common sequences of function calls
- Windows API offers numerous synchronization and communication mechanisms

Windows API principles (contd.)

- Thread is unit of executions
  (vs. process in Unix)
  - A process can contain one or more threads
- Function names are long and descriptive
  (as in VMS)
  - *WaitForSingleObject()*
  - *WaitForMultipleObjects()*

Windows API Naming Conventions

- Predefined data types are in uppercase
  - BOOL (32 bit object to store single logical value)
  - HANDLE
  - DWORD (32 bit unsigned integer)
  - LPTSTR
  - LPSECURITY_ATTRIBUTE
- Prefix to identify pointer & const pointer
  - LPTSTR (defined as TCHAR *)
  - LPCTSTR (defined as const TCHAR *)

    (Unicode: *TCHAR* may be 1-byte *char* or 2-byte *wchar_t*)
  - See \$MSDEV\INCLUDE\WINDOWS.H, WINNT.H, WINBASE.H

    (MSDEV=C:\Program Files\Microsoft Visual Studio\VC\)

64-bit vs. 32-bit Windows APIs

- Pointers and types derived from pointer, e.g. handles, are 64-bit long
  - A few others go 64, e.g. WPARAM, LPARAM, LRESULT, SIZE_T
  - Rest are the same, e.g., 32-bit INT, DWORD, LONG
- Only five replacement APIs!
  - Four for Window/Class Data
    - Replaced by Polymorphic (_ptr) versions
    - Updated constants used by these APIs
  - One (_ptr) version for flat scroll bars properties

Win32 and Win64 are referred to as the Windows API

API	Data Model	`int`	`long`	pointer
Win32	ILP32	32	32	32
Win64	LLP64 (P64)	32	32	64
UNIXes	LP64	32	64	64

Differences from UNIX

- HANDLEs are opaque (no short integers)
  - No analogy to file descriptors 0,1,2 in Windows
- No distinctions between HANDLE and process ID
  - Most functions treat file, process, event, pipe identically
- Windows API processes have no parent-child relationship
  - Although the Windows kernel keeps this information
- Windows text files have CR-LF instead of LF (UNIX)
- Anachronisms: “long pointer“ (32 bit)
  - LPSTR, LPVOID

Portability: The Standard C Library

- Included in the Windows API
- C library contains functions with limited capability to manage OS resources (e.g.; files)
- Often adequate for simple programs
- Possible to write portable programs
- Include files:
    - <stdlib.h>, <stdio.h>, <string.h>

Example Application

- Sequential file copy:
  - The simplest, most common, and most essential capability of any file system
  - Common form of sequential processing
- Comparing programs:
  - Quick way to introduce Windows API essentials
  - Contrast different approaches
  - Minimal error processing

Sequential File Copy

UNIX:

- File descriptors are integers; error value: -1
- read()/write() return number of bytes processed,
  - 0 indicates EOF
  - Positive return value indicates success
- close() works only for I/O objects
- I/O is synchronous
- Error processing depends on perror() & errno (global)

Basic cp file copy program. UNIX Implementation

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <errno.h>
#define BUF_SIZE 256

int main (int argc, char *argv []) {
   int input_fd, output_fd;
   ssize_t bytes_in, bytes_out;
   char rec [BUF_SIZE];
   if (argc != 3) {
      printf ("Usage: cp file1 file2\n");
      return 1;
   }
   input_fd = open (argv [1], O_RDONLY);
   if (input_fd == -1) {
      perror (argv [1]); return 2;
   }
   output_fd =
      open(argv[2],O_WRONLY|O_CREAT,0666);
   if (output_fd == -1) {
      perror (argv [2]); return 3;
}
```

```
/* Process the input file a record
   at atime. */


while ((bytes_in = read
   (input_fd, &rec, BUF_SIZE)) > 0) {
      bytes_out =
         write (output_fd, &rec, bytes_in);
      if (bytes_out != bytes_in) {
         perror ("Fatal write error.");
         return 4;
      }
   }
   close (input_fd);
   close (output_fd);
   return 0;
}
```

File Copy with Standard C Library

- Open files identified by pointers to FILE structures
  - NULL indicates invalid value
  - Pointers are “handles” to open file objects
- Call to fopen() specifies whether file is text or binary
- Errors are diagnosed with perror() of ferror()

- Portable between UNIX and Windows
- Competitive performance
- Still constrained to synchronous I/O
- No control of file security via C library

Basic cp file copy program. C library Implementation

```
#include <stdio.h>
#include <errno.h>
#define BUF_SIZE 256

int main (int argc, char *argv []) {
   FILE *in_file, *out_file;
   char rec [BUF_SIZE];
   size_t bytes_in, bytes_out;
   if (argc != 3) {
      printf ("Usage: cp file1 file2\n");
      return 1;
   }
   in_file = fopen (argv [1], "rb");
   if (in_file == NULL) {
      perror (argv [1]);
      return 2;
   }
   out_file = fopen (argv [2], "wb");
   if (out_file == NULL) {
      perror (argv [2]);
      return 3;
   }
```

```
/* Process the input file a record
at a time. */

while ((bytes_in =
   fread (rec,1,BUF_SIZE,in_file)) > 0) {
   bytes_out =
      fwrite (rec, 1, bytes_in, out_file);
   if (bytes_out != bytes_in) {
      perror ("Fatal write error.");
      return 4;
   }
}

fclose (in_file);
fclose (out_file);
return 0;
}
```

File Copying with Windows API

- <windows.h> imports all Windows API function definitions and data types
- Access Windows objects via variables of type HANDLE
- Generic CloseHandle() function works for most objects
- Symbolic constants and flags
  - INVALID_HANDLE_VALUE, GENERIC_READ
- Functions return boolean values
- System error codes obtained via GetLastError()
- Windows security is complex and difficult to program

Basic cp file copy program. Windows API Implementation

```
#include <windows.h>
#include <stdio.h>
#define BUF_SIZE 256

int main (int argc, LPTSTR argv []) {
   HANDLE hIn, hOut;
   DWORD nIn, nOut;
   CHAR Buffer [BUF_SIZE];
   if (argc != 3) {
      printf("Usage: cp file1 file2\n");
      return 1;
   }
   hIn = CreateFile (argv [1],
         GENERIC_READ,
         FILE_SHARE_READ, NULL,
         OPEN_EXISTING,
         FILE_ATTRIBUTE_NORMAL,
         NULL);
   if (hIn == INVALID_HANDLE_VALUE) {
      printf ("Input file error:%x\n",
             GetLastError ());
      return 2;
   }
```

```
   hOut = CreateFile (argv [2],
            GENERIC_WRITE, 0, NULL,
            CREATE_ALWAYS,
            FILE_ATTRIBUTE_NORMAL,
            NULL);
   if (hOut == INVALID_HANDLE_VALUE) {
      printf("Output file error: %x\n",
             GetLastError ());
      return 3;
   }
   while (ReadFile (hIn, Buffer,
               BUF_SIZE, &nIn, NULL)
               && nIn > 0) {
      WriteFile (hOut, Buffer,nIn,&nOut,NULL);
      if (nIn != nOut) {
         printf ("Fatal write error: %x\n",
                  GetLastError ());
         return 4;
      }
   }
   CloseHandle (hIn);
   CloseHandle (hOut);
   return 0;
}
```

File Copying with Windows API Convenience Functions

- Convenience functions may improve performance
  - Programmer does not need to be concerned about arbitrary buffer sizes
  - OS manages speed vs. space tradeoffs at runtime

```
#include <windows.h>
#include <stdio.h>

int main (int argc, LPTSTR argv [])
{
    if (argc != 3) {
        printf ("Usage: cp file1 file2\n"); return 1;
    }
    if (!CopyFile (argv [1], argv [2], FALSE)) {
        printf ("CopyFile Error: %x\n", GetLastError ()); return 2;
    }
    return 0;
}
```

Further Reading

- Mark E. Russinovich *et al.*, Microsoft Windows Internals, 5th Edition, Microsoft Press, 2009, Chapter 3 - System Mechanisms
  - Object Manager (from pp. 133)
  - System Worker Threads (from pp. 198)
  - Advanced Local Procedure Calls (ALPCs) (from pp. 202)
  - Wow64 (from pp. 211)
- Johnson M. Hart, Win32 System Programming: A Windows® 2000 Application Developer's Guide, 2nd Edition, Addison-Wesley, 2000.
  - (This book discusses select Windows programming problems and addresses the problem of portable programming by comparing Windows and Unix approaches).
- Jeffrey Richter, Programming Applications for Microsoft Windows, 4th Edition, Microsoft Press, September 1999.
  - (This book provides a comprehensive discussion of the Windows API – suggested reading).

Source Code References

- Windows Research Kernel sources
  - \base\ntos\ob – Object Manager
  - \base\ntos\ex\handle.c – handle management
  - \base\ntos\ex\pool.c, \base\ntos\inc\pool.h – Kernel memory pools (nonpaged, paged)
    - Also see \base\ntos\mm\allocpag.c
  - \base\ntos\lpc – Local Procedure Call
  - exceptn.c, trap.asm in \base\ntos\ke\i386, \base\ntos\ke\amd64 – Exception Dispatching

Lab: 2013-9-23

Handles & ALPC

Viewing Handles

- Handle: a non-transparent pointer
- Use Handle.exe
- Use Process Explorer
- View the Maximum number of handles

ALPC Port Objects

- Use Winobj.exe to view ALPC Port Objects