MCA Part-II

Paper-XIII: Operating System

Topic: Windows 2000

PREPARED BY: DR. KIRAN PANDEY

(School of Computer Science)

Email-id: kiranpandey.nou@gmail.com

INTRODUCTION

Microsoft Windows 2000, Win2K in short, is the WindowsNT series 32 bit Windows operating system released by Microsoft in 1999. It is called Windows NT5.0 at first. The English version is released in December 19, 1999, and Chinese version is released in the spring of next year. Windows2000 is a interruptible and imaging operating system that faces business environment. It is designed for single CPU or SMP 32 bit Intel X86 computer. There are four versions of Windows2000:

Windows 2000 Professional

It is used for workstation and Notebook. The original name is Windows NT5.9 Workstation. It supports two balanced multiprocessors at the most. The minimum internal storage is 64MB and maximum 4GB.

Windows 2000 Server

It is developed for the small enterprises server field. Its original name is WindowsNT5.0 Server. It supports 4 CPU on one computer at the most. The minimum internal storage is 128MB and maximum 4GB.

Windows 2000 Advanced Server

It is developed for the large and medium enterprises server field. Its original name is Windows NT 5.0 Server Enterprise Edition. It supports 8 CPU at the most. The minimum internal storage is 128MB and maximum 8GB.

Windows 2000 Datacenter Server

It is developed for the highest level and scalability, usability, and reliability of large enterprises or state organizations server field. It supports 8 or more CPU (the maximum 32 CPU). The minimum internal storage is 256MB and maximum 64GB.
In addition, Microsoft has provided the limited version Windows 2000 Advanced Server Limited Edition in 2001. It is used for the pure 64 bit microprocessor of Intel IA-64.

Windows 2000 is a complex operating system consisting of over 29 million lines of C /C++ code among which eight million lines of code is written for drivers. Windows 2000 is currently by far one of the largest commercial projects ever built.

Features of Windows 2000

Some of the significant features of Windows 2000 are:

- Support for FAT16, FAT32 and NTFS
- Increased uptime of the system and significantly fewer OS reboot scenarios
- Windows Installer tracks applications and recognizes and replaces missing components
- Protects memory of individual apps and processes to avoid a single app bringing the system down
- Encrypted File Systems protect sensitive data
- Secure Virtual Private Networking (VPN) supports tunneling in to private LAN over public Internet
- Personalized menus adapt to the way you work
- Multilingual version allows for User Interface and help to switch, based on logon
- Includes broader support for high-speed networking devices, including Native ATM and cable modems
- Supports Universal Serial Bus (USB) and IEEE 13910 for greater bandwidth devices.

Windows 2000 is really a NT 5.0, as it inherits many properties from NT 10.0. It is a true 32bit / 610bit multiprogramming system with protected processes. Each process consists of 32-bit (or 610 bit) demand paged virtual address space. The user process runs in user mode and operating system runs in kernel mode resulting in complete protection, thereby eliminating the Windows 98 flaws. Processes can have greater than one thread, which are scheduled by the operating system. It provides security for all files, directories, processes, and other shareable objects. And above all it supports for running on symmetric multiprocessors with upto 32 CPUs.

Windows 2000 is better than NT 10.0 with the Windows 98 user interface. It contains a number of features that are found only in Windows 98 like complete support for plug-and-play devices, the USB bus, FireWire (IEEE 13910), IrDA (Infrared link for portable computers/printers), power management etc. The new features in Windows 2000 operating system are:

- Active directory service,
- Security using Kerberos,

- Support for smart cards,
- System monitoring tools,
- Better integration of laptop computers with desktop computers,
- System management infrastructure,
- Single instance store, and job objects.
- The file system NTFS has been extended to support encrypted files, quotas, linked files, mounted volumes, and content indexing etc.
- Internationalization.

Windows 2000 consists of a single binary that runs everywhere in the world and the language can be selected at the run time. Windows 2000 uses Unicode, but a Windows 2000 does not have MS-DOS (use command line interface).

When the system is installed, version is recorded in the registry (internal database). At boot time, the operating system checks the registry to see the version.

WINDOWS 2000 PROGRAMMING

In this section we will see the details of programming interface and the registry, a small in-memory database.

Application Programming Interface

Windows 2000 has a set of system calls, which was not made public by Microsoft. Instead Microsoft has defined a set of function calls called the Win32 API (Application Programming Interface,) shown in Figure 1, which are publicly known and well documented. These are a set of library procedures that make system calls to perform certain jobs or do the work in the user space. These Win32 API calls do not change with new releases of Windows, although new API calls are added.

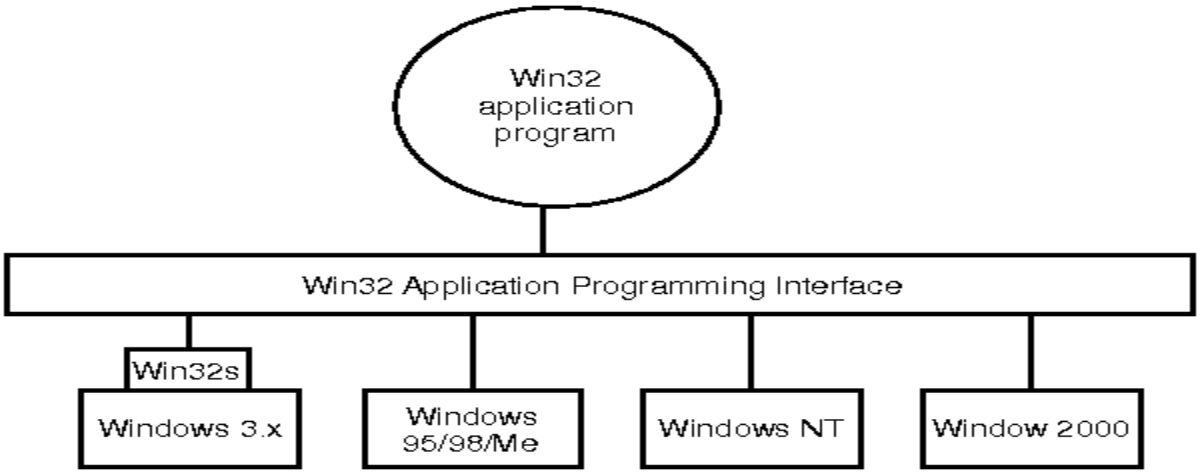

Figure 1: The Win32 Application Programming Interface

- Binary programs for the Intel x86 that conforms to Win32 API interface will run unmodified on all versions of Windows operating system. An extra library is required for Windows 3.x to match a subset of the 32-bit API calls to the 16-bit operating system. Windows 2000 has additional API calls, which will not function on older versions of Windows operating system.
- The Win32 API concept is totally different from the UNIX philosophy. In the case of UNIX, the system calls are all publicly known and form a minimal operating system and removing any of these would affect the functionally of the operating system. Whereas, Win32 provides a very comprehensive interface and the same thing can be performed in a number of ways.
- Most of Win32 API calls creates kernel objects including files, processes, threads, pipes, etc. and returns a result called a handle to the caller. Not all system-created data structures are objects and not all objects are kernel objects. True kernel objects are those that need to be named, protected, or shared in some way, has system defined type, well-defined operations on it and occupies storage in kernel memory. This handle is used to perform operations on the object and does the handle refer to specific to the process that created the object. Handles cannot be passed directly to another process (in the same manner as UNIX file descriptors cannot be passed to another process and used there). But is possible to duplicate a handle and then it can be passed to another processes in a protected manner, allowing them controlled access to the objects of their process. Every object has a security descriptor, containing information regarding who may and may not perform what kinds of operations on the object. Windows 2000 can also create and use objects.
- Windows 2000 is partially object oriented because the only way to manipulate objects is by invoking operations on their handles by invoking Win32 calls. But there is not concept of inheritance and polymorphism.
- Memory management system is invisible to programmer (demand paging), but a significant feature is visible, that is the ability of a process to map a file onto a region of its virtual memory.
- File I/O: A file is a linear sequence of bytes, in Win32 environment. Win32 environment provides over 60 calls for creating and destroying files/directories, opening, closing, reading, writing, requesting/setting attributes of files, etc.
- Every process has a process ID telling who it is and every object has an access control list describing who can access it and which operation are allowed, thus providing a fine-grained security.
- Windows 2000 file names are case sensitive and use the Unicode character set.

- On Windows 2000, all the screen coordinates given in the graphic function are true 32-bit numbers.

The Registry

All the information needed for booting and configuring the system and tailoring it to the current user was gathered in a big central database called the registry. It consists of a collection of directories, each of which contains either subdirectories or entries (the files). A directory is called a key and all top level key (directories) starts with the string HKEY (handle to key). At the bottom of the hierarchy are the entries, called values. Each value consists of three parts: a name, a type, and the data. At the top level, the Windows 2000 registry has six keys, called root keys, as shown below. One can view it using one of the registry editors (regedit or regedt32).

- HKEY_LOCAL_MACHINE
- HKEY-USERS
- HKEY_PERFORMANCE_DATA
- HKEY_CLASS_ROOT
- HKEY_CURRENT_CONFIG
- HKEY_CURRENT_USER

WINDOWS 2000 ARCHITECTURE

Windows 2000 consists of two major components:

- Operating system itself, which runs in kernel mode
- Environment subsystems, which run in user-mode.

The kernel handles the process management, memory management, file systems, and so on. The lowest two software layers, the HAL (hardware abstraction layer) and the kernel, are developed in C and in assembly language and are partly machine dependent. The upper layers are written in C and are almost machine independent. The drivers are written in C, or in a few cases C++.

There are two main components: (a) Kernel Mode: the operating system itself, which runs in kernel mode, and (b) User Mode: the environment subsystems. The kernel handles process management, memory management, file systems, and so on. The environment subsystem is separate processes which help user program carry out certain specific system functions. Windows 2000 follows modular structure. The simplified of Windows 2000 is given below in Figure 2.

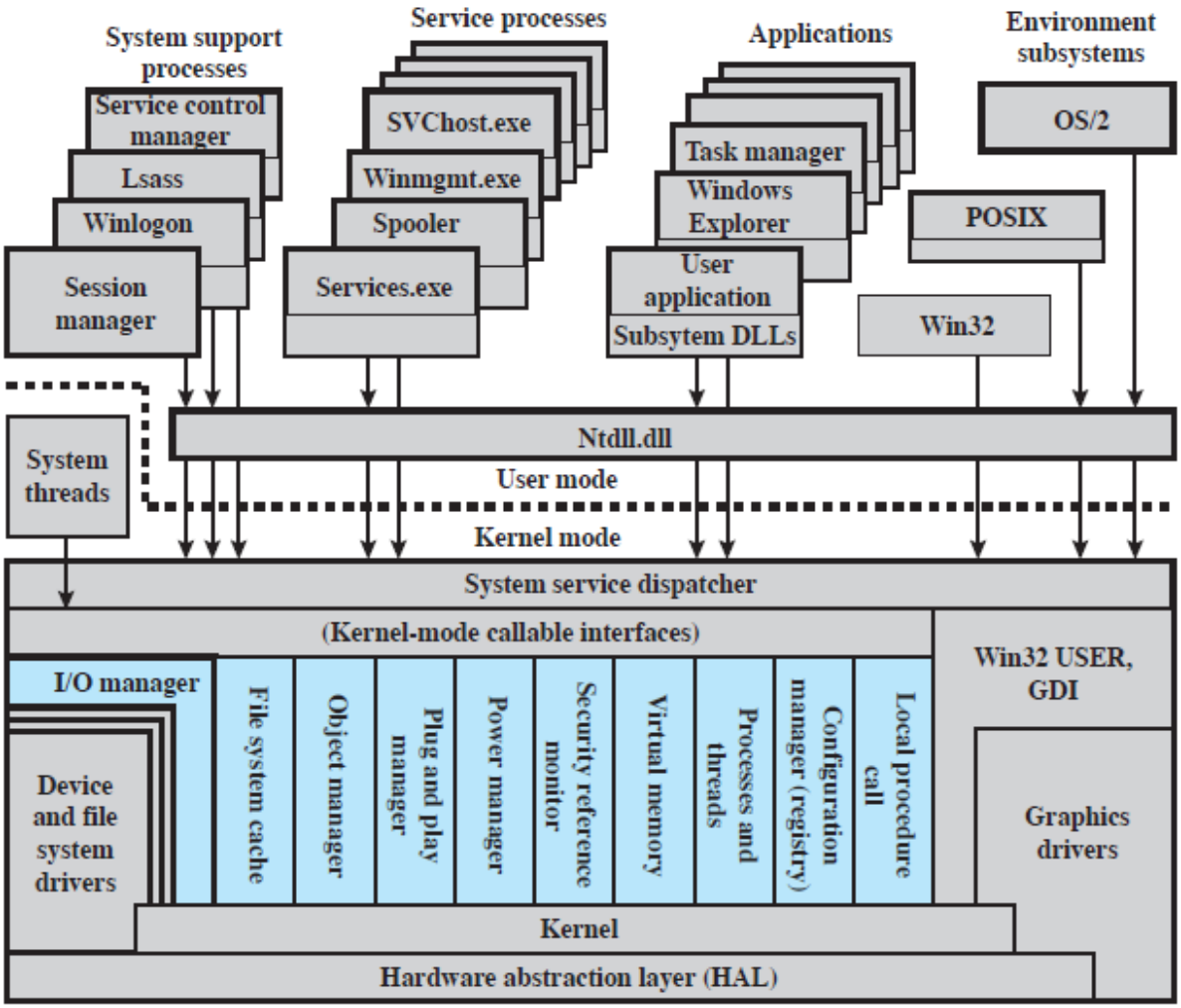

Figure 2: The structure of Windows 2000.

Hardware Abstraction Layer

Maps between generic hardware commands and responses and those unique to a specific platform. It isolates the operating system from platform-specific hardware differences. The HAL makes each machine's system bus, direct memory access (DMA) controller, interrupt controller, system timers, and memory module look the same to the kernel. It also delivers the support needed for symmetric multiprocessing (SMP), explained subsequently. It makes the operating system portable across platforms. Microsoft attempts to hide many of the machine dependencies in a HAL (Hardware Abstraction Layer). The job of the HAL is to present the rest of the operating system with abstract hardware devices. These devices are available in the form of machine-independent services (the procedure calls and macros). By not addressing the hardware directly, drivers and the kernel require fewer changes when being ported to a new hardware. Furthermore, HAL services are identical on all Windows 2000 systems

irrespective of the underlying hardware and porting HAL is simple because all the machine dependent code is located in one place and the goals of the port are well defined. The HAL address those services which relate to the chipset on the parent board and which vary from system to system. This hides the differences between one vendor's parent board and another one's, but not the differences between a Pentium and an Alpha. The various HAL services are given below, but HAL does not provide abstraction or services for specific I/O devices such as mouse, keyboard, or disk or for memory management unit:

- Access to device registers
- Bus-independent device addressing
- Interrupt handling
- Resetting
- DMA transfer
- Control of the times and real time clock
- Low-level spin locks and multiprocessor synchronization
- Interfacing with the BIOS and its CMOS configuration memory

Kernel Layer

The purpose of the kernel is to make the rest of the operating system hardware-independent. It accesses the hardware via the HAL and builds upon the extremely low level HAL services to build higher-level abstractions. In addition to providing a higher-level abstraction of the hardware and handling thread switches, the kernel also has another key function: providing low level support for control objects and dispatcher objects. It consists of the most used and most fundamental components of the operating System. The kernel manages thread scheduling, process switching, and exception and interrupt handling, and multiprocessor synchronization. The kernel's own code does not run in threads. Hence, it is the only part of the operating system that is not pre-emptible or pageable. Figure below shows some hardware functions:

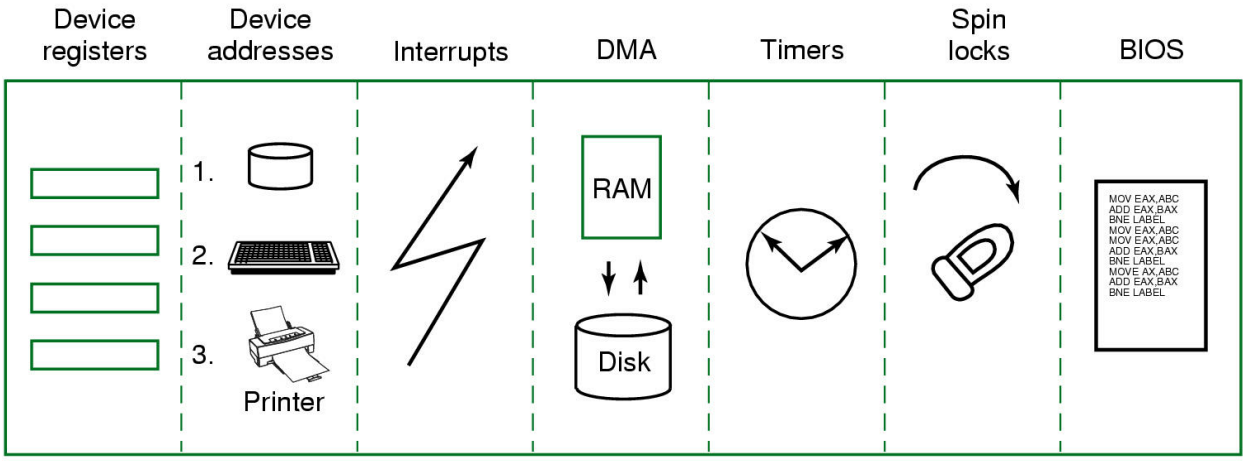

Figure 3: Some of the hardware functions the HAL manages

Executive

The executive is the upper portion of the operating system. It is written in C and is architecture independent. The executive consists of ten components, each of which is just a collection of procedures that work together to accomplish a certain task. These components are:

- **I/O manager:** Provides a framework through which I/O devices are accessible to applications, and is responsible for dispatching to the appropriate device drivers for further **processin**g. The I/O manager implements all the Windows I/O APIs and enforces security and naming for devices and file systems (using the object manager). Windows I/O.

- **Cache manager:** Improves the performance of file-based I/O by causing recently referenced disk data to reside in main memory for quick access, and by deferring disk writes by holding the updates in memory for a short time before sending them to the disk.

• **Object manager:** Creates, manages, and deletes Windows Executive objects and abstract data types that are used to represent resources such as processes, threads, and synchronization objects. It enforces uniform rules for retaining, naming, and setting the security of objects. The object manager also creates object handles, which consist of access control information and a pointer to the object. Windows objects are discussed later in this section.

• **Plug and play manager:** Determines which drivers are required to support a particular device and loads those drivers.

• **Power manager:** Coordinates power management among various devices and can be configured to reduce power consumption by putting the processor to sleep.

• **Security reference monitor:** Enforces access-validation and audit-generation rules. The Windows object-oriented model allows for a consistent and uniform view of security, right down to the fundamental entities that make up the Executive. Thus, Windows uses the same routines for access validation and for audit checks for all protected objects, in files, processes, address spaces, and I/O devices.

• **Virtual memory manager:** Maps virtual addresses in the process's address space to physical pages in the computer's memory.

• **Process/thread manager:** Creates and deletes objects and tracks process and thread objects.

• **Configuration manager:** Responsible for implementing and managing the system registry, which is the repository for both system wide and per-user settings of various parameters.

• **Local procedure call (LPC) Facility:** Enforces a client/server relationship between applications and executive subsystems within a single system, in a manner similar to a remote procedure call (RPC) facility used for distributed processing.

On booting, Windows 2000 is loaded into memory as a collection of files and the description of important files is given below in Table 1:

FILE NAME	DESCRIPTION
ntoskrnl	Kernel and executive
hal.dll	HALL
Win32k.sys	Win32 and GDI
*.sys	Driver files

Table 1: Important files in Windows 2000 OS

Device Drivers

Device drivers are not part of ntoskrnl.exe binary and when a drive is installed on the system, it is added to a list of the registry and is loaded dynamically on booting. A device driver can control one or more I/O devices and can perform encryption of a data stream or just providing access to kernel data structures. The largest device drivers are Win32 GDI and video, handles system calls and most of the graphics.

Objects

Objects are most important concept in Windows 2000. Objects provide a uniform and consistent interface to all system resources and data structures such as threads, processes, semaphores, etc. An object is a data structure in RAM. A file on disk is an object, but an object is created for file when it is opened. When a system boots, there are no objects present at all (except for the idle and system processes, whose objects are hardwired into the ntoskrnl.exe file). All objects are created on the fly as the system boots up and various initialization programs run. Some of the common executive object types are shown in Table 2 below:

Type	Description
Process	User process
Thread	Thread within a process
Semaphore	Counting semaphore used for interprocess synchronization
Mutex	Binary semaphore used to enter a critical region
Event	Synchronization object with persistent state (signaled/not)
Port	Mechanism for interprocess message passing
Timer	Object allowing a thread to sleep for a fixed time interval
Queue	Object used for completion notification on asynchronous I/O
Open file	Object associated with an open file
Access token	Security descriptor for some object
Profile	Data structure used for profiling CPU usage
Section	Structure used for mapping files onto virtual address space
Key	Registry key
Object directory	Directory for grouping objects within the object manager
Symbolic link	Pointer to another object by name
Device	I/O device object
Device driver	Each loaded device driver has its own object

Table 2: Some of the Common Executive Object Types Managed by Object Manager

As objects are created and deleted during execution, the object manager maintains a name space, in which all objects in the system are located. A process uses this name space to locate and open a handle for some other process' object, provided it has been granted permission to do so. Windows 2000 maintains three name spaces: (1) object name space, (2) file system name space, and (3) the registry name space. All three-name space follows a hierarchical model with multiple levels of directories for

organizing entries. The object name space is not visible to users without special viewing tools.

User-Mode Processes

Four basic types of user-mode processes are supported by Windows:

- **Special system support processes:** Include services not provided as part of the Windows operating system, such as the logon process and the session manager.
- **Service processes:** Other Windows services such as the event logger.
- **Environment subsystems:** Expose the native Windows services to user applications and thus provide an operating system environment or personality. The supported subsystems are Win32, Posix, and OS/2. Each environment subsystem includes dynamic link libraries (DLLs) that convert the user application calls to Windows calls.
- **User applications:** Can be one of five types: Win32, Posix, OS/2, Windows 3.1, or MSDOS.

Windows is structured to support applications written for Windows 2000 and later releases, Windows 98, and several other operating systems. Windows provides this support using a single, compact Executive through protected environment subsystems. The protected subsystems are those parts of Windows that interact with the end user. Each subsystem is a separate process, and the Executive protects its address space from that of other subsystems and applications. A protected subsystem provides a graphical or command-line user interface that defines the look and feel of the operating system for a user. In addition, each protected subsystem provides the API for that particular operating environment. This means that applications created for a particular operating environment may run unchanged on Windows, because the operating system

interface that they see is the same as that for which they were written. So, for example, OS/2- based applications can run under the Windows operating system without modification. Furthermore, because the Windows system is itself designed to be platform independent, through the use of the hardware abstraction layer (HAL), it should be relatively easy to port both the protected subsystems and the applications they support from one hardware platform to another.

In many cases, a recompile is all that should be required. The most important subsystem is Win32. Win32 is the API implemented on both Windows 2000 and later releases and Windows 98. Some of the features of Win32 are not available in Windows 98, but those features implemented on Windows 98 are identical with those of Windows 2000 and later releases.

PROCESS AND THREADS

Each process in Windows 2000 operating system contains its own independent virtual address space with both code and data, protected from other processes. Each process, in turn, contains one or more independently executing threads. A thread running within a process can create new threads, create new independent processes, and manage communication and synchronization between the objects.

By creating and managing processes, applications can have multiple, concurrent tasks processing files, performing computations, or communicating with other networked systems. It is even possible to exploit multiple processors to speed processing.

The following sections explain the basics of process management and also introduce the basic synchronization operations.

Windows Processes and Threads

Every process consists of one or more threads, and the Windows thread is the basic executable unit. Threads are scheduled on the basis of the following factors: (a) availability of resources such as CPUs and physical memory, (b) priority, (c) fairness, and so on. Windows has supported symmetric multiprocessing (SMP) since NT10, so threads can be allocated to separate processors within a system. Therefore, each Windows process includes resources such as the following components:

- One or more threads.
- A virtual address space that is distinct from other processes' address spaces, except where memory is explicitly shared. Note that shared memory-mapped files share physical memory, but the sharing processes will use different virtual addresses to access the mapped file.
- One or more code segments, including code in DLLs.
- One or more data segments containing global variables.
- Environment strings with environment variable information, such as the current search path.
- The process heap.
- Resources such as open handles and other heaps.

Each thread in a process shares code, global variables, environment strings, and resources. Each thread is independently scheduled, and a thread has the following elements:

- A stack for procedure calls, interrupts, exception handlers, and automatic storage.
- Thread Local Storage (TLS)—arrays of pointers giving each thread the ability to allocate storage to create its own unique data environment.

- An argument on the stack, from the creating thread, which is usually unique for each thread.
- A Context Structure maintained by the kernel, with machine registers values.

Windows 2000 has a number of concepts for managing the CPU and grouping resources together. In the following section we will study these concepts.

Windows 2000 supports traditional process that communicates and synchronizes with each other. Each process contains at least one thread, which contains at least one fiber (a lightweight thread). Processes combine to form a job for certain resource management purposes. And jobs, processes, threads, and fibers provides a very general set of tools for managing parallelism and resources.

- **Job**: collection of processes that share quotas and limits
- **Process:** container for holding resources
- **Thread**: Entity scheduled by the kernel
- **Fiber:** Lightweight thread managed entirely in user space.

A job in Windows 2000 is a collection of one or more processes. There are quotas (maximum number of processes, total CPU time, memory usage) and resource limits associated with each job, store in the corresponding job object.

Every process has a 10-GB address space, with the user occupying the bottom 2 GB (3 GB on advanced server and Datacenter Server) and operating system occupying the rest. Process is created using Win32 call and consist of a process ID, one or more threads, a list of handles, and an access token containing security related information. Each process starts out with one thread and new threads can be created dynamically using a win32 call. Operating system selects a thread to run for CPU scheduling. Every thread has a state like ready, running, blocked, etc. Every thread has a thread ID, which is taken from the same space as the process IDs, so an ID can never be in use for both a process and a thread at the same time. Processes and thread can be used as byte indices into kernel tables, as they are in multiple of four. A thread is normally run in user mode, but when it invokes a system call it switches to kernel mode. There are two stacks for each thread, one for use when it is in user mode and one for use when it is in kernel mode. A thread consists of the following

- Thread ID
- Two stacks
- A context (to save its register when not running)
- A private are for its own local variables and
- Possibly its own access token. If a thread has its own access token, this variable overrides the process access token in order to let client threads pass their access rights to server threads who are doing work for them.

Threads are a scheduling concept and not a resource ownership concept. In addition to normal threads that run in user processes, Windows 2000 provides a number of daemon threads (associated with the special system or idle processes) that runs in kernel space and are not linked or associated with any user process. Switching threads in Windows 2000 are relatively expensive, as they require entering and exiting kernel mode. Further, Windows 2000 provides fibers, which are similar to threads, but are scheduled in user space by the originating program (or its run time system). A thread can have multiple fibers. There are no executive objects relating to fibers, as there are for jobs, processes, and threads. In fact, the operating system knows nothing about the fibers and there are no true system calls for managing fibers. Windows 2000 is capable of running on a symmetric multiprocessor system. The relationship between jobs, processes, and threads is illustrated in Figure 4.

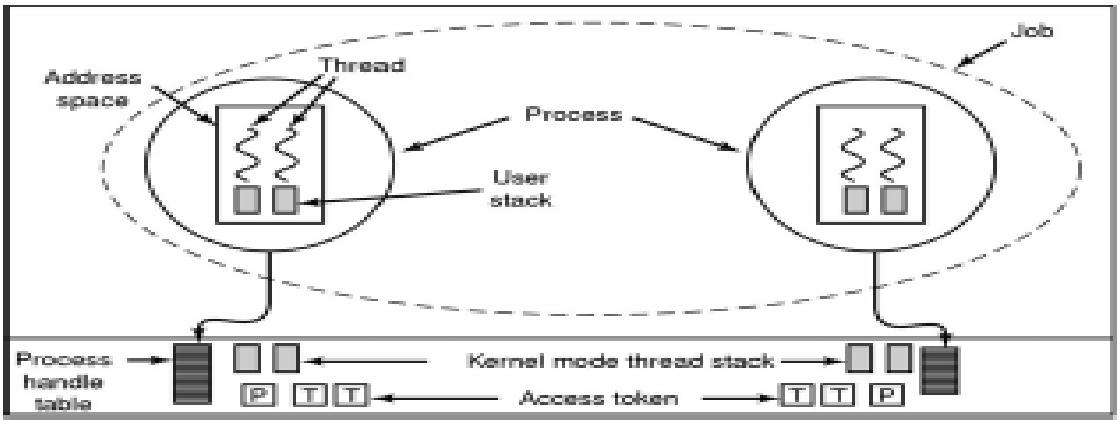

Figure 4: Relationship between jobs, processes, and threads

File System API Calls in Windows 2000

The principal Win32 API functions for file management are listed in table 3.

Win32 API function	UNIX	Description
CreateFile	open	Create a file or open an existing file; return a handle
DeleteFile	unlink	Destroy an existing file
CloseHandle	close	Close a file
ReadFile	read	Read data from a file
WriteFile	write	Write data to a file
SetFilePointer	lseek	Set the file pointer to a specific place in the file
GetFileAttributes	stat	Return the file properties
LockFile	fcntl	Lock a region of the file to provide mutual exclusion
UnlockFile	fcntl	Unlock a previously locked region of the file

Table 3: Win32 API functions for file management

There are actually many more, but these give a reasonable first impression of the basic ones. Let us now examine these calls briefly. CreateFile can be used to create a new file and return a handle to it. This API function must also be used to open existing files as there is no FileOpen API function. We have not listed the parameters for the API functions because they are so voluminous. As an example, CreateFile has seven parameters, which are roughly summarized as follows:

1. A pointer to the name of the file to create or open.
2. Flags telling whether the file can be read, written, or both.
3. Flags telling whether multiple processes can open the file at once.
4. A pointer to the security descriptor, telling who can access the file.
5. Flags telling what to do if the file exists/does not exist.
6. Flags dealing with attributes such as archiving, compression, error mode, priority, debugging, consoles etc.
7. The handle of a file whose attributes should be cloned for the new file.
8. A bit telling whether the new process inherits the creator's handles.
9. Pointer to the name of the new process current working directory.
10. Pointer for initial window on the screen.
11. Pointer to a structure that returns 18 values.

There is no concept of inheritance in Windows 2000 and it does not enforce any kind of parent-child or other hierarchy. All processes are created equal. But there is an implicit hierarchy in terms of who has a handle to whom. One of the eighteen parameters returns to the creating process is a handle to the new process, thus allowing control over the new process.

Initially each process is created with a one thread and process can create further threads, which is simpler than the process creation. The kernel performs the thread creation function and is not implemented purely in user space as is the case in some other operating systems. CreateThread has only six parameters:

1) Security descriptor (optional)
2) Initial stack size
3) Starting address
4) User defined parameter

5) Initial state of the thread (ready/blocked)

6) Thread ID.

Interprocess Communication

Threads can communicate in many ways including: pipes, named pipes, mailslots, sockets, remote procedure calls, and shared files. The various communications modes are shown in Table 4 below:

Thread communication	Mode
Pipes	byte and message, selected at creation time.
Name Pipes	Byte and message
Mailslots	A feature of Windows 2000 not present in UNIX. They are one-way, whereas pipes are two-way. They can also be used over a network but do not provide guaranteed delivery. They allow the sending process to broadcast a message to many receivers, instead of to just one receiver.
Sockets	Are like pipes, except that they normally connect processes on different machines. For example, one process writes to a socket and another one on a remote system reads from it.
Remote Procedures Calls	Are a mechanism for a process A to have process B call a procedure in B's address space on A's behalf and return the result to A.
Shared Files	Processes can share memory by mapping onto the same file at the same time

Table 4: Thread Communication Mode

Besides Windows 2000 providing numerous interprocess communication mechanism, it also provides numerous synchronization mechanisms, including semaphores, mutexes, critical regions, and events. All of these mechanisms functions on threads, not process.

Semaphore

A semaphore is created using function 'CreateSemaphore' API function. As Semaphores are kernel objects and thus have security descriptors and handles. The handle for a semaphore can be duplicated and as a result multiple process can be synchronised on

the same semaphore. Calls for up (ReleaseSemaphore) and down (WaitForSingleObject) are present. A calling thread can be released eventually using WaitForSingleObject, even if the semaphore remains at 0.

Mutexes

Mutexes are kernel objects used for synchronization and are simpler than semaphores as they do not have counters. They are locks, with API functions for locking (WaitForSingleObject) and unlocking (releaseMutex). Like Semaphores, mutex handles can be duplicated.

Critical Sections or Critical Regions

This is the third synchronization mechanism which is similar to mutexes. It is pertinent to note that Critical Section are not kernel objects, they do not have handles or security descriptors and cannot be passed between the processes. Locking and unlocking is done using EnterCriticalSection and LeaveCriticalSection, respectively. As these API functions are performed initially in user space and only make kernel calls when blocking is needed, they are faster than mutexes.

Events

This synchronization mechanism uses kernel objects called events, which are of two types: manual-reset events and auto-reset events.

The number of Win32 API calls dealing with processes, threads, and fibres is nearly 100. Windows 2000 knows nothing about fibres and fibres are implemented in user space. As a result, the CreateFibre call does its work entirely in user space without making 12-13 system calls.

Scheduling

There is no central scheduling thread in Windows 2000 operating system. But when a thread cannot run any more, the thread moves to kernel mode and runs the scheduler itself to see which thread to switch to and the concurrency control is reached through the following conditions

- Thread blocks on a semaphore, mutex, event, I/O, etc.: In this situation the thread is already running in kernel mode to carry out the operation on the dispatcher or I/O object. It cannot possibly continue, so it must save its own context, run the scheduler to pick its successor, and load that threat's context to start it.
- It signals an object – In this situation, thread is running in kernel mode and after signaling some object it can continue as signaling an object never blocks. Thread runs the scheduler to verify if the result of its action has created a higher priority thread. If so, a thread switch occurs because Windows 2000 is fully pre-emptive.
- The running thread's quantum expires: In this case, a trap to kernel mode occurs, thread executes the scheduler to see who runs next. The same thread

may be selected again and gets a new quantum and continue running, else a thread switch takes place.

The scheduler is also called under two other conditions:

1. An I/O operation completes: In this case, a thread may have been waiting on this I/O and is now released to run. A check is to be made to verify if it should pre-empt the running thread since there is no guaranteed minimum run time. The win32 API provides two hooks (SetPriorityClass, SetThreadPriority) for process to influence thread scheduling.

2 A timed wait expires.

For SetPriorityClass, the values are: realtime, high, above normal, normal, below normal, and idle. For SetThreadPriority, the values are: time critical, highest, above normal, normal, below normal, lowest, and idle. With six process classes and seven thread classes, a thread gets any one of 102 combinations, which is an input to the scheduling algorithm.

The scheduler has 32 priorities, numbered from 0 to 31. The 102 combinations are mapped onto the 32 priority classes.

BOOTING WINDOWS 2000

To start Windows 2000 system, it must be booted first. The boot process generates the initial processes that start the system. The process is as follows:

Pre-boot and Boot Sequences

On Intel-based systems, the boot process is made up of a pre-boot sequence and boot sequence. The pre-boot sequence consists of the following steps:

1. Power-On Self Tests (POST) are run.
2. The boot device is found, the Master Boot Record (MBR) is loaded into memory, and its program is run.
3. The active partition is located, and the boot sector is loaded.
4. The Windows 2000 loader (NTLDR) is then loaded.

The boot sequence executes the following steps:

1. The Windows 2000 loader switches the processor to the 32-bit flat memory model.
2. The Windows 2000 loader starts a mini-file system.

3. The Windows 2000 loader reads the BOOT.INI file and displays the operating system selections (boot loader menu).

4. The Windows 2000 loader loads the operating system selected by the user. If Windows 2000 is selected, NTLDR runs NTDETECT.COM. For other operating systems, NTLDR loads BOOTSECT.DOS and gives it control.

5. NTDETECT.COM scans the hardware installed in the computer, and reports the list to NTLDR for inclusion in the Registry under the HKEY_LOCAL_MACHINE_HARDWARE hive.

6. NTLDR then loads the NTOSKRNL.EXE, and gives it the hardware information collected by NTDETECT.COM. Windows NT enters the Windows load phases.

Process	Details
Idle System	Home to the idle thread
Smss.exe	Creates smss.exe & paging files; reads registry, open DLLs
Csrss.exe	First real process, creates csrss & winlogon
Winlogon.exe	Win32 process
Lsass.exe	Login daemon
Services.exe	Authentication manager
	Looks in registry and starts services
Printer server	Remote job printing
File server	Local files requests
Telnet server	Remote logins
Incoming mail handler	Accepts and stores inbound email
Incoming fax handler	Accepts and prints inbound faxes
DNS Resolver	Internet domain name system server
Event logger	Logs various system events
Plug-and-play manager	Monitors hardware

Table 5: The processes starting up during system booting.