* **I/O Manager:** allows devices to communicate with user-mode subsystems. It translates user-mode read and write commands in read or write IRPs which it passes to device drivers. It accepts file system I/O requests and translates them into device specific calls, and can incorporate low-level device drivers that directly manipulate hardware to either read input or write output. It also includes a cache manager to improve disk performance by caching read requests and write to the disk in the background
* **Security Reference Monitor (SRM):** the primary authority for enforcing the security rules of the security integral subsystem . It determines whether an object or resource can be accessed, via the use of access control lists (ACLs), which are themselves made up of access control entries (ACEs). ACEs contain a security identifier (SID) and a list of operations that the ACE gives a select group of trustees — a user account, group account, or logon session — permission (allow, deny, or audit) to that resource.
* **IPC Manager:** short for Interprocess Communication Manager, this manages the communication between clients (the environment subsystem) and servers (components of the Executive). It can use two facilities: the *Local Procedure Call* (LPC) facility (clients and servers on the one computer) and the *Remote Procedure Call* (RPC) facility (where clients and servers are situated on different computers. Microsoft has had significant security issues with the RPC facility .
* **Virtual Memory Manager:** manages virtual memory, allowing Windows 2000 to use the hard disk as a primary storage device (although strictly speaking it is secondary storage). It controls the paging of memory in and out of physical memory to disk storage.
* **Process Manager:** handles process and thread creation and termination
* **PnP Manager:** handles Plug and Play and supports device detection and installation at boot time. It also has the responsibility to stop and start devices on demand — sometimes this happens when a bus gains a new device and needs to have a device driver loaded to support that device. Both FireWire and USB are hot-swappable and require the services of the PnP Manager to load, stop and start devices. The PnP manager interfaces with the HAL, the rest of the executive (as necessary) and with device drivers.
* **Power Manager:** the power manager deals with power events and generates power IRPs. It coordinates these power events when several devices send a request to be turned off it determines the best way of doing this.
* The display system has been moved from user mode into the kernel mode as a device driver contained in the file *Win32k.sys*. There are two components in this device driver — the Window Manager and the GDI:
  + **Window Manager:** responsible for drawing windows and menus. It controls the way that output is painted to the screen and handles input events (such as from the keyboard and mouse), then passes messages to the applications that need to receive this input
  + **GDI:** the Graphics Device Interface is responsible for tasks such as drawing lines and curves, rendering fonts and handling palettes. Windows 2000 introduced native alpha blending into the GDI.

**Hardware abstraction layer**

The Windows 2000 Hardware Abstraction Layer, or HAL, is a layer between the physical hardware of the computer and the rest of the operating system. It was designed to hide differences in hardware and therefore provide a consistent platform on which applications may run. The HAL includes hardware specific code that controls I/O interfaces, interrupt controllers and multiple processors.

### Kernel & kernel-mode drivers

The kernel sits between the HAL and the Executive and provides multiprocessor synchronization, thread and interrupt scheduling and dispatching, and trap handling and exception dispatching. The kernel often interfaces with the process manager. The kernel is also responsible for initialising device drivers at bootup that are necessary to get the operating system up and running.

# User mode and kernel mode

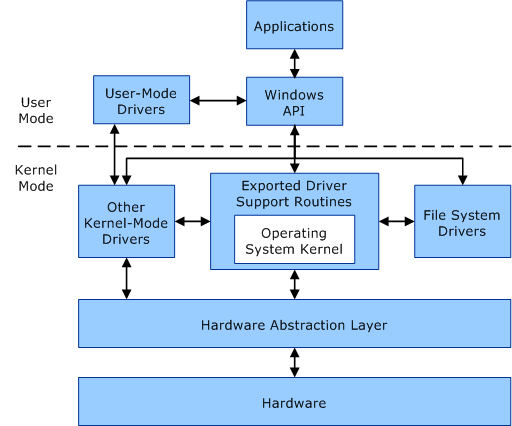
A processor in a computer running Windows has two different modes: *user mode* and *kernel mode*. The processor switches between the two modes depending on what type of code is running on the processor. Applications run in user mode, and core operating system components run in kernel mode. While many drivers run in kernel mode, some drivers may run in user mode.

When you start a user-mode application, Windows creates a *process* for the application. The process provides the application with a private *virtual address space* and a private *handle table*. Because an application's virtual address space is private, one application cannot alter data that belongs to another application. Each application runs in isolation, and if an application crashes, the crash is limited to that one application. Other applications and the operating system are not affected by the crash.

In addition to being private, the virtual address space of a user-mode application is limited. A processor running in user mode cannot access virtual addresses that are reserved for the operating system. Limiting the virtual address space of a user-mode application prevents the application from altering, and possibly damaging, critical operating system data.

All code that runs in kernel mode shares a single virtual address space. This means that a kernel-mode driver is not isolated from other drivers and the operating system itself. If a kernel-mode driver accidentally writes to the wrong virtual address, data that belongs to the operating system or another driver could be compromised. If a kernel-mode driver crashes, the entire operating system crashes.

This diagram illustrates communication between user-mode and kernel-mode components.



# Kernel

A kernel is the core component of an operating system. Using interprocess communication and system calls, it acts as a bridge between applications and the data processing performed at the hardware level.  
  
When an operating system is loaded into memory, the kernel loads first and remains in memory until the operating system is shut down again. The kernel is responsible for low-level tasks such as disk management, task management and memory management.

A computer kernel interfaces between the three major computer hardware components, providing services between the application/user interface and the CPU, memory and other hardware I/O devices.  
  
The kernel provides and manages computer resources, allowing other programs to run and use these resources. The kernel also sets up memory address space for applications, loads files with application code into memory, sets up the execution stack for programs and branches out to particular locations inside programs for execution.   
  
**The kernel is responsible for:**

* Process management for application execution
* Memory management, allocation and I/O
* Device management through the use of device drivers
* System call control, which is essential for the execution of kernel services

There are five types of kernels:

1. **Monolithic Kernels**: All operating system services run along the main kernel thread in a monolithic kernel, which also resides in the same memory area, thereby providing powerful and rich hardware access.
2. **Microkernels:** Define a simple abstraction over hardware that use primitives or system calls to implement minimum OS services such as multitasking, memory management and interprocess communication.
3. **Hybrid Kernels:** Run a few services in the kernel space to reduce the performance overhead of traditional microkernels where the kernel code is still run as a server in the user space.
4. **Nano Kernels:** Simplify the memory requirement by delegating services, including the basic ones like interrupt controllers or timers to device drivers.
5. **Exo Kernels:** Allocate physical hardware resources such as processor time and disk block to other programs, which can link to library operating systems that use the kernel to simulate operating system abstractions.

# Subsystems

The subsystem is where work is processed on the system. A subsystem is a single, predefined operating environment through which the system coordinates the work flow and resource use. The system can contain several subsystems, all operating independently of each other. Subsystems manage resources.

# Hardware Abstraction Layer (HAL)

A hardware abstraction layer (HAL) is a logical division of code that serves as an abstraction layer between a computer's physical hardware and its software. It provides a device driver interface allowing a program to communicate with the hardware.  
  
The main purpose of a HAL is to conceal different hardware architectures from the OS by providing a uniform interface to the system peripherals.

**Explanation:**

A hardware abstraction layer is included in many OSs to avoid modifying the OS kernel to run the program on computers with varying hardware architecture. A PC may include the HAL in the OS kernel or in the form of device drivers that provide a consistent interface for applications to interact with the hardware peripherals.   
  
The HAL provides the following benefits:

* Allowing applications to extract as much performance out of the hardware devices as possible
* Enabling the OS to perform regardless of the hardware architecture
* Enabling device drivers to provide direct access to each hardware device, which allows programs to be device-independent
* Allowing software programs to communicate with the hardware devices at a general level
* Facilitating portability

Some of the OSs that feature HALs include the Mac OS, Linux, DOS, Solaris, BSD, Windows NT, Windows 2000 and IBM’s AS/400.

**DLL**

* A Dll is a Dynamic Link Library can not run itself, used as a supportive file to other application.
* The Library Functions are Linked to the Application at Run Time (Dynamically) So the name is Dll.
* A Dll does not contain an entry point (main function) so can not run individually.

**EXE**

* An Exe is executable file and is not a supportive file rather itself an application.
* An Exe will contain an entry point (main function) so runs individually.

|  |  |  |
| --- | --- | --- |
| **SNO** | **DLL** | **EXE** |
| 1 | Can not run Individually | Runs Individually |
| 2 | Used as supportive file for other Application | Itself an Application |
| 3 | Does not contain an entry point (no main function) so can not run individually | Contains an entry point (Main function) so can run individually |
| 4 | A Program /Application with out main creates a DLL after compilation. | A Program /Application With main creates an EXE after compilation. |
| 5 | OS does not create a separate process for any DLL rather DLL will run in the same process created for an EXE | OS Creates a separate process for each EXE it executes. |

1.EXE is an extension used for executable files while DLL is the extension for a dynamic link library.   
2.An EXE file can be run independently while a DLL is used by other applications.   
3.A DLL file can be reused by other applications while an EXE cannot.   
4.Exe is for single use whereas you can use Dll for multiple use   
5.Dll is an In-Process Component whereas EXE is an OUt-Process Component.   
6.You can create an objects of Dll but not of the EXE.