OS Service	Hardware Support
Protection	Kernel/user mode, base/limit registers, Protected instructions
Interrupts	Interrupt vectors
System calls	Trap instructions and Trap vectors
I/O devices	Interrupts and memory handling
Scheduling ,error recovery	Timer
Synchronization	Atomic instructions
Virtual Memory	Translational look-aside buffers

Architecture and Its Support for OS.

Protection

Kernel mode vs User Mode: To protect the system from aberrant users and processors, some instructions are restricted to use only by the OS. Users may not

-Address I/O directly

-use instructions that manipulate the state of memory(page table pointers, TLB load etc.)

- set the mode bits that determine user/kernel mode

-disable/enable interrupts

-halt the machine

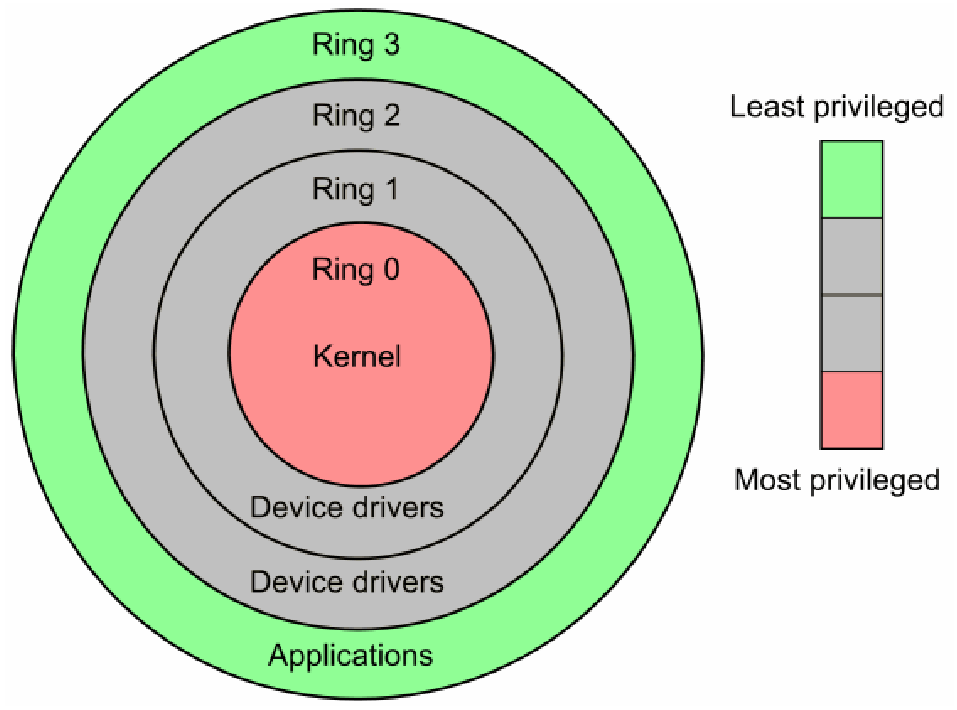
But in kernel mode, the OS can do all these things

Thus the hardware must support at least two modes, the User and the Kernel mode.

-A status bit in a protected processor register determine the mode.

Protected/Privileged instructions can be only executed in Kernel mode

If code running in user mode attempts to execute a privileged instruction the Illegal excecutin trap



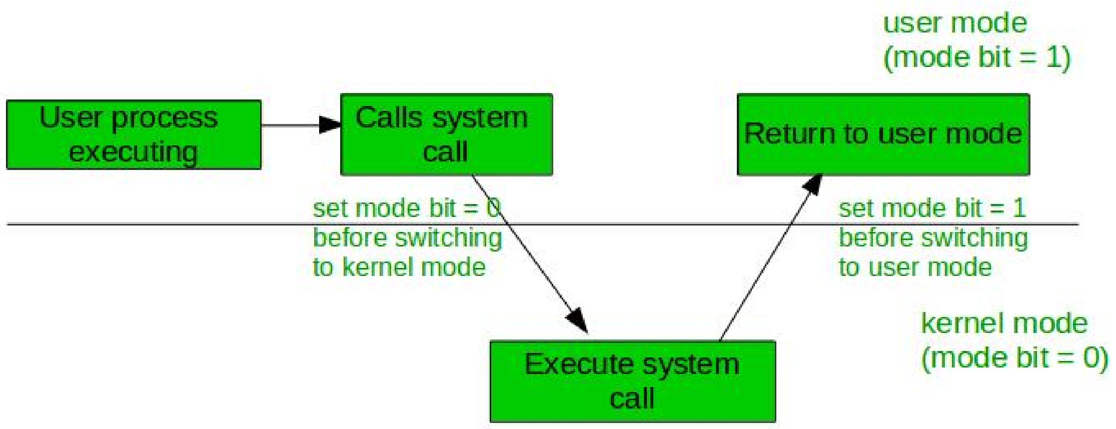
X86 supports 4 protection modes

Crossing Protection Boundaries

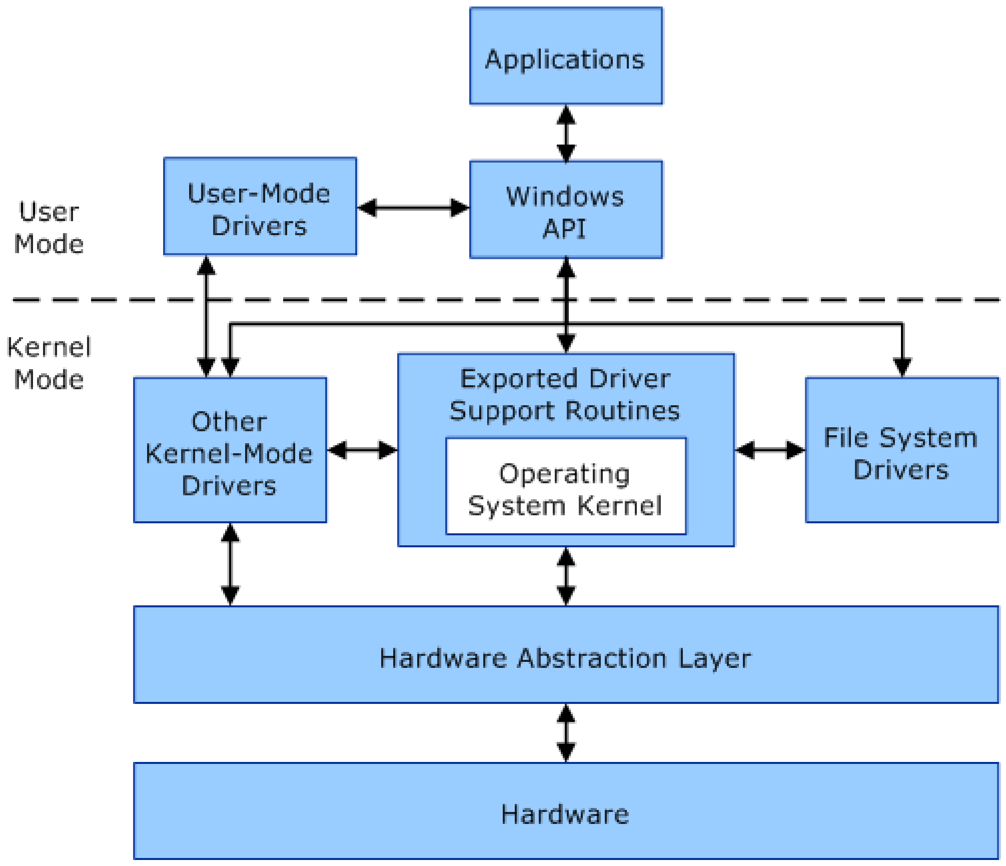
System Call: The programs that we the users can't execute, we ask the OS kernel to do on our behalf. So this is what we call the System call. So System call is a OS procedure that manages privileged instructions(e.x- I/O).- Also API exported by the kernel .

- Causes a trap, which vectors (jumps) to the trap handler in the OS kernel.

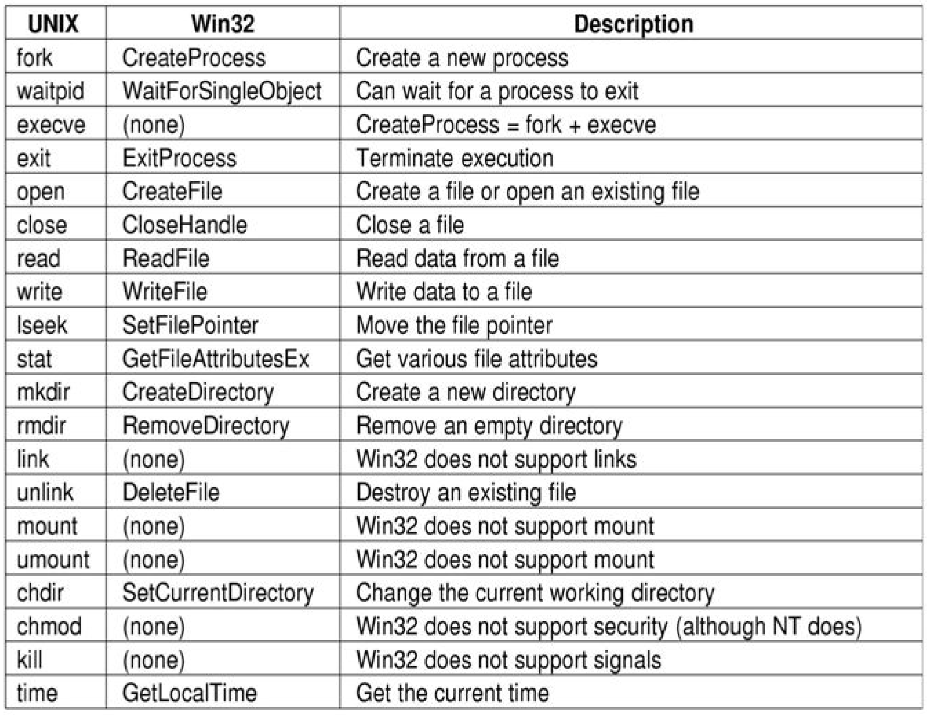
- The trap handler uses the parameter in the system call to jump to the appropriate handler (I/O, terminal etc.)
- The handler saves caller state (PC, mode bit) so it can restore the user processes.
- The architecture must permit the OS to verify the caller's parameters.



- The architecture must also provide a way to return to the user mode when finished.


Applications
User Mode
User-Mode Drivers
Windows API
Kernel Mode
Other Kernel-Mode Drivers
Exported Driver Support Routines
Operating System Kernel
File System Drivers
Hardware Abstraction Layer
Hardware

Windows System Calls



UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

Some Win32 API calls

Memory Protection

Architecture must provide support so that th OS can

- Protect the user programs from each other , and
- Protect the OS from user programs(may be malicious)

Memory protection is a way to control memory access rights on a computer, and is a part of most modern instruction set architectures and operating systems. The main purpose of memory protection is to prevent a process from accessing memory that has not been allocated to it. This prevents a bug or malware within a process from affecting other processes, or the operating system itself. Protection may encompass all accesses to a specified area of memory, write accesses, or attempts to execute the contents of the area.

Memory protection prevents one process from affecting the confidentiality, integrity, or availability of another. This is a requirement for secure multiuser (more than one user logged in simultaneously) and multitasking (more than one process running simultaneously) systems.

- The simplest technique to get the memory protection is to Use Base(a register in the hardware) and Limit Registers.
- Base and Limit registers are loaded by the OS before starting a program.
- The CPU checks each user reference(instructions and data addresses), ensuring it falls between the base and limit registers .

Process Isolation

Process isolation is a logical control that attempts to prevent one process from interfering with another. This is a common feature among multiuser operating systems such as Linux, UNIX, or recent Microsoft Windows operating systems. Older operating systems such as MS-DOS provide no process isolation. A lack of process isolation means a crash in any MS-DOS application could crash the entire system.

If you are shopping online and enter your credit card number to buy a book, that number will exist in plaintext in memory (for at least a short period of time). Process isolation means that another user's process on the same computer cannot interfere with yours.

Interference includes attacks on the confidentiality (reading your credit card number), integrity (changing your credit card number), and availability (interfering or stopping the purchase of the book).

Techniques used to provide process isolation include **virtual memory, object *encapsulation* and time multiplexing**. Object encapsulation treats a process as a “black box” . Time multiplexing shares (multiplexes) system resources between multiple processes, each with a dedicated slice of time.

Hardware Segmentation

Hardware segmentation takes process isolation one step further by mapping processes to specific memory locations. This provides more security than (logical) process isolation alone.

Simulated segmentation

Simulation is the use of a monitoring program to interpret the machine code instructions of some computer architectures. Such an instruction set simulator can provide memory protection by using a segmentation-like scheme and validating the target address and length of each instruction in real time before actually executing them. The simulator must calculate the target address and length and compare this against a list of valid address ranges that it holds concerning the thread’s environment, such as any dynamic memory blocks acquired since the thread’s inception, plus any valid shared static memory slots. The meaning of "valid" may change throughout the thread’s life depending upon context. It may sometimes be allowed to alter a static block of storage, and sometimes not, depending upon the current mode of execution, which may or may not depend on a storage key or supervisor state.

It is generally not advisable to use this method of memory protection where adequate facilities exist on a CPU, as this takes valuable processing power from the computer. However, it is generally used for debugging and testing purposes to provide an extra fine level of granularity to otherwise generic storage violations and can indicate precisely which instruction is attempting to overwrite the particular section of storage which may have the same storage key as unprotected storage.

Consider the Intel architecture as an example of segmentation. It uses the code segment (CS) register as a segment selector and stores the current privilege level (CPL) as its two lower bits. When the executing code tries to access a data segment, the descriptor privilege level (DPL) is checked. Depending on whether the loaded segment is data, code, or a system call, the check ensures the CPL allows loading the segment descriptor based on the DPL. For example, a data segment DPL specifies the highest privilege level (CPL) that a task can have in order to be allowed access, so if DPL is 2 then access is granted only to tasks with CPL of 0, 1, or 2. A third privilege level, the requested privilege level (RPL), is used when invoking OS services through call gates and prevents less privileged applications from elevating privileges and gaining access to restricted system segments.

Most open or commercial OSs ignore or make limited use of segmentation. OS/2 is a modern commercial OS that uses the full features of segmentation. Some virtualization techniques (such as the VMWare ESX hypervisor) do reserve a segment for a resident memory area and rely on segment limit checks to catch illegal accesses.

Virtual Memory

Virtual memor*y* provides virtual address mapping between applications and hardware memory. Virtual memory provides many functions, including multitasking (multiple tasks executing at once on one CPU), allowing multiple processes to access the same shared library in memory, swapping, and others.

Swapping and Paging

Swapping uses virtual memory to copy contents in primary memory (RAM) to or from secondary memory (not directly addressable by the CPU, on disk). Swap space is often a dedicated disk partition that is used to extend the amount of available memory. If the kernel attempts to access a page (a fixed-length block of memory) stored in swap space, a page fault occurs (an error that means the page is not located in RAM), and the page is "swapped" from disk to RAM.

Most computers configured with virtual memory, as the system in Figure 4.10, will use only RAM until the RAM is nearly or fully filled. The system will then swap processes to virtual memory. It will attempt to find idle processes so that the impact of swapping will be minimal.

Eventually, as additional processes are started and memory continues to fill, both RAM and swap will fill. After the system runs out of idle processes to swap, it may be forced to swap active processes. The system may begin "thrashing," spending large amounts of time copying data to and from swap space, seriously impacting availability.

Swap is designed as a protective measure to handle occasional bursts of memory usage. Systems should not routinely use large amounts of swap: in that case, physical memory should be added, or processes should be removed, moved to another system, or shortened.

Protecting Memory from Malicious Applications

Windows Vista includes Data Execution Prevention (DEP), a memory protection application that monitors the use of memory by applications. By default, it does not allow applications to use those memory locations reserved for the system. This essentially helps protect the system from malicious code that may insert itself into a system memory location. In other words, DEP is a security feature that allows only system applications to use system memory locations. If DEP detects that any application is inappropriately using system memory locations, it shuts down the application and displays a message.

By default, DEP is enabled only for essential Windows services and applications. You can configure DEP for all services and programs as well as specify exceptions. Vista provides this feature as a software-based solution for protecting system memory. Some hardware vendors provide DEP as a hardware-based solution. Even if your computer hardware does not support DER a malicious application such as a virus program can be prevented by Windows Vista DEP from undesired use of system memory. This helps in keeping the system stable by preventing inappropriate memory usage by malicious software