Architecture and Its Support for OS.

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

Events

- **An event is an unnatural change in control flow**
  - **Events immediately stop current execution**
  - **Changes mode, context (machine state), or both**
- **The kernel defines a handler for each event type**
  - **Event handlers always execute in kernel mode**
  - **The specific types of events are defined by the machine**

- **Once the system is booted, all entry to the kernel occurs as the result of an event**
  - **In effect, the operating system is one big event handler**

Categorizing Events

There are two kinds of events –

- **Interrupts**
- **Exceptions**

- **Exceptions are caused by executing instructions**

- CPU requires software intervention to handle a fault or trap

- **Interrupts are caused by an external event**

-

Device finishes I/O, timer expires, etc.

There are two reasons for events -

- **Unexpected : unexpected events are well, and unexpected**
- **Deliberate : Deliberate events are scheduled by OS or application**

All these description is shown in the given table:

	Unexpected	Deliberate
Exceptions (sync)	Fault	System call trap
Interrupts (async)	interrupt	Software interrupt

Faults

- Hardware detects and reports exceptional conditions such as page fault, write to a read-only page, overflow, trace trap, odd address trap, privileged instruction trap, system call etc.
- It must transfer control to handler within the OS.
- Hardware must save state on fault (PC, etc.) so that the faulting process can be restarted afterwards.
- Modern operating systems use VM faults for many functions such as Debugging, distributed VM, Garbage collection, copy-on-write etc.
- Fault exceptions are a performance optimization, i.e., faults could be detected by inserting extra instructions into the code (but it requires high cost)

Handling Faults

- Some faults are handled by fixing the exceptional condition and returning to the faulting context. For example-
  - Page faults cause the OS to place the missing page into memory
  - Fault handler resets PC of faulting context to re-execute instruction that caused the page fault
- Some faults are handled by notifying the process
  - Fault handler changes the saved context to transfer control to a user-mode handler on return from fault
  - Handler must be registered with OS

- **The kernel may handle unrecoverable faults by killing the user process**
  - **Program fault with no registered handler**
  - **Halt process, write process state to file, destroy process**

Traps

- **Traps: special conditions detected by the architecture**

  - Examples: page fault, write to read-only page, overflow, system call etc.

- **On detecting a trap, the hardware**
  - **Saves the state of the process (PC, Stack, etc.)**
  - **Transfers control to appropriate trap handler (OS routine)**
    - **The CPU indexes the memory-mapped trap vector with the trap number,**
    - **then jumps to the address given in the vector, and**
    - **starts to execute at that address.**
    - **On completion, the OS resumes execution of the process**

  Trap Vector:

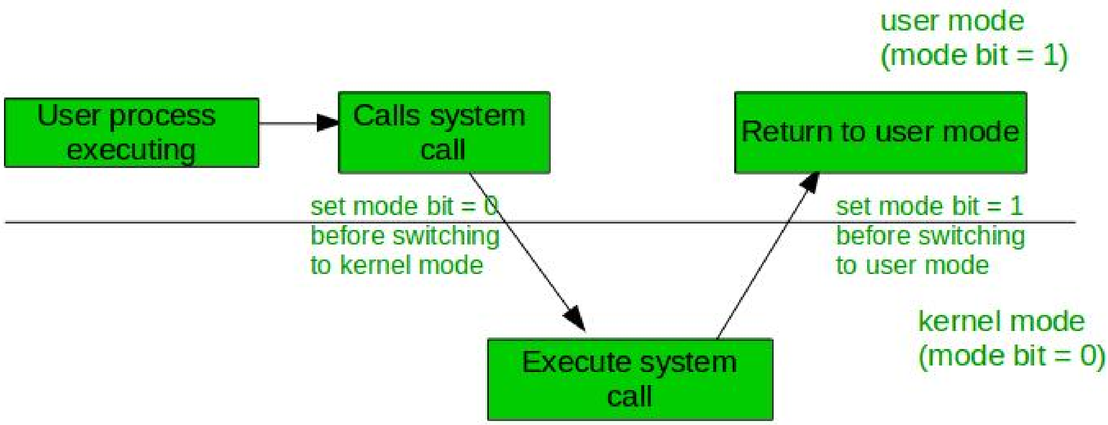
0: 0x00080000	Illegal address
1: 0x00100000	Memory violation
2: 0x00100480	Illegal instruction
3: 0x00123010	System call

- Modern OS use Virtual Memory traps for many functions: debugging, disturbed VM, garbage collection, copy-on-write, etc.
- Traps are a performance optimization. A less efficient solution is to insert extra instruction into the code everywhere a special condition could arise.

System Call

- For a user program to do something "privileged" (e.g., I/O) it must call an OS procedure
- Known as *crossing the protection boundary*, or a *protected procedure call*

- Architecture provides a system call instruction that causes a trap, which vectors (jumps) to the trap handler in the OS kernel.
- The trap handler uses the parameter to the system call to jump to the appropriate handler (I/O, Terminal, etc.).
- The handler saves caller's state (PC, mode bit) so it can restore control to the user process.
- The architecture must permit the OS to verify the caller's parameters.
- The architecture must also provide a way to return to user mode when finished.



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

36

Interrupts

- There are two types of interrupts:
  - Precise interrupts – CPU transfers control only on instruction boundaries
  - Imprecise interrupts – CPU transfers control in the middle of instruction execution
- OS designers like precise interrupts, and CPU designers like imprecise interrupts
- Interrupts signal asynchronous events
- Timer, I/O, etc.

Interrupt based asynchronous I/O

  - Device controller has its own small processor which executes asynchronous with the main CPU.
  - Device puts an interrupt signal on the bus when it is finished.
  - CPU takes an interrupt.
- Save critical CPU state (hardware state),
- Disable interrupts,
- Save state that interrupt handler will modify (software state),
- Invoke using the

  in-memory Interrupt Vector

- Restore software state
- Enable interrupts
- Restore hardware state, and continue execution of interrupted process.

Timer and atomic instruction

Timer

- Time of Day
- Accounting and billing
- CPU protected from being hogged using timer interrupts that occur at say every 100 microsecond.
- At each timer interrupt, the CPU chooses a new process to execute.

Interrupt Vector:

0: 0x2ff080000	Keyboard
1: 0x2ff100000	mouse
2: 0x2ff100480	timer
3: 0x2ff123010	Disk 1