OS Interface: Interrupts and System Calls
How to invoke the services of the OS

I. Instruction Execution

Basic Instruction Cycle

Program Execution (Text)

1. FETCH: The PC contains 300, the address of the first instruction. This instruction is loaded into the IR.
   Comment: The FETCH: The PC contains 300, the address first four bits in the IR ("1") give the opcode for "LOAD." The remaining twelve bits ("940") specify the memory address where the data is to be loaded from.

2. EXECUTE: The data is loaded into the accumulator. AC now has the value "003"

3. FETCH: The PC is incremented to 301 and the next instruction is fetched.
   Comment: The first four bits ("9") give the opcode for "ADD". The remaining twelve bits ("941") give the address of the data to be added.

Program Execution (Text)

4. EXECUTE: The contents of the AC and the contents of location 941 are added and stored back into the AC. AC now has the value "006"

5. FETCH: The PC is incremented to 302 and the next instruction is fetched.
   Comment: The first four bits ("2") give the opcode for "STORE." The remaining twelve bits ("941") give the address of where to store the result.

6. EXECUTE: The contents of the AC are stored at address 941.
II. Interrupts

Definition: Interrupt is an event that causes the normal fetch-execute cycle to be interrupted so that the event can be taken care of by the OS.

Types of Interrupts:
- I/O
- Timer
- Program
  - divide by zero
  - overflow or underflow
  - memory protection violation
  - illegal instruction
  - Parity error
  - Hardware failure

Why Interrupts?

- Why it's desirable to have interrupts
  - Without interrupts process 4 has to keep checking to see if its I/O is done (wasted CPU cycles).
  - With interrupts, Process 2 can run while the I/O is being done. I/O interrupt will let the CPU know when 4's I/O is completed.
  - Process 2 gets interrupted and the I/O interrupt handler takes over. It then passes control to the scheduler who chooses Process 5 to run next.

Program Timing

Interrupt Handler

- Interrupt Handler - OS code to take care of each type of interrupt.

Instructions w/Interrupts

III. Clocks and Timer Interrupts

- PURPOSE OF CLOCKS
  - To generate a timer interrupt to control the amount of time a processes has the CPU.
  - Keep time of day
  - Accounting, monitoring
  - Alarm clock processes, watch dog processes
Interrupts

### Countdown Interrupt Clock
- Works like a kitchen timer
- Capable of being programmed to cause an interrupt at desired intervals.
- OS sets clock for a certain number of ticks. Clock interrupts the CPU after that many ticks.

Crystal oscillator generates clock pulse (ticks).

Counter is decremented on each clock pulse.

OS clock routine loads counter with desired countdown interval (in ticks).

### IV. OS System (Kernel) Calls
- System call is entry to the OS kernel code from a user or system running program.
- It is a request for OS services
- Common types
  - Process control (create/fork; terminate/kill, wait, signal)
  - File management (open, close, read, write)
  - I/O management (read, write, seek)
  - Memory management (allocate, deallocate)
  - Information management (get time, get attributes)
  - Communications (create, delete, send, receive)

### Sample System Calls (Windows and Unix)

### OS System Calls
- Programmer can invoke system calls (kernel calls) directly in his/her code so that OS is called from within an executing program.
- Programmer can indirectly invoke system call while sitting at the keyboard through a command interpreter (shell) program that is always running on the system.
- User mode vs. kernel mode
- System calls vs. library routines
  - System call is access to the OS kernel and CPU starts running kernel code on behalf of the user that called it.
  - Library routine is pre-compiled code that is combined with user’s program code and runs as part of his/her program.

### Summary
Interrupts

Summary

• Hardware runs programs via the Fetch-Execute cycle.

• A hardware-generated interrupt breaks the Fetch-Execute cycle and transfers control to the OS to take appropriate action.

• An executing program can voluntarily call the OS to take action on its behalf through system calls.