OS and Architectures

- What an OS can do is dictated, at least in part, by the architecture.
- Architecture support can greatly simplify (or complicate) OS tasks.
- Example: PC operating systems have been primitive, in part because PCs lacked hardware support (e.g., for VM).

Computer-System Operation

- I/O devices and the CPU can execute concurrently.
- Each device controller is in charge of a particular device type.
- Each device controller has a local buffer.
- CPU moves data from/to main memory to/from local buffers.
- I/O id to/from the device to local buffer of controller.
Problem

• You work at Wintel Corp. as an OS designer. The architects unveil their latest chip design, the Septium. They have reengineered the entire instruction set. The Septium runs at 5 GHz, and costs $5.
  – They have only tested it with a matrix multiply program. The results are impressive.
  – The new Septium instruction set only supports arithmetic, jumps/branches, loads/stores.
  – The Septium has no interrupts, traps or exceptions, supports only physical addressing and uses only programmed I/O.
  – The technical writers are really happy as well, because the design specification fits on a single page.
• Your task is to come up with a list of features they will need to add to the chip design to support a modern PC operating system. Note that a modern PC OS will at least guarantee the integrity of users’ data in the face of multiple, potentially malicious users and concurrent applications.

Architectural Features for OS

• Features that directly support OS needs include:
  – 1. Protected instructions
  – 2. System calls
  – 3. Synchronization (atomic instructions)
  – 4. Memory protection
  – 5. I/O control and operation
  – 6. Interrupts and exceptions
  – 7. Timer (clock) operation

Protected Instructions

• Some instructions need to be restricted to the O.S.
  – Users cannot be allowed direct access to some hardware resources
  – Direct access to I/O devices like disks, printers, etc.
  – Must control instructions that manipulate memory management state (page table pointers, TLB load, etc.)
  – Setting of special mode bits
  – Halt instruction

OS Protection

• How do we restrict users from issuing such instructions?
  – Hardware supports a scheme that allows us to tell apart the trusted programmer (operating system designer) from untrusted programmers (regular users).
  – Most architectures support at least two modes of operation: kernel mode and user mode
  – The OS executes in kernel mode, user programs execute in user mode
  – Mode is indicated by a status bit in a protected processor register
• Protected instructions can only be executed in kernel mode.
**Crossing Protection Boundaries**

- For a user to do something “privileged” (e.g., I/O) it must call an OS procedure.
- How does a user-mode program call a kernel-mode service?
- There must be a system call instruction that switches from user to kernel mode.
- The system call instruction usually does the following:
  - causes an exception, which vectors to a kernel handler
  - passes a parameter, saying which system routine to call
  - saves caller’s state (PC, SP, other registers, etc.) so it can be restored
  - arch must permit OS to verify caller’s parameters
  - must provide a way to return to user mode when done

**Protection Modes and Crossing**

![Diagram of protection modes and crossing](image)

**Partitioning Functionality**

- Problem: The user-kernel mode distinction poses a performance barrier
  - Crossing this hardware barrier is costly. System calls take 10x to 1000x more time than a procedure call
- Solution: Perform some system functionality in user mode
  - Libraries (DLLs) can reduce the number of system calls, e.g. by caching results (getpid) or buffering operations (open/read/write vs. fopen, fread, fwrite).

**Memory Protection**

- Need to protect a user program from accessing the data in other user programs
- Need to protect the OS from user programs
- Simplest scheme is base and limit registers:
  - Virtual memory and segmentation are similar
Traps and Exceptions

- Traps and exceptions are initiated by the application
- Hardware must detect special conditions: page fault, write to a read-only page, overflow, trace trap, odd address trap, privileged instruction trap...
- Must transfer control to handler within the O.S.
- Hardware must save state on fault (PC, etc) so that the faulting process can be restarted afterwards
- Modern operating systems use VM traps for many functions: debugging, distributed VM, garbage collection, copy-on-write...
- Exceptions are a performance optimization, i.e., Conditions could be detected by inserting extra instructions in the code (at high cost)

Interrupts

- Interrupts are device-initiated
- Interrupts transfer control to the interrupt service routine, through the interrupt vector, which contains the addresses of all the service routines.
- Interrupt architecture must save the machine context at the interrupted instruction.
- Incoming interrupts are often disabled while another interrupt is being processed to prevent a lost interrupt.
- Most operating systems are interrupt driven.

I/O Structure

- I/O issues:
  - how to start an I/O (special instructions or memory-mapped I/O)
  - I/O completion (interrupts)
- Synchronous I/O: After I/O starts, control returns to user program only upon I/O completion.
  - wait instruction idles the CPU until the next interrupt
  - wait loop (contention for memory access).
  - At most one I/O request is outstanding at a time, no simultaneous I/O processing.
- Asynchronous I/O: After I/O starts, control returns to user program without waiting for I/O completion.
  - System call—request to the operating system to allow user to wait for I/O completion.
  - Device status table contains entry for each I/O device indicating its type, address, and state.
  - Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt.

Two I/O methods

- Synchronous
  - User
    - Requesting process
    - Polling
  - Device driver
  - Interrupt handler
  - Hardware
  - Data transfer

- Asynchronous
  - User
    - Requesting process
    - User
  - Device driver
  - Interrupt handler
  - Hardware
  - Data transfer

  - Time (a)
  - Time (b)
**I/O Control (cont)**

- Device interrupts
  - CPU stops current operation, switches to kernel mode, and saves current PC and other state on kernel stack
  - CPU fetches proper vector from vector table and branches to that address (the routine to handle interrupt)
  - Interrupt routine examines device database and performs action required by interrupt
  - Handler completes operation, restores saved (interrupted state) and returns to user mode (or calls scheduler to switch to another program)

---

**Direct Memory Access (DMA)**

- Used for high-speed I/O devices able to transmit information at close to memory speeds.
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.
- Only one interrupt is generated per block, rather than one interrupt per byte.

---

**Timer Operation**

- How does the OS prevent against runaway user programs (infinite loops)?
  - A timer can be set to generate an interrupt in a given time.
  - Before it transfers to a user program, the OS loads the timer with a time to interrupt.
  - When the time arrives, the executing program is interrupted and the OS regains control.
  - This ensures that the OS can get the CPU back even if a user program erroneously or purposely continues to execute past some allotted time.
  - The timer is privileged: only the OS can load it.

---

**Synchronization**

- Interrupts cause potential problems because an interrupt can occur at any time -- causing code to execute that interferes with code that was interrupted
- OS must be able to synchronize concurrent processes
- This requires guaranteeing that certain instruction sequences (read-modify-write) execute atomically
- One way to guarantee this is to turn off interrupts before the sequence, execute it, and re-enable interrupts; CPU must have a way to disable interrupts
  - When would this not be sufficient
- Another is to have special instructions that can perform a read/modify/write in a single bus transaction, or can atomically test and conditionally set a bit, based on its previous value