Lecture Topics: 10/29

- Architectural support for operating systems
  - timers
  - kernel mode
  - system calls
  - protected instructions

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)

Architectural Features for OS

- **Features that directly support OS needs include:**
  1. timer (clock) operation
  2. sychronization (atomic instructions)
  3. memory protection
  4. I/O control and operation
  5. interrupts and exceptions
  6. OS protection (kernel/user mode)
  7. protected instructions
  8. system calls

Protected Instructions

- Some instructions are typically restricted to the O.S.
  - users cannot be allowed direct access to I/O (disks, printers, etc)
  - setting of special mode bits (kernel mode)
  - halt instruction
  - must control instructions that manipulate memory management state (page table pointers, TLB load, etc)

OS Protection

- How do we know if we can execute a protected instruction?
  - arch must support (at least) two modes of operation: *kernel* mode and *user* mode
  - mode is indicated by a status bit in a protected processor register
  - user programs execute in user mode; the OS executes in kernel mode
- 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:
  - causes an exception, which vectors to a kernel handler
  - passes a parameter, saying which system routine to call
  - saves caller's state (PC, mode bit) 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

OS Services

- Make a new process
- Identify a process
- Kill a process
- Open/close a file
- Read/write the file
- Write to the console device
- Send/receive a network message

Making a System Call

User Program

1. Make system call

user mode

---

2. Well known entry point

kernel mode

3. Trap handler

4. System service routine

OS Kernel

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.