DEVICE-CPU COMMUNICATION

Instructor: Prasun Dewan (FB 150, dewan@unc.edu)
Centralized broadcast communication architecture instead of peer to peer.

Components?

Address bus
- Lines carrying addresses accessed by CPU

Data bus:
- Data in main memory or device registers referenced by address.

Control bus:
- Posting of interrupts.
- Whether information is being read or written by bus.
- Size of data
- ....
Device Registers

Content
- Input and output data, overwritten with each new I/O
- Ready Status, reset when input read or output displayed
- Error Indication (e.g. register overflow, if new input before previous one read)
- Whether Posting of Interrupts Enabled

Accessing device registers
- No special instruction for each device in the spirit of file access for all devices at OS level
- Devices same physical address space as main memory
- Each device has a sequence of addresses to reference its registers
- When a device address is posted on bus, the associated device register is accessed,
- Device address kept by OS is address of first register.
INTERRUPTS

- Interrupt events:
  - When device goes from non ready to ready.
  - On output devices, when interrupts are enabled after being disabled.

- Device control over interrupts.
  - Each device has an interrupt enable bit which decides if device signals interrupts to the CPU.

- CPU control over interrupts
  - The CPU has a register to decide if it reacts to posted interrupts for a device or not. Not reacting to device interrupts is raising processor priority (over devices, which also have priority).

- Interrupt reaction
  - Cycle stealing: After cycle, CPU checks (polls) for interrupt status of each device
  - CPU goes into kernel mode, reset on return from trap instructions
  - Executes an interrupt handler in main memory associated with the device.
  - Handler can do anything allowed in kernel mode
  - executed special return from interrupt instruction to restore mode
Clock Interrupts and Interrupt Priority

- Time of day clock keeps time of day readable by cpu.
- Real-time clock interrupts periodically allowing kernel to keep track of time.
- Given higher priority than other interrupts?
  - as CPU should not lose track of time.
- Higher priority interrupts serviced before lower ones.
- Process can disable receipt of interrupts from all devices lower than a certain priority by setting its priority to that level.
ATOMICITY AND BLOCKING

- Kernel code can potentially be executed by multiple threads?
  - A thread T1 that executes a user’s `getChar()`.
  - A thread T2 that is scheduled in the middle of `getChar()`
  - An interrupt handler invoked in response to keyboard input.

- Kernel code while accessing global kernel memory (e.g., scheduling tables) that must be accessed atomically:
  - Does not call methods to reschedule the processor.
  - Disables receipts of interrupt that can access global data or cause rescheduling (clock interrupts).
  - Disabling also prevents cascaded interrupts that can overflow stack

- While interrupts are disabled:
  - Kernel code should not block, e.g., call `wait()`
  - otherwise interrupts will not be serviced
DATA BLOCKS

- Network and disk
  - Interrupt on each byte received?
- Direct Memory Access
  - Device stores or outputs data blocks directly in main memory and interrupts.
USB

- Single port for heterogeneous devices
- Device registers?
  - Has the maximum number of registers?
  - Use DMA to store all data?
  - Proxy of device at or near port.
  - USB Port just provides a general packet-based protocol for communication with bus
MULTIPROCESSORS

- What does it mean to enable/disable interrupts receipt?
  - Each processor has a separate register indicating interrupt receipt is enabled, that is, its priority.

- If multiple processors enable interrupts, which one executes the handler?
  - A computer decided by the hardware.
  - A computer designated by the OS.
  - The first computer that polls for the interrupt.
  - ....
Kernel code can potentially be executed by multiple threads?
- Two threads executing on different processors.
- A thread T1 that executes a user’s getChar().
- A thread T2 that is scheduled in the middle of getChar().
- In response to clock interrupt or a yielding (rescheduling) call.

Kernel code while accessing global kernel memory (e.g. scheduling tables) that must be accessed atomically:
- Uses polling (test and set instruction) to prevent threads on different processors from interfering.
- Does not call methods to reschedule the processor, e.g. notify().
- Disables receipts of interrupt that can access global data or cause rescheduling (clock interrupts).
- Disabling also prevents cascaded interrupts that can overflow stack.

While interrupts are disabled, allow blocking calls such as wait():
- Interrupts will not be serviced.
Traps vs. Interrupts

- Like interrupts
  - Associated with handlers in main memory
  - Cause mode to change from user to kernel, restored by a special return from trap instruction

- Triggering Difference:
  - Can be executed in response to an internal CPU event
    - An error such as divide by zero
  - An explicit instruction trap executed by system call that takes as argument indicating which system call.
  - Not in response to an external event.

- Enabling or disabling of traps?
  - No, as CPU triggers and reacts to this event
  - Hardware version of exceptions
CPU in kernel mode has more power than one in user mode
- Can access device registers
- Turn on/off interrupts
- Access kernel memory
- Change page tables
- ....

Cost of transitions between user mode and kernel mode?
- more expensive than a library call in user space.
- multiple actions done in one system call for efficiency
Memory divided into

- Processes address space to which process virtual memory is mapped
- Kernel address space consisting of:
  - interrupt and trap handlers (addresses fixed by hardware)
  - other kernel data structures such as schedulers and virtual memory manager
Part of OS that runs in kernel mode is called the kernel.
Can/should some part of the OS can run in user mode in process virtual memory?
  - File Manager has been a candidate for flexibility
  - User-defined file manager ala user-defined shell.
Replacement for traps:
  - IPC
  - Non Kernel OS component can be distributed – accessed by remote process.