Slide 1

Administrivia

- Homework 3 due date extended until Monday.
- Homework 4 (more problems on memory management) to be on Web by Friday. Due following Monday.
- Courses next term:

  CSCI 3294 ("Unix Power Tools") will be similar to course last spring, syllabus/notes from under "Old course materials" on my Web page.

  CSCI 3394 ("Topics in Parallel Computing") will be a follow-on to CSCI 3366.

Slide 2

Memory Management — Short Wrap-Up

- Goals: Sharing/partitioning of (real) memory among processes, program relocation, memory protection, "virtual memory".
- Different mechanisms for sharing/partitioning memory — contiguous-allocation schemes, paging, segmentation. Which one is used depends a lot on hardware (MMU).
- Different mechanisms for supporting virtual memory — swapping, paging. All involve decision-making; for paging, "page replacement algorithms".
- Review minute essays for 11/02 and 11/04.

Slide 3

I/O Management

- Operating system as resource manager — share I/O devices among processes/users.
- Operating system as virtual machine — hide details of interaction with devices, present a nicer interface to application programs.

Slide 4

I/O Hardware, Revisited

- First, a review of I/O hardware — simplified and somewhat abstract view, mostly focusing on how low-level programs communicate with it.
- Many, many kinds of I/O devices — disks, tapes, mice, screens, etc., etc. Can be useful to try to classify as “block devices” versus “character devices”.
- Many/most devices are connected to CPU via a “device controller” that manages low-level details — so o/s talks to controller, not directly to device.
- Interaction between CPU and controllers is via registers in controller (write to tell controller to do something, read to inquire about status), plus (sometimes) data buffer.

  Example — parallel port (connected to printers, etc.) has control register (example bit — linefeed), status register (example bit — busy), data register (one byte of data). These map onto the wires connecting the device to the CPU.

Slide 5

Accessing Device Controller Registers

- Two basic approaches:
  - Define "I/O ports" and access via special instructions.
  - "Memory-mapped I/O" — map some (real) addresses to device-controller registers.

  Some systems use hybrid approach.
- Making either one work requires some hardware complexity, and there are tradeoffs; memory-mapped I/O currently more common. (Notice implications for writing device drivers — which scheme allows writing them without assembly language?)

Slide 6

Direct Memory Access (DMA)

- When reading more than one byte (e.g., from disk), device controller typically reads into internal buffer, checking for errors. How to then transfer to memory?
- One way — CPU makes transfer, byte by byte.
- Another way — DMA controller makes transfer, having been given a target memory location and a count.
- Which is better? consider speed of DMA versus speed of CPU, potential for overlapping data transfer and computation.

Slide 7

Interrupts, Revisited

- When I/O device finishes its work, it generates interrupt, typically actually signalling interrupt controller.

  Interrupt controller signals CPU, with indication of which device caused interrupt, or ignores interrupt (so device controller keeps trying) if interrupt can't be processed right now.

- Processing is now similar to what happens on traps (interrupts generated by system calls, page faults, other errors):

  Hardware locates proper interrupt handler (probably using interrupt vector), saves critical info such as program counter, and transfers control (probably switching into supervisor mode).

  Interrupt handler saves other info needed to restart interrupted process, tells interrupt controller when another interrupt can be handled, and performs minimal processing of interrupt.

Slide 8

Interrupts, Revisited, A Bit More

- Notice how pipelining complicates things — restarting is much easier with precise interrupts (all instructions before interrupted one complete, none past interrupted one complete, etc.), but these are difficult to get with pipelined processor.

Slide 9

Mechanics of I/O — Polling Versus Interrupts

- Programmed I/O: Program tells controller what to do and busy-waits until it says it's done.

  Simple but potentially inefficient.
- Interrupt-driven I/O: Program tells controller what to do and then blocks. While it's blocked, other processes run. When requested operation is done, controller generates interrupt, interrupt handler unblocks original program,
- I/O using DMA: Similar to interrupt-driven I/O, but transfer of data to memory done by DMA controller, only one interrupt per block of data.

Slide 10

Goals of I/O Software

- Device independence — application programs shouldn't need to know what kind of device.
- Uniform naming — conventions that apply to all devices (e.g., Unix path names, Windows drive letter and path name).
- Error handling — handle errors at as low a level as possible, retry/correct if possible.
- "Synchronous interface to asynchronous operations."
- Buffering.
- Device sharing / dedication.

Slide 11

Minute Essay

- None — sign in.