ECS150 Discussion Section

Sophie Engle

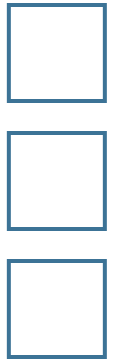
(*February 04/06 2004*)

Announcements

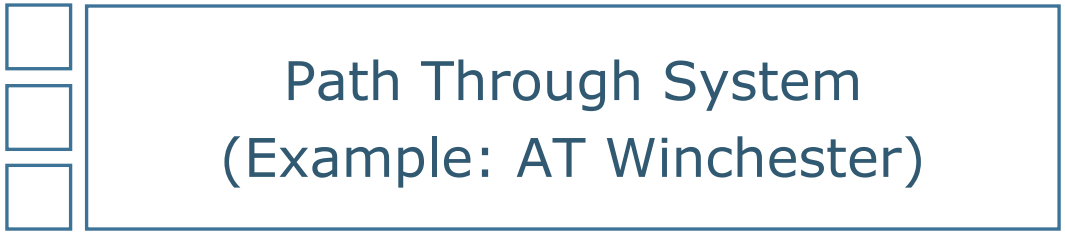
- **Midterm**
  - Thursday February 19th
  - Open book, open note
- **Homework**
  - Homework 1 solution and grades on website
  - Next homework assignment most likely due Friday February 13th

Agenda

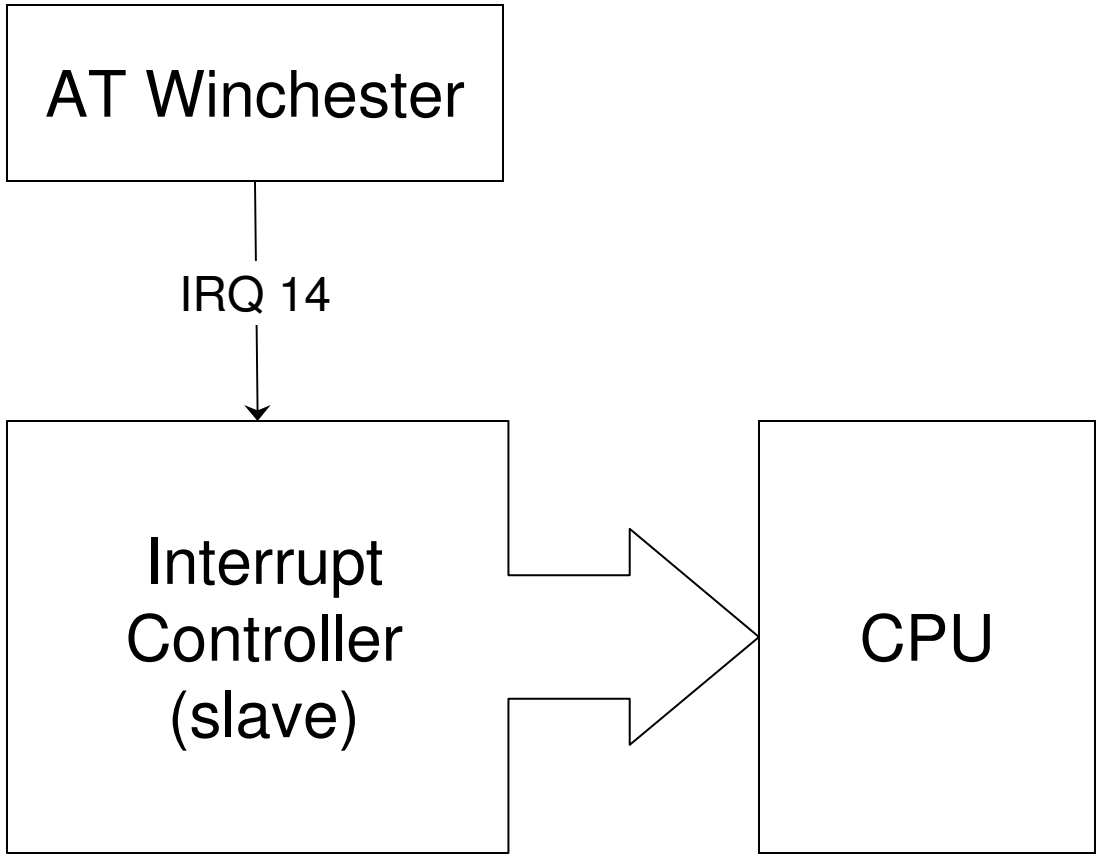
- Discuss interrupt handling in Minix
  - Hardware interrupts
  - Software interrupts, System calls
- Resources
  - Minix book, pages 128 – 140
  - Minix source



Hardware Interrupts

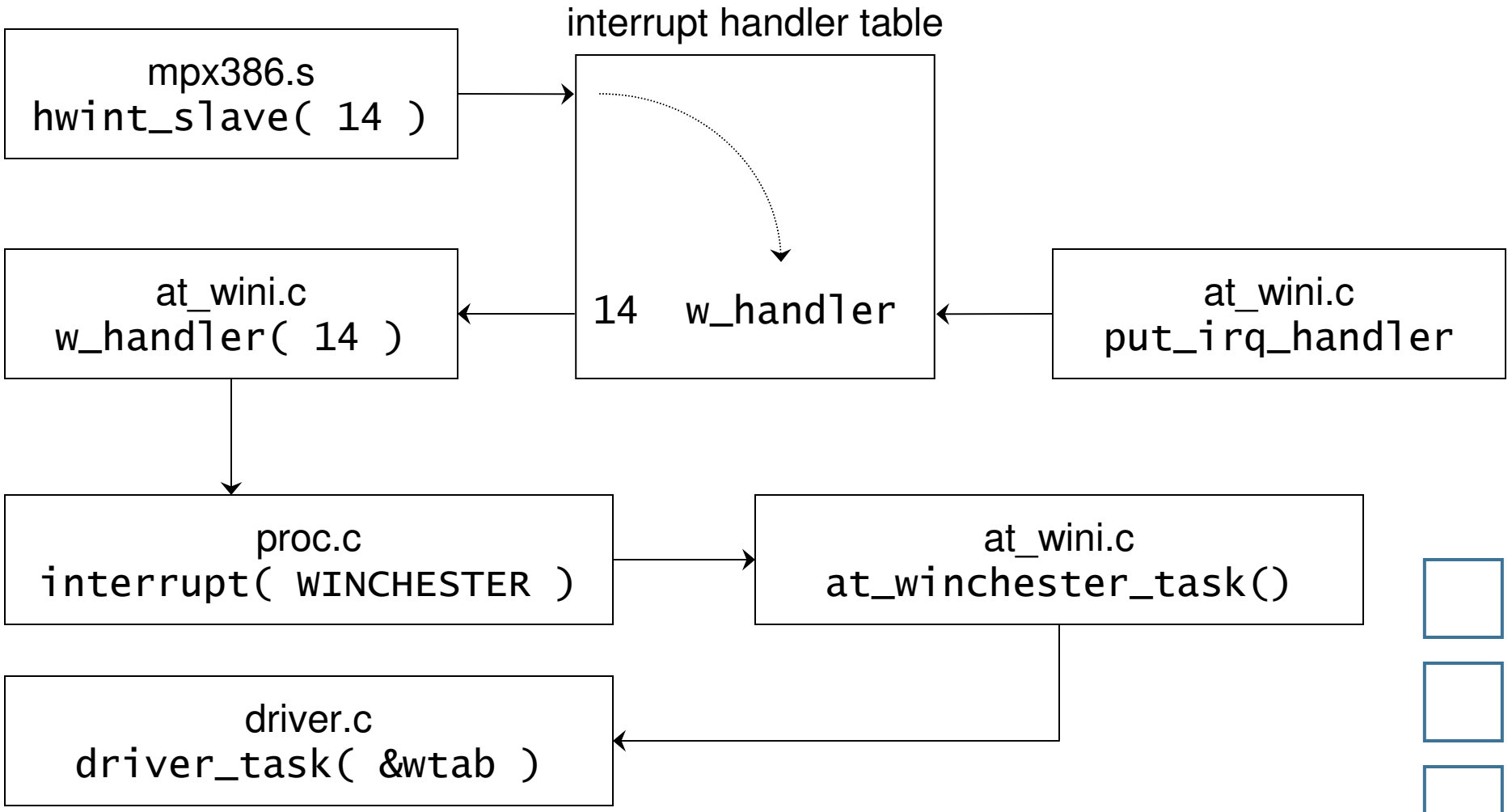


Simplified Hardware Path



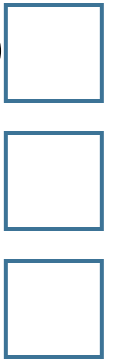
* see **Interrupt Processing Hardware** diagram on page 128

Simplified Software Path



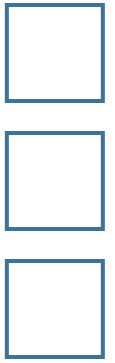
Software Path: mpx386.s

- mpx386.s: hwinit_master(int irq)
  - hwinit_master(irq) or hwinit_slave(irq) respond to the actual interrupt
  - save pushes all registers necessary to restart the interrupted process
  - the irq is disabled until the interrupt is handled
  - the controller is reset and the CPU is allowed to receive interrupts from other sources



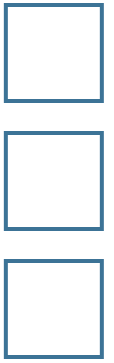
Software Path: mpx386.s

- mpx386.s: hwinit_master(int irq)
  - the handler specified in the table of low-level routines is called
    - (more details later)
  - interrupts are disabled again after the call instruction returns
  - interrupt controller prepared to respond to interrupting device
  - interrupts (and irq) are re-enabled



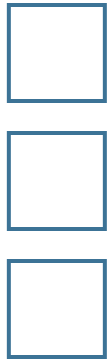
Software Path: at_wini.c

- at_wini.c : w_identify()
  - called by the driver to find out if device exists
  - if exists, registers w_handler as the interrupt handler for irq 14
    - put_irq_handler(wn->irq, w_handler);
    - enable_irq(wn->irq);
  - this is the handler called by hwinit_slave(irq) when interrupted by the AT Winchester



Software Path: at_wini.c

- at_wini.c : w_handler(int irq)
  - reads status of drive
  - calls interrupt(WINCHESTER)



Software Path: proc.c

- proc.c : interrupt(int task)
  - converts the interrupt into a message for the task that handles the interrupting device
    - task in this case is at_winchester_task() in at_wini.c
    - eventually calls driver_task() located in driver.c
  - first checks if an interrupt was already being serviced (k_reenter)
    - if so, adds current interrupt to queue of held interrupts
    - queue of held interrupts is handled in unhold()

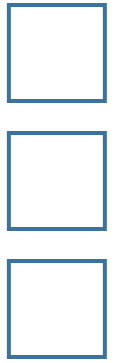
Software Path: proc.c

- proc.c : interrupt(int task)
  - next checks if task is waiting for an interrupt
    - task must be ready to receive interrupt
    - if not, task is blocked
      - function mini_rec checks for blocked interrupts
  - otherwise, sends message (with interrupt) to task
  - then schedules task to run

Software Interrupts

Software Interrupts

- System calls
  - Basically “software interrupts”
  - Behave similarly to hardware
    - Call converted to message, sent to task
    - Interrupt originates from software versus hardware



System Calls

- Kernel Code
  - _s_call in mpx386.s handles software interrupt (versus hwinit_master() or hwinit_slave())
  - sys_call in proc.c converts interrupt into message similarly to interrupt
    - if message sending needed, calls mini_send
    - if message receiving needed, calls mini_rec
  - See also sys_task() in system.c