• The following slides describe how an x86-processor based computer running Linux boots up
• This is part of your reading assignment
• Adapted from “Understanding the Linux Kernel” by Bovet and Cesati
  – I recommend this book to those interested in delving deeper into the Linux kernel

System Bootstrap
System Bootstrap or Bootup

- Bringing OS into memory and having the processor execute it
- Initialization of kernel data structures
- Creation of user processes and transfer of control to one of them
- Device initialization
  - Agreement on interrupts, bringing RAM to known state, setting certain registers, telling the PIT how frequently to interrupt, …

Basic Input/Output System (BIOS)

- When a computer is powered on
  - H/W raises logical value of RESET pin of CPU
  - Some registers are assigned fixed values
  - Code at address 0xfffffff0 is executed, mapped by H/W to a ROM
    - Set of programs stored in ROM called BIOS which includes several interrupt-driven procedures to handle the H/W devices
    - Some OSes use these procedures, Linux provides its own device drivers
- BIOS bootstrap performs these 4 operations
  - 1. Power-on Self-Test (POST)
    - Tests to establish which devices are present
  - 2. Initialization of H/W devices
    - So devices can operate without conflicts on ITQ lines and I/O ports
  - 3. Searches for OS to boot
    - Search order defined in BIOS. E.g., floppy, then any hard disk, then any CD-ROM
  - 4. Loads the Boot Loader
    - Copies the contents from the first sector of a valid device into RAM, starting from a fixed address (0x00007c00), jumps to that address, and executes the code just loaded
Boot Loader

- BIOS loads a small boot loader into RAM at 0x00007c00
  - Small enough so it can fit into a disk sector
- This program moves itself to 0x0009a000 and loads the rest of the boot loader at 0x0009b000
- This latter program reads map of available Oses from disk, offers the user a prompt to choose the OS to boot
- After the choice is made (by user or timeout+default), copies setup() assembly code to 0x00090200 and kernel to 0x0001000
- Jumps to setup()

setup()

- Initializes H/W devices
  - BIOS has already done most of this, but Linux does not rely on it
- Important operations
  - Invokes a BIOS procedure to find how much RAM there is
  - Sets keyboard repeat delay and rate
  - Init. Video adapter card
  - Reinit. Hard disk controller and determine disk parameters
  - Sets up interrupt descriptor table
  - Reprogrammable Interrupt Controller (PIC) and maps the 16 IRQ lines to the range of vectors from 32 to 47
    - The BIOS maps H/W interrupts in 0-15, which is used for CPU exceptions
  - Switches CPU to kernel mode
  - Jumps to startup_32() assembly function
startup_32()

- Salient operations
  - 1. Init. Segemention registers and a provisional stark
  - 2. Decompress kernel image, place it at a fixed location and jump to it
  - 3. Set kernel mode stack for process 0
  - 4. Fill the IDT with null interrupt handlers
  - 5. Identify processor model
  - 6. Load the idtr register with address of the IDT table
  - 7. Jump to the start_kernel() function

start_kernel()

- Completes the init. of the kernel. Important operations are
  - Init. Page tables
  - Finalize the IDT
  - Memory manager related data structs init.
  - Init. system time and date
  - Kernel thread for process 1 is created
    - This thread creates other kernel threads and executes /sbin/init
  - Finally, the login prompt appears (or graphical screen if X window is launched at startup)
  - Linux is up and running!