Loading and Bootstrapping

November 13

CSC201 Section 002
Fall, 2000
“Loading” a Program for Execution

• This is managed by the operating system (O.S.)

1. Allocate memory for the program

2. Copy the data, stack, and code into memory
   - Adjust operand addresses / offsets based on this placement
   - "Dynamic" linking: at load time

3. Transfer control to this program
If the Operating System Loads Programs, Who Loads the Operating System?

- At hardware power up: execute the instruction at a specific address (ex.: 000FFFF0h)

- A ROM (read-only memory) contains a startup program at this address
  - Read the “first part” of the operating system from a specific location on the system disk drive (ex.: sector 1)
  - Copy this first part to a specific location in memory
  - Unconditionally jump to the first instruction in this first part
Loading (cont.)

• The “first part” of the O.S.:
  - Reads the remainder of the O.S. from disk into memory
  - Unconditionally jumps to the first instruction of a *command reader* (shell)
Command Reader (Shell) Processing

1. Print the "prompt" character and wait for input from the keyboard
2. Read the command typed by the user
3. Look (on disk) for a file with the name typed by user
4. Check memory allocation table to find space for program, make a record in table
5. Copy the program executable into memory
Shell (cont.)

6. Resolve any instructions/addresses noted by the assembler (i.e., link)

7. Initialize the ESP

8. Unconditional jump to first instruction of the program
Return to O.S. (When the program terminates)

• The operating system is no longer running when the user program starts!
  - no monitoring or "oversight" by the O.S.

• "Ret" statement by program exits back to the operating system
  - Deallocate program memory
  - Unconditional jump to command reader

• Other ways of returning to O.S.
  - Exceptions, interrupts... wait til next lectures!
Sharing Control With the Debugger

• How can a program run "under control" of a debugger?

• Method #1: insert code into the user's program that jumps to the debugger at specific points

• Method #2: Set a bit in the program status word that generates an interrupt after every instruction
  - interrupt service routine transfers control to the debugger