## *Chapter 3 – Dietel – Lesson Plan for Abrams' Section*

### Items to Highlight

- Why do we need concurrency in a single-processor computer? Ask class for examples from every-day life.
  - Your mail client wants to asynchronously change a desktop icon to show that you have new mail.
  - You want to compose a new email while your computer sends/receives other email.
  - You want to print a document while editing another document.
  - You are a developer, and find it convenient to architect code using modules which run concurrently – such as a windowing system that listens for GUI events while other code draws the controls in a window.
- What is a process?
- What data structures does the OS need to…
  - Keep track of what processes exist?
    - Process table – essentially what you see from the "ps" command.
  - Keep track of the "state" of each process?
    - What info must be kept about a process in a PCB? See slide 11.
- What are the basic states of a process?
  - Ready, Running, Blocked
  - What are the transitions? Draw on board.
- What process management must be done?
  - See slide 6.
- Processes are created in hierarchical relationship – See slide 14.
  - INIT is process 1.
  - Child inherits I/O descriptors from parent
  - If parent dies, what happens to kids? (Inherited by login shell.)
  - If a parent dies before a child, and you look at the child's PCB, what is recorded for the parent pid?
    - For this orphan, its parent pid becomes 1 (init), because init never terminates.
  - In Unix, if child dies after parent, child becomes zombie waiting forever for parent to execute "wait".
- Suspend/Resume: section 3.3.4. Read book. Not really used today.
- Context switching (3.3.5)
  - What causes a context switch?
    - Timer interrupt
    - Kernel call (e.g., to request I/O)
    - HW interrupt
  - Mention: when interrupt raised (e.g., timer), current instruction finishes.
  - Talk with class to reason out how exactly a context switch is implemented.
  - Talk about RISC having multiple context registers.

- Mention that interrupt handler on some machines must save its own registers on stack upon entry to avoid cost of saving all registers.