INTER-PROCESS COMMUNICATION AND PROCESS MANAGEMENT

DUE DATES, ETC.

WHAT YOU'LL BE IMPLEMENTING

- **spawn(<process name>)**
- **Message boxes**
  - bounded buffer inter-process communication
- **Keyboard Input**
  - `putchar()`
  - `do_getchar()`
- **kill(pid)**
- **wait(pid)**
- **This is a reasonable order to do it in!**

GENERAL NOTES

- **Still need to think about interrupts**
  - Use critical sections sparingly
- **The supplied scheduler uses lottery scheduling**
  - *Don't* break it (`total_ready_priority`)
- **Implement carefully**
  - the given test cases won't test everything

MESSAGE BOXES

- **Look to Tanenbaum (MOS)**
- **Reclaim them**
  - usage count

KEYBOARD HANDLIN'

- **Use a message box to capture keystrokes in `putchar()`**
  - Discard characters when the buffer is full
- **Read keystrokes from the message box in `do_getchar()`**
- **Initialize at kernel startup**
- **The basic IRQ1 interrupt handling is setup in init_idt(), entry.S:irq1_entry and keyboard.c**

SPAWN

- **Collect information for the task**
  - Entry point -> look at ramdisk_find()
  - What about field task_type = ?
- **Setting up resources and scheduling**
  - Allocate a PCB
  - Assign a PID
  - Allocate stacks
  - Remember `total_ready_priority`

KILL

- **A process should be killed immediately**
  - Ready, blocked, or sleeping, doesn't matter
- **If blocked on a synchronization primitive, other processes should be unaffected by its death**
  - But don't recover locks
- **Reclaiming memory is important**
  - PCB
  - Look at the robinhood test case, and think about why it needs to have reclamation
- **`total_ready_priority`**

WAIT

- Allows a process to block until a given process completes execution
- Basically, wake up on kill's and exit's
- How could the PCB be changed to make this behavior possible?

TESTING

- **tasks**

DEMO