CSCE313 Refsheet; © Josh Wright 2015; Last Updated: November 2, 2016

Batch OS
- jobs are submitted in batches
- just used DMA for IO
- basically FIFO, but with using DMA for loading P2 while P1 is still running

Time Sharing OS
- CPU cycles through jobs in defined order
- basically round robin

Exception Control Flow
- exception: transfer of control from the user program to the OS in response to an event
- event: interrupt or system call or something
- CPU checks for interrupts before every instruction
- **Asynchronous Exception**: caused by something external to the processor. (e.g. not a system call)
  - ∗ e.g. IO interrupts, hard/soft reset
- **Interrupt Vector Table:** stored in CPU, has function pointers at indexes identified by interrupt codes
  - ∗ one slot for every possible type of interrupt
  - ∗ first thing handler does is disable interrupts, then re-enable when done
- **Synchronous exceptions:** caused by executing an instruction (internal to CPU)
  - ∗ Trap: intentional; system calls, breakpoints
    - returns code to next userland instruction
  - ∗ Faults: unintentional. maybe recoverable; generally retry or abort. e.g. page fault, floating point exception
  - ∗ Abort: unintentional; unrecoverable
- RTU: Return To User

Dual Mode
- CPU supports two modes: kernel mode and user mode
  - ∗ keeps a flag register depending on current mode
- user mode is disallowed some dangerous instructions and only allowed it's mapped memory regions
- kernel mode gets full control over everything
- must switch between the two, which has overhead
- Memory Protection: uses MMU hardware
  - ∗ kernel has no memory protection
- hardware timer: fires interrupts at predefined intervals
  - ∗ kernel will set timed interrupts to switch control back to the kernel from the user program periodically
  - ∗ setting/clearing timers is a privileged instruction
- user→kernel: on interrupt, syscall
- kernel→user: end of interrupt handler, or at proc start

Unix Processes
- process: instance of running program
- has it's own private address space
- Zombie Process: when a child process exits, the kernel must keep it's memory around just in case the parent needs to know if it exited cleanly
  - ∗ this child process is a zombie process.
  - ∗ call `wait()` to avoid this
  - ∗ reaping is the process of killing zombies

Process Lifetime/States
- all this stuff looks like continuous running to the process
- **Ready:** scheduled to be run
- **Running:** currently running
- **Blocked:** waiting on something. (IO or other wait)
- transitions:
  - ∗ newly created processes go to ready; after exit, leaves running position
  - ∗ ready→running: dispatch
  - ∗ running→ready: preempt
    - also happens when scheduling timer expires
  - ∗ running→blocked: triggers IO event or wait
  - ∗ blocked→ready: IO event or wait finishes
- only one process is in running state per CPU core
- Additional states: (managed by memory scheduler)
  - ∗ Suspended Ready: ready to run, but swapped out of main memory due to memory constraints. ; get here from start or ready ; leave to ready when ready
  - ∗ Suspended Blocked: swapped out of main memory, and also waiting on some external event ; get here from blocked ; leave to suspended ready or blocked
- **Process Control Block** (PCB)
  - ∗ contains process id, state, saved registers, memory maps, child processes, owned resources, file descriptors
  - ∗ whenever a context switch happens, the previous process state is stored in and restored from the PCB
- Job Queue: set of all processes in a system
- Ready Queue: in main memory, waiting to be run
- Device Queues: waiting for IO by device

Scheduling
- Latency: how long it takes for the job to complete
- Throughput: # of jobs can be completed per unit time
- Overhead: how much work the scheduler must do
- Fairness: do different users/process types/other factors influence scheduling priority?
- Predictability: also consistency
- Preemptive Scheduler: one that takes control away from a process (usually through timed interrupts)
- Work Conserving: never leave a CPU idle
- Time Quantum: length of scheduler interrupt timer
- Waiting Time: time spend in not running
- Service (Execution) Time: how long the job is running
- Response (Completion) Time: wall clock time process takes ; response = waiting + ready
- Long-Term Scheduler: (Job Scheduler)
  - ∗ selects which job to brought into the ready queue
  - ∗ should select a good mix of CPU and IO bound processes
  - ∗ Short-Term Scheduler: (CPU Scheduler)
  - ∗ selects what to run from the ready queue
  - ∗ runs every time quantum, so it shall be fast
- Medium-Time Scheduler: swaps out programs from main memory (suspends them)
- First In First Out (FIFO), or FCFS:
  - ∗ works well on one really long task with many small tasks, if the long task comes first (e.g. servers)
- **Shortest Remaining Time First:** (SRTF) (or Shortest Job First (SJF))
  - ∗ the ideal scheduler in most cases
    - advantage best for average response time
    - disadvantage: longer tasks can get starved
    - disadvantage: high variance on average response time
  - ∗ not really possible because we can't know how long a job will take ahead of time
    - can be approximated by guessing the next CPU burst
  - ∗ if all tasks are the same length, runs them all in order, no preemption
  - ∗ shortest task starts and finishes first
- **Round Robin:**
  - ∗ each task gets run for one time quantum
    - choice of time quantum is critical
  - ∗ compromise between SJF and FIFO
  - ∗ always fair
  - ∗ time quantum too small: too much overhead
  - ∗ time quantum approaches ∞: equivalent to FIFO
- **Multi-Level Feedback Scheduling:**
  - ∗ have multiple ready queues
    - sub-queue are prioritized
  - ∗ processes start in highest priority queue, then are moved to lower queue after using CPU too much
  - ∗ must schedule between queues:
    - fixed priority: highest priority, then lower (not fair)
    - time slicing: fixed percentage of time to each queue.
  - ∗ approximates SRTF: CPU-bound suffers, lets IO bound stay near top
  - ∗ example: putting in superfluous IO waits
- utilization approaches 100%, latency and throughput suffer