Today

http://PollEv.com/sprenkle

- Scheduling Processes

Review

- How does a process create another process?
  - What are the characteristics of that other process?
- How do you make a process execute a different program?
- How do you make a process wait for its child process to finish?
- What causes a zombie process?
- What is serial execution? What are the benefits and limitations of serial execution?

Review: Process Management

- Fork
  - Creates a new process
    - Copy of the original process, its state
  - Returns twice – once to parent, once to child
  - Both processes start/continue executing after call to fork()
- Wait
  - Parent waits for one of its children to exit
    - Can get the child's exit status
- Exec
  - Make a process execute a different program

Review: Zombie (or defunct) Processes

- Process has terminated
  - All of process's resources are cleaned up
  - EXCEPT its PCB
  - Dead, but not gone…
  - All child processes go into this state, though, likely just briefly
- Parent process has not yet collected its status
  - Parent hasn't completed its call to `wait`

Serial Execution

Benefits

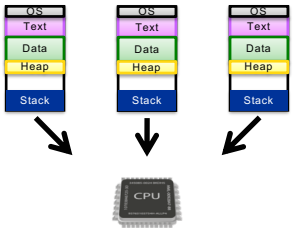
- Simpler model
- Don't need to worry about conflicts between applications
  - (can't accidentally touch each other's memory)
- Individual programs will execute faster

Limitations

- It's not really feasible for us to only have one process run at a time
- CPU is idle when process needs I/O or is blocking for some other reason

Goal: Multiprogramming

- Multiprogramming: have multiple programs available to the machine, even if you only have one CPU core that can execute them.



- Switch between programs when there is downtime

Goal: Multiprogramming

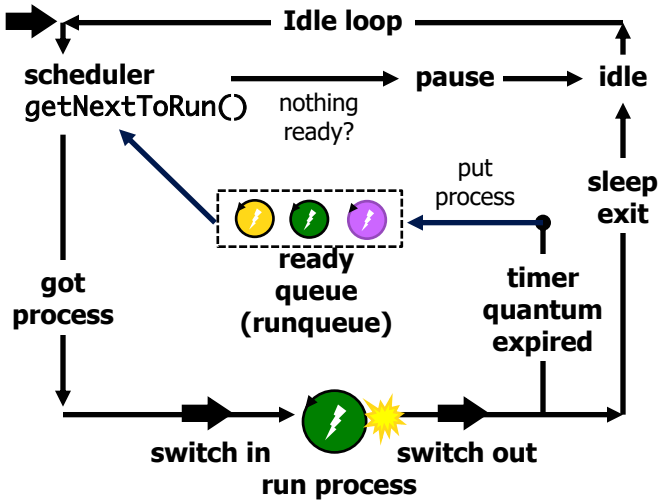
- Multiprogramming: have multiple programs available to the machine, even if you only have one CPU that can execute them
- When I/O requested by process, CPU not needed!
  - Allow another program to run
- Other reasons that the process ***yields*** the CPU
- Challenge: machine is NOT exclusive to one executing process
  - What if one running program…
    - Monopolizes CPU, memory?
    - Reads/writes another's memory?
    - Uses I/O device being used by another?

Foreshadowing …

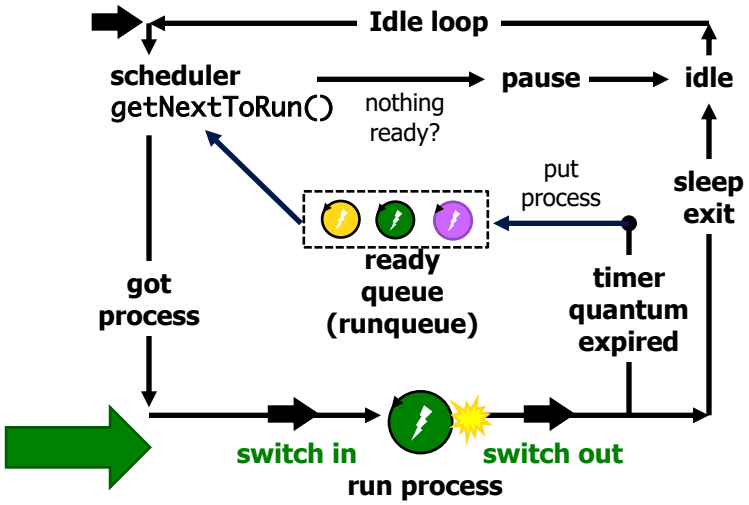
CPU Scheduling 101

- The OS scheduler makes a sequence of "moves"
  - Next move: if a CPU core is idle, pick a ready thread from the ready pool and dispatch it (run it).
  - Scheduler's choice is "nondeterministic"
  - Scheduler and machine determine the interleaving of execution (a schedule).

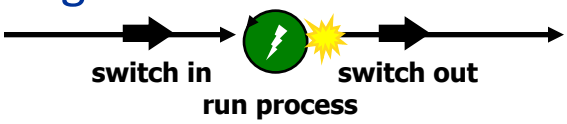
CPU's Flow



CPU's Flow



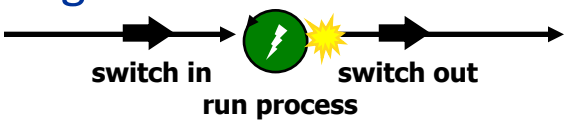
Switching



- What causes a CPU to switch out of the current process?
  - Process exits
  - Timer interrupt: quantum expired
  - Process needs a resource that isn't available or I/O
  - Process yields
  - …?
- Switching takes time and resources

Based on what we know about dual mode, what does a "switch" likely look like?

Switching

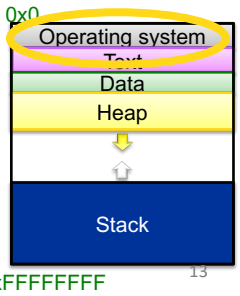


- What causes a CPU to switch out of the current process?
  - Process exits
  - Timer interrupt: quantum expired
  - Process needs a resource that isn't available or I/O
  - Process yields
  - …?
- Switching takes time and resources
  - Need to copy state so can start up again at the same place

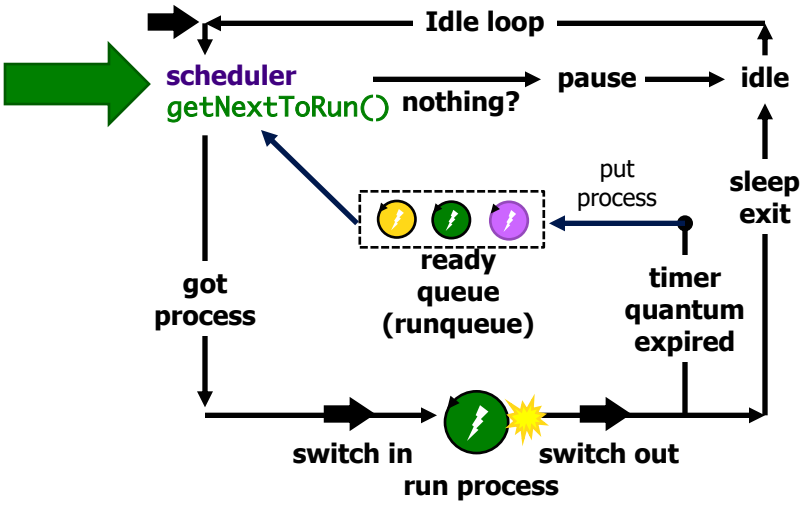
Called a *context switch*—switching between process contexts

Context Switching: Performance

- Even though it's fast, context switching is expensive:
  1. time spent is 100% overhead
  2. must invalidate other processes' resources (caches, memory mappings)
  3. kernel must execute – it must be accessible in memory
- Solution to #3:
  - keep kernel mapped in every process VAS
  - protect it to be inaccessible



CPU's Flow



CPU Scheduler Scenario

- One CPU
- Cards represent number of seconds for the process to run
  - (This info is not usually known *a priori*)
- When the scheduler "picks up", all processes are on the ready queue

Exploring Scheduling Algorithms

- **Requirements**: Execute processes to completion
- What are some possible algorithms to schedule the processes?
  - Should be relatively simple – we need the scheduler to be fast
  - Should come up with at least 3 algorithms
  - What are the outcomes from the algorithms?
- Discuss their tradeoffs
  - What would be a "better" schedule?
  - How do you **measure** "good"?

A new option: Non-Preemptive vs Preemptive

- Depending upon which scheduling opportunities are used by a scheduler, the scheduling can be:
  - **Non-Preemptive**: The scheduler will allow the running process to continue to run as long as it remains ready (i.e., doesn't block or exit).
  - **Preemptive**: The scheduler may set aside the running process in favor or another at any scheduling opportunity
    - Enables time-sharing, priority scheduling

Exploring Scheduling Algorithms

- **Algorithm**: Time slice: .5 seconds
  - After .5 seconds, boot the currently executing process
  - Assume .1 seconds of switching cost/overhead
- Discuss:
  - What is a valid schedule?
  - What would be a "better" schedule?
  - How do you measure "better"?

Changing it up: Priority

- Cards now have a priority
- In Unix, “niceness”
  - Highest priority: −20
  - Lowest priority: 19
  - Default: 0
- Processes inherit niceness from parent
- Cards:
  - Clubs (lowest): 19
  - Diamonds: 0
  - Hearts: -2
  - Spades: -20

Impact of Priority

- **Algorithm**: Time slice: .5 seconds
  - After .5 seconds, boot the currently executing process
  - Assume .1 seconds of switching cost/overhead
- Discuss:
  - How do your metrics of “better” change?
  - How do your algorithms change?

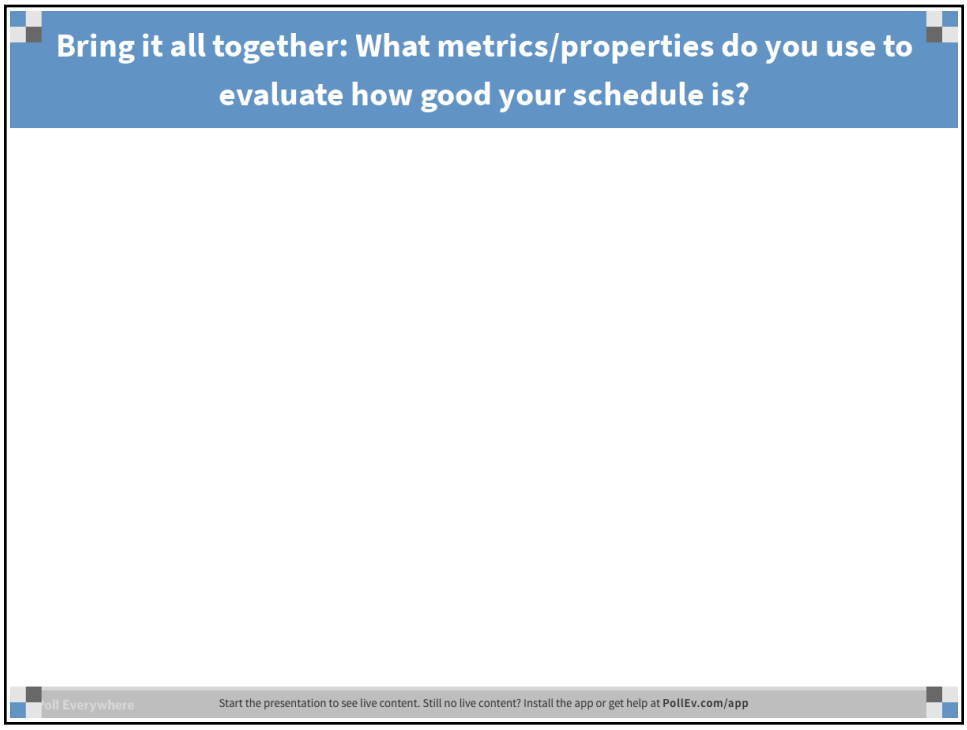
Other considerations

- How would your recommendations change if the time slice was 10 seconds? 1 minute?
- What other information would be helpful to make decisions?

Impact of System

- Consider how your algorithms would change if your system is
  - Super computer
  - Nuclear power plant or medical device
  - Your personal computer
- Discuss:
  - How do your metrics of "better" change?

(stopped here)



Looking Ahead

- Project 1 due tomorrow