Scheduling

Dave Eckhardt
de0u@andrew.cmu.edu

Synchronization

- “Pop Quiz”
- Project 2
- Outline
  - Chapter 6

CPU-I/O Cycle

- Process view: 2 states
  - Running
  - Waiting for I/O
- Life Cycle
  - I/O (load), CPU, I/O, CPU, .., CPU (exit)

CPU Burst Lengths

- Overall
  - Exponential fall-off in CPU burst length
- CPU-bound
  - Batch job
  - Long CPU bursts
- I/O-bound
  - Copy, Data acquisition, ...
  - Tiny CPU bursts

Preemptive?

- Four opportunities to schedule
  - Running process waits (I/O, child, ...)
  - Running process exits
  - Waiting process becomes runnable (I/O done)
  - Other interrupt (clock, page fault)
- Multitasking types
  - Preemptive: All four cause scheduling
  - “Cooperative”: only first two

CPU Scheduler

- Invoked when CPU becomes idle
  - Current task blocks
  - Clock interrupt
- Select next task
  - ***Quickly***
  - PCB's in: FIFO, priority queue, tree
- Switch (using “Dispatcher”)

Dispatcher

- Set down running task
  - Save register state
  - Update CPU usage information
  - Store PCB in “run queue”
- Pick up designated task
  - Activate new task's memory
    - Protection, mapping
  - Restore register state
  - Transfer to user mode

Scheduling Criteria

- Maximize/trade off
  - CPU utilization (“busy-ness”)
  - Throughput (“jobs per second”)
- Process view
  - Turnaround time (everything)
  - Waiting time (runnable but not running)
- User view
  - Response time (input/output latency)

Algorithms

- Don't try these at home
  - FCFS
  - SJF
  - Priority
- Reasonable
  - Round-Robin
  - Multi-level (plus feedback)
- Multiprocessor, real-time

FCFS- First Come, First Served

- Basic idea
  - Run task until relinquishes CPU
  - When runnable, place at end of FIFO queue
- Waiting time ***very*** dependent on mix
- “Convoy effect”
  - N tasks each make 1 I/O request, stall
  - 1 tasks executes very long CPU burst
  - Lather, rinse, repeat

SJF- Shortest Job First

- Basic idea
  - Choose task with shortest ***next*** CPU burst
  - Provably “optimal”
    - Minimizes average waiting time across tasks
  - ***Practically impossible*** (oh, well)
    - Could ***predict*** next burst length...
      - Text presents exponential average
      - No evaluation (Why not? Hmm...)

Priority

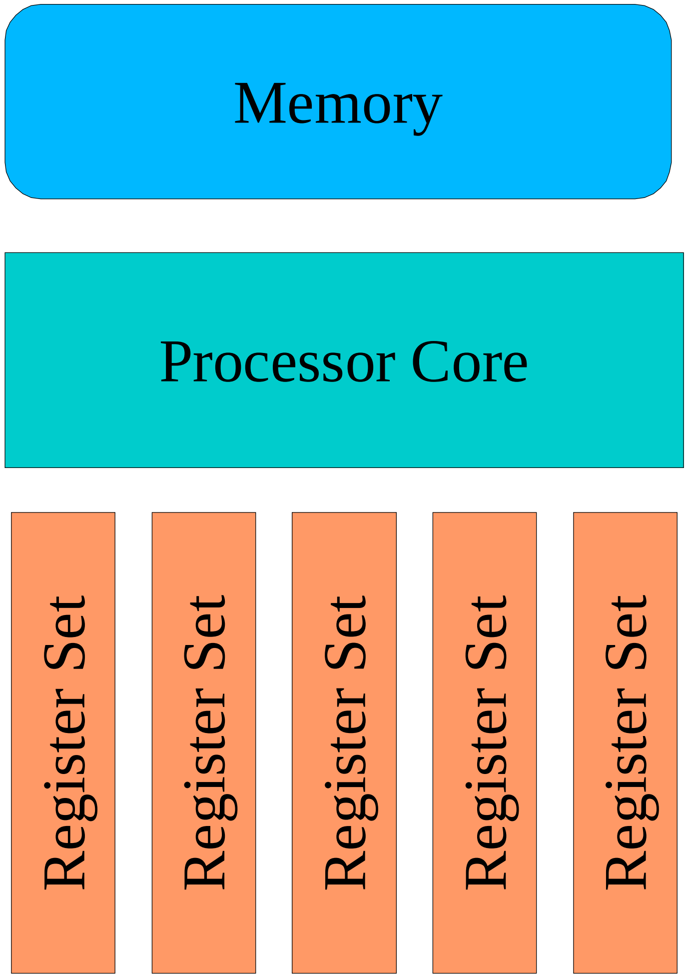
- Basic idea
  - Choose “most important” waiting task
    - Does “high priority” mean p=0 or p=255?
- Priority assignment
  - Static: fixed property (engineered?)
  - Dynamic: function of task behavior
- Big problem: ***Starvation***
  - Possible hack: aging

Round-Robin

- Basic idea
  - Run each task for a fixed “time quantum”
  - When quantum expires, append to FIFO queue
- “Fair”
  - But not “provably optimal”
- Choosing quantum length
  - Infinite = FCFS, Infinitesimal = “Processor sharing”
  - Balance “fairness” vs. context-switch costs

True "Processor Sharing"

- CDC Peripheral Processors
- Memory latency
  - Long, predictable
  - Every instruction
- Solution: round robin
  - Quantum = 1 instruction
- ~ Intel "superthreading"



Multi-level Queue

- N independent process queues
  - One per priority
  - Algorithm per queue
    - Interactive: round-robin
    - Batch: FCFS
- Inter-queue scheduling
  - Strict priority
  - Time slicing (e.g., weighted round-robin)

Multi-level ***Feedback*** Queue

- N queues, different quanta
- Exhaust your quantum?
  - Demoted to slower queue
    - Longer quantum
    - Lower priority
- Can you be promoted back up?
  - Maybe I/O promotes you
  - Maybe you “age” upward
- Popular “time-sharing” scheduler

Multiprocessor Scheduling

- Common assumptions
  - Homogeneous processors (same speed)
  - Uniform memory access (UMA)
- Load sharing / Load balancing
  - Single global ready queue – no false idleness
- Processor Affinity
  - Some processor may be more desirable or necessary
    - Special I/O device
    - Fast thread switch

Multiprocessor Scheduling - “SMP”

- Asymmetric multiprocessing
  - One processor is “special”
    - Executes all kernel-mode instructions
    - Schedules other processors
  - “Special” aka “bottleneck”
- Symmetric multiprocessing - “SMP”
  - “Gold standard”
  - Tricky

Real-time Scheduling

- Hard real-time
  - System must ***always*** meet performance goals
    - Or it's ***broken*** (think: avionics)
  - Designers must describe task requirements
    - Worst-case execution time of instruction sequences
  - “Prove” system response time
    - Argument or automatic verifier
  - Cannot use indeterminate-time technologies
    - Disks!

Real-time Scheduling

- Soft real-time
  - “Occasional” deadline failures tolerable
    - CNN video clip on PC
    - DVD playback on PC
  - ***Much*** cheaper than hard real-time
    - Real-time extension to timesharing OS
      - POSIX real-time extensions for Unix
    - Can estimate (vs. prove) task needs
  - Priority scheduler
  - Preemptible OS

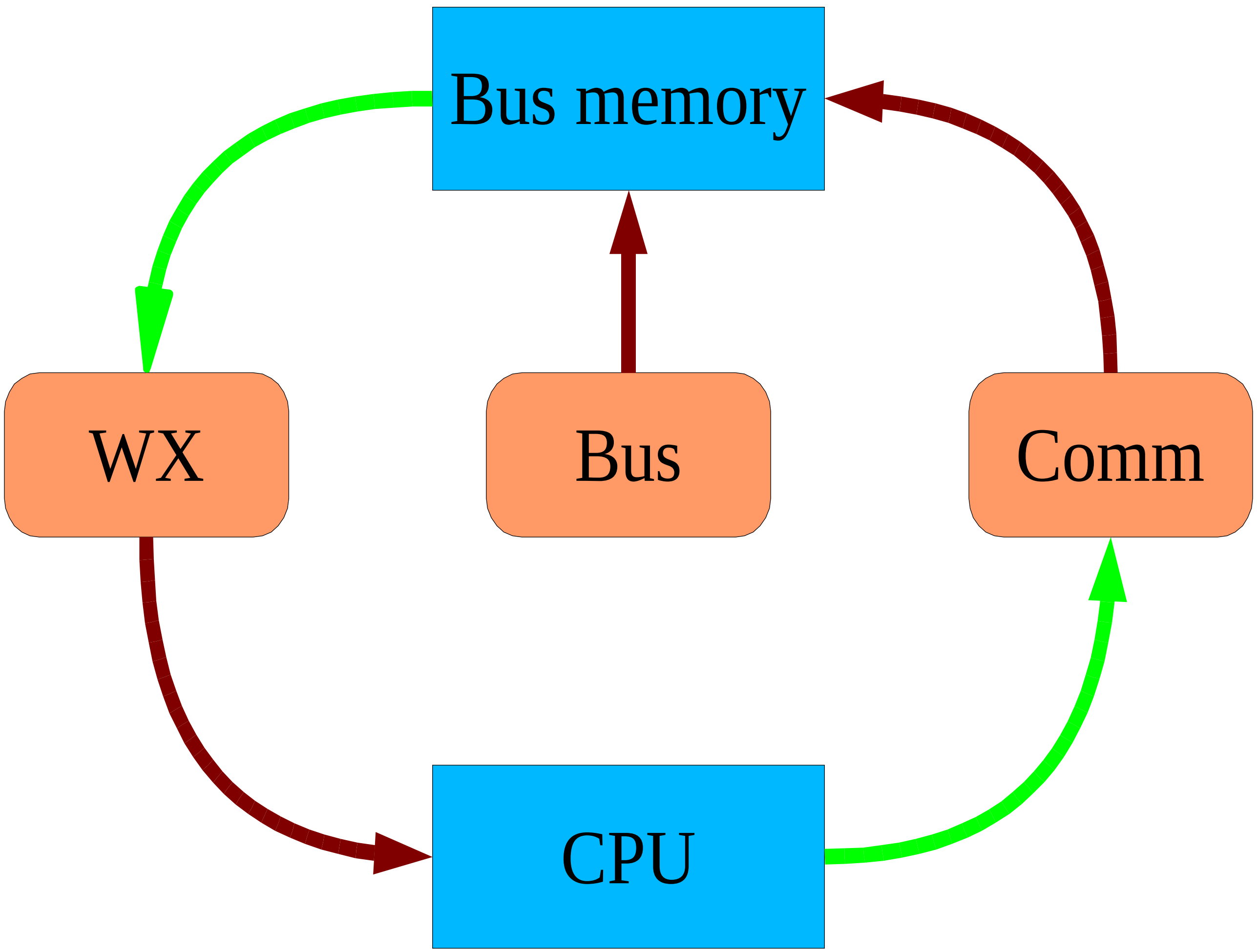
Mars Pathfinder probe (1997)

- "Information bus" (blackboard/whiteboard/...)
  - High-priority bus manager thread
- Meteorological data gathering
  - Occasional activity
  - Low-priority weather data thread
- Communication with Earth
  - Medium-priority communication thread
- Watchdog thread: "too quiet"? ***Reboot***

What could go wrong?

- Weather data locks “bus” memory to publish
- High-priority bus manager must wait
- ***Interrupt*** makes communication runnable
  - “Medium” priority, so preempts weather data
  - Highest-priority runnable task, so it spin-waits

What could go wrong?



History of an Idea

- Priority Inheritance Protocols: An Approach to Real-Time Synchronization
  - IEEE Transactions on Computers 39:9
    - Lui Sha (CMU SEI)
    - Ragunathan Rajkumar (IBM Research -> CMU ECE)
    - John Lehoczky (CMU Statistics)

History of an Idea

- Events
  - 1987-12 “Manuscript” received
  - 1988-05 Revised
  - 1990-09 Published
  - 1997-07 Rescues Mars Pathfinder
- History courtesy of Mike Jones
  - http://www.cs.cmu.edu/~rajkumar/mars.html

Scheduler Evaluation Approaches

- "Deterministic modeling"
  - aka "hand execution"
- Queueing theory
  - Math gets big fast
  - Math sensitive to assumptions
    - May be unrealistic (aka "wrong")
- Simulation
  - Workload model or trace-driven
  - GIGO hazard (either way)

Summary

- Round-robin is ok for simple cases
  - Certainly 80% of the conceptual weight
  - ***Certainly*** good enough for P3
- "Real" systems
  - Some multi-level feedback
  - Probably some soft real-time