Nathan Tung

CS111 Chapter 6.3 Outline #6

- Scheduling
  - Occasionally a queue builds up due to temporary overload
    - What order should disk requests be run to minimize latency?
    - Do we go in order of requests?
  - PCs have reduced the need for scheduling, but handling many internet users/load makes it important
- Scheduling Resources
  - Computer systems may (at different abstractions) reallocate varying memory to processes/users
  - Computer systems are collections of entities that require resources
    - Examples: threads, users, clients, services, requests, etc.
  - Scheduling is a set of policies/dispatch mechanisms to allocate these resources
    - Examples: processor time, physical memory space, disk space, network capacity, I/O-bus time
    - Policies include even priority distribution, priority of entities over another, or admission control
    - Scheduler is the component that implements a policy
  - It's difficult to choose what policy to use (high level goal vs. available policies)
  - Difficulty in policy
    - Giving the heavy-consumer more memory; user might not have purchased anything before
  - Difficulty in implementing mechanism/policy
    - Modules need to all implement a scheduling policy, or the policy will be ineffective
  - Difficulty in getting the actual implementation of mechanism right
    - Computer might collapse under overload
  - Receive Livelock
    - When rate of requests rises, the server might end up serving no one (server too busy)
      - Occurs when system is overloaded and requests come faster than can be processed
      - Example: modify policy so service thread gets a chance to run
- Scheduling Overload
  - Restructuring the thread manager allows easier policy implementation
  - Separate mechanism from policy
    - Separate dispatch mechanism (suspending/resuming thread) from scheduling policy (choosing which thread runs next) into distinct procedures
      - Easier to modify just the scheduling policy
    - Scheduling policy gets changed around a lot, since there is no standard for "best"
      - Throughput vs. utilization
  - Scheduler shouldn't take a lot of processing resource
  - To achieve high throughput, scheduler should minimize the number of preemptions (delaying a thread for another one)
  - Measuring a request response:
    - Turnaround time: time for response to arrive and complete
    - Response time: time for request to arrive and start producing output (slightly more useful)
    - Waiting time: time for response to arrive and start processing request (slightly better than turnaround; ideally 0)
  - Average waiting time is the average of the waiting times of all requests
  - In an interactive system, the user's perception is the most important measure of goodness
  - Degree of fairness: sharing the service equally for each request (unfairness is not necessarily bad)
- Scheduling Policies
  - Threads go through a cycle of running/waiting, so we classify them as a series of jobs (burst of activities)
  - First-come, first-served (FCFS)
    - Thread manager uses a FIFO queue; jobs are run sequentially
    - FCFS can favor long jobs over short ones, leading to an undesirable state
    - When I/O-bound threads get executed in sequence and each one waits, there are often periods where all the threads are waiting for input; this is the convoy effect
      - Never overlaps I/O computation, thus missing threads
  - Shortest-Job-First

        - Scheduler chooses to run the job that is shortest in expected running time
            - Prediction can be difficult
        - Total time taken is better than FCFS
        - Difficulty in determining amount of work a job has to do
            - Assume different cases; interactive jobs are usually short; computation-intensive jobs are usually long
            - Use Exponentially Weighted Moving Average (EWMA) to base prediction on past jobs
        - Starvation might occur; a long job might never be run
    - Round-Robin
        - Break down all long jobs into smaller jobs via preemptive scheduling
        - Scheduler maintains a queue of runnable jobs; it uses FCFS to some extent, but after some time, it moves onto the next process
        - Quantum: the fixed time value which causes an interrupt to call YIELD
    - Priority Scheduling
        - Jobs are assigned a priority number; dispatcher selects jobs with highest priority numbers
            - Rules are used to break ties
        - Predefined assignments can be used (user jobs have 0, system jobs have 1) or it can also be computed dynamically
        - Can be used to avoid starvation problems
        - Preemptive vs. non-preemptive
            - Preemptive: high-priority job might preempt a currently-running low-priority job
    - Scheduler interactions might actually cause highest priority threads to receive least processing time (priority inversion)
    - Real-Time Schedulers
        - Real-time constraints: certain applications require delivery of results before a deadline
            - Example: water valve that needs to open to prevent overflow
            - Use a hard real-time scheduler for disastrous systems (chemical/nuclear plants, etc.)
                - With the latter, it is hard to predict the time taken
                - Often, worst-case performance is assumed
            - Use a soft real-time scheduler for less disastrous systems (digital music system, etc.)
                - Attempts to meet all deadlines, but no guarantees
        - Earliest-deadline-first-scheduler (heuristic for avoiding missed deadlines)
            - Queues the jobs sorted by deadline
    - Rate monotonic scheduler: good algorithm for dynamically scheduling; give priority based on frequency
- Case study: scheduling the disk arm
    - Mechanical disk arm scheduling is important, as it creates an I/O bottleneck
        - Goal: optimize throughput
        - Better than FCFS is an algorithm that sorts requests by track number and process them, sorted
    - Algorithms
        - Shortest seek first
        - Elevator algorithm (good for buildings)
        - Disk controllers use a combination of both for maximizing throughput