Scheduling Lecture

Scheduling

- what is scheduling?
    - Goals
    - Mechanisms
- Scheduling on batch systems
- scheduling on interactive systems
- other kinds of scheduling
    - real-time scheduling

Why Scheduling processes?

- Busts of CPU usage alternate with periods of I/O wait
- some processes are CPU-bound: they don't gave many I/O requests
- other processes are I/O bound and make kernel requests

CPU bound processes: computing stuff
I/O bound processes: waits for input, sleeps a lot

1 second CPU time = 1 week

When are processes scheduled?

- at the time they enter the system
    - common in batch systems
    - they types of batch scheduling:
        - submission of a new jon causes the scheduler to run
        - scheduling only done when a job voluntarily gives up the CPU
- At relatively fixed intervals (clock interrupt)
    - necessary for interactive systems
    - may also be used for batch systems
    - scheduling algorithm at each interrupt, and picks the next process from the pill of "ready" processes

Scheduling Goals:

- All systems:
    - Fairness: give each process a fair share of the CPU
    - Enforcement: ensure that the stated policy is carried out

    - Balance: keep all parts of the system busy
- Batch systems:
    - throughput: maximize jobs per unit time (hour)
    - Turnaround time: minimize time users wait for jobs
    - CPU utilization: keep the CPU as busy as posssible
- Interactive Systems:
    - response time: respond quickly to users' request
    - Proportionality: meet users' expectations
- Real-Time Systems:
    - meet deadlines: missing deadlines is a system failure
    - Predictability: dame type of behavior for each time slice

Interactive vs batch scheduling:

- batch
    - First-Come-First-Served
    - Shortest Job First
    - Shortest Remaining Time First
    - Priority
- Interactive:
    - Round-Robin
    - Priority (preemptive)
    - Multi-Level Feedback Queue
    - Lottery Scheduling

First Come, First Served (FCFS):

- goal: do jobs in the order they arrive
    - fair in the same way a bank teller line is fair
- ProblemL long bond delay every job after them

Shortest job first:

- goal: do the shortest job first
- optimal for throughput
    - throughput: number of jobs accomplished
- jobs sorted in increasing order of execution time
- shortest remaining time first: preemptive form of SJF:
    - re-evaluate when a new jobs is submitted
- problem: how does the scheduler know how long a job will take
- starvations: a job with a high execution time will keep getting pushed behind the line

as faster jobs are introduced

Three-level Scheduling

- jobs held in input queue winter moved into memory
    - pick “complementary jobs” small and large, CPU and I/O intensive
    - jobs moved into memory when admitted
- CPU scheduler picks next job to run
- Memory scheduler picks some jobs from main memory and move them to disk if sufficient memory space
- When process is in memory then the process is able to run

Round_robin scheduling

- scheduling interactive process
    - give each process a fixed time slot (quantum)
    - rotate through “ready” processes
        - have to select through the processes
        - only one process per core can run
    - each process makes some progress
- What’s a good quantum?
    - too short: many process switches hurt efficiency
    - too long: poor response to interactive requests
    - typical length: 10- 100 ms
- strict rotation: round robin
- context switching: switching through processes, each one gets a quantum

Priority Scheduling:

- Assign a priority to each process
    - ready process with highest priority allowed to run
    - running process may be interrupted after its quantum expires
- priorities may be assigned dynamically
    - reduced when a process uses CPU time
    - increased when a process waits for I/O
- often, processes grouped into multiple queues based on priority, and run round-robin per queue
- starvation: one high priority process is always ready, and another process doesn’t get to run

Shortest process next

- run the process that will finish the soonest
    - in interactive systems, job completion time is unknown!
- guess at completion time based on previous runs
    - update estimate each timothy job is run
    - estimate is a combination of previous estimate and most recent run time
- not often used because round robin with priority works so well!

Lottery scheduling

- give processes “tickets” for CPU time
    - more tickets -> higher share of CPU
- each quantum, pick a ticket at random
    - if there are n tickets, pick a number from 1 to n
        - pseudo-random number is ok if its a good RNG
    - Process holding the ticket gets to run for a quantum
    - this can be implemented efficiently without “real” tickets
        - track range of tickets belonging to each process
- Over the long run, each process gets the CPU m/n of the time if the process has m of the n existing tickets
- tickets can be transferred
    - cooperating processes can exchange tickets
    - clients can transfer tickets to a server so it can have a higher priority
    - parent (shell) can transfer tickets a child process
- don’t keep track of ticket numbers
- doesn’t matter where the scheduler starts, everyone will still get to run
- random number generated: subtract the lottery ticket numbers until 0 or negative number and whatever process that lands on choose that process to run
- no starvation with this scheduling

Scheduling in BSD4

- quantum is 100ms : longest thats ok for interactive scheduling
- scheduler is based on mutli-level feedback queues
    - priority is based on two thinfs
        - resource requirements: blocked threads have higher priority when rescheduled
        - previous CPU usage: CPU …

Calculating priorityy in BSD4:

- thread priority is set by
    - priority = MIN + [estcpu/4] + 2xnice
        - values above MAX are set to MAX
        - MIN = 160, MAX = 223
        - nice is set by the user to manually over thread priority
        - cstcpu is an estimate of the number of “ready” processes in the CPU when the calculation is made
            - has a bit of “memory” so it doesn’t change too quickly
            - estcpu is update each clock tick
        - higher numbers indicate lower priority: threads with lowest priority values are scheduled first
- thread priority is set every 40ms
- scheduling is more complex for multiprocessors…

Scheduling user-level threads:

- kernel picks a process to run next
- if you have user-level threads, the kernel chooses the process to run
- Thread table:
    - non preemptive scheduling

Scheduling kernel level threads:

- kernel schedules each thread
    - no restrictions on ordering