CPU Scheduling

CS 537 - Introduction to Operating Systems

Objectives

- High throughput
- Low response time
- Good utilization of system resources
  - low waiting time in system
- Avoid starvation
- Fairness
- Be efficient in scheduling - it's done often

Bursts

- CPU burst
  - interval of time a process would run before blocking if not preempted
- I/O burst
  - time process spends doing a single I/O operation
- Burst times do not include waiting times

CPU and I/O Bounded Processes

- CPU bound process
  - process spends most of its time using processor
  - very long CPU bursts
  - compiler, simulator, scientific application
- I/O bound process
  - process spends most of its time using I/O
  - very short CPU bursts
  - word processors, database applications
- Processes usually either CPU or I/O bound
  - behavior may change over time (drastically)

Schedulers

- Short-term scheduler
  - pick best job from those currently in memory
  - pick a currently active process
  - this is what we will be concerned with
- Long-term scheduler
  - pick best jobs to place in memory
  - batch system (CONDOR)
  - need to pick jobs that will run well together
    - want a good mix of CPU bound and I/O bound jobs

Invoking Scheduler

- 4 situations that could lead to scheduling a new process
  1. running process blocks
  2. running process terminates
  3. running process switches to ready state
     - timer interrupt
  4. blocked process switches to ready state
     - finish I/O operation
- if first 2 only, non-preemptive scheduler
- if 3 or 4, preemptive scheduler
- Notice that all 4 do involve interrupts
  - either software or hardware
  - no interrupts, no context switches

Analyzing Schedulers

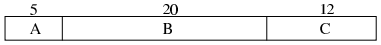
- Many possible parameters to measure
  - throughput, avg waiting time, utilization, etc.
- Must pick important parameters
  - system dependant
  - example
    - maximize throughput with all waiting times < 1 sec
- Various methods to analyze algorithm
  - deteministic, queueing theory, simulation
  - will examine these at end of lecture

Analysis

- Consider following system

Process	Burst Time
A	5
B	20
C	12

- Gantt chart



- Calculations
  - avg waiting time = Σ(start times) / # of procs
  - throughput = # of finished jobs / time period
  - utilization = time busy / total time

Scheduling Policies

- First-Come, First Serve (FCFS)
- Shortest Job First (SJF)
  - non-preemptive and preemptive
- Priority
- Round-Robin
- Multi-Level Feedback Queue

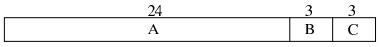
FCFS

- Processes get processor in order they arrive to ready queue
- Very easy to manage
  - new job goes to tail of queue
  - next job to run is removed from the head
- Poor policy for CPU scheduling
  - very sensitive to the order in which jobs arrive

FCFS

- Example

Process	Burst
A	24
B	3
C	3



W = (0 + 24 + 27) / 3 = 17 ms

now switch processes A and C

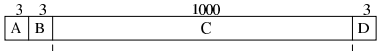
W = (0 + 3 + 6) / 3 = 3 ms

Convoy Effect

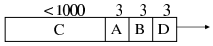
- I/O bound jobs have short CPU bursts
- CPU bound jobs have long CPU bursts
- Once CPU bound job does go to I/O, all of the I/O bound jobs will rush through CPU and group behind the CPU bound job
- Leads to poor utilization
- Leads to poor response time

Convoy Effect

Process	Burst
A	3
B	3
C	1000
D	3



A and B finish I/O and get in line behind D



- Disk is now Idle as all jobs wait on CPU
- Process C finishes and goes to I/O
- D, A, and B all finish quickly and go to I/O
- CPU sits idle while all jobs at Disk

SJF non-Preemptive

- Schedule the job with the shortest burst
  - better titled Shortest Burst First
- Provably the lowest response time (highest throughput)

$a < b < c < d < \ldots$
$R = [0 + a + (a+b) + (a+b+c) + \ldots] / n$
now switch any 2, for example b and c
$R' = [0 + a + (a+c) + (a+c+b) + \ldots] / n$
$(a+c) > (a+b)$ and all other terms are the same
$R < R'$

SJF non-Preemptive

- Requires knowledge of future
  - IMPOSSIBLE!
- So why study this
  - if we can analyze after the fact, can compare to ideal
  - fortunately, consecutive bursts tend to be similar
    - if $B_n = X$, then $B_{n+1} \approx X$
    - this allows us to predict the future

SJF non-Preemptive

- How do we do prediction of future?
  - could just use the time of the last burst
    - Shortest Last Burst First
    - anomalous burst will give bad prediction
  - use an exponential average
    - consider all past bursts
    - smooth out anomalous bursts
    - give less weight to bursts that happened longer ago

Exponential Averaging

- Equation:

  $\tau_{n+1} = \alpha t_n + (1- \alpha) \tau_n \qquad 0 \le \alpha \le 1 \qquad$ eqt. 1

  t_n = time of burst just finished

  τ_n = predicted time of burst just finished

  τ_{n+1} = predicted time of the next burst

  α = weight to give past events

  if $\alpha = 1$, just consider the last burst

  if $\alpha = 0$, just use a default prediction
- Let's expand out the above function

  $\tau_n = \alpha t_{n-1} + (1- \alpha) \tau_{n-1} \qquad$ eqt. 2

  combine equations 1 and 2 to get:

  $\tau_{n+1} = \alpha t_n + \ldots + (1- \alpha)^j \alpha t_{n-j} + \ldots + (1- \alpha)^{n+1} \tau_0$

  τ_0 = arbitrary value (perhaps a system wide average burst time)
- For scheduling, pick the job with the lowest τ value

SJF Preemptive

- Identical to SJF non-Preemptive *except*:
  - if new job has shorter burst than current job has left to run, stop the current job and run the new job
- Often called Shortest Remaining Time First

SJF Algorithm Problems

- Starvation
  - long burst never gets to run because lots of short jobs in the system
- Fairness
  - long jobs get to run very infrequently because of lots of short jobs in the system

Aging

- Common solution to starvation / fairness problem
  - when job enters queue, give it a value of 0
  - after every scheduling decision it loses, increase its value by 1
  - if the value become greater than some threshold, it becomes the next job scheduled no matter what
  - if multiple jobs above threshold, pick the one with the highest value

Priority Scheduling

- Each job has a priority associated with it
- Run the job with the highest priority
  - ties can be broken arbitrarily (FCFS, perhaps)
- How do priorities get set?
  - externally
    - programmer
    - administrator
  - internally
    - OS makes decision
    - avg size of burst, memory requirements, etc.
- Starvation and Fairness are still issues
  - use priority aging (increase priority over time)

Round-Robin

- Give each burst a set time to run
- If burst not finished after time, preempt and start the next job (head of the ready queue)
  - preempted process goes to back of ready queue
- If burst does finish, start the next job at the head of the ready queue
- New jobs go to the back of the ready queue
- Similar to FCFS except time limits on running
- Time a burst gets to run before preemption is called a *Quantum*

Round-Robin

- Need hardware timer interrupts
- Very fair policy
  - everyone gets an equal shot at processor
- Fairly simple to implement
- If quantum is large enough to let most short bursts finish, short jobs get through quickly
- Must consider overhead of switching processes
  - if quantum is too small, overhead hurts performance
  - if quantum is too large, RR becomes like FCFS

Round-Robin

Process	Burst
A	6
B	10
C	7

quantum = 3

3	3	3	3	3	3	3	1	1
A	B	C	A	B	C	B	C	B

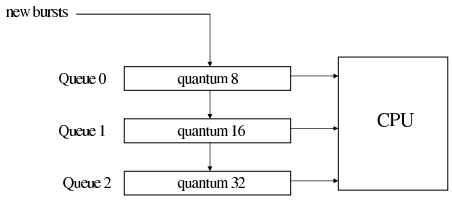
Multilevel Queue Scheduling

- Maintain multiple queues
- Each queue can have a different scheduling policy
- Or maybe same policy but different parameters
  - All are Round-Robin with different quantums

Multilevel Feedback Queue

- Multiple queues for jobs depending on how long they have been running (current burst)
- All jobs enter at queue 0
  - these jobs run for some quantum, *n*
- If jobs do not complete in *n*, they move to queue 1
  - these jobs run for some quantum, *m* (*m* > *n*)
- If these jobs do not complete in time, they are moved to yet another queue
- A job can only be selected to run from a queue if all queues above it are empty
- Jobs higher up, preempt jobs lower down

Multilevel Feedback Queue



Multilevel Feedback Queue

- Longer a job is in the system, the longer it can be expected to stay
- Let long jobs run only when there are no short jobs around
- Different levels of long jobs
- Can using aging to prevent starvation
  - if a job sits in a low level queue for too long, move it up one or more levels

Multilevel Feedback Queue

- Important issues in a multilevel queue
  1. number of queues
  2. scheduling algorithm at each queue
  3. method for upgrading a job to higher level
  4. method used for demoting a job to lower level
  5. which queue does a process enter on arrival