see https://www.cs.rutgers.edu/~pxk/416/notes/07-scheduling.html

RR metrics

what is the overhead associated with different time slice lengths?

Depends on time it takes to switch contexts!

For instance say

t_s => t_s length

C => context switch time

$$\text{context switch overhead} = \frac{C}{t_s + C}$$

$t_s \uparrow$ increases efficiency but decreases average response time (bad for interactive processes, like games)

$t_s \downarrow$ great for interactivity, but spend more time context switching

ex. $t_s = 100ms$ or $t_s = 10ms$ $C = 5ms$

Proc #	$t_s = 100ms$ delay	$t_s = 10ms$ delay
0	0	0
1	105	15
2	210	30
3	315	45
4	420	60 ← ready
	⋮	

process 4 takes ½ sec before it runs too slow for interactive ↑ gets worse as more processes added

Overhead

$t_s = 100$ overhead = $\frac{5}{100+5} = 4.8\%$ overhead

$t_s = 10$ overhead = $\frac{5}{10+5} = 33.3\%$ overhead

you pay a price for responsive

Priority

- RR assumes all processes the same
- Not the case, usually hierarchy
  - Priority
    - low — cpu intensive, non-interactive
    - med — interactive
    - high — critical system processes

system picks highest priority process to run, or preemptive, system switches to higher priority process when they appear.

Priorities
- internal - assigned by system
- external - assigned by admin

can coexist:
- static - sticks for lifetime
- dynamic - system modifies (scheduler)

Priority is just a number — higher better? windows; lower better? linux

Advantage

- relative importance of each process precisely defined

Disadvantage

- high priority processes can hog CPU, starve low priority ones.

Starvation

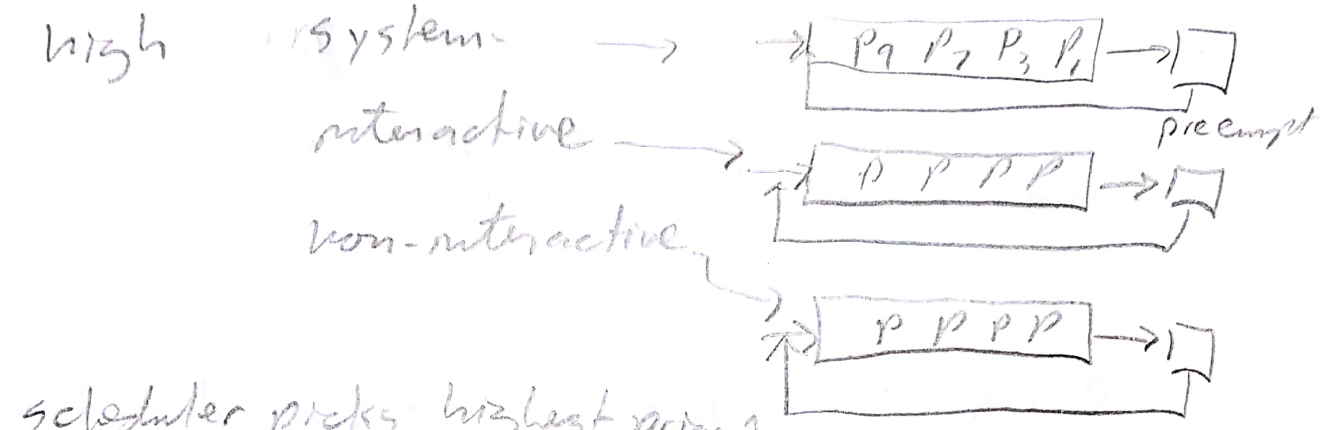
- dynamic priority - at end of each ts, system can diminish process priority, eventually lower priority processes run.

(ageing) or keep track of low priority & gradually increase their priority until they run. Then reset & restart

Multilevel Queues (Priority)

Group processes into classes (real time, kernel, interactive, background) Diff schedulers for each class. Used: Win, Linux, OSX

priority



each Q can have diff scheduling algorithm

- usually round robin, but if time critical can use FCFS (no preemption) gets all time it needs no interrupt

- scheduler can also choose diff time slice per Q., interactive has small time slice to ensure snappy response, non-interactive has longer ts. Better utilization.
  - Don't get to run as often but when they do they run longer

How to choose priority?
what if CPU bound gets high priority? not good for process or throughput (it slows interactive)

Multilevel Feedback Queues

- scheduler adjust priority of process moves among classes.

- goal place process in queue appropriate to CPU burst behavior.
  I/O intensive, end up on High priority ques
  CPU intensive, end up on lower " "

2 Basic rules

1. New process placed on highest priority queue
2. If a process does not finish ts. (it blocks) it will stay on current queue, otherwise moves to next lower.

- so long CPU burst, use time slice, denoted to queue that lets it run longer ts's
- highly interactive, stays high priority

Ageing + Starving

- If lots of interactive processes or lots of new processes, low level queue processes starve
- Or start a game, CPU intensive while initializing, then interactive, ↑ cause donation to lesser priority queue

Solution

- periodically increase priority of process so it eventually runs!
- CPU intensive, quickly trickls back down, while Game above, makes it away to interactive queue + stays there!