

see <https://www.cs.rutgers.edu/~pvxk/416/notes/07-scheduling.htm> ①

RR metrics

what is the overhead associated with different time slice lengths?

Depends on time it takes to switch contexts!

For instance say

$t_s \Rightarrow t_s$ length

$C \Rightarrow$ 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. $t_s = 100\text{ms}$ or $t_s = 10\text{ms}$ $C = 5\text{ms}$

Proc #	$t_s = 100\text{ms}$ delay	$t_s = 10\text{ms}$ delay
0	0	0
1	105	15
2	210	30
3	315	45
4	420	60

process 4
takes $\approx \frac{1}{2}$ sec
before it runs
too slow for
interactive &
gets worse
as processes added

60 \leftarrow zippy

overhead

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

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

you pay a price for responsive

Priority

- RR assumes all processes the same

- Not the case, usually hierarchy

<u>Priority</u>	
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

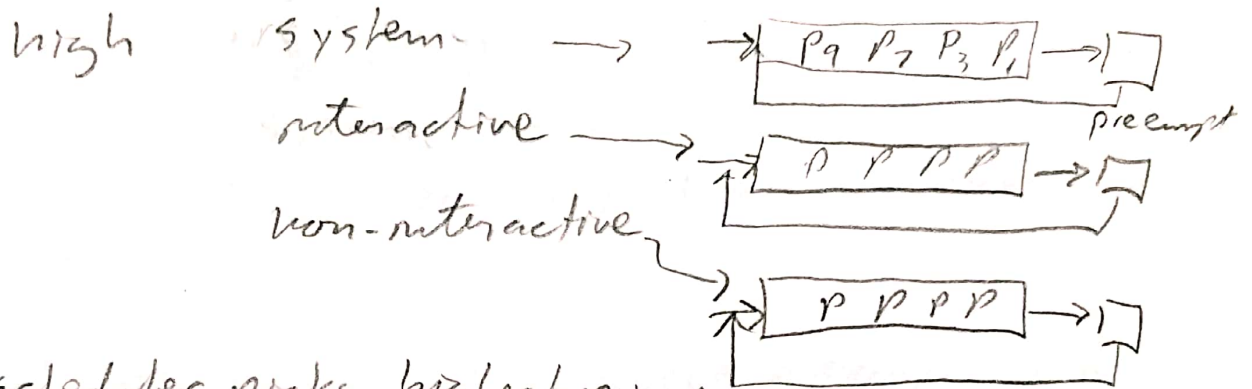
- dynamic priority - at end of each t_s , system can diminish process priority, eventually lower priority processes run.

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

Multilevel Queues (Priority)

④

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



scheduler picks highest priority queue with job

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

who 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 queue
 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, demoted to queue that lets it run longer ts's
 - highly interactive, stays high priority

6

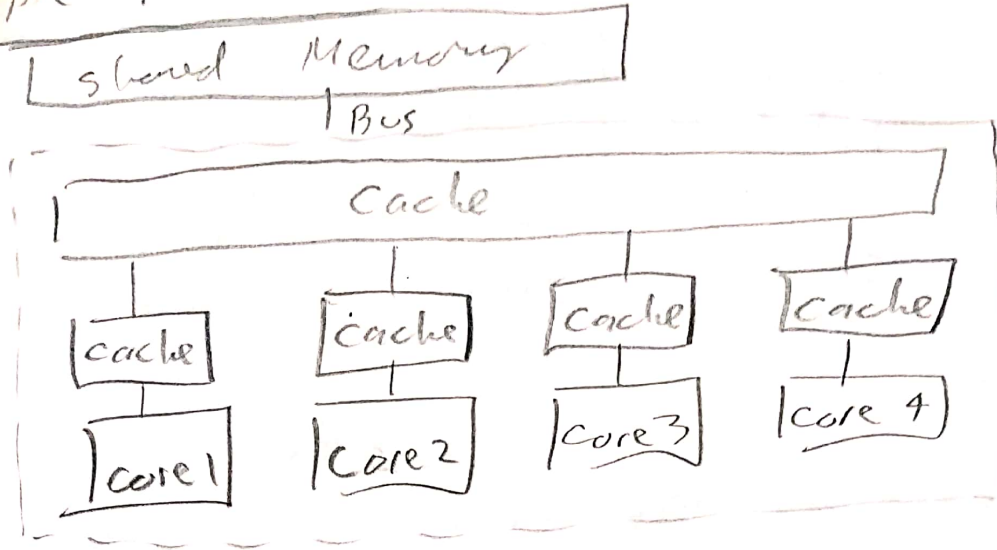
Starving + Starving

- 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, [↑] causes demotion to lesser priority queue

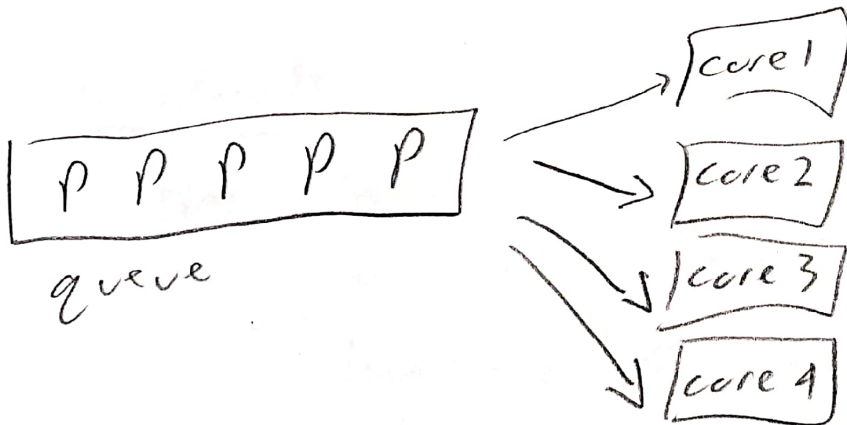
Solution

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

Multiple Processors



How about readyQ that feeds them all



Advantage

good allocation (RR to each core)

Fair

Disadvantages

contention for global ReadyQ lock
poor core affinity

Per CPU Q



(8)

Advantages

easy to implement

scales well (no ready Q contention)

better thread locality (if process stays

on a core then its prev. cache info may stick around)

Disadvantage

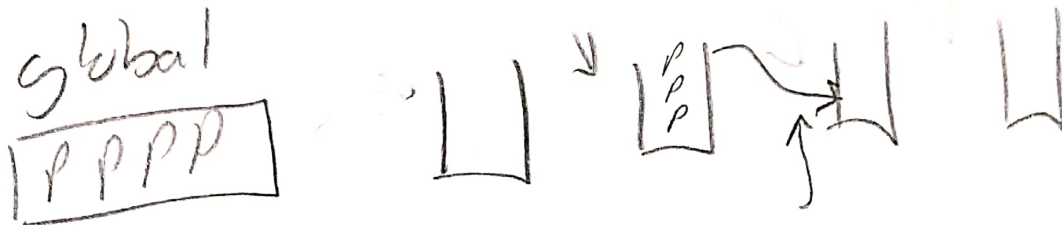
imbalance

what if Core 0 has lots & Core 1 has none? Or what if diff kinds of processes?

possible lower CPU utilization

Hybrid use both (global n per cpu)

⑨



Balance jobs accross Queues