CPU Scheduling, Part Deux

44-550: Operating Systems

Previously

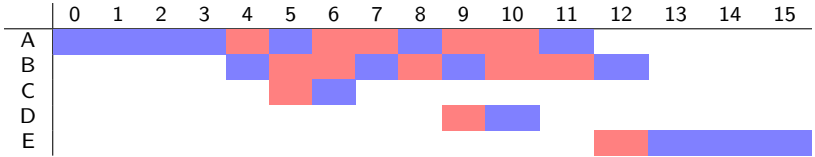
- FCFS
  - Fair
  - Simple to implement
  - Far from optimal
- SJF
  - Not Fair
  - Requires ability to estimate CPU burst
  - Could push longer running jobs to never run
- What happens if we try a more sophisticated approach?

Round Robin (Yum!)

- Each process is given quanta (equal time slices)
- Takes turns of one quantum each
- Let's see what happens with the original example

Process	Burst Time(t)	Arrival Time
A	7	0
B	4	4
C	1	5
D	1	9
E	3	12

RR Example



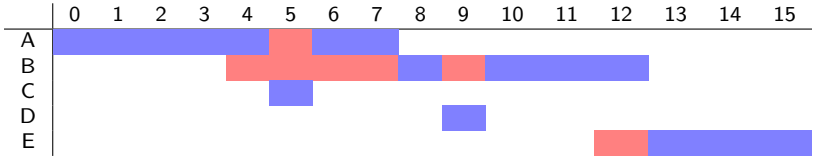
Process	Required Time (T)	Wait Time	Total Time
A	7	5	12
B	4	6	10
C	1	1	2
D	1	1	2
E	3	2	5
Average		2.6	5.8
		Wait Time	Turnaround

RR Properties

- Benefits short burst processes greatly
- Can hurt the runtime of processes with long bursts
  - Doesn't excessively penalize them; they will not starve
- Preemptive

Shortest Remaining Time (SRT)

- When a process arrives in the ready queue with an expected CPU burst time smaller than the remaining time of the running processes, the new one preempts the running process



Process	Required Time (T)	Wait Time	Total Time
A	7	1	8
B	4	4	8
C	1	0	1
D	1	0	1
E	3	1	4
Average		1.4 Wait Time	4.6 Turnaround

SRT Properties

- Long processes can starve
- Short processes get serviced quickly
- Works well in many cases
- Works poorly if estimation of remaining time is bad

A Real Life Implementation: CFQ

- Default scheduler for the Linux kernel since the 2.6.23 release
- Not based on run queues
  - A red black tree implements a timeline of future task execution
- Uses nanosecond granularity
  - We don't need no stinkin' time slices
- Does not need heuristics to determine the interactivity of a process
  - how often it hits up I/O

Red Black Trees

- Built on top of BST (for our example)
- A node is either red or black
- All leaves are black, and have the value NULL/NIL
- If a node is red, all children are black
- Ever path from a node to any descendant NIL nodes has the same number of black nodes in the path
  - The uniform number of black nodes in the path from root to leaves is called the *black height* of the tree

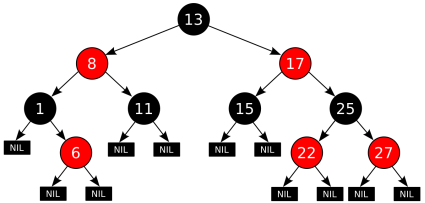


Figure: From https://en.wikipedia.org/wiki/Red%E2%80%93black_tree

The Algorithm (as per wikipedia)

Processes are inserted into the tree based on their current spent execution time

- The left most node of the scheduling tree is chosen and sent for execution
  - it has the lowest spent execution time
- If the process completes, removed from the system and tree
- If the process reaches its maximum execution time or otherwise stopped, reinserted into the scheduling tree based on its new spent execution time.
- The new left most node will then be selected from the tree. Repeat

CFQ

- Not Completely Fair (as the name implies)
- Processes that sleep a lot are not penalized (as their spent execution time remains low), and will be given priority when they need the CPU

Dynamic Scheduling Policies

- Processes can alternate between I/O and CPU bursts
  - We could give processes that need I/O followed by CPU higher priority to keep the I/O busy (bottleneck)
- How do we decide what processes to give high priority?
- Dynamic Scheduling adjusts the priority as the process runs
  - Predicts what will happen next (branch prediction is hard)
- We want to keep all the resources busy (I/O and CPU)

Possible Dynamic Scheduling Heuristic

- Allocate CPU to highest priority process
- When a process is selected to be executed, give it a time slice
- If a process makes an I/O request before it's timeslice is up, raise it's priority (assume it will make an I/O request shortly after a small CPU boost)
- If the process reaches the end of its timeslice without making an I/O request, lower the priority

Multiprocessor Scheduling

- So far, assumed only one processor
- Most computers this day have multiple cores (or multiple processors)
- Many different ways to accomplish scheduling across the cores
  - Have a scheduler data structure per core
  - Have one scheduler data structure, and distribute tasks from it to each core
  - ...