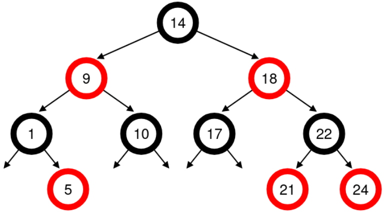


CSL 301

OPERATING SYSTEMS

Lecture 6

Proportional-Share Scheduling

Instructor
Dr. Dhiman Saha

Proportional-Share Scheduling

- The core idea is to guarantee that each job receives a certain percentage of CPU time.
- This contrasts with schedulers that optimize for turnaround or response time.
- Two early examples are Lottery Scheduling and Stride Scheduling.

Lottery Scheduling

- Assigns **tickets** to each process to represent its share of a resource.
- Periodically holds a lottery to select the next process to run.
- Probabilistic, not deterministic, so fairness is not guaranteed over short time intervals.
- Implementation relatively simple
- Just a good random number generator
- A data structure to track the processes of the system

Open Problem

How To Assign Tickets?

Lottery Scheduling - Example

- Imagine two processes, A and B
- A has 75 tickets while B has only 25
- Assuming A holds tickets 0 through 74
- B has tickets 75 through 99
- The winning ticket simply determines whether A or B runs
- The scheduler[1] then loads the state of that winning process and runs it.

Here is an example output of a lottery scheduler's winning tickets:

```
63 85 70 39 76 17 29 41 36 39 10 99 68 83 63 62 43  0 49 12
```

Here is the resulting schedule:

```
A       A   A       A   A   A   A   A   A       A       A   A   A   A   A   A
    B           B                           B       B
```

[1] Scheduler must know the total number of tickets.

Stride Scheduling: Example

- A deterministic fair-share scheduler.
- Each process has a **stride**, inversely proportional to its tickets.
- The scheduler picks the process with the lowest **pass** value.
- Example: 3 processes A, B, C with 100, 50, and 250 tickets. A large number (e.g., 10000) is divided by the tickets to get the stride.
- Stride A = 100, Stride B = 200, Stride C = 40.
- C runs 5 times, A twice, and B once in a cycle.

Stride Scheduling: Trace

Pass(A) (stride=100)	Pass(B) (stride=200)	Pass(C) (stride=40)	Who Runs?
0	0	0	A
100	0	0	B
100	200	0	C
100	200	40	C
100	200	80	C
100	200	120	A
200	200	120	C
200	200	160	C
200	200	200	...

- Better proportional share scheduling that lottery
- What the catch?
- What happens if a new job enters in the middle of our stride scheduling?

Completely Fair Scheduler (CFS)

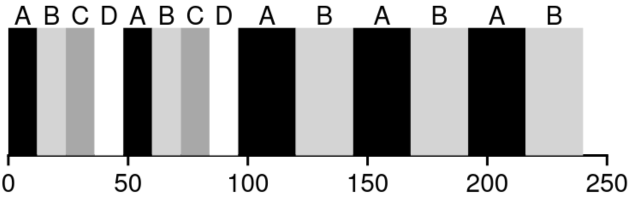
- The default scheduler in Linux.
- Implements fair-share scheduling efficiently and scalably.
- Aims to give each process a fair share of the processor.

Core Concept: Virtual Runtime (vruntime)

- CFS tries to divide the CPU evenly among all competing processes.
- It uses a simple counting-based technique called **virtual runtime (vruntime)**.
- As a process runs, its vruntime accumulates.
- CFS always picks the process with the **lowest vruntime** to run next.

Scheduling Decisions: Example

- **sched_latency**: If set to 48ms and there are 4 processes, each gets a 12ms time slice.
- **min_granularity**: If 'sched_latency' / 'nr_running' is less than 'min_granularity' (e.g. 6ms), the time slice is set to 'min_granularity'.
- This prevents excessive context switching overhead.

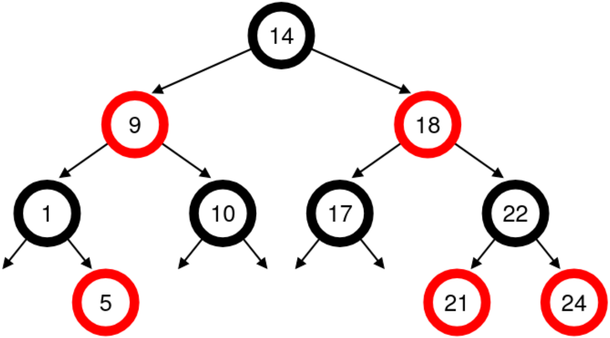


Weighting (Niceness): Example

- A process with a ‘nice‘ value of -5 has a weight of 3121.
- A process with a ‘nice‘ value of 0 has a weight of 1024.
- The ‘vruntime‘ is scaled by the weight. For the higher priority process, ‘vruntime‘ accumulates at about 1/3 the rate of the default-priority process.
- This means it gets to run roughly 3 times as long.

Efficient Process Selection: Red-Black Trees

- CFS uses a **red-black tree** to store runnable processes, ordered by vruntime.
- This allows for efficient (logarithmic time) selection of the next process to run.



Dealing with Sleeping Processes

- A process that wakes up after sleeping for a long time could have a very low vruntime and monopolize the CPU.
- To prevent this, when a process wakes up, CFS sets its vruntime to the minimum vruntime currently in the red-black tree.
- This prevents starvation of other processes but can be unfair to processes that sleep for short periods.

Comparison of Schedulers

Scheduler	Pros	Cons
Lottery	Simple, no global state	Not deterministic
Stride	Deterministic	Global state is complex
CFS	Efficient, fair, scalable	Complex, potential for starvation

Summary

- CFS is a proportional-share scheduler that aims for fairness.
- It uses **vruntime** to track how long each process has run.
- It selects the process with the lowest vruntime to run next.
- Process priorities are handled through **niceness** and weights.
- A **red-black tree** is used for efficient process selection.