Fair Scheduling

How to be fair to all processes?

By Matthew Tancreti
For COM S 352
Iowa State University

Fair Scheduling

We have seen how schedulers can optimize turnaround and response time

But we have also seen issues such as starvation that result in some processes getting little run time, even while the overall system has good turnaround and response time

How to make scheduling fair?

Lottery Scheduler

Each job assigned numbered “tickets”

Every time slice, scheduler randomly picks a winning ticket and the job runs for that time slice

Over time, jobs run time is proportional to percent of tickets held

How to Assign Tickets?

In effect, the lottery scheduler is a kind of priority scheduler

- More tickets means higher priority
- But, fairer than priority, low ticket jobs can't be completely starved

For example, how can we create the effect of SJF?

- Assign number of tickets inversely proportional to each job's runtime

But there are differences

- Pro: No job can be starved, everyone gets their fair share of runtime
- Con: A short job can be unlucky and have higher response time

Example Implementation

```
1  // counter: used to track if we've found the winner yet
2  int counter = 0;
3
4  // winner: use some call to a random number generator to
5  //         get a value, between 0 and the total # of tickets
6  int winner = getrandom(0, totaltickets);
7
8  // current: use this to walk through the list of jobs
9  node_t *current = head;
10 while (current) {
11     counter = counter + current->tickets;
12     if (counter > winner)
13         break; // found the winner
14     current = current->next;
15 }
16 // 'current' is the winner: schedule it...
```

Stride Scheduling

Lottery has good fairness over the long run, but randomness can result in suboptimal choses over a short time

Stride scheduling is deterministic (not random) but has same fairness

For each job set its **stride** as a large number (e.g., 10000) divided by its number of tickets

After every time a job runs increment it's **pass** counter by its stride

Scheduler always picks the job with the lowest pass counter to run next

Linux Completely Fair Scheduler (CFS)

CFS gives a proportion of CPU time to each process, there are two questions

When to preempt the currently running process?

- Every process assigned a time slice within one **sched_latency**
  - If equal priority, each process gets a 1/N slice of sched_latency
  - Therefore, every process should run at least once in sched_latency

Which process to run next?

- The process with the smallest **virtual runtime (vruntime)**

Being Nice

Nice value set by user to indicate "priority", weights the 1/N time slice a process gets before preemption

$time\_slice_k = weight_k$ / (sum of all processes weights) * sched_latency

- Lower nice value will receive larger time slice
- Higher nice value will receive smaller time slice
- time_slice of all processes still add up to sched_latency

Scheduler accumulates **virtual run time (vruntime)**, which is a weighted version of the real **runtime** of each process

$vruntime_i = vruntime_i$ + (weight ratio based on nice value) * $runtime_i$

- if nice value = 0: weight ratio = 1 and vruntime = actual run time
- if nice value < 0: weight ratio < 1 and vruntime < actual run time
- if nice value > 0: weight ratio > 1 and vruntime > actual run time

When current running process uses its $time\_slice_k$, or leaves for some other reason (e.g., blocked), scheduler picks process with lowest vruntime to run next

When blocked process goes to ready queue, its vruntime is set to smallest vruntime of currently running or runnable processes

Efficiency

Scheduler gets called often, efficiency matters

CFS uses red-black tree (self-balancing binary search tree) to quickly find process with lowest vruntime