ITP 30002 Operating System

Proportional Share Scheduler

OSTEP Chapter 9

Shin Hong

Lottery Scheduler

- Proportional-share schedulers aim to guarantee that a process is given a certain portion of the CPU time

- Lottery scheduling: hold a lottery to determine a process for the next turn
  - a process holds **tickets** which represent the resource share that a process should receive

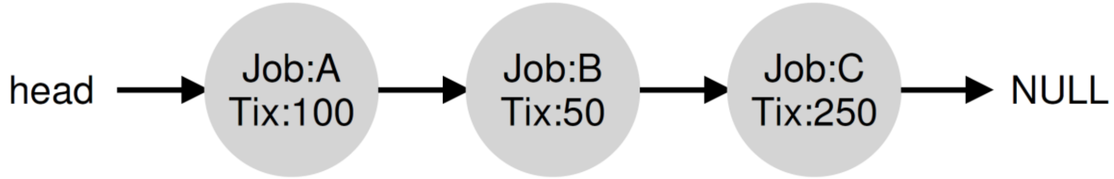
  - for each scheduling decision:
    - the scheduler counts the total number of tickets held by the ready processes
    - pick a winning ticket from the whole tickets
    - the scheduling goes to the process that owns the ticket

  - example: Process-A with 75 tickets (0 to 74), Process-B with 25 tickets (75 to 99)

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

Lottery Scheduling - Good Points

- The algorithm is simple and does not require much computation



- By managing the total number of tickets, the system can easily and reliably control the scheduling priorities of different processes
  - mechanisms
    - ticket currency
    - ticket transfer
    - ticket inflation

- The scheduler shows good fairness in scheduling processes

Stride Scheduling

- A deterministic fair-share scheduler

- A process is given with a **stride** value which is inverse in proportion to the number of tickets
  - the shorter a slide, the more time slices (steps) the process have in a time unit

- A process has a **pass** value which increases by slide when it is dispatched

- The scheduler selects a ready process with the smallest pass value

```
curr = remove_min(queue);   // pick client with min pass
schedule(curr);             // run for quantum
curr->pass += curr->stride; // update pass using stride
insert(queue, curr);        // return curr to queue
```

Example

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	...

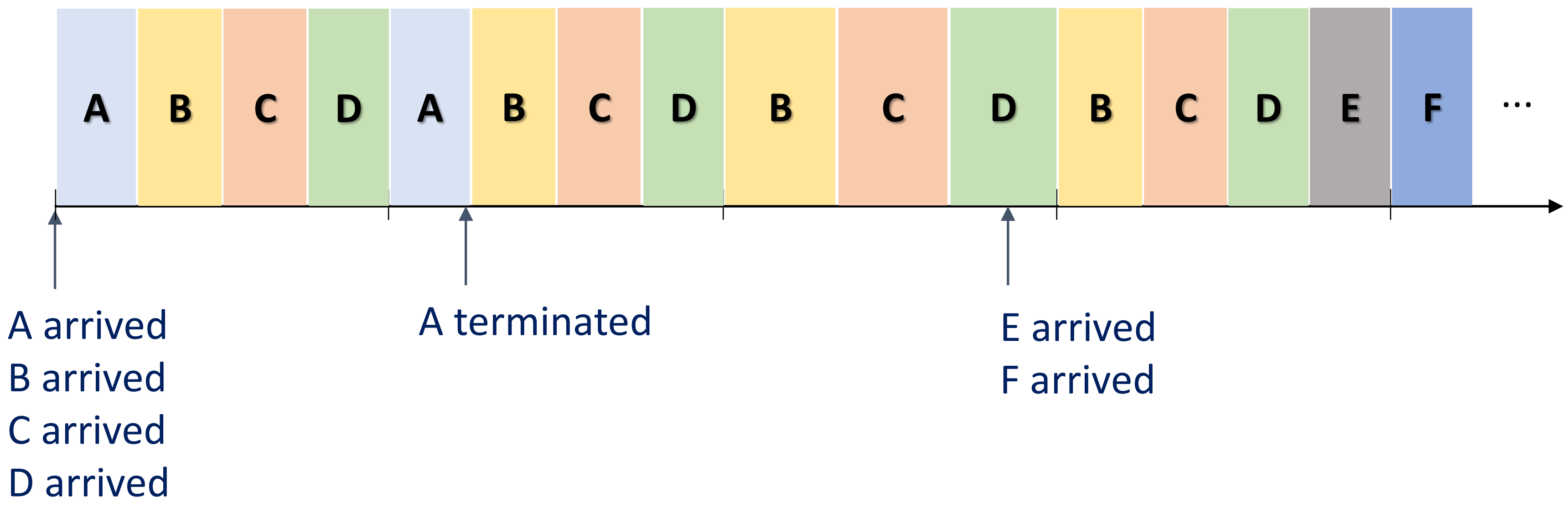
The Linux Completely Fair Scheduler (CFS)

- The cost of running scheduler is a significant factor of system efficiency
  - scheduling consumes around 5% of overall CPU time of Google datacenter

- CFS approach
  - reduce context-switching
    - divide a period of CPU time (`sched_latency`) evenly among all ready processes while keeping a time-slice from being too fine-grainted (`min_gradularity`)
    - enable users to give time-slice weighting to control process priority
  - share CPU time fairly
    - use a periodic timer interrupt to identify the expiration of a current time slice
    - count virtual runtime to trace the accumulated CPU time of each process
    - select the runnable process of the minimum virtual runtime

Example

- sched_latency: 48 ms
- min_granularity: 12 ms



Process Weighting

- The nice level in Unix represents the priority of a process
  - the lower a nice value, the higher priority is.
- The time slice of a process is proportional to its weight

$$\texttt{time\_slice}_k = \frac{\texttt{weight}_k}{\sum_{i=0}^{n-1} \texttt{weight}_i} \cdot \texttt{sched\_latency}$$

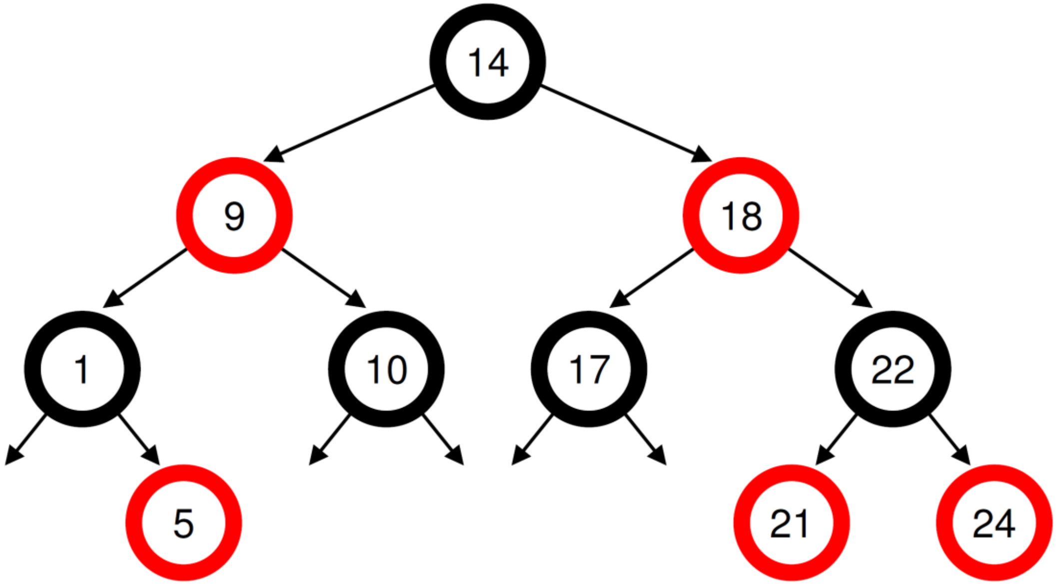
```
static const int prio_to_weight[40] = {
    /* -20 */ 88761, 71755, 56483, 46273, 36291,
    /* -15 */ 29154, 23254, 18705, 14949, 11916,
    /* -10 */  9548,  7620,  6100,  4904,  3906,
    /*  -5 */  3121,  2501,  1991,  1586,  1277,
    /*   0 */  1024,   820,   655,   526,   423,
    /*   5 */   335,   272,   215,   172,   137,
    /*  10 */   110,    87,    70,    56,    45,
    /*  15 */    36,    29,    23,    18,    15,
```

- accumulated CPU runtime is measured with respect to process weighting scheme
  - otherwise, a high priority process can starve

$$\texttt{vruntime}_i = \texttt{vruntime}_i + \frac{\texttt{weight}_0}{\texttt{weight}_i} \cdot \texttt{runtime}_i$$

Process Management

- A ready queue is formed as a red-black tree having vruntime as a key
  - balanced binary tree
  - log(n) for search, insertion, removal



- When a process is awken from a sleep, the scheduler sets its vruntime as the minimum vruntime found in the ready queue to avoid monopoly