All about Cdac Hyderabad - DESD

Thursday, 5 January 2017

How Shortest-Job-First (SJF) scheduler works

Shortest-Job-First (SJF)

- another name is *Shortest Process Next algorithm*
- a better name may be *shortest next-CPU-burst first*
- assumes we know the length of the next CPU burst of all ready processes
- the length of a cpu burst is the length of time a process would continue executing if given the processor and not preempted
- SJF estimates the length of the next burst based on the lengths of recent cpu bursts
- starts with a default expected burst length for a new process
- suppose that time intervals are numbered 1 for first cpu burst, 2 for second cpu burst, etc.
- default length is e(1), the expected length of time for the first cpu burst
- unlike other scheduling algorithms, this algorithm assumes that information about a process's burst length is stored between the times when it is ready.
- in keeping with the need for efficiency, only a small amount of info is stored and only a simple calculation is performed
- we can weight the previous expectation (representing all previous bursts) and the most recent burst with any two weights that add up to 1, e.g., say 0.5 and 0.5, or 0.9 for previous expectations and 0.1 for actual time for most recent cpu burst
- the expected burst length is calculated as follows:

```
a(t) = actual amount of time required during cpu burst t

e(t) = amount of time that was expected for cpu burst t

e(t+1) = expected time during the next cpu burst


then... e(t+1) =  0.5 * e(t) + 0.5 * a(t)
```

- **For example:** Suppose a process p is given a default expected burst length of 5 time units. When it is run, the actually burst lengths are 10,10,10,1,1,1 (although this information is not known in advance to any algorithm). The prediction of burst times for this process works as follows.

```
Let e(1) = 5, as a default value.

When process p runs, its first burst actually runs 10 time
units, so, a(1) = 10.

   e(2) = 0.5 * e(1) * 0.5 + a(1) = 0.5 * 5 + 0.5 * 10 = 7.5

This is the prediction for the second cpu burst

The actual second cpu burst is 10. So the prediction for the
third cpu burst is:

   e(3) = 0.5 * e(2) * 0.5 + a(2) = 0.5 * 7.5 + 0.5 * 10 = 8.75

   e(4) = 0.5 * e(3) * 0.5 + a(3) = 0.5 * 8.75 + 0.5 * 10 = 9.38,
```

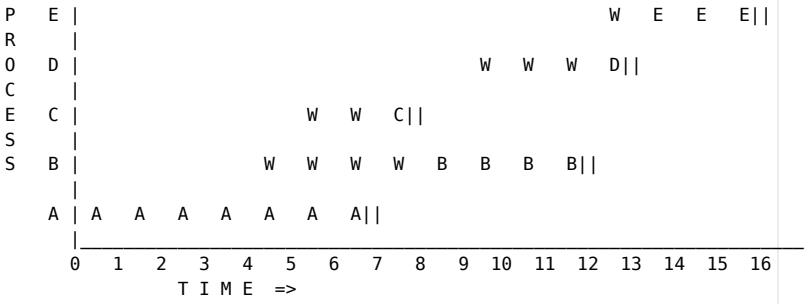
```
after rounding to two decimal places.

So, we predict that the next burst will be close to 10  (9.38)
because we recent bursts have been of length 10.

At this point, it happens that the process starts having shorter
bursts, with a(4) = 1, so the algorithm gradually adjusts its estimated
cpu burst (prediction)

   e(5) = 0.5 * e(4) * 0.5 + a(4) = 0.5 * 9.38 + 0.5 * 1 = 5.19

   e(6) = 0.5 * e(5) * 0.5 + a(5) = 0.5 * 5.19 + 0.5 * 1 = 3.10

   e(7) = 0.5 * e(6) * 0.5 + a(6) = 0.5 * 3.10 + 0.5 * 1 = 2.05

Once again, the algorithm has gradually adjusted to the process's
recent burst lengths.

If the bursts lengths continued to be 1, the estimates would
continue to adjust until, by rounding, they reached 1.00.

Each separate process will have a separate prediction of
the length of its next burst, i.e., e(b+1).
```

- Now to implement the algorithm...
  - choose the process expected to take the least time, run it first until the end of its CPU burst

  Notes:
  - the Shortest Job First algorithm is a non preemptive algorithm
  - the text says that SJF can be either preemptive or not; we will always refer to the preemptive version as Shortest Remaining Time instead.
  - shortest job first is optimal at finishing the maximum number of cpu bursts in the shortest time, if estimates are accurate

  Problems:
  - SJF cannot handle infinite loops
  - poor performance for processes with short burst times arriving after a process with a long burst time has started
  - processes with long burst times may starve
    - **starvation** - when a process is indefinitely postponed from getting on the processor

  Example:
  - suppose that based on previous information about the processes, our estimates are exactly correct, i.e., we expect process A to take 7 units, B to take 4 units, etc.




```
at time:

  0:      e(A) = 7                       it runs for 7 time units nonpreemptively
 7:      e(B) = 4 and e(C) = 1          choose C, it runs for 1 time unit
 8:      e(B) = 4       choose B, it runs for 4 time units
 12:     e(D) = 1;  e(E) = 3            choose D, it runs for 1 time unit
 13:     e(E) = 3                       choose E, it runs for 3 units
```

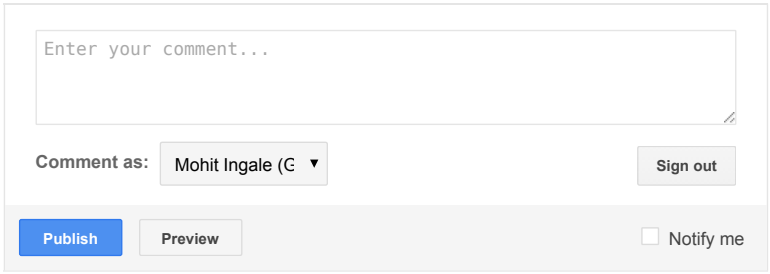
- very short processes get very good service

- a process may mislead the scheduler if it previously had a short bursts, but now may be cpu intensive (this algorithm fails very badly for such a case)
- the penalty ratios are small; this algorithm works extremely well in most cases

Posted by Prakash Arunakar at 11:54

No comments:

Post a Comment



Home
Older Post

Subscribe to: Post Comments (Atom)