CSE 421/521 - Operating Systems
Fall 2014

LECTURE - VII

CPU SCHEDULING - II

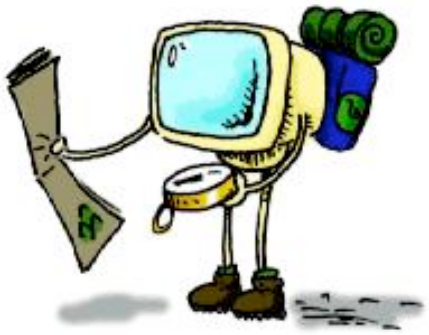
Tevfik Koşar

University at Buffalo

September 16th, 2014

Roadmap

- CPU Scheduling
  - Comparison of CPU Scheduling Algorithms
  - Estimating CPU bursts
  - 4.4BSD SCheduler



Comparison of Scheduling Algorithms

FCFS

PROS:

- It is a fair algorithm
  - schedule in the order that they arrive

CONS:

- Average response time can be lousy
  - small requests wait behind big ones (convoy effect)
- May lead to poor utilization of other resources
  - FCFS may result in poor overlap of CPU and I/O activity
    - E.g., a CPU-intensive job prevents an I/O-intensive job from doing a small bit of computation, thus preventing it from going back and keeping the I/O subsystem busy

SJF

PROS:

- Provably optimal with respect to average waiting time
  - prevents convoy effect (long delay of short jobs)

CONS:

- Can cause starvation of long jobs
- Requires advanced knowledge of CPU burst times
  - this can be very hard to predict accurately!

Priority Scheduling

PROS:

- Guarantees early completion of high priority jobs

CONS:

- Can cause starvation of low priority jobs
- How to decide/assign priority numbers?

Round Robin

PROS:

- Great for timesharing
  - no starvation
- Does not require prior knowledge of CPU burst times
- Generally reduces average response time

CONS:

- What if all jobs are almost time same length?
  - Increases the turnaround time
- How to set the “best” time quantum?
  - if small, then context switch often, incurring high overhead
  - if large, then response time degrades

How to estimate CPU burst time?

Determining Length of Next CPU Burst

- Can only estimate the length
- Can be done by using the length of previous CPU bursts, using exponential averaging

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n.$$

1. t_n = actual lenght of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define :

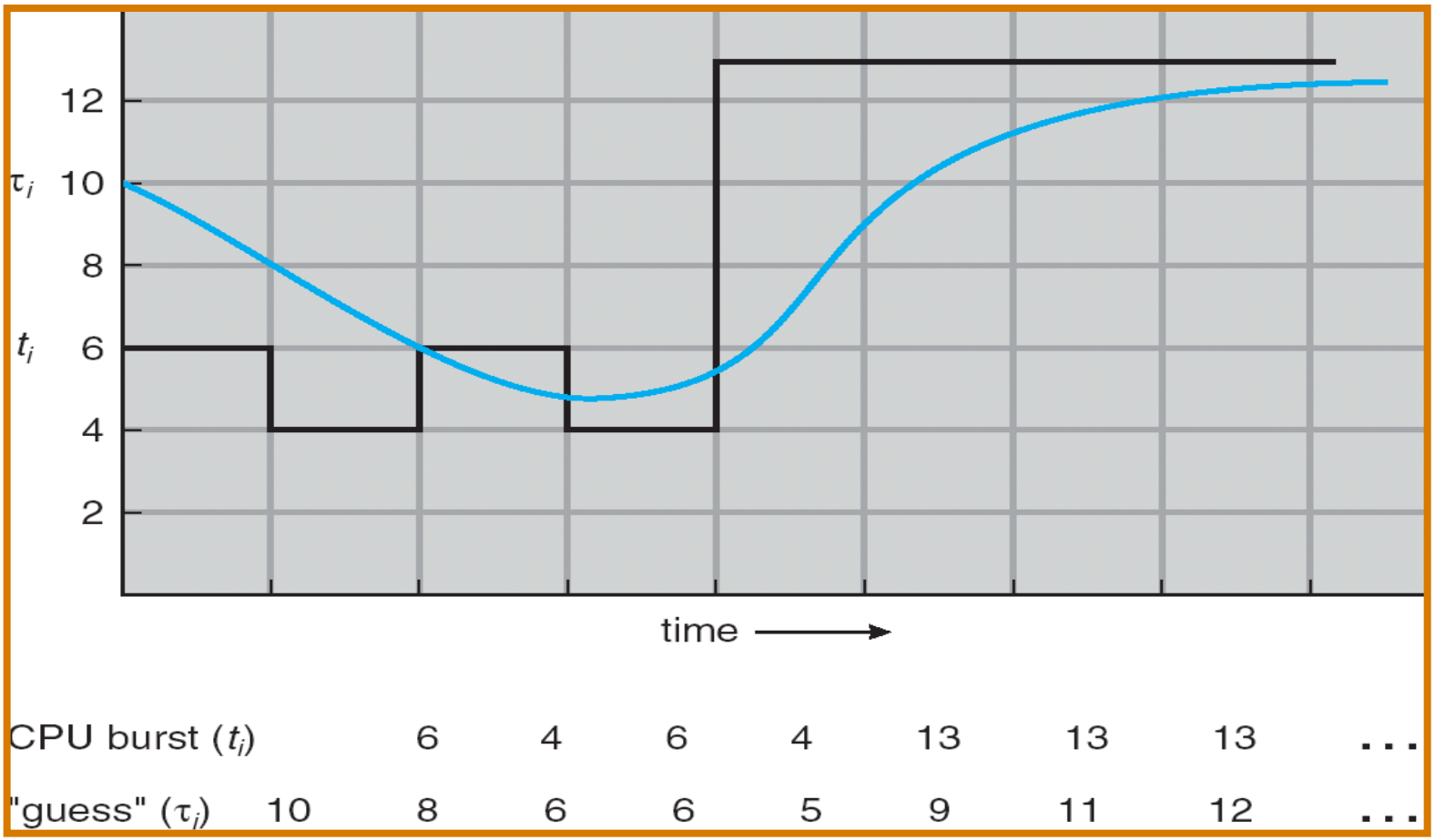
Examples of Exponential Averaging

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count
- $\alpha = 1$
  - $\tau_{n+1} = \alpha\, t_n$
  - Only the actual last CPU burst counts
- If we expand the formula, we get:

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\alpha\, t_{n-1} + \ldots$$
$$+ (1 - \alpha)^j \alpha\, t_{n-j} + \ldots$$
$$+ (1 - \alpha)^{n+1} \tau_0$$

- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

Prediction of the Length of the Next CPU Burst



CPU burst (t_i)	6	4	6	4	13	13	13	...
"guess" (τ_i)	10	8	6	6	5	9	11	12	...

Alpha = 1/2, T0 = 10

Exercise

Consider the exponential average formula used to predict the length of the next CPU burst. What are the implications of assigning the following values to the parameters used by the algorithm?

a. $\alpha = 0$ and $\tau_0 = 100 milliseconds$

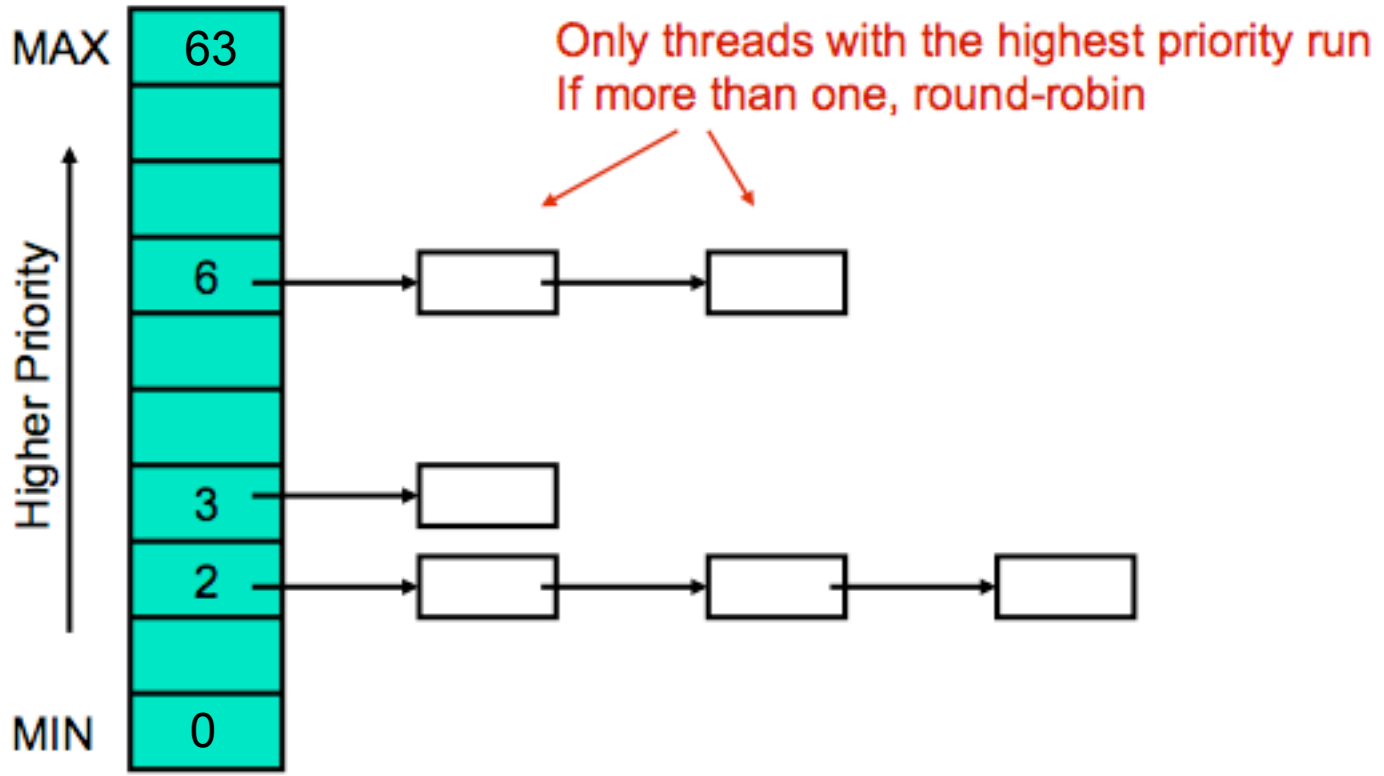
b. $\alpha = 0.99$ and $\tau_0 = 10 milliseconds$

Answer: When $\alpha = 0$ and $\tau_0 = 100 milliseconds$, the formula always makes a prediction of 100 milliseconds for the next CPU burst. When $\alpha = 0.99$ and $\tau_0 = 10 milliseconds$, the most recent behavior of the process is given much higher weight than the past history associated with the process. Consequently, the scheduling algorithm is almost memory-less, and simply predicts the length of the previous burst for the next quantum of CPU execution.

4.4 BSD Scheduler

4.4BSD Priority Based Scheduling

4.4BSD scheduler has 64 priorities and thus 64 ready queues, numbered 0 (PRI_MIN) through 63 (PRI_MAX).



Calculating Priority

- NOTE: Lower numbers correspond to lower priorities in 4.4BSD, so that priority 0 is the lowest priority and priority 63 is the highest.

- priority = PRI_MAX - (recent_cpu / 4) - (nice * 2)

  (rounded down to the nearest integer)

- It gives a thread that has received CPU time recently lower priority for being reassigned the CPU the next time the scheduler runs.

Nice Value

==> how \nice" the thread should be to other threads.

- A nice of zero does not affect thread priority.
- A positive nice, to the **maximum of 20**, decreases the priority of a thread and causes it to give up some CPU time it would otherwise receive.
- A negative nice, to the **minimum of -20**, tends to take away CPU time from other threads.

Calculating recent_cpu

- An array of n elements to track the CPU time received in each of the last n seconds requires O(n) space per thread and O(n) time per calculation of a new weighted average.

- Instead, we use a exponentially weighted moving average:

  - **`recent_cpu(0) = 0`** *// or parent thread's value*
  - *at each timer interrupt,* **`recent_cpu++`** *for the running thread.*
  - *and once per second, for each thread:*

  recent_cpu(t) = a * recent_cpu(t-1) + nice

  where, **`a = (2*load_avg)/(2*load_avg + 1)`**

Calculating load_avg

- Estimates the average number of threads ready to run over the past minute.
- Like recent_cpu, it is an exponentially weighted moving average.
- Unlike priority and recent_cpu, load_avg is system-wide, not thread-specific.
- At system boot, it is initialized to 0. Once per second thereafter, it is updated according to the following formula:

load_avg(t) = (59/60)*load_avg(t-1) + (1/60)*ready_threads

Functions to implement

- Skeletons of these functions are provided in “threads/threads.c”:

- `int thread_get_nice (void)`
- `void thread_set_nice (int new_nice)`

- `void thread_set_priority (int new_priority)`
- `int thread_get_priority (void)`

- `int thread_get_recent_cpu (void)`

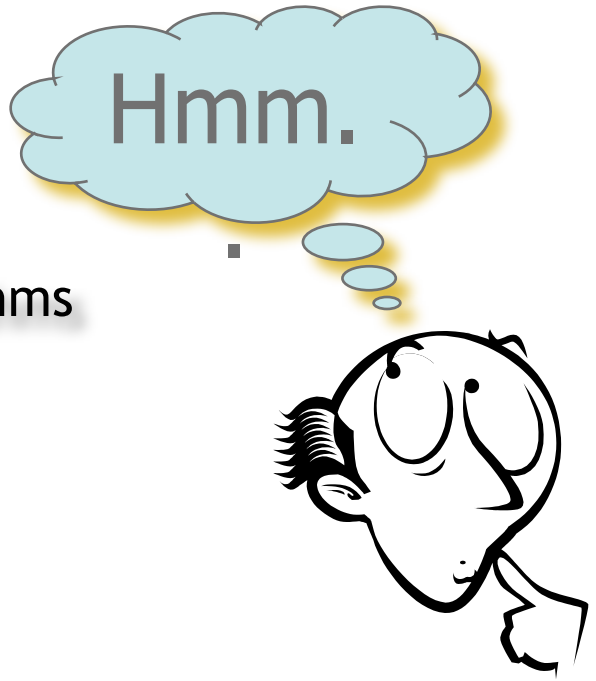
- `int thread_get_load_avg (void)`

Exercise

Process ID	Arrival Time	Priority	Burst Time
A	0	3	20
B	5	1	15
C	10	2	10
D	15	4	5

- Draw gantt charts, find average turnaround, waiting, and response times for above processes, considering:
- 1) First Come First Served Scheduling
- 2) Shortest Job First Scheduling (non-preemptive)
- 3) Shortest Job First Scheduling (preemptive)
- 4) Round-Robin Scheduling
- 5) Priority Scheduling (non-preemptive)
- 6) Priority Scheduling (preemptive)

Summary



- CPU Scheduling
  - Comparison of CPU Scheduling Algorithms
  - Estimating CPU bursts
  - 4.4BSD SCheduler

- Next Lecture: Synchronization

Acknowledgements

- “Operating Systems Concepts” book and supplementary material by A. Silberschatz, P. Galvin and G. Gagne
- “Operating Systems: Internals and Design Principles” book and supplementary material by W. Stallings
- “Modern Operating Systems” book and supplementary material by A. Tanenbaum
- R. Doursat and M. Yuksel from UNR, Ed Lazowska from UWashington