CSC 4103 - Operating Systems
Spring 2008

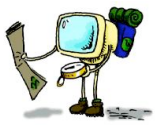
LECTURE - VII
CPU SCHEDULING - II

Tevfik Koşar

Louisiana State University
January 12th, 2008

Roadmap

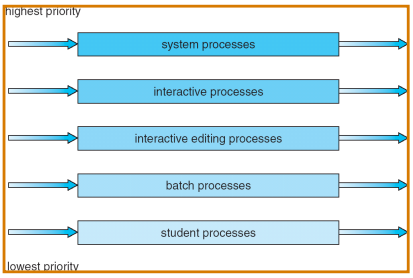
- Multilevel Feedback Queues
- Estimating CPU bursts
- Process Synchronization



Multilevel Queue

- Ready queue is partitioned into separate queues:
  foreground (interactive)
  background (batch)
- Each queue has its own scheduling algorithm
  - foreground – RR
  - background – FCFS
- Scheduling must be done between the queues
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
  - 20% to background in FCFS

Multilevel Queue Scheduling



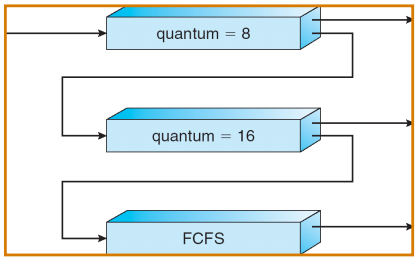
Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS
- Scheduling
  - A new job enters queue Q_0 which is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue Q_1.
  - At Q_1 job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2.

Multilevel Feedback Queues



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

- α =0
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count
- α =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 - α) are less than or equal to 1, each successive term has less weight than its predecessor

Exercise

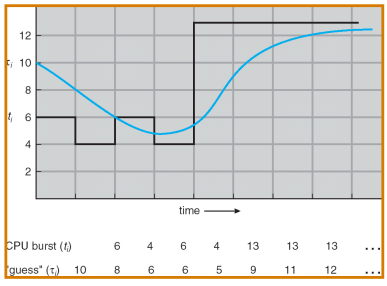
Consider the exponential average formula used to predict the length of the next CPU burst. What are the implications of assigning the following values to the parameters used by the algorithm?

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.

Prediction of the Length of the Next CPU Burst



Alpha = 1/2, T0 = 10

Process Synchronization

Background

- Concurrent access to shared data may result in data inconsistency
- Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes
- Consider consumer-producer problem:
  - Initially, count is set to 0
  - It is incremented by the producer after it produces a new buffer
  - and is decremented by the consumer after it consumes a buffer.

Producer

```
while (true)
 /* produce an item and put in nextProduced
            while (count == BUFFER_SIZE)
                    ; // do nothing
            buffer [in] = nextProduced;
            in = (in + 1) % BUFFER_SIZE;
            count++;
  }
```


Consumer

```
while (1)
 {
            while (count == 0)
                    ; // do nothing
            nextConsumed =  buffer[out];
            out = (out + 1) % BUFFER_SIZE;
            count--;
/*  consume the item in nextConsumed
}
```


Race Condition

- count++ could be implemented as
  register1 = count
  register1 = register1 + 1
  count = register1
- count-- could be implemented as
  register2 = count
  register2 = register2 - 1
  count = register2
- Consider this execution interleaving with "count = 5" initially:
  S0: producer execute register1 = count {register1 = 5}
  S1: producer execute register1 = register1 + 1 {register1 = 6}
  S2: consumer execute register2 = count {register2 = 5}
  S3: consumer execute register2 = register2 - 1 {register2 = 4}
  S4: producer execute count = register1 {count = 6 }
  S5: consumer execute count = register2 {count = 4}

Summary

- Multilevel Feedback Queues
- Estimating CPU bursts
- Process Synchronization

- Next Lecture: Process Synchronization
- Reading Assignment: Chapter 6 from Silberschatz.

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