Lecture 6: DEADLOCK I (Feb 3, 2005) Yap

February 17, 2005

1 ADMIN

- The Programming part of hw1 is due today.
- Hw2 is out today. Due on Feb 10.
- READING: start to read up on deadlocks (CHAPTER 3)

2 Review

- Q: To give scheduling priority to short processes, we need some way of estimating the runtime of the process. Give two ways to do this automatically.

  A: (1) Aging method to estimate the runtime of a program. (2) Quantum Queues: Jobs that have used up 2^i quanta will be put in the i-th queue. Jobs in Q_i has lower priority than jobs in Q_{i+1} but have a larger quanta.

- Q: To give scheduling priority to IO-Bound processes, we need some way estimating the probability that the process is IO-Bound. How can we do this?

- Q: What is the Aging Method of estimating the length of time for running a particular program?

  A: It keeps track of a current estimate of length of time. After each execution of this program, the estimate is updated by a formula that involves the previous estimate and the time for the latest execution.

  The simplest version of this (Old-Estimate + Current-Time)/2. Thus, the "weight" of a runtime that is i generations ago will be only 2^{-i-1}.

- Q: Suppose that in a Realtime Environment, we have periodic events E_i $(i = 1, 2, \ldots)$ which occurs every T_i seconds and which requires C_i seconds to run each time. Give a simple criteria to know whether these events are feasible.

  A: The formula is $\sum_i C_i/T_i \leq 1$.

3 DEADLOCKS

- WHAT IS THE DEADLOCK PROBLEM?

  1. Deadlock Condition: a set of processes, each waiting on an event that only another in the set can cause.
  2. Example:

- Process-Resource Model:

  1. Each process may need one or more of a resource.
  2. Resouce Examples: monitor, printer, scanner, CD recorder, plotter, tape drive, etc.
  3. Some resouces are exclusive.
  4. Some resources are preemptable but some are not.
  5. There may be many copies of the same resource.
  6. Some process may be several resource.

- 4 NECESSARY AND SUFFICIENT CONDITIONS FOR DEADLOCK (Coffman et all (1971):

  1. Processes require EXCLUSIVE use of resources
  2. Processes requesting resources will WAIT until granted.
  3. Use of resource is NON-PREEMPTABLE (e.g., printer, CD burner)
  4. CIRCULARITY of waits among 2 or more processes.

  REMARK: this is an if and only if list of conditions.

- Holt's RESOURCE GRAPHS are bipartitite dgraphs with 2 kinds of nodes:

  1. Circles= processes, Squares=resources
  2. (circle P →square R): process P is blocking on resource R.
  3. (square R →process P): resource R is currently used by process P.
  4. DEADLOCK: cycle

  Note that this is a runtime graph.

- Example (Figure 3-4):

  1. A: Request R, Request S, Release R, Release S.
  2. B: Request S, Request T, Release S, Release T.
  3. C: Request T, Request R, Release T, Release S.
  4. SHOW THE HOLT GRAPH after each step of the following execution sequence:

5. A request R, B request S, C request T,
6. A request S, B request T, C request R.

- 4 APPROACHES TO DEADLOCKS
  1. Ostrich Algorithm: Ignore (Unix, Windows)
  2. Detect and Recover (reboot?)
  3. Dynamic avoidance: careful resource allocation
  4. Prevention: negate one of the 4 conditions for deadlock.
- Detection of Deadlock
  1. Assume each resource is unique (single copy).
  2. ALGORITHM: use depth first search from EACH node of the graph.
  3. MORE PRECISELY: initially all nodes and edges are marked "unseen". A driver loop will run a dfs search from each unseen node.

     When a node or edge is first seen, it is marked "seen". However, when all the outgoing edges of a node have been marked "seen", the node is marked "done".

     When we first traverse an edge, we check to see if the node at the other end of the edge is "unseen", "seen" or "done". If unseen, we extend our recursive search to this node; if seen, we have detected a cycle; if "done", we back up from our DFS search.
  4. REMARK: This algorithm is superior to the one in the text. It has running time $O(m+n)$ where the digraph has m edges and n nodes.
- Generalization to Multiple Copies of Resources.
  1. Processes $p_1, \ldots, p_n$
  2. Resource i $(i = 1, \ldots, m)$ has $e_i \geq 0$ copies of its resource. Let $E = (e_1, \ldots, e_m)$.
  3. Let $A = (a_1, \ldots, a_m) \leq E$ be the **available vector**.
  4. **Current allocation matrix** C is a $n \times m$ matrix, where p_i holds C_{ij} copies of resource i.
  5. **Request matrix** R is a $n \times m$ matrix, where p_i requests C_{ij} copies of resource i.
  6. To detect deadlock, initially all processes are unmarked. Take any unmarked process for with the ith row of R is less than or equal to A. Add this row to A, mark the process, and repeat. When stopped, we have deadlock iff there are unmarked processes left.