Lecture VII
Resource Allocation and Deadlocks

§1. Introduction

§2. 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: Dining Philosophers.
  3. System Deadlock: no process in the system can may progress. (CONTRAST: a subset of processes may be locked out, but the system is not deadlocked)
- Process-Resource Model:
  1. Each process may need one or more of a resource.
  2. Physical Resource Examples: CPU, monitor, printer, scanner, CD recorder, plotter, tape drive, etc.
  3. Logical Resources Examples: files, semaphores, monitors, etc.
  4. Some resouces are exclusive.
  5. Some resources are preemptable but some are not.
  6. There may be many copies of the same resource.
  7. Some process may need several resources simultaneously.
  8. PROTOCOL: a process must REQUEST a resource before use, and must RELEASE resource after use.
- 4 NECESSARY AND SUFFICIENT CONDITIONS FOR DEADLOCK (Coffman et all (1971):
  1. (MUTUAL EXCLUSION) Processes require EXCLUSIVE use of resources
  2. (HOLD AND WAIT) Processes requests more than ONE resources, and must WAIT until all requests have been granted.
  3. (NON-PREEMPTION) Use of resource is non-preemptable (e.g., printer, CD burner)
  4. (CIRCULARITY) Two or more processes are waiting in a cyclic fashion: process P_i waits for $P_{(i+1) \bmod n}$ for $i = 0, 1, \ldots, n-1$.

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 requesting resource R.
  3. (square R → process P): resource R is currently held by process P.
  4. CONSTRAINT: for any process P and resource R, at most one of the edges (P → R) and (R → P) can appear in the graph.
  5. DEADLOCK: cycle
  6. How to draw bipartite graph: we list all the processes on the left hand side, and all the resources on the right hand side. This is illustrated in the sequence of Holt Graphs in Figure 1.
  7. Note that this is a runtime graph.

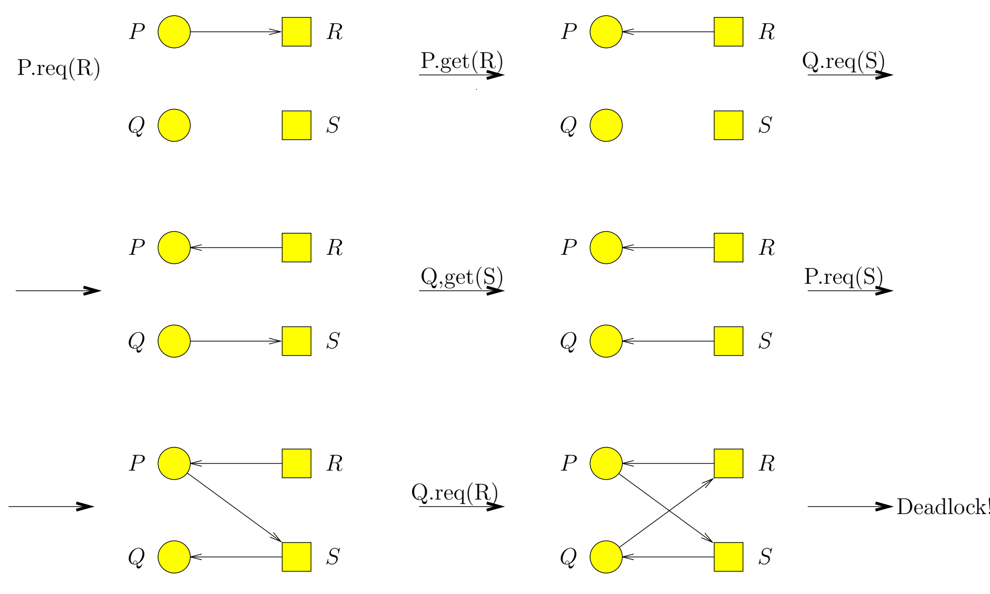

Figure 1: A sequence of Holt graphs, illustrating the transformations

Consider two processes P and Q, and both requires the resources R and S for their work.

1. P: REPEAT(Request R; Request S; WORK; Release R; Release S).
2. Q: REPEAT(Request S; Request R; WORK; Release R; Release S).
3. Suppose we have the following execution sequence:

$$P.req(R), Q.req(S), P.req(S), Q.req(R)$$

4. The figure Figure 1 illustrates the eventual deadlock.

- 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 in the Simple Case:
  1. Simple Case means this: each resource is unique (single copy).
  2. In the simple case, we have deadlock IF and ONLY IF there is a cycle in the resource graph.
  3. ALGORITHM: use depth first search from EACH node of the graph.
  4. 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.
  5. REMARK: This algorithm is superior to the one in Tanenbaum text. It has running time $O(m+n)$ where the digraph has m edges and n nodes.
- Generalization to Multiple Copies of Resources.
  1. QUESTION: Why can't the general case of multiple copies be reduced to this one? E.g., if there are 5 copies of resource r1, just duplicate r1 five times. ANSWER: You could, but it does not truly reflect the fact that these 5 copies are interchangeable. So, you lose something in this reduction to the simple case.
  2. Assume n processes and m resources.
  3. Processes are $p_1, \ldots, p_n$, resources are $r_1, \ldots, r_m$.
  4. We know the **Maximum Request Matrix** M is a $n \times m$ matrix, where indicating that p_i never request more than M_{ij} copies of r_j. Let $M[i]$ denote the ith row of M.
  5. There is a total of $e_j \geq 0$ copies of r_j $(j = 1, \ldots, m)$ Let $E = (e_1, \ldots, e_m)$.
  6. The **(current) Allocation matrix** A is a $n \times m$ matrix, where p_i holds A_{ij} copies of r_j. Let $A[i]$ denote the ith row of A.

7. Let $F = (f_1, \ldots, f_m) \le E$ be the **free vector**. This can determined from A and E. In fact, $f_j = e_j - \sum i = 1^n A_{ij}$.
8. PROBLEM FORMULATION: the ith process can, at any time, release ALL of its allocated resources, or request a vector $B = (b_1, \ldots, b_m)$ of resource. We require $b_j \le C_{ij}$ for $j = 1, \ldots, m$. Our algorithm has to respond to each request for resources, or release of resources. When a request is made, we either fulfil it (and update A by incrementing the ith row with vector B), or put it on hold (by placing the vector B in the Q). When resources are released, we check to see if any pending requests in Q can be fulfilled.
9. STATE: this is defined as the the current allocation matrix A.
10. ASSUMPTION: a process will terminate in finite time when all its requests are satisfied, and at that time, all its allocated resources will be released.
11. A state is **safe** if there is way to grant any future requests so that eventually all the allocated resources will be released.
12. GENERALIZED RESOURCE GRAPHS: In the text (section 8.5.2), a graphical representation of the state is given by introducing a third kind of edge, a "claimed edge".

- Safety Test Algorithm: returns true iff the allocation matrix A is safe.

  Initially, all rows of A are unmarked.

  Let F be the current free vector.

  While some rows are unmarked, do:

  Check if there is some i such that $A[i] + F \ge M[i]$. If there are no such i, then return "unsafe". Otherwise, we mark row i and update F to $F + A[i]$.

  When we exit from this while-loop, all rows are marked. We return "safe".

  This easily leads to an $O(n^2)$ time algorithm to check for safety.

- BANKER'S ALGORITHM:

  1. There is a current allocation matrix A is is safe. This determines a free vector F.
  2. Each customer can either ask to borrow more money (up to its credit limit) or return borrowed money.
  3. The banker has to decide to satisfy a request for money or to put it on hold. and if this limit is reached, it will eventually return all the borrowed money (if not earlier).
  4. Suppose process i makes a request $W = (w_1, \ldots, w_m)$ where w_i is the number of r_j that p_i wants. Note that $W + A[i] \le M[i]$, be assumption.

5. Let A' be the updated allocation matrix where $A[i]$ is replaced by $A[i] + W$. The banker's algorithm says: grant the request iff A' is safe.
6. EXAMPLE:

§3. Model of Resource Allocation and Deadlocks

- Execution Trajectories: Figure 3-8 [p.176] of text.
- The Banker's algorithm (only one resource, money):

  For any request, if allocating it is safe, do it.

  Safety: if there is a way to deplete the currently allocations to 0.

  While there is a non-zero allocation A_i to some process p_i, if the current free amount f is at least as great as the claimed amount c_i (or, need amount) we let $f = f + A_i$ and set $A_i = 0$. When all $A_i = 0$, we are safe. Else, not.
- SIMPLE EXAMPLE of Banker's Algorithm (1 resource)

	Has	Max
A	0	6
B	0	5
C	0	5
D	0	7
Free	10	

	Has	Max
A	1	6
B	1	5
C	2	5
D	4	7
Free	2	

	Has	Max
A	1	6
B	2	5
C	2	5
D	4	7
Free	1	

Initially, Safe, Unsafe.

Exercises

Exercise 3.1: State the 4 necessary conditions for a deadlock to occur.

♢

End Exercises