

6. Deadlocks

- 6.1 Deadlocks with Reusable and Consumable Resources
- 6.2 Approaches to the Deadlock Problem
- 6.3 A System Model
 - Resource Graphs
 - State Transitions
 - Deadlock States and Safe States
- 6.4 Deadlock Detection
 - Reduction of Resource Graphs
 - Special Cases of Deadlock Detection
 - Deadlock Detection in Distributed Systems
- 6.5 Recovery from Deadlock
- 6.6 Dynamic Deadlock Avoidance
 - Claim Graphs
 - The Banker's Algorithm
- 6.7 Deadlock Prevention

ICS 143

1

Deadlocks

- informally: process is blocked on resource that will never be released
- deadlocks waste resources
- deadlocks are rare:
 - many systems ignore them
 - resolved by explicit user intervention
 - critical in many real-time applications
 - may cause damage, endanger life

ICS 143

2

Reusable/consumable resources

- reusable resource
 - number of units is constant
 - unit is either free or allocated; no sharing
 - process requests, acquires, releases unitsExamples: memory, devices, files, tables
- consumable resources
 - number of units varies at runtime
 - process may create new units
 - process may consume unitsExamples: messages, signals

ICS 143

3

Examples of deadlocks

```
p1: ...           p2: ...
  open(f1,w);     open(f2,w);
  open(f2,w);     open(f1,w);
  ...             ...
```

- deadlock when executed concurrently

```
p1: if (C) send(p2,m);  p2: ...
    while(1){...       while(1){...
    recv(p2,m);         recv(p1,m);
    send(p2,m);         send(p1,m);
    ... }               ... }
```

- deadlock when C not true

ICS 143

4

Deadlock, livelock, starvation

- deadlock: processes are blocked
- livelock: processes run but make no progress
 - both deadlock and livelock lead to starvation
- but starvation may have other causes:
 - ML scheduling where one queue is never empty
 - memory requests: unbounded stream of 100MB requests may starve a 200MB request

ICS 143

5

Approach to deadlock problem

- detection and recovery
 - allow deadlock to happen and eliminate it
- avoidance
 - runtime checks disallow allocations that might lead to deadlocks
- prevention
 - restrict type of request and acquisition to make deadlock impossible

ICS 143

6

System model

- resource graph:
 - process: circle
 - resource: rectangle with small circles for each unit
 - request: edge from process to resource class
 - allocation: edge from resource unit to process
- Example: Figure 6-1

ICS 143

7

System model

- state transitions
 - request: create new request edge $p_i \rightarrow R_j$
 - p_i must have no outstanding requests
 - number of edges between p_i and $R_j \leq$ units in R_j
 - Example: Figure 6-2
 - acquisition: reverse request edge to $p_i \leftarrow R_j$
 - all requests of p_i must be satisfiable
 - Example: Figure 6-2
 - release: remove edge $p_i \leftarrow R_j$
 - p_i must have no outstanding requests
 - Example: Figure 6-2

ICS 143

8

System model

- process is **blocked** in state S : cannot request, acquire, or release any resource
- process is **deadlocked** in state S : blocked now and remains blocked in all states
- **deadlock state**: contains a deadlocked process
- **safe state**: no deadlock state can ever be reached using request, acquire, release

ICS 143

9

Example

- p1 and p2 both need R1 and R2
- p1 requests R1 before R2 and releases R2 before R1 and
- p2 requests R2 before R1 and releases R1 before R2

- transitions by p1 only: Figure 6-3
- transitions by p1 and p2: Figure 6-4

ICS 143

10

Deadlock detection

- graph reduction
 - repeat:
 - select unblocked process p
 - remove p and all request and allocation edges
- deadlock \Leftrightarrow graph not completely reducible
- all reduction sequences lead to the same result
- Example: Figure 6-5

ICS 143

11

Special cases of detection

- testing for specific process
 - reduce until p is removed or graph irreducible
- continuous detection
 - current state not deadlocked
 - next state T deadlocked only if:
 - operation was a request by p
 - p is deadlocked in T
 - try to reduce T by p

ICS 143

12

Special cases of detection

- immediate allocations
 - all satisfiable requests granted immediately
 - expedient state: no satisfiable request edges
 - knot in expedient state \Rightarrow deadlock
 - knot K:
 - every node in K reachable from any other node in K
 - no outgoing edges (only incoming)
 - intuition:
 - all processes must have outstanding requests in K
 - expedient state: requests not satisfiable

ICS 143

13

Special cases of detection

- single-unit resources
 - cycle \Rightarrow deadlock
 - intuition:
 - every p must have a request edge to R
 - every R must have an allocation edge to p'
 - R is not available and thus p is blocked
- wait-for graph
 - show only processes
 - replace $p1 \rightarrow R \rightarrow p2$ by $p1 \rightarrow p2$: p1 waits for p2
 - Example: Figure 6-6

ICS 143

14

Detection in distributed systems

- central coordinator
 - each machine maintains a local wfg
 - changes are reported to central coordinator
 - coordinator constructs and analyzes global wfg
- problems
 - coordinator is a performance bottleneck
 - communication delays may cause phantom deadlocks
 - Example: Figure 6-7

ICS 143

15

Detection in distributed systems

- distributed approach
 - detect cycles using probes
 - process p_i blocked on p_j : launches probe $p_i \rightarrow p_j$
 - p_j sends probe $p_i \rightarrow p_j \rightarrow p_k$ along all request edges, and so on
 - when probe returns to p_i , cycle is detected
- Example: Figure 6-8

ICS 143

16

Recovery from deadlock

- process termination
 - kill all processes involved in deadlock
 - kill one at a time; in what order:
 - by priority: consistent with scheduling
 - by cost of restart: length of recomputation
 - by impact on other processes: CS, producer/cons.
- resource preemption
 - direct: temporarily remove resource (e.g. memory)
 - indirect: rollback to earlier checkpoint

ICS 143

17

Dynamic deadlock avoidance

- maximum claim graph
 - process indicates maximum resources needed
 - potential request edge $p_i \rightarrow R_j$ (dashed)
 - may turn into real request edge
- Example: Figure 6-9

ICS 143

18

Dynamic deadlock avoidance

- Theorem: prevent acquisitions that do not produce reducible graph \Rightarrow all state are safe
- the banker's algorithm (Dijkstra)
 - given a satisfiable request $p \rightarrow R$
 - temporarily grant request: change $p \rightarrow R$ to $R \rightarrow p$
 - reduce new claim graph
 - if reducible, proceed, else reverse acquisition
- Example: Figure 6-10

ICS 143

19

Dynamic deadlock avoidance

- single-unit resources
 - check for cycles after tentative acquisition
 - disallow if cycle is found
 - Example: Figure 6-11(a)
- if claim graph contains no (undirected) cycles, all states are safe (no directed cycle can ever be formed)
- Example: Figure 6-11(b)

ICS 143

20

Deadlock prevention

- deadlock requires the following conditions:
 - mutual exclusion:
 - resources not sharable
 - hold and wait:
 - process must be holding one resource while requesting another
 - circular wait:
 - at least 2 processes must be blocked on each other

ICS 143

21

Deadlock prevention

- eliminate mutual exclusion:
 - not possible in most cases
 - spooling makes I/O devices sharable
- eliminate hold-and-wait
 - request all resources at once
 - release all resources before a new request
 - release all resources if current request blocks
- eliminate circular wait
 - order all resources: $SEQ(R_i) \neq SEQ(R_j)$
 - process must request in ascending order

ICS 143

22
