

Safety & Liveness

Uwe R. Zimmer - The Australian National University

Safety & Liveness Fairness $(P(I) \wedge Processes (I,S)) \Rightarrow \diamondsuit Q(I,S)$ where $\diamondsuit Q$ means that Q does eventually hold (and will then stay true)

Weak fairness: ◊□R ⇒ ◊G ... eventually, if a process requests continually.

avoid starvation): Resources will be granted ...

• Strong fairness: $\Box \Diamond R \Rightarrow \Diamond G$... eventually, if a process requests infinitely often.

Linear waiting: ○R ⇒ ○G ... before any other process had the same resource granted more than once (common faimess in distributed systems).

First-in, first-out c > c > c... before any other process which applied for the same resource at a later point in time (common fairness in single-node systems).

Safety & Liveness Towards synchronization

Circular dependencies

Safety & Liveness Deadlocks Necessary deadlock conditions:

Mutual exclusion: resources cannot be used simultaneously.

2. Hold and wait: a process applies for a resource, while it is holding another resource (sequential requests). 3. No pre-emption: resources cannot be pre-empted; only the process itself can release resources.

Circular wait: a ring list of processes exists, where every process waits for release of a resource by the next one.

(Chandy 1983)
Chandy, K. Misra, layadev & Haas, Laura
Distributed deadfock detection
Transactions on Computer Sys-tems (TOCS) 1983 vol. 1(2)

Safety & Liveness

References for this chapter

[Silberschatz 2001]
Silberschatz Abraham, Galvin, Peter & Gagne, Greg
Operating System Concepts
John Wiley & Sons, Inc., 2001

Safety & Liveness

Correctness concepts in concurrent systems

 $(P(I) \land Processes(I,S)) \Rightarrow \Box Q(I,S)$ where $\Box Q$ means that Q does always hold

(Avoidance / prevention of deadlocks is one central safety property)

Deadlocks

Most forms of synchronization may lead to

Deadlocks

Safety & Liveness

☞ ... or are there structurally dead-lock free forms of synchronization?

ræ How to predict them?
ræ How to find them?
ræ How to resolve them?

Mutual exclusion (no resource collisions) is has been add

Absence of deadlocks er to be addressed now (and other forms of 'silent death' and 'freeze' conditions)

Specified responsiveness or free capabilities as Real-time system (typical in real-time / embedded systems or server applications)

Safety & Liveness

Deadlocks

Necessary deadlock conditions:

Necessary deadlock conditions:

. Mutual exclusion: resources cannot be used simultaneously

Deadlocks

Safety & Liveness

Mutual exclusion: resources cannot be used simultaneously.

2. Hold and wait: a process applies fo

Safety & Liveness Deadlocks

• Ignorance & restart

© Kill or restart unresponsive processes, power

Necessary deadlock conditions:

Safety & Liveness

Deadlocks

Deadlock strategies:

Deadlock detection & recovery
 ind deadlocked processes and recover

Deadlock avoidance
 The resulting system state is checked before

Deadlock prevention

re A system may become deadlocked, if all these conditions apply!

No pre-emption: resources cannot be pre-empted; only the process itself can release resources.

Circular wait: a ring list of processes exists, where every process waits for release of a resource by the next one.

2. Hold and wait: a process applies for a resource, while it is holding another resource

. Mutual exclusion: resources cannot be us

Safety & Liveness

Correctness concepts in concurrent systems

 $(p(l) \land Processes(l,S)) \Rightarrow \Diamond Q(l,S)$ where $\Diamond Q$ means that Q does eventually hold (and will then stay true

Correctness concepts in concurrent systems

Safety & Liveness

Extended concepts of correctness in concurrent systems: — Termination is aften not intended or even considered a failure

 $(P(I) \land Processes\ (I,S)) \Rightarrow \Box Q\ (I,S)$ where $\Box Q$ means that Q does always hold

 $(P(I) \land Processes(I,S)) \Rightarrow \Diamond Q(I,S)$ where $\Diamond Q$ means that Q does eventually hold (and will then stay true) and S is the current state of the concurrent system

Requests need to complete eventually.
 The state of the system needs to be displayed eventually.
 No part of the system is to be delayed forever (fairness).

Safety & Liveness

Towards synchronization

Reserving resources in reverse order

Sequence of operations $A \to B \to C$; $X \to Y \to Z_2[XZ \mid A,B,C]$ $[A,C \mid X,Y,Z] \to [B \mid Y]$ on: $[A \mid X]$ followed by a deadlock situation.

Safety & Liveness Deadlocks Necessary deadlock conditions:

I. Mutual exclusion: resources cannot be used simultaneously.

Hold and wait: a process applies for a resource, while it is holding ano

Deadlock prevention (Remove one of the four necessary deadlock conditions) Safety & Liveness Deadlocks

1. Break Mutual exclusion:

Mutual exclusion Hold and wait No pre-emption Circular wait

Deadlocks

Deadlock prevention

(Remove one of the four necessary deadlock conditions)

1. Break Mutual exclusion: By replicating critical resources, mutual exclusion becomes un-necessary (only applicable in very specific cases).

Mutual exclusion Hold and wait

480

1

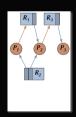
2 Break Hold and wait:

Safety & Liveness Deadlocks Resource Allocation Graphs holds (Silberschatz, Galvin & Gagne) $RAG = \{V, E\}$; Resource allocation graphs consist of vertices V and edges E. $V = P \cup R$; Vertices V can be processes P or Resource types R. with processes $P = \{P_1, ..., P_n\}$ and resources types $R = \{R_1, ..., R_k\}$ $E = E_c \cup E_r \cup E_{a'}$ Edges E can be "claims" E_{cr} "requests" E_r or "assignments" in requests E = { claims

Safety & Liveness

Deadlocks Resource Allocation Graphs (Silberschatz, Galvin & Gagne)

™ No circular dependency ™ no deadlock:



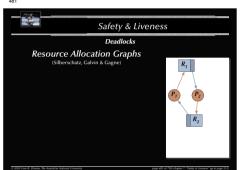
489

Safety & Liveness Deadlocks Resource Allocation Graphs (Silberschatz, Galvin & Gagne) r Knowledge of claims: Claims are potential future requests which have no blocking ef R₂ fect on the claiming process - while actual requests are blocking

477 Safety & Liveness Deadlock prevention (Remove one of the four necessary deadlock conditions) . Break Mutual exclusion: Mutual exclusion Hold and wait By replicating critical resources, mutual exclusion becomes un-necessary (only applicable in very specific cases). No pre-emptior Circular wait 2. Break Hold and wait:

Allocation of all required resources in one request.

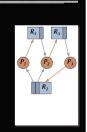
Processes can either hold *none* or *all* of their required resources. 3. Introduce Pre-emption: :



Deadlocks

Safety & Liveness

Resource Allocation Graphs (Silberschatz, Galvin & Gagne)



Safety & Liveness Deadlocks Resource Allocation Graphs (Silberschatz, Galvin & Gagne) **R**₃ Filt Assignment of resources such that circular dependencies are avoided: R_2

(Silberschatz, Galvin & Gagne) Earlier derived rule: If there are cycles in the resource allocation graph

Safety & Liveness Deadlocks

Deadlock prevention

(Remove one of the four necessary deadlock conditions)

1. Break Mutual exclusion:

By replicating critical resources, mutual exclusion becomes un-necessary (only applicable in very specific cases). 2. Break Hold and wait: Allocation of all required resources in one request.

Processes can either hold none or all of their required resources.

Introduce Pre-emption:
 Provide the additional infrastructure to allow for pre-emption of resources. Mind that resources cannot be pre-empted, if their states cannot be fully stored and recovered.

4 Brook Circular waite

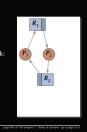
Deadlocks

482



Resource Allocation Graphs (Silberschatz, Galvin & Gagne)

Two process, reverse allocation deadlock:



Mutual exclusio Hold and wait

No pre-emption Circular wait

Safety & Liveness Deadlocks Resource Allocation Graphs (Silberschatz, Galvin & Gagne)

F Two circular dependencies F deadlock:

Derived rule: If some processes are deadlocked then there

are cycles in the resource allocation graph.

Safety & Liveness

Deadlocks

Resource Allocation Graphs

If some processes are deadlocked then there are cycles in the resource allocation graph.

Reverse rule for multiple instances: If there are cycles in the resource allocation graph and there are multiple instances per resource then the involved processes are potentially deadlocked.

Reverse rule for single instances:

and there is exactly one instance per resource then the involved processes are deadlocked.

Safety & Liveness

Deadlocks

Deadlock prevention (Remove one of the four necessary deadlock conditions)

1. Break Mutual exclusion: By replicating critical resources, mutual exclusion becomes un-necessary (only applicable in very specific cases).

2. Break Hold and wait: Allocation of all required resources in one request.

Processes can either hold none or all of their required resources.

3. Introduce Pre-emption: Provide the additional infrastructure to allow for pre-emption of resources. Mind that re-sources cannot be pre-empted, if their states cannot be fully stored and recovered.

4. Break Circular waits:
E.g. order all resources globally and restrict processes to request resources in that order only.

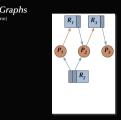
479

Safety & Liveness

Deadlocks

Resource Allocation Graphs

(Silberschatz, Galvin & Gagne)



Hold and wait

No pre-emptior Circular wait

Safety & Liveness

Deadlocks

Edge Chasing

blocking processes: Send a probe to all requested yet unassigned resources containing ids of: [the blocked, the sending, the targeted node].

nodes on probe reception:

Propagate the probe reception:

Propagate the probe to all processes holding the critical resources or to all requested yet unassigned resources – while updating the second and third entry in the probe.

a process receiving its own probe:

Circular dependency detected.

Safety & Liveness Deadlocks

Resource Allocation Graphs

(Silberschatz, Galvin & Gagne)

Reverse rule for single instances: If there are cycles in the resource allocation graph

and there is exactly one instance per resource then the involved processes are deadlocked.

№ Actual deadlock identified



1

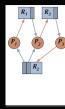
Deadlocks

Resource Allocation Graphs

(Silberschatz, Galvin & Gagne)

Reverse rule for multiple instances: If there are cycles in the resource allocation graph and there are multiple instances per resource then the involved processes are potentially deadlocked.

■ Potential deadlock identified



Safety & Liveness

Deadlocks

Banker's Algorithm

1. Simulated_Free \Leftarrow Free; \forall i: Completed [i] \Leftarrow False;

2.While ∃i: →Completed [i]

Completed [i] ← True;

and ∀j: Requested [i, j] < Simulated_Free [j] do: \forall j: Simulated_Free [j] \Leftarrow Simulated_Free [j] + Allocated [i, j];

3. If ∀i: Completed [i] then the system is currently deadlock-free! else all processes i with -Completed [i] are involved in a deadlock!

Safety & Liveness

Deadlocks

Deadlock recovery

A deadlock has been detected ≈ now what?

Breaking the circular dependencies can be done by

Either pre-empt an assigned resource which is part of the deadlock.

r or stop a process which is part of the deadlock

Usually neither choice can be implemented 'gracefully' and deals only with the symptoms.

Deadlock recovery does not address the reason for the problem! (i.e. the deadlock situation can re-occur again immediately)

Safety & Liveness

Atomic & idempotent operations

Atomic operations

Important implications:

- 1. An atomic operation is either performed in full or not at all.
- 2. A failed atomic operation cannot have any impact on its surroundings (must keep or re-instantiate the full initial state).
- 3. If any part of an atomic operation fails, then the whole atomic operation is declared failed.
- 4. All parts of an atomic operations (including already completed parts) must be prepared to declare failure until the final global commitment.

Safety & Liveness

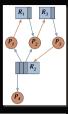
Resource Allocation Graphs

(Silberschatz, Galvin & Gagne)

Reverse rule for multiple instances:

If there are cycles in the resource allocation graph and there are multiple instances per resource
then the involved processes are potentially deadlocked.

 Potential deadlock identified - yet clearly not an actual deadlock here



Safety & Liveness

Deadlocks

Banker's Algorithm

1. Simulated_Free \leftarrow Free; \forall i: Completed [i] \leftarrow False;

2. While ∃i: —Completed [i]

Completed [i] ← True;

and $\forall j$: Claimed [i, j] < Simulated_Free [j] do: ∀j: Simulated_Free [j] ← Simulated_Free [j] + Allocated [i, j];

3. If ∀i: Completed [i] then the system is safe!

A safe system is a system in which future deadlocks can be avoided assuming the current set of available resources.

Safety & Liveness

Deadlocks

Deadlock strategies:

Deadlock prevention System prevents deadlocks by its structure or by full verification

Deadlock avoidance
 System state is checked with every resource assignment.

 Deadlock detection & recovery Detect deadlocks and break them in a 'coordinated' way,

Ignorance & random kill kill or restart unresponsive processes, power-cycle the computer, ...

Safety & Liveness

Atomic & idempotent operations

Idempotent operations

Definition of idempotent operations:

ation are identical for the cases of executing the operation

- infinitely often

- Idempotent operations are often atomic, but do not need to be.
- Atomic operations do not need to be idempotent
- Idempotent operations can ease the requirements for synchronization.

Safety & Liveness

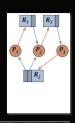
Deadlocks

Resource Allocation Graphs

(Silberschatz, Galvin & Gagne)

How to detect actual deadlocks in the general case?

(multiple instances per resource)



498

Safety & Liveness

Deadlocks

Banker's Algorithm

Check potential future system safety by simulating a granted request: (Deadlock avoidance)

if (Request < Claimed) and (Request < Free) the Free := Free - Request; Claimed := Claimed - Request; Allocated := Allocated + Request;

if System_is_safe (checked by e.g. Banker's algorithm) then r Grant request

Restore former system state: (Free, Claimed, Allocated)

Safety & Liveness

Atomic & idempotent operations

Atomic operations

Definitions of atomicity:

An operation is atomic if the processes performing it ...

 (by 'awareness') ... are not aware of the existence of any other active process, and no other active process is aware of the activity of the processes during the time the processes are performing the atomic operation.

(by communication) ... do not communicate with other s while the atomic operation is performed

(by means of states) ... cannot detect any outside state change and do not reveal their own state changes until the atomic operation is complete.

An atomic operation can be considered to be indivisible and instantaneous.

506

Safety & Liveness Reliability, failure & tolerance

'Terminology of failure' or 'Failing terminology'?

Reliability ::= measure of success

with which a system conforms to its specification. ::= low failure rate.

::= a deviation of a system from its specification.

::= the system state which leads to a failure.

::= the reason for an error.

Safety & Liveness

Deadlocks

Banker's Algorithm There are processes $P_1 \in \{P_1, ..., P_n\}$ and resource types $R_1 \in \{R_1, ..., R_m\}$ and data structures:

FIF the number of currently available resources of type 1.

- Allocated [i, j]
- # the number of resources of type j currently allocated to process i. • Free Fil
- Claimed Fi il
- at the number of resources of type i required by process i eventually Requested [i. i]
- Fir the number of currently requested resources of type 1 by process 1 • Completed [i]
- ser boolean vector indicating processes which may complete.
- Simulated Free [i]

r Number of available reso ources assuming that complete processes deallocate their resources.

499

Safety & Liveness

Deadlocks Distributed deadlock detection

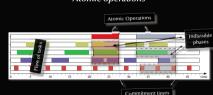
Observation: Deadlock detection methods like Banker's Algorithm are too communication intensive to be commonly applied in full and at high frequency in a distributed system.

■ Therefore a distributed version needs to:

- Split the system into nodes of reasonable locality (keeping most processes close to the resources they require).
- FIF Organize the nodes in an adequate topology (e.g. a tree).
- *** Check for deadlock inside nodes with blocked resource requests and detect/avoid local deadlock immediately.
- Exchange resource status information between nodes occasionally and detect global deadlocks eventually

Safety & Liveness

Atomic & idempotent operations Atomic operations



Safety & Liveness

Reliability, failure & tolerance Faults during different phases of design

 Inconsistent or inadequate specifications Fig. frequent source for disastrous faults

 Software design errors ex frequent source for disastrous faults

Component & communication system failures

rare and mostly predictable



Safety & Liveness

Reliability, failure & tolerance Faults in the logic domain

Non-termination / -completion

Systems 'frozen' in a deadlock state, blocked for missing input, or in an infinite loop

Watchdog timers required to handle the failure

· Range violations and other inconsistent states

· Value violations and other wrong results

User-level exception handling required to handle the failure

512



Safety & Liveness

Reliability, failure & tolerance

Fault tolerance

Full fault tolerance

Graceful degradation (fail soft)
 the system continues to operate in the presence of foreseeable error conditions, while accepting a partial loss of functionality or performance.

· Fail safe

□ Graceful degradation might have multiple levels of reduced functionality.



513



Safety & Liveness Reliability, failure & tolerance Observable failure modes

