Deadlock and Starvation

Giovanni Agosta

Piattaforme Software per la Rete – Modulo 2

Outline

1. Deadlock Basics Review

2. Deadlock in Client-Server Systems

3. Starvation

Deadlock Basics Review

Definition

A situation in which computation cannot proceed because a set of two or more components in the system is block and each component is waiting on another component in the set.

- A “component” can be any independent control flow, and location on the same or different machines is not a factor
- Usually, a component waits on another to release control on an exclusively accessed resource, or to send some message which only it can generate

Deadlock Basics Review

Conditions

The following conditions are needed for a *deadlock* to occur:

Mutual Exclusion There must be shared resources (which may include abstract resources) that are accessed in mutual exclusion;

No Preemption Components cannot be interrupted;

Hold & Wait Components can hold exclusive access to resources while waiting for other resources;

Circular Wait The waiting relation between components, modeled as a graph, has cycles.

Deadlock Basics Review

Conditions

When considering a distributed computing environment, similar conditions may appear as:

Mutual Exclusion Responses must be received from a single source;

No Preemption Components cannot be interrupted;

Hold & Wait Components can forestall sending responses while waiting for responses from other components;

Circular Wait The waiting relation between components, modeled as a graph, has cycles.

Deadlock Basics Review

Detection is difficult

- Need to know which resources are held by each component
- Abstract resources are difficult to track

Endline: no practical program can be built to detect deadlock in a distributed environment

Deadlock in Client-Server Systems

Avoiding Deadlock

- Understand the conditions
- Plan protocols and software to avoid deadlock conditions
- Use of request-response paradigm in Client-Server systems helps, but has some remaining issues:
    - Need for full synchronization specification
    - Unreliable transport may cause deadlock

Deadlock in Client-Server Systems

Lack of Full Synchronization Specification

Protocol Example

- Client establishes connection with Server
- Either Client or Server sends initial message; the other end waits for initial message and sends initial response
- After the initial message exchange, Client sends requests, and Server sends responses to each request
- After receiving response to its last request, Client closes the connection

Both Client and Server may end up waiting for initial response.

Deadlock in Client-Server Systems

Unreliable Transport

Exchange example

Assume UDP is used with a protocol designed to work with reliable transport:

- Client establishes connection with Server
- Client sends request, which is lost
- Server waits for request, Clients waits for response

Solution: use reliable transfer or timeout mechanism (equivalent to removing *no preemption* condition)

Starvation

Definition

A situation in which some clients cannot access a service, while others can.

Example

- Iterative servers allowing arbitrarily long interactions
- Can be exploited for denying service to others

Solutions

- Timeout in waiting for requests (idle timer)
- Maximum number of requests serviced per connection

Busy Connection and Starvation

Buffering issues

- Buffers are used on both sender and receiver ends
- Buffer size differences can be exploited to delay or prevent transmission (e.g., specify a small receive buffer size while requesting transmission of large data)
- Since the idle timer measures delay between response receipt and next request, it is left at zero...

Solutions

- Concurrent server
- Server makes only non-blocking calls