CS 455: Introduction To Distributed Systems [Distributed Coordination/Mutual Exclusion]

Synchronize Thy Actions for Coordination

Trying to have processes coordinate?
The key is getting them to wait

For it's exclusive authorization
That predicates entry into the critical section

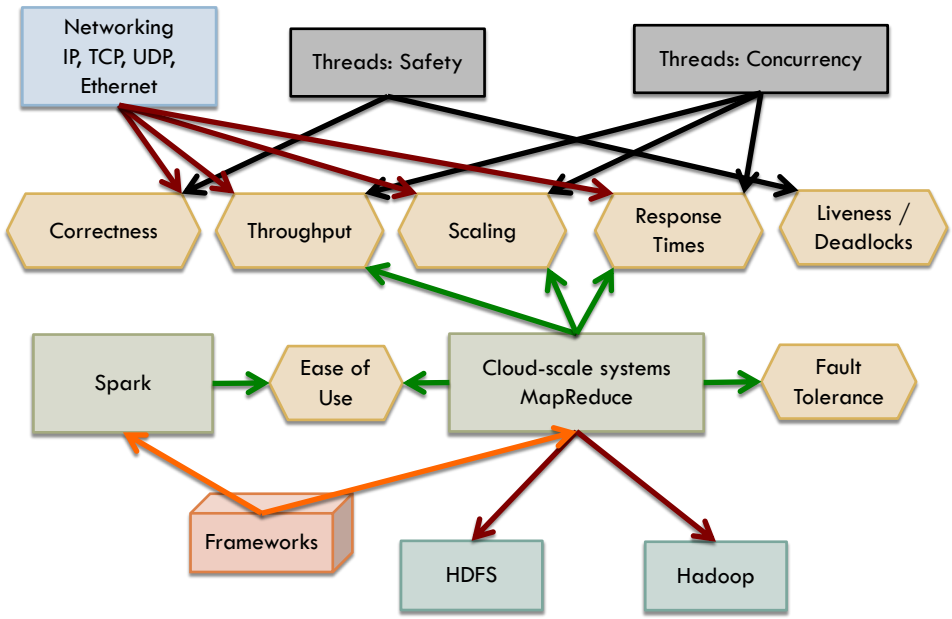
Either a token a process must possess
Or permissions it should collect for the access

Shrideep Pallickara
Computer Science
Colorado State University

Topics covered in this lecture

- Distributed Coordination
- Distributed Mutual Exclusion

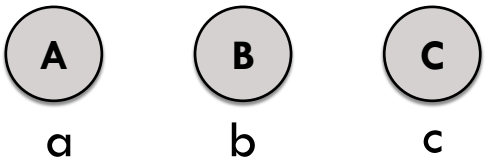
THE JOURNEY SO FAR



Motivating Example [1/2]

- Three processes A, B, and C have each come up with a number
- They need to know the sum of these 3 numbers **without** the other processes knowing what "individual" number each process has
  - Each process A, B and C is holding on to a number
    - a, b, and c respectively
  - The objective is for A, B, and C to "know" the sum of a + b + c
    - WITHOUT giving away information about the number that they hold

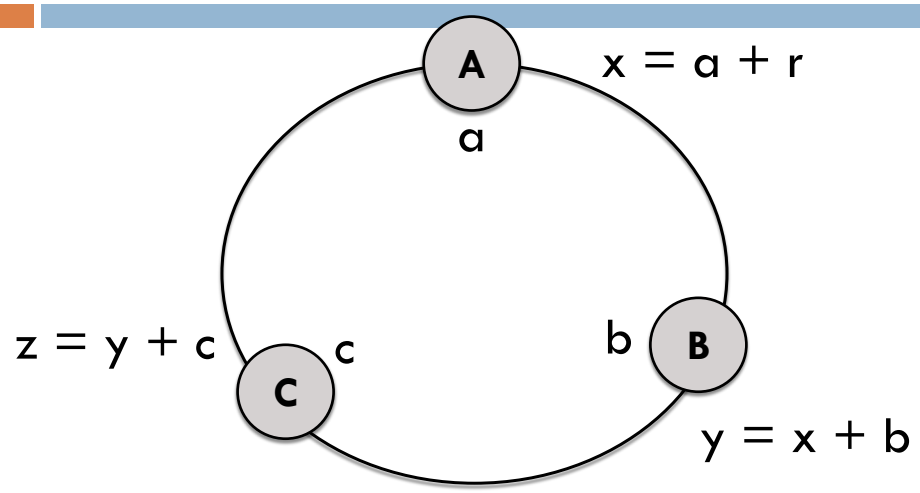
Motivating Example [1/2]



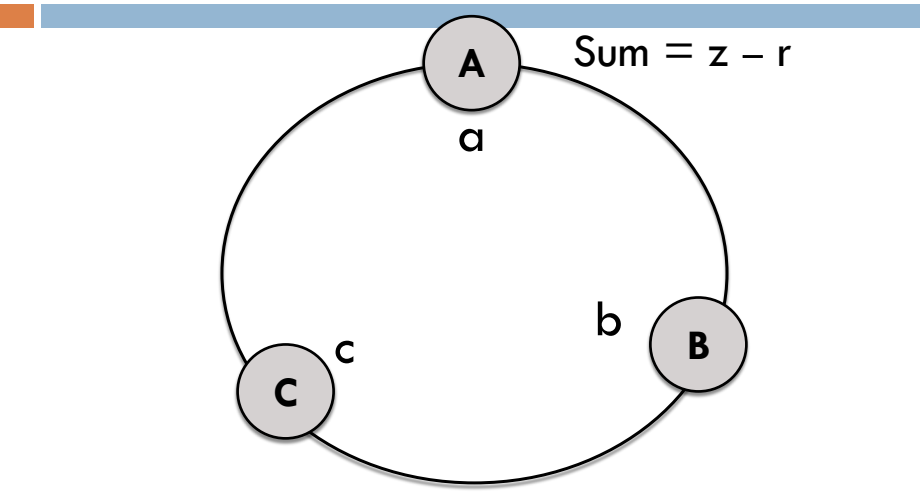
Objective: a + b + c

Without A revealing a, B revealing b, and C revealing c

Solution [1/2]



Solution [2/2]



DISTRIBUTED COORDINATION

What we will cover

- Collection of algorithms whose goals vary, but share an aim that is fundamental in distributed systems
  - For a set of processes to:
    - **Coordinate** their actions
    - **Agree** on one or more values

Communication styles

- Asynchronous communications
  - No timing assumptions
- Synchronous communications have bounds on
  - Maximum message transmission delay
  - Time to execute each step of a process
  - Clock drift rates

Allows us to use timeouts to detect process crashes

Coordination & Agreement

- A set of processes need to **coordinate** actions or **agree** on a set of values
- Must be able to do so even when *hierarchical* relationships do not exist
  - E.g.: Controller-Worker where a single point of failure exists

Example: Spaceship

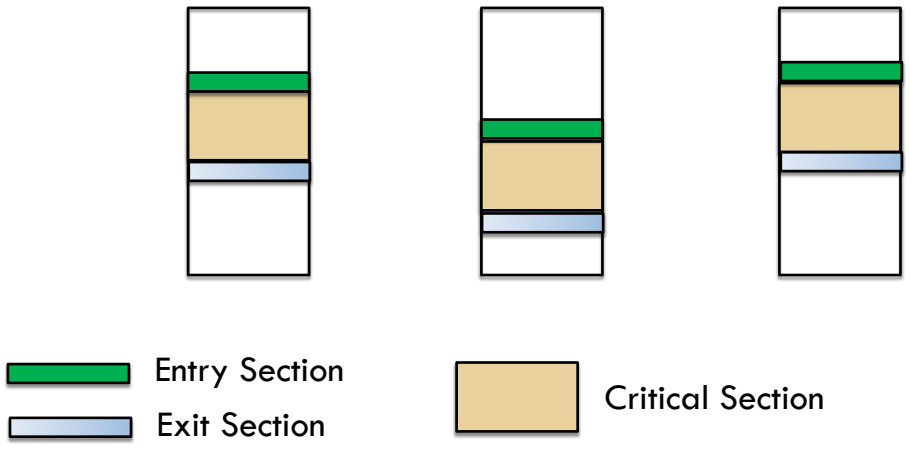
- Multiple computers
- Computers that control spaceship must agree on several conditions
  - E.g., Status: Proceed or abort mission
- Coordinate access to shared resources
  - Sensors, actuators, etc.

DISTRIBUTED MUTUAL EXCLUSION

Distributed processes often need to coordinate their activities

- If a collection of processes share a set of resources **mutual exclusion** is needed to:
  - Prevent interference
  - Ensure consistency
- This is the critical section problem in OS

Critical section problem



Distributed mutual exclusion

- Extension to distributed systems of the familiar problem of avoiding race conditions
  - In kernels and multi-threaded applications
- Shared variables or facilities provided by a local kernel cannot be used to solve this
- Solution must be based *solely* on **message passing**

Distributed mutual exclusion

- Consider a set of N processes p_i $i=1, 2, \ldots, N$
  - These **do not share variables**
- Processes access common resources
  - They do so in a critical section

SUMMARY OF APPROACHES

Approaches to distributed mutual exclusion

- Token-based solutions
- Permission-based solutions

Token-based solutions

- Mutual exclusion is achieved by *passing* a special message (**token**) between the processes
- There is only one token
  - Whoever has that token is allowed to access shared resource
- When finished, token is passed to another process

Token-based solutions: Advantages

- Depending on how processes are organized, fairly easy to avoid starvation
- Deadlocks can also be avoided

Token-based solutions: Disadvantages

- When the token is **lost** – for e.g., process holding the token crashes, complex actions need to be taken
- After a failure, intricate distributed process needs to be initiated
  - Ensure that a ***new*** token is created
  - But above all, make sure that that is the ***only*** token

Permission-based solutions

- Process wanting to access resource first requests **permission** from *other* processes
- Many different ways to granting this permission

Structural considerations for the solution

- With a central server
- Without a central server
  - Peer processes must coordinate their accesses to shared resources
  - Occurs routinely on Ethernets and IEEE 802.11 wireless
    - Network interfaces cooperate as peers so that only one node transmits at a time on the shared medium
    - Ethernet: Method of operation "Carrier Sensing, Multiple Access with Collision Detection" or CSMA/CD
    - Wireless: "Carrier Sensing, Multiple Access with Collision Avoidance" CSMA/CA

ASSUMPTION & REQUIREMENTS

Assumptions in our algorithms

- The system is asynchronous
- Processes do not fail
- Message delivery is reliable
  - Delivered eventually and exactly-once

Application level protocol for entering the critical section

- `enter()`
  - **Block** if necessary
- `resourceAccesses()`
  - **Access shared resources** in the critical section
- `exit()`
  - Allow other processes to enter

Requirements for distributed mutual exclusion

- **ME1**: *At most one* process may execute in the critical section at a time
  - **Safety**
- **ME2**: Requests to enter and exit the critical section *eventually succeed*
  - **Liveness**: Freedom from **deadlocks** and **starvation**
- **ME3**: If one request *happened-before* another, then entry to the CS is granted in that order
  - **Fairness**

Evaluation of the algorithms

- **Bandwidth consumed**
  - Proportional to number of messages sent in each entry and exit operation
- **Client delay** incurred by process for each entry or exit operation
- Effect on **throughput** of the system
  - **Synchronization delay** between one process exiting critical section and next process entering it
  - Throughput is greater when synchronization delay is shorter

THE CENTRAL SERVER ALGORITHM

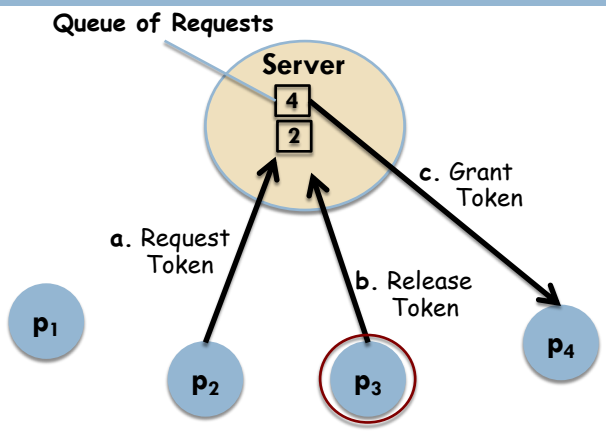
The central server algorithm

- Simplest way to achieve mutual exclusion
- Central server *grants authorization* to enter the critical section
- To enter a critical section, process sends request message to the server
  - Awaits reply from server
  - Reply constitutes **token** signifying authorization to enter critical section

Acquisition of token

- If no process holds the token?
  - Server replies immediately granting token
- If the token is held by another process?
  - Server does not reply, but *queues the request*
  - When that process exits the critical section, it sends a message giving server back the token
    - If the queue of waiting processes is non-empty, **server chooses oldest entry** in the queue and sends it the token

Server managing a mutual exclusion token



Evaluating the central server algorithm [1/2]

- Entering critical section
  - Requires 2 messages: `Request` followed by `grant`
  - Delay at the requesting process?
    - Round trip delay
      - There is also the *queuing delay* for messages residing in the queue
- Exiting the critical section requires one *release* message
  - Assuming asynchronous communications means that this does not delay the exiting process

Evaluating the central server algorithm [2/2]

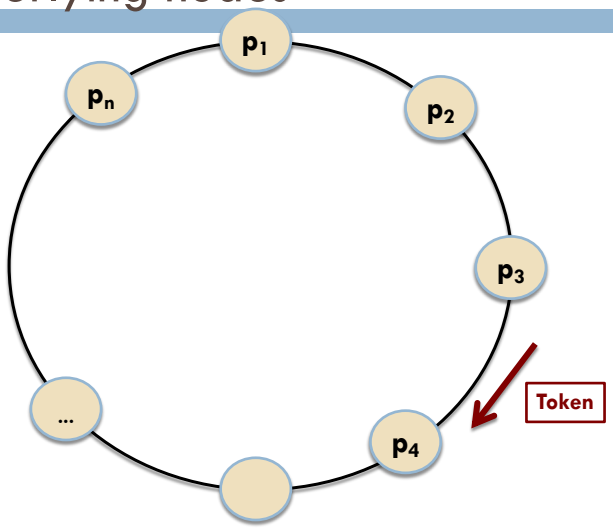
- Synchronization delay
  - `Release` message to server followed by `grant` to another process: Round trip time
- Server is a *performance bottleneck* for the system
  - Single point of failure as well

RING BASED ALGORITHM

Ring-based algorithm

- Arrange mutual exclusion between N processes **without** requiring an additional process
- Each process p_i has a communication channel to the next process in the ring, $p_{(i+1) mod N}$
- Exclusion is conferred by obtaining a token that is *passed from process to process* in a single direction around the ring
  - E.g. clockwise

Ring topology is unrelated to physical connections between underlying nodes



Acquisition of token

- When a process that does not need to enter critical section receives the token?
  - Immediately forwards token to its neighbor
- Process that requires token, **waits** until it receives it **and then retains it**
- To **exit** the critical section, process **sends token** to neighbor

Properties satisfied by the ring algorithm

- Satisfies **ME1** and **ME2**
- Token is not necessarily acquired in a happened-before manner (**ME3**)

Performance analysis [1/2]

- **Continuously** consumes network bandwidth (except when process is in critical section)
  - Processes send messages around ring even when no process requires critical section entry
- Delay experienced by process requesting entry to critical section?
  - 0: when it has just received the token
  - N messages when it has just passed on the token

Performance analysis [2/2]

- Exit from critical section
  - Requires only 1 message
- Synchronization delay between one process' exit and another process' entry into critical section
  - Anywhere between 1 and N message transmissions

The contents of this slide set are based on the following references

- Distributed Systems: Concepts and Design. George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair. 5th Edition. Addison Wesley. ISBN: 978-0132143011 [Chapter 15]