Question 1. Mutual Exclusion:

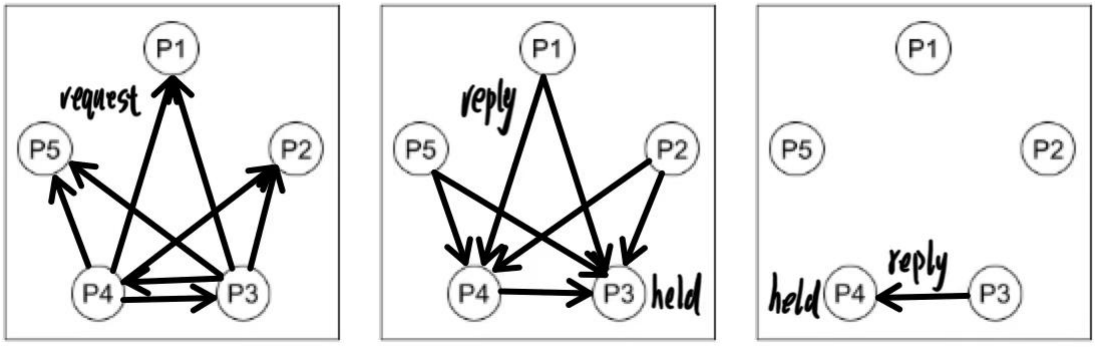
Recall the drone-based smart firefighting use case from Homework 1. Given the risk of fires, suppose we now have drones regularly inspecting wildfire-prone areas in the Angeles National Forest. These drones capture and process video to detect the possible onset and spread of fires. Given the remote nature of the locations and limited power availability for drones to recharge, CalFire has stationed drone recharge stations, i.e. docking stations where the drones conducting inspections can dock and recharge. To avoid collisions and interference with each other's routes, drones operate in a single region and use message passing to coordinate mutually exclusive access to the docking station in that region.

Let us consider 5 processes, each on a different drone (P1, P2, P3, P4, and P5 respectively) coordinating access to the single local docking station in that region. The drones use the Ricart-Agrawala Algorithm1 **(non-token based)** to request access to the single resource (docking station) when they are close to running out of power.
Note: At any instance, the docking station can only charge a single drone, i.e. use of the docking station is executed in a critical section.
The following requests are issued to obtain mutually exclusive access to the docking station:

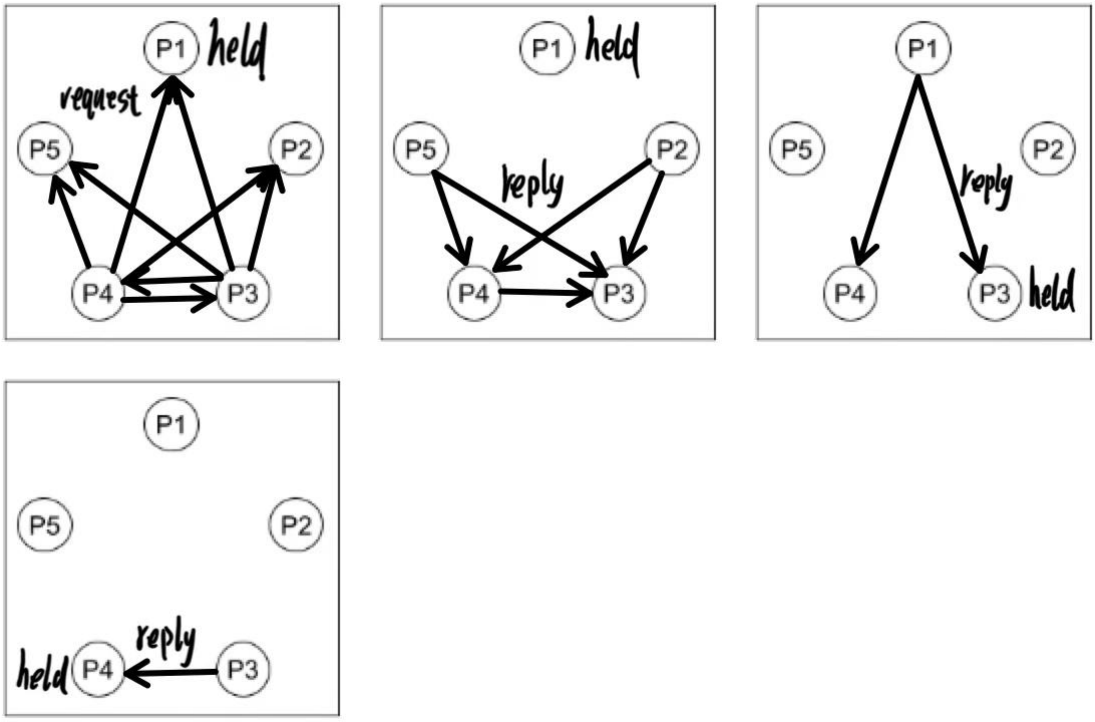
P1: Request at timestamp 50
P4: Request at timestamp 65
P3: Request at timestamp 65

Assume that P3 and P4 requests are initiated simultaneously, request messages are delivered in negligible time (i.e. 0) and no messages are lost.

(a) Show an execution path where P1 has completed executing in its critical section when it simultaneously receives the request from P3 and P4. Illustrate below how processes obtain access to the docking station by showing the ordered flow of Request and Reply messages. (Use as many template boxes as you need):



(b) Show an execution path where P1 is still executing in its critical section when it simultaneously receives the request from P3 and P4. Illustrate below how processes obtain access to the docking station by showing the ordered flow of Request and Reply messages. (Use as many template boxes as you need):



(c) How many messages of each type (Request, Reply) are required in total to satisfy all the requests in question (a)?

Request: 4+4=8
Reply: 4+4=8

(d) What will happen in case (a) if drone1 runs out of power as it reaches the docking station, i.e. P1 experiences a failure at the instant the P3/P4 requests are initiated?

P1 will not send Reply messages to P3 and P4, P3 and P4 will wait indefinitely for a Reply from P1, which will result in a deadlock because P3 and P4 cannot proceed without receiving a Reply from P1, and P1 is unable to send the Reply due to its failure.

Question 2. Mutual Exclusion (token-based)

Another firefighting team in a different region of the Angeles National Forest is using drones that employ the **Token-based Ricart-Agrawala Second Algorithm.** This team has 5 drones/processes and one docking station, similar to the situation in Question 1.
The following requests are issued for access to the critical section:

P2: Request at timestamp 70
P1: Request at timestamp 88
P4: Request at timestamp 99

Assume the request messages are delivered in negligible time (i.e. 0) and no messages are lost.
Assume the token is initially at P2 and its recorded timestamps for requests from all processes are 0.
P2 enters the critical section (i.e. charging at the docking station) since it has the token and **completes executing in its critical section at timestamp 120.**
Note: Follow the algorithm shown in the lecture slides to answer the questions below.

(a) Write out the sequence in which the token is passed among the processes,

P2->P1->P4

(b) What is the total number of messages required for all requests to be satisfied?

Request: 4+4=8
Token: 2
Total: 8+2=10

(c) Show the content of the token and the request queues at P1, P2, and P4 when all requests are satisfied.

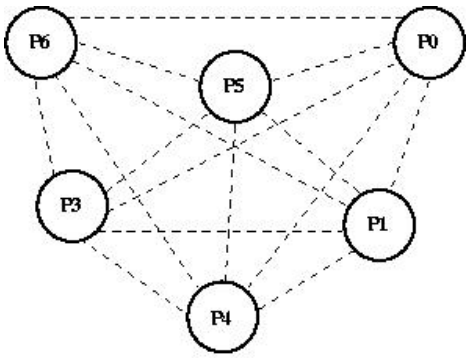
token = [88, 70, 0, 99, 0]

Each process maintains a queue of outstanding requests that have not yet been satisfied. After all requests are processed, the queues should be empty.

requestP1 = requestP2 = requestP4 = none

Question 3. Leader Election:

A firefighting team in San Diego County is using a centralized coordinator scheme to handle drone-based sensing. There are 6 drones (processes P1 P6) that can all communicate with each other and coordinate through a single leader. Suppose that the drone running Process P6 is the leader (as it has the highest number).



The drones are sent to monitor a fire. At some later point in time, the drone running Process P6 flies into a tree and breaks. The drone running Process P3 notices this, suspects a failure and initiates the Leader Election process.

(a) Show all the steps taken to elect a new leader using the **Bully Leader Election Algorithm**.

1. Failure Detection:
   P3 detects that P6 (the current leader) has failed and initiates an election.

2. Election Initiation:
   P3 sends Election messages to all processes with a higher identifier: P4, P5, and P6.

3. Responses from Higher-Id Processes:
   P4 receives the Election message from P3:
   P4 replies with an Answer message to P3.
   P4 starts its own election by sending Election messages to P5 and P6.
   P5 receives the Election message from P3:
   P5 replies with an Answer message to P3.
   P5 starts its own election by sending an Election message to P6.

4. Election at P4:
   P4 sends Election messages to P5 and P6.
   P5 receives the Election message from P4:
   P5 replies with an Answer message to P4.
   P5 starts its own election by sending an Election message to P6.
   P6 has failed, so no response is received.

5. Election at P5:
   P5 sends an Election message to P6.
   P6 has failed, so no response is received.

6. Declaring the Coordinator:
   P5 does not receive any Answer messages from higher-id processes (P6 is down).
   P5 declares itself the coordinator and broadcasts a Coordinator message to all processes (P1, P2, P3, P4, and P5).

7. Election Completion:
All processes receive the Coordinator message from P5 and recognize P5 as the new leader.

(b) How many messages are sent in total in the above execution of the Bully Leader Election Algorithm?

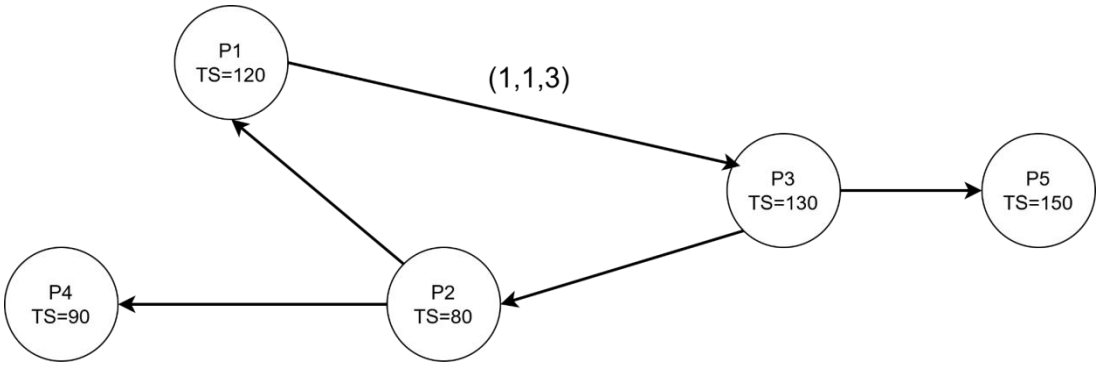
Election messages: 3+2+1=6
Answer messages: 2+1=3
Coordinator messages: 5
Total: 6+3+5=14

(c) How many messages would be sent in total if Process P1 noticed the failure of Process P6 instead of Process P3?

Election messages: 5+4+3+2+1=15
Answer messages: 4+3+2+1=10
Coordinator messages: 5
Total: 15+10+5=30

Question 4. Distributed Deadlocks

To monitor changing weather conditions, there are multiple environmental monitoring stations installed around the forest region. Each such station is equipped with a variety of sensing capabilities. Consider 5 such stations, each of which is equipped with a range of resources. Processes obtain access to resources that they must use in a mutually exclusive manner. Consider the following Wait-For Graph, with 5 processes P1, P2, P3, P4, and P5 the links indicate the respective dependencies. The "TS" value in a process's circle indicates the timestamp associated with when the process was created.



(a) Suppose process P1 initializes a deadlock detection using the **Chandy-Misra-Haas Algorithm**. Draw all the probe messages sent by each process. **Hint:** The probe message sent by P1 is (1,1,3).

P1->P3:(1,1,3)

P3->P5:(1,3,5)

P3->P2:(1,3,2)

P2->P1:(1,2,1)

P2->P4:(1,2,4)

(b) If we use the **Wait-Die Algorithm** to prevent deadlocks, which processes remain alive in the end?

P1, P2, P4, P5

(c) If we use the **Wound-Wait Algorithm** to prevent deadlocks, which processes remain alive in the end?

P2, P5

Question 5. Paper summaries:

Summarize **1** paper selected from the *Distributed Operating Systems* papers in the reading list.

Guidelines for paper summaries:

1. Each should be about 1 page of text (suggested size 10-11 pt., single-spaced, 1-inch margins).
2. Summaries should provide the following information about the paper in your own words:
    a. The main contributions of the paper (the key problem(s), proposed techniques, and approaches).
    b. Critique of approach, its advantages, and its limitations.

Summary of "Implementing Remote Procedure Calls" by Andrew D. Birrell and Bruce Jay Nelson

Main Contributions:

The paper presents the design and implementation of a Remote Procedure Call (RPC) system, which extends the familiar concept of local procedure calls to distributed systems, enabling communication across a network. The authors outline the key challenges in designing an RPC system, including handling machine and communication failures, managing address spaces, and ensuring data integrity and security. The paper describes the overall structure of their RPC mechanism, focusing on the binding process, transport-level communication protocol, and performance optimizations.

The RPC system is built around the concept of stubs, which act as intermediaries between the caller (client) and the callee (server). The user-stub and server-stub are automatically generated by a tool called Lupine, which simplifies the process of writing distributed applications. The system uses the Grapevine distributed database for binding, allowing clients to locate servers dynamically. The transport protocol is designed to minimize overhead, with optimizations such as implicit acknowledgments, lightweight connection management, and efficient handling of process switches.

The authors also discuss the importance of security in RPC, introducing encryption-based mechanisms to ensure data integrity and authentication. They provide performance measurements, demonstrating that their RPC system achieves high efficiency, with minimal overhead compared to local procedure calls.

Critique of Approach:

Advantages:

Simplicity and Familiarity: By leveraging the well-understood semantics of local procedure calls, the RPC system provides a straightforward and intuitive way to build distributed applications. This reduces the complexity of writing networked programs, making it accessible to a broader range of developers.

Efficiency: The authors focus heavily on performance, achieving low latency and high throughput. The use of implicit acknowledgments, lightweight connection management, and optimized process handling reduces the overhead typically associated with remote communication.

Dynamic Binding: The use of the Grapevine distributed database for binding allows clients to dynamically locate servers, providing flexibility and scalability. This is particularly useful in environments where services may be replicated or moved across different machines.

Security: The inclusion of encryption-based security mechanisms is a significant advancement, addressing the need for secure communication in open networks. This ensures that data integrity and authentication are maintained, which is critical for distributed systems.

Limitations:

Complexity of Implementation: While the RPC system simplifies the development of distributed applications, the underlying implementation is complex. The need for automatically generated stubs, a distributed database for binding, and a custom transport protocol requires significant infrastructure, which may not be feasible in all environments.

Scalability Concerns: The paper does not extensively address the scalability of the system in very large or highly dynamic networks. While the use of Grapevine for binding works well in their research environment, it may face challenges in larger, more heterogeneous networks.

Limited Error Handling: The RPC system assumes that failures are rare and does not provide extensive mechanisms for handling partial failures or retries. This could be problematic in less reliable networks or in scenarios where servers may crash frequently.

Performance Trade-offs: While the system is optimized for simple calls, it may not be as efficient for bulk data transfers or long-running operations. The authors acknowledge that their protocol sends more packets than necessary for large data transfers, which could be a limitation in certain use cases.