Communications in Distributed Systems

Shubhangi Shreya
Assistant Professor(I)
School of Computer Engineering
KIIT Deemed to be University
Bhubaneswar

UNIT-3: Synchronization and Processes

- Clock Synchronization
- Clock Synchronization Algorithms
- Mutual Exclusion Algorithms
- Election Algorithms
- Atomic Transactions & Modeling
- Atomic Transaction Implementation
- Concurrency Control Algorithms in Atomic Transaction

Some critical challenges and principles of synchronization in distributed systems

1.Information Scattered Across Multiple Machines:

- Decentralization: In distributed systems, data and control are distributed across many machines. Algorithms must be designed to work without gathering all the data at a single point, avoiding bottlenecks and single points of failure.
- Local Decision Making: Processes rely on information that is available locally and communicate with other processes to coordinate. This can involve passing messages or synchronizing state without central oversight.

2.Decisions Based on Local Information:

- Autonomy: Each process or node operates based on its local view of the system. This local information may include the state of the process itself and recent communications with other processes.
- Consistency Models: Distributed systems often use consistency models (like eventual consistency) to ensure that, despite local decision-making, the system as a whole remains consistent over time.

3.Avoiding Single Points of Failure:

- Fault Tolerance: Distributed systems are designed to tolerate failures by ensuring that the failure of one machine does not bring down the entire system. This is achieved through redundancy, replication, and robust failure recovery mechanisms.
- Failover Mechanisms: Techniques such as leader election and backup nodes ensure that if one node fails, another can take over its responsibilities.

4.Lack of Global Clock or Precise Time Source:

- Logical Clocks: Since global clocks are impractical, distributed systems use logical clocks (like Lamport timestamps) to order events and ensure that the system can track causality and consistency without synchronized time.
- Vector Clocks: For a finer granularity of event ordering and to track causality more precisely, vector clocks can be used. They provide a way to capture the partial ordering of events in a distributed system.

Centralized Resource Allocation: Problems

1.Heavy Burden on a Single Process:

- Scenario: In a centralized approach, all resource requests (e.g., for I/O devices) are sent to a single manager process. This manager is responsible for examining these requests, making decisions, and allocating resources.
- Problem: In a large distributed system, this manager would be overwhelmed by the sheer volume of requests and information it needs to handle. This centralized process becomes a bottleneck because it must manage all resource requests from potentially thousands of other processes. As the system scales up, the load on this single process becomes impractical and inefficient.

2.Single Point of Failure:

- Scenario: If the centralized manager process fails, it can result in significant disruptions. All resource allocation decisions depend on this one manager, so its failure can bring the whole system to a halt. For example, if the resource allocator crashes, none of the processes can get the resources they need, causing a cascade of failures or blocking.
- Problem: This makes the system unreliable because the failure of one component (the manager) affects the entire distributed system. Ideally, a distributed system should continue to function even if some of its components fail. If a single point of failure can cause widespread disruption, it contradicts the goal of high reliability and fault tolerance.

Challenges in a Distributed System:

- **Inconsistent Timestamps**: In a distributed environment where multiple machines may be involved, each machine could have its own local clock. If the clocks are not synchronized perfectly, the timestamps on files could be inconsistent or misleading. For example, one machine might think a file was modified at a certain time, while another might see a different timestamp due to clock discrepancies.

- **Inconsistent Build Results**: If make were running across a distributed system, inconsistent timestamps could lead to incorrect build decisions, such as not recompiling a file that should be updated or unnecessarily recompiling files that haven't changed.

Logical Clocks

Introduction to Computer Timers

- Most computers come equipped with a circuit referred to as a clock, although a more accurate term would be a "timer." A computer timer typically consists of a precisely machined quartz crystal that oscillates at a specific frequency based on its characteristics. This frequency determines how the timer functions.

Structure of a Computer Timer

- A computer timer usually has two registers: a counter and a holding register. Each oscillation of the quartz crystal decreases the counter by one. When the counter reaches zero, an interrupt is generated, and the counter is reloaded from the holding register. This setup allows the timer to generate interrupts at any desired frequency, such as 60 times per second. Each interrupt is called a clock tick.

Initialization and Updating of the Software Clock

- Upon system boot, the operator typically enters the current date and time, which is then converted to the number of ticks since a known starting date and stored in memory. The system clock is updated at every clock tick by incrementing this stored time.

Single vs. Multiple Clocks

- For a single computer with a single clock, slight inaccuracies are generally inconsequential as all processes use the same clock, maintaining internal consistency. Problems arise with multiple CPUs, each having its own clock. Even though crystal oscillators are stable, variations in their frequencies cause clocks to drift apart, leading to clock skew. This discrepancy can cause issues in programs relying on correct and consistent timestamps across different machines.

Logical Clocks vs. Physical Clocks

- For many purposes, internal consistency of clocks suffices, even if the time does not match real-world time exactly. Clocks in this context are called **logical clocks.**
- When clocks must align closely with real-world time, they are referred to as **physical clocks.**

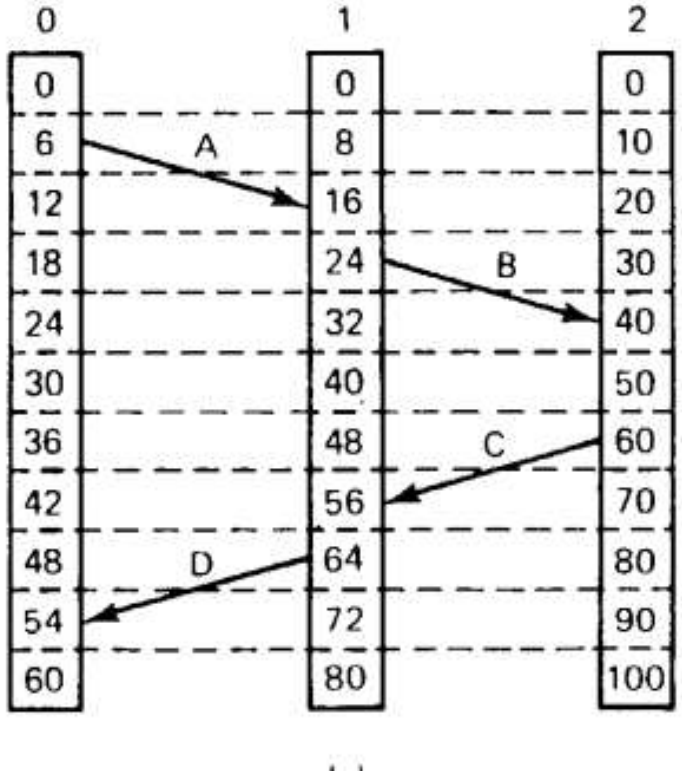
Lamport's Happens-Before Relation

To synchronize logical clocks, Lamport introduced the happens-before relation, denoted as "a happens before b" (a → b). This relation is observed in two scenarios:

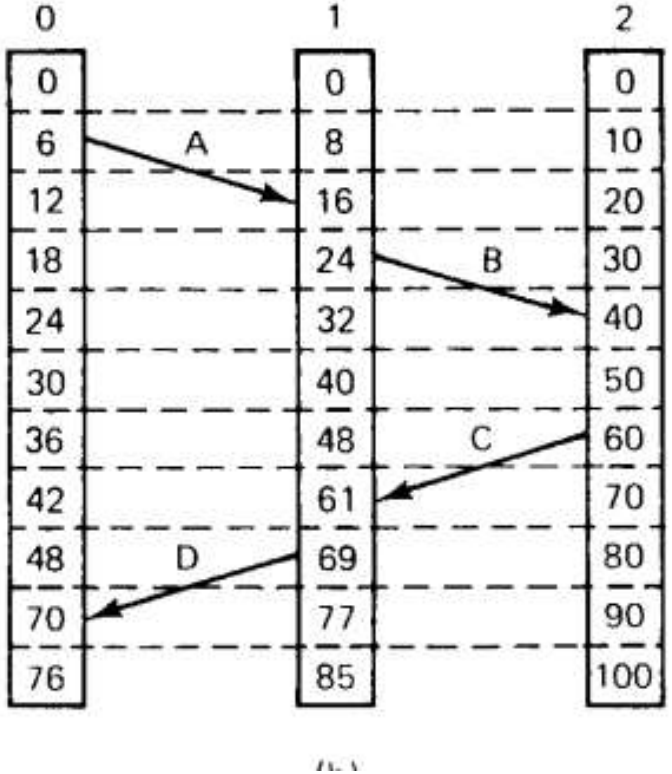
- If events a and b occur in the same process and a happens before b, then a → b.
- If event a is the sending of a message by one process and event b is its receipt by another process, then a → b since a message cannot be received before it is sent.

The happens-before relation is transitive, meaning if a → b and b → c, then a → c. If events x and y occur in different processes without exchanging messages, they are concurrent, meaning neither x → y nor y → x holds.

Lamport solution



a)clock run at different rate



b)Lamport algo correct the clocks

Way to assign times to events in a distributed system

- If a happens before b in the same process, then C(a) < C(b).
- If a and b are the sending and receiving of a message, then C(a) < C(b).
- All events a and b must have unique times, C(a) ≠ C(b).

Physical Clocks

Importance of Physical Clocks

While Lamport's algorithm provides an unambiguous ordering of events, the assigned time values do not necessarily reflect the actual occurrence times. In systems where real-time accuracy is critical, such as in real-time systems, external physical clocks are essential. Having multiple physical clocks enhances efficiency and redundancy, but it introduces two significant challenges:

- Synchronizing these clocks with real-world time.
- Synchronizing the clocks with each other.

Measurement of Time

Accurate time measurement is complex. Historically, time measurement was astronomical, based on the solar day—the interval between two consecutive transits of the sun, which reaches its highest point in the sky at noon each day. A solar second was defined as exactly 1/86400th of a solar day.

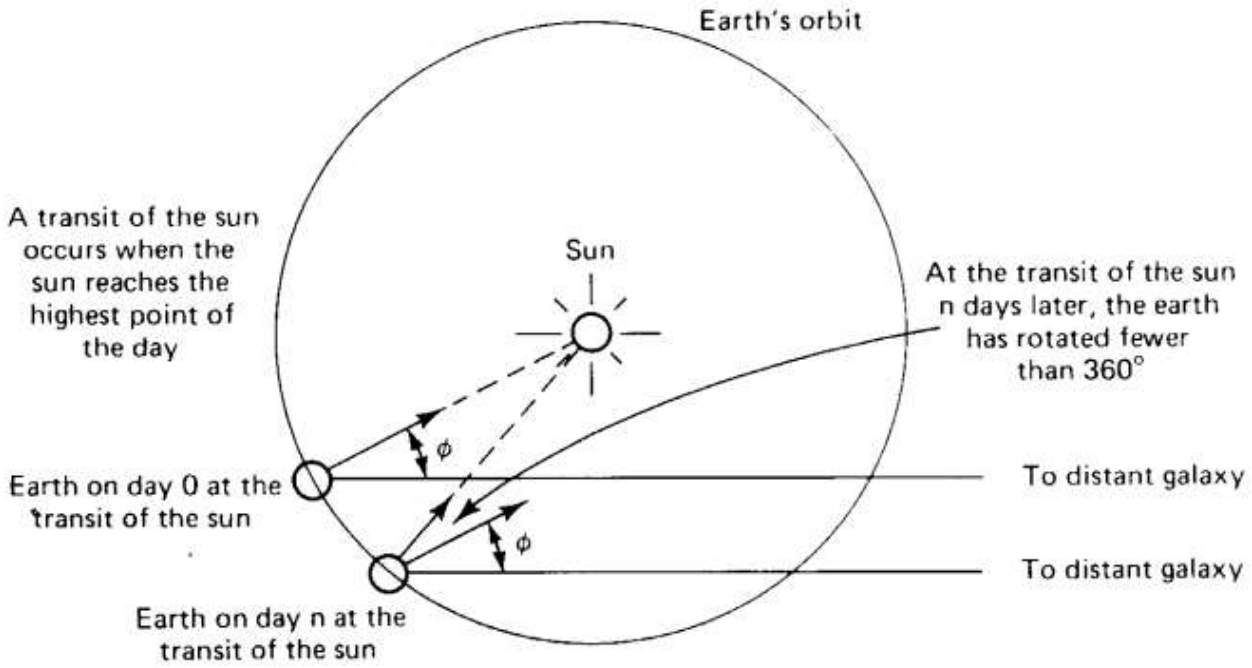
Variations in Earth's Rotation

In the 1940s, it was discovered that the Earth's rotation period is not constant due to tidal friction and atmospheric drag. Geological studies suggest that 300 million years ago, there were about 400 days in a year, indicating that the Earth's rotation has slowed down. Additionally, short-term variations in day length occur due to turbulence in the Earth's core.

Mean Solar Second and Atomic Clocks

Astronomers calculated the mean solar second by averaging the length of many days. With the invention of the atomic clock in 1948, timekeeping became more precise. The atomic second was defined based on the cesium 133 atom's transitions, making the atomic second equal to the mean solar second at the time of its introduction.

Computation of the mean solar day



International Atomic Time (TAI)

- Approximately 50 laboratories worldwide use cesium 133 clocks to measure time. These laboratories report their clock ticks to the Bureau International de l'Heure (BIH) in Paris, which averages them to produce International Atomic Time (TAI). TAI is the mean number of cesium 133 clock ticks since January 1, 1958, divided by 9,192,631,770.

Leap Seconds and Universal Coordinated Time (UTC)

- Due to the Earth's slowing rotation, 86,400 TAI seconds are now about 3 milliseconds shorter than a mean solar day. To prevent discrepancies, BIH introduces leap seconds when the difference between TAI and solar time reaches 800 milliseconds. This adjustment maintains synchronization with the sun's apparent motion, resulting in Universal Coordinated Time (UTC), the modern civil timekeeping standard.

Distribution and Accuracy of UTC

- UTC is disseminated through various means, including shortwave radio stations like WWV in Fort Collins, Colorado, and MSF in Rugby, Warwickshire. These broadcasts offer UTC with an accuracy of about ±1 millisecond, though atmospheric conditions can reduce practical accuracy to ±10 milliseconds. Satellites also provide UTC services with high accuracy, requiring precise knowledge of sender-receiver positions to account for signal propagation delays.


Q&A

Homework Questions

1) What is synchronization in distributed system?
2) What is clock synchronization?
3) Logical clock vs Physical clock.
4) Name at least three sources of delay that can be introduced between WWV broadcasting the time and the processors in a distributed system setting their internal clocks.
5) Consider the behavior of two machines in a distributed system. Both have clocks that are supposed to tick 1000 times per millisecond. One of them actually does, but the other ticks only 990 times per millisecond. If UTC updates come in once a minute, what is the maximum clock skew that will occur?

Thank You!

Synchronization and Processes

Dr. Namita Panda
Associate Professor
School of Computer Engineering
KIIT Deemed to be University
Bhubaneswar

Clock Synchronization Algorithms

- Clock Synchronization
- Clock Synchronization Algorithms
- Mutual Exclusion Algorithms
- Election Algorithms
- Atomic Transactions & Modeling
- Atomic Transaction Implementation
- Concurrency Control Algorithms in Atomic Transaction

Basic Concepts:

Goal:

1.If one machine has a WWV(UTC provider used by National Institute of Standard Time(NIST) receiver ,then to keep all other machines synchronized .

2.If no machine have WWV receiver, then each machine keeps track of its own time as well as keep all other machines together w.r.t the average time stamp.

Basic Concepts cont..

Assumtions:

- Each machine hava a timer and its interrupt is *H* times a second.
- When the timer goes off,the interrupt handler adds 1 to a s/w clock(value #C),which keeps track of the number of interrupts.
- When UTC(Universal Coordinated Time) time is *t*, the value of the clock on machine *p is* $C_p(t)$.
- In Perfect world

$$C_p(t)=t \text{ for all } p \text{ and } t$$

OR

$$dC/dt=1$$

Basic Concepts cont..

- Practically real timers do not interrupts exactly *H* times a second.
- If *H*=60 then no of ticks per hour 216,000
- As modrn timer chips have error about 10^{-5},so number of ticks are 215,998 to 216,002 per hour.
- By considering the constant p then

$$\mathbf{1-p <= dC/dt <=1+p}$$

(p is known as **maximum drift rate)**

Basic Concepts cont..

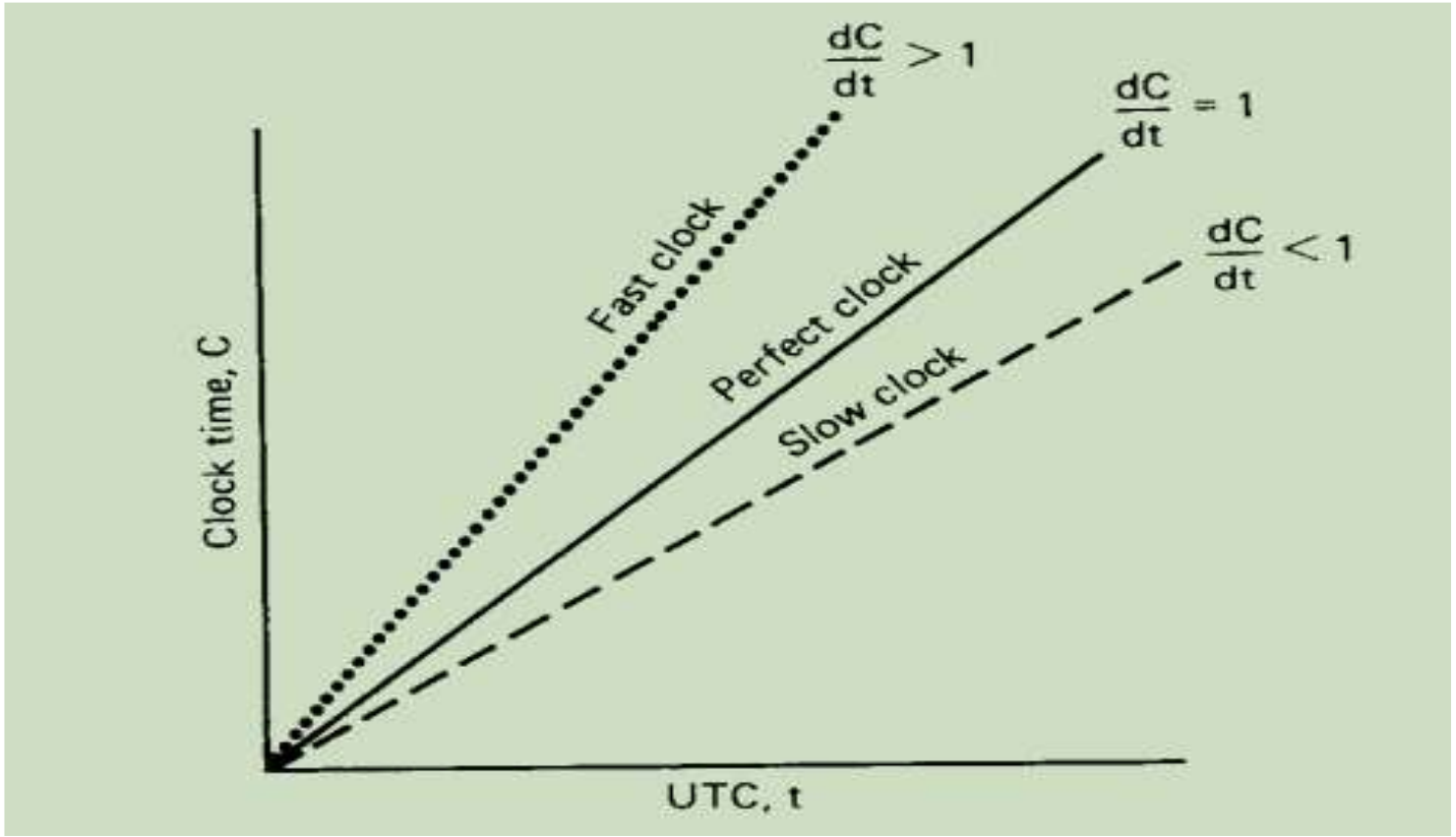


fig: Not all clocks tick precisely at the correct rate

Cristian's Algorithm

- One machine uses WWV receiver considered as **Time server.**
- Goal is to have all other machines stay synchronised.
- Periodically , each machine sends a request message to the time server asking it for the current time.
- Time server responds as fast as it can with a message containing its current time, C_{UTC} shown in following Fig.
- Let **d** is the estimate of delay from the time server to the sender which is also known as **message propagation time.**
- So sender should set its clock to $\mathbf{C_{UTC}+d}$

Cristian's Algorithm cont..

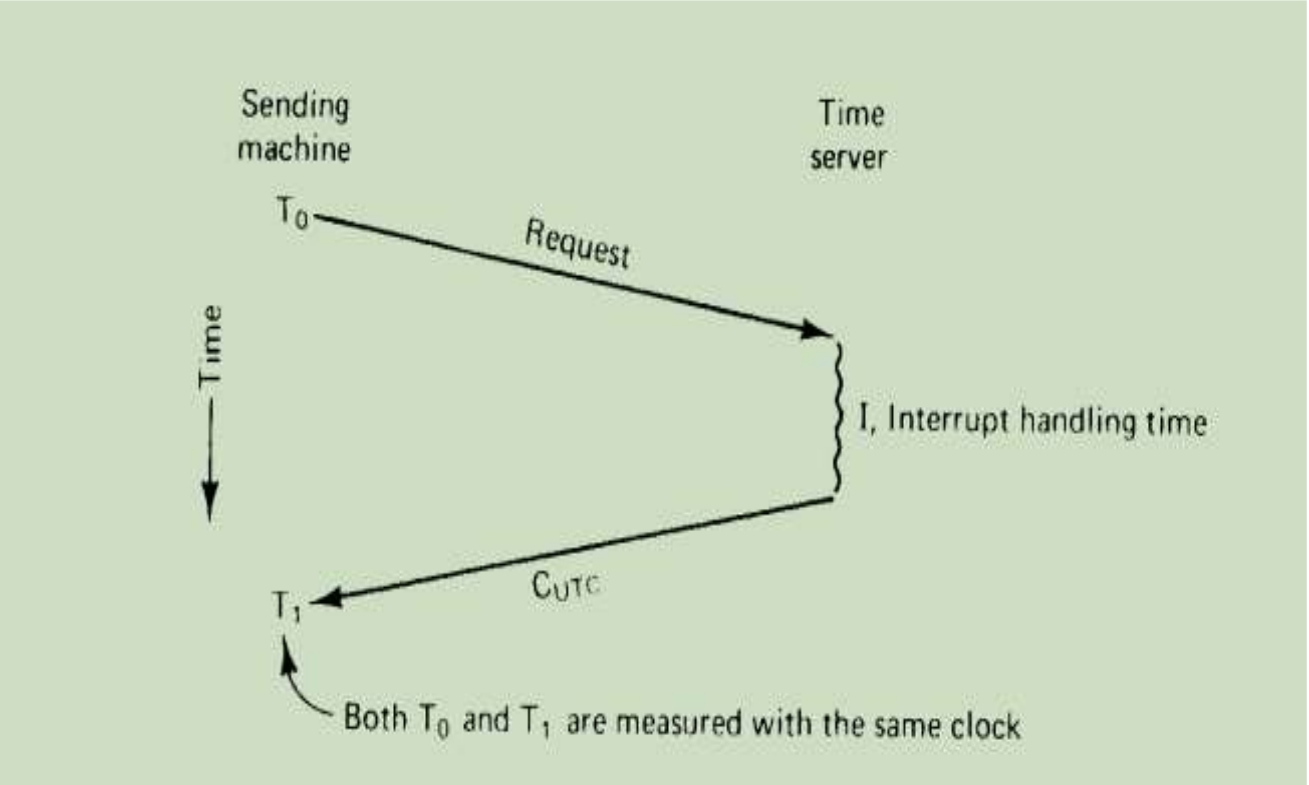


Fig:Getting the current time from the time server

Cristian's Algorithm cont..

- If nothing is known about **I**(interrupt handling time),

  esimated **$d=(T_1-T_0)/2$**

  where starting time T_0 and ending timeT_1
- If **I** is known ,then estimated **$d=(T_1-T_0-I)/2$**

OR

- **d** may be estimated using **average or minimum** of multiple requests.

Cristian's Algorithm cont..

PROBLEMS OF Cristian's Algorithm:

1.(Major problem)

Time must never run backward.

If the sender's clock is fast, C_{utc} will be smaller than the sender's current value of C which is serious problem.

SOLUTION:

Such change must be introduced gradually,instead of jumping it forward all at once.

EXAMPLE:Suppose the timer is set to generate 100 interrupts per second ,each interrupt would add 10msec to the time and when slowing down,the interrupt routine adds 9msec each time,until the correction has been made.

Cristian's Algorithm cont..

2. (Minor problem)

It takes a nonzero amount of time for the time server's reply to get back to the sender.

SOLUTION:

Sender must record accurately the T_0 and T_1 which are measured using the same clock, so the interval will be relatively accurate.

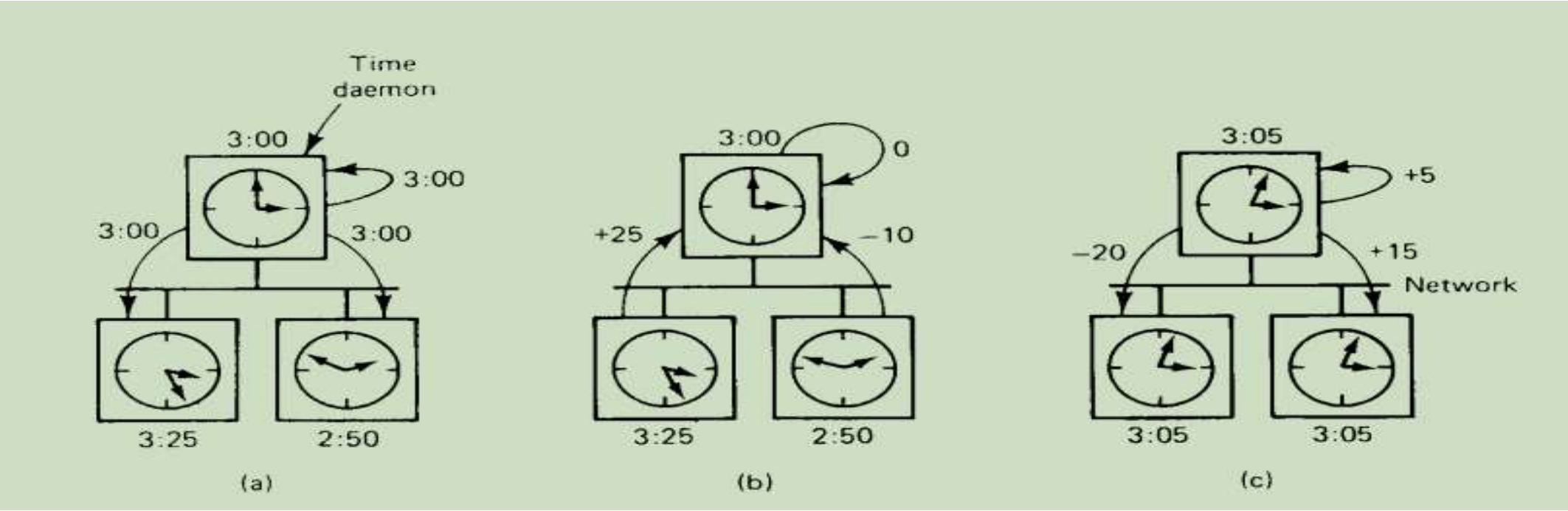
Berkeley Algorithm

- In Cristian's algorithm time server is passive.
- In Berkeley Algorithm time server is active known as Time Daemon.
- In this case no machine has a WWV receiver.
- The time server periodically gathers time data from all other machines by polling technique and determine the average time. Then it communicates the average time (in fact the difference) to all machine to advance their clocks to the new time or slow their clocks down until some reduction has been achived.

Berkeley Algorithm cont..

- The time daemon's time must be set mannually by the operator periodically.

Fig(a)the time daemon asks all other machines for their clock value.
(b)The machines answer
(c)The Time daemon tells everyone how to adjust their clock



Averaging Algorithms:

- This algorithm is known as decentralised algorithm.
- Devide the time into fixed length resynchronization interval
- The *i*th interval starts at $T_0 + iR$ and runs until $T_0+(i+1)R$
- T_0 is agreed upen moment on past and R is system parameter.
- At the beginning of each interval, every machine **broadcasts the current time** according to its clock as different clock has different speed.
- Then it **starts a local timer to collect all other broadcasts** during some interval *S*.
- The algorithm runs to **compute a new time from them by averaging the values from other machines**.

Averaging Algorithms cont...

Variations:

1.**Discard the *m* highest and *m* lowest values** and average the rest,as the extreme values consider as faulty clocks.

2. Collect each message **including its propagation time** from the sources.

Use of synchronized clocks

1. At most once message delivery:

- Traditional approch:
- Each messsage has an unique number and server stores all the numbers of message and can detect new messages from retransmission.
- Problem:
- If server crashes and reboots it loses its table of message numbers.
- Modified approch(With time):
- Each message carries a connection ID and timestamp.
- for each connection server records in table the most recent timestamp it has seen,if message for a connection is lower than the timestamp stored for that connection,is rejected.

Use of synchronized clocks cont..

2.**Clock based cache consistency**

- It is used in distributed file system.
- Traditional approch:
- Each client maintains local cache which is partitioned as read and write file.
- Problem :
- If a client has a file cached for reading,before another client can get a copy for writting,the the server has to first ask the reading client to invalidate its copy.
- Solution for the extra overhead can be eliminated by synchrinized clock

Use of synchronized clocks cont..

- Modified approch(With synchronized clock):
- When a client wants a file,it is given a **lease** on it which specifies how long the copy is valid.
- If the lease expires,the client can ask for it to be renewed.
- If the lease expires and the file is needed again shortly then the client can ask the server if the copy it has(identified by timestamp) is still the current one then a new lease is generated,but the file need not be retransmitted.

Q&A

QUESTIONS:

Q1.Consider the behavior of two machine in a distributed system .Both have clocks that areSupposed to tick 1000 time per milisecond. One of them actually does,but the other ticks only 990times per milisecond.If UTC updates come in once a minute,what is the maximum clock skew that will occur?

Q2.Discuss the problems and solutions of **Cristian's Algorithm.**

Q3.How diffrent clocks are synchronised using Berkeley Algorithm?

Q4.In the approch to cache consistency using leases,is it really essential that the clocks are synchronized?

Thank You!

Mutual Exclusion

Mrs. Swati Priyambada Satpathy
Assistant Professor
School of Computer Engineering
KIIT Deemed to be University
Bhubaneswar

UNIT-3:Synchronization in Distributed Systems

- Clock Synchronization
- Mutual Exclusion
- Election Algorithms
- Atomic Transactions
- Deadlocks in Distributed Systems

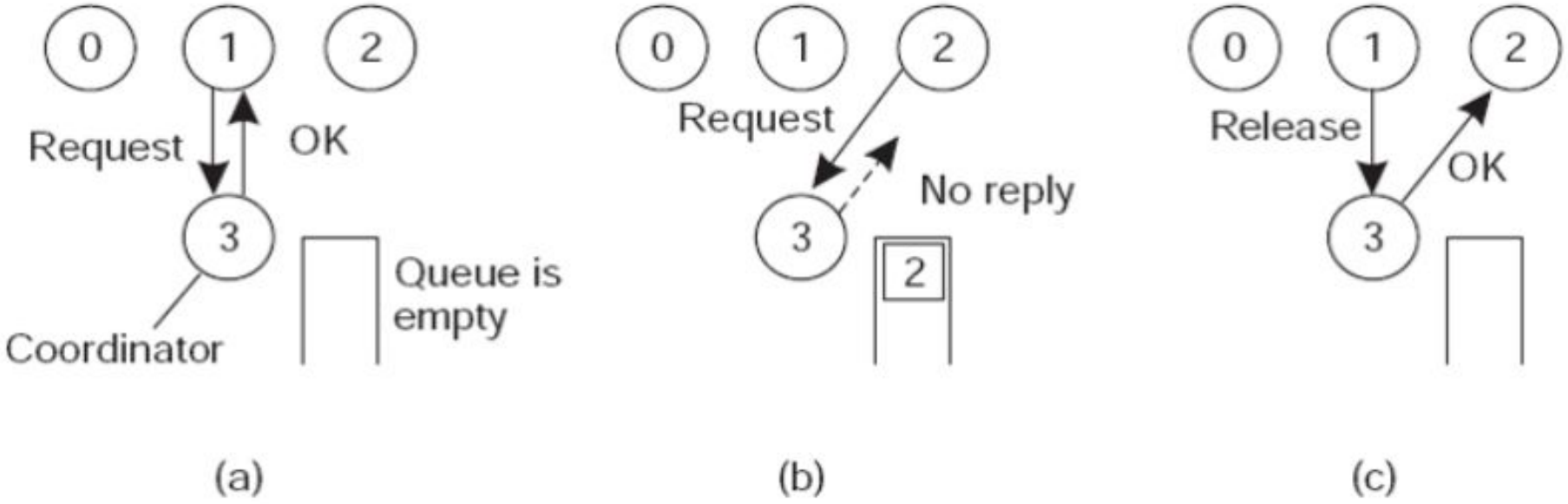
What is Mutual Exclusion ?

- When a process is accessing a shared resource, the process is said to be in a critical section (CS).
- It ensures that no two process can be in the same CS at the same time.
- In uniprocessor systems, CS's are protected using semaphores, monitors, and some similar constructs.
- Different algorithms based on message passing to implement mutual exclusion in distributed systems are

  - Centralized algorithms
  - Distributed algorithms
  - Token Ring algorithms

Centralized Algorithms: (1 Boss)

- One process is elected as the coordinator (highest network address).
- Whenever a process wants to access a shared resources, it sends request to the coordinator to ask the permission.
- Queue is maintained to track the request.



Contd..

Case 1:

Process 1 asked permission to Process 3 to access shared resources (as no other processes is currently in the CS) so, permission granted.

Case 2:

Process 2 asked permission to Process 3, to access some shared resources, but it is already full, so no reply and Process 2 has been pushed into the queue.

Case 3:

Process 1 released the CS, hence shared resources is free.

Dequeue has been done(popping of Process 2), and send OK for accessing it.

Advantages

- Mutual exclusion has been achieved.
- No starvation (as processes are queued, and no priority).
- Simple to use.
- 3 messages are enough (request, release, grant).

Disadvantages

- If the single system (co-ordinator) fails, the whole system will collapse.
- In large system, it is not feasible.

Distributed Algorithm

- Whenever a process wants to enter a CS, it builds a message containing the (name of the CS it wants to enter, its process number, and the current time) and sends to all processes, conceptually itself.
- When a process receives a request message from another process; the action it takes depends on its state with respect to the CS named in the message. Three cases have to be distinguished.

Case 1: If the receiver is not in the CS and doesn't want to enter, it sends back OK message to the sender.

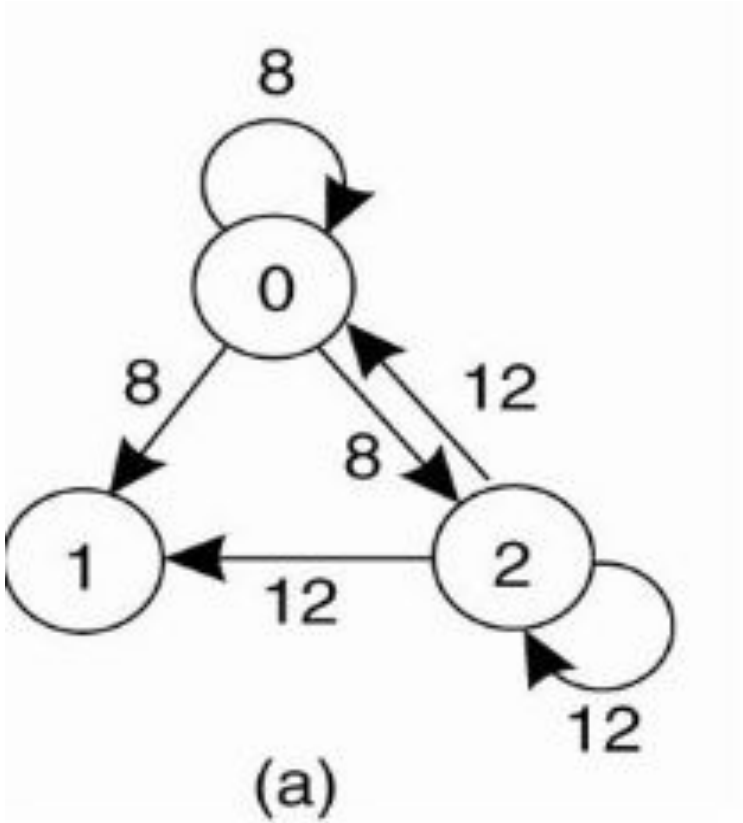
Case 2: If the receiver is already in the CS, it doesn't reply, instead it queues the request.

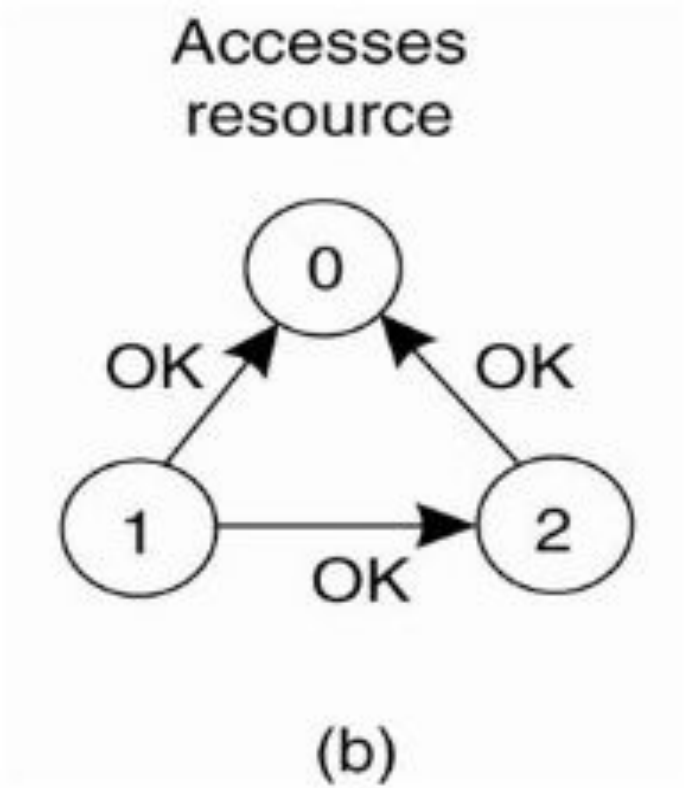
Contd…

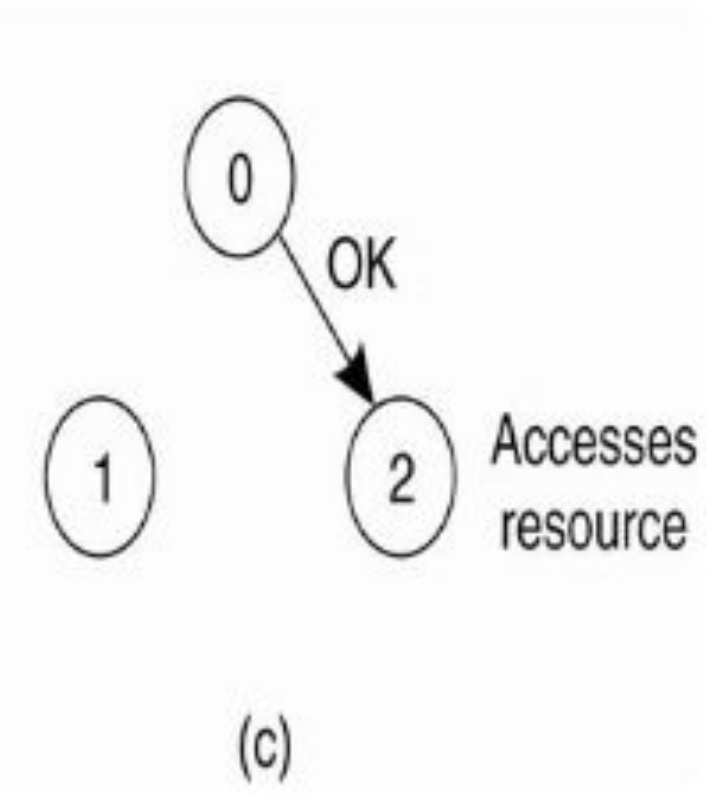
Case 3:

- If the receiver wants to enter the CS but not yet done.
- It compares the timestamp in the incoming message with the one contained in the message that it has sent to everyone.
- Whichever has lowest (wins), and the winner will not say OK, rather the looser will say OK, and the winner will continue in the CS.
- After exists the CS, it sends OK message to all processes on its queue and deletes them all from the queue.

Contd…


8
0
8
12
8
1
2
12
12
(a)


Accesses
resource
0
OK
OK
1
2
OK
(b)


0
OK
1
2
Accesses
resource
(c)

Contd…

a) Process 0 and Process 2 wants to enter the same CS at the same time.

b) Process 0 has the lowest time stamp, so its win.

c) When Process 0 is done, it sends OK also, so Process 2 can now enter CS.

Note:

- In centralized algorithm, mutual exclusion is guaranteed without deadlock and starvation.
- In distributed algorithm ,the number of messages required per entry is now 2(n-1), where the total number of process in the system is ‘n’, no single point failure exists.
- But unfortunately single point failure has been replaced by n points of failure. If any process crashes, it fails to respond to requests.

Contd…

- Therefore, when a request comes in, the receiver should always sends a reply, either granting or denying permission.

- Whenever either a request or a reply is lost, the sender times out and keeps trying until either a reply comes back or the sender concludes that the destination is dead.

- This algorithm is slower, more complicated, more expensive, and less robust than the original centralized one.

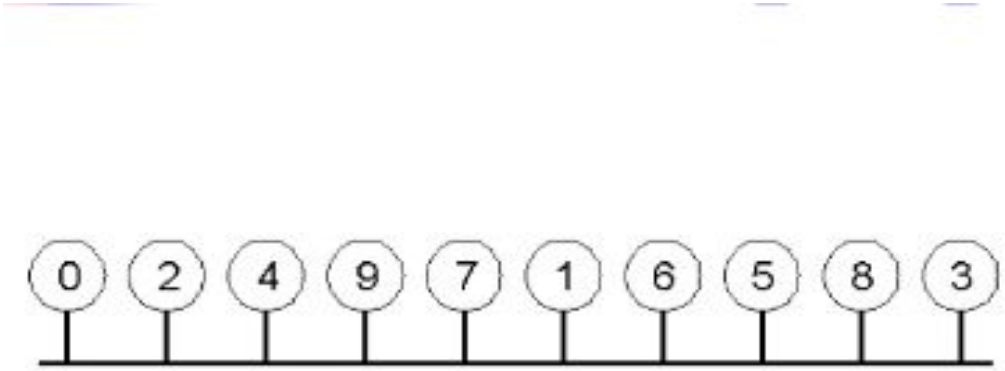
Token Ring Algorithm

- A logical ring is constructed in which each process is assigned a position in the ring.
- It doesn't matter what the ordering is, but each process knows who is next in line after itself.
- When the ring is initialized, process 0 is given a token.
- The token circulates among the ring, as it passed from process K to process K+1 (modulo the ring size) in point to point messages.
- When a process acquires the token, it checks whether it needs to enter CS, if yes, the process enters, does all the work, and leaves the CS.
- After exited, it passes the token along the ring.

Continue…

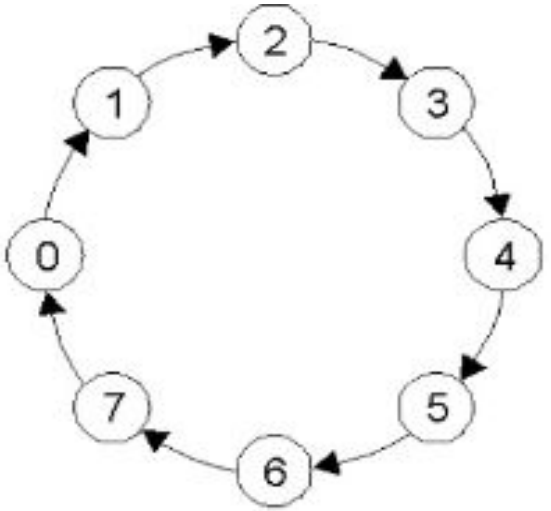
- Not allowed to enter the second CS, with the same token.
- If the process is handed the token by its neighbour but doesn't need it for CS, it just passes it along .
- So, when no process wants to enter any CS, the token just circulates at a speed around the ring.
- Only one process can have the token at any instant and only one process can be in a CS.
- No starvation.
- Once a process decides it wants to enter CS, at worst, it will has to wait for every other process to enter and leave one CS.

Continue…



(a)

An unordered group of processes on a network.



(b)

A logical ring constructed in software.

Disadvantages

- If the token is ever lost, it must be regenerated.
- Detecting that, it is lost is difficult, since the amount of time between successive appearance of token on the network is unbounded.
- It could not said, whether it is lost or it is still being used by some other processes.

Note:

- If the process crashes, it is little easier to recover as compared to other cases.
- A dead process will be detected by neighbour and can be removed from the group, and the token can be thrown to the next process in the queue.

Comparison of the three Algorithms

Algorithms	Message per entry/exit	Delay before entry (in message times)	Problems
Centralized	3	2	Coordinator crash
Distributed	2(n-1)	2(n-1)	Crash of any process
Token ring	1 to ∞	0 to n-1	Lost token, process crash

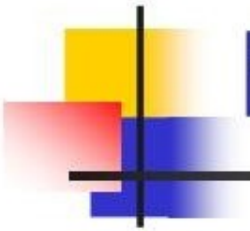
Q&A

Thank You!

Distributed Operating Systems

Election Algorithms

1. **A Bully Algorithm**
2. **A Ring Algorithm**



ELECTIONS

- There are at least two basic strategies by which a distributed system can adapt to failures.
  - operate continuously as failures occur and are repaired
  - The second alternative is to temporarily halt normal operation and to take some time out to reorganize the system.
    - The reorganization of the system is managed by a single node called the coordinator.
    - So as a first step in any reorganization, the operating or active nodes must elect a coordinator.

ELECTION AND SYNCHRONIZATION

- Similar
  - Like Synchronization, all processors must come to an agreement about who enters the critical region (i.e. who is the leader)
- Different
  - The election protocol must properly deal with the case of a coordinator failing. On the other hand, mutual exclusion algorithms assume that the process in the critical region (i.e., the coordinator) will not fail.
  - A new coordinator must inform all active nodes that it is the coordinator. In a mutual exclusion algorithm, the nodes not in the critical region have no need to know what node is in the region.

ELECTION ALGORITHMS

- The two classical election algorithms by Garcia-Molina
  - Bully Algorithm
  - Invitation Algorithm
- Ring Algorithm

The Bully Algorithm

- Assumptions
  - Each node has access to some permanent storage that survives node failures.
  - There are no transmission errors.
  - The communication subsystem does not fail

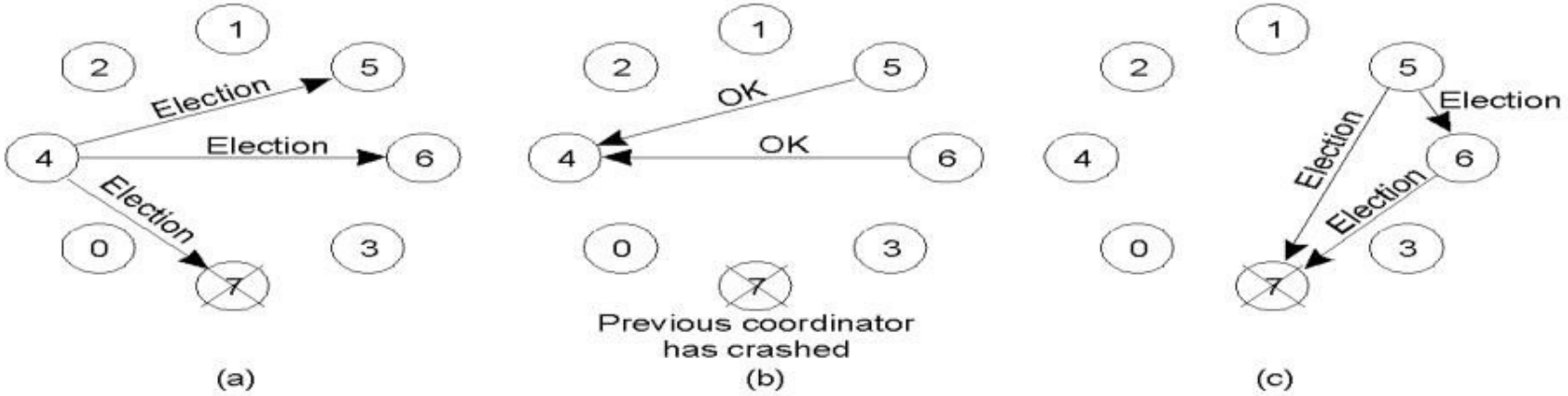
The Bully Algorithm

- State Information For Election
  - Status
    - Down,Election,Reorganization,Normal
  - Co-ordinator
    - The Current co-ordinator of the node
  - Definition
    - The State Information of the task being performed - the application algorithms, list of the participating nodes

The Bully Algorithm

- Each process has an associated priority (weight). The process with the highest priority should always be elected as the coordinator.
- How do we find the heaviest process?
- Any process can just start an election by sending an election message to all other processes
- If a process *P*heavy receives an election message from a lighter process *P*light, it sends a take-over message to *P*light. *P*light is out of the race.
- If a process doesn't get a take-over message back, it wins, and sends a victory message to all other processes.

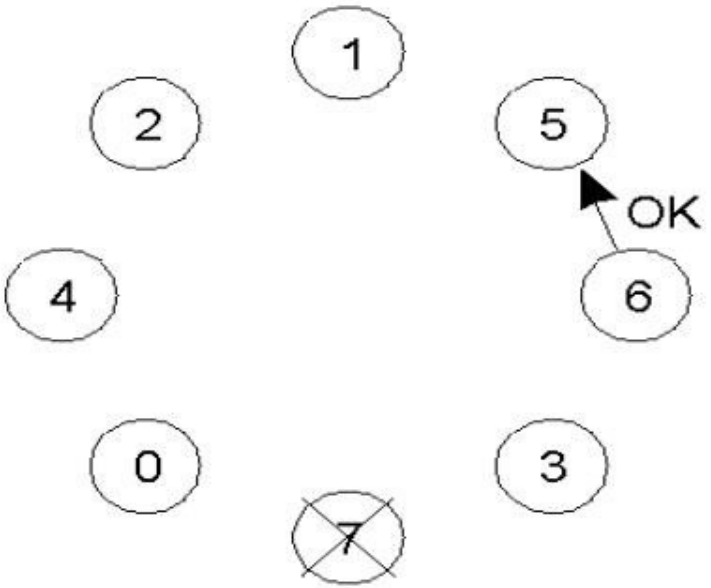
The Bully Algorithm



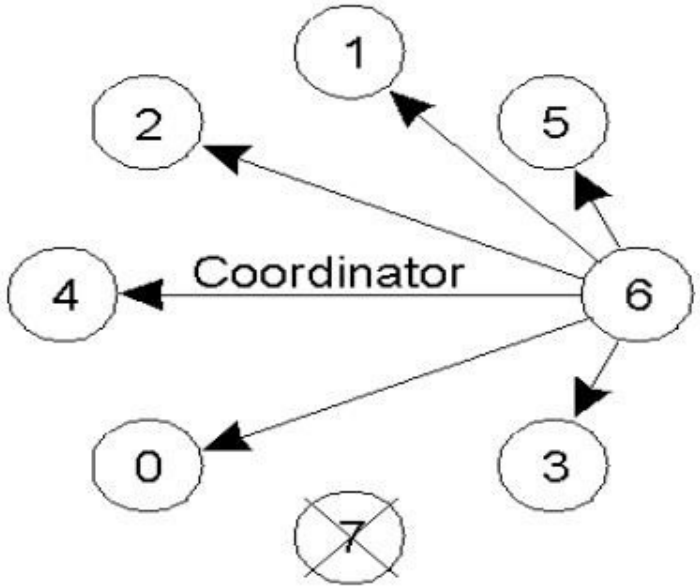
The bully election algorithm

Process 4 holds an election
Process 5 and 6 respond, telling 4 to stop
Now 5 and 6 each hold an election

- Process 6 tells 5 to stop
- Process 6 wins and tells everyone



(d)



(e)

Bully Algorithm: Properties

- Assume n processes initially
  - Worst Case:
    - Smallest process initiates election
    - Requires O(n2) messages
  - Best Case:
    - Eventual leader initiates election
    - Requires (n-1) messages

The Invitation Algorithm

- Bullying algorithm fails for asynchronous systems
- The invitation algorithm classifies nodes into ‘groups’ and elects a co-ordinator for every group
- An additional state variable is indroduced for this asynchronous election
  - Group Number
    - Identifier of the group the processor belongs to

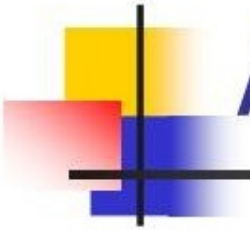
The Invitation Algorithm

- When Processor p wishes to establish itself as the new co-ordinator , it sends out invitations to the potential participants
- Groups that can communicate try to coalesce into larger groups
- Periodically the group co-ordinator searches for other groups

The Invitation Algorithm

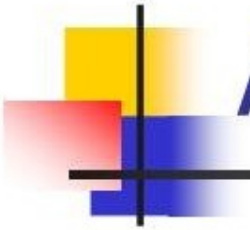
- Starting a new Group
  - If a node does not hear from its co-ordinator for a long time, the node declares its own group
  - It could start sending invitations to all the other participants of the old group

A Comparison of the two Algorithms

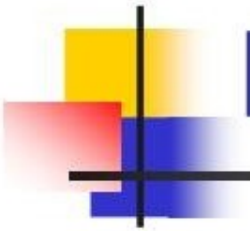


- Logical Structure
  - Bully –
    - Simple and Efficient because it imposes a logical structure on processors
  - Invitation –
    - Might take a long time for groups to merge because of no strongly defined structure

A Comparison of the two Algorithms



- Synchronous Vs Asynchronous
  - Bully
    - Makes very strong use of bounded response time which are mostly unrealistic
  - Invitation
    - Works correctly in the presence of timing failures

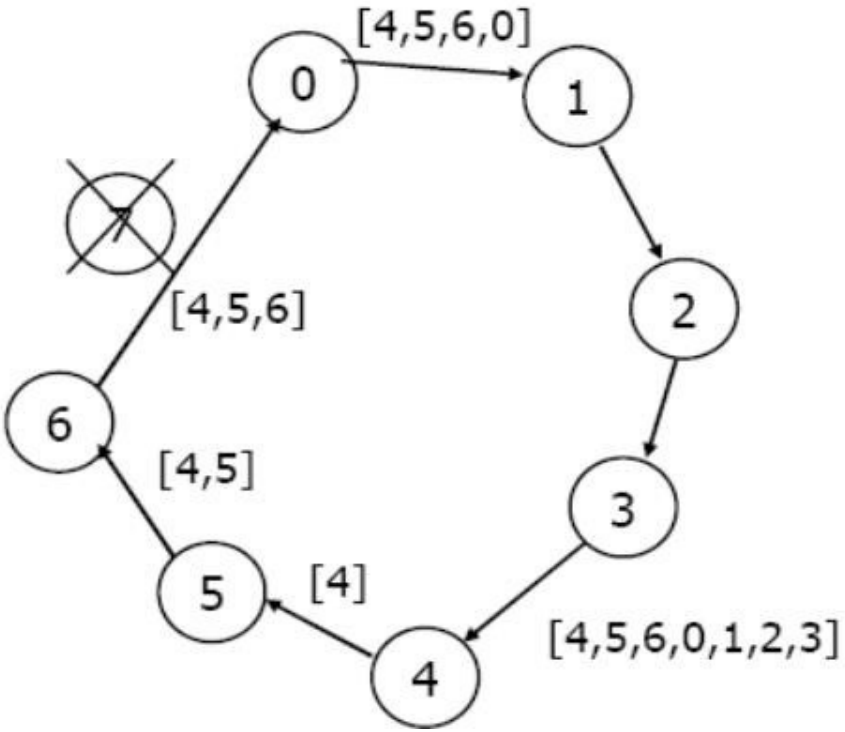


Election in a Ring

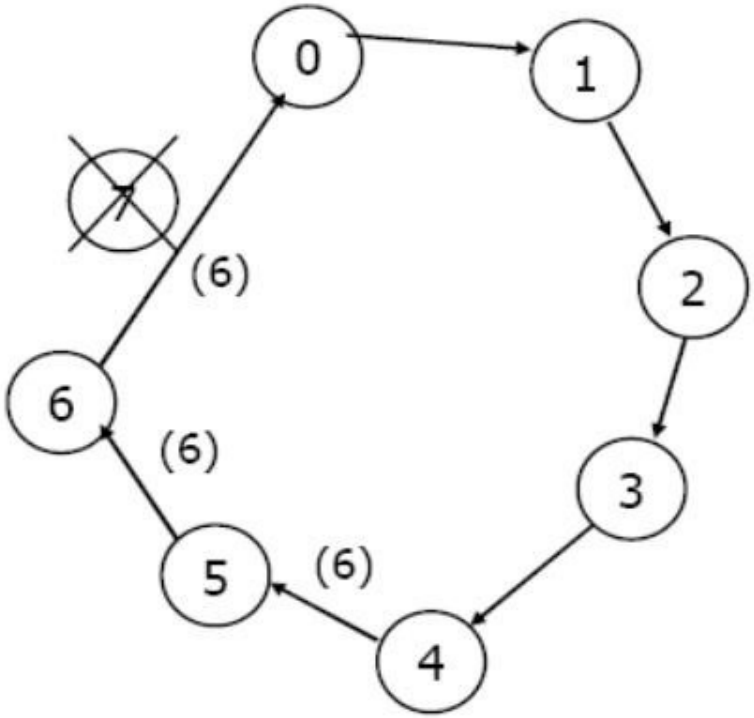
- Process priority is obtained by organizing processes into a (logical) ring. Process with the highest priority should be elected as coordinator.

- Each process has a successor
  - Initiation:
    - A process sends an ELECTION message to its successor (or next alive process) with its ID
    - Each process adds its own ID and forwards the ELECTION message
  - Leader Election:
    - Message comes back to initiator
    - Initiator announces the winner by sending another message around the ring

Ring Algorithm: Initiation



Ring Algorithm - Election



Ring Algorithm: Properties

- If only 1 process initiates election:
  - Requires 2n messages
- Two or more processes might simultaneously initiate elections
  - Still ensures election of the same leader
  - Results in extra messages

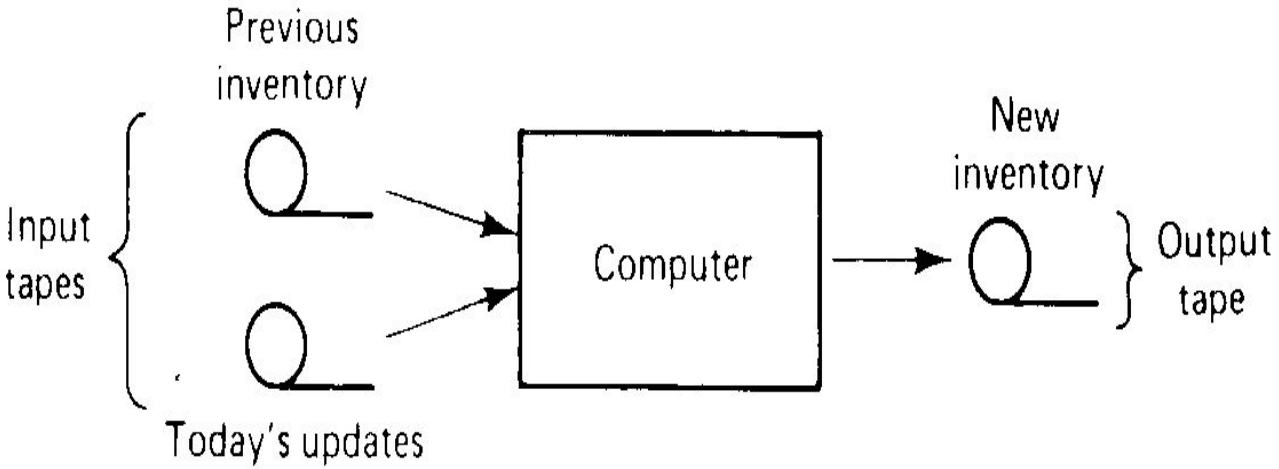
Thank You

ATOMIC TRANSACTION

ATOMIC TRANSACTION

- In Distributed system , a transaction is an operation that involves multiple nodes .
- To maintain consistency and reliability these system must ensure that a transaction is atomic means it is treated as single , indivisible operation which is either commits all changes or none to the system.
- Use,Design and Implementation

ATOMIC TRANSACTION USE:



ATOMIC TRANSACTION USE:

1. Withdraw(amount, account1).
2. Deposit(amount, account2).

THE TRANSACTION MODEL

- Thee types of storage

1. Ordinary RAM
2. Disk storage
3. **Stable storage**: It can survive any thing except major calamities such as floods and Earthquacks.

Stable storage

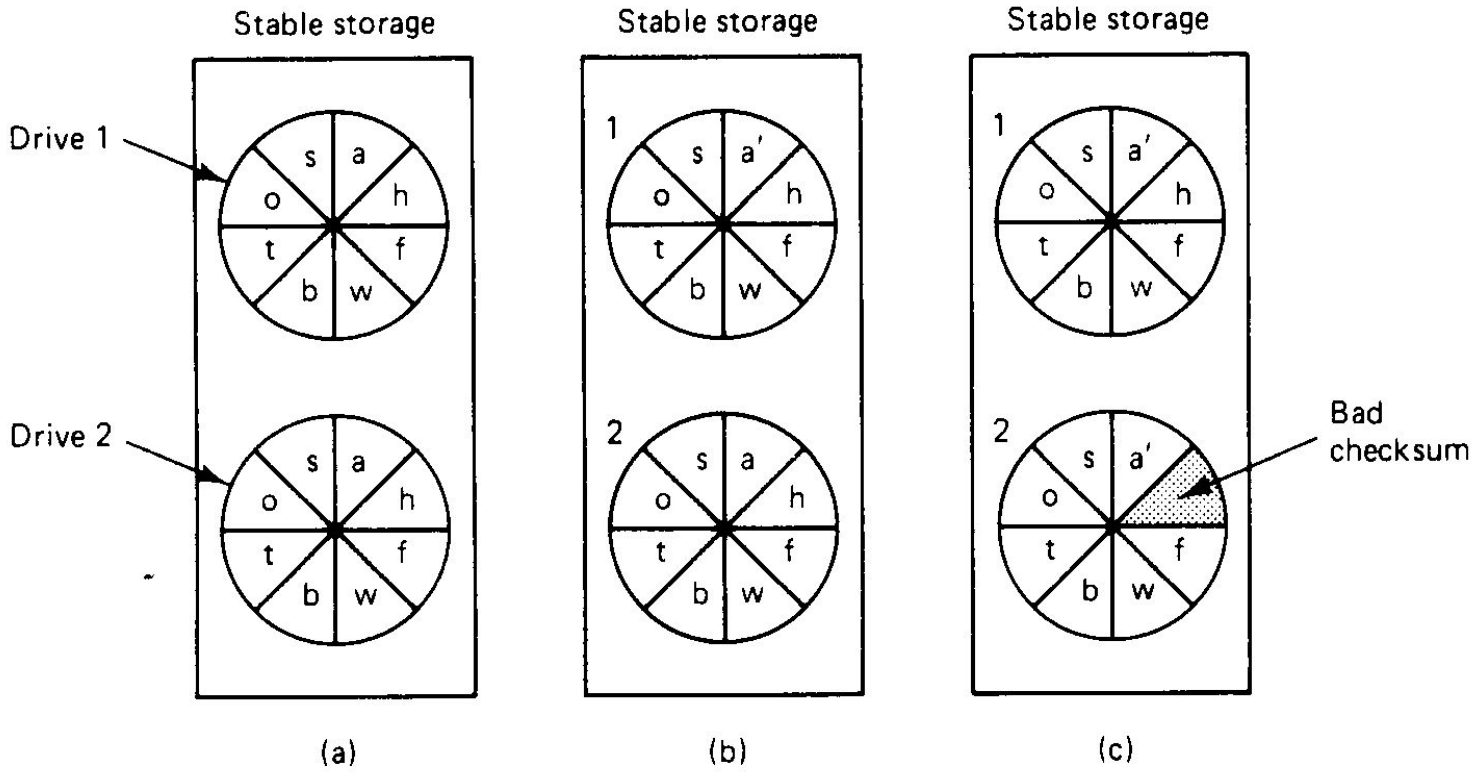


Fig. 3-15. (a) Stable storage. (b) Crash after drive 1 is updated. (c) Bad spot.

Transaction Primitives

Special primitives that must either be supplied by the operating system or language runtime system.

1. BEGIN_TRANSACTION: Mark the start of a transaction.
2. END_TRANSACTION: Terminate the transaction and try to commit.
3. ABORT_TRANSACTION: Kill the transaction; restore the old values.
4. READ: Read data from a file (or other object).
5. WRITE: Write data to a file (or other object).

Properties of Transaction

Transactions have four essential properties. Transactions are:

1. Atomic: To the outside world, the transaction happens indivisibly.
2. Consistent: The transaction does not violate system invariants.
3. Isolated: Concurrent transactions do not interfere with each other.
4. Durable: Once a transaction commits, the changes are permanent.

These properties are often referred to by their initial letters, **ACID**.

Nested Transaction

- Transaction may contain subtransaction,which is called nested transaction .
- Process->child process(fork)->subchild(fork)

Atomic Transaction Implementation

A RANJITH

ASISSTANT PROFESSOR

SCHOOL OF COMPUTER SCIENCE

Atomic Transaction Implementation

- Private Workspace,
- Writeahead Log,
- Two-Phase Commit Protocol.

Private Workspace.

When a process starts a transaction, it is given a private workspace containing all the files (and other objects) to which it has access. Until the transaction either commits or aborts, all of its reads and writes go to the private workspace, rather than the "real" one, by which we mean the normal file system.

Advantage of this method is actually giving a process a private workspace at the instant it begins a trans-action.
Disadvantage is that the cost of copying everything to a private workspace is prohibitive, but various optimizations make it feasible. The first optimization is based on the realization that when a process reads a file but does not modify it, there is no need for a private copy. It can just use the real one (unless it has been changed since the transaction started).

When a process starts a transaction, it is sufficient to create a private workspace for it that is empty except for a pointer back to its parent's workspace.

When a file is opened for writing, it can be located in the same way as for reading, except that now it is first copied to the private workspace but here also we can optimize even for write operation entire file need not to be copied into private workspace,just copy entire indix and modifieble block has both at original file and private workspace. when transaction commit, changes made in private work space blocks will effected/saved in original file of respected blocks.

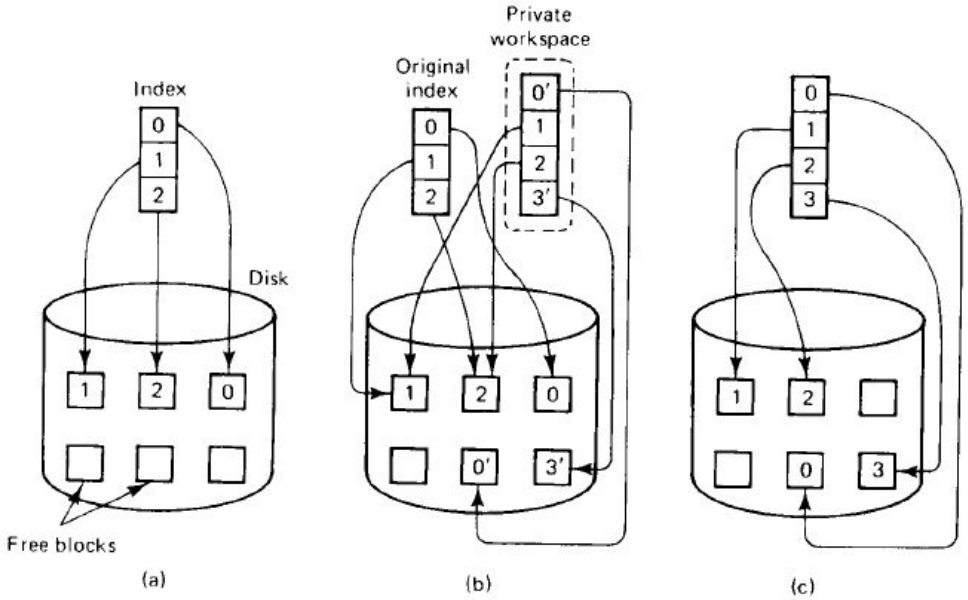


Fig. 3-18. (a) The file index and disk blocks for a three-block file. (b) The situation after a transaction has modified block 0 and appended block 3. (c) After committing.

As can be seen from Fig. 3-18(b), the process running the transaction sees the modified file, but all other processes continue to see the original file. In a

In UNIX, the index is the i-node. Using the private index, the file can be read in the usual way, since the disk addresses it contains are for the original disk blocks. However, when a file block is first modified, a copy of the block is made and the address of the copy inserted into the index, as shown in Fig. 3-18. The block can then be updated without affecting the original. Appended blocks are handled this way too. The new blocks are sometimes called shadow blocks.

In Fig. 3-18.
(a) indexes of file before modifying.
(b) Situation transacion has modified block 0 and new block (block 3) appended to file. here observe private workspace indexes and extra copis.
(c) after comminting transactions

If the transaction aborts, the private workspace is simply deleted and all the private blocks that it points to are put back on the free list. If the transaction commits, the private indices are moved into the parent's workspace atomically, as shown in Fig. 3-18(c). The blocks that are no longer reachable are put onto the free list.

Writeahead Log(intentions list)

With this method,

Files are actually modified in place, but before any block is changed, a record is written to the writeahead log on stable storage telling which transaction is making the change, which file and block is being changed, and what the old and new values are.

Only after the log has been written successfully is the change made to the file.

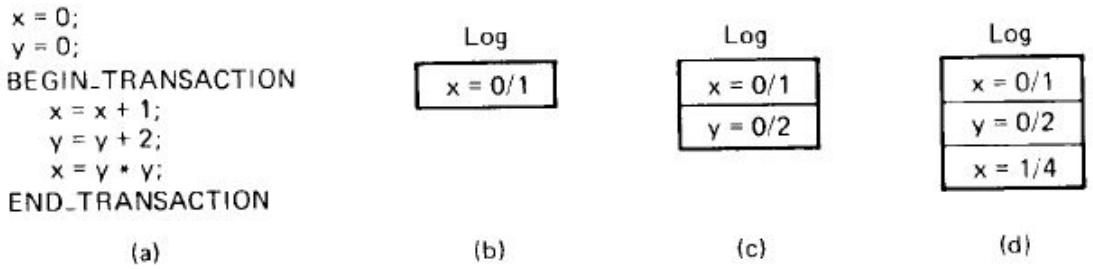


Fig. 3-19. (a) A transaction. (b)–(d) The log before each statement is executed.

Figure 3-19 gives an example of how the log works. In Fig. 3-19(a) we have a simple transaction that uses two shared variables (or other objects), x and y, both initialized to 0.

Fig.3-19 (b),(c) and (d) For each of the three statements inside the transaction, a log record is written before executing the statement, giving the old and new values, separated by a slash.

Case 1: If the transaction succeeds and is committed:

If the transaction succeeds and is committed, a commit record is written to the log, but **the data structures do not have to be changed, as they have already been updated.**

Case 2: If the transaction aborts

If the transaction aborts, the log can be used to back up to the original state. Starting at the end and going backward, each log record is read and the change described in it undone. This action is called a rollback.

Case 3: The log can also be used for recovering from crashes.

Suppose that the process doing the transaction crashes just after having written the last log record of Fig. 3-19(d), but before changing x. After the failed machine is rebooted, the log is checked to see if any transactions were in progress at the time of the crash.

When the last record is read and the current value of x is seen to be 1, it is clear that the crash occurred before the update was made, so x is set to 4.

If, on the other hand, x is 4 at the time of recovery, it is equally clear that the crash occurred after the update, so nothing need be changed.

Using the log, it is possible to go forward(do the transaction) or go backward(undo the transaction)

Two-Phase commit Protocol

As we have pointed out repeatedly, the action of committing a transaction must be done atomically, that is, instantaneously and indivisibly.

In a distributed system, the commit may require the cooperation of multiple processes on different machines, each of which holds some of the variables, files, and data bases, and other objects changed by the transaction.

Two-phase commit protocol one of widely used protocol for achieving atomic commit in a distributed system.

The basic idea is illustrated in Fig. 3-20. One of the processes involved functions as the coordinator. Usually, this is the one executing the transaction.

The commit protocol begins when the coordinator writes a log entry saying that it is starting the commit protocol, followed by sending each of the other processes involved (the subordinates) a message telling them to prepare to commit.

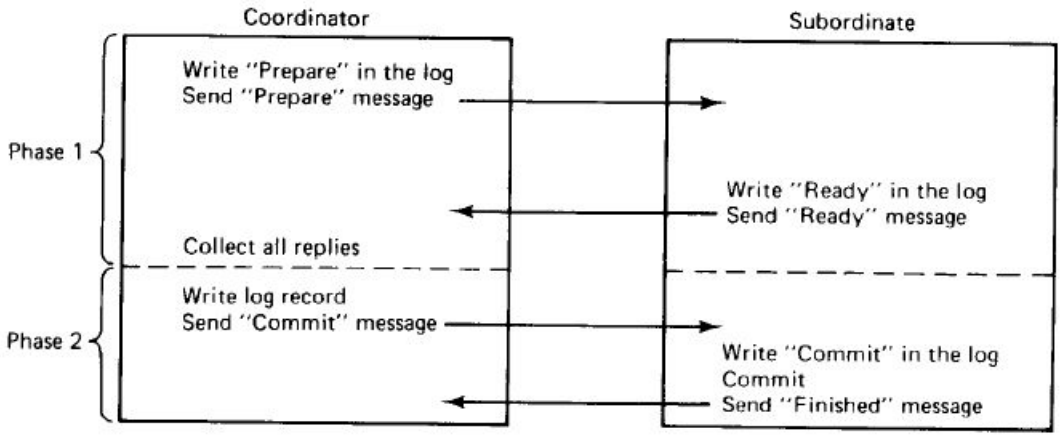


Fig. 3-20. The two-phase commit protocol when it succeeds.

When a subordinate gets the message it checks to see if it is ready to commit, makes a log entry, and sends back its decision.
When the coordinator has received all the responses, it knows whether to commit or abort. If all the processes are prepared to commit, the transaction is committed.

If one or more are unable to commit (or do not respond), the transaction is aborted. Either way, the coordinator writes a log entry and then sends a message to each subordinate informing it of the decision.

It is this write to the log that actually commits the transaction and makes it go forward no matter what happens afterward.

Due to the use of the log on stable storage, this protocol is highly resilient in the face of (multiple) crashes.

Q&A

Thank You!