Time and Global States

1. Define the following terms:
a) Physical clock b) Clock skew and clock drift
c) Coordinated Universal Time

Physical Clock

Every computer has its own physical clock. These clocks are electronic devices that count oscillations occurring in a crystal that vibrates at a specific frequency when electricity is applied, and typically divide this count and store the result in a counter register. Clock devices can be programmed to generate interrupts at regular intervals in a particular order. The speed of a computer processor is measured in clock speed.

Clock Skew and Clock Drift

Computer clocks, like any other clocks, tend not to be in perfect agreement. The instantaneous difference between the readings of any two clocks is called their *skew.* Also, the crystal-based clocks used in computers are, like any other clocks, subject to *clock drift*, which means that they count time at different rates, and so diverge. A clock’s drift rate is the change in the offset between the clock and a nominal perfect reference clock per unit of time measured by the reference clock.

- Clock Skew = Relative Difference in clock values of two processes.
- Clock Drift = Relative Difference in clock frequencies (rates) of two processes.

Coordinated Universal Time

Computer clocks can be synchronized to external sources of highly accurate time. The most accurate physical clocks use atomic clocks and are used as the standard for elapsed real-time, known as International Atomic Time. Coordinated Universal Time also known as UTC is an international standard for timekeeping. It is based on atomic time. UTC signals are synchronized and broadcast regularly from land-based radio stations and satellites covering many parts of the world.

2. Explain different modes of synchronizing a physical clock.

In order to know at what time of day events occur at the processes in our distributed system – for example, for accountancy purposes – it is necessary to synchronize the processes' clocks, Ci , with an authoritative, external source of time. This is external synchronization. And if the clocks Ci are synchronized with one another to a known degree of accuracy, then we can measure the interval between two events occurring at different computers by appealing to their local clocks, even though they are not necessarily synchronized to an external source of time. This is internal they are not necessarily synchronized to an external source of time. This is internal of real time I (captital i):

External Synchronization:

For a synchronization bound D 0 , and for a source S of UTC time, St – Cit < D, for i = 1 2N and for all real times t in I. Another way of saying this is that the clocks Ci are accurate to within the bound D.

Internal synchronization:

For a synchronization bound D 0 , Cit – Cjt D for i j = 1 2N , and for all real times t in I. Another way of saying this is that the clocks Ci agree within the bound D.

3. Explain internal synchronization between two processes in a synchronous distributed system.

In an synchronous system, bounds are known for the drift rate of clocks, the maximum message transmission delay, and the time required to execute each step of a process. One process sends the time t on its local clock to the other in a message m. In principle, the receiving process could set its clock to the timer $t + T_{trans}$, where T_{trans} is the time taken to transmit m between them. The two clocks would then agree. Unfortunately, T_{trans} is subject to variation and is unknown. In general, other processes are competing for resources with the processes to be synchronised at their respective nodes, and other messages compete with m for the network resources. Nonetheless, there is always a minimum transmission time, min , that would be obtained if no other process executed and no other network traffic existed.

4. Explain Cristian’s method for synchronizing clocks.

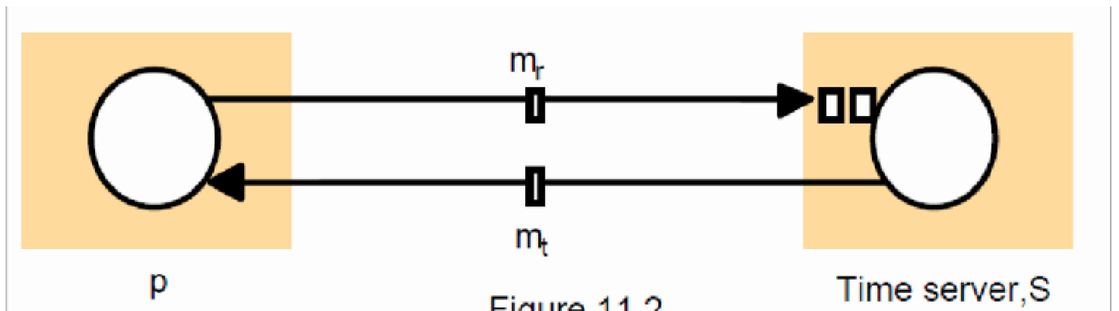


Figure 11.2

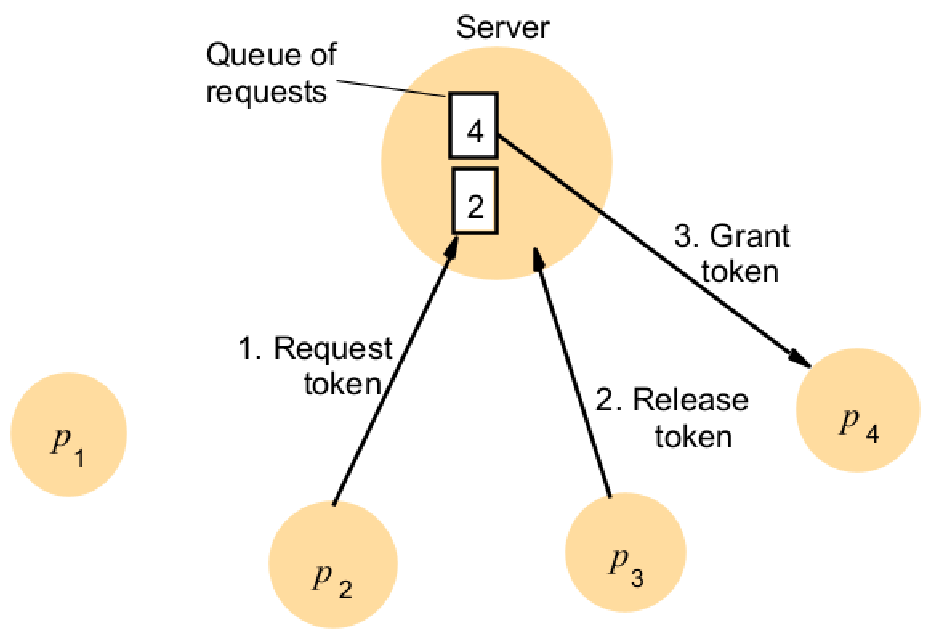
Cristian suggested the use of a time server, connected to a device that receives signals from a source of UTC, to synchronise computers externally. Upon request, the server process S supplies the time according to its clock as shown in the figure. Cristian Observed that while there is no upper bound on message transmission delays in an asynchronous system, the round-trip times for messages exchanged between pairs of processes are often reasonably short - a small fraction of a second. He describes the algorithm as *probabilistic*: the method achieves synchronisation only if the observed round-trip times between client and server are sufficiently short compared with the required accuracy. A process *p* request the time in a message m_r, and receives the time value *t* in a message m_t. Process p records the total round trip time T_{round} taken to send the request m_r and receive the reply m_t. It can measure this time with reasonable accuracy if it's rate of clock drift is small.

5. Explain the Berkeley algorithm for internal synchronization.

Gusella and Zatti describe an algorithm for internal synchronisation that they developed for collections of computers running Berkeley Unix. In it, a coordinator computer is chosen to act as the *master*. Unlike in Cristian’s protocol, this computer periodically polls the other computers whose clocks are to be synchronised, called *slaves*. The slaves send back their clocks values to it. The master estimates their local clock times by observing the round-trip times, and it averages the values obtained. The balance of probabilities is that this average cancels out the individuals clocks’ tendencies to run fast or slow. The accuracy of the protocol depends upon a nominal maximum round-trip time between the master and the slaves. The master eliminates any occasional readings associated with larger times than this maximum. Instead of sending the updated current time back to the other computers, the master sends the amount by which each individual slave's clock requires adjustment. This can be positive or negative value.

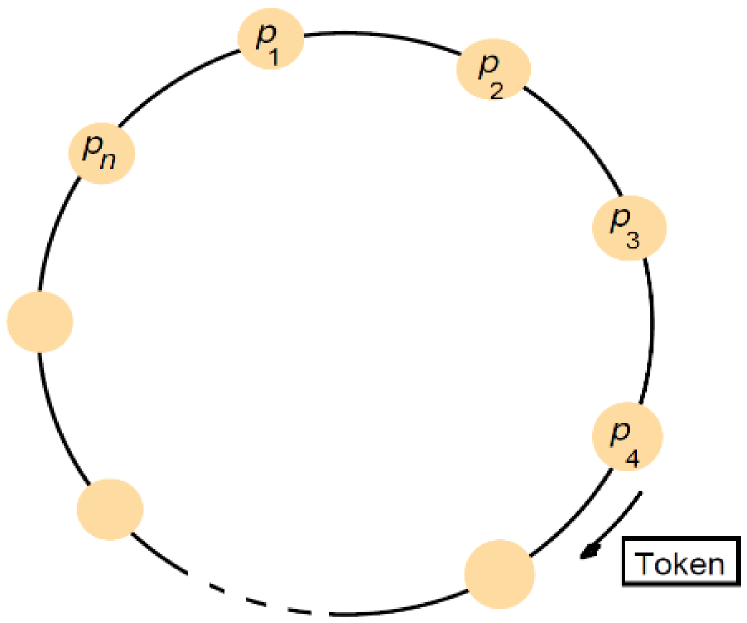
Co-ordination and Agreement

1. Explain with a neat diagram Central Server Algorithm.



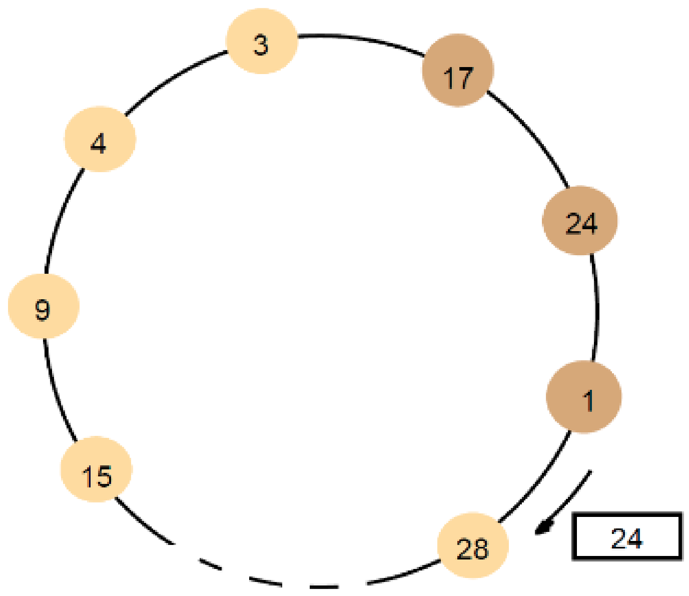
The simplest way to achieve mutual exclusion is to employ a server that grants permission to enter the critical section. The figure shows the use of this server. To enter a critical section, a process sends a request message to the server and awaits a reply from it. Conceptually, the reply constitutes a token specifying permission to enter the critical section. If no other process has the token at the time of the request, then the server replies immediately, granting the token. If the token is currently held by another process, then the server does not reply, but queues the request. When a process exits the critical section, it sends a message to the server, giving it back the token. If the queue of waiting processes is not empty, then the server chooses the oldest entry in the queue, removes it and replies to the corresponding process.

2. With a neat diagram explain Ring based Algorithm w.r.t. mutual exclusion.



One of the simplest ways to arrange mutual exclusion between the N processes without requiring an additional process is to arrange them in a logical ring. This requires only that each process p_i has a communication channel to the next process in the ring, $p_{(i+1) mod N}$. The idea is that exclusion is conferred by obtaining a token in the form of a message passed from process to process in a single direction, ex: clockwise or around the ring. The ring topology may be unrelated to the physical interconnections between the underlying computers. If a process does not required to enter the critical section when it receives the token, then it immediately forwards the token to its neighbour. A process that requires the token waits until it receives it, but retain it. To exit the critical section, the process sends the token on to its neighbour.

3. With a neat diagram explain Ring based Election Algorithm.



Note: The election was started by process 17.
The highest process identifier encountered so far is 24.
Participant processes are shown in a darker colour

The algorithm of Chang and Roberts is suitable for a collection of processes arranged in a logical ring. Each process p_i has a communication channel to the next process in the ring, $p_{(i+1) mod N}$, and all the the messages are sent clockwise around the ring. We assume that no failures occur, and that the system is asynchronous. The goal of this algorithm is to elect a single process called the *coordinator*, which is the process with the largest identifier.

Initially, every process is marked as non-participant in an election. Any process can begin an election. It proceeds by marking itself as a participant, placing its identifier in an election message and sending it to its clockwise neighbour.

When a process receives an election message, it compares the identifier in the message with its own. If the arrived identifier is greater, then it forwards the message to its neighbour. If the arrived identifier is smaller and the receiver is not a participant, then its substitute its own identifier in the message and forwards it. If, however, the received identifier is that of the receiver itself, then this process's identifier must be the greatest, and it becomes the coordinator. The coordinator marks itself as a non-participant once more and sends an elected message to its neighbour, announcing its election and enclosing its identity.