1. **What is global state of a system?**

The global state of a distributed system is the set of local states of each individual processes involved in the system plus the state of the communication channels. Notationally global state GS is defined as,

$$GS = \{ \cup_i LS_i , \cup_{i,j} SC_{ij} \}$$

2. **Define Access transparency?**

Access Transparency allows the same operations to be used to access local and remote resources. The file distribution must be hidden from the clients. The storing of data on separate servers that are physically separated, and a common set of actions should be available to access both remote and local files. Applications for local files are to be designed such that they should be able to run on remote files as well.

3. **State atleast two different motivation behind process migration?**

Two different motivation behind process migration are as follows,

- **Dynamic Load Balancing:** It permits processes to exploit less stacked nodes by relocating from overloaded ones.
- **Recovery of faults:** The component to stop, transport and resume a process is actually valuable to support in recovering the fault in applications that are based on transactions.
- **Accessibility:** Processes that inhibit defective nodes can be moved to other perfect nodes.

4. **Why token based algorithm said to be inherently safe?**

Token based algorithms said to be inherently safe because tokens don't have to contain a user's personal data and are algorithm/software generated, they keep this data safer from hackers. This is a huge improvement over enterprises using a person's social security number or other personal/private information as their account number, making it easier for bad actors or hacker to steal identities. In other words when we use a token that comes from an application such as PingID, a key fob, or a dongle that we plug into our computer, we keep outside actors from interfering because the token is based on the private key of the device.

5. **What is Distributed shared memory ?**

DSM is a mechanism that manages memory across multiple nodes and makes inter-process communications transparent to end-users. A distributed shared memory (DSM) system is a collection of many nodes/computers which are connected through some network and all have their local memories. The DSM system manages the memory across all the nodes. The

DSM does not have any physical memory, instead, a virtual space address is shared among all the nodes, and the transfer of data occurs among them through the main memory. DSM is a mechanism of allowing user processes to access shared data without using inter-process communications.

6. **What are the commonly used methods to solve thrashing problem in DSM system?**

Distributed shared memory is used to allow applications on separate computers to work together and share memory. Distributed shared memory might experience '**Thrashing'**.

- Data is locked for a short period of time to prevent nodes from accessing data and thus will prevent thrashing. For this method, an application-controlled lock can be associated with each data block.
- Different coherence protocols for shared data having different characteristics can be used to minimize thrashing.
- A block is disallowed to be taken away from a node until a minimum amount of time(t) passes after it has been allocated to the node.
- There are ***two way***s to tune the value of t. They are as follows:
  Based on the past access pattern of the block- the value of t can be tuned. And,
  Based on the length of the queue of processes waiting to access that block.

7. **What is preemptive and non-preemptive process migration?**

Non-preemptive process: If a process is moved before it begins execution on its source node which is known as a non-preemptive process migration.

Preemptive process: If a process is moved at the time of its execution that is known as preemptive process migration. Preemptive process migration is all the more expensive in comparison to the non-preemptive on the grounds that the process environment should go with the process to its new node.

8. **State the condition for happen before relation between events?**

The condition for happen before relation is as follows,

- *A* -> *B* if *A* and *B* are within the same process (same sequential thread of control) and *A* occurred before *B*.
- *A* -> *B* if *A* is the event of sending a message *M* in one process and *B* is the event of receiving *M* by another process
- if *A* -> *B* and *B* -> *C* then *A* -> *C*

-> denotes happen before

9. **What is stub?**

A stub is a piece of code that translates parameters sent between the client and server during a remote procedure call in distributed computing. An RPC's main purpose is to allow a local computer (client) to call procedures on another computer remotely (server) because the client and server utilize distinct address spaces. That is the reason parameters used in a function (procedure) call must be translated. Client stubs transform (marshaling) parameters used in function calls and then reconvert the result returned by the server when the function is completed. On the other side, client arguments are reconverted by server stubs, and results are converted back after function execution.

10. How stud are generated?

Stubs can be created in two different ways. Stubs are generated either manually or automatically.

- **Manual Generation of Stub:** In the manual generation of stubs, the RPC implementer provides a collection of translation functions from which a user can create their own stubs using this way. This method is easy to use and can handle a wide range of argument types.
- **Automatic Generation of Stub:** In the automatic generation of stubs, client and server interfaces are defined using Interface Definition Language (IDL). An interface specification, for example, contains information indicating whether each argument is input, output, or both; only input arguments must be passed from client to server, while only output elements must be copied from server to client. This is the most popular way of generating stubs.

11. What is false Sharing?

False sharing occurs when processors in a shared-memory parallel system make references to different data objects within the same coherence block (cache line or page), thereby inducing "unnecessary" coherence operations. False sharing is a situation that can occur in a computer program when two applications that are running simultaneously attempt to access information in the same logical memory region that each program or process has stored in its own cache. The data in each application's cache are copied from a common source, so modifying one cache causes the other to have to be reloaded from the source. The false aspect of the sharing arises when the changes made to the cache line by one program do not actually affect the data that the second program is using, in which case forcing the cache to be reloaded is a waste of system resources and can negatively affect the performance of the program.