GOVERNMENT ENGINEERING COLLEGE JHALAWAR

Model Question Paper- Distributed Systems (8CS3A)

B.Tech CS VIII Semester

Faculty In charge: Rukhsar Sultana **Date: 31/01/2018**

Unit I

Q. 1 What is Distributed Systems? Explain features and challenges in distributed systems.

Q. 2 Explain architecture models of distributed system.

Q. 3 What is centralized operating system? Explain functions and concepts of centralized operating system.

Q. 4 What are the different design issues in distributed operating system?

Q. 5 Explain DCE. Explain architecture and services of DCE.

Q. 6 What is mutual consistency of states? Explain different algorithm to record state of distributed system.

Solution 1:

Distributed Systems: A distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages. Computers that are connected by a network may be spatially separated by any distance. They may be on separate continents, in the same building or in the same room. We can say that

A distributed system is a collection of independent computers that appear to the users of the system as a single computer.

Example DS:

- Web (and many of its applications like Online bookshop)
- Data Centers and Clouds
- Wide area storage systems
- Banking Systems
- User-level communication (Facebook, Skype)

DSs have the following consequences:

1. Concurrency – Each system is autonomous.
   - Carry out tasks independently
   - Tasks coordinate their actions by exchanging messages.
2. Heterogeneity
3. No global clock
4. Independent Failures

Features: A distributed system has following features:

1. Parallel activities: Participants can execute their own tasks in parallel, with little or no synchronisation

2. Communication via message passing: No shared memory

3. Resource sharing: Printer, database, other services

4. No global state: No single process can have knowledge of the current global state of the system

5. No global clock: Only limited precision for processes to synchronize their clocks

Challenges:

1. Heterogeneity: Heterogeneous components must be able to interoperate

2. Distribution transparency: Distribution should be hidden from the user as much as possible

3. Fault tolerance: Failure of a component (partial failure) should not result in failure of the whole system

4. Scalability: System should work efficiently with an increasing number of user and system performance should increase with inclusion of additional resources

5. Concurrency: Shared access to resources must be possible

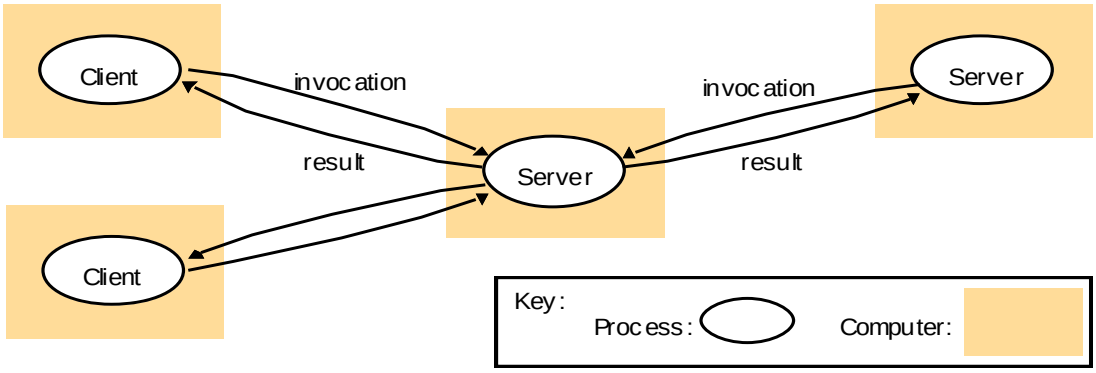
6. Openness: Interfaces should be publicly available to ease inclusion of new components

7. Security: The system should only be used in the way intended

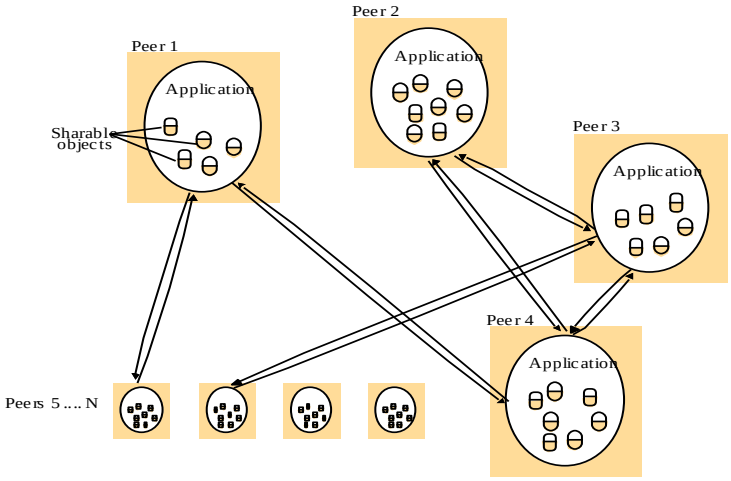
Solutions 2:

Architecture Models: An Architectural model of a distributed system is concerned with the placement of its parts and relationship between them. There are following architecture models:

1. **Client-server:** This is the architecture that is most often cited when distributed systems are discussed. It is historically the most important and remains the most widely employed. Figure below illustrates the simple structure in which processes take on the roles of being clients or servers. In particular, client processes interact with individual server processes in potentially separate host computers in order to access the shared resources that they manage.

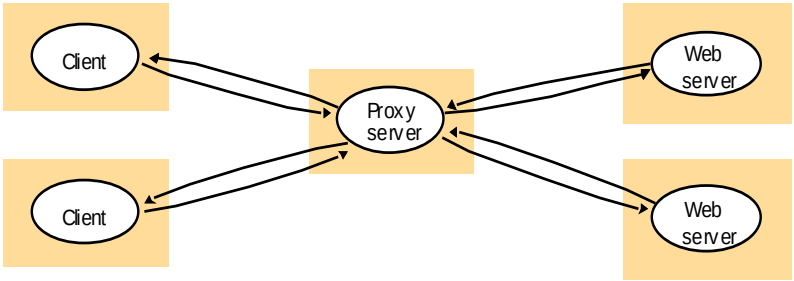


2. **Peer-to-peer**: In this architecture all of the processes involved in a task or activity play similar roles, interacting cooperatively as peers without any distinction between client and server processes or the computers on which they run. In practical terms, all participating processes run the same program and offer the same set of interfaces to each other. While the client-server model offers a direct and relatively simple approach to the sharing of data and other resources, it scales poorly.



3. Proxy servers: Proxy servers (replication transparency) and caches: Web proxy server

A cache is a store of recently used data.



4. Mobile code: Applets are a well-known and widely used example of mobile code – the user running a browser selects a link to an applet whose code is stored on a web server; the code is downloaded to the browser and runs there.

5. Mobile agents: A mobile agent is a running program (including both code and data) that travels from one computer to another in a network carrying out a task on someone's behalf, such as collecting information, and eventually returning with the results. A mobile agent may make many invocations to local resources at each site it visits.

Solution 3:

Centralized Operating systems: Run on a single computer system and do not interact with other computer systems.

General-purpose computer system: one to a few CPUs and a number of device controllers that are connected through a common bus that provides access to shared memory.

Single-user system (e.g., personal computer or workstation): desk-top unit, single user, usually has only one CPU and one or two hard disks; the OS may support only one user.

Multi-user system: more disks, more memory, multiple CPUs, and a multi-user OS. Serve a large number of users who are connected to the system vie terminals. Often called server systems.

Goals of Centralized Operating systems: The fundamental goals of an operating system are:

- Efficient use: Ensure efficient use of a computer's resources.
- User convenience: Provide convenient methods of using a computer system.
- Noninterference: Prevent interference in the activities of its users.

Operations in OS: The primary concerns of an OS during its operation are execution of programs, use of resources, and prevention of interference with programs and resources. Accordingly, its three principal functions are:

• Program management: The OS initiates programs, arranges their execution on the CPU, and terminates them when they complete their execution. Since many programs exist in the system at any time, the OS performs a function called scheduling to select a program for execution.

• Resource management: The OS allocates resources like memory and I/O devices when a program needs them. When the program terminates, it deallocates these resources and allocates them to other programs that need them.

• Security and protection: The OS implements noninterference in users' activities through joint actions of the security and protection functions. As an example, consider how the OS prevents illegal accesses to a file. The security function prevents nonusers from utilizing the services and resources in the computer system, hence none of them can access the file. The protection function prevents users other than the file owner or users authorized by him, from accessing the file.

Solution 4:

Design issues in distributed operating system:

The user of a distributed system expects its operating system to provide the look and feel of a conventional OS. To meet these expectations, the OS must fully exploit the capabilities of all nodes by distributing data, resources, users, and their computations effectively among the nodes of the system. It gives rise to the following design issues.

1. Transparency of Resources and Services: Transparency implies that names of resources and services do not depend on their locations in the system. It enables an application to access local and nonlocal resources identically. It also permits an OS to change the location of a resource freely because a change in location does not affect the name of the resource and hence does not affect the applications that use the resource. The OS can exploit transparency to perform data migration to speed up applications, reduce network traffic, or optimize use of disks. Transparency also facilitates computation migration because the computation can continue to access resources as it did before it was migrated.

2. Distribution of Control Functions: A control function is a function performed by the kernel to control resources and processes in the system, e.g., resource allocation, deadlock handling, and scheduling. Centralized control functions face two problems in a distributed system: Because of

network latency, it is not possible to obtain consistent information about the current state of processes and resources in all nodes of the system, so the centralized function may not be able to arrive at correct decisions. A centralized function is also a potential performance bottleneck and a single point of failure in the system. To handle these problems, a distributed OS performs a control function through a distributed control algorithm, who sections are performed in several nodes of the system in a coordinated manner.

3. System Performance: In addition to techniques of conventional OSs, a distributed OS uses two new techniques to provide good system performance—data migration and computation migration. Data migration is employed to reduce network latencies and improve response times of processes. Computation migration is employed to ensure that nearly equal amounts of computational load are directed at all CPUs in the system. This technique is called load balancing. A distributed system typically grows in size over time through addition of nodes and users.

4. Reliability: Fault tolerance techniques provide availability of resources and continuity of system operation when faults occur. Link and node faults are tolerated by providing redundancy of resources and communication links. If a fault occurs in a network path to a resource or in the resource itself, an application can use another network path to the resource or use another resource. This way, a resource is unavailable only when unforeseen faults occur.

5. Consistency: Consistency of data becomes an issue when data is distributed or replicated. When several parts of distributed data are to be modified, a fault should not put the system in a state in which some parts of the data have been updated but others have not been. A distributed OS employs a technique called two-phase commit protocol to ensure that it does not happen Parts of a computation may be performed in different nodes of a system. If a node or link fault occurs during execution of such a computation, the system should assess the damage caused by the fault and judiciously restore some of the sub computations to previous states recorded in backups. This approach is called recovery. The system must also deal with uncertainties about the cause of a fault.

Solution 5:

Distributed Computing Environment: OSF's DCE builds a distributed system on top of existing operating systems. Thus gives users a way of introducing distributed services without discarding their current operating systems. DCE is a suite of distributed services rather than a distributed operating system. DCE was developed by the Open Software Foundation (OSF), a consortium of computer manufacturers (HP, DEC, IBM, others) organized to develop standard (corss-platform) computing solutions.

The DCE framework includes

- Remote Procedure Call (RPC) mechanism known as DCE/RPC.
- Naming (directory) Service.
- Time Service.
- Authentication Service.
- Authorization Service.
- Distributed File System (DFS) known as DCE/DFS.

Architecture: The largest unit of management in DCE is a cell. The highest privileges within a cell are assigned to a role called cell administrator , who is a real OS – Level user.

Privileges can be awarded to or removed from the following categories :

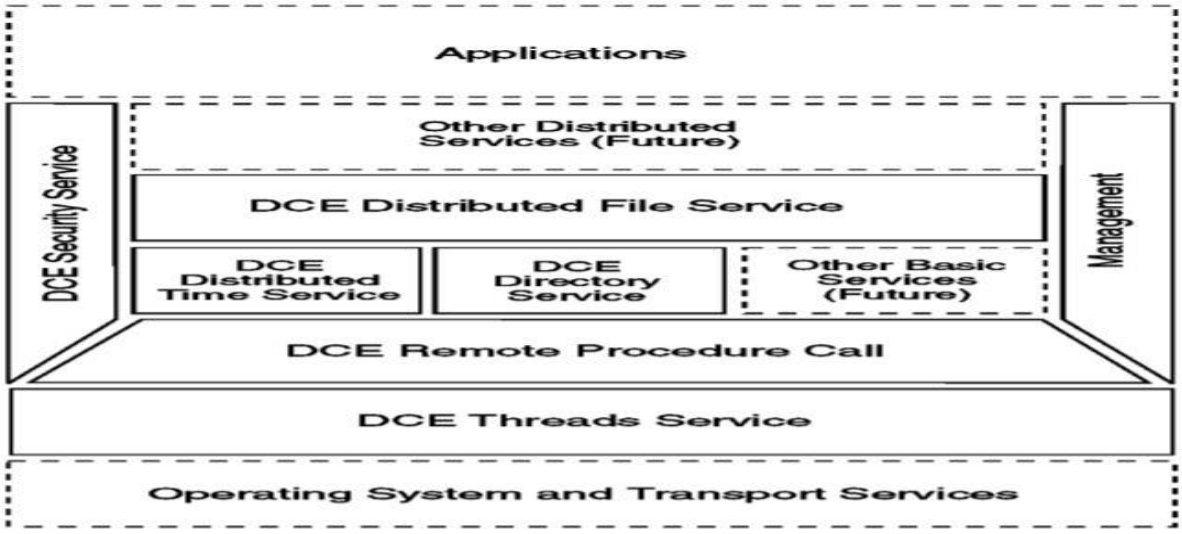
- user_obj : Owner
- group_obj : Group member
- other_obj : Any other DCE/non-DCE principal.

Major Components of DCE Cell

1) Security Server : that is responsible for authentication.

2) C.D.S : that is the repository of resources and ACLs.

3) D.T.S : that provides an accurate clock for proper functioning of the entire cell.



Architecture Of DCE

Services of DCE

1. Remote Procedure Call (RPC): A procedure call is a method of implementing the Client/Server Communication. The procedure call is translated into network communications by the underlying RPC mechanism.

2. Directory Service: The DCE Directory Service advertises that the server supports the new interface defined using the IDL. DCE Security Service also ensures that only authorized client end users can access the newly defined server function.

3. Security Service: There are three aspects to DCE security:

   - Authentication: This identifies that a DCE user or service is allowed to use the service.
   - Secure communications: Communication over the network can be checked for tampering or encrypted for privacy.
   - Authorization: This issues the permission to access the service. These are implemented by several services and facilities which include the Registry Service, Privilege Service, Access Control List (ACL) Facility, and Login Facility.

4. Time Service: The DCE Time Service (DTS) provides synchronized time on the computers participating in a Distributed Computing Environment. DTS synchronizes a DCE host's time with Coordinated Universal Time (UTC), an international time standard.

5. File Service: The DCE File Service (DFS) allows users to access and share files stored on a File Server anywhere on the network, without having to know the physical location of the file.

6. Threads: DCE Threads supports the creation, management, and synchronization of multiple threads of control within a single process. This component is conceptually a part of the operating system layer, the layer below DCE.

Solution 6:

Mutual consistency of states

Consistent State Recording: A state recording is a collection of local states of entities in a system obtained through some algorithm. A consistent state recording is one in which process states of every pair of processes in the system are consistent according to Definition 6.1.

Definition 6.1 Mutually Consistent Local States Local states of processes Pk and Pl are mutually consistent if 1. Every message recorded as "received from Pl" in Pk's state is recorded as "sent to Pk" in Pl's state, and 2. Every message recorded as "received from Pk" in Pl's state is recorded as "sent to Pl" in Pk's state.

Recording state of distributed system:

State of a Channel: The state of a channel Chij is the set of messages contained in Chij, i.e., the messages sent by process Pi that are not yet received by process Pj. We use the following notation to determine the state of a channel Chij:

Recorded_sentij: The set of messages recorded as sent over channel Chij in the state of Pi Recorded_recdij: The set of messages recorded as received over channel Chij in the state of Pj Recorded_sentij = Recorded_recdij implies that all messages sent by Pi have been received by Pj. Hence the channel is empty. Recorded_sentij − Recorded_recdij = φ, where "−" represents the set difference operator, implies that some messages sent by Pi have not been received by Pj. These messages are still contained in channel Chij.

Recorded_recdij −Recorded_sentij = φ, implies that process Pj has recorded as received at least one message that is not recorded as sent by process Pi. This situation indicates inconsistency of the recorded local states of Pi and Pj according to Definition 6.1.

An Algorithm for Consistent State Recording: This section describes the state recording algorithm by Chandy and Lamport (1985). The algorithm makes the following assumptions:

1. Channels are unidirectional.
2. Channels have unbounded capacities to hold messages.
3. Channels are FIFO.

The assumption of FIFO channels implies that messages received by a destination process must be the first few messages sent by a sender process, and messages contained in a channel must be the last few messages sent by a process. To initiate a state recording, a process records its own state and

sends a state recording request called a marker on every outgoing channel. When a process receives a marker, it records the state of the channel over which it received the marker. If the marker is the first marker it received from any process, it also records its own state and sends a marker on every outgoing channel. We use the following notation to discuss how the state of a channel is determined:

Receivedij: The set of messages received by process Pj on channel Chij before it received the marker on channel Chij.

Recorded_recdij: The set of messages recorded as received over channel Chij in the state of process Pj.

Algorithm 6.2 Chandy–Lamport Algorithm

1. When a process Pi initiates the state recording:

   Pi records its own state and sends a marker on each outgoing channel connected to it.

2. When process Pj receives a marker over an incoming channel Chij: Process Pj performs the following actions:

   a. If Pj had not received any marker earlier, then

      i. Record its own state.

      ii. Record the state of channel Chij as empty.

      iii. Send a marker on each outgoing channel connected to it.

   b. Otherwise, record the state of channel Chij as the set of messages Receivedij −Recorded_recdij.

Rules of Algorithm 6.2 are executed atomically, i.e.,as in divisible operations. Recording of the channel state by the algorithm can be explained as follows: Let a process Pi send messages mi1,mi2,...min on channel Chij before recording its own state and sending a marker on Chij. Let process Pj have two incoming channels Chij and Chkj. If the marker on channel Chij is the first marker Pj received, it would record its own state, which would show Recorded_recdij and Recorded_recdkj as the messages received by it. Pj would also record the state of Chij as empty. Because channels are FIFO, process Pj would have received the marker after receiving messages mi1,mi2,...,min on Chij, so it is correct or record the state of channel Chij as empty. Let Pj receive two more messages mk1 and mk2 on Chkj before it received the marker. Hence Receivedkj = Recorded_recdkj ∪{mk1,mk2} and the state of channel Chkj would be recorded as the set of messages Receivedkj −Recorded_recdkj i.e.,{mk1,mk2}.It is correct because process Pk would have sent messages mk1,mk2 before it recorded its own state and sent the marker on channel Chkj, so if these messages were not received by Pi by the time it recorded its own state, they must have been in the channel.