SUMMARY of Chapter 17 (TASK 1)

Distributed System is a system with multiple components located on different machines that communicate and coordinate actions in order to appear as a single coherent system to the end-user.

Benefits of Distributed system:

Resource sharing: sharing of hardware and software resources.
Examples:

Hardware Devices - printers, disks, memory, sensors
Software Sharing - compilers, libraries, toolkits, computational kernels
Data - databases-files

Openness: Systems designed around standard Internet protocols so that equipment and software from different vendors can be utilized together.

Hardware extensions (adding peripherals, memory, communication interfaces..)

Concurrency: In a distributed system, several processes may operate at the same time on separate computers on the network to enhance the performance.

Scalability: Capabilities of the system can be increased by adding new resources based on the requirement.

Fault tolerance: The ability of a system to continue to be dependable (both **available** and **reliable**) in the presence of certain component or subsystem failures by Hardware redundancy or Software Recovery .

Issues in Distributed Systems

Distributed systems are more complex compared to single processor systems. There is no single authority in charge of the entire distributed system. Thus, it causes unpredictability in the operation in the distributed system.

Design Issues :

Transparency: The distributed systems should be perceived as a single entity by the users or the application programmers rather than as a collection of autonomous systems, which are cooperating. Impossible to achieve in real world because there is no central control over the system as a whole.

E.g. :

Access Transparency - enables local and remote objects to be accessed using identical operations (e g., read file..)
File system in Network File System (NFS), SQL queries, and Navigation of the web, ATM, middleware applications used to achieve transparency.

Openness: At the networking level, openness is now taken for granted, with systems conforming to Internet protocols, but at the component level, openness is still not universal.

Scalability: The scalability of a system reflects its ability to deliver high-quality service as demands on the system increase.

- Scaling up means replacing resources in the system with more powerful resources. For example, you may increase the memory in a server from 16 Gb to 64 Gb.
- Scaling out means adding more resources to the system (e.g., an extra web server to work alongside an existing server).

Scaling out is often more cost-effective than scaling up

Security: Attackers may use one system to use it as back door to cripple the complete system. Security mechanisms, such as encryption and authentication, are used to enforce the security policy.

There are multiple types of attack that causes security risks:

- **Interception**
- **Interruption**
- **Modification**
- **Fabrication**

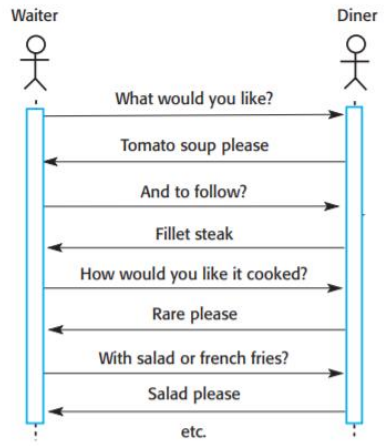
Quality of service: Ability to deliver its services dependably and with a response time and throughput that are acceptable to its users.

Examples of applications where quality is an important concept: movie on demand, tele-conferencing, news casting, tele-immersive group collaborations, multimedia mail system

Models of interaction:

Procedural interaction:

- Synchronous interactions.
  Below is perfect example of procedural interactions.



- Implemented using remote procedure calls (RPCs)

RPC:

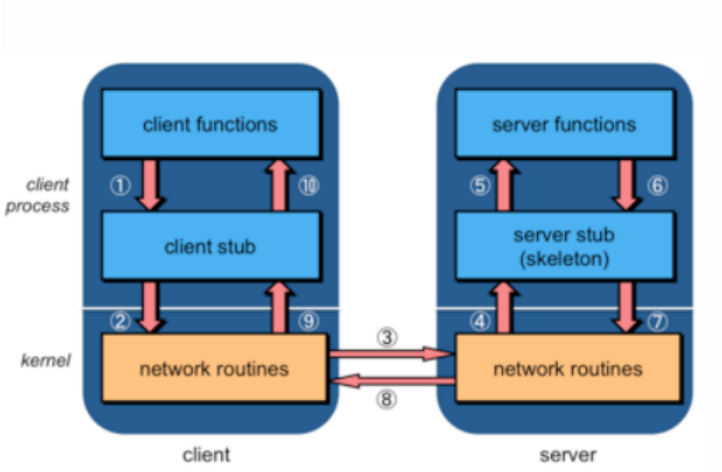
- Remote procedure calls require a “stub” for the called procedure to be accessible on the computer that is initiating the call.
- Converts the representation of the parameters into a standard format, and copying each parameter into the message.
- The client stub passes the message to the transport layer, which sends it to the remote server machine.

- On the server, the transport layer passes the message to a server stub, which **unpacks** the parameters and calls the desired server routine using the regular procedure call mechanism.

- When the server procedure completes, it returns to the server stub which marshalls the return values into a message. The server stub then hands the message to the transport layer.

- The transport layer sends the result message back to the client transport layer, which hands the message back to the client stub.

- The client stub demarshalls the return parameters and execution returns to the caller.



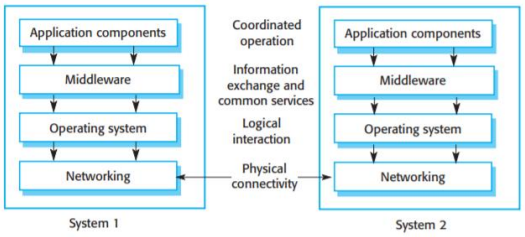
Example: remote method invocations (RMIs)

Message-based interaction:

- Asynchronous interactions.
- Receiver does not have to be active when message is sent by the sender.
- If the system component that is processing the message is unavailable, the message simply stays in a queue until the receiver comes back online.
- Communication is done through **middleware** and the sender and receiver might not know the name of each other.

Use of Middleware

- A software that sits in the middle between application and operating system to stich all the different components together and provides location transparency.
- Models of data, different processor, and protocols for communication may all be different.



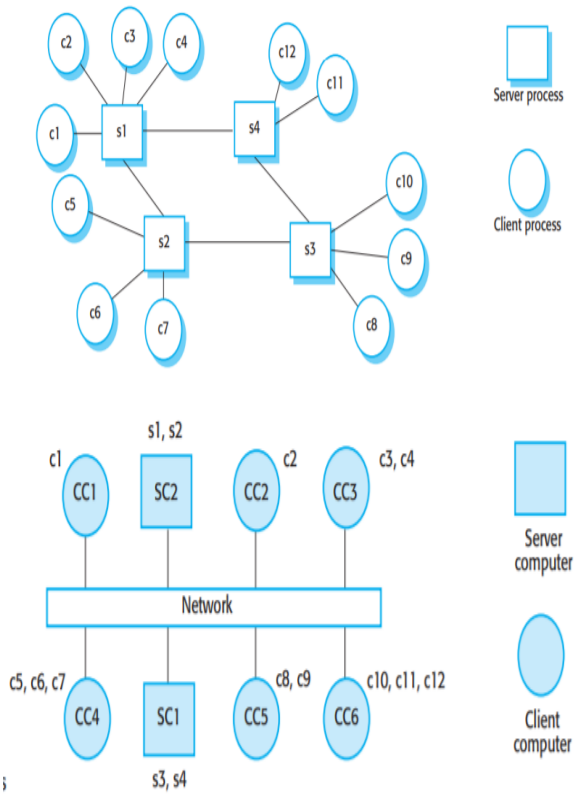
Client Server Model:

Mother of distributed applications systems.

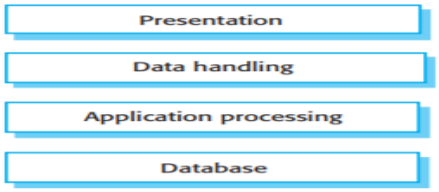
User interacts with a program running on the local computer.

Example: web browser , app on a mobile device.

Mail client, printing client running on PC



- Several different server processes may run on the same processor
- Load-balancing software can be used to distribute work.
- Client–server systems can be designed to keep logical separation between presentation of information and computation.



If application processing logic runs on server – **thin** client

If application processing logic runs on client – **fat** client

Master Slave architecture:

Commonly used for real time systems

Slave processors can be used for computationally intensive operations.

This is used in order to handle I/O bottleneck.

Apache Hadoop HDFS Architecture follows a Master-Slave Architecture, where a cluster comprises of a single NameNode (Master node) and all the other nodes are DataNodes (Slave nodes).

Other different types of architectures:

- Two-tier client–server architecture
- Multi-tier client–server architecture
- Distributed component architecture
- Peer-to-peer architecture