# Distributed Systems

## Assignment-1

S. Gayathri
CSE - III year
Sem:- 06
Regno:- 311519104017

1) Relation to parallel systems.

Parallel processing systems divide the program into multiple segments and process them simultaneously.

The main objective of parallel systems is to improve the processing speed. They are sometimes known as multiprocessor or multicomputers or tightly coupled systems. They refer to simultaneous use of multiple computer resources that can include a single computer with multiple processors, a number of computers connected by a network to form a parallel processing cluster or a combination of both.

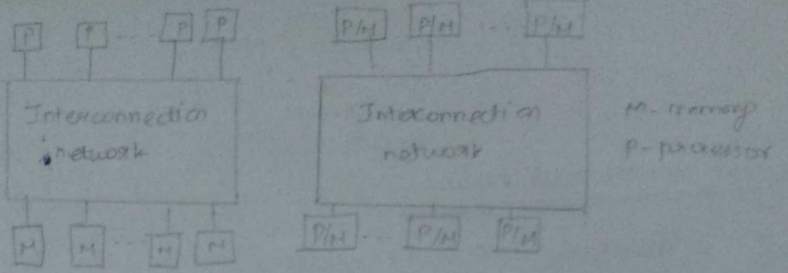
Characteristics of parallel systems.

A parallel system may be broadly classified as belonging to one of three types:

1) A multiprocessor system
2) A multicomputer parallel system.
3) Array processors.

1) A multiprocessor system

A multiprocessor system is a parallel system in which the multiple processors have direct access to shared memory which forms a common address space.




(a) uniform memory access (b) non-uniform memory access

i) Uniform Memory Access (UMA).

* Here, all the processors share the physical memory in a centralized manner with equal access time to all the memory words.

* Each processor may have a private cache memory. same rule is followed for peripheral devices.

* when all the processors have equal access to all the peripheral devices, the system is called a Symmetric multiprocessor.

* when only one or a few processors can access the peripheral devices, the system is called an assymmetric multiprocessor.

ii) Non-uniform Multiple Access (NUMA):-

* In NUMA multiprocessor model, the access time varies with the location of the memory word.

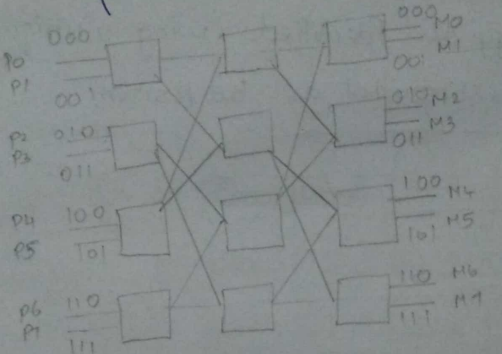
* The collection of all local memories form a global address space which can be accessed by all the processors

The two popular interconnection networks are Omega network and Butterfly network, each of which is a multistage network formed of 2x2 switching elements.

Omega Interconnection function.

The omega interconnection network which connects n processors to n memory units has n/2 log2 n switching elements of size 2x2 arranged in log2 n stages. The generation function as

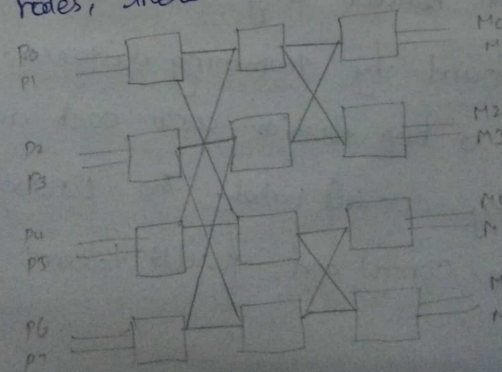
$$j = \begin{cases} 2i & \text{for } 0 \le i \le n/2 - 1 \\ 2i+1-n & \text{for } n/2 \le i \le n-1 \end{cases} \quad \text{where } s \in [0, \log n - 1]$$




3 stage omega network (n=8, m=4)

Butterfly network

A butterfly network links multiple computers into a high speed network. For a butterfly network with n processor nodes, there need to be n(log n+1) switching nodes.



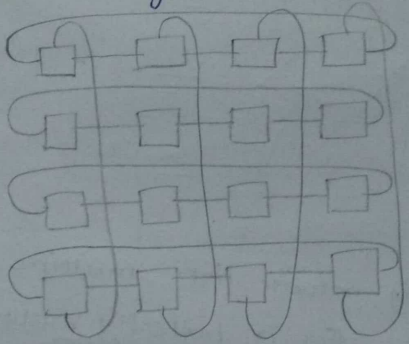

3 stage butterfly network (n=8, M=4)

2) A multicomputer parallel system:

It is a parallel system in which the multiple processors do not have direct access to shared memory. The memory of a multiple processors may or may not form a common address space.

Torus or 2D Mesh Topology:

A k×k mesh will contain $k^2$ processor with maximum path length as $2 * (k/2 - 1)$. Every unit in the torus topology is identified using a unique label, with dimensions distinguished as bit position.

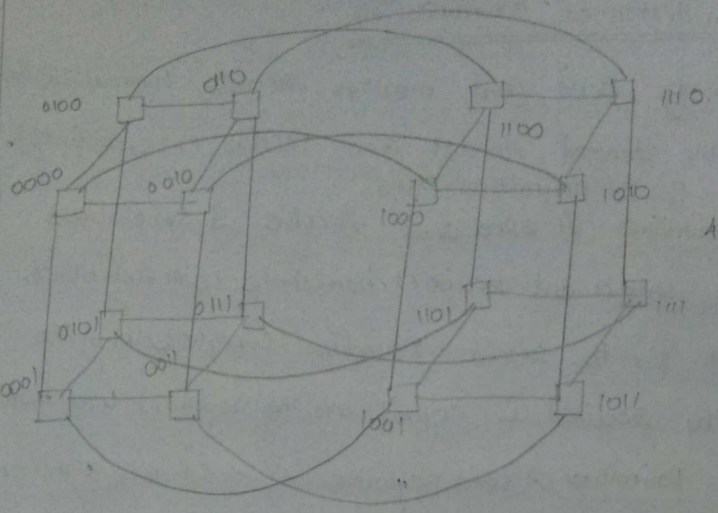


2-D Mesh

Hypercube:

The path between any two nodes in 4-D hypercube is found by Hamming distance. Routing is done in hop to hop fashion with each adjacent node differing by one bit Label. The topology has a good congestion control and fault tolerant mechanism.

2) A multicomputer parallel system:




4-D hypercube

3) Array processors:

"They are a class of processors that executes one instruction at a time in a array or table of data at the same time rather than on single data elements on a common clock".

They are also known as vector processors. An array processor implement the instruction set where each instruction is executed on all data items associated and then move on the other instruction. Array elements are incapable of operating autonomously, and must be driven by the control unit.

## Primitives for distributed communication:

Message send and message receive communication primitives are denoted send() and receive() respectively. It consists of four primitives are,

Synchronous primitives: A send or a receive is synchronous if both the send() and receive() handshake with each other. The processing for the receive primitive completes when the data to be received is copied into the receiver's user buffer.

Asynchronous primitives: A send primitives is said to be asynchronous if control returns back to the invoking process after the data from to be sent has been copied out of the user specified buffer.

Blocking primitives: A primitive is blocking if control returns to the invoking process after the processing for the primitive completes.

Non-blocking primitives: A primitive is non-blocking if control returns back to the invoking process immediately after invocation, even though the operation was not completed.

At the time that wait() is issued, the processing for the primitives has completed, the wait returns immediately.

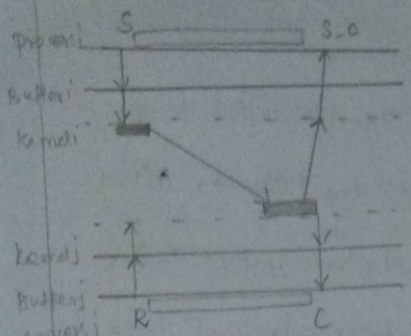
Blocking Synchronous:

The data gets copied from the user buffer to the kernel buffer and is then sent over the network. After the data is copied to the receiver's system buffer and a receive call has been issued, an acknowledgement back to the sender causes control to return to the process that invoked send operation and completes the send. The receive call blocks until the data expected arrives are in user buffer.

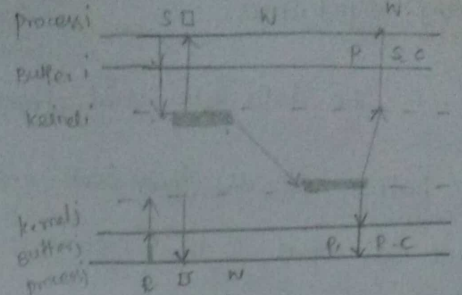
Unblocking (or) Nonblocking Synchronous: control returns back to the invoking process as soon as the copy of data from the user buffer to the kernel buffer is initiated.

Blocking asynchronous: The user process that invokes the send is blocked until the data is copied from the user's buffer to the kernel buffer.

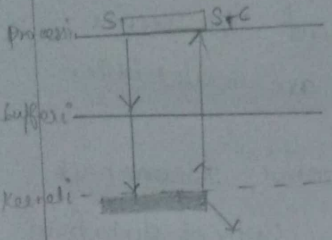
Non-blocking asynchronous: The user process that invokes the send is blocked until the transfer of the data from the user's buffer to the kernel buffer initiated. The checking for the completion may be necessary if the user wants to reuse the buffer from which the data was sent.



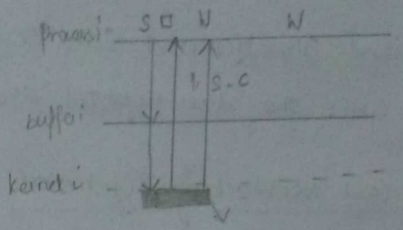

(a) Blocking Synchronous




(b) Non blocking Synchronous



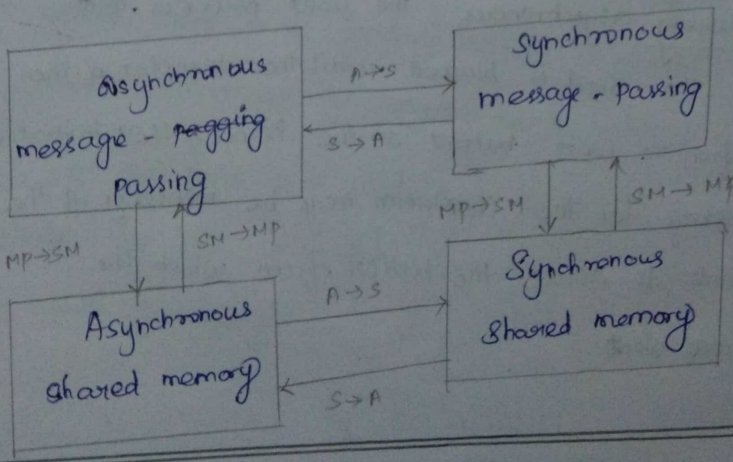

(c) Blocking asynchronous




(d) non-blocking asynchronous

Emultions:

A shared memory system could be emulated by a message-passing system and vice-versa



3) Design Issues and challenges:

Issues and challenges in designing distributed System involves three sub-topics namely

(1) Distributed system challenges from a system perspective.

(2) Algorithmic challenges in Distributed Computing.

(3) Applications of Distributed computing and new challenges.

① Distributed system challenges from a system perspective:

Communication:- Designing appropriate mechanisms for Communication among the processes in network.

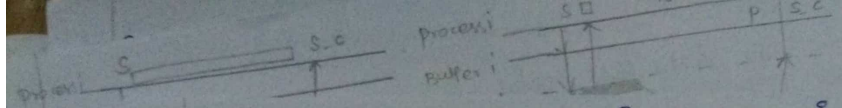
Processes: Management of processes and threads at clients/servers; Code migration; and design of software and mobile agents.

Naming:- Names, Identifiers, addresses is essential for locating resources and processes in transparent and scalable manner.

Synchronization: co-ordination or Synchronization mechanisms among the Processes are essential.

Data storage and access:- Schemes for storage and implicitly for accessing the data is important.

Consistency and Replication:- These are desirable to avoid bottlenecks to provide fast access to data and to provide Scalability.



Fault tolerance :- It requires managing efficient operation inspite of any failures.

Security: It involves cryptography, secure channels, access control, key management and secure group management.

API: (Application programmable Interface) and Transparency.

⇒ API is important for the ease of use and wider adoption of Distributed Services by non-technical users.

⇒ Transparancy involves hiding implementation policies from user.

## ⑧ Algorithmic challenges in Distributed computing.

* Distributed computed involves these following challenges,

→ Designing useful execution models and frameworks.

→ Dynamic distributed graph algorithm and distributed routing algorithm.

→ Time and Global State in Distributed System.

→ Synchronization or coordination mechanisms.

→ Group Communication, multicast and ordered message delivery.

→ Monitoring distributed events and predicates.

→ Distributed programs design and verification tools.

→ Debugging distributed programme.

→ Data replication, Consistency models and Caching

→ WWW (world wide web) design - caching - searching and scheduling.

→ Distributed shared Memory abstraction.

→ Reliable and fault tolerance Distributed system.

→ Load Balancing.

→ Real time scheduling.

→ Performance.

## (3) Applications of Distributed computing and Newer challenges.

Mobile Systems:-

These systems use wireless communication with electromagnetic waves and utilizes shared broadcast medium.

Sensor Networks:-

"Event Streaming" is an important paradigm for monitoring distributed events.

Ubiquitous / Pervasive Computing:-

These systems can be self-organizing and network centric, while also being resource constrained.

Peer-to-peer computing (P2P):-

These networks are self-organizing may or may not have a regular structure of the network.

*Publish, subscribe, Content distribution and multimedia: There on overlap between content distribution mechanisms and publish subscribe mechanisms exists. Multimedia is large information intensive.

Distributed agents:-

These can move around the system to do specific tasks for which they are specially programmed.

Distributed Data mining:-

This algorithms examine large amounts of data to detect patterns and trends in data, to mine useful information.

Grid computing:-

There are a number of challenges in making grid computing a reality.

Security in Distributed system:-

There are several traditional challenges of security in distributed system, these challenges have been addressed in traditional distributed settings.