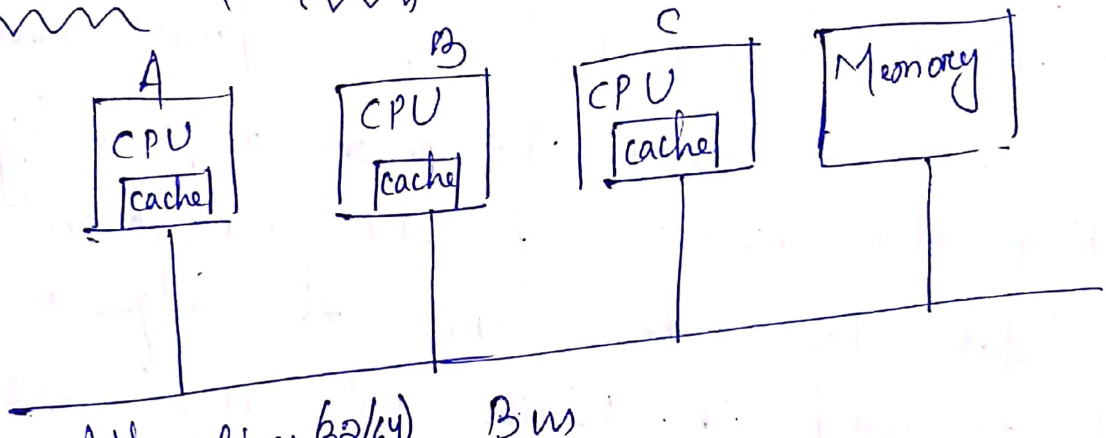
27/7/16

Bus-based multiprocessors



Address lines (32/64)
Data lines (32/64)
Control lines (32)

(If we want to increase the capacity of this system then we have to put a high capacity cache mem. in each CPU. In this case every r/w request may not go to mem. It will go to the mem. only when there's a cache miss.)

This system consists of no. of CPUs all connected to a common bus along with a mem. module. It has a high speed ~~back~~ backplane (backbone) or motherboard into which CPU & mem. cards can be inserted. ~~It's~~ Its bus has 32 or 64 address lines, 32 or 64 data lines & 32 control lines.

To read a word from mem., a CPU puts the addr. of the word on the address lines, then puts a signal on the appropriate control lines to indicate that it wants to read. The mem. responds by putting the value of the word on the data lines to allow the requesting CPU to read it. Write opn also works in the similar way.

If CPU A writes a word to mem. & then CPU B reads that word back, B will get the value written by A. The mem. having this property is said to be Coherent, which plays a crucial role for distributed systems.

This scheme works fine with 4 to 5 processors. With more processors the bus will be overloaded & the performance will drop. For better performance we can use high speed cache mem. b/w the CPU & the bus. All the mem. requests will go through the cache. If the requested word is in cache, it will respond to the CPU & there will be no bus request. If the cache is large enough the hit rate will be high & the amount of bus traffic per CPU will drop dramatically. So this can allow more CPU.

Introduction of cache mem. brings another challenge, i.e. mem. can be incoherent (Suppose CPU A & B read the same word from mem. & then A overwrites the word. When B reads that word next time, it gets the old value from its cache but not the updated value written by A.).

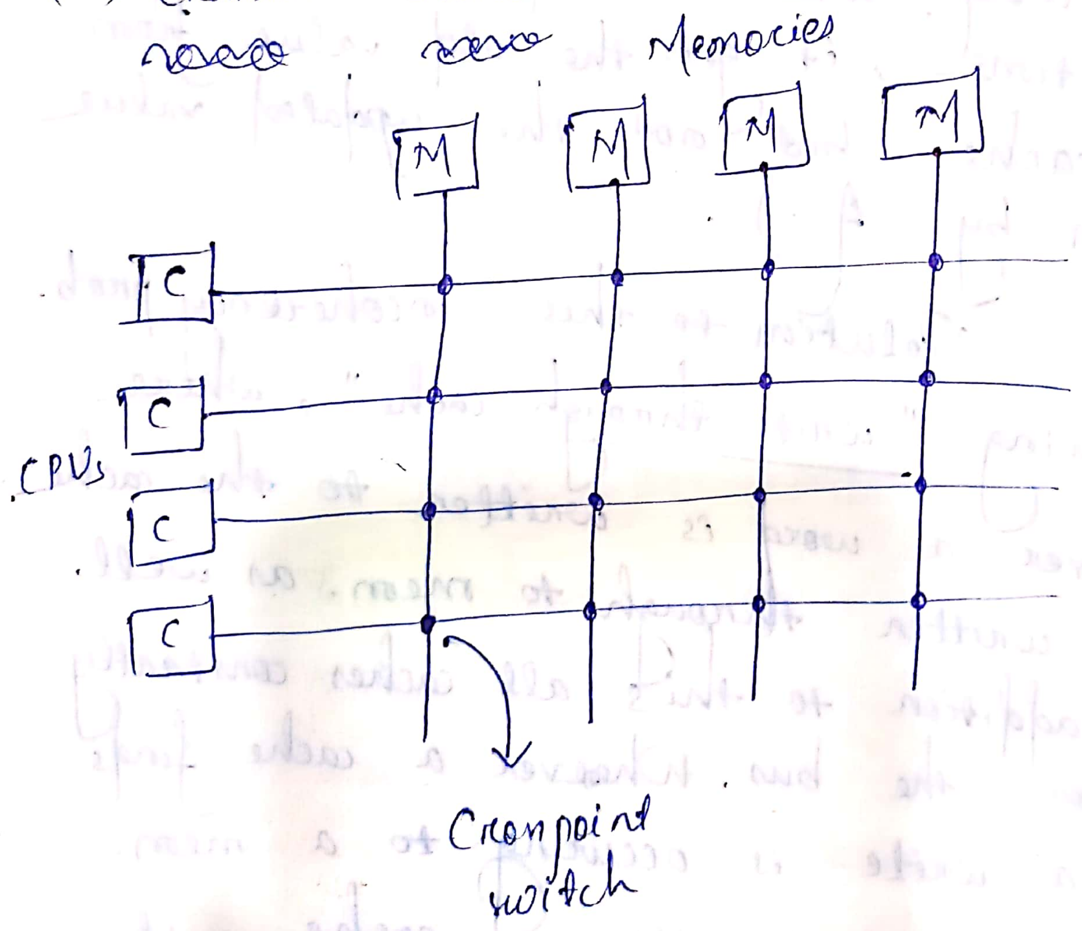
Solution to this incoherency prob. is using "write-through cache", where whenever a word is written to the cache it's written through to mem. as well. In addition to this all caches constantly monitor the bus. Whenever a cache finds that a write is occuring to a mem. address present in its cache, it either removes that entry from its cache or it updates the cache entry with the new value. Such a cache is called "Snoopy cache" and this process

is called Snooping (Eavesdropping in networking).
A design consisting of "snoopy write-through cache" is coherent if is invisible to the programmer.
Using this design we can put upto 64 CPUs on a single bus.

1/8/16

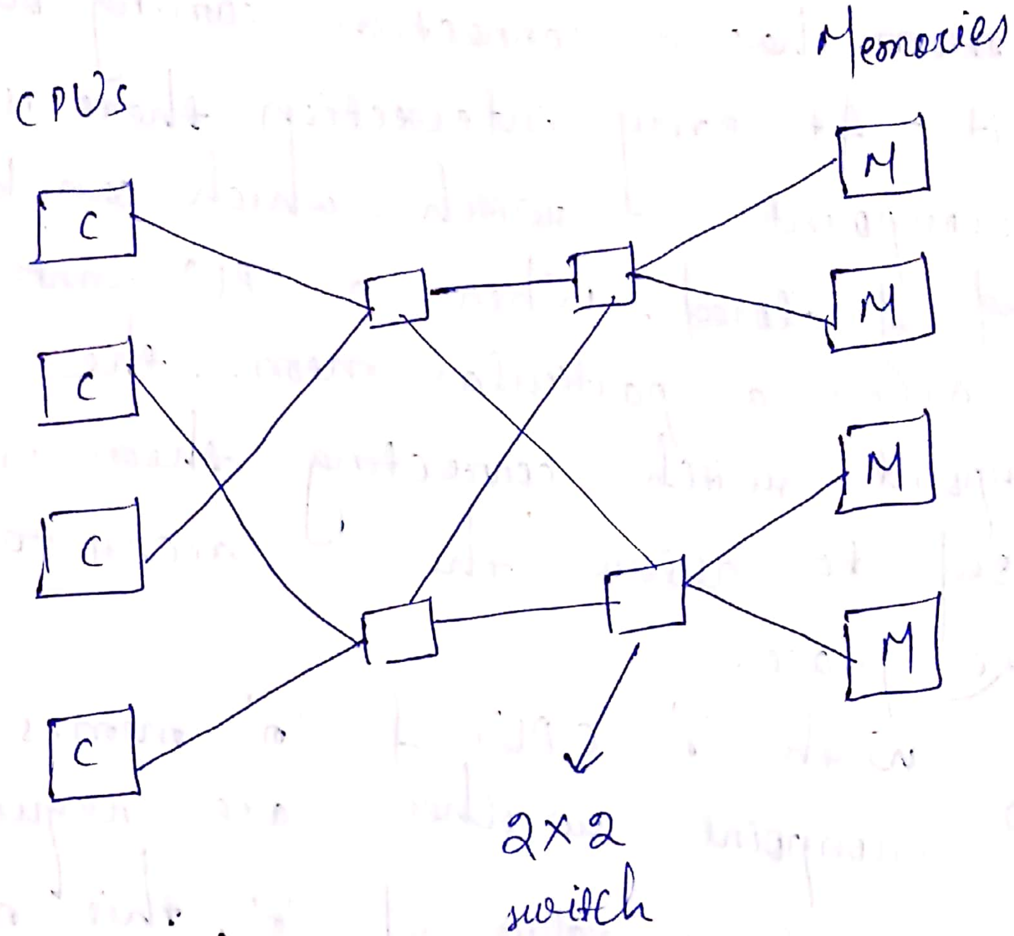
Switched Multiprocessors

(a) Crossbar Switch :-



Total no. of switch = $n \times n = n^2$

(b) Omega switch network :-



Total no. of switch = $\log_2 n$ switching stages × $\frac{n}{2}$ switches

$= \frac{n \log_2 n}{2}$ switches

(Here 4 CPUs = 2^2 → 2 switching stages.)

To build a multiprocessor with more than 64 processors, we can divide the mem. into modules & connect them to CPUs with switches. There are 2 ways to do this:

• Crossbar switch, where each CPU & each memory has a connection coming out of it. At every intersection there is a crosspoint switch, which can be opened & closed. When a CPU wants to access a particular memory, the crosspoint switch connecting them is closed to allow the access to take place.

With 'n' CPUs & 'n' memories, n^2 crosspoint switches are required. For a large value of 'n', this no. becomes ~~prohibited~~ prohibitive.

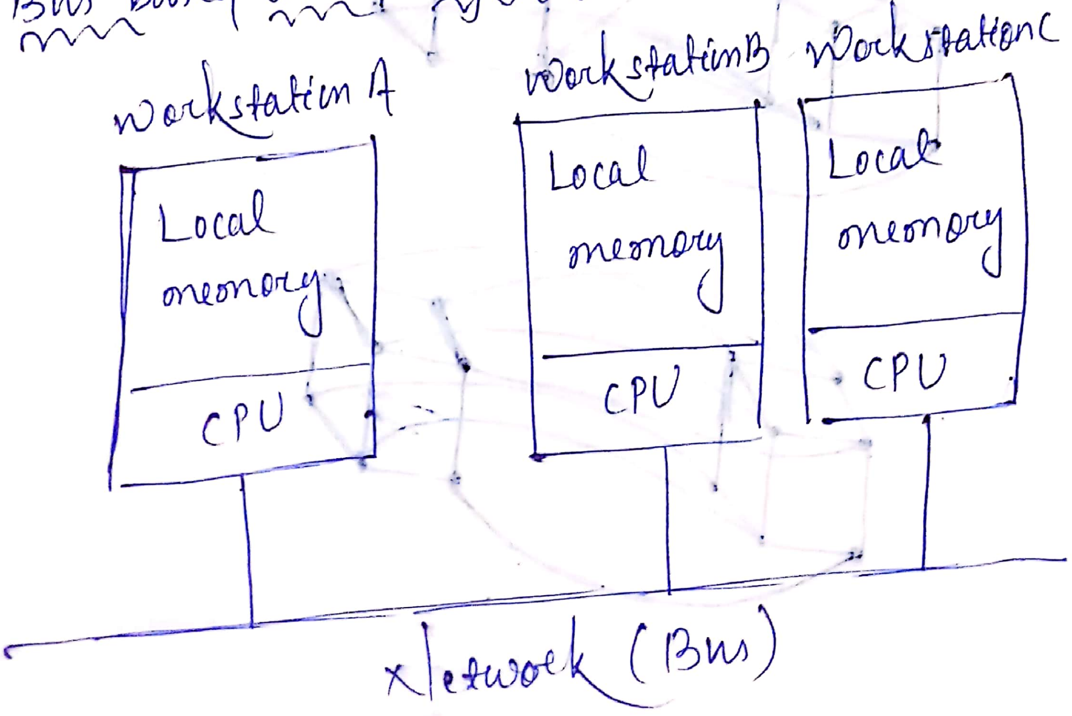
• Omega switch network :- In the example there are 4 nos of 2x2 switches, each having 2 i/ps & 2 o/ps. Each switch can route either i/p to either o/p. Every CPU can access every memory.

With 'n' CPUs & 'n' memories, the omega network requires $\log_2 n$ switching stages, each containing $\frac{n}{2}$ switches, for a total of $\frac{n \log_2 n}{2}$ switches.

For a large value of 'n', this no. is better than n^2. But it introduces a delay factor, because it has to move through multiple switching stages. (Ex→ For n = 1024, there will be 10 switching stages & in addition there will be 10 stages further requested words to come back, for a total of 20 stages movement.) ↳ The conclusion is building a large, tightly coupled, shared mem. multiprocessor is possible but ~~each~~ is difficult & expensive.

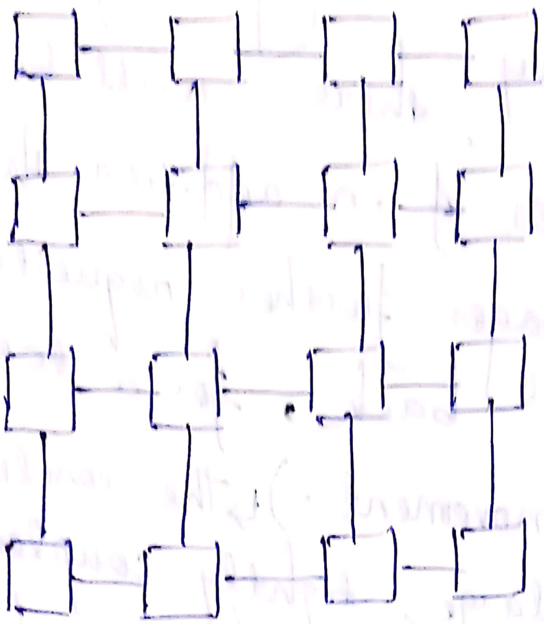
3/8/16 (page 1)

Bus based multicomputers

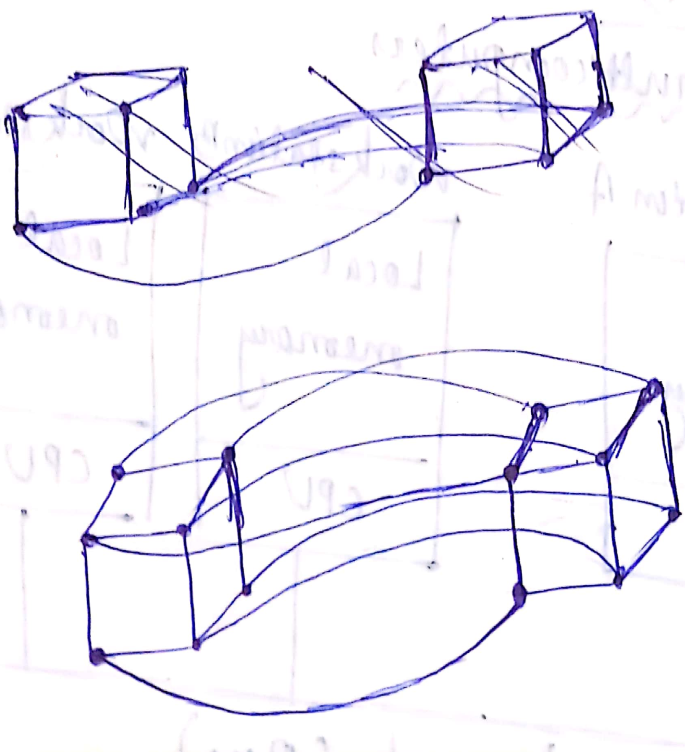


Switched multicomputers (page-3)

(a) Grid



(b) Hypercube



(Fig.) There are 2 topologies for switched multicomputers:

- Grid → Grids are easy to understand & its layout on printed circuit boards. They are best suited for problems, which are inherently 2-dimensional in nature, like Graph Theory problems.

- Hypercube → It's a n-dimensional cube. In the figure it's 4-dimensional. It consists of 2 cubes, each with 8 vertices & 12 edges. Each vertex is a CPU & each edge is a connection b/w 2 CPUs. The corresponding vertices of 2 cubes are connected.

For a n-dimensional hypercube, each CPU has n-connections to other CPUs. The complexity of wiring increases by logarithm. Since only the nearest neighbours are connected, many messages have to make several hops to reach their destination.

Software Concepts

The image that a system presents to its users & how they think about the system is ~~not~~ largely determined by the OS s/w. There are 2 kinds of OS, e.g.

- For multiple CPU systems (multiprocessor or multicomputer)
- Loosely-coupled
- Tightly-coupled

(i) Loosly coupled OS :-

" " s/w allows machines & users of a system to be independent of one another, but still interact to a limited degree whenever necessary. It's a group of PCs, each having its own CPU, its own mem., its own harddisk & its own OS. But they share some resources, like printers, databases ~~etc~~ etc. over the ~~length~~ LAN. Ex-> Novell Netware is a loosly coupled os.

(ii) Tightly coupled OS :-

" " systems are multiprocessor systems dedicated to running a single program, like chess prog. in parallel. Each CPU is assigned a board to evaluate & it spends its time examining that board. This kind of appl^n's are tightly coupled.

Four combinations of h/w & s/w

① Loosely (NOS) coupled s/w on loosely coupled h/w
Not used in reality ② " " " " tightly " "
DOS ③ Tightly " " " loosely " "
④ " " " " tightly " "
→ Multiple processor time sharing system (here mem. & load are also shared)

NOS → Network OS
DOS → Distributed OS

(page- 2)

In the multicomputer system there is no shared mem. Each CPU has direct connection to its own local mem. The interconnection network is not used for CPU to memory traffic. It is needed only for CPU to CPU communication. Therefore there will be much less traffic over the network

(contd...)

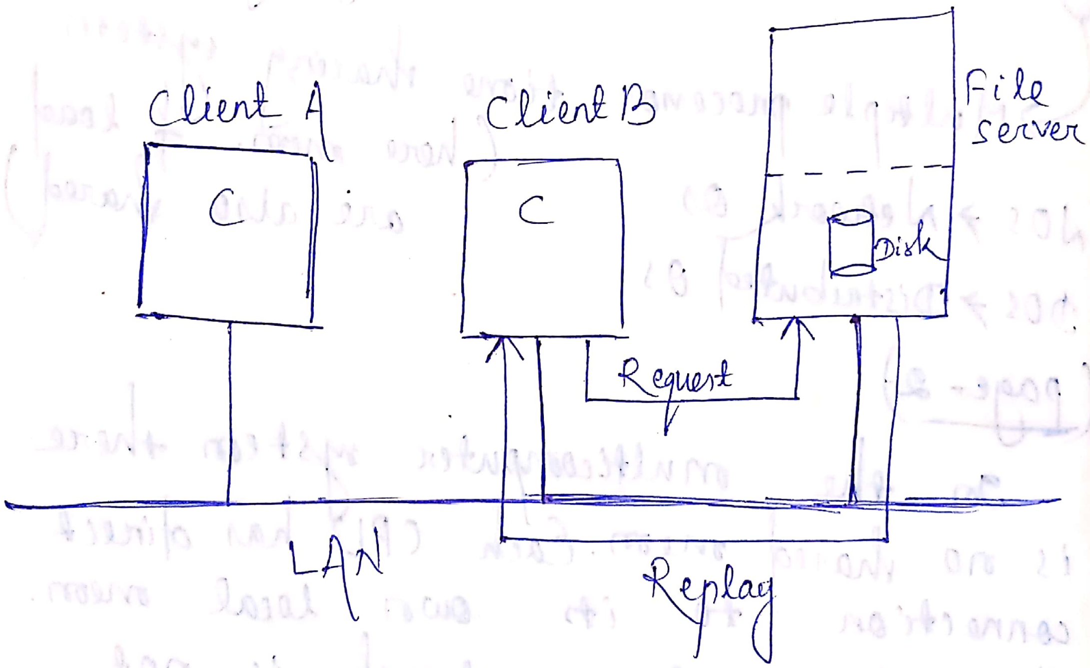
(contd...)
If a high speed bus is not required. Rather a lower speed bus, like (10-100) Mbps (ethernet speed is 10Mbps) length can be used for communication.

As compared to 300Mbps high speed bus, this kind of bus is basically a collection of workstations over a LAN

5/8/16

Network Operating System

(iii) These are loosely-coupled s/w on loosely-coupled h/w.



It's a network of workstations connected by a LAN. Each user has a workstation for his/her exclusive use. It may or mayn't have a hard disk. It will have its own operating system. All commands (Appl^n layer in OSI model deals with incompatibility) are run locally right on the workstation. Any user can remotely log into another workstation. After this login the user's own workstation is turned into a remote terminal, logged into another remote machine. At any point of time only one machine can be used & the selection of machine is entirely manual. It provides a facility for remote ~~log~~ copy command (rcp) to copy files from one machine to another machine.

Ex→ rcp m1:f1 m2:f2

Here file1 from machine1 is coppied into file2 in machine2.

the movement of files is explicit & requires the user to be aware of where all the files are located & where all commands are being executed. This approach provides a shared, global file system accesible from all workstations. The file system is supported by one or more file servers. The file servers accept requests from the user programs running on the other machines called ~~etc~~ clients to read & write files. Each incoming request is examined & executed & the replay is sent back. The file servers maintain hierarchical file systems. The work stations can import or mount these file systems, augmenting their local file systems with ~~to~~ those located on the servers. (In case of import, the server files are coppied. In case of mount, a link is created to server files)

(command for mount:

mount filename destination

command for unmount: umount)

The OS used in this kind of environment manage the individual workstations & file servers & also take care of communication b/w them. It's posible that all the machines run the same OS, but it's not mandatory.

If the clients & servers run on diff systems (diffn os), they must agree on the format & meaning of all messages they exchange. (This process is called handshaking & the rules upon which they are agreed are called protocols.)

Distributed Operating System

These are tightly-coupled s/w on loosely-coupled h/w.

The goal of this system is to create the illusion in the minds of the users that the entire network of computers is a single time sharing system, rather than the collection of machines. It's also referred to as Single System Image. In other words distributed system runs on a collection of networked machines but acts like a virtual uniprocessor. Here the essential idea is that the users shouldn't be aware of the existance of multiple CPUs in the system.

* Characteristics :-

- There must be a single, global interprocess communication mechanism so that any process can talk to any other process in the system.

(Ph. call → direct communication
Message sending → indirect ")

It can't have diffⁿ mechanisms on diffⁿ machines ~~for~~ or diffⁿ mechanisms for local communication & remote communication.

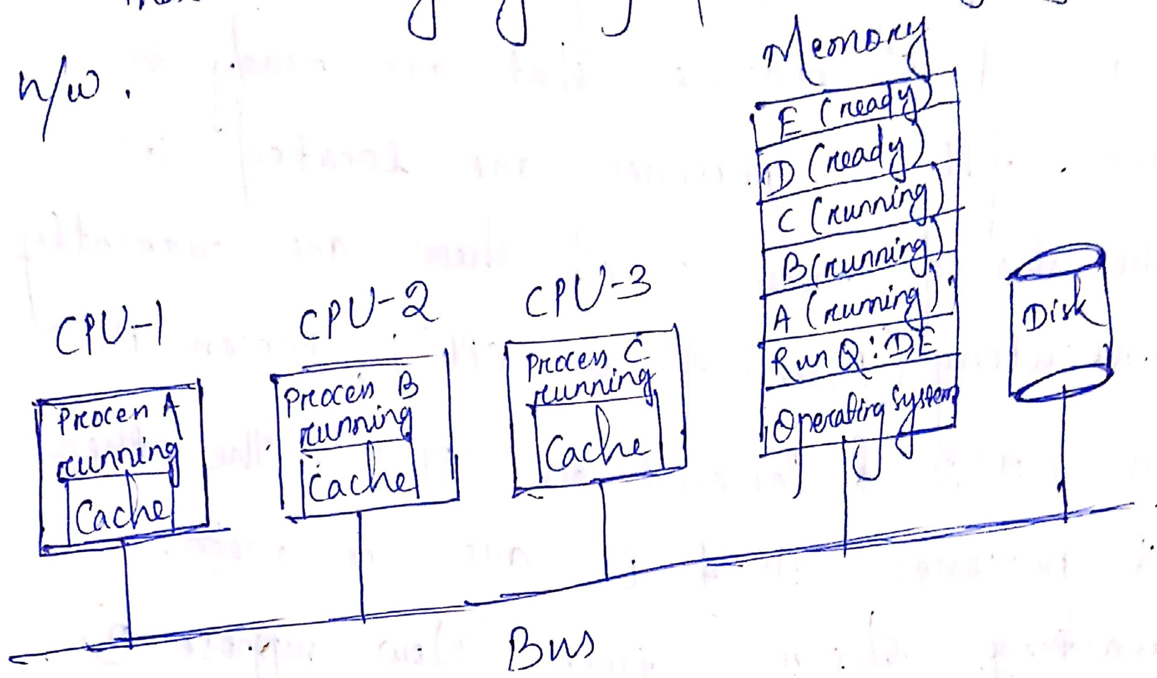
- Process management must be same everywhere. How processes ~~processors~~ are created, destroyed, started & stopped must not differ from machine to machine.

- File system must look the same everywhere. Having the file names restricted to certain characters in some locⁿ's & being unrestricted in other locⁿ's isn't allowed. Every file should be visible to every locⁿ subject to protection & security.

8/8/16

Multiprocessor Timesharing Systems

These are tightly-coupled s/w on tightly-coupled h/w.



(A multiprocessor with a single run Q)

These systems are used as dedicated database machines. These are the examples of multiprocessors that are operated as a UNIX timesharing system, but with multiple CPUs instead of one CPU. It uses time sharing as well as load sharing. The main characteristics of this class of system is their existance of single run Q, which contains the list of processes in the system that are ready to run. This run Q is the data structure, which is kept in the shared mem.

In the example there are 3 CPUs & 5 processes that are ready to run. All 5 processes are located in the shared mem. 3 of them are currently executing (Process A on CPU 1, Process B on CPU 2 & Process C on CPU 3). The other 2 processes D & E are in mem. waiting their turn. Now suppose B blocks for I/O or its TQ (Time quantum) runs out, CPU 2 must suspend it & find another process to run. After saving all the registers related to B, it will run the critical section to look for another process to run. It's essential that the scheduler be run as a critical section to prevent 2 CPUs from choosing the same process to run. Once the CPU 2 has gained exclusive access to Run Q, it can remove the 1st entry from the Q (i.e. D), exit from the critical section & then begin executing D. Initially execution will be slow, because cache mem. of CPU2 still contains data related to process P2. And after a while the cache mem. will hold the data & code related to process P1 & the execution will speed up.

Because none of the CPUs can have local mem. & all progs. are stored in the shared mem., it doesn't matter on which CPU a process runs. If a long running process is scheduled many times before it completes, on the average it will spend same amount of time running on each CPU. We can have a slight gain in performance, when a process runs on a CPU, which is currently caching parts of that process. If all the CPUs are idle, waiting for I/O & one process becomes ready, it's preferable to allocate it to the CPU it was last running, assuming that no other process has used that CPU in b/w.

If a process blocks for I/O on a multiprocessor, the OS has the choice of suspending it or just let it do busy waiting. If the I/O is completed in less time than it takes to do a process switch, busy waiting is preferable.

***File system:**

The OS contains a traditional file system, including a single, unified block cache. When a process executes a system call, a trap is made to the OS, which carry out the opⁿ using critical section to lock out other CPUs, while critical sections are being executed. On the whole the file system is hardly diffⁿ from a single processor file system.

Comparision of 3 categories

Item	NOS	DOS	MTS
① Does it look like a virtual uniprocessor?	No	Yes	Yes
② Do all have the same OS?	No	Yes	Yes
③ How many copies of the OS are there?	N (no. of copies for N sys.)	N (no. of copies for N sys.)	1

Item	NOS	DOS	MTS
(4) How is communication achieved?	Shared files	Message passing	Shared mem.
(5) Are agreed upon network protocols required?	Yes	Yes	No
(6) Does file sharing have well defined semantics?	Usually No	Yes	Yes

10/8/16

Design Issues (for Distributed System)

(1) Transparency :- It's related to how to achieve single system image, i.e. to fool everyone into thinking that the collection of machines is the old fashioned time sharing system. Transparency can be achieved at 2 levels; i.e. user & system level. At the lower level it's possible to make the system look transparent to the programs. A system call interface can be designed in such a manner that existence of multiple processors is not visible.

* Several aspects of transparency :-

(a) Location transparency :- The users can't