3/8/18

811

Distributed OS = distributes load (space distribution)

IPC	RPC
(1). Inter Process Communic.	(1). Remote Procedure call (remote)
(2). Same machine	(2). Diff. mach.

Intraprocess Communic. - within a particular process (many threads) - Multiple threads running within the process.

Obj. > Java is abided by fully OOPS.
> POP vs OOP.

4/8/18 | Obj. Read -> No polymorphic ; Already classes are defined. Eg. - VB, ADA.

IPC: Mechanism that allows exchange of data b/w processes, by providing a user with a set of programming interfaces. IPC helps a programmer organize the activities among diff. processes. It allows 1 applic. to control another thereby enabling data sharing without interference.

IPC mechanism can be classified into pipes (filtering), FIFO & shared memory.
" were introduced in UNIX OS where data flow is unidirectional. Working principle of FIFO is based on data stru., FIFO FCFS basis. " " also unidir. & is identified by access pts.
In shared mem. data sharing is done based on connection oriented or connection-less protocols.

In IPC data communic. is allowed using segments, semaphores & other methods for info. sharing. IPC facilitates efficient msg. transfer b/w processes. The idea of IPC is based on task control architec. (TCA). It is a flexible technique that can send & receive variable length arrays, data struc. & lists.

Main IPC Methods:

Method	Descrip.
File	A record stored in disk that can be accessed by name for any process.
Signal	A sys. msg. sent from 1 process to another, not usually stored but instead provide commands.
Socket	A data stream sent over a N/W interface either to a diff. process in same comp. or to another ".
Msg Queue	An anonymous data stream similar to a pipe, but that stores & retrieves info in pckts.
Pipe	A 2-way data stream interfaced thro' std I/P & O/P & is read char by char.

Named Pipe	A pipe implemented thro' a file on a file sys. instead of std. I/P & O/P.
Semaphore	A simple structure that synchronizes threads or process acting on the shared resources.
Shared Mem.	Multiple process given access to same memory allowing all to change & read the changes made by other processes.
Msg. passing (Shared nothing)	similar to msg. queue.
Mem. Mapped File	A file mapped to the RAM & it can be modified by changing the memory addresses directly instead of O/Ping to a ~~string~~. ~~Stream~~. It shares the same benefits as a std. file.

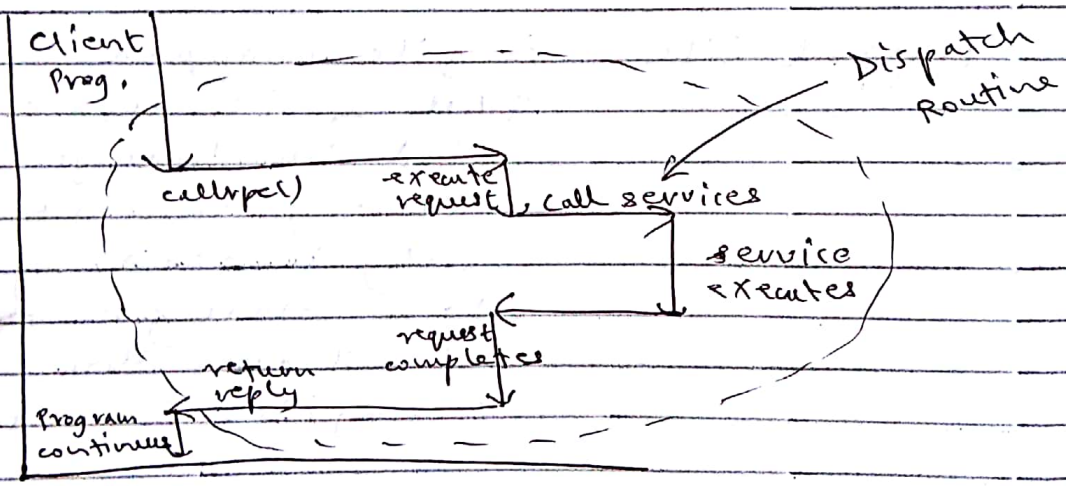
RPC: is also an IPC technology that allows a comp. prog. to cause a subroutine or procedure to execute in another addr. space. ⟹ without the programmer explicitly coding the details for this remote interaction.

So RPC is just one kind of IPC viz a set of methods for exchange of data among multiple threads in 1/more processes.

The processes may be running in 1 or more comp. connec. by a N/W but the IPC methods are divided into methods for msg. passing, synchron., shared mem. & RPCs. The method of IPC here may vary based on the bandwidth & latency of communication b/w the threads & type of data being communicated. Several reasons for providing an envt. allowing proces co-oper. are:

1. info sharing
2. Speed up
3. Modularity
4. Convenience
5. Privilege separation

Q. How RPC works?

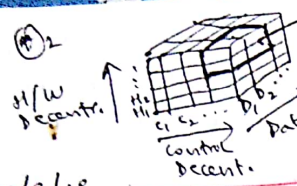


An RPC is analogous to a func. call like a func. call, when an RPC is met. The calling args. are passed to the RP & the caller waits for a response to be returned from the RP. The thread is blocked from processing until either a reply is received or it times out.

When the request arrives, the server calls a dispatch routine that performs the requested service & sends the reply to the client. After the RPC call is completed the client prog. continues. RPC specifically supports N/W architec.

In the fig. the RPC mechanism is uniquely identified by a triplet (prog. no., version no. & procedure no.). " identifies group of related remote procedures which are unique, each prog. consists of multiple versions, " version " " a collec. of procedures available to be called remotely.

(*)2



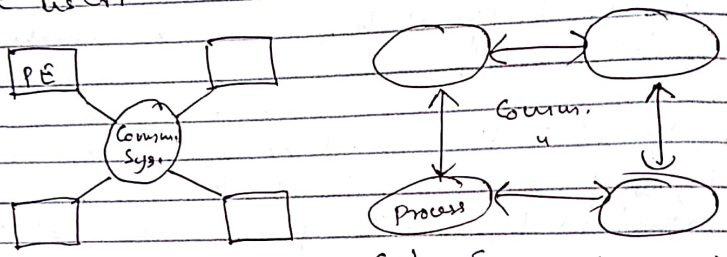
> Data cube concept - combo of 3 diff axes

10/8/18

SM

> A distributed sys. is a collec. of independent computers that appear to the users of the sys. as a single comp.

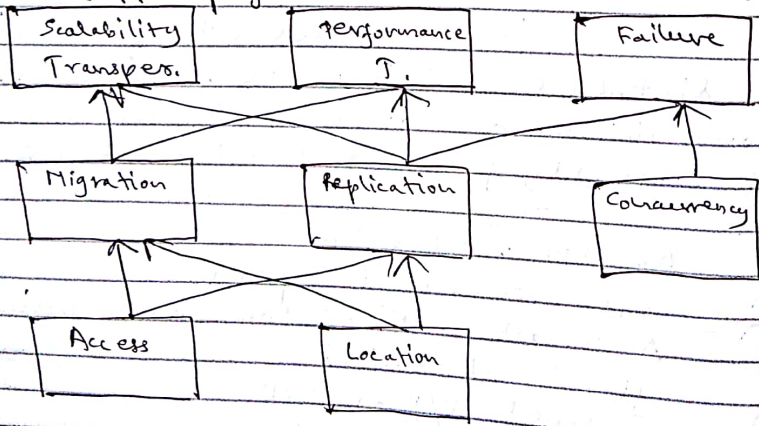
> Distributed systems are "seamless", the interfaces among functional units on the N/W are for the most part invisible to the user.



(a). Sys. structure from physical view

(b). Sys. struc. from logical view

(*)2 Transperancy - Distrib. sys. should be perceived by users & applic. programmers as a whole rather than a (*)

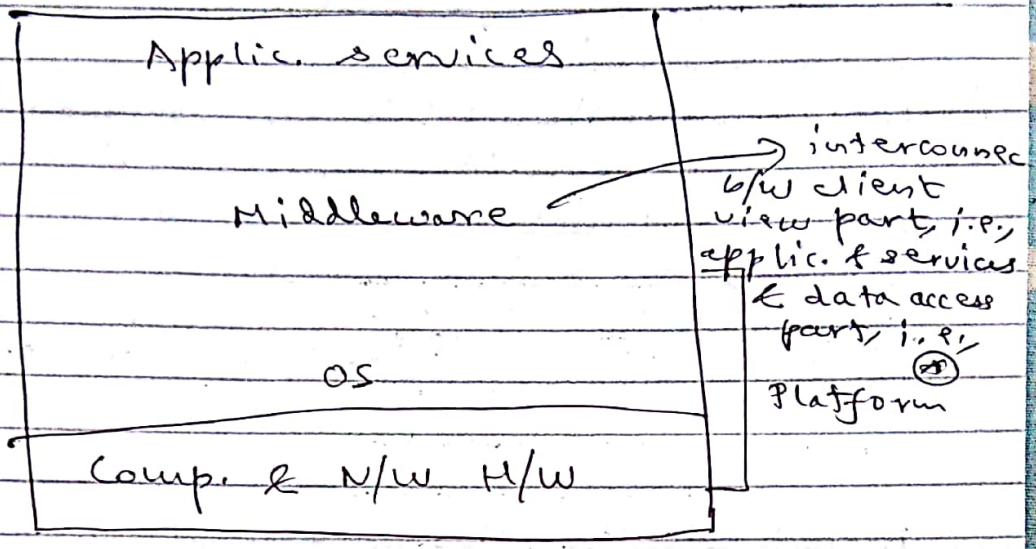


↑ determines a bottom-up appr., i.e., the dependency

(*) collec. of cooperating components. Transp. has diff. dimen- -sions that were identified by ANSA. These rep. various p"/ -erties " dist. sys. should have.

(*) db part. It deals with business logic.
Thin server fat client - We are providing middlew along with client.
Fat & thin c. - Student site. stud./customers are not allowed to modify any news or notice.

S/W & H/W service layers in distributed sys.



A working model of a distrib. sys. -> (**) #

" service provided by multiple servers

" distributed applic. based on peer processes

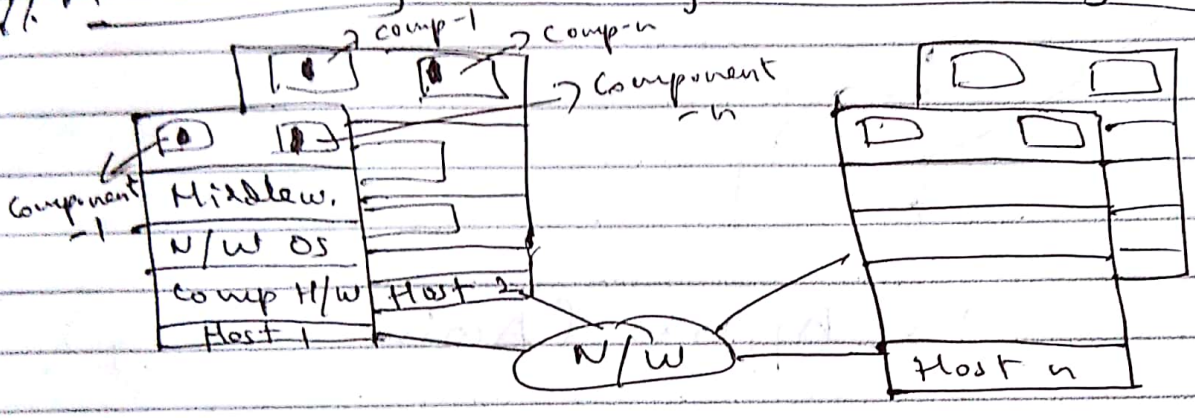
Web applets -> active web pgs.

~~The clients~~ Thin clients & compute servers.

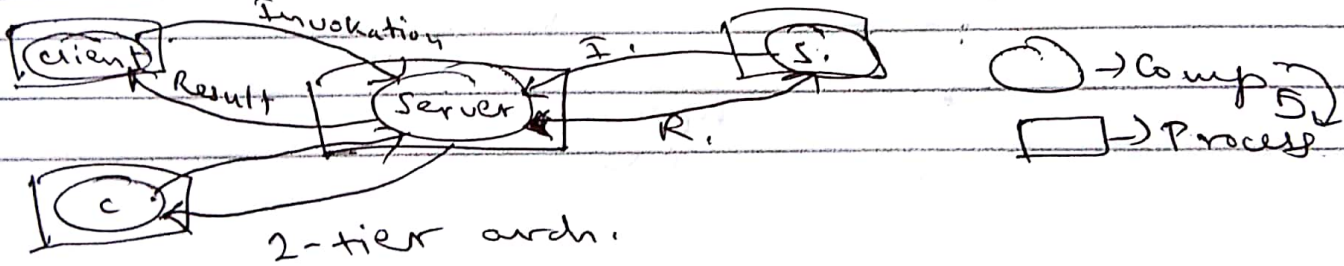
CIA upto fault tolerance

(20) 11

A Working Model of a dist. sys

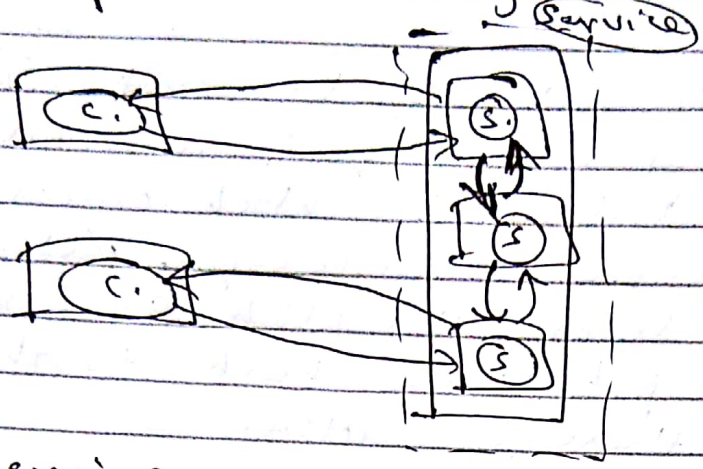


Clients invoke individ. servers



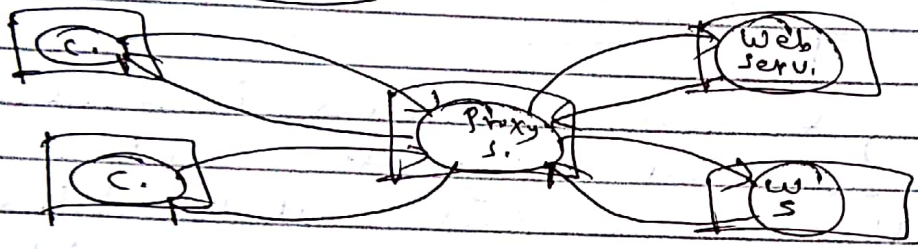
2-tier arch.

Proxy Server
Backup " - When sys crashes

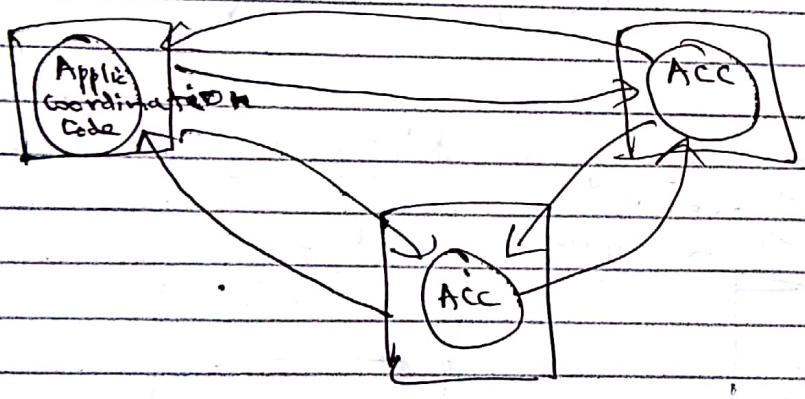


A service provided by multiple servers.

Web proxy s.



A dist. applic based on peer processes

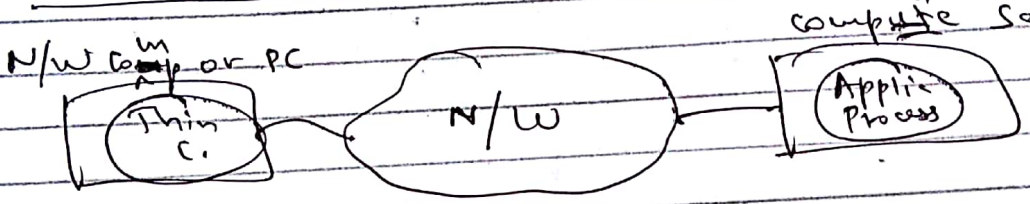


Web Applets (Active web pg)

a). Client request results in downloading of applet code

b) Client interacts with the applet

Thin client & Compute servers



11/8/18

Fault Tolerance :

A system's ability to tolerate failure

1. Reliability : likelihood that a sys. will remain operational for the duration of a mission.
- requirement might be stated as 0.9999999 availability for a 10 hr mission → probab of failure during the mission must be at most 10^{-6}.
- very high reliab. is most imp in critical applic. - space shuttle, industrial controls in which failure could mean loss of life.

Availability: expresses the fraction of time a sys. is operational.

> A sys. with high availability may in fact fail > its recovery time & failure frequency must be small enough to achieve the desired availab.

- high availab. is imp in airline reservations, telephone switching, etc, in which every minute of downtime translates into revenue loss.

Importance of Design

> A good fault-tolerant sys. design req. a careful study of failures causes of failures, & sys. responses to failures.

> Such a study should be carried out in detail by the Requirement specification - 1

design begins & must remain part of the designing process.

SM's notes (contd.):

(*) 4 Req. Specifi.:

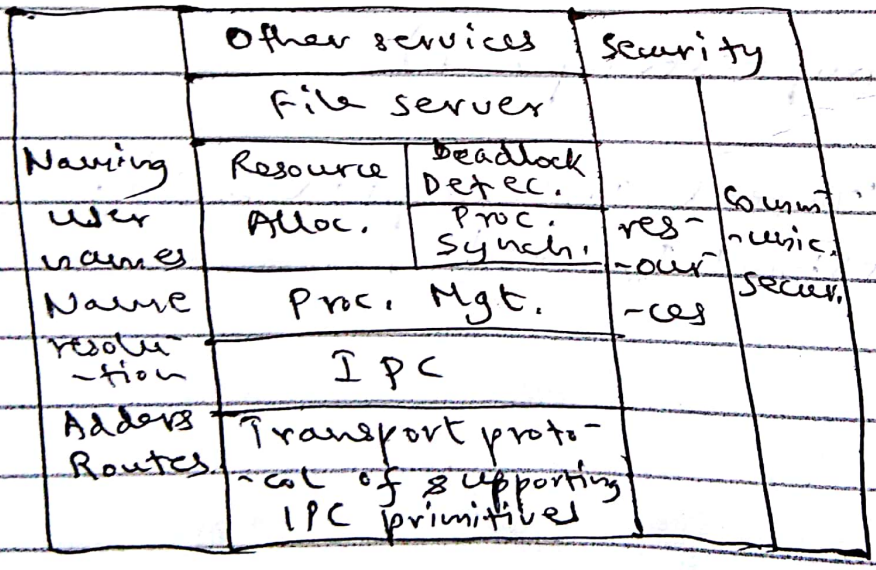
- Planning to avoid failures is most imp.
- A designer must analyze the envt & determine the failures that must be tolerated to achieve the desired level of reliab.
- To optimise fault tolr., it's imp to estimate actual failure rate for each possible failure.

Notes of SM

10/8/18:

Estructure of a dist. syst:

↳ electronic struc.



Classific. of faults:

- Transient " - occur once then disappear.
- Intermittent " - " , goes away, then goes back, " " "
- Permanent faults - doesn't go away by itself, like disk failures.

Failure types:

- Some are more probable than others
- " transient others permanent
- " occur in H/W, " in S/W,

Design issues of Fault Tolerant Sys. :

Design-1

(1)

> Basic Principle - redundancy
> - Spatial - redundant H/W
> - Informational - " data struc.
> - Temporal - " computation.

(2) Design of sys. that tolerate faults that occur while sys. is in use.

D-2

> Redundancy costs money & time.
> Hence - one must optimize the design by trading off amt of redundancy used against derived level of fault tol.

SM's Notes:

Design Issues:

D-2 (contd.)

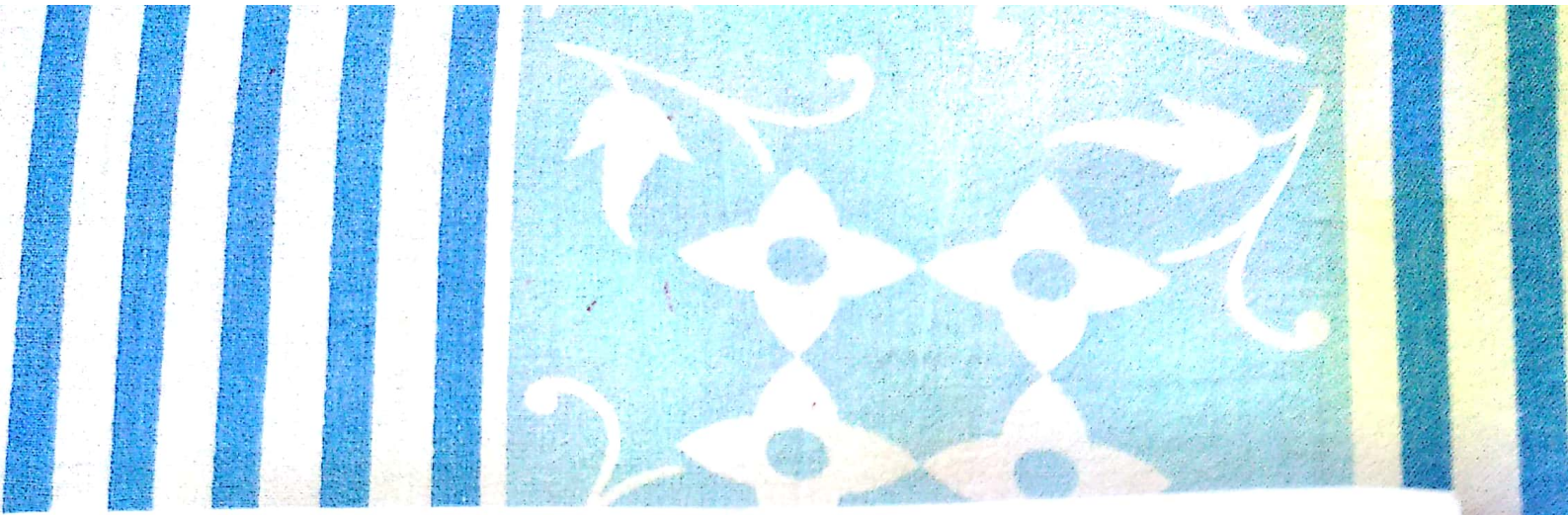
- Temporal redundancy usually requires re-computation & it results in slower recovery from failure.
- Spatial has faster recovery but ↑ h/w costs, space, power, etc. requirements.

D-3

7 Commonly used techniques for redundancy
- Modular redundancy.

D-3 (c)

- Modular redundancy
  - Uses multiple, identical replicas of H/W modules & a voter mechanism.
  - The o/p's from the replicas are compared & correct o/p is determined - majority vote.



- Can tolerate most H/W faults that can affect minority of the H/W modules.

D-4

> N-version Programming

D-4(c.)

N-V. Prog.

- Write multiple versions of a s/w module
- O/Ps from these n are received & correct O/P is determined via voting mech.
- Each ver. is written by diff team, with the hope that they will not contain the same bugs
- Can tolerate s/w bugs that affect a minority of versions.

– can't tolerate correlated fault - reason for failure is common to 2 (or more) modules eg. 2 modules share a single power supply failure of which causes both to fail.

Commit in SQL creates a checkpt.
Rollback - move back to the 4.

Error - Control Coding - 2

Recovery Blocks

18/8/18

SM

Dependability Evaluation-1 :

> Once a sys. has been designed, it must be evaluated to determine if it meets reliability & dependability objectives

> 2 " approaches :

> Use an Analytical Model
- can help developers to determine a system's possible states & possibilities of transitions among them
- can be difficult to analyze models accurately

> Injecting faults
- Various types of faults can be injected to determine various dependability metrics.

[Smoke testing - how many process can run at same time, Load testing, Otherw. crash/hang

Dependability Evaluation-2 :

> In distributed systems a transac. based service can accept occasional failures followed by a lengthy recovery procedure

> A real-time service - Process control
- may have I/Ps that are readings taken from sensors
- may have o/ps to actuators used to control a process directly or " activate alarms so that humans can intervene in the process.
- due to strict timing requirements, recovery must be achieved within a v. small time limit, e.g., air traffic control, monitoring patients, controlling reactors.

→ V. V. imp.

Fault Models

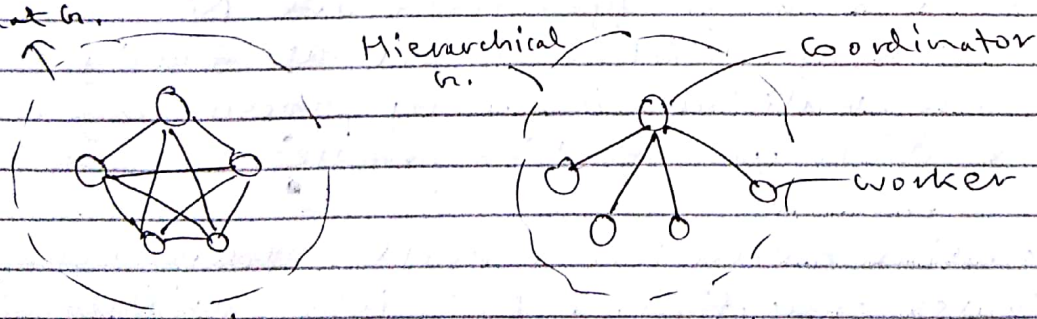
> Omission failure
- A server omits to respond to a request or receive request

> Response Failure
- Value " – returns wrong val.
- state transition failure – has wrong effect on resources

> Timing failure – any response that is not available to a client within a specified real time interval . . .

> Server crash failure – a server repeatedly fails to respond to requests until it's restarted
- Amnesia – crash – a server starts in its initial state, having forgotten its state at the time of the crash, i.e., loses the values of the data items.
- Pause – crash – a server restarts in the state before the crash.
- Halting – crash – server never restarts.

Flat Groups vs Hierarchical Gr.



Communic. in a flat group

Comm. in a simple hierarchical grp.

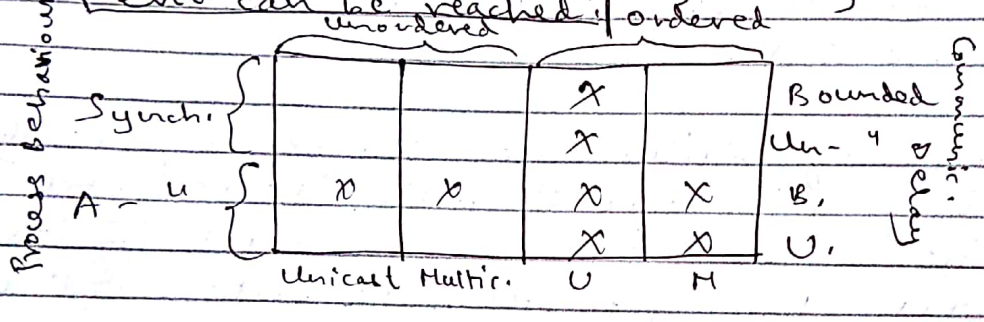
Agreem. prob.

Agreement issue in faulty systems:

Possible assumptions about the underlying sys.:

1. Synchr. v. asynch. sys.
2. Communic. delay is bounded or not
3. Msg. delivery is ordered or not
4. " transmission is done thro' unicasting or multicasting.

Circumst. under which distributed agreement can be reached:



Byzantine Agreement prob.:
Lamport et al. 1982

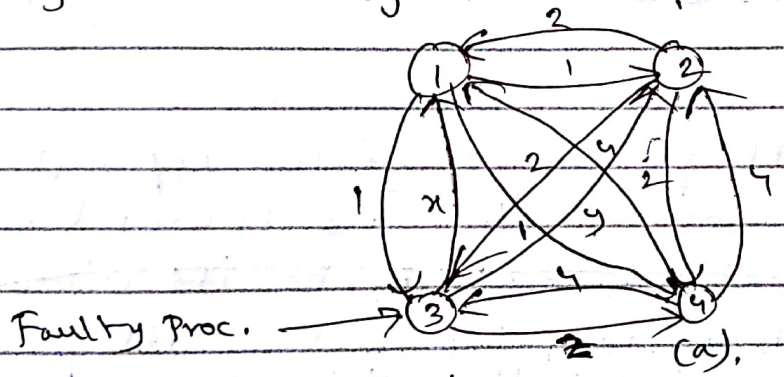
> Assume reliable synch. ordered uni-cast based msg. sys. There are N process, K of which may act as faulty or even malicious. A faulty process may send diff val. to diff processes.

Byzantine failure: In fault-tolerant distributed computing, a B.F. is an arbitary fault that occurs during exec of an algo in dist. sys. When a Byz. failure has occured sys. may respond in an unpredictable way.

> These arbitary failures may be loosely categorized as :

- a failure to take another step in the algo, also known as a crash failure;
- a failure to correctly exec. a step of the algo; &
- arbitary exec. of a step other than the one indicated by the algo.

Eg. Byzantine Agreement prob. for 4 processes



> 3 non-faulty & 1 faulty process. Each proc. sends their val. to the others. Proc. 3 lies, giving diff val. x, y, z to diff proc.

24/8/18

SM

Byzantine Agreement Prob. - 2

		1 Got	2 Got	4 Got
1	Got (1, 2, x, 4)			
2	" (1, 2, y, 4)	(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
3	" (1, 2, 3, 4)	(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
4	" (1, 2, z, 4)	(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

(b)

(c)

(b) The vectors that each process assembles based on (a).

(c) The vec. " " " " receives in step 3. Proc. 3 sends diff vectors to diff processes.

V. Imp.

Byzantine Agreement prob - 3

> 3 processes can agree on the values received from 1, 2, 4. so that malicious proc. 3 val. is irrelevant...

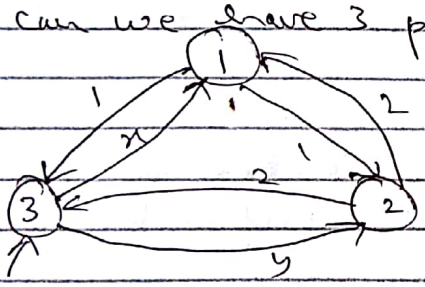
> If N = 3 & K = 1, i.e., only 2 non-faulty processes this will not work!

> Lamport provided that, with 2K+1 non faulty proc, the sys. will survive K faulty proc., which makes a total of 3K+1 proc. [Lamp. Alg.] (Byzan. Ag. P.)

BAP, with 2 correct or faulty processes..

Q. Can we have 3 proc. having 1 faulty p.?

Imp.



(a)

1 Got (1, 2, x)
2 " (1, 2, y)
3 " (1, 2, 3)

(b)

1 Got	2 Got
(1, 2, y)	(1, 2, x)
(a, b, c)	(d, e, f)

(c)

> Fig. 8-6. The same as Fig. 8-5, except now with 2 correct processes & 1 faulty proc.

→ 2-tier arch. → if client server comm. comes in exam

Client server communication:

> TCP - Transport control protocol at a connect. oriented end-to-end commu- unic. protocol is a reliable protocol. But it does not prevent connection crash, failure, which require searching for new connections.

RPC semantics in the presence of failures:

> 5 diff classes of failures that can occur in RPC systems.

1. The client is unable to locate the server.
2. The request msg from the client to the server is lost.
3. The server crashes after receiving a request. This can be handled with principles such as:
   - At least once
   - " most "
   - The preferred principle is exactly once, which is virtually impossible to implement.
4. The reply msg from the server to the client is lost.
5. The " crashes after sending a request.

> Each case needs to be reserved prop- -erly to mask the failures.

Failure Eg's.

> UDP service
- has omission failures coz it occasionally looses msgs.
- does not have val. failures coz it doesn't transmit corrupt msgs.

UDP uses checksums to mask the val. failures of the underlying IP by converting them to omission failures.

Reliable Multicasting

> Use -ve acknowledgement, known as scalable reliable multicasting - SRM

> Non-hierarchical & hierar. solns. are possible.

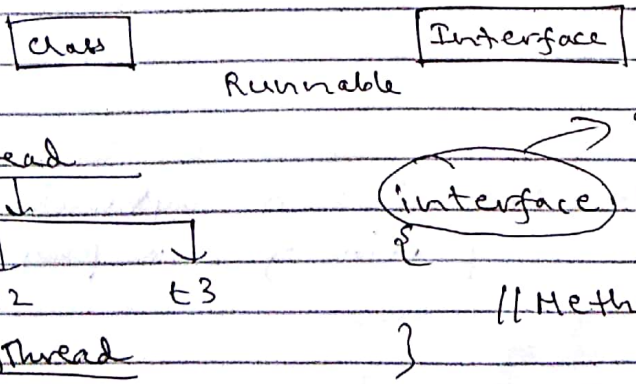
> Atomic multicasting requires all the replicas reaching agreement on the success or failure of multicast. This is known as dist. commit: two-phase or three-phase commit protocols can be used.

25/8/18

SM

→ V. Imp. → Blocking, non-blocking

Threads: → intra process communic.



(2)

There's a class called Abstract class in Java.
Abs. class is —

```
class A
{
    void getData()
    {
        ;
    }
    abstract void calculate();
}
```

ABC.java
↓ ⟶ Abc.java
Abc.class
A.class
BC.class

```
class A
{
    ;
    abstract void calculate();
}
interface BC
{
    abstract void c1();
               "    "  c2();
}
class Abc extends A implements BC
{
    psvm(...)
    {

    }
}
```

Java beans ⇒ .class files or byte code after compilation of Java prog.

(3)

better to write → implements Runnable

```
class MyThread extends Thread
{
    p s v m(...)
    {
        MyThread t1 = new MyThread();   // we can provide name of thread in quotes.
                 "  t2 =  "     "  ();
        t1.start();
        t2.  "  ();
        SOPln("InThread | started");
              " ("In  " 2  " );
    }
    p. v. run()
    {
        ;
    }
}
```

MIN_PRIORITY – 1
NORM_ " – 5
MAX_ " – 10
} → Priorities

31/8/18

Thread :

Class → Thread Interface → Runnable

```
int pl;
pl = t1.getPriority();
pl = pl + 2;
t2.setPriority(pl);

class myThread extends Thread{
    public void run(){
        if(Thread.currentThread().isDaemon()){//che-
        //-cking for daemon thread
            soPln("daemon thread work");
        }
        else{
        soPln("user thread work");
    }
    }
    public static void main(...){
        MyThread t1 = new MyThr.(); //creating thread
              "   t2 =  "    "  ();
              "   t3 =  "    "  ();

        t1.setDaemon(true); // now t1 is daemon
                                thread
        t1.start(); //starting thread
        t2.  "  (); t3.start();
    }
}
```

```
class NewThread implements Runnable
{
    String name;
    Thread t;
    NewThread(String threadname)
    {
        name = threadname;
        t = new Thread(this, name);
        System.out.println("New Thread: " + t);
        t.start();
    }
    public void run()
    {
        try
        {
            for(int i = 5; i > 0; i--)
            {
                System.out.println(name + ": " + i);
                Thread.sleep(1000);
            }
        }
        catch(InterruptedException e)
        {
            System.out.println(name + " Interrupted");
        }
        System.out.println(name + " exiting");
    }
}

class MultiThreadDemo
{
    public static void main(String args[])
    {
        new NewThread("One");
        new NewThread("Two");
        new NewThread("Three");
```

```
        try
        {
            Thread.sleep(1000);
        }
        catch(InterruptedException e)
        {
            System.out.println("Main Thread Interrupted");
        }
        System.out.println("    "    "    "    Exiting");
    }
}

public class threads
{
    public static void main(String args[])
    {
        Thread t = new Thread();
        System.out.println("Nos are printing line after 2 seconds...");
        try
        {
            for(int i=1; i<=10; i++)
            {
                System.out.println(i);
                t.sleep(2000);
            }
        }
        catch(InterruptedException e)
        {
            System.out.println("Thread Interrupted...");
            e.printStackTrace();
        }
    }
}
```

Error-control coding

- Replic. is expensive
- For certain applic. – RAM, Buses, error correcting codes can be used.
  - Hamming or other codes.

- Checkpts. & rollbacks
- " are copies of applications' state saved in some storage that's immune to the failures under consideration.
- A rollback restarts the exec. from a previously saved checkpt.
- When a failure occurs, applic. state is rolled back to the prev. checkpt & restarted from there.
- Can be used to recover from transient as well as permanent H/W failures.
- Can be used for uniprocessor & distributed applic.

Recovery Blocks (for s/w failures)

- Uses multiple alternates to perform the same func.
- One module is primary while others are secondary.
- When primary completes exec, its outcome is checked by an acceptance test.
- If O/P is not accep., a secondary module executes ~~until~~ either so on until an acceptable O/P is obtained or alternates are exhausted.
- This meth. can tolerate s/w failures coz alternates are usually implemented with diff approaches (S/W algos).

1/10/18

SM

Real Time Dist. Sys.:

> In general the correctness of a prog depends on the logical sequence of instructions, not when they are executed.

> A rel-time programs & sys. interact with the external world in a way that involves time.

> When a stimulus appears, the sys must respond to it in a certain way & by a certain deadline.

> If it delivers the correct and but after deadline, the sys is regarded as having failed.

> Here ~~totoon~~ WHEN the ans is produced is as imp as WHICH ans is prod.

How a real time sys works?

> Real Time applic. & sys are highly structured much more so than general purpose dist sys.

> Typically, an external device generates a stimulus for the comp, which must perform some action by a deadline.

> When req. work has been completed the sys becomes idle, until next stimulus arrives.

Types of Stimulii:

1). periodic - with a stimulus occurring regularly every ΔT seconds such as a comp in a TV set or VCR getting a frame every 1/60 of a sec.

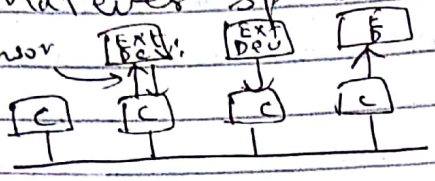
2). Aperiodic - they are recurrent, but not regular as in the arrival of an aircraft in a air traffic controller's air space.

3). Sporadic (unexpec.) - such as a device overheating.

New Ideas

> There may happen that 2 events placed in the same time instant & to serve one other will be negler.

- > Some designers are experimenting with the idea of putting a dedicated μprocessor in front of each real time device to accept o/p from it whenever it has something to say, & give it I/p at whatever s/ped it requires.



Dist. Real time sys. :

- > A collec of computers connec. by a N/w.
- > Some of these are connec. to ext. devices that produce or accept data or expect to be controlled in real time.
- > The comp. may be tiny μ-controllers built into the devices or stand alone m/cs.
- > In both cases they usually have sensors for receiving signals from the devices &/or actuators for sending signals to them.
- > The sensors & actuat. may be dig. or analog.

Soft real-time sys. - means that missing an occasional deadline is alright, for eg, a telephone switch might be permitted to lose or misroute one call in T time per under Overload conditions & still be within specification.

Hard real-time s/s: is unacceptable as this might lead to loss of life or an environmental catastrophe.

There are also interned sys. where missing a deadline means you have to kill of the current activity but the consequences is not fatal.

4/10/18

SM

Some Myths:

1. Real Time sys are about writing device drivers in assembly code.
2. Real T. computing is fast computing.
3.

Design Issues:

1. Clock synch.
2. Event-triggered vs Time-trigg. sys.
3. Predictability
4. Fault Tolerance

Event-t s/s:

> When a significant event in the outside world happens, it's detected by some sensor, which then causes the attached CPU to get an intern.

10

> It's an Interr. driven Sys.
> For soft real time systems with lots of computing power to spare this appr. is simple, works well & is still widely.
> for complex sys, it " "
if comp can analyze the prog & know all there's to know about the sys behaviour once an event happens, even if it can't tell when the event will happen.

Disadv : They can fail under conditions of heavy load that is when many events are happening at once.

(a)

Time - trigg sys :

> " " " assumes that all the clocks are synch. with a precision on the order of tens of μs. This prec. is possible as the protocol itself provides continuous clock synch. & has been designed to allow it to be in H/W to extremely high prec.

> A clk interr. occurs every ΔT milli-sec.
> At each clk tick

Time t. s.:

- ΔT must be chosen with extreme care.
- If it's too small, the sys will get many clk interr & waste too much time fielding them.
- If it's too large, serious may not be noticed until it's too late.
- Also the decision about which sensors to check on every clock tick & which to check on every other tick & so on is critical.
- Also, some events may be shorter than a clk tick so they must be saved to avoid losing them. They can be preserved electrically by latch ckts or by embedded μ-processors in the external devices.

Comparision:

- Event t designs give faster response at low load but more overhead & chance of failure at high load.
- Time trig. designs have the opposite properties & are furthermore only suitable in a relatively static envt in which a great deal is known about sys behaviour in advance.

Predictability:

- Real time systems behaviour is predictable.
- It should be clear at design time that the sys can meet all of its deadlines even at peak load.

Fault Tol.: A sys that can halt operation without danger is said to be fail-safe.

Lang – In which lang real time sys will be interpreted → Study from bk.

5/10/18 SM

Token Ring with time limit

TDMA (Time Division Multiple Access):

A potential problem with wide area real time distributed sys-tems is ~~there~~ their relatively high pckt loss rates. Std. protocols deal with pckt loss by setting a timer when each" is transmitted. If the " goes off by the ack. is received, the pckt is sent again. One sol. to avoid it" always to transmit each pckt twice.

↳ related to time-trigg|prot.

Real time scheduling:

- Dynamic
- Static

Dyn. S.: A periodic real time dist. sys has n tasks & n processors to run them on. Let

C_i be the CPU time needed by task i & let P_i is its period, i.e., the time b/w consecutive interr. To be feasible the utilisation of the sys. U must be related to N as

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq N$$

3 types of scheduling for Dist. dynamic scheduling real time s.i

1. ~~Red~~ Rate Monotonic Algo.
2. Earliest Deadline First
3. Laxity — How much CPU BT (Burst T_i) pending for a process.

Static Sched.: